Stony Brook University

Development of a Wireless Health Monitoring System with an Efficient Power Management Scheme based on Localized Time-Varying Data Analyses

A Dissertation Presented

by

He Zhao

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Biomedical Engineering

Stony Brook University

May 2008

Stony Brook University

The Graduate School

He Zhao

We, the dissertation committee for the above candidate for the **Doctor of Philosophy** degree, hereby recommend acceptance of this dissertation.

Dr. Ki H. Chon, Department of Biomedical Engineering
Dissertation Advisor

Dr. Emilia Entcheva, Department of Biomedical Engineering
Chairperson of Defense

Dr. Yingtian Pan, Department of Biomedical Engineering

Dr. Samir R. Das, Computer Science Department

This dissertation is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Development of a Wireless Health Monitoring System with an Efficient Power Management Scheme based on Localized Time-Varying Data Analyses

by

He Zhao

Doctor of Philosophy

in

Biomedical Engineering

Stony Brook University

2008

This work involves development of an integrated health monitoring system based on wireless sensor motes for continuous wireless monitoring of vital signs for large populations. The system incorporates efficient power management schemes based on localized data analysis, and will be useful in a variety of settings such as nursing homes, emergency rooms, military fields, and individual health monitoring. Remote monitoring of patients via wireless sensor systems has the potential to change the way health care is delivered. However, until now very few cost-effective wireless remote monitoring systems have been developed and put into use. One of the key challenges in making wireless monitoring ubiquitous in health care is efficiently managing the limited energy in individual sensor nodes. To this end, the current work utilizes smart sampling and sufficient processing at the sensor so that minimal data is transmitted. Our technique is based on the development of appropriate hardware and software implementations. It has the added benefit of significantly reducing the amount of data that needs to be analyzed at the central server, which further facilitates faster real-time alerts. Finally, our localized data analysis scheme also results in minimal usage of available radio transmission bandwidth, which allows for robust wireless transmission of many other vital signs' data from multiple patients in close vicinity. In this context of wireless health monitoring, this work has the following four specific aims. The first specific aim involves design and development of integrated scalable wireless motes containing an electrocardiograph, pulse oximeter sensor and other various sensors. The goal is to determine if these additional physiological parameters lead to better on-demand sampling rate strategies, better fault tolerance, and more accurate medical decision alerts. The second specific aim is to design and develop an on-demand variable sampling data transmission scheme. The sampling rate strategy will be based on localized real-time data analysis using a microprocessor attached to a mote combined with the subject's a priori medical data and the published risk factors. The third specific aim is to design wireless networking protocols for transmitting data from a large number of monitored subjects in the same physical space. The fourth specific aim involves development of new algorithms to facilitate more accurate medical decision alerts.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

ABP	*Arterial Blood Pressure*
AD	*Analog-Digital*
ADC	*Analog-Digital Convertor*
AF	*Atrial Fibrillation*
AIC	*Akaike Information Criterion*
AM	*Active Message*
ANS	*Autonomic Nervous System*
API	*Application Programming Interface*
AR	*Autoregressive*
ARMA	*Autoregressive Moving Average*
BF	*Blood Flow*
BJ	*Born-Jordan Distribution*
BP	*Blood Pressure*
BPS	*Bit Per Second*
CCA	*Clear Channel Assessment*
C/D TP	*Command/Data Transport Protocol*
CF	*Coherence Function*
CHIP	*Compatible Host Interface Protocol*
CPU	*Central Processing Unit*
CSW	*Channel Selection Word*
DFT	*Discrete Fourier Transform*
DLL	*Dynamic Linked Library*
DMA	*Direct Memory Access*
DPSS	*Discrete Prolate Spheroidal Sequence*
DSP	*Digital Signal Processing*
ECG	*Electrocardiograph*
FFT	*Fast Fourier Transform*
FIFO	*First In First Out*
FPS	*Frame Per Second*
FSM	*Finite State Machine*
GND	*Ground*
GPRS	*General Packet Radio Service*
GTS	*Guaranteed Time Slot*
GWN	*Gaussian White Noise*
HF	*High Frequency*
HR	*Heart Rate*
HRV	*Heart Rate Variability*
ICU	*Intensive Care Unit*
IDE	*Integrated Development Environment*
IRSDT	*Iteratively Refined Surrogate Data Technique*
IV	*Instantaneous Variance*
LAN	*Local Area Network*
KB	*Kilobyte*
LF	*Low Frequency*
LMS	*Least Mean Square*
LR-WPAN	*Low-Rate Wireless Personal Area Network*
MA	*Moving Average*
MAC	*Medium Access Control*
MDL	*Minimum Description Length*
MSE	*Mean Square Error*
ODT	*On-command Data Transmission*
OEM	*Original Equipment Manufacturer*
OOP	Object-Oriented Programming
OPS	*Optimal Parameter Search*
OS	*Operation System*
PAN	*Personal Area Network*
PC	*Personal Computer*
PDA	*Personal Digital Assistant*
PDM	*Principle Dynamic Mode*
PED	*Prediction Error Difference*
PHY	*Physical Layer*
PPG	*Photoplethysmography*
PPP	*Peer-to-Peer Protocol*
PSD	*Power Spectrum Density*

RAM	*Random Access Memory*
RAP	*Renal Arterial Pressure*
RLS	*Recursive Least Square*
RMSSD	*Root Mean Square of the Successive Difference of R-R intervals*
ROM	*Read-Only Memory*
RTOS	*Real-Time Operation System*
SD	*Surrogate Data*
SDNN	*Standard Deviation of Normal-to-Normal R-R intervals*
SHIP	*Standard Host Interface Protocol*
SNR	*Signal Noise Ratio*
SNT	*Star Network Topology*
SoC	*System on Chip*
SPI	*Serial Peripheral Interface*
SpO_2	*Saturation of Peripheral Oxygen*
SP-DSP	*Software Platform of DSP*
SPWV	*Smoothed Pseudo Wigner Ville*
STFT	*Short-Time Fourier Transform*
TDMA	*Time-Division Multiple Access*
TF	*Time-Frequency*
TFD	*Time-Frequency Distribution*
TGF	*Tubuloglomerular Feedback*
TIV	*Time Invariant*
TV	*Time-Varying*
TVCCF	*Time-Varying Causal Coherence Function*
TVCF	*Time-Varying Coherence Functions*
TVOPS	*Time-Varying Optimal Parameter Search*
TVSD	*Time-Varying Surrogate Data*
TVTF	*Time-Varying Transfer Functions*
UART	*Universal Asynchronous Receive / Transmit*
USART	*Universal Synchronous / Asynchronous Receive / Transmit*
USB	*Universal Serial Bus*
VAR	*Vector Autoregressive*
WiFi	*Wireless Fidelity*
WMSPM	*Wireless Multiple Subject-Parameter Monitoring*
WPAN	*Wireless Personal Area Network*

INTRODUCTION AND BACKGROUND

Growth in the elderly population of the United States continues to place increasing demands on heath care services. Telemedicine is growing in popularity, both among patients and at health insurance companies. Patients appreciate remaining independent within their communities. Medical insurers are supportive, as it reduces the incidence of costly admittance to hospitals. In-home health monitoring is rapidly emerging as a cost-effective mode of health care delivery for an elderly baby boomer population [1, 2]. Typically, in-home monitoring systems collect a variety of vital signs such as heart rate, blood pressure, oxygen saturation, body weight and temperature. These raw data are usually sent via phone lines to a central monitoring unit where a health professional reviews the information and responds appropriately. Currently, there is no cost-effective wireless in-home monitoring system, and thus, most monitoring companies rely on phone lines to transmit data, thus, the goal of constant monitoring is not completely accomplished. Furthermore, monitoring is mostly based on parameters other than the electrocardiograph (ECG) signal, since it requires a much greater sampling rate than parameters such as the respiration, blood pressure (BP), and pulse oximeter signal, to name a few.

There has been an increasing demand in the field of care industry that vital signals need to be continuously recorded for those patients with chronic conditions such as cardiovascular disease. The recorded signals can be further analyzed to provide more detailed information about the patient. It has valuable benefit for the physician to track the status of patients and to response quickly should any changes occur in the patient's condition. For example, according to America Heart Association, coronary heart disease is America's No. 2 killer, and stroke is the No. 3 and a leading cause of serious disability [3]. However most heart attacks start slowly, with mild pain or discomfort that is mostly ignored by those people who are affected. The only way to save these people is to respond to these mild symptoms before it is too late. This situation brings up the demand of monitoring the vital signals continuously, conveniently, and most importantly, be cost effective so that it can be widely deployed.

Varieties of remote monitoring systems have been developed to meet this demand. A typical example of remote monitoring system is a home care device, such as an electronic blood pressure or glucose meter. These devices use either radio to transmit the measured data to a nearby monitor for further analysis and transmission, or store the measured data into a local storage media (such as a flash card) so that it can be downloaded by the physicians after measurement. The ambulatory ECG Holter, used since 1960s, is such kind of a device, providing a reliable measurement of patient's ECG signal.

Some commercially developed health monitoring systems are discussed below.

LifeSync Wireless ECG System. This system consists of three components: 1) Monitor Transceiver; 2) Patient Transceiver; and 3) LeadWear Disposable Cable replacement system [4]. It uses Bluetooth wireless protocol to collect and transmit patient ECG and respiration data to the hospital existing ECG monitors. LifeSync System eliminates the lead wires and trunk cables between patients and beside, as well as eliminates the need to detach and reattach lead wires when transporting patients, and also facilitates patient mobility and ambulation. The system uses a rechargeable 3.6 volt lithium-ion battery. The battery lasts about 24 hours running on patient transceiver. The effective radio range of this system is around 10 meters [5].

ViTelCare Home Health Monitoring. The ViTelCare™ monitoring system is a comprehensive home-based disease management tool for a broad range of diagnostic groups including: Heart Failure, Chronic Obstructive Pulmonary Disease, Diabetes Mellitus, Hypertension, Major Depressive Disorder and Wound Care Assessment. The patient uses an integrated medical device to take health measurements, such as: weight, BP and Saturation of Peripheral Oxygen (SpO_2). They then answer some health assessment questions that are specific to their diagnosis. After completion, the monitoring session results are sent to the central repository to update the database. Once the clinician logs onto the database, the updated monitoring results are brought in front of him so that he is able to follow-up with a revised treatment plan with the patient [6].

Card Guard PMP4 System. Card Guard's new PMP4 Wireless Healthcare System provides the tools required for screening, monitoring, and managing of General Consumer Health, Disease Management and Fitness. It includes portable and wireless based medical monitors such as: 1) SelfCheck ECG 1/12-lead ECG monitor; 2) Spiro Pro Spirometer; 3) Oxy Pro Pulse Oximeter; SelfCheck BP; 4) BP Pro Blood Pressure; 5) BP Pro Plus Blood Pressure; 6) SelfCheck Gluco Blood Glucose Meter; and 7) A feature rich PMP4 Web-Based Medical Center. The PMP4 Wireless Healthcare System uses the most up-to-date communication technologies (Bluetooth, Wireless Fidelity (WiFi), Internet, GPRS) to enable the PMP4 Medical monitors to measure and transmit medical data to handheld devices, which can then upload to the dedicated PMP4 Web-Based Medical Center. The PMP4 Web-Based Medical Center provides secure, private data protection, a user friendly interface and access by physicians and patients using an Internet browser. It enables patients, physicians and health care providers an enhanced medical management and care with the benefit of paperless transactions [7].

Given the fact that aforementioned systems have been developed, they have their own limitations. For example, the LifeSync System lacks data analysis feature. It simply provides recorded raw data instead of analyzed results that are more important for both diagnosis purpose and preservation of limited system resources (memory, bandwidth and battery power etc.). The ViTel Care and Card Guard PMP4 systems have some data process functions, but they lack mobility and the processed results cannot be reviewed in a real-time fashion.

At present, a fully integrated portable low-cost wireless system that incorporates many essential physiological variables (e.g., ECG, respiration, BP and pulse oximeter) with efficient power management scheme does not exist. For example, most emergency medical centers as well as nursing homes still rely on individual devices that are all attached to patients via direct wires to collect data. A handful of emergency medical centers do employ wireless ECG monitors, but these are rife with technical problems such as having too many artifacts in the collected data (the signal-to-noise ratio is low), a limited range of transmission of data, and medical decision alerts that are prone to false alarms. Even at these hospitals, fully integrated wireless systems do not exist, due to both the high cost of deployment and immature technology.

Recent advances in wireless technologies have made some inroads in the aforementioned problems associated with a fully integrated wireless system becoming a reality. For example, Becker et al. demonstrated wireless transmission of a few vital physiological parameters using a Bluetooth protocol for a Personal Digital Assistant (PDA) device [8]. In other works, a cell phone has been used to transmit biomedical signals [1, 9, 10]. More recently, Rasid and Woodward have developed a Bluetooth telemedicine processor for a multichannel biomedical signal transmission via mobile cellular networks [10]. Specifically, their system utilizes the newly-developed cellular protocol known as General Packet Radio Service (GPRS), which has much higher data transmission rates than the global system for mobile communications. In another study, Hung et al. utilized a wireless application protocol for telemetry application of biomedical signals [9]. However, because this system uses an analog wireless transmission

module, the data are more sensitive to noise contamination. The common feature among all of the aforementioned systems is that they are all limited to data collection from a single or at most a few subjects.

To this end, a portable, battery-powered, low-cost wireless system that is capable of transmitting simultaneous multiple parameters from many subjects with smart sampling and transmission scheme is brought forward in this work. The developed wearable device, with wireless biosensors connected to self organizing wireless sensor network, will allow physicians to continuously monitor vital signs while the wearer performs normal activities, helping physicians to capture not only snapshots of patient's vital signals but also long-term trends and patterns that provide invaluable information about patient's ongoing condition. In contrast to the above systems, the system developed in this work features in: 1) low-cost; 2) low-power consumption; 3) real-time data processing and analysis; 4) on-command variable sampling rate; and 5) smart decision-making.

The device developed in this work is powered by 2 AA batteries to provide good mobility. Thus it is critical important to use the limited battery power smartly and efficiently so that the device can be used for long-term recording. In this work, we developed an On-command Data Transmission (ODT) scheme to facilitate an efficient power management by performing localized real-time Digital Signal Processing (DSP) at the device end. Since majority of the power is consumed on the wireless transmission, it will save significant power only if critical physiology parameters are transmitted instead of the raw data. For example, our experimental tests have shown that transmitting one lead raw ECG signal at 200Hz sampling rate, the wireless transmission module consumes approximately 25 mA current. If we incorporate R-wave detection algorithm on-board and calculate the power spectrum of heart rate every 5 minutes, the average power consumption of the transmission module is approximately only 4 mA. Another great advantage of on-board data processing is that it also saves a lot of bandwidth, which greatly increases the capacity of the system. In the aforementioned test, the difference of the transmission load is significant:

*raw data transmission: 2 byte/point × 200 points/sec = **400** bytes/sec*

*after data processing: 2 byte/parameter × 1 parameter/5 min < **0.007** bytes/sec*

The calculation shows that bandwidth usage is dramatically curtailed by transmitting only critical parameters.

Although, processed data provides critical markers indicating the patient's condition, it is not rare that raw data is needed by the care giver to further assess the patient's status. The system developed in this work has a unique feature to facilitate the easy access to patient's unprocessed raw data. An on-command smart sampling rate scheme has been developed for this system. At the monitoring center, if any abnormities are detected in the processed parameter either by the health carer or by the software, an alert is generated and a command is sent to the remote device to resume the continuous transmission of the raw data.

Since processed parameters are used as vital conditions markers, it becomes fundamentally important to determine how accurate and reliable the processed parameters can be to avoid false alarms. This brings the demand of more sophisticated data processing and analysis algorithms. The first part of this work is to develop advanced data analysis algorithms to extract useful information from raw physiology data. At the device end, these algorithms generate vital condition makers (a few processed parameters) that are transmitted when the subject is in normal conditions. At the monitor software end, these algorithms are applied to raw signal to provide the care giver more diagnosis information and generate appropriate alerts. The processed results are also associated with a decision-making module at the monitor software end. The decision-

making module is an expert system that guards those critical vital parameters and generates certain alarms, if abnormality is detected, in terms of its knowledge database and historical record of the patient.

The developed system can be easily modified depending on the scenario it is going to be used. It is very easy to tailor the system to fit different applications with a few hardware and software changes.

PART 1 ALGORITHM DEVELOPMENT

Chapter 1 ARMA Model Identification using Multiple Basis Functions [1]

1.1 Introduction

In this chapter, a previous developed algorithm that expands the time-varying parameters onto a single set of basis functions has been extended to multiple sets of basis functions. This feature allows the capability to capture many different dynamics that may be inherent in the system. A single set of basis functions that has its own unique characteristics can best capture dynamics of the system that have similar features. Therefore, for systems that have multiple dynamics, the use of a single set of basis functions may not be adequate. Computer simulation examples do indeed show the benefit of using multiple sets of basis functions over the single set of basis functions for cases with many switching dynamics. Moreover, the proposed method remains accurate even under significant noise contamination. Application of the proposed approach to blood pressure data likewise indicate better tracking capability of the two sets of basis function than the recursive least squares or a single set of basis functions.

In the past years, a novel algorithm for estimating time-varying (TV) autoregressive (AR) moving average models (MA), from which time-varying impulse response and transfer functions can be determined [11, 12]. The algorithm, termed, the time-varying optimal parameter search (TVOPS) is based on an expansion of time-varying parameters onto a set of orthogonal basis functions [11, 12]. This manipulation leads to two significant benefits: 1) TV parameters become time-invariant, which makes the underlying estimation task analytically tractable, and 2) a considerable reduction in the number of parameters needed to track each TV coefficient can be obtained. The choice of appropriate basis functions is predicated on a priori knowledge of the dynamics of the system, but in most cases, this information is not available. Some recent works have provided approaches to finding the best basis selection using either entropy-based algorithms [13] or wavelet packets [14, 15]. All of the aforementioned basis functions are designed for a particular form of dynamics, therefore, a choice of one particular basis function may not be appropriate, but multitudes of different basis functions may be necessary to capture multiple time-varying dynamics. Especially in biological systems, there are multitudes of varying dynamics rather than one primary dynamic, therefore, a choice of one particular basis function may not suffice. Having a multitude of different basis functions may alleviate the importance of choosing a single most appropriate basis function, but the problem is then among the initially chosen sets of basis functions, how to determine automatically which form of the basis functions best captures the dynamics at that time segment. This is of paramount importance for systems that may undergo multiple switching dynamics that are different from one state to another.

The subject of this chapter is to present a method which resolves this problem. The new method employs the previously-developed TVOPS for identification of TV autoregressive or

1 This work was supported by the National Institute of Health, R01 HL69629. Relevant work has been published on IEEE Transaction on Biomedical Engineering, Vol. 52, No. 5, pp 956-960, May, 2005.

autoregressive moving average (ARMA) models [11], whereby instead of one set of basis functions, two or more sets of basis functions with different properties are used to track varying parameters. Multiple sets of basis functions can be used to fit various forms of model parameters. The task of determining appropriate basis functions for particular dynamics among the initially-chosen sets of basis functions is performed by the TVOPS algorithm. For the detail of TVOPS algorithm, the reader is referred to Appendix I.

The advantages of the proposed method are that first, even without a priori knowledge of the characteristics of the nonstationary system, the criticality of choosing suitable basis functions is minimized. Second, the expansion of TV parameters onto multiple sets of basis functions is more accurate than projecting them onto only a single set of basis functions.

1.2 Using Multiple Basis Functions for ARMA Model Identification

The input-output relationship of a TV-ARMA process is described by the following equation:

$$y(n) = \sum_{i=1}^{P} a(i,n)\, y(n-i) + \sum_{j=0}^{Q} b(j,n)\, x(n-j) + e(n) \tag{1-1}$$

where $a(i,n)$ and $b(j,n)$ are the time-varying AR and MA parameters to be determined, respectively, and are functions of time. Indices P and Q are the maximum model orders of AR and MA models, respectively, and are chosen by the user. We assume that the maximum model orders are time invariant. The term $e(n)$ is the prediction error. The proposed method is to expand the TV parameters $a(i,n)$ and $b(j,n)$ onto multiple sets of basis functions instead of expanding them onto only a single set of basis functions $\pi^{(l)}(n)$ for $l = 1,2,\ldots,L$, such that the following expressions hold:

$$\begin{aligned} a(i,n) &= \sum_{l=1}^{L} \sum_{k_l=0}^{V_l} \alpha^{(l)}(i,k_l) \cdot \pi_{k_l}^{(l)}(n) \\ b(j,n) &= \sum_{l=1}^{L} \sum_{k_l=0}^{V_l} \beta^{(l)}(j,k_l) \cdot \pi_{k_l}^{(l)}(n) \end{aligned} \tag{1-2}$$

where $\alpha^{(l)}(i,k)$ and $\beta^{(l)}(i,k)$ represent the expansion parameters, V_l is the maximum number of basis sequences, $\pi_{k_l}^{(l)}(n), k_l = 1,2,\ldots,V_l$ represents one set of basis functions when $l = 1$ and multiple sets of basis functions for $l > 1$. Substituting Eq. (1-2) into Eq. (1-1), we obtain the following:

$$y(n) = \sum_{l=1}^{L} \left\{ \begin{aligned} &\sum_{i=1}^{P} \sum_{k_l=0}^{V_l} \alpha^{(l)}(i,k_l)\, \pi_{k_l}^{(l)}(n)\, y(n-i) \\ &+ \sum_{j=0}^{Q} \sum_{k_l=0}^{V_l} \beta^{(l)}(j,k_l)\, \pi_{k_l}^{(l)}(n)\, x(n-j) \end{aligned} \right\} + e(n) \tag{1-3}$$

To eliminate possible linear dependence (lack of uniqueness) among different sets of

basis functions, we explicitly orthogonalize these different sets of basis functions using the Gram-Schmidt orthogonalization procedure. Thus, all of the vectors within each set of basis functions, as well as vectors of different sets of basis functions, are all orthogonalized. This mutual orthogonality ensures that all of the vectors are linearly independent, which ensures obtaining a unique solution. One drawback to the orthogonalization of different sets of basis functions is that the inherent properties of the basis functions change. For example, the resultant Walsh functions orthogonalized to the Legendre no longer retains their inherent square wave characteristics; they are a composite of square and sinusoidal-like waveforms.

Once proper basis functions have been chosen, we define new variables such that:

$$\begin{aligned} y_{k_l}^{(l)}(n-i) &= \pi_{k_l}^{(l)}(n)\, y(n-i) \\ x_{k_l}^{(l)}(n-j) &= \pi_{k_l}^{(l)}(n)\, x(n-j) \end{aligned} \tag{1-4}$$

Substituting Eq. (1-4) into Eq. (1-3) results in the following expression:

$$y(n) = \sum_{l=1}^{L} \left\{ \sum_{i=1}^{P} \sum_{k_l=0}^{V_l} \alpha^{(l)}(i,k_l)\, y_{k_l}^{(l)}(n-i) + \sum_{j=0}^{Q} \sum_{k_l=0}^{V_l} \beta^{(l)}(j,k_l)\, x_{k_l}^{(l)}(n-j) \right\} + e(n) \tag{1-5}$$

Eq. (1-5) shows that the TV-ARMA model can now be considered a time invariant (TIV) ARMA model, since $\alpha^{(l)}(i,k)$ and $\beta^{(l)}(i,k)$ are not functions of time.

The next step of the algorithm uses the optimal parameter search (OPS) method, which selects only the linearly-independent vectors from the pool of candidate vectors. This step should not be confused with the previous step of using the Gram-Schmidt orthogonalization procedure to ensure a unique solution among different sets of basis functions. Linear independent vectors are determined by selecting, for example $y(n-1)$ as the first candidate vector. If the next candidate vector is $x(n)$, then this vector and the first candidate vector $y(n-1)$ are then used to determine the linear independence (e.g., use $y(n-1)$ to fit $x(n)$ using the least squares method and calculate the error between $x(n)$ and the estimated vector). Because signals are contaminated by noise we preset threshold value so that if the error value is larger than the preset threshold, then the vector $x(n)$, for example can be selected as an independent candidate vector. The preset threshold is set to 0.0001 for all simulation examples and for application of the method to experimental data, as provided in 1.3 and 1.4. Assuming that $y(n-1)$ and $x(n)$ are the determined independent vectors and the next candidate vector is $y(n-2)$, then we use both $y(n-1)$ and $x(n)$ to fit $y(n-2)$ using the least squares. Calculate the error between $y(n-2)$ and already determined independent vectors to determine if the vector $y(n-2)$ should be included as a candidate vector as described above. This procedure is continued until all the linearly independent vectors are selected to form a new candidate vector.

The final step of the algorithm involves calculation of the coefficients using the least squares and retaining only those coefficients that are considered to be significant. One approach that can be adopted to determine the significance of model terms is provided in [16]. Further details of this final step as well as the procedure of the OPS can be found in [16].

To determine the proper number of basis functions, a modified version of the Akaike Information Criterion (AIC) can be used. The modified AIC for multiple sets of basis functions for the AR process:

$$AIC = N\log\sigma^2 + 2\left[p\sum_{i=1}^{l} k_i + 1 \right] \tag{1-6}$$

where N is the length of the data, σ^2 is the prediction error, p is the number of significant model terms (not to be confused with the maximum model order), k_i denotes the number of basis functions in the l sets of basis functions.

1.3 Applying the Method to Simulated Data

Monte Carlo simulations of 100 realizations were performed. Thus, all plots and tables represent mean values.

To illustrate the advantage of using two sets of basis functions over a single set of basis functions, we consider a TV-AR(2) model that has the following form:

$$y(n) = a(1,n)\,y(n-1) + a(2,n)\,y(n-2) + e(n) \tag{1-7}$$

where $e(n)$ is zero-mean Gaussian white noise with a variance of 0.25. TV parameters have the following expression:

$$\begin{aligned} &a(1,n) = 2\cos\left[2\pi f(n)\right], a(2,n) = -1 \\ &n = 1,\ldots,600 \end{aligned} \tag{1-8}$$

where

$$f(n) = \begin{cases} 0.3 & n = 1,\ldots,333 \\ 0.15 - 0.1\sin\left[2\pi\left(n - 2/N - 1\right)/N\right] & n = 334,\ldots,600 \end{cases} \tag{1-9}$$

We have purposely selected a sharp transition at $n = 333$ because this point does not match a transition point of any Walsh function, so as not to bias our simulation results using Walsh functions. We purposely select an incorrect AR model order of 6 despite the fact that the correct AR model order is 2. The OPS correctly determined that there are only two significant model terms $y(n-1)$ and $y(n-2)$ among the pool of 6 candidate terms $\left[y(n-1),\ldots,y(n-6)\right]$. With the correctly chosen model order of two, determination of the proper number of basis functions was calculated using Eq. (1-6) for the single set and multiple sets of basis functions. Fig. 1-1(a), (b) and (c) show the performance of the parameter estimation using 11 Legendre polynomials and 16 Walsh functions, 20 Legendre polynomials, and 20 Walsh functions, respectively. The method based on two sets of basis functions appears to outperform those with a single set of basis functions. When using only the Legendre polynomials (Fig. 1-1(b)), because they have a sinusoidal waveform, the estimated waveform has a predominately sinusoidal shape. The Walsh functions alone, shown in Fig. 1-1(c), are dominated by the waveform that is square-shaped. The result with two sets of basis functions is impressive because it is able to track three distinct waveforms: a constant value, an abrupt change, and the sinusoidal waveform. Table 1-1 statistically confirms that the two sets of basis functions yield smaller mean square error (MSE) values than either a single set of basis functions or the recursive least square (RLS).

Table 1-1 The MSE values of the simulation example (clean signal). The Kruskal-Wallis one way analysis of variance on ranks was performed (α=0.05). There are significant differences ($p<0.001$). Student-Newman-Keuls test was performed (α=0.05) where Legendre & Walsh $\neq$ Legendre $\neq$ Walsh $\neq$ RLS.

Approach	*RLS*	*Legendre & Walsh*	*Legendre*	*Walsh*
MSE	1.73±0.07	0.87±0.88	1.40±1.32	1.75±1.72

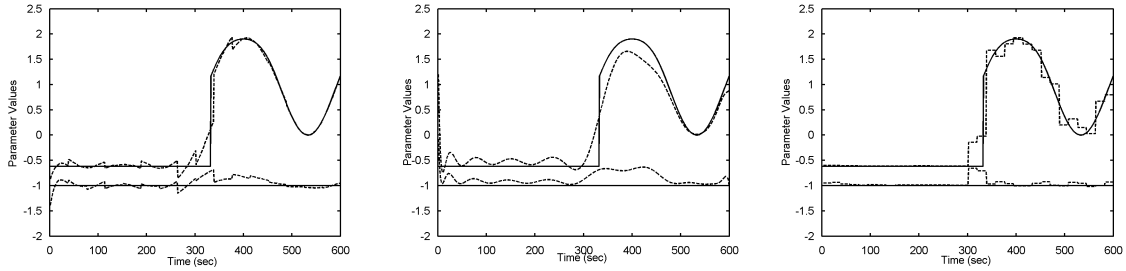


Fig.1-1. Comparison of simulation example with two sets of basis functions (11 Legendre polynomials and 16 Walsh functions) with those with one set of basis functions (20 Legendre polynomials and 20 Walsh functions) for TV parameters estimation: (a) actual (solid lines) and estimated (dotted lines) model parameters with two sets of basis functions, (b) estimated model parameters with 20 Legendre polynomials, (c) estimated model parameters with 20 Walsh functions.

To further challenge the method, we add Gaussian white noise to the system output of Eq. (1-7) so that the signal-to-noise ratio is 20 dB. We compare the proposed approach to the recursive least squares (RLS) in addition to single sets of basis functions. As in the previous example, we initially set the maximum AR model order to be 6. Despite the added noise, the algorithm correctly determines the model order to be two. Therefore, we assume the correct model order of 2 for all methods.

As in the previous example without noise contamination, Table 1-2 statistically confirms better performance with two sets of basis functions than either the RLS or a single set of basis functions. In this example, especially the use of Legendre basis functions alone has resulted in a significantly large MSE value.

Table 1-2 The MSE values of the simulation example (SNR = 20db). The Kruskal-Wallis one way analysis of variance on ranks was performed (α=0.05). There are significant differences ($p<0.001$). Student-Newman-Keuls test was performed (α=0.05) where Legendre & Walsh $\neq$ Legendre $\neq$ Walsh $\neq$ RLS

Approach	*RLS*	*Legendre & Walsh*	*Legendre*	*Walsh*
MSE	6.82±6.48	5.61±5.29	18.09±16.79	8.78±8.22

Fig. 1-2 shows comparison of time-frequency spectra among various methods for the simulation example. We compare multiple sets of basis functions (Fig. 1-2(b)) to solely Legendre functions (Fig. 1-2(c)), solely Walsh functions (Fig. 1-2(d)), RLS (Fig. 1-2(e)), and to three other time-frequency spectral methods: short-time Fourier transform (STFT) (Fig. 1-2(f)), smoothed pseudo Wigner Ville (SPWV) (Fig. 1-2g) and Born-Jordan (BJ) distribution (Fig. 1-2(h)). The method proposed is able to provide the correct time-frequency spectrum, and it does so with higher resolution in both time and frequency domains than any of the other methods compared.

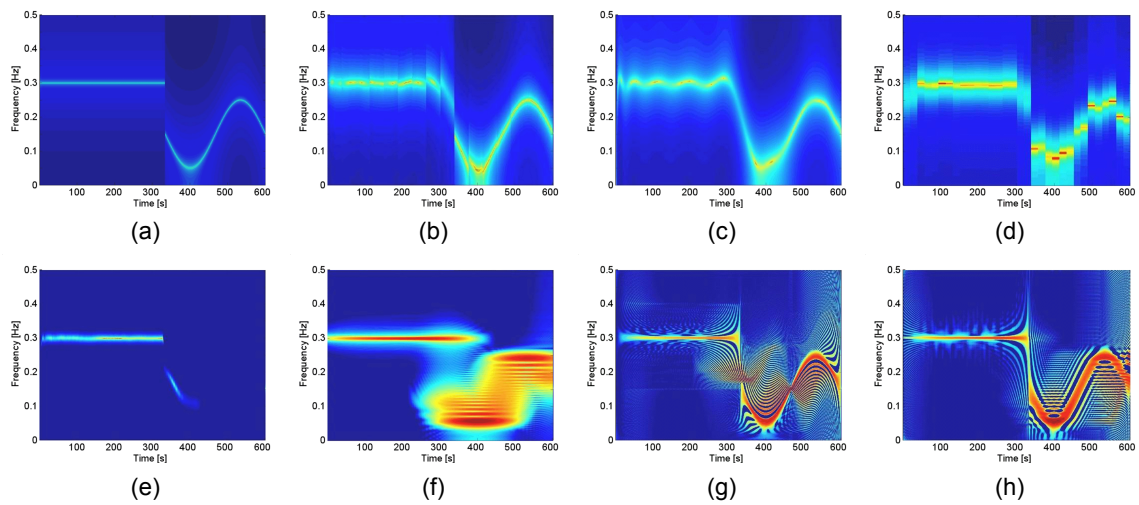


Fig. 1-2. Time-frequency spectral characteristics of the simulated signal. (a) True Spectrum, (b) Legendre and Walsh, (c) Legendre, (d) Walsh, (e) RLS, (f) STFT, (g) smoothed pseudo Wigner-Ville and (h) Born-Jordan TF spectrum.

1.4 Applying the Method to Renal Blood Pressure Data

A previous work has shown that a more direct and comprehensive approach to eliciting dynamic responses of the autoregulatory mechanisms is to induce a rapid step changes in renal arterial pressure (top panel of Fig. 1-3) [17]. This has been reported to invoke dynamic responses all involved autoregulatory mechanisms. Most studies have mainly relied on inducing either spontaneous or random renal arterial pressure (RAP) to elicit autoregulatory responses [18]. Thus, the RAP (normalized blood pressure) tracing shown in the top panel of Fig. 1-3 provides a good example of how the TVOPS employing multiple sets of basis functions can be applied to track a time-varying signal that consists of both slow and fast time-varying dynamics. The TVOPS determined 5 parameters from the initial AR model order of 10. Overlaid on top of the RAP (solid line) is the tracking of the RAP by the TVOPS (with 8 Legendre and 4 Walsh basis functions), shown in dashed line of the top panel of Fig. 1-3. The middle panel of Fig. 1-3 shows a segment (600 to 800 seconds) of the top panel tracings (indicated by two dashed vertical lines) while the bottom panel shows the residual error (MSE = 0.016) of the TVOPS algorithm. These plots indicate that the TVOPS employing both Legendre and Walsh functions is well suited to track both the slow and fast transients of the RAP signal. The MSE for the RLS is 0.014 and this is based on 10,000 parameters (5 parameters at each time point for 2000 data points), versus 0.016 for the TVOPS using 60 parameters (5 parameters times 12 (total number of basis functions used) = 60). If we reduced the number of parameters from 5 to 4 for the RLS (8000 parameters), then the MSE increases to 0.042. Thus, the TVOPS performs better when one factors in the significantly fewer parameters it uses compared to the RLS.

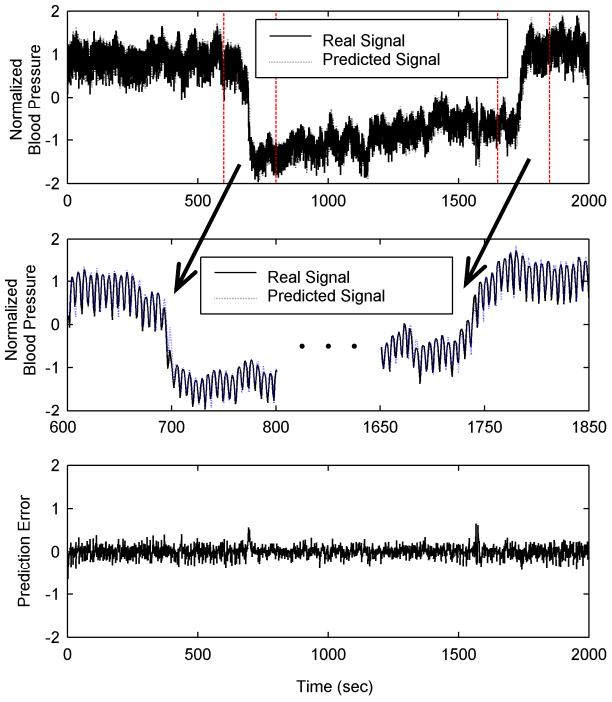


Fig. 1-3. One-step-ahead prediction of the normalized renal arterial pressure (RAP) using the TVOPS. Top panel: RAP (solid line) and tracked RAP by the TVOPS (dashed line); middle panel: a segment of the top panel tracing where solid and dashed lines represent the RAP and tracked RAP by the TVOPS, respectively; bottom panel: prediction error of the TVOPS.

1.5 Summary

A new method has been presented based on using multiple sets of basis functions to account for a multitude of varying dynamics, which can be especially prevalent in the dynamics of physiological systems. We have previously developed a robust algorithm based on a single set of basis functions that has been shown to be more accurate than the recursive least square method [19]. However, as shown in the present paper, a single set of basis functions cannot account for multiple dynamics. While we have primarily demonstrated simulation examples using two sets of basis functions, the proposed approach suggests that for a practical application where there are many unique dynamics present, the method can handle and should incorporate many different sets of basis functions. While the result is not shown, the TVOPS is a reliable method for determining the correct model order even when the data length is as short as 200 points [16]. One drawback with the method is that the computational load is far greater (~ 100%) than when using the RLS.

Chapter 2 Development of a New Method to Estimate Coherence Function [1]

2.1 Introduction

In this chapter, a new method to estimate reliable time-varying coherence functions (TVCF) for causal systems has been introduced. The technique is based on our previously developed method to estimate time-varying transfer functions (TVTF), known as the time-varying optimal parameter search algorithm [12]. The TVCF is estimated by the multiplication of two TVTFs. The two TVTFs are obtained using signal x as the input and signal y as the output to produce the first TVTF, and signal y as the input and signal x as the output to produce the second TVTF. Demonstration of the feasibility and efficacy of the proposed approach is provided with both simulation examples and application to renal blood flow and pressure data. The proposed approach provides higher time-frequency resolution TVCF than afforded by the short time Fourier transform based TVCF.

Applications of the time-frequency distribution (TFD) have had a tremendous impact in understanding dynamic processes of various physiological systems. Physiological systems are inherently time varying, thus, immediate acceptance of the TFD-based methods by the biomedical and biological communities was readily expected. Recently, new methods were introduced to compute time-varying transfer functions (TVTF) that have the potential to further advance the understanding of the dynamic processes underlying physiological systems [11, 12]. The characterization of physiological systems with the TVTF is important because the admittance gain between input and output signals is correctly characterized to be transient, and not stationary, as is assumed with the time-invariant transfer function.

In this chapter, we utilize a parametric (model based, e.g, autoregressive moving average model) TVTF to estimate the TV coherence function (TVCF). Currently, the estimation of the TVCF is mainly based on the use of the short-time Fourier transform (STFT) where the ratios between the magnitudes of the TV cross spectrum and the TV auto spectra of the two signals are calculated. Two disadvantages of the STFT are the assumption that time segments are stationary and the limited time-frequency resolution, which preclude the STFT being the method of choice in obtaining accurate and high resolution TVCF. Furthermore, the STFT suffers from high variance. Consequently, the likelihood of obtaining statistical significance of the coherence value is diminished. Recent separate works by Xu et al. [20], and Lovette and Ropella [21] have alleviated the high variance problem of the STFT by introducing multiple window time-frequency (TF) analysis. However, the inherent problem of limited resolution with the STFT was not improved by this approach, but in fact the variance reduction scheme had the undesired effect of further reducing the TF resolution.

As in the time-invariant case, high-resolution TF spectra, TVTF, and TVCF can be obtained via parametric approaches. Arnold et al. [22] have developed a parametric TV spectrum and TVCF by means of Kalman filtering. In their work, a parametric TV cross spectrum was obtained by reformulating an ARMA process into a vector AR model, from which the vector TV autospectrum between two signals were obtained. The accuracy of any time-invariant and TV parametric approaches greatly hinges on the proper choice of the model order. In the work by Arnold et al. [22], determination of a model order was not discussed.

---

1 Relevant work has been published on *Annals of Biomedical Engineering*, Vol. 33, No. 11, pp 1582-1594, 2005.

An accurate approach for estimating the TVCF is greatly needed, because even today many investigators still resort to time-invariant coherence function analysis [23-25]. The coherence function is important because it provides information as to how two signals are phase-coupled or "coherent" with each other. For most physiological systems, it is expected that the coherence between two signals is not the same throughout the time duration of a data set. Thus, the TVCF is expected to reveal more accurate insights into the time-varying nature of the coupling between two signals. In this work, we propose a different parametric method to estimate TVCF than the approach proposed by Arnold et al. [22]. Our approach is based on the algorithm termed time-varying optimal parameter search (TVOPS), which we have recently developed for estimating TVTF [11, 12]. The TVOPS has been shown to be effective in selecting only the significant time-varying model terms, and is more accurate than the two most well-established model order criteria: the Akaike Information criterion (AIC) and the minimum description length (MDL) [16].

The proposed approach is presented in 2.2. Illustrative simulation examples as well as the application of the proposed method to experimental data are presented in 2.3 to demonstrate the feasibility and efficacy of the proposed approach.

2.2 The Development of a Parametric Method to Estimate Coherence Function

In this section, we demonstrate that the TVCF can be obtained by using the TVTF relationships. To demonstrate the use of the TVTF in obtaining the TVCF, we first define the TVCF via the nonparametric time-frequency spectra:

$$\left|\gamma(t,f)\right|^4 = \frac{\left|S_{xy}(t,f)\right|^2}{S_{xx}(t,f)S_{yy}(t,f)} \frac{\left|S_{yx}(t,f)\right|^2}{S_{yy}(t,f)S_{xx}(t,f)} \tag{2-1}$$

where $S_{xy}(t,f)$ and $S_{yx}(t,f)$ represent the time-frequency cross spectrum, and $S_{xx}(t,f)$ and $S_{yy}(t,f)$ denote the autospectra of the two signals x and y, respectively. The above expression $\frac{\left|S_{xy}(t,f)\right|^2}{S_{xx}(t,f)S_{yy}(t,f)}$ is the coherence function when x is considered as the input and y as the output while $\frac{\left|S_{yx}(t,f)\right|^2}{S_{yy}(t,f)S_{xx}(t,f)}$ is the coherence function when y is considered as the input and x as the output. We note that for a linear time-varying system with x and y as the input and output signals, respectively, the following TVTF in terms of time-frequency spectra can be obtained:

$$H_{x\to y}(t,f) = \frac{S_{xy}(t,f)}{S_{xx}(t,f)} \tag{2-2}$$

where $H_{x\to y}(t,f)$ denotes the TVTF from the input x to the output y signals. Similarly, if we reversed the input and output relationship such that the variables y and x represent input and output signals, respectively, as Fig. 2-1 depicts, then the following TVTF can be obtained:

$$H_{y\to x}(t,f) = \frac{S_{yx}(t,f)}{S_{yy}(t,f)} \tag{2-3}$$

The desired relationship of Eq. (2-1) can be obtained by multiplying the two TVTF relationships of Eqs. (2-2) and (2-3), which yields:

$$\begin{aligned}
&\left|H_{x\to y}(t,f)H_{y\to x}(t,f)\right|^2 \\
&=\left|\left(\frac{S_{xy}(t,f)}{S_{xx}(t,f)}\right)\left(\frac{S_{yx}(t,f)}{S_{yy}(t,f)}\right)\right|\left|\left(\frac{S_{xy}(t,f)}{S_{xx}(t,f)}\right)\left(\frac{S_{yx}(t,f)}{S_{yy}(t,f)}\right)\right| \\
&=\left[\frac{\left|S_{xy}(t,f)\right|^2}{S_{xx}(t,f)S_{yy}(t,f)}\right]\left[\frac{\left|S_{yx}(t,f)\right|^2}{S_{yy}(t,f)S_{xx}(t,f)}\right] \\
&=\left|\gamma(t,f)\right|^4
\end{aligned} \tag{2-4}$$

Thus, time-varying magnitude squared coherence, $\left|\gamma(t,f)\right|^2$ is then obtained by multiplying the two transfer functions, $\left|H_{x\to y}(t,f)H_{y\to x}(t,f)\right|$, together.

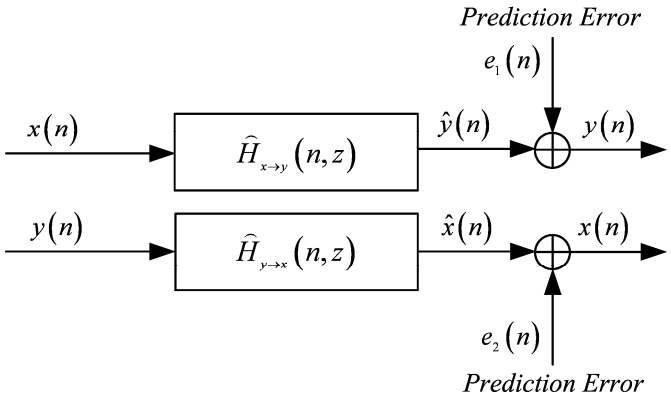


Fig. 2-1. Block diagram of two procedures involved in calculation of time-varying coherence function.

Given the relationship of Eq. (2-4), we can obtain a high resolution TVCF via the parametric TVTF. Specifically, each of the two transfer functions in Eq. (2-4) can be obtained using autoregressive moving average (ARMA) models:

$$\begin{aligned}
y(n) &= -\sum_{i=1}^{P_1} a(n,i)\,y(n-i) + \sum_{j=0}^{Q_1} b(n,j)\,x(n-j) \\
x(n) &= -\sum_{i=1}^{P_2} \alpha(n,i)\,x(n-i) + \sum_{j=0}^{Q_2} \beta(n,j)\,y(n-j)
\end{aligned} \tag{2-5}$$

where the top expression in Eq. (2-5) represents $y(n)$ and $x(n)$ as the output and input, respectively, and the reverse for the bottom expression of Eq. (2-5). Our definition of the ARMA model refers to two measured signals of $y(n)$ and $x(n)$. Given the ARMA models of Eq. (2-5), the two transfer functions of Eq. (2-4) can be obtained by the following:

$$H_{x\to y}\left(n,e^{jw}\right)=\frac{B\left(n,e^{jw}\right)}{A\left(n,e^{jw}\right)}=\frac{\sum_{l=0}^{Q_1}b\left(n,j\right)e^{-jwl}}{1+\sum_{i=1}^{P_1}a\left(n,i\right)e^{-jwi}}$$
$$H_{y\to x}\left(n,e^{jw}\right)=\frac{\beta\left(n,e^{jw}\right)}{\alpha\left(n,e^{jw}\right)}=\frac{\sum_{l=0}^{Q_2}\beta\left(n,j\right)e^{-jwl}}{1+\sum_{i=1}^{P_2}\alpha\left(n,i\right)e^{-jwi}} \tag{2-6}$$

By using the TVOPS approach, we are able to obtain an accurate and high resolution TVTF of Eq. (2-6) [11, 12].

In the two references [11, 12], it was demonstrated that more accurate estimate of the TVTF can be obtained than of the STFT, including an example concerning TV amplitude modulated signals. Furthermore, because the method proposed does not utilize segmentation of the data records, as is the case with the STFT, neither time nor frequency resolutions are compromised. Note that for the proposed approach, the TVTF, and consequently the TVCF, are obtained for every time point and are based on a few parameters chosen by the TVOPS from the initially-selected model orders P_1, Q_1, P_2 and Q_2 in Eq. (2-6). Therefore, while the STFT assumes that each segment of data is stationary, the proposed method is based on modeling TV characteristics at each time point. The temporal resolution of the TVTF, however, is limited by the number and choice of basis functions. For short data records, the use of the STFT is especially challenging, but the method proposed remains accurate even for short data records; it has been shown that the method remains accurate for data lengths as short as 500 points. It should be noted, however, that the approach we propose requires the input signal to be spectrally rich to estimate parameters of the parametric transfer functions.

An abbreviated version of the algorithm for estimating TVTF via the TVOPS is outlined in the Appendix I. For full details of the TVOPS algorithm, the reader is referred to [11, 12].

In the proceeding section, we show that theoretical values of TVCF are bounded in the interval of zero to one. To illustrate, consider two signals modeled by the following equations:

$$\begin{aligned} y\left(n\right)&=\hat{h}_{x\to y}\left(n\right)*x\left(n\right)+e_1\left(n\right)\\ x\left(n\right)&=\hat{h}_{y\to x}\left(n\right)*y\left(n\right)+e_2\left(n\right)\end{aligned} \tag{2-7}$$

Here, $e_1\left(n\right)$ and $e_2\left(n\right)$ are prediction error terms that are uncorrelated to $y\left(n\right)$ and $x\left(n\right)$, respectively, and $*$ denotes the convolution operator. Taking the Fourier transform on both sides of Eq. (2-7) yields:

$$\begin{aligned} S_{yy}&=\left|\hat{H}_{x\to y}\right|^2S_{xx}+S_{e_1}=\left|\hat{H}_{x\to y}\right|^2S_{xx}+\sigma_{e_1}^2\\ S_{xx}&=\left|\hat{H}_{y\to x}\right|^2S_{yy}+S_{e_2}=\left|\hat{H}_{y\to x}\right|^2S_{yy}+\sigma_{e_2}^2\end{aligned} \tag{2-8}$$

Note that:

$$\begin{aligned} \left|S_{xy}\right|&=\left|\hat{H}_{x\to y}S_{xx}\right|\\ \left|S_{yx}\right|&=\left|\hat{H}_{y\to x}S_{yy}\right|\end{aligned} \tag{2-9}$$

Substituting the expressions in Eq. (2-8) and Eq. (2-9) into Eq. (2-1):

$$
\begin{aligned}
\left|\gamma(t,f)\right|^4 &= \frac{\left|S_{xy}\right|^2}{S_{xx}S_{yy}}\frac{\left|S_{yx}\right|^2}{S_{yy}S_{xx}} \\
&= \frac{\left|\hat{H}_{x\to y}\right|^2 S_{xx}^2}{S_{xx}\left(\left|\hat{H}_{x\to y}\right|^2 S_{xx}+\sigma_{e_1}^2\right)}\frac{\left|\hat{H}_{y\to x}\right|^2 S_{yy}^2}{S_{yy}\left(\left|\hat{H}_{y\to x}\right|^2 S_{yy}+\sigma_{e_2}^2\right)} \\
&= \frac{\left|\hat{H}_{x\to y}\right|^2 S_{xx}^2\left|\hat{H}_{y\to x}\right|^2 S_{yy}^2}{\left(\left|\hat{H}_{x\to y}\right|^2 S_{xx}^2+S_{xx}\sigma_{e_1}^2\right)\left(\left|\hat{H}_{y\to x}\right|^2 S_{yy}^2+S_{yy}\sigma_{e_2}^2\right)} \\
&= \frac{\left|\hat{H}_{x\to y}\right|^2 S_{xx}^2\left|\hat{H}_{y\to x}\right|^2 S_{yy}^2}{S_{xx}^2S_{yy}^2\left|\hat{H}_{x\to y}\right|^2\left|\hat{H}_{y\to x}\right|^2+S_{xx}^2S_{yy}\left|\hat{H}_{x\to y}\right|^2\sigma_{e_2}^2+S_{yy}^2S_{xx}\left|\hat{H}_{y\to x}\right|^2\sigma_{e_1}^2+S_{xx}S_{yy}\sigma_{e_1}^2\sigma_{e_2}^2} \\
&= \frac{1}{1+\dfrac{\sigma_{e_2}^2}{S_{yy}\left|\hat{H}_{y\to x}\right|^2}+\dfrac{\sigma_{e_1}^2}{S_{xx}\left|\hat{H}_{x\to y}\right|^2}+\dfrac{\sigma_{e_1}^2\sigma_{e_2}^2}{S_{yy}\left|\hat{H}_{y\to x}\right|^2 S_{xx}\left|\hat{H}_{x\to y}\right|^2}} \\
&= \frac{1}{1+\dfrac{1}{SNR_1}+\dfrac{1}{SNR_2}+\dfrac{1}{SNR_1SNR_2}} < 1
\end{aligned}
\tag{2-10}
$$

For simplicity, the time and frequency arguments have been omitted in Eq. (2-10).

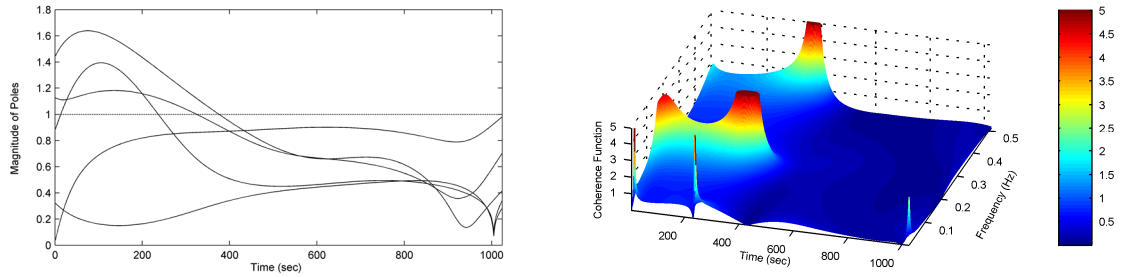
As Eq. (2-10) indicates, the value of $\left|\gamma(t,f)\right|^4$ is bounded by 1. Note that when signal-to-noise ratios SNR_1 and SNR_2 approach ∞, $\left|\gamma(t,f)\right|^4$ tends to 1, and when SNR_1 and SNR_2 approach zero, $\left|\gamma(t,f)\right|^4$ tends to zero.

As shown above, theoretically $\left|\gamma(t,f)\right|^4$ is bounded by 1. However, when the estimated transfer functions have poles lying on the unit circle, the value of $\left|\gamma(t,f)\right|^4$ may be greater than one. Note that because the system is time-varying, the poles are also naturally time-varying. When a pole is initially stable but becomes unstable (or vice versa), it will cross the unit circle periphery, as it moves from interior to exterior or vice versa. If this occurs, the magnitude of the transfer function becomes infinite. As a consequence, $\left|\gamma(t,f)\right|^4$ will have a value greater than one. Under this circumstance, the coherence relationship between two signals is undefined because the system undergoes a transition from stable to unstable or vice versa.

Stability only places restriction on the poles and not the zeros. For the inverse system (one with system function $1/H(z)$), the poles become zeros and vice versa, thus, for the minimum phase system where both poles and zeros are inside the unit circle, the maximum TVCF values will be no larger than a unit value. Thus, in the event where the estimated transfer functions do not satisfy a minimum phase system, the TVCF will also have values greater than one.

Fig. 2-2(a) shows an example of the time course of poles, which drift from instability into

stability and the resultant coherence function estimate is provided in Fig. 2-2(b). During the first 390 seconds, the magnitudes of the three poles have values outside the unit circle and consequently, coherence values are greater than one during this time period, as shown in the bottom panel of Fig. 2-2(b). As the poles cross over from the unstable to stable condition (at the instant poles cross the unit circle), even higher coherence values at two distinct frequencies are observed at time points centered around 400 seconds.



(a) (b)

Fig. 2-2. Poles crossing over the unit circle introduces high values in the estimated coherence function. (a) Magnitude of poles of the estimated transfer functions, and (b) Estimated time-varying coherence function.

It should be noted that, in certain circumstances, the method might yield greater than unit coherence values beyond the frequency band of interest. This situation may arise because signal dynamics are missing in these frequency bands, which can lead to erroneous coherence values. However, this is trivial and can be effectively ignored since the coherence values are not in the frequency range of interest.

Simulation examples are used to demonstrate both the feasibility and efficacy of the proposed method for obtaining the TVCF. In particular, the expected decrease of TV coherence values with noise and nonlinearity are demonstrated with 100 Monte Carlo realizations. The plots shown in Figs. 2-3(middle panel) ~2-5 are results of one single realization, while the averaged results are reported in Table 2-1. With the use of the previously-developed TVOPS algorithm to compute the TVTF, three parameters that need to be preset are the number of Legendre functions, the model order and the threshold value. Based on these three preset parameters, TVOPS determines only those model orders that are deemed to be significant among initial candidate model terms. Details regarding this procedure are provided in Appendix I. as well as in references [11, 12]. For both simulation examples and application of the method to the experimental data, we used the Legendre basis functions based on our *a priori* knowledge that underlying dynamics do not change abruptly. For fast-changing dynamics, more appropriate basis functions to use would be the Walsh functions. The threshold value is used for the linear independent candidate term search, as this value is dependent on the signal-to-noise ratio as well as on whether the signal is colored or white. Since *a priori* knowledge of the aforementioned conditions is unknown, we normally set the threshold value to 0.0001 and 0.001 for clean and noise-corrupted signals, respectively. Thus, for simulation examples and experimental data analyses to follow, we set the number of Legendre functions to 5 and the threshold value to 0.0001 for a noiseless simulation example and 0.001 for those cases (including experimental data) that are contaminated by noise.

To determine the proper number of basis functions, we have modified the Akaike Information criterion to:

$$AIC = N \log \sigma^2 + 2[p \cdot k + 1] \tag{2-11}$$

where N is the data length, σ^2 is the prediction error, p is the number of significant terms (not to be confused with the maximum model order), k denotes the number of basis functions. The

minimum value of the AIC is used as the criterion for determining the proper number of basis functions among the initially-chosen V sets of basis functions.

Table 2-1 Mean and standard deviations of coherence function. No statistically significant differences were observed between the three methods under the conditions reported in columns 2 (linear, noise free) and 5 (before the onset of nonlinearity). Student-Newman-Keuls test was performed; * denotes $p < 0.05$.

	Linear	*0db noise*	*Nonlinear*	*Linear-nonlinear*	
				t = 1 to 512	**t = 513 to 1024**
TVOPS	1.00 ± 0.01	0.44 ± 0.04	**0.32 ± 0.01***	0.97 ± 0.03	**0.14 ± 0.01***
STFT	0.99 ± 0.02	**0.62 ± 0.07***	**0.59 ± 0.05***	0.98 ± 0.07	**0.60 ± 0.06***
RLS	0.99 ± 0.01	0.43 ± 0.02	**0.36 ± 0.03***	0.97 ± 0.01	**0.39 ± 0.03***

2.3 The Application of the Method to Computer Simulations

2.3.1 Results of an ARMA model with noise and nonlinearity

For the first simulation example, the following TV expression ($N = 1024$) was generated with a sampling rate of 1 Hz:

$$y(n) = a_1(n)\,y(n-1) + a_2(n)\,y(n-2) + x(n) \tag{2-12}$$

with

$$\begin{aligned} a_1(n) &= 0.3\sin(2\pi n/N) + 0.2 \\ a_2(n) &= 0.09\sin(2\pi n/N) + 0.15 \end{aligned} \tag{2-13}$$

where x and y are the input and output signals, respectively. TVOPS was employed to estimate the transfer functions (both $H_{x\to y}$ and $H_{y\to x}$) with an initial model order of ARMA(6, 6) from which only 4 (out of 13) and 3 (out of 13) significant model terms were selected by the TVOPS for the estimation of $H_{x\to y}$ and $H_{y\to x}$, respectively. For the above expression, because it is linear, TV, and without noise corruption, we expect the TVCF to be of unit value for all times, as shown in the left panel of Fig. 2-3(a). Indeed, the TVCF values, shown in the middle (based on a single realization) and right (averaged over 100 realizations) panels of Fig. 2-3(a), have near unit values across all times and frequencies (see Table 2-1). For all plots in Fig. 2-3, the left, middle and right panels represent, in order, the theoretical values, a single realization estimated via the TVOPS, and the averaged estimates from applying the TVOPS to multiple realizations. The middle and right panels of Fig. 2-3(b) show the decrease of coherence values across all times and frequencies, as expected, when Eq. (2-12) is contaminated with 0 dB Gaussian White noise (GWN) that is time invariantly added to the output of Eq. (2-12). There is a good agreement with the estimated TVCF values and the theoretical value (0.44) shown in the left panel of Fig. 2-3(b). To further demonstrate another scenario of decrease in TVCF values, the following nonlinear expression was generated:

$$y(n) = a_1(n)\,y(n-1) + a_2(n)\,y(n-2) + x(n) + x^2(n) \tag{2-14}$$

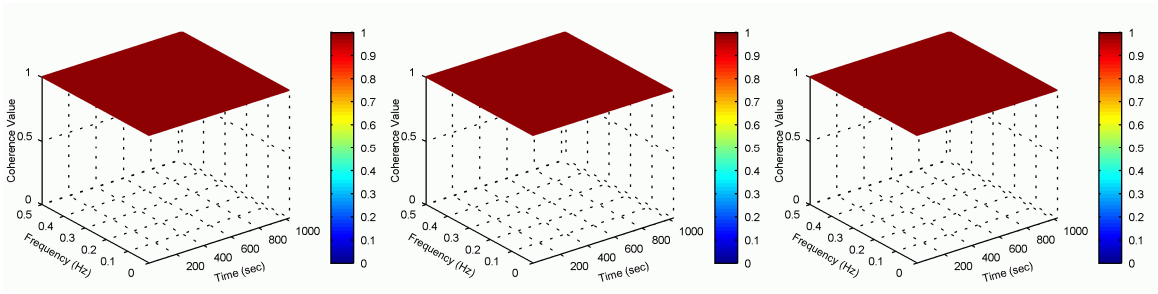
The initial model order was set to ARMA(6,6) (for both $H_{x\to y}$ and $H_{y\to x}$) from which a total of 4 and 6 significant model terms were selected for the estimation of $H_{x\to y}$ and $H_{y\to x}$,

respectively. The linear terms in Eq. (2-14) are the same as shown in Eq. (2-12). Note in Fig. 2-3(c) the significantly lower coherence values than in the linear case, for all times and frequencies, due to the overall TV nonlinearity of the system response of Eq. (2-14). The estimated TVCF values shown in the middle and right panels correspond well to the theoretically expected TVCF value (0.33) shown in the left panel of Fig. 2-3(c).

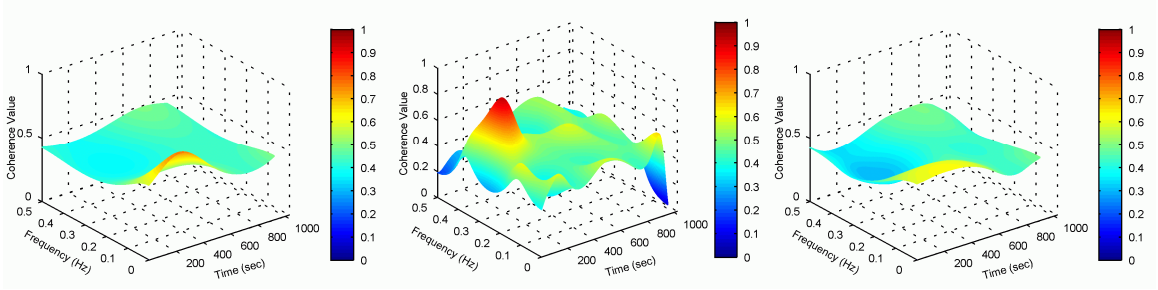
The next simulation example considers the time course of a noiseless TV linear and nonlinear system as described by Eqs. (2-12 ~ 2-13) and (2-14). The linear dynamic (Eq. (2-12 ~ 2-13)) occurs from 0 to 512 seconds and the nonlinear dynamic (Eq. (2-14)) occurs thereafter. The initial model order was set to ARMA(6,6) (for both $H_{x\to y}$ and $H_{y\to x}$) from which 6 significant parameters were selected for both $H_{x\to y}$ and $H_{y\to x}$. The results of the TVCF for this combination of TV linear and nonlinear systems are shown in the middle and right panels of Fig. 2-3(d). Correctly shown in the middle and right panels of Fig. 2-3(d), high coherence values are observed from 0 to 512 seconds followed by low coherence values thereafter. Thus, the proposed approach, due to its high-resolution capability, is able to show transition from high coherence to low coherence values. Note that the TVCF obtained using a technique such as the short time Fourier transform (STFT) would have difficulty in pinpointing this transition from high to low coherence values due to its requirement of segmenting the data record. For example, if the segmentation did not occur exactly at the time point 512 seconds, then the STFT would not be able to show this clear transition from high to low coherence values.

For the middle and right panels of Fig. 2-3(d), the TVCF values for the nonlinear period are lower than the theoretical value, shown in the left panel in Fig. 2-3(d). This lower coherence value is due to greater prediction error for the nonlinear portion of the signal. Greater prediction error results because we restrict the estimated parameters to be linear and are limited to the 6 most significant parameters for the entire signal, thus, the linear portion of the signal will be better captured than the nonlinear portion, which consequently results in lower coherence values for the nonlinear portion of the signal. In Fig. 2-3(c), TVCF values via the TVOPS are very close to the theoretical value, since all the selected parameters were solely devoted to fitting nonlinear data. The RLS estimated TVCF values, shown in Fig. 2-3(d), are closer to the theoretical value because RLS uses far more parameters than does the TVOPS. For example, the RLS uses 6 parameters at each time instant, resulting in a total of 6144 parameters, whereas the TVOPS uses only 72 parameters for the entire data.

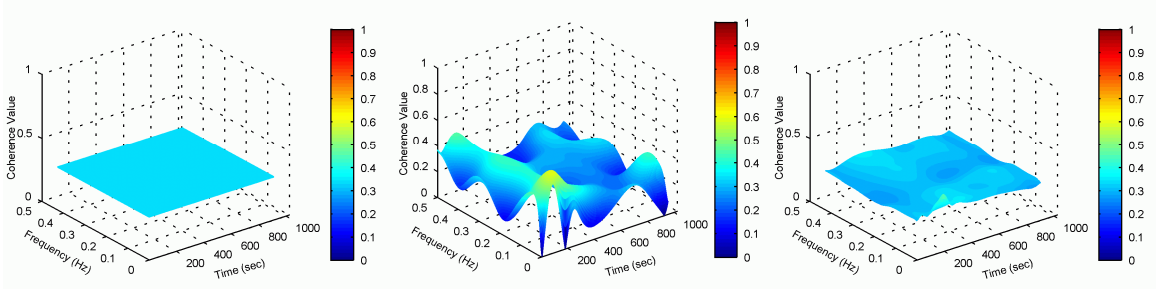
To further demonstrate the efficacy of the proposed method, we applied the method to surrogate data, which was generated from the input and output of Eqs. (2-12) ~ (2-13) and (2-14) (a linear and nonlinear system). The initial model order was set to ARMA(6,6) (for both $H_{x\to y}$ and $H_{y\to x}$) from which 4 significant parameters were selected for both $H_{x\to y}$ and $H_{y\to x}$. Surrogate data technique destroys any linear and nonlinear correlations that may exist between the input and output data. Therefore we would expect low coherence values across all times. This is exactly what Fig. 2-3(e) shows.



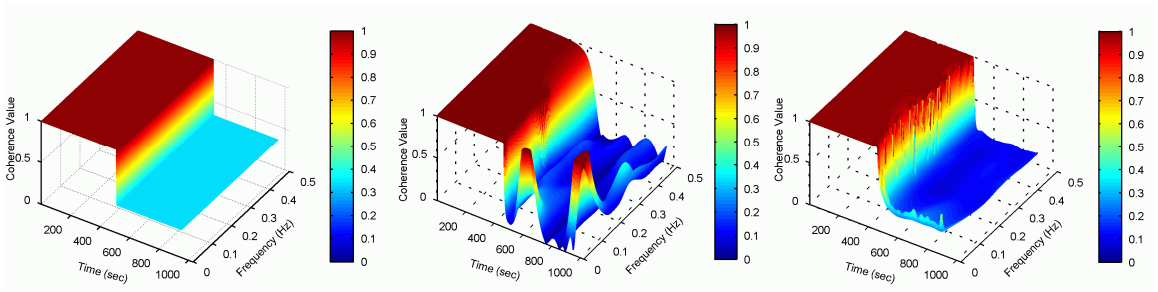
(a)



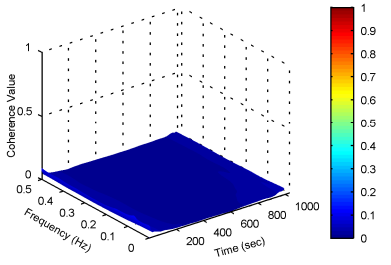
(b)



(c)



(d)



(e)

Fig. 2-3. The left panels are the theoretical TVCF values; the middle are the TVCFs via TVOPS based on a single realization; and the right panels are the averaged TVCFs via TVOPS based on 100 realizations. a) linear system, b) linear system with additive 0 dB noise, c) nonlinear system, d) linear system followed by nonlinear system, e) surrogate data of combined linear and nonlinear systems shown in (d).

Figs. 2-4(a) ~ (d) show comparative results based on the STFT for the same simulations as described above with the proposed method. As compared to Figs. 2-3(a) ~ (d), the STFT has a very low resolution both in time and frequency domains. This is an unavoidable and well-documented limitation of the STFT. While these particular simulation examples involve 1024 data points, had there been an even smaller number of data points, the utility of the STFT would be questionable. Another deficiency that is apparent is the relatively high coherence values even when the signal is contaminated with noise as well as when the system is nonlinear. The third row of Table 2-1 provides mean TVCF values via the STFT for all four cases considered. As compared to the second row of Table 2-1, coherence values are significantly higher with the two aforementioned cases, which should have resulted in low coherence values.

We have also compared the proposed approach to the multiple windows using Slepian sequences approach [20]. The results are slightly better than with the STFT, but a similar resolution limitation as with the STFT remains with the multiple windows using Slepian sequences method.

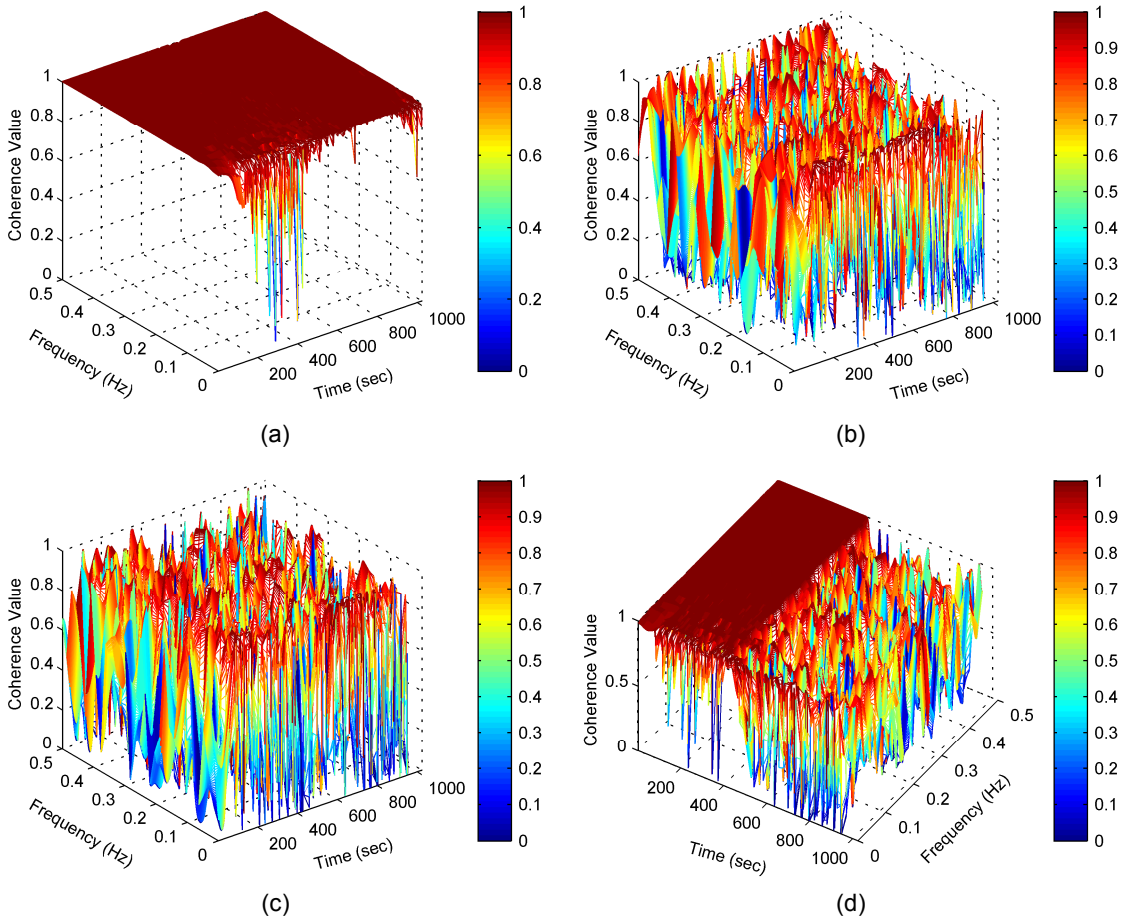


Fig. 2-4. TVCF via the STFT: (a) linear system, no noise, (b) linear system with additive 0 dB noise, (c) nonlinear system, no noise, and (d) linear system followed by nonlinear system, no noise.

2.3.2 Comparing the new method to RLS and STFT method

In order to compare the performance of our method with the Recursive Least Squares approach (RLS), we show the corresponding results of the RLS in Figs. 2-5(a) ~ (d). The fourth

row of Table 2-1 shows the mean coherence values for Figs. 2-5(a) ~ (d). The model order for the RLS was based on the minimum description length followed by the coefficient estimates using the RLS, and the subsequent calculation of the TVTF and TVCF using Eq. (2-6) and Eq. (2-4), respectively. As the figures show, RLS also gives consistently better time-frequency resolution than the STFT while providing similar time-frequency resolution to the proposed method. It is interesting to note that the RLS has an edge effect similar to a high amplitude oscillation at the very low (~0 Hz) and high frequencies (~ 0.5 Hz), and they occur only when the system is nonlinear (Fig. 2-5(c) and (d)). The estimated parameters are unstable during the nonlinear case (not shown). The recursive nature of the RLS, which may lead to over-parameterization and unstable model parameters in nonlinear systems, can be the source of the high amplitude oscillations seen in Figs. 2-5(c) and (d).

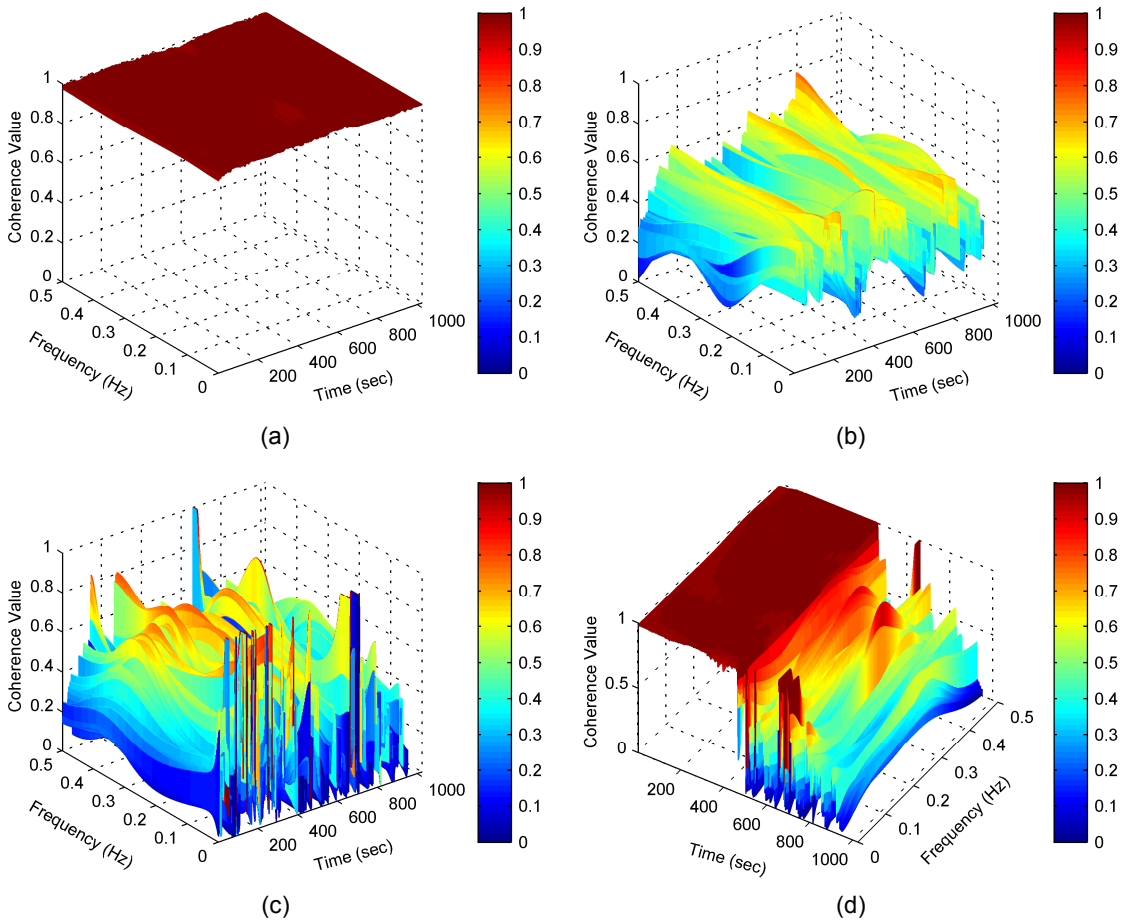


Fig. 2-5. TVCF via the RLS: (a) linear system, no noise, (b) linear system with additive 0 dB noise, (c) nonlinear system, no noise, and (d) linear system followed by nonlinear system, no noise.

2.3.3 Time-Frequency resolution comparison

The following simulation example explores time and frequency resolution of the TVCF using comparison between the proposed method and the STFT. The simulation is based on a linear complex frequency modulation signal. For this case, we consider coherences between real and complex portions of the signal. A GWN signal was added to the real portion of the signal and another GWN signal, independent from the first, was added to the complex portion of the signal. In both cases, the resultant signal-to-noise ratio was equal to 20 dB. It is expected that

high coherences should be observed only during linear increasing frequencies and time. Fig. 2-6(a) and (b) are the result of the proposed method and the STFT, respectively. It is apparent that only the proposed method is able to correctly discriminate high coherence values only during the linearly-increasing frequencies and time. Due to the limited time-frequency resolution of the STFT, it is not as accurate as the proposed method in delineating high coherence values appearing only at frequencies that are linearly increasing with time.

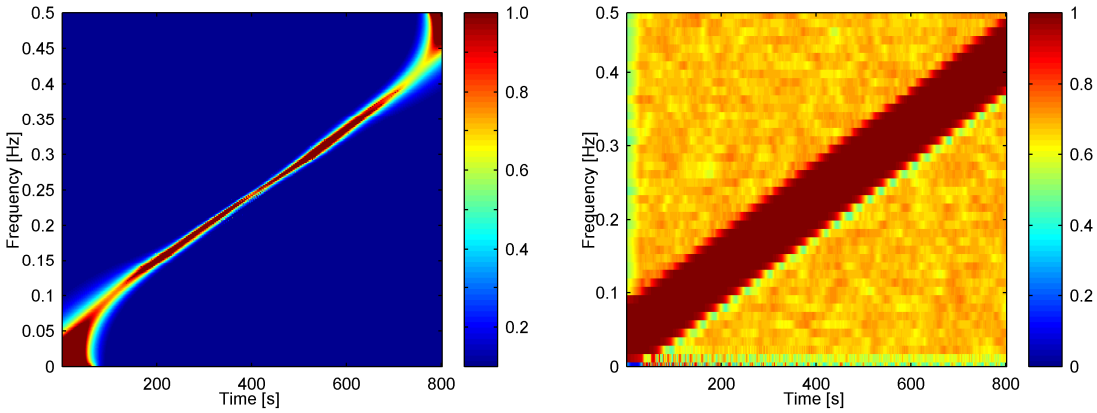


Fig. 2-6. Comparison of TVOPS and the STFT for a complex linear chirp signal. (a) Result of TVOPS; and (b) Result of STFT.

2.4 Application of the method to renal system data

The proposed method is applied to study the nonstationary features of the renal autoregulatory mechanisms and how these features might be different between time-invariant and TV correlation functions for normotensive rats. Renal autoregulation is the process in which fluctuations in renal blood flow caused by fluctuations in blood pressure are minimized. The myogenic mechanism and tubuloglomerular feedback (TGF) are the two mechanisms known to be responsible for renal autoregulation. The myogenic mechanism and TGF are shown to oscillate in the frequency range of 0.1 ~ 0.2 Hz and 0.03 ~ 0.05 Hz, respectively [26-28].

To estimate the TVCF of the renal autoregulatory system, the renal blood flow data were recorded under broadband forced arterial blood pressure fluctuations. The detailed experimental data collection procedures can be found in reference [28].

Data analysis is based on 4 recordings from normotensive rats. Each of the experimental data records used for analysis was 256 seconds long, with a sampling rate of one sample per second, after digital low-pass filtering to avoid aliasing. Each data record, containing 256 data points, was subjected to second-degree polynomial trend removal (which included demeaning) and was normalized to unit variance. Representative time series of the blood pressure (top panel) and blood flow (bottom panel) are shown in Fig. 2-7.

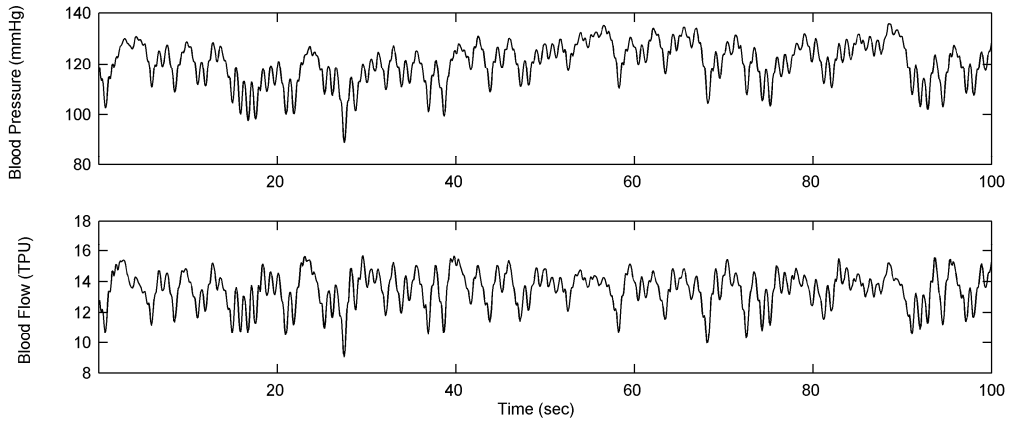


Fig. 2-7. Representative blood pressure and blood flow data. Top panel: blood pressure data; and bottom panel: blood flow data.

The estimated TVCFs for the 4 normotensive rats are shown in the left panels of Fig. 2-8. The top right panels of Fig. 2-8 show a portion of the TV coherence values of the left panels from 0.03 to 0.05 Hz. These curves represent the frequency range where the TGF mechanism is known to operate and is shown for the readers' convenience. The bottom-most curves represent 0.03 Hz and the top-most curves represent 0.05 Hz. The bottom right panels represent the time-invariant coherence functions, shown for convenient comparison to the TVCFs.

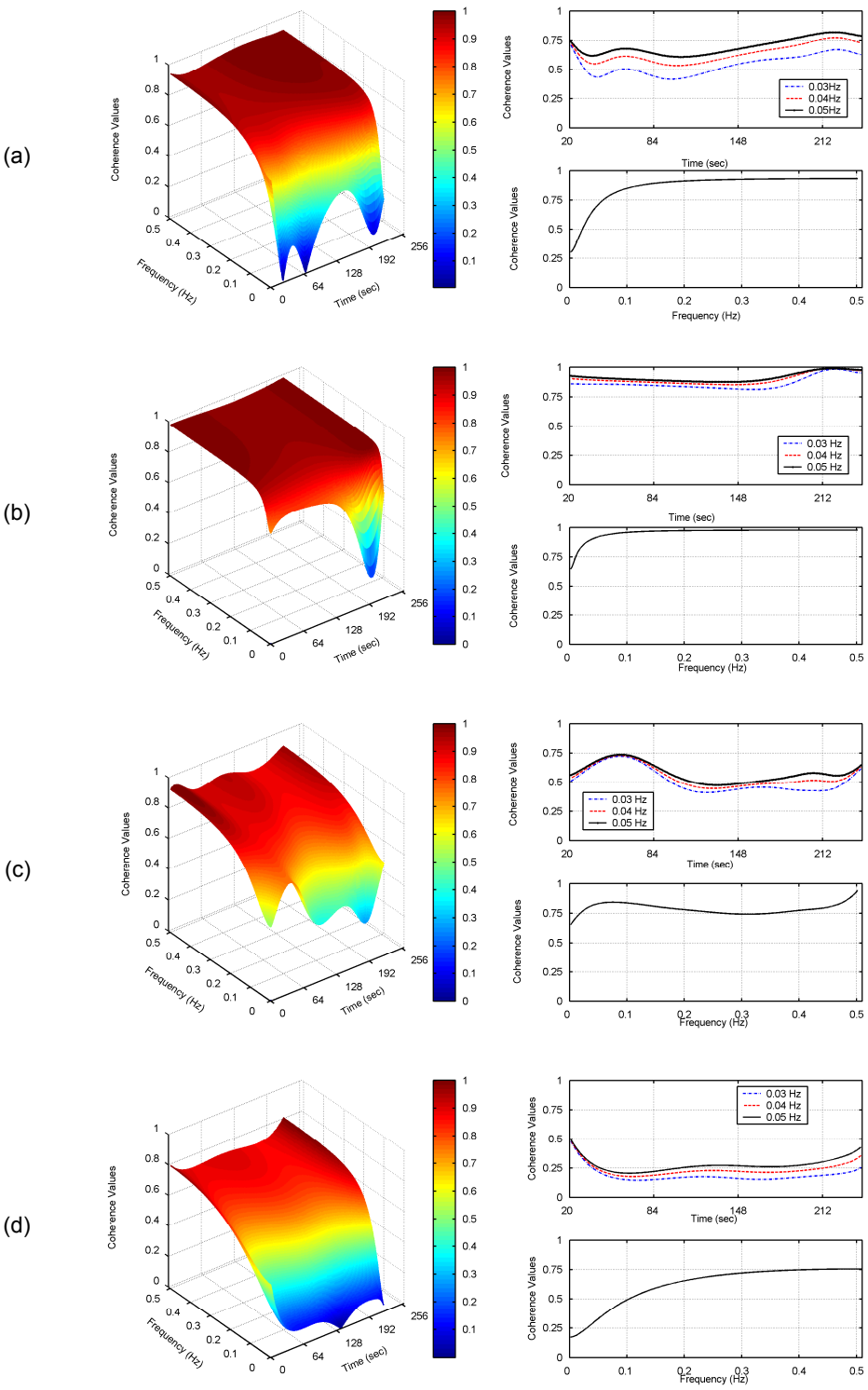


Fig. 2-8. (a) ~ (d): TVCFs on renal blood flow and pressure data obtained from 4 rats. Top right panels show a portion of the TV coherence values of the left panels from 0.03 to 0.05 Hz. Bottom-right panels represent time-invariant coherence values.

For the myogenic mechanism (0.1-0.2 Hz), high coherence values are observed for all times for all four rats. Time invariant coherence functions also show high coherence values for all four rats at the myogenic mechanism's frequency range. The TGF, however, shows lower coherence values that are more time-dependent than those of the myogenic mechanism. The low coherence values for the TGF with the TVCF estimate are in agreement with the time-invariant coherence function estimates as well as previous other data analyses [26-28]. Two of the four plots, in the region of TGF show time dependence, as the coherence values fluctuate above or lower than the value of 0.5 (plots a and c). Note that this type of observation is lost with the time-invariant coherence functions. The observation of low coherence values for the TGF mechanism (plots (a), (c) and (d)) is expected, as previous reports have amply demonstrated nonlinear characteristics of the TGF mechanism [29, 30]. Furthermore, fluctuations of TVCF values seen in Fig. 2-8(a) and (c) are all greater than the time constant of TGF (~ 33 sec). The new insight gained from our data analysis is that nonlinearity does not occur for all times (at certain time points, TV coherence values are greater 0.5), but rather at certain time points. The physiological implication of the increase or decrease of coherence values for the renal blood pressure and flow data is that at certain time points, there is either a high or low degree of coherence between two sets of data (at certain frequency ranges), respectively. It is expected that coherence values fluctuate over time since renal autoregulation is a dynamic process whereby the myogenic and TGF mechanisms work in concert to maintain steady renal blood flow while blood pressure varies over a wide pressure range. For a complete understanding of the physiological basis of nonstationary and nonlinear characteristics of the TGF mechanism, we will require further data analyses on more experimental data, which is currently ongoing in our laboratory.

2.5 Summary

TVCF based on variants of the STFT are simple to calculate. However, high resolution and meaningful estimates of the TVCF have been unattainable. A simple and yet high-resolution TVCF technique has been introduced and is based on our previously developed TVTF estimate. Specifically, the method is based on multiplication of transfer function estimates. The accuracy of the TVCF, thus, depends on the precise estimation of the TVTF. Both simulation examples and application to renal data have shown that in most cases, the proposed method is reliable and yields coherence values within the bounds of 0 and 1. However, we have shown that with the proposed approach, in certain cases, TVCF values can be greater than 1. This scenario occurs when the poles are either on the unit circle periphery or as they undergo transition from stable to unstable poles (or vice versa), in which case these poles will lie on the unit circle for an instantaneous moment of time. Poles on the unit circle will result in coherence values greater than 1. In certain biological data, the aforementioned scenario may occur. For example, signals that bifurcate from stable to unstable dynamics or bifurcation from deterministic to stochastic systems may all result in larger than unit coherence values. Note that noise and nonlinearity are two contributors to low coherence values, sudden spikes in the data, which can be due to transient noise glitches, will result in low coherence values since the linear TVTF is not suited to capture nonlinear characteristics of sudden spikes.

Furthermore, it needs to be stressed that the accuracy of the proposed approach depends on the level of broadband characteristics of both the input and output signals; better estimates are obtained with broadband signals than with narrow band signals. With time-invariant coherence function analysis, low coherence values are usually attributed to noise, nonlinearity or nonstationarity. However, with a technique capable of estimating reasonably accurate TVCF, the possible cause of the low coherence values due to nonstationarity, at least, can be eliminated.

Chapter 3 Development of Time-varying Causal Coherence Functions [1]

3.1 Introduction

The coherence function (CF) is a general tool used to investigate the correlation relationship between two signals in the frequency domain. The CF is a technique that can also be used to determine the appropriateness of performing linear analyses on given data. The biological community has readily applied the CF to many diverse sets of experimental data, recognizing the useful information it provides. For example, it was found that one of the two mechanisms responsible for renal blood flow autoregulation, tubuloglomerular feedback (TGF), shows low coherence at 0.02 ~ 0.05 Hz [31-34]. This indicates that linear modeling of renal autoregulation is not adequate.

The CF, however, is most useful when applied to open-loop systems. A work by Möller et al. uses least mean squares (LMS) to estimate traditional time-varying coherence functions based on a multivariate autoregressive moving average model of event-related potentials[35]. Recently, a new algorithm based on multiplication of time-varying transfer functions was used to estimate the traditional time-varying coherence function (TVCF) [36]. To avoid the slow convergence of the LMS algorithm, this recently developed algorithm uses basis functions to obtain time-varying coefficients, and has been shown to provide accurate results.

Porta et al. have recently developed a method for estimating a causal time-invariant CF, and with its application to cardiovascular data, they have demonstrated that the use of the traditional CF on a causal system may provide an incomplete description of the system [37]. The novelty of Porta's method resides in its ability to detect the causal correlation relationship between two signals. For example, it can quantify how many of the fluctuations in arterial blood pressure are correlated to heart rate, and vice versa. An additional contribution of this work is the use of the vector autoregressive (VAR) model, as the power spectrum approach to calculating the causal CF cannot be used for causal systems.

While the contributions by Porta et al. are significant, their theoretical derivation of the causal CF using the VAR model is incomplete. Specifically, it is unclear whether the VAR-based approach also provides coherence values that are bounded between 0 and 1. Furthermore, the full utility of the method is not realized since the method is based on assuming time-invariance.

This chapter describes the development of a model-based approach to estimating both feedforward and feedback paths of causal time-varying coherence functions (TVCF). Theoretical derivations of the coherence bounds of the causal TVCF using the proposed approach are also provided. Both theoretical derivations and simulation results revealed interesting observations, and they were corroborated using experimental renal blood pressure and flow data. Specifically, both theoretical derivations and experimental data showed that in certain cases, the calculation of the traditional TVCF was inappropriate when the system under investigation was a causal system. Moreover, the use of the causal TVCF not only provides quantitative assessment of the coupling between the two signals, but it also provides valuable insights into the composition of the physical structure of the renal autoregulatory system.

1 This work was supported in part by a grant from NIH HL69629. Relevant work has been published on *IEEE Transaction On Biomedical Engineering*, Vol. 54, Issue 12, pp 214-2150, 2007

3.2 Mathematic Derivation of the TVCCF

In this section, the mathematic model of the TVCCF is developed. Based on this mathematic model, the theoretical bounds of the TVCCF are obtained by mathematic derivations. The analysis has shown that the TVCCF is bounded by 0 and 1. Thus, the validity of the proposed method is justified.

3.2.1 Mathematic model

A bivariate causal system can be modeled as a VAR process:

$$\begin{aligned} x_1(n) &= \sum_{k=1}^{p_1} a_{11}(n,k)\,x_1(n-k) + \sum_{k=0}^{q_1} a_{12}(n,k)\,x_2(n-k) + e_1(n) \\ x_2(n) &= \sum_{k=0}^{q_2} a_{21}(n,k)\,x_1(n-k) + \sum_{k=1}^{p_2} a_{22}(n,k)\,x_2(n-k) + e_2(n) \end{aligned} \tag{3-1}$$

where $e_1(n)$ and $e_2(n)$ are uncorrelated white noise. The unknown model coefficients, a_{11}, a_{12}, a_{21} and a_{22}, are all time-varying since they are functions of time and need to be estimated. Rewriting (1) in a vector form, we obtain:

$$\mathbf{x}(n) = \mathbf{A}(n,k) \otimes \mathbf{x}(n) + \mathbf{e}(n) \tag{3-2}$$

where the symbol $\otimes$ in Eq.(3-2) denotes the convolution operator:

$$\mathbf{x}(n) = \left[x_1(n), x_2(n)\right]^T, \mathbf{A}(n,k) = \begin{pmatrix} A_{11}(n,k) & A_{12}(n,k) \\ A_{21}(n,k) & A_{22}(n,k) \end{pmatrix}, \mathbf{e}(n) = \left[e_1(n), e_2(n)\right]^T$$

and

$$\begin{aligned} A_{11}(n,k) &= \left[a_{11}(n,1), a_{11}(n,2), \cdots, a_{11}(n,p_1)\right] \\ A_{12}(n,k) &= \left[a_{12}(n,1), a_{12}(n,2), \cdots, a_{12}(n,q_1)\right] \\ A_{21}(n,k) &= \left[a_{21}(n,1), a_{21}(n,2), \cdots, a_{21}(n,q_2)\right] \\ A_{22}(n,k) &= \left[a_{22}(n,1), a_{22}(n,2), \cdots, a_{22}(n,p_2)\right] \end{aligned}$$

The unknown model coefficients can be obtained by using a recursive least squares adaptive filter algorithm [35], or by using a basis function approach [12]. Note that Porta et al. [37] have used a variant of the least squares approach to obtain unknown time-invariant model coefficients.

Similar to the derivation of the time-invariant coherence function [37], it can be shown that the traditional TVCF is defined by the ratio between time-varying cross-spectrum and time-varying autospectra:

$$\gamma^2(n,f) = \frac{\left|s_{12}(n,f)\right|^2}{s_{11}(n,f)\,s_{22}(n,f)} \tag{3-3}$$

with time-varying autospectra ($s_{11}(n,f)$ and $s_{22}(n,f)$) and cross-spectrum ($s_{12}(n,f)$) defined in terms of time-varying AR filters:

$$
\begin{aligned}
s_{11}(n,f) &= \left\{\sigma_1^2\left[1-A_{22}(n,z)\right]\left[1-A_{22}^*(n,z)\right]+\sigma_2^2 A_{12}(n,z)A_{12}^*(n,z)\right\}\Big/\left|\Delta(n,z)\right|^2\Big|_{z=e^{j2\pi f\Delta t}} \\
&= \left\{\sigma_1^2\left|1-A_{22}(n,z)\right|^2+\sigma_2^2\left|A_{12}(n,z)\right|^2\right\}\Big/\left|\Delta(n,z)\right|^2\Big|_{e^{j2\pi f\Delta t}} \\
s_{12}(n,f) &= \left\{\sigma_1^2\left[1-A_{22}(n,z)\right]A_{21}^*(n,z)+\sigma_2^2 A_{12}(n,z)\left[1-A_{11}^*(n,z)\right]\right\}\Big/\left|\Delta(n,z)\right|^2\Big|_{z=e^{j2\pi f\Delta t}} \\
s_{22}(n,f) &= \left\{\sigma_1^2 A_{21}(n,z)A_{21}^*(n,z)+\sigma_2^2\left[1-A_{11}(n,z)\right]\left[1-A_{11}^*(n,z)\right]\right\}\Big/\left|\Delta(n,z)\right|^2\Big|_{z=e^{j2\pi f\Delta t}} \\
&= \left\{\sigma_1^2\left|A_{21}(n,z)\right|^2+\sigma_2^2\left|1-A_{11}(n,z)\right|^2\right\}\Big/\left|\Delta(n,z)\right|^2\Big|_{z=e^{j2\pi f\Delta t}}
\end{aligned}
\tag{3-4}
$$

where $\Delta(n,z)=\left[1-A_{11}(n,z)\right]\left[1-A_{22}(n,z)\right]-A_{12}(n,z)A_{21}(n,z)$, and σ_1^2 and σ_2^2 are the variances of $e_1(n)$ and $e_2(n)$, respectively at each time instant n.

Similarly, the causal TVCF is calculated by setting either $A_{12}(n,z)$ (feedforward) or $A_{21}(n,z)$ (feedback) to zero [37]:

$$
\begin{aligned}
\gamma_{1\to 2}^2(n,f) &= \gamma^2(n,f)\Big|_{A_{12}(n,z)=0,\,z=e^{j2\pi f\Delta t}} \\
\gamma_{2\to 1}^2(n,f) &= \gamma^2(n,f)\Big|_{A_{21}(n,z)=0,\,z=e^{j2\pi f\Delta t}}
\end{aligned}
\tag{3-5}
$$

Thus, with the causal TVCF we obtain both feedforward and feedback TVCF, whereas for the traditional TVCF, we obtain only one TVCF value. Derivations of theoretical bounds for both the traditional and the causal TVCF values are presented in the proceeding section.

3.2.2 Derivation of the theoretical bounds of TVCF

To derive theoretical bounds for both the traditional and the causal TVCF, we define time-varying transfer functions between the two signals:

$$
H_1(n,z)=\frac{A_{12}(n,z)}{1-A_{11}(n,z)},\ H_2(n,z)=\frac{A_{21}(n,z)}{1-A_{22}(n,z)} \tag{3-6}
$$

For simplicity, the time and frequency arguments have been omitted in the following derivations. It can be shown that with proper substitutions the causal TVCF can be defined as:

$$\gamma^2=\frac{|s_{12}|^2}{s_{11}s_{22}}=\left.\frac{\left|\sigma_1^2\left[1-A_{22}\right]A_{21}^*+\sigma_2^2A_{12}\left[1-A_{11}^*\right]\right|^2}{\left\{\sigma_1^2|1-A_{22}|^2+\sigma_2^2|A_{12}|^2\right\}\left\{\sigma_1^2|A_{21}|^2+\sigma_2^2|1-A_{11}|^2\right\}}\right|_{z=e^{j2\pi f\Delta t}}$$

$$=\left.\frac{\left[\sigma_1^2A_{21}A_{21}^*/H_2+\sigma_2^2A_{12}A_{12}^*/H_1^*\right]\left[\sigma_1^2A_{21}A_{21}^*/H_2+\sigma_2^2A_{12}A_{12}^*/H_1^*\right]^*}{\sigma_1^4|A_{21}|^4/|H_2|^2+\sigma_1^2\sigma_2^2|A_{12}|^2|A_{21}|^2\left[1+1/\left(|H_1|^2|H_2|^2\right)\right]+\sigma_2^4|A_{12}|^4/|H_1|^2}\right|_{z=e^{j2\pi f\Delta t}}$$

$$=\left.\frac{\sigma_1^4|A_{21}|^4/|H_2|^2+\sigma_1^2\sigma_2^2|A_{21}A_{12}|^2\left[1/\left(H_1H_2\right)+1/\left(H_1^*H_2^*\right)\right]+\sigma_2^4|A_{12}|^4/|H_1^*|^2}{\sigma_1^4|A_{21}|^4/|H_2|^2+\sigma_1^2\sigma_2^2|A_{12}A_{21}|^2\left[1+1/\left(|H_1|^2|H_2|^2\right)\right]+\sigma_2^4|A_{12}|^4/|H_1|^2}\right|_{z=e^{j2\pi f\Delta t}}$$

$$=\left.\frac{\sigma_1^4|A_{21}|^4|H_1|^2+\sigma_2^4|A_{12}|^4|H_2|^2+\sigma_1^2\sigma_2^2|A_{21}A_{12}|^2|H_1H_2|^2\left[1/\left(H_1H_2\right)+1/\left(H_1^*H_2^*\right)\right]}{\sigma_1^4|A_{21}|^4|H_1|^2+\sigma_1^2\sigma_2^2|A_{12}A_{21}|^2\left(|H_1|^2|H_2|^2+1\right)+\sigma_2^4|A_{12}|^4|H_2|^2}\right|_{z=e^{j2\pi f\Delta t}} \tag{3-7}$$

We define the following:

$$\alpha=\sigma_1^2|H_1||A_{21}|^2\geq 0,\beta=\sigma_2^2|H_2||A_{12}|^2\geq 0,c_H=H_1H_2 \tag{3-8}$$

The last numerator term in the last line of (3-7) can be simplified by:

$$\begin{aligned}&|H_1H_2|^2\left[1/\left(H_1H_2\right)+1/\left(H_1^*H_2^*\right)\right]=\left(H_1H_2H_1^*H_2^*\right)/\left(H_1H_2\right)+\left(H_1H_2H_1^*H_2^*\right)/\left(H_1^*H_2^*\right)\\&=H_1^*H_2^*+H_1H_2=|H_1H_2|\times\frac{H_1^*H_2^*+H_1H_2}{|H_1H_2|}\\&=|H_1H_2|\times\left(\frac{c_H^*}{|c_H|}+\frac{c_H}{|c_H|}\right)=|H_1H_2|\times 2\times real\left(\frac{c_H}{|c_H|}\right)\end{aligned} \tag{3-9}$$

Substituting. Eq. (3-8) and Eq. (3-9) into (3-7) yields:

$$\gamma^2=\frac{|s_{12}|^2}{s_{11}s_{22}}=\frac{\alpha^2+\beta^2+2\alpha\beta\times real\left(c_H/|c_H|\right)}{\alpha^2+\beta^2+\alpha\beta\left(|c_H|+1/|c_H|\right)} \tag{3-10}$$

Given the following:

$$\alpha^2+\beta^2+2\alpha\beta\times real\left(c_H/|c_H|\right)\leq\alpha^2+\beta^2+2\alpha\beta\times 1=\alpha^2+\beta^2+2\alpha\beta \tag{3-11}$$

$$\alpha^2+\beta^2+\alpha\beta\left(|c_H|+1/|c_H|\right)\geq\alpha^2+\beta^2+\alpha\beta\times 2\sqrt{|c_H|\times 1/|c_H|}=\alpha^2+\beta^2+\alpha\beta\times 2 \tag{3-12}$$

and substituting. Eq. (3-11) and Eq. (3-12) into Eq. (3-10) yields:

$$\gamma^2=\frac{|s_{12}|^2}{s_{11}s_{22}}=\frac{\alpha^2+\beta^2+2\alpha\beta\times real\left(c_H/|c_H|\right)}{\alpha^2+\beta^2+\alpha\beta\left(|c_H|+1/|c_H|\right)}\leq\frac{\alpha^2+\beta^2+2\alpha\beta}{\alpha^2+\beta^2+\alpha\beta\times 2}=1 \tag{3-13}$$

Thus, the traditional TVCF value is bounded by 1. Eqs. (3-11) to (3-13) suggest that the traditional TVCF is high if c_H is very close to $1,\alpha\gg\beta$, or $\beta\gg\alpha$.

The causal TVCF from signal $x_1(n)$ to signal $x_2(n)$ is calculated by:

$$\gamma_{1\to 2}^2(n,f)=\gamma^2(n,f)\Big|_{A_{12}(n,z)=0,z=e^{j2\pi f\Delta t}}=\frac{\left|\sigma_1^2[1-A_{22}]A_{21}^*\right|^2}{\sigma_1^2\left|1-A_{22}\right|^2\left(\sigma_1^2\left|A_{21}\right|^2+\sigma_2^2\left|1-A_{11}\right|^2\right)}$$
$$=\frac{\sigma_1^2\left|A_{21}^*\right|^2}{\sigma_1^2\left|A_{21}\right|^2+\sigma_2^2\left|1-A_{11}\right|^2}=\frac{1}{1+\left[\left|1-A_{11}\right|^2\sigma_2^2\right]\Big/\left[\left|A_{21}^*\right|^2\sigma_1^2\right]}=\frac{1}{1+a^2} \tag{3-14}$$

where $a=\left[\left|1-A_{11}\right|\sigma_2\right]\Big/\left[\left|A_{21}^*\right|\sigma_1\right]$. Similarly, the causal TVCF from signal $x_2(n)$ to signal $x_1(n)$ is described by:

$$\gamma_{2\to 1}^2(n,f)=\gamma^2(n,f)\Big|_{A_{12}(n,z)=0,z=e^{j2\pi f\Delta t}}=\frac{1}{1+b^2} \tag{3-15}$$

where $b=\left[\left|1-A_{22}\right|\sigma_1\right]\Big/\left[\left|A_{12}\right|\sigma_2\right]$. Note that:

$$\left|c_H\right|=\left|H_1H_2\right|=\left|\frac{A_{12}}{1-A_{11}}\frac{A_{21}}{1-A_{22}}\right|=\left|\frac{1}{\left[\left|1-A_{11}\right|\sigma_2\right]\Big/\left[\left|A_{21}^*\right|\sigma_1\right]}\frac{1}{\left[\left|1-A_{22}\right|\sigma_1\right]\Big/\left[\left|A_{12}\right|\sigma_2\right]}\right|=\frac{1}{ab} \tag{3-16}$$

As a consequence, the following corollary holds:

$$\gamma_{1\to 2}^2(n,f)+\gamma_{2\to 1}^2(n,f)=\frac{1}{1+a^2}+\frac{1}{1+b^2}=\frac{a^2+b^2+2}{1+a^2+b^2+a^2b^2}\approx\frac{a^2+b^2+2}{2+a^2+b^2}=1 \tag{3-17}$$

In Eq. (3-17), when both a and b are equal to 1, the traditional TVCF equals one, while the causal TVCF (feedforward and feedback) can have values of only 0.5. In other words, what Eq. (3-17) indicates is that a high traditional coherence value between two signals is not necessarily reflective of a strongly coupled causal system, whereas a low causal coherence function does reflect a weakly coupled causal system. An example illustrating this scenario is presented in Fig. 3-7. See details in 3.3 and 3.4.

The above derived bounds for the TVCF are predicated on the premise that the transfer functions noted in Eqs. (3-6) and (3-7) are stable, in which case the bounds of the TVCF follow from those of the time-invariant coherence function [37]. The above derived TVCF bounds also establish a formal extension to the case of unstable transfer function estimates. We have shown in our previous work that when the estimated transfer functions have poles lying on or outside the unit circle, the coherence values can be greater than one [36]. Under this circumstance, the coherence relationship between two signals is undefined.

In 3.3, a bivariate causal process as detailed below was used to examine the efficacy of the proposed method. In addition, application of the method to renal blood pressure and flow data is provided. The time-varying optimal parameter search (TVOPS) was employed to identify the time-varying model coefficients[12, 38]. For the TVOPS, the following parameters have to be known a priori: (1) the number of basis functions to be used, (2) the initial model orders, and (3) the threshold number to be used to reduce linear redundancy among the observation vectors.

3.3 Simulation Examples and Results

3.3.1 Simulation of a linear causal system

Fig. 3-1 illustrates the structure of a linear causal system represented by the following expression:

$$\begin{cases} y_1(n) = -c_{21}(n)\, y_2(n-2) + e_1(n) \\ y_2(n) = c_{12}(n)\, y_1(n-1) + e_2(n) \end{cases} \tag{3-18}$$

where $e_1(n)$ and $e_2(n)$ are independent Gaussian white noise with unit variance. For convenience of notation, the 1st subsystem, $y_1(n)$, is defined as the "feedforward" and the 2nd subsystem, $y_2(n)$, as the "feedback" system. Linear interactions between the feedforward and feedback systems are generated by the time-varying coupling coefficient terms $c_{21}(n)$ and $c_{12}(n)$, as shown in Eq.(3-18). The coupling strength coefficient, $c_{21}(n)$, representing information flow from $y_2(n)$ to $y_1(n)$ decreases from 1.8 to 0 while the coupling strength coefficient, $c_{12}(n)$, representing information flow from $y_1(n)$ to $y_2(n)$, increases from 0 to 1.8.

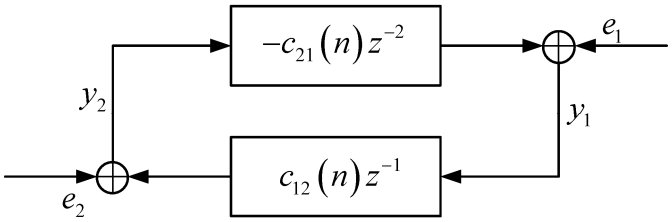


Fig. 3-1. Block diagram of a causal system with feedforward and feedback paths. The coupling strengths are determined by the two coefficients c_{12} and c_{21}.

3.3.2 Simulation model based on the Poincaré oscillator

Fig. 3-2 depicts the structure of the 2-subsystem Poincaré oscillator. For convenience of notation, the 1st subsystem is defined as the "Feedforward" and the 2nd subsystem as the "Feedback" systems. Linear interaction between the feedforward and feedback systems are introduced by the coupling terms $\left(g_{x_1}, g_{y_1}\right)^T$ and $\left(g_{x_2}, g_{y_2}\right)^T$, shown in Fig. 3-2. If these two coupling terms are missing, then there is no interaction and the two subsystems will oscillate at their own characteristic frequencies ω_1 and ω_2. The interaction term $\left(g_{x_1}, g_{y_1}\right)^T$ introduces frequency ω_2 into the feedforward, and vice versa for the feedback subsystem with $\left(g_{x_2}, g_{y_2}\right)^T$. A high causal coherence value is expected from the feedback to feedforward at frequency ω_2 if the coupling coefficient $\eta_{2\to 1}$ value is greater than 0.1. Likewise from the feedforward to feedback if the value of $\eta_{1\to 2}$ is greater than 0.1. Theoretically, in the noise-free case and provided that the coupling coefficients are greater than 0.1, the causal coherence value should have a value of 1 since the frequency ω_2 in the feedforward subsystem resonated from the feedback subsystem, and

vice versa for the feedback subsystem. The aforesaid remains true even for the nonlinear Poincaré oscillator at the frequencies of ω_1 and ω_2 as the strength of the coupling is the dominant factor. However, all frequencies other than ω_1 and ω_2 will have low coherence values since the system is nonlinear.

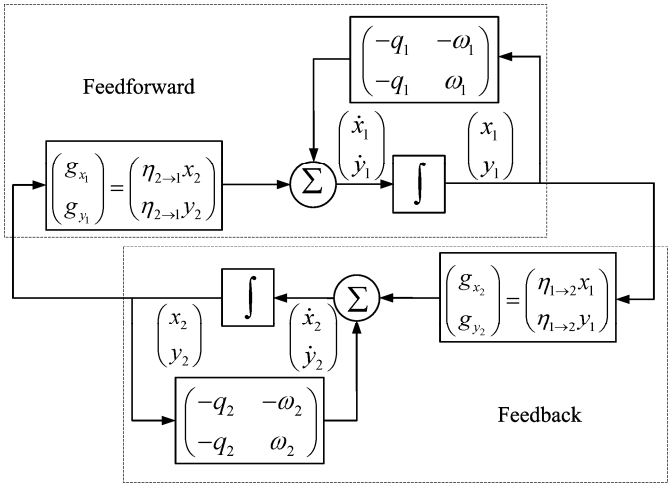


Fig. 3-2. Block diagram of the 2-subsystem Poincaré Oscillator.

3.3.3 Generation of time-varying surrogate data

Our approach to generating time-varying surrogate data is similar in concept to the short time Fourier transform (STFT) for nonstationary signals, as opposed to the power spectral density for time-invariant systems. Details regarding the implementation of this algorithm are described in Chapter 4. Thus, it will be just briefly summarized here. Our technique, similar to the STFT, is to segment data and compute surrogate time series for each of the segmented time series. Within each segment, the signal is assumed to be stationary. The length of the segment depends on the trade-off between time and frequency resolution as well as the validity of the stationarity assumption within the chosen segment length. For example, for highly time-varying systems, small segment lengths are necessary, but a consequence is decreased frequency resolution, and vice versa.

Based on the stationarity assumption within the chosen segment length, we can then use any of the many known time-invariant surrogate data techniques [39, 40], which can be applied to test the statistical significance of the time-invariant coherence function estimates [37]. We chose the iteratively refined surrogate data technique (IRSDT) [39]. The IRSDT will destroy any nonlinearity in the signal, and has been shown to be more accurate than the amplitude adjusted Fourier transform technique [40] because it iteratively corrects for deviations in the spectrum as well as maintaining the correct distribution of the signal.

The threshold value for time-varying surrogate data was based on the mean plus two standard deviations over the 20 TVCCF values obtained from the surrogate data set. Any time-varying coherence value (both causal and traditional) greater than the threshold value for each frequency represents 95% statistical confidence that it did not occur by some random occurrence. For further details and results of the time-variant surrogate data, the reader is referred to Chapter 4.

For the simulation examples, 512 data points of the causal systems (linear and nonlinear),

as described in the 3.3.1 and 3.3.2 were generated. For the first simulation example, linearly increasing and decreasing coupling strength were examined. Given this scenario, the TVCCF values should gradually increase from $y_1(n)$ to $y_2(n)$ and vice versa for $y_1(n)$ to $y_2(n)$. The time-varying spectra for the two subsystems $y_1(n)$ and $y_2(n)$ are shown in the left and right panels of Fig. 3-3(a), respectively. Both systems have time-varying dynamics at 0.2 and 0.5 Hz since the magnitudes of these frequencies wax and wane with time as expected due to increasing and decreasing coupling coefficients. The theoretical and estimated time-varying traditional and causal coherence function values are provided in Figs. 3-3(b) ~ (c), respectively. For both the estimated traditional coherence function and causal coherence function, we used the TVOPS algorithm based on two Legendre basis functions to determine the $c(n)$ parameters in Eq. (3-18), and the threshold value was set to 0.0001. The Walsh functions are appropriate for modeling signals with fast transients and burst-like dynamics, while Legendre polynomials are better suited for smoothly changing dynamics [12]. The initial model order was chosen as ARMA (6, 6) from which the TVOPS algorithm selected only a single most significant model term (not model order). The estimated traditional coherence function and causal coherence function values are identical to theoretical values. Note, however, that the traditional coherence function approach (left panels of Figs. 3-3(b) and (c)) is inappropriate, as the expected increasing or decreasing coherence values with time are entirely missing. Furthermore, the low coherence values in most frequencies except 0.2 Hz and 0.5 Hz, observed in the traditional coherence plot, can mistakenly be attributed to either a low signal-to-noise ratio or nonlinearity. Thus, this simple example illustrates an important point, that is, estimation of traditional coherence functions on a causal system is inappropriate and can lead to misinterpretation of the results.

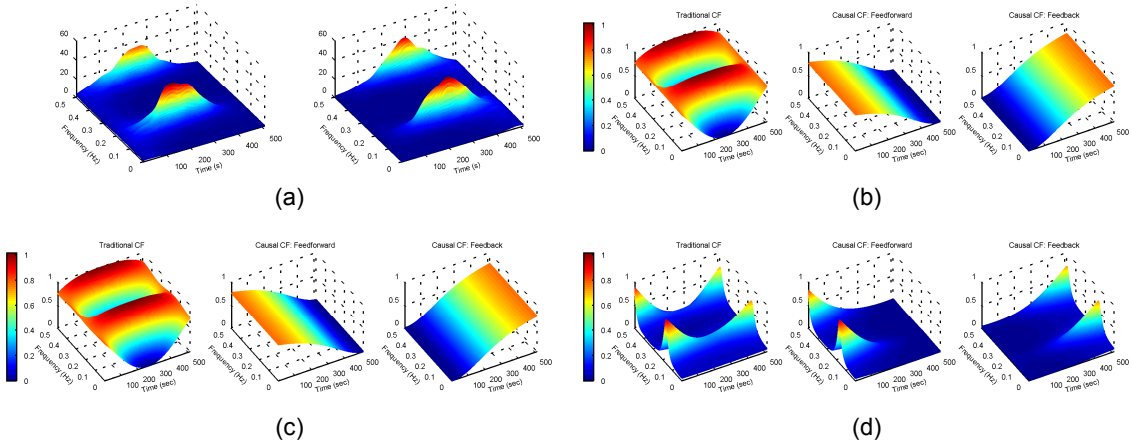


Fig. 3-3. TVCCF analysis of an ARMA system. (a) time-varying spectra of the two subsystems $y_1(n)$ and $y_2(n)$, (b) true TVCF values, (c) estimated TVCF values, and (d) estimated TVCF of the surrogate data. For all figures, the left panels represent the traditional coherence values, while the middle and right panels represent the feedforward and feedback causal coherence values, respectively.

Time-varying surrogate data results as detailed in the Methods section are provided in Fig. 3-3(d). Note the proportional waxing and waning of surrogate coherence values with either increasing or decreasing coupling strength values, especially at 0.2 Hz and 0.5 Hz. This is expected since the surrogate data technique used maintains the fidelity of the spectrum. The TVCCF values (2nd row, middle and right panels) at time points greater than 200 seconds are all considered to be significant as they are greater than the threshold values. Likewise, this is also true for the traditional time-varying coherence values at frequencies of 0.2 Hz and 0.5 Hz.

For the second simulation examples, 512 data points of the Poincaré oscillator, shown in

Eq.(3-19), were generated:

$$\begin{cases} \dot{x}_i = -x_i q_i - \omega_i y_i + g_{x_i} \\ \dot{y}_i = -y_i q_i + \omega_i x_i + g_{y_i} \\ q_i = \alpha_i \left(\sqrt{x_i^2 + y_i^2} - a_i \right) \end{cases} \tag{3-19}$$

where the subscript i denotes the index of subsystems; x_i and y_i are state variables of the ith subsystem; ω_i is the characteristic frequency of the ith subsystem; α_i and a_i are constants; and g_{x_i} and g_{y_i} are the coupling terms. For example, η in Eq. (3-20) quantifies the strength of coupling. The state variables x_1 and x_2 were used for data analysis. The sampling rate was 1 Hz. Note that the Poincaré oscillator is a nonlinear process and generates self-autonomous oscillations without a driving noise. A 10 dB additive Gaussian white noise was added after signals from Eq. (3-19) were generated and this noise should not be confused with the driving noise as denoted in Eq. (3-1).

$$\begin{cases} g_{x_1} = \eta_{2\to 1} x_2 \\ g_{y_1} = \eta_{2\to 1} y_2 \end{cases} \text{and} \begin{cases} g_{x_2} = \eta_{1\to 2} x_1 \\ g_{y_2} = \eta_{1\to 2} y_1 \end{cases} \tag{3-20}$$

For this simulation example, both α_i and a_i for the two subsystems were set to 1. Time-dependent coupling occurs between the two subsystems as shown in Fig. 3-4. The characteristic frequencies (ω_1 and ω_2) of the 1st and 2nd subsystems were set to $0.3 \times 2\pi$ Hz and $0.167 \times 2\pi$ Hz, respectively. As shown in Fig. 3-4, the coupling strength is time varying for both subsystems. As a result, high causal coherence values from the 1st subsystem to the 2nd should be observed in the time period from 1 to 256 seconds, followed by low coherence values from 257 to 512 seconds. The opposite scenario exists from the 2nd subsystem to the 1st, where low and high causal coherence values occur from 1 to 256 seconds and from 257 to 512 seconds, respectively. We used the TVOPS algorithm based on two Walsh basis functions for the entire 512 data point sample period to determine the $a(t,k)$ parameters in Eq. (3-1), and the threshold value was set to 0.0001. While the jump in coupling coefficients coincides with a transition point of Walsh functions in this example, we have previously shown that the performance of Walsh functions does not degrade even if their transition points do not match [41]. The initial model order was chosen as ARMA (6,6) from which the TVOPS algorithm selected only the eight most significant model terms (not model order) for both feedforward and feedback directions.

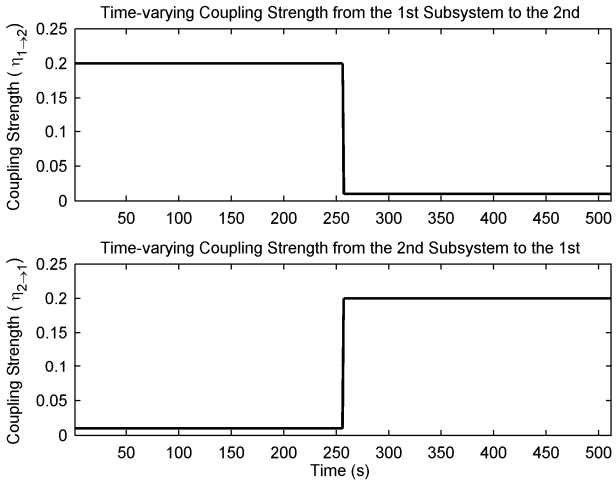
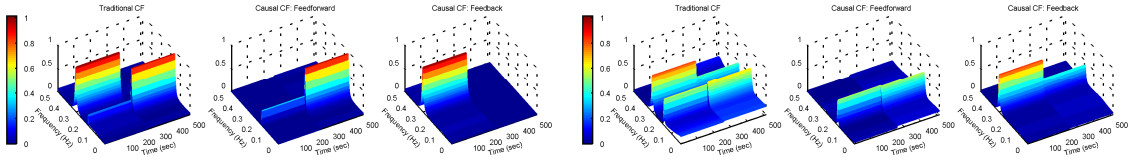


Fig. 3-4. Time-dependent coupling strength between the two subsystems. The upper panel is the coupling strength from the 1st subsystem to the 2nd; the lower panel is the time-dependent coupling strength from the 2nd subsystem to the 1st.

Based on the above specification of the Poincaré oscillator, the estimated traditional and the causal TVCF values are shown in Fig. 3-5(a). Fig. 3-5(b) shows the corresponding time-varying surrogate data. These time-varying causal coherence functions and the traditional coherence function were calculated using Eq. (3-3) ~ (3-4). As expected, unit coherence values at 0.3 Hz occur from the 1st subsystem to the 2nd subsystem only at the first half of the data segment (1 to 256 seconds). Similarly, unit coherence values at 0.167 Hz occur from the 2nd subsystem to the 1st from 257 to 512 seconds. In this example, the traditional TVCF values are essentially the composite of the two causal TVCF values. These estimated coherence values are all significant as they are greater than the threshold coherence values generated from time-varying surrogate data. Small coherence values observed during the first 256 points for the traditional and the feedforward path of the causal coherence functions are insignificant as they are less than the surrogate's threshold values.



(a) (b)

Fig. 3-5. (a): traditional TVCF (left panel); feedforward (middle panel) and feedback (right panel) causal TVCF; (b) surrogate coherence values corresponding to those shown in the panel (a).

To make a comparison to the time-varying causal coherence function, time-invariant traditional and causal coherence functions are calculated for the data generated using. Eq. (3-18) ~ (3-19). Results of the time-invariant traditional coherence function are shown in the left panel of Fig. 3-6, and the middle and right panels of Fig. 3-6 show causal coherence functions. Compared to the results of the top panels of Fig. 3-5, where the maximum coherence values are all unit values, these time-invariant traditional and causal coherence function values have smaller than unit coherence values. This is due to the averaging effect of time-invariant coherence methods. If the value of 0.5 were chosen as the threshold to determine the significance of coherence values, one would erroneously conclude that a significant causal coherence relationship only exists from the 1st subsystem to the 2nd, and not also from the 2nd to the 1st subsystem. The time-invariant surrogate threshold values are provided as dotted lines in Fig. 3-6.

Similar to the arbitrary threshold of 0.5, the surrogate threshold also indicates the insignificance of the coherence value at 0.167 Hz for both the traditional coherence function and the feedforward causal coherence function.

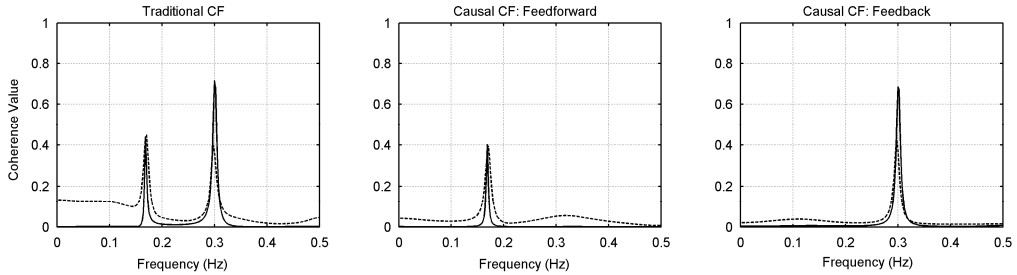
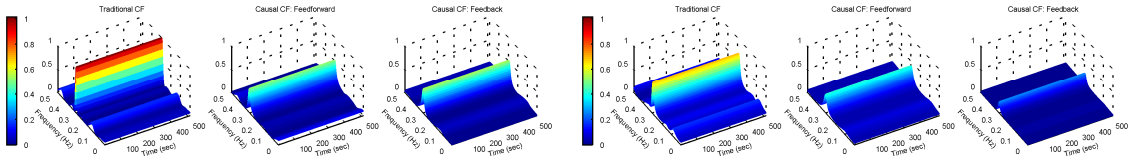


Fig. 3-6. Time-invariant traditional and causal coherence functions (solid lines). Left panel represents time-invariant traditional coherence function; middle panel and right panels represent time-invariant feedforward and feedback causal coherence values. Dotted lines represent time-invariant surrogate coherence values.

The last simulation is also based on the Poincaré oscillator as described by. Eq. (3-18) ~ (3-19). However, instead of the two distinct characteristic frequencies as provided in the first example, the two subsystems' characteristic frequencies are both set to 0.3 Hz. The parameters a and α for both subsystems was chosen to be 1 and 0.5, respectively. Coupling occurs for both directions with strength of 0.1, for all time points. Note that this coupling strength of 0.1 is less than the previous example where we set the value to be 0.2. This decrease in coupling strength should result in lower than unit coherence values. The results are shown in Fig. 3-7. Parameters related to the calculation of the TVOPS were set the same as in the previous example. The traditional TVCF values (top left panel) show unit values for all times at the frequency 0.3 Hz. However, the two causal TVCF estimates show relatively weak values for all times (top middle and right panels). This is in agreement with the condition described in (A12), and also due to the fact that we used the smaller coupling strength value of 0.1. The bottom panels of Fig. 3-7 show time-varying surrogate threshold values corresponding to those in the top panels of Fig. 3-7. These surrogate threshold values are lower than those shown in the top panels; therefore, the top panel values are all significant.



(a) (b)

Fig. 3-7. (a) Traditional TVCF (left panel), feedforward (middle panel) and feedback (right panel) causal TVCF values; (b) surrogate data TVCF values corresponding to those shown in the top panels.

3.4 Application to Renal Blood Pressure and Blood Flow Data

3.4.1. Animal model

Renal arterial blood pressure (ABP) and volumetric blood flow (BF) were recorded from 5 normotensive Wistar rats weighing 210-300 g. The surgical preparation and BP forcing

procedures were as described previously [42]. Data (BP and BF) were filtered at 20 Hz and digitized at 100 Hz. The signals were processed with a digital low-pass filter to exclude any frequency components beyond 0.5 Hz, followed by a trend removal procedure to remove the mean and linear trends. The signals were down-sampled to 1 Hz and normalized to unit variance. Data were acquired under three experimental conditions: during control with autoregulation intact; during intravenous infusion of the diuretic furosemide (Lasix, 10 mg/kg/min) to inhibit TGF [43]; and during continued Lasix infusion after injection of L-nitro arginine methyl ester (L-NAME, 10 mg/kg) to inhibit synthesis of nitric oxide and augment myogenic activity [42].

3.4.2. Results

The frequency responses of the two renal autoregulatory mechanisms are 0.1-0.3 Hz for the myogenic and 0.02-0.05 Hz for TGF [31-34]. Recently, it has been suggested that a third mechanism operating below 0.02 Hz is also responsible for autoregulation, although the contribution of the third mechanism to the total renal autoregulation capacity is minimal [44]. Fig. 3-8(a) shows typical renal ABP (top panel) and BF (bottom panel) recordings without any pharmacological intervention.

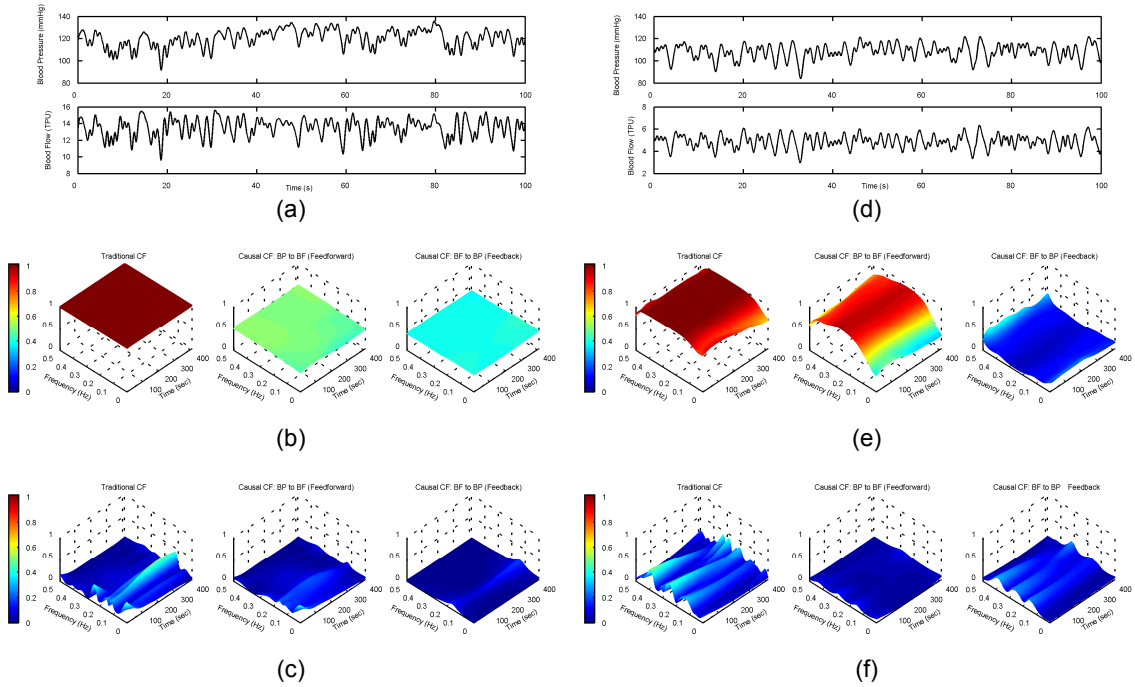


Fig. 3-8. (a) Representative time series of BP (top) and BF (bottom) during control; (b) traditional TVCF (left), and causal TVCF during control; (c) surrogate data TVCF values corresponding to those in row (b); (d) representative time series of BP (top) and BF (bottom) after NO blockade and Lasix infusion; (e) traditional TVCF and causal TVCF after NO blockade and Lasix infusion; (f) surrogate data TVCF values corresponding to those in row (e). For all panels, the left panels represent the traditional coherence values, while the middle and right panels represent the feedforward and feedback causal coherence values, respectively.

For the analysis involving both the TVCCF and traditional TVCF methods, 5 Legendre functions were used. The initial model order was set to ARMA (8,8) and the threshold value was set to 0.00001; the TVOPS selected 3 significant model terms from the selected model order for both feedforward and feedback paths. The traditional TVCF values (left panel of Fig. 3-8(b)) are essentially close to unity for all frequencies. For the two causal TVCF values, however, far lower values are observed, as shown in the middle and right panels of Fig. 3-8(b). Surrogate threshold

value counterparts to Fig. 3-8(b) are shown in Fig. 3-8(c). These threshold values are all lower than those shown in Fig. 3-8(b), suggesting the statistical significance of the experimentally-obtained coherence values for both traditional and causal coherence functions. Representative ABP and BF tracings, the resultant CF, and surrogate data CF estimates based on the application of pharmacological blockades (NO and Lasix) are shown in Figs. 3-8(d) ~ (f), respectively. Similar to the control case, the traditional CF values are near unity for all frequencies, as shown in the left panel of Fig. 3-8 (e). The middle and right panels of Fig. 3-8 (e) show feedforward and feedback causal TVCF values, respectively. Note that ABP is the input and BF is the output, and vice versa for the feedback system. Note the relatively low TVCF values in the frequency range (0.02 ~ 0.5Hz) associated with TGF in these two causal TVCF estimates, and high coherence values only in the feedforward (middle panel) TVCF estimate. In addition, these values are all significantly higher than the surrogates' threshold values. This result suggests that the feedforward and feedback paths of a causal system can be represented by the myogenic and TGF mechanisms, respectively. The use of the traditional TVCF is inappropriate for causal systems since we obtain similar high coherence values for all frequencies for two conditions that are completely different. Slow deviations of these coherence values over time reflect that the system is slowly time-varying and that our method is especially appropriate for such systems. Because the data presented are slowly time-varying, the use of the time-invariant causal coherence functions is likely to provide similar results as does the TVCCF.

The group average results based on five animals are shown in Table 3-1. We observe statistical differences ($p < 0.05$) between the baseline and pharmacological perturbation conditions for both traditional and causal TVCF values for frequencies associated with the myogenic and TGF mechanisms. Interestingly, while both the traditional and causal TVCF values decrease from the baseline to pharmacological perturbation for frequencies associated with TGF, the myogenic-associated traditional and causal coherence values significantly decrease and increase, respectively. We believe the increase in causal TVCF values associated with the myogenic mechanism with the administration of Lasix after NO blockade is the expected event, since Lasix is essentially a TGF blocker, and the effect of NO blockade is to enhance the reactivity of both mechanisms. Therefore, the present results imply that the feedforward and feedback paths are primarily dominated by the myogenic and TGF mechanisms, respectively.

Table 3-1 TV Coherence Group Average. *Significant difference with p < 0.05; Student-Newman Keuls.

GROUP (N = 5)	TRADITIONAL COHERENCE		CAUSAL COHERENCE	
	TGF	MYO	TGF (Feedback)	MYO (Feedforward)
Baseline	1.00±0.01	1.00±0.01	0.49±0.01	0.50±0.02
Lasix after NO Blockade	**0.80±0.10***	**0.98±0.01***	**0.22±0.10***	**0.87±0.09***

3.5 Summary

The coherence function provides correlation information between two signals in the frequency domain. A traditional nonparametric approach to calculating the coherence function is based on the calculation of the ratio between the cross-spectrum and the auto-spectrum between two signals. Calculation of auto-spectra and cross-spectra involves segmentation of the signals, which results in a tradeoff between time and frequency resolutions, and also in variance of the spectral estimates. For example, more segmentation produces less variance of the spectral estimates but this action results in less frequency resolution. This aforementioned tradeoff problem becomes compounded for time-varying coherence functions. Furthermore, one of the

major limitations of the calculation of coherence functions based on power spectra, either time-varying or time-invariant, is the inability to calculate causal coherence functions.

One way to calculate both traditional and causal coherence functions is to use a parametric approach. For example, a vector AR-based method has been developed to calculate both traditional and causal time-invariant causal coherence function [37]. This method assumes a causal model in which system identification procedures are used to estimate the model coefficients, and subsequently the coherence function values. It was unclear whether the parametric approach also leads to coherence values that are bounded between 0 and 1. Our theoretical derivations of both time-invariant and time-varying coherence functions show that indeed coherence values are bounded between 0 and 1 for both traditional and causal coherence functions.

In our work, we extend a time-invariant vector AR model to calculate both traditional and causal TVCF. Biological systems are time-varying, and on a time scale of seconds to minutes, these systems may undergo slow and subtle transitions from one dynamic to another. Thus, the time-invariant CF may not be able to delineate subtle changes in CF values because it essentially provides time averaged CF values which can mask subtle changes. The ability to track subtle changes in CF values may have an important prognostic value, as they can be correlated to the progression of diseases. It should be noted that the method developed is most applicable to causal systems, and should be applied if there is a priori knowledge that the system under investigation is a causal system. Otherwise, the causal TVCF values may provide erroneous results.

As demonstrated with renal data in this chapter, the proper use of the causal coherence functions can reveal plausible functional structures of systems, as postulated for the renal autoregulatory mechanisms. The implication of feedforward and feedback paths being represented by the myogenic and TGF mechanisms, respectively, is currently under investigation in our laboratory using many more diverse datasets, as well as a block-structured model incorporating these feedforward and feedback paths of a causal system.

Chapter 4 Development of Time-varying Surrogate Data Method

4.1 Introduction

Theiler developed a time-invariant surrogate data technique for statistical evaluation of the presence of nonlinear dynamics in time series [40] . Improved surrogate data techniques have followed thereafter [45] and have found applications in many different disciplines [45, 46]. In addition to the nonparametric method based on Fourier transform that Theiler had developed, parametric method based on autoregressive model was also developed as an alternative way to generate surrogate data [39, 47].

While these time-invariant surrogate data techniques were originally developed to determine the presence of nonlinearity, they have also been applied to evaluating the statistical significance of linear time-invariant coherence functions [48, 49]. With recent advances in the development of time-varying coherence techniques [36, 50], the need for a time-varying surrogate data (TVSD) technique is apparent. Surrogate data techniques are designed to destroy any coupling present in the signal, and because they are designed to generate multiple realizations of the non-coupled data, the statistical significance of the coherence can be evaluated. Without surrogate data, quantification of the strength of coherence is arbitrary as any values higher than 0.5 are considered to be an indication of highly coherent signals. Thus, this arbitrary demarcation is most appropriate for highly coherent signals and incorrectly ignores any coherent values that are less than 0.5. Time-invariant surrogate data techniques remove this bias toward only the highly coherent signals, and were found to be sensitive even for weakly coupled signals [51]. However, they are inappropriate for time-varying data as they provide time averaged statistical significance values. Thus, in a case where coherence values wax and wane with time, and waning of the coherence values is more prevalent, the time-invariant surrogate data technique is most likely to result in the incorrect interpretation that there is no coherence for all time points. However, with the TVSD technique, the statistical significance of the time-dependent changes in the observed coherence values at a particular frequency can be evaluated. It should be noted that the TVSD method is also applicable to time-varying bispectrum analysis as it can be used to determine the statistical significance of the nonlinearly coherence values.

Since the traditional time-invariant surrogate techniques have been successfully applied to the detection of nonlinearity and determination of threshold for coherence functions, it is intuitive to extend both the parametric and nonparametric methods to time-varying cases to generate threshold that is applicable to time-varying systems to determine the significance of nonlinearity and coherence.

In this chapter, we have extended the parametric and nonparametric methods to generate TVSD. The extended methods have been tested both on simulation examples and physiological data for nonlinearity detection and causal coherence analysis.

4.2 Method to Generate Time-varying Surrogate Data

4.2.1 Nonparametric method

The nonparametric approach to generating time-varying surrogate data is more similar in

concept to the Short-Time Fourier Transform (STFT) for nonstationary signals than it is to the power spectral density for time-invariant systems. Our technique, similar to the STFT, is to segment the data and compute surrogate time series for each of the segmented time series. Within each segment, the signal is assumed to be stationary. The length of the segment depends on the trade-off between time and frequency resolution as well as the validity of the stationarity assumption within the chosen segment. For example, for highly time-varying systems, small segments are necessary, but the consequence is decreased frequency resolution, and vice versa.

Based on the stationarity assumption within the chosen segment length, we can then use any of the many known time-invariant surrogate data techniques. We chose the iteratively refined surrogate data technique (IRSDT) [51]. The IRSDT will destroy any nonlinearity in the signal, and has been shown to be more accurate than the amplitude adjusted Fourier transform technique because it iteratively corrects for deviations in the spectrum as well as maintains the correct distribution of the signal in time domain [51].

4.2.2 Parametric: AR model based

The first step involves fitting an AR model to the time series. The model coefficients can be estimated by many published system identification methods such as recursive lease square (RLS) or optimum parameters search (OPS) [16] and its time-varying version (TVOPS). The spectrum of the time series is represented by the estimated model coefficients as AR model assumes the prediction error to be white. The second step involves generation of surrogate data realizations by regressing the AR coefficients with white noise. This procedure yields a nearly identical spectrum to that of the original time series but is uncorrelated with the original time series. The assumption of the parametric method is that the residue error of the AR model is stationary. That is, it has constant variance over time. When the parametric method is employed to investigate the coherence values between too signals, two separate AR models will be used to the signals the time-frequency spectrum estimation. Two independent white noises will be fed to the AR models to generate the surrogate pair.

4.3 Applying TVSD to Simulation Examples and Physiology Data

4.3.1 Determine the threshold for nonlinearity detection

To quantify the nonlinearity in a system, we employed the prediction error difference (PED) as the statistic index of nonlinearity. In order to calculate PED, the signal under investigation is modeled by a separate linear and nonlinear AR model, respectively. TVOPS algorithm was employed to estimate the time-varying model coefficients. For the detail of TVOPS algorithm, the reader is referred to Appendix I. The number of selected model terms was kept the same for linear and nonlinear AR models since more model terms commonly tend to yield lower prediction error. By keeping this consistence, the introduction of bias in PED values due to over- or under-fitting is avoided. The PED is defined as how much improvement can be achieved by fitting the data with nonlinear AR instead of linear AR:

$$PED = \frac{v_L - v_N}{v_y} \times 100\% \tag{4-1}$$

For time-varying case, v_L, v_N and v_y are the instantaneous variance (IV) series of linear AR prediction error, nonlinear AR prediction error and the signal. Thus, the PED is a function of time. The IV is calculated by using a windowing technique based on the same philosophy of STFT. To calculate the IV, a moving window is centered at each point of the prediction error series. The prediction error within this window is assumed stationary. The IV value at each temporal point is estimated as the variance of the prediction error within that window. Suppose the window width is $2M+1$, the IV value at time n is calculated by:

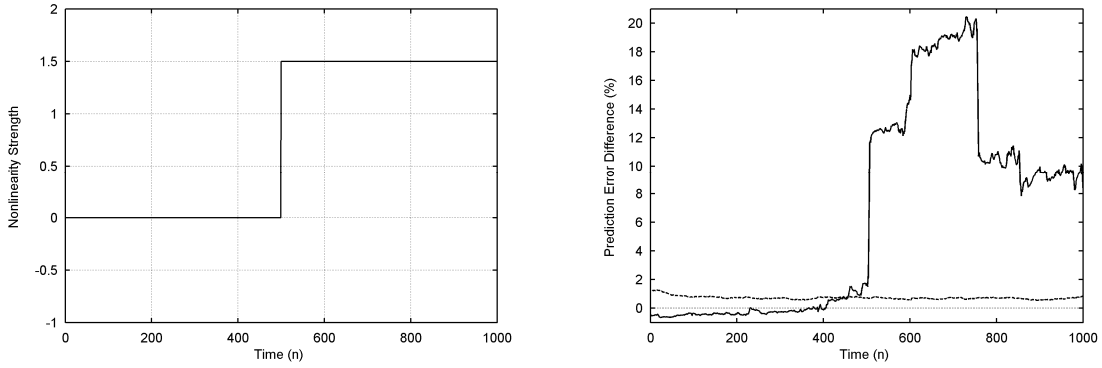
$$v_I(n) = \mathrm{var}\left(e(n-M), e(n-M+1), \ldots, e(n), e(n+1), \ldots, e(n+M)\right) \tag{4-2}$$

The first simulation demonstrates the efficacy of TVSD generated by nonparametric method for the detection of time-varying nonlinearity.

Consider the following simulation example:

$$y(n) = 0.6y(n-1) - 0.6y(n-2) - c(n)\left[y(n-1)y(n-3) + y^2(n-2)\right] + e(n) \tag{4-3}$$

The model is driven by noise $e(n)$ with the variance of 0.01. The coefficient $c(n)$ has the on-off feature as shown in Fig. 4-1(a), introducing time-varying nonlinearity into signal y. The signal was modeled by linear and nonlinear time-varying AR model respectively. Two Walsh functions were used to capture the on-off feature of coefficient $c(n)$. Among the 10 initial model terms, TVOPS picked out 4 significant model terms. For the calculation of the IV series, we chose the window width one quarter of the series length, which is 250 points (the length of the signal is 1000). The threshold was obtained based on 100 realizations of surrogate data. The threshold was chosen as the critical value representing 95% confident level. As shown in Fig. 4-1(b), the threshold shows that the nonlinearity is insignificant from time 0 ~ 450 since the PED of the signal is lower than the threshold. The nonlinearity became significant afterwards because the PED values of the signal are much higher than the threshold. Compared to Fig. 4-1(a), which shows the true onset of nonlinearity is at time 500, the TVSD detected the occurrence of nonlinearity at time 450. Notice that the window width is 250 and the window is centered at each temporal point, so at time 450, the window covers points from time 326 to 575. So in this simulation example, the nonparametric TVSD only needs 75 ($575-500=75$) points to detection the nonlinearity, which accounts for 30% ($75/250\times100\%=30\%$) of the window length.



(a) (b)

Fig. 4-1. Nonparametric and parametric thresholds to detect time-varying nonlinearity: (a) time-varying nonlinearity strength. Nonlinearity only presents in the second half of the signal; and (b) surrogate thresholds to detect significance of nonlinearity: dashed line is the threshold obtained by nonparametric surrogate; dotted line is the threshold obtained by parametric surrogate; and solid line is the prediction difference of the original signal.

The nonparametric TVSD was also applied to heart rate time series to detect nonlinearity due to posture change. The ECG signal was recorded in supine and upright position from a healthy subject. An R-peak detection algorithm was then applied to the ECG signal to extract heart rate time series. The extracted heart rate is shown in the top panel of Fig. 4-2. According to cardiovascular physiology, in supine position, parasympathetic auto nerves are more active and dominant. Since parasympathetic possesses nonlinearity characteristic, hear rate series should exhibit more nonlinearity in supine position. While in upright position, sympathetic gets elevated. As a consequence, heart rate data becomes more linear and the nonlinearity may vanish due to the posture change. In the heart rate series shown in the top panel of Fig. 4-2, the first half of the data (from time 1~512) was recorded in supine position, followed by the heart rate data in the upright position (from time 513 ~ 1024). In modeling the heart rate data by linear and nonlinear AR model, 4 model terms were chosen by TVOPS for both models. 2 Walsh basis functions were used to capture the posture change. The PED series of heart rate data is shown in the bottom panel of Fig. 4-2 as black line. The threshold obtained by the nonparametric TVSD is shown as red line in Fig. 4-2. It is clearly shown that in the supine position (time 1~512), significant nonlinearity persistently presents since the PED values of the heart rate are greater than the threshold thoroughly. After time 527, the PED values of heart rate dropped abruptly and stayed below the threshold, which indicates the hear rate series is highly linearly after time 527. This is a consistent agreement with the fact hear rate is more linear in upright position due to the reduced parasympathetic and elevated sympathetic activities.

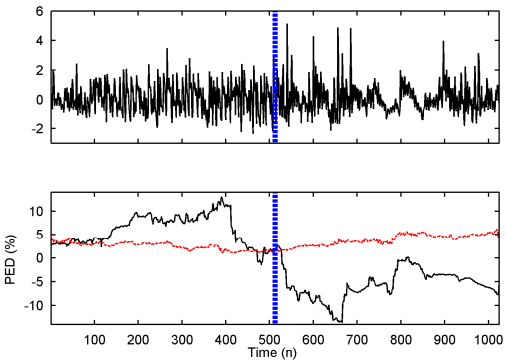


Fig. 4-2. Detect time-varying nonlinearity in heart rate data using time-varying surrogate data. Top panel: the heart rate time series, and bottom panel: the PED time series of the heart rate data (black line); and the threshold obtained by nonparametric method (red line).

In the case of nonlinearity detection, it may be inappropriate to use parametric method to generate TVSD because parametric TVSD uses a linear AR model to estimate the time-frequency spectrum of the original signal. However, some nonlinearity can not be modeled by linear AR no matter how many model terms are chosen. For example, Henon map is a nonlinear system and it can not be modeled by linear AR. The parametric TVSD can be used to determine the threshold of coherence functions, as shown in the proceeding section.

4.3.2 Determine the threshold for time-varying coherence functions

Traditional surrogate data has been used to determine the significant threshold of coherence analysis [49]. However, all these applications are based on time invariant assumption. With the extension of coherence functions to time-varying case [36, 50], corresponding time-varying surrogate data technique is demanded to determine the threshold for nonstationary signals. In this simulation, we will demonstrate how the developed time-varying surrogate data can be efficiently used in determining the significant level of causal coherence functions in a closed-loop time-varying system. For the detail of TVCCF analysis, the reader is referred to Chapter 3.

The closed-loop system structure is illustrated in Fig. 4-3. The system is driven by two independent white noise $e_1(n)$ and $e_2(n)$. The $A_x\left(z^{-1}\right)$ and $A_y\left(z^{-1}\right)$ are the autoregressive

polynomial of signal x and y, respectively, where z^{-1} is the one step time delay operator. $C_{xy}\left(z^{-1}\right)$ and $C_{yx}\left(z^{-1}\right)$ are the coupling terms from x to y and from y to x, respectively. Given this notation, the transfer functions from x to y and y to x can be expressed as:

$$\begin{cases} H_{xy}\left(z^{-1}\right)=\dfrac{C_{xy}\left(z^{-1}\right)}{1-A_{y}\left(z^{-1}\right)} \\ H_{yx}\left(z^{-1}\right)=\dfrac{C_{yx}\left(z^{-1}\right)}{1-A_{x}\left(z^{-1}\right)} \end{cases} \tag{4-4}$$

To simulate the time-varying features, the coefficients of the polynomials in Eq. (4-4) were designed as functions of time. In this simulation example, 512 seconds of x and y were recorded. Within this period, the system experienced a transition at time 256 seconds. From 0~255 seconds, the system polynomials were chosen as:

$$\begin{cases} A_{x}\left(z^{-1}\right)=2.422z^{-1}-3.250z^{-2}+2.415z^{-3}-1.057z^{-4}+0.200z^{-5} \\ A_{y}\left(z^{-1}\right)=-2.422z^{-1}+3.250z^{-2}-2.415z^{-3}+1.057z^{-4}-0.200z^{-5} \\ C_{xy}\left(z^{-1}\right)=0.0637z^{-2}-0.0016z^{-3}+0.0724z^{-4}+0.0724z^{-5}-0.0016z^{-6}+0.0637z^{-7} \\ C_{yx}\left(z^{-1}\right)=-0.0637z^{-1}+0.0016z^{-2}-0.0724z^{-3}-0.0724z^{-4}+0.0016z^{-5}-0.0637z^{-6} \end{cases} \tag{4-5}$$

With this set of parameters, the transfer function from x to y, as defined in Eq. (4-4), is a low pass filter with the cut off frequency of 0.15 Hz; while the transfer function from y to x is a high-pass filter and the cutoff frequency is 0.35 Hz. In the duration from 256 ~ 512 seconds, all the coefficients in Eq.(4-5) reversed their signs producing a system with high-pass filtering characteristic from x to y and low-pass filtering from y to x. With this varying characteristic both in time and frequency domains, the time-varying causal coherence function (TVCCF) from x to y should have high values only in low frequency band (0~0.15 Hz) from time 0~255 seconds; and high values only in high frequency band (0.35 ~ 0.5 Hz) from time 256 ~ 512 seconds. The TVCCF from y to x should have the opposite pattern.

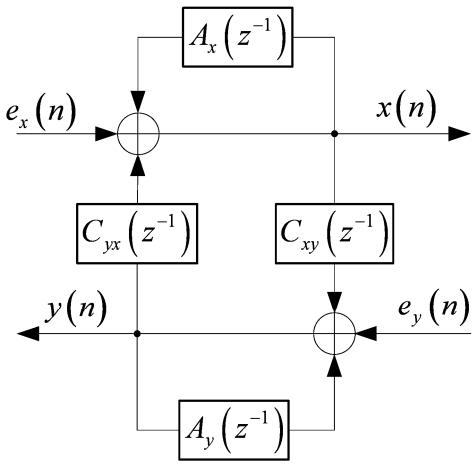


Fig. 4-3. The structure diagram of a closed-loop system.

Results of this simulation are shown in Fig. 4-4. The left panel of Fig. 4-4(a) is the TVCCF from signal x to y. As designed, the transfer function from x to y in the duration 0~255 seconds is a low-pass filter with cutoff frequency 0.15 Hz. Thus, the causal coherence function from x to y within this time frame should only have high values only in the low frequency band of 0~0.15 Hz. This can be clearly seen from the left panel of Fig. 4-4(a), where high coherence values (around 0.75) are observed in the time frame 0 ~ 255 seconds and frequency band 0 ~ 0.15 Hz only. To test if the high

values are significant or not, the TVCCF is compared against the thresholds obtained by nonparametric and parametric surrogates, shown in the up-right panel in Fig. 4-4(a). To make the comparison more clarified, we only show a single representative of the TVCCF and corresponding thresholds for either part of the signal (the first part: 0 ~ 255 seconds; and second part: 256 ~ 512 seconds) since the values of either part does not change within that particular time duration. Notice that the TVCCF values are higher than both the parametric and nonparametric thresholds in the frequency 0 ~ 0.15 Hz and lower than the thresholds otherwise. This indicates the high coherence values within this frequency range are significant and signal x is causally correlated with signal y only within this frequency range. This is a consistent agreement with the design of the simulation. After time 255 seconds, the transfer function from x to y switched to a high-pass filter with the cutoff frequency of 0.35 Hz. Again, the TVCCF is compared against the thresholds and the comparison is shown in the bottom-right panel in Fig. 4-4(a). Both of the two thresholds are below the TVCCF values in the high frequency band (0.35~0.5 Hz) and above the TVCCF values otherwise. This once again validated that the TVCCF values were significant high only in the high frequency, as expected. Shown in Fig. 4-4(b) is the TVCCF from y to x. Similar results can be seen from the plot and the explanation is analogous. The difference is that the TVCCF is high in high frequency and low in low frequency from time 0 ~ 255 seconds; while it is high in low frequency and low in high frequency from time 256 ~ 512 seconds.

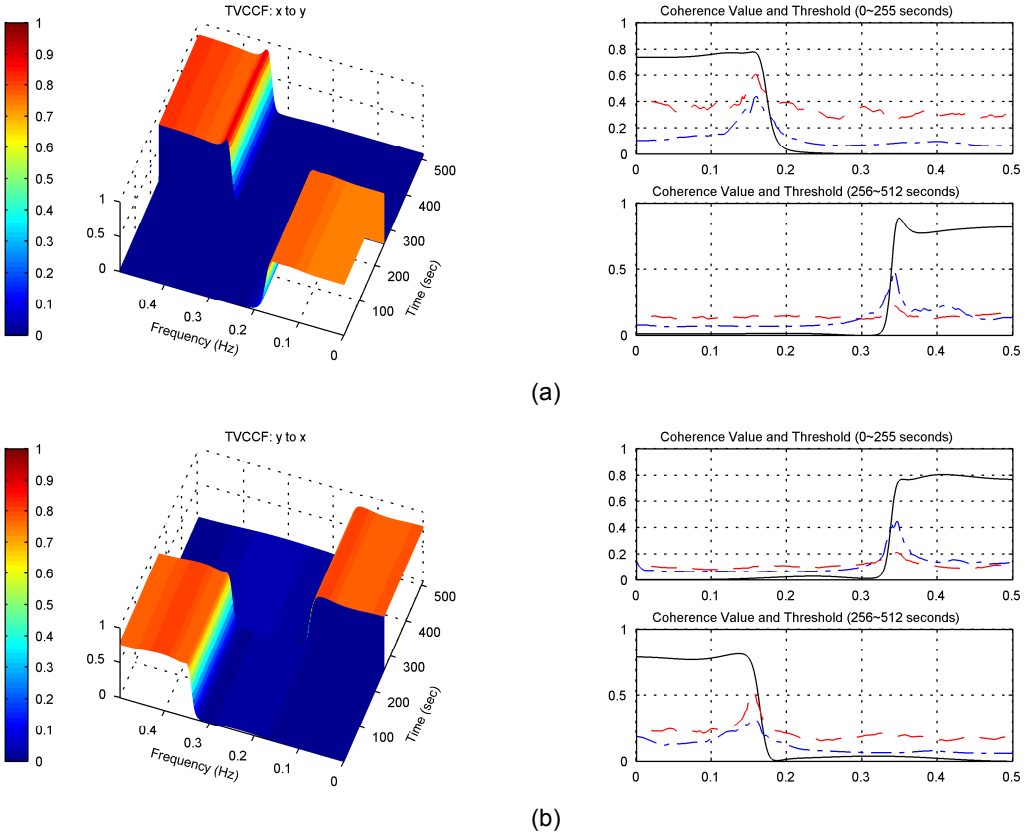


Fig. 4-4. Estimated causal coherence functions: (a) estimated TVCCFs of the simulation data, and (b) estimated statistical threshold coherence values based on the STFT surrogate data.

The TVSD technique was also tested on previously collected blood pressure (BP) and heart rate (HR) data [52]. Details regarding data collection and data preprocessing procedures are described in our previous study [52]. Data plotted in Fig. 4-5(a) represent BP and HR data during

the control state followed by the application of atropine, which blocks the parasympathetic nervous activities. The application of atropine occurs at 256 seconds in all panels of Fig. 4-5. The left and right panels of Fig. 4-5(b) show TVCCF representing BP to HR and HR to BP, respectively. The notation "BP to HR" and similar shall mean the first variable (BP) is the input signal of the system and the second variable (HR) is the output signal. In the left panel, representing the coherence relationship from BP to HR, during the control state (time less than 256 seconds), we note statistically high coherence values especially at the low frequency (LF: 0.04 to 0.15 Hz) and high frequency (HF: 0.2 to 0.4 Hz) bands. The LF is known to contain dynamics pertaining to both sympathetic and parasympathetic nervous activities whereas the HF band is attributed to the dynamics of the parasympathetic nervous system [53]. With the application of atropine (> 256 seconds), we note insignificant coherence at all frequencies as confirmed by the TVSD coherence values shown in the left panels of Fig. 4-5(c) and (d). This is the expected result, since the parasympathetic activities which reside in both LF and HF bands have been blocked with atropine. The right panel of Fig. 4-5(b) shows the relationship from HR to BP, which is the baroreceptor activity. With application of atropine, the expected result is an increase in HR which in turn activates the baroreceptors to decrease the HR. The baroreceptors excite parasympathetic activity to decrease HR, but since the parasympathetic nervous system is blocked, the only recourse is to decrease sympathetic nervous activity. Thus, this increased baroreceptor activity required to lower sympathetic activity should only be reflected in the LF region. This is exactly what we observe in the LF region of the right panel in Fig. 4-5(b); the coherence values in the LF band are greater as compared to the control state and are higher than the TVSD coherence values, as shown in the right panel of Fig. 4-5(c) and (d).

As expected, we do not see any changes in the coherence values in the HF region from the control state to the application of atropine in the right panel of Fig. 4-5(b). Insignificant coherence values in the HF band in these two states are confirmed by the TVSD coherence values shown in the right panels of Fig. 4-5(c) and (d). Plots in Fig. 4-5(e) represent time-invariant SD results. Note that with this approach, the expected insignificant coherence in the HF region is also confirmed since time-invariant SD coherence and the TVSD coherence values are similar. This example illustrates the additional insight as well as the correct physiological interpretation that can be obtained with TVSD as well as time-varying coherence function estimates.

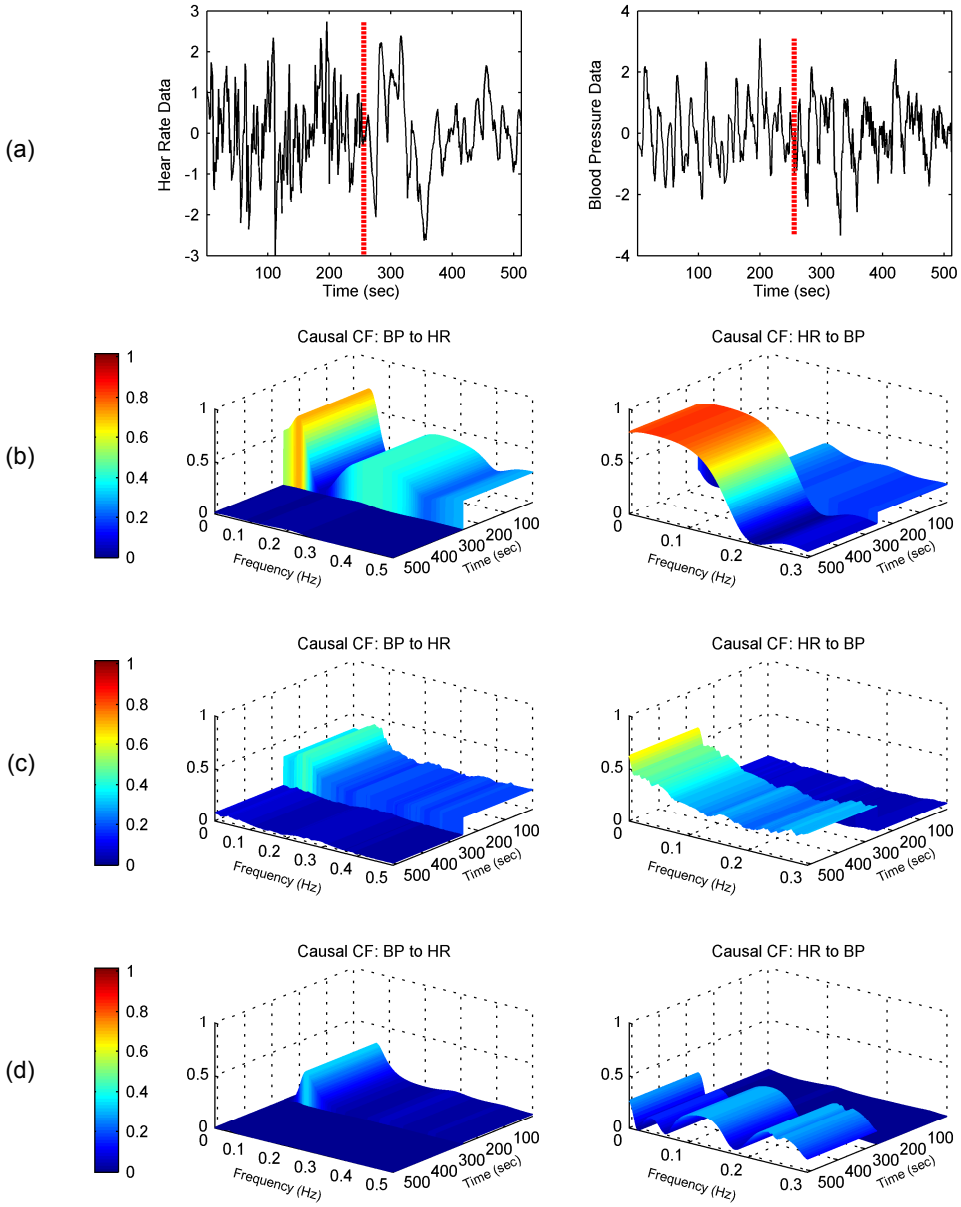


Fig. 4-5. Application of the proposed method to BP and HR data. (a) the heart rate data (left panel) and blood pressure data (right panel) (b) causal coherence functions from BP to HR (left), and from HR to BP (right); (c) threshold values based on 100 realizations of the STFT surrogate data; and (d) threshold of parametric TVSD.

4.4 Summary

With recent new development in time-varying coherence analysis [36, 50], as well as the development of other time-varying data analysis method such as time-varying nonlinearity detection, time-varying surrogate data techniques to properly take into account the statistical significance of TVCF values and time-varying nonlinearity presence are certainly needed. Towards this goal, the nonparametric and parametric TVSD techniques have been extended time-

varying case. The nonparametric method is based on STFT with the assumption that the signal is stationary within each segment. Thus the time-invariant surrogate data can be generated each segment. Stitching each piece of the time-invariant surrogate data will give an approximation of the TVSD. The parametric method uses a time-varying AR model to fit the signal, and the surrogate is generated by feeding the estimated AR model with an independent white noise. The parametric method has the assumption that the time-frequency spectrum of the signal can be represented by the estimate time-varying AR coefficients.

Simulation results have demonstrated the efficacy and applicability of the proposed parametric and nonparametric method to nonlinearity detection and evaluation of significance of coherence functions. For the nonlinearity detection, the simulation showed that the nonparametric TVSD correctly detected the onset of significant nonlinearity. Appling the TVSD to HR data revealed that nonlinearity significantly presented in supine position while the nonlinearity vanished due the posture change from supine to upright. This is a consistent finding with other literatures since the nonlinear Autonomic Nervous System (ANS) activity, parasympathetic, is more active in supine position; while the linear ANS activity, sympathetic, is more dominant in upright position [54-57].

In the TVCCF analysis, both parametric and nonparametric TVSD provided thresholds to correctly determine the significance of causal correlation, as confirmed by the simulation. Application in the HR and BP data before and after atropine administration showed that in control state, the causal coherence function from BP to HR has high values both in low-frequency and high-frequency. This is due to the fact that parasympathetic presents in both LF and HF. After the atropine injection, which blocks the parasympathetic, this causal correlation is eliminated and the causal coherence values from BP to HR are insignificant in both frequency bands. For the causal coherence function from HR to BP, which reflects the baroreceptor activities, the high values are observed in low frequency band after atropine injection. This is an agreement with the fact that parasympathetic is blocked, so the correlation relationship from HR to BP is elevated in sympathetic frequency band.

PART 2 INSTRUMENTATION

Chapter 5 Wireless Multiple Subject-Parameter Monitoring [1]

5.1 Introduction

Recent advances in wireless technology have brought us closer to fully integrated wireless vital parameters monitoring systems becoming a reality. For example, Becker et al. demonstrated wireless transmission of a few vital physiological parameters using a Bluetooth protocol for a Personal Digital Assistant (PDA) device [8]. In other works, a cell phone has been used to transmit biomedical signals [1, 9, 10]. More recently, Rasid and Woodward have developed a Bluetooth telemedicine processor for multichannel biomedical signal transmission via mobile cellular networks [10]. Specifically, their system utilizes the cellular protocol known as general packet radio service (GPRS), which has much higher data transmission rates than the global system (GS protocol) for mobile communications. In another study, Hung et al. utilized a wireless application protocol for telemetry application of biomedical signals [9]. However, because this system uses an analog wireless transmission module, the data are more sensitive to noise contamination. The common feature among all of the aforementioned systems is that they are all limited to data collection from a single or at most a few subjects.

To overcome this limitation, we are developing a system which we term wireless multiple subject-parameter monitoring (WMSPM). This system is based on a low cost commercially-available wireless transmission mote known as *Tmote Sky*. It also features ultra-low power consumption, a relatively long range of radio transmission (50 m indoor and 125 m outdoor) [58], and great mobility due to its small size. Another salient feature of this system is that a single mote receiver supports data collection from multiple patients of multiple physiological parameters, making it suitable for scenarios as diverse as battlefields, emergency rooms, and nursing homes.

5.2 System Design

5.2.1 System overview

The WMSPM system is constructed based on the Wireless Personal Area Network (WPAN) protocol as specified by the IEEE 802.15.4 standard [59]. The WMSPM system consists of three parts: transmitter and receiver, which are both comprised of motes, and a display terminal, as shown in Fig. 5-1. A transmitter mote includes a wireless transmission circuit (*Tmote Sky*) which can be connected to analog or digital based physiologic parameter acquisition modules

1 Relevant work was published on the *Proceedings of IEEE-EMBS 2006 Conference*, Vol. 1, pp. 5896-5899. September 1-4, 2006, New York

such as an ECG and pulse oximeter sensor. A receiver mote is also based on the *Tmote Sky* wireless circuit, but it is programmed to receive data from many individual transmitter motes. Furthermore, a receiver mote's function is to transfer the data to the display terminal as well as to receive commands from the display terminal and pass them on to the appropriate motes. The display terminal can be a personal computer, PDA, or a packaged mote with an incorporated digital signal processor with a liquid crystal display.

Communication between the transmitter and receiver mote was based on the star network topology (SNT), as shown in Fig. 5-1. Simply, the SNT involves data transmission from individual motes to the receiver mote only when the consent command is given by the receiver mote. Specifically, the receiver mote queues each available mote sequentially at a given time interval. During each queue, the receiver mote sends a command to a particular transmitter mote to send data. Once the command is received from the receiver mote, the transmitter mote needs to send the requested data before the Guaranteed Time Slot (GTS) expires. The duration of the GTS can be programmed depending on how many subjects are monitored simultaneously. It varies from 65 milliseconds to 200 milliseconds. When the transmission of data is completed, the mote becomes inactive and yields the radio channel to other available motes for subsequent data transfer. This data transfer scheme precludes transmitter motes from sending data at the same time. Therefore, competition between radio channels race is avoided. The SNT communication scheme reduces channel congestion and minimizes data loss.

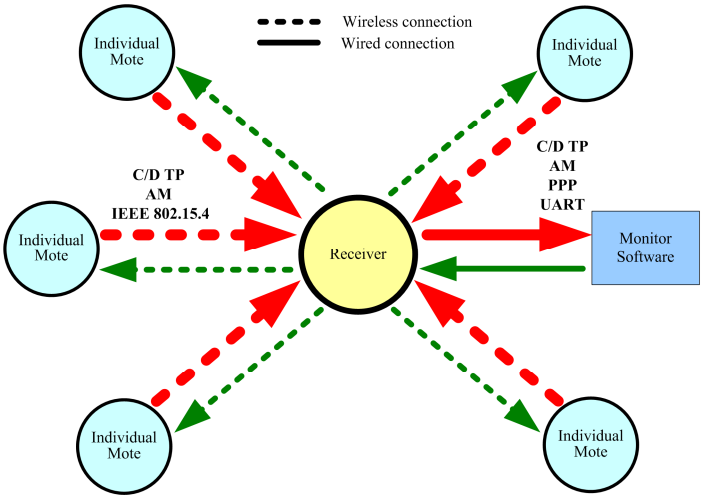


Fig. 5-1. System architecture of the Star Network Topology

5.2.2 The Hardware Gear

Tmote Sky is a wireless transmitter/receiver circuit originally developed by UC Berkeley. Its main components are comprised of a wireless dual-purpose transmitter and receiver and a microcontroller, together with other supporting peripheral components. *Tmote Sky*'s radio module is the Chipcon single-chip wireless transceiver – CC2420. The CC2420 is a single-chip IEEE wireless personal area network (PAN) standard 802.15.4 compliant (commonly referred to as Zigbee) wireless transceiver designed for low-power and low-voltage wireless applications [60]. It operates in the 2.4 GHz ISM band that is license-free in the USA as well as in most of the world, and provides a 250 kbps effective data rate. The CC2420 provides hardware support for packet handling, data buffering, burst transmissions, data encryption, data authentication, clear channel assessment (CCA), and packet timing information. These features reduce the load on the host controller and allow CC2420 to interface to low-cost microcontrollers [60]. The CC2420

has link quality indication as well as programmable transmit power control. These features make it possible to adjust the radio transmit power according to wireless link quality. This adaptive transmission power adjustment allows us to reduce power consumption. In our experiments we found it capable of transmitting data over a 30-50m range in indoor office/lab environments.

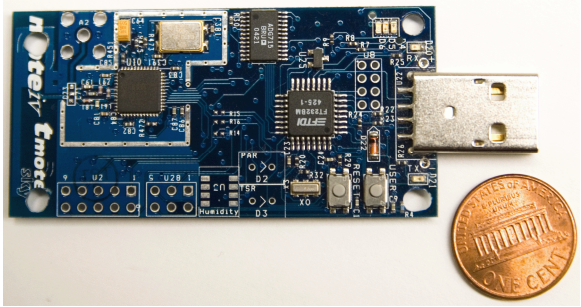


Fig. 5-2. Tmote Sky Wireless Transmission Module

On the *Tmote Sky* printed circuit board, the CC2420 is controlled by a low power consumption microcontroller, the MSP430F1611, which was developed by Texas Instruments. It provides 10 KB of RAM and 48 KB of flash memory for persistent storage and uses a 16-bit RISC CPU and 16-bit registers. The processor's clock speed is 8 MHz, which is fast enough for real-time implementation data collection and transmission. The MSP430 family also features extremely low active and sleep current consumption that permits long battery life. The digitally controlled oscillator allows wake-up from low-power modes to active mode in less than 6μs, which further provides more flexibility in power management. Another salient feature of the MSP430 is the integrated 8 channel 12-bit analog-to-digital converter (ADC) [60] allowing up to 8 analog biosensors to be connected. The 8 channel can be programmed at different sampling rates without interfering with each other.

Furthermore, the MSP430 can be programmed to sample the analog signals at varying sampling rates and has the capacity to store a small amount of data. These temporary stored data can be sent to a receiver mote when the MSP430 receives a command to transmit the data. The time delay between transmissions of data is dependent on how many motes are queued in the receiver and what the sampling rate for each mote is. In our system, this delay is less than 2 seconds when 10 motes are simultaneously active. Finally, the MSP430 can be programmed to transfer data to the CC2420 or a USB port using two universal synchronous/asynchronous receiver/transmitters (USART) ports designated as: USART0 and USART1. In the *Tmote Sky* mote, USART1 is used to control the USB port and USART0 is used to control the CC2420 for wireless transmission. In our system, the CC2420 was programmed to be periodically turned on and off according to the commands sent by the receiver mote. This allows USART0 to be available most of the time to communicate with the additional digital signal processing module, which will be discussed in detail in Chapter 7 of this dissertation.

5.2.3 Communication protocols

A communication protocol is a set of standards that specify the rules for data representation, signaling, authentication and error detection that are basic requirement in the general communication field. Communication protocols provide the necessary specification for the data to be correctly received and interpreted. In our system, several communication protocols have to be implemented and combined together to make the system work properly. These protocols are critical because they standardize all the network components, from hardware specifications to software implementation. These protocols are also used to control the medium access to avoid channel racing problems during the wireless transmission. Fig. 5-3 depicts how these protocols are responsible for different network mediums or layers. We have developed a protocol for the communication in the application layer, termed Command/Data Transport Protocol (C/D TP). The other four communication protocols in Fig. 5-3 are standardized. In the

proceeding paragraphs details regarding each of the five communication protocols shown in Fig. 5-3 will be described.

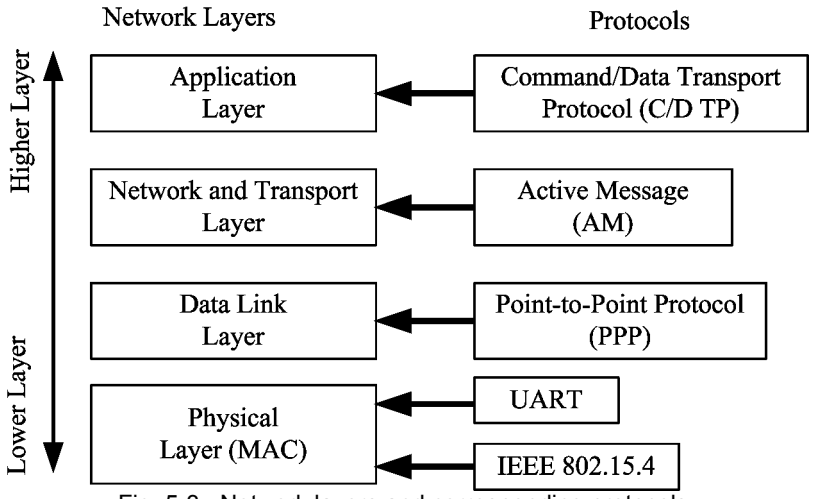


Fig. 5-3. Network layers and corresponding protocols

IEEE 802.15.4 Standard

Tmote Sky is compliant with the IEEE 802.15.4 wireless standard. This standard encompasses the MAC and Physical Layer (PHY) specifications for the Low-Rate Wireless Personal Area Networks (LR-WPANs). The CC2420 of the *Tmote Sky* fully supports the PHY requirement but has only limited support for the MAC layer. Consequently, the application software we have developed implements the MAC layer.

Universal Asynchronous Receive/Transmit (UART)

In the current version of our system, the display terminal (personal computer) and the receiver mote are connected via a USB port. The USB driver maps the USB port to a virtual serial communication port (COM port) so that data can be exchanged between a computer and the receiver mote via the standard UART link. To establish correct UART communications, the following three parameters need to be specified: the baud rate, byte size and stop bit. We configured the baud rate to be 262,144 bps, which is the highest value supported by *Tmote Sky*. The byte size and stop bit were configured to be 8 bits/byte and 1 bit without parity check, respectively.

The Peer-to-Peer Protocol (PPP)

The PPP is employed in *TinyOS* to provide generic data transfer. In our system, the PPP is implemented between the serial communications from the receiver to the computer. The PPP link, which is implemented by *TinyOS* for wireless communication, was combined by the C/D TP protocol that we have developed to gain more organized medium access control as well as more reliable and efficient wireless transmission. Unlike a general wireless PAN, in which the data transmission from each device is random and unorganized, the transmission time to send and receive data from each mote in our system is pre-determined. This feature offers much more flexibility and greater control of data transmission by simplifying the MAC strategy. The star network topology architecture and the hardware design of the mote can physically establish a point-to-point link between a mote and the receiver when each mote is controlled by C/D TP.

Active Message (AM) protocol

Active Message (AM) protocol, which is the standardized component of *TinyOS*, was

adapted to our system to implement the transport layer functions. We modified the usage of AM to incorporate C/D TP since AM is not specific enough to describe the interaction behavior between the receiver and transmitters. With this modification, not only the data collected from different sensors can be transmitted to the computer, but also those decisions or commands issued by the result of data analyses can be delivered to each mote. This feature allows a person to program each mote online, tailoring the performance of each mote to the patients' medical histories.

Command/Data Transmission Protocol (C/D TP)

The C/D TP was specifically developed for our system. While it is designed for the application-layer protocol, it can also be used in the MAC layer. The C/D TP consists of `REQUEST-RESPONSE` in transmitting commands/data. A `REQUEST` C/D TP message requires a mandatory `RESPONSE` from the targeted recipient to ensure a successful communication link between the transmitter and receiver motes. A C/D TP `REQUEST` is typically a COMMAND that is issued to the target mote. The target mote must take some actions according to the `REQUEST`. For example, if the receiver wants to get some data from a certain mote, it sends a `REQUEST_GET_WODATA` message (see Table 5-2) to the intended mote. Once the target mote receives this `REQUEST` message, it knows that it should send the data back to the receiver immediately if the data are available. The `RESPONSE` message to a certain `REQUEST` command provides dual functions. First, the `RESPONSE` message functions as an acknowledgement of the receival of the `REQUEST` message. Thus, the sender of the `REQUEST` message knows in this way that the `REQUEST` message has been successfully received by the intended recipient. Second, the `RESPONSE` message returns the execution result of the `REQUEST` command. Instead of just returning a simple result like either "success" or "fail", the `RESPONSE` message returns critical data back to the sender. For example, the `RESPONSE` message to the `REQUEST_GET_WODATA` command contains the actual physiological data.

The structure of a typical C/D TP message with an arbitrary data field is provided in Fig. 5-4(a). The last two bytes in a C/D TP message represent the Channel Selection Word (CSW). As shown in Table 5-1, specific bits are assigned to different channels, which can represent different physiological parameters to be collected. In the current system, C/D TP supports up to 15 channels. However, given the presence of a "reserved bit" (bit 15) in CSW, it can be easily extended to more than 30 channels, if necessary. C/D TP also provides the capability to obtain variable sampling rates for different physiologic parameters.

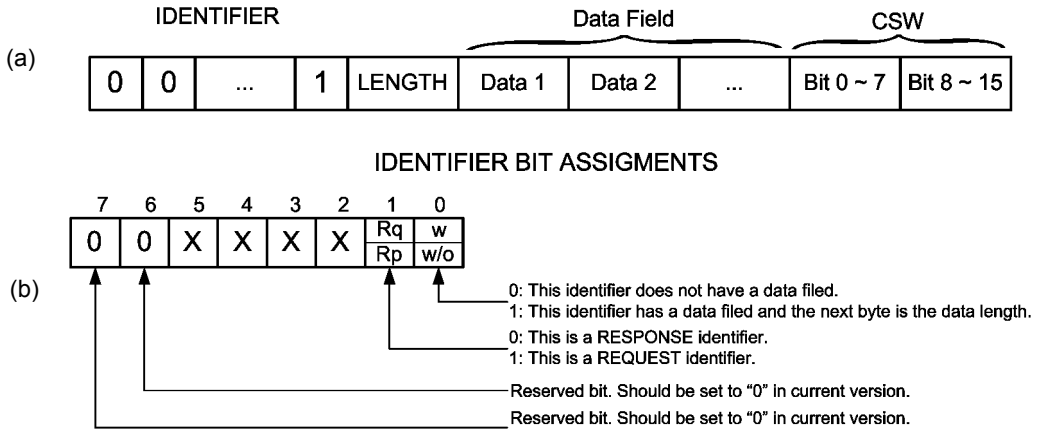


Fig. 5-4. Structure of a C/D TP message with data field. (a) Frame structure of C/D TP message; (b) Bit assignments of the IDENTIFIER filed.

The identifier shown in Fig. 5-4(a) is further detailed in Fig. 5-4(b). As shown in Fig. 5-4(b), bit0 indicates if the message contains a data field; bit1 distinguishes the REQUEST message from the RESPONSE message; bit2 to bit5 define the operation of the message; and bit6 and bit7 are not used but are available for future use, if necessary. Based on the above bit assignments, the identifier space is divided into the following four subspaces:

REQUEST with data field:	4×n + 3
REQUEST without data field:	4×n + 2
RESPONSE with data field:	4×n + 1
RESPONSE without data field:	4×n
	(n = 0, 1, 2, …, 15)

Table 5-1 Channel Select Word (CSW) in C/D TP

BIT	CHANNEL	BIT	CHANNEL
0	Not mapped	8	Not mapped
1	ECG	9	Not mapped
2	SpO2	10	Not mapped
3	BP	11	Not mapped
4	Respiration	12	Not mapped
5	Glucose	13	Not mapped
6	Temperature	14	Not mapped
7	Not mapped	15	Reserved

In the current version of C/D TP protocol, some COMMANDs and RESPONSEs have been defined. These COMMANDs and RESPONSEs involve the request of retrieving data from each individual mote, registering and configuring a mote, and changing sampling rate etc. The defined COMMANDs and RESPONSEs are shown in Table 5-2.

Table 5-2 Defined C/D TP Commands

ID	COMMAND	DESCRIPTION
0x02	REQUEST_REG_WODATA	Request to register a mote, no data field in the message
0x03	REQUEST_REG_WIDATA	Request to register the motes listed in the data field
0x06	REQUEST_DER_WODATA	Request to de-register the mote, no data filed in the message
0x07	REQUEST_DER_WIDATA	Request de-register the mote listed in the data field
0x0A	REQUEST_GET_WODATA	Request the mote to send data, no data field
0x0E	REQUEST_SCH_WODATA	Request to select signal channels, no data field
0x0F	REQUEST_SCH_WIDATA	Request to select signal channels, with data field
0x0B	REQUEST_CHG_SPRATE	Request to change sampling rate
0x17	REQUEST_GET_WIDATA	Request the mote to send data, with data field
0x1B	REQUEST_CHG_TIMEINT	Request to change the GTS interval
0x00	RESPONSE_GET_WODATA	Response to request of sending data, no data field
0x01	RESPONSE_GET_WIDATA	Response to request of sending data, with data field
0x04	RESPONSE_REG_WODATA	Response to request of mote registration, no data field
0x05	RESPONSE_REG_WIDATA	Response to request of mote registration, with data field

0x08	RESPONSE_DER_WODATA	Response to request of de-registration, no data field
0x09	RESPONSE_DER_WIDATA	Response to request of de-registration, with data field
0x0C	RESPONSE_SCH_WODATA	Response to request of selecting signal channels, no data field
0x0D	RESPONSE_SCH_WIDATA	Response to request of selecting signal channels, with data field
0x10	RESPONSE_CHG_SPRATE	Response to request of changing sampling rate, no data field
0x14	RESPONSE_CHG_TIMEINT	Response to request of changing GTS interval, no data field

5.2.4 Development platform and programming tools

The software used for SNT-based communication between the transmitter and receiver motes was developed by *nesC* language, which was designed to support the operating system known as *TinyOS*. *TinyOS* is an open-source operating system designed for wireless embedded sensor networks that have very limited resources [61]. It features a component-based architecture and an event-driven execution model. *nesC* is an extension of the C language, designed to embody the structuring concepts and execution model of *TinyOS* [62]. Specific details concerning communication protocols using *TinyOS* are described in the proceeding paragraphs.

TinyOS and *nesC*

One of the advantages of using *Tmote Sky* as the wireless transmission module is that it supports *TinyOS*. *TinyOS* is a small open-source operating system designed for wireless embedded sensor networks that have very limited resources [63]. It features a component-based architecture and event-driven execution model. This architecture and execution model makes the development tasks much easier and more systematic. In *TinyOS*, the operation system (OS) services are decomposed into separate components, allowing unused services to be excluded from the application. These system components are reusable. An application simply connects all the components that it needs by using a wiring specification; while the wiring specification is independent of component implementations. This solution provides much flexibility of programming when the code size is maintained as small as possible. *TinyOS* has two concurrency models: tasks and events. Tasks are a deferred computation mechanism. They run to completion and do not preempt each other. While on the contrary, events can preempt the execution of a task or even another event although they also run to completion. *TinyOS* execution is ultimately driven by events representing hardware interrupts. Events concurrency model is critically important in this real-time monitoring system. Many sources may generate a hardware interrupt, demanding a right-away response and processing. For example, an AD conversion, timer expiration, and a reception of command message etc.

To implement *TinyOS*, a new programming language, *nesC*, has been developed [63]. *nesC*'s contribution is to support the special needs in the domain of embedded systems. It incorporates the flexible event-driven concurrency model and component-oriented application design. By putting restrictions on the programming model, *nesC*'s compiler is able to perform whole-program analyses, including data-race detection. Early data race detection is very important in embedded real-time system, because most embedded systems are more prone to data race problem than other programs due to the fact that embedded systems have to respond to many external events and interrupts. To make it worse, it is extremely difficult to locate the data race problem at run-time because data race problems often occur irregularly. An embedded program may be able to run for a long time without any problems but can crash mysteriously due to an occurrence of data race. The feature of data-race detection at compiling time provided by *nesC* greatly enhances the reliability of the application code and shortens the development of the

embedded system. The whole-program analysis also perform optimizations at compiling time, simplifies application development, reduces code size, and eliminates many sources of potential bugs [63]. To take advantage of the available features of *TinyOS*, *nesC* is chosen as the development programming language at the transmitter and receiver ends in our system. *nesC* is a C-like language. Its compiler is composed of a "translator" and a general C complier. The translator translates the *nesC* source files into C files and the general C complier (like `gcc`) is employed to do the actual compiling work and generate the objective files. The data race detection is performed by the translator before the *nesC* code is translated.

Matlab, Microsoft Visual C++ and Borland Delphi

The monitor software was developed mainly by using Borland Delphi. Compared to Microsoft Visual C++ or other software development tools, Delphi is more suitable for the development of a software for which friendly user interface is of most importance. Taking into the consideration that most users of this system are health care givers and they do not have solid computer background, providing a friendly and easy-to-use interface in the monitor software is of much concern. Using Delphi can accelerate the development of such kind software because Delphi has provided many components that can be used by a developer. Besides, there are many third-party components available to enhance the performance of the software regarding display, data collection, and interactive functions etc. For example, the monitor software uses a well know serial communication component, `Spcom`, to read the data from the receiver through USB port. The use of these components saves a lot time and we can focus our attention on the improvement of the system's performance.

The algorithms we have developed are written in Matlab language. To incorporate these algorithms in the monitor software, either translates them into Objective Pascal (the programming language of Delphi) or somehow use the Matlab sources code directly. Luckily, Matlab has its own complier that translates Matlab M-files into C files. To facilitate the use of these Matlab-generated c source files, Matlab also provides the developer a standard math library, called Matlab C Math Library. This library is a collection of many build-in functions and c header files and source files defining data types, structures and functions etc. The advantage of using Matlab to implement those algorithms is significant. As we all know, Matlab was invented specifically for scientific computation. It has been optimized to provide a fast and accurate computation tool for scientists. Especially, it has a lot of built-in functions and tool boxes that can be used. Thus we can focus on the development of the algorithms themselves instead of being distracted by the details of programming. Although Matlab is an ideal tool for algorithm development and the source code can be compiled to C files, these C files cannot be used directly in Delphi. In this dissertation, Microsoft Visual C++ will be used to develop an "adaptor" project to fill the gap between the C files and Delphi. The "adaptor" project uses Dynamic Link Library (DLL) technique by which the C files will be compiled to generate a general DLL file that is loadable in Delphi. The DLL file exports a collection of functions, each of which performs a specific data analysis as they are developed in Matlab. The sequence of utilizing Matlab to develop data analysis algorithms and make them accessible by the monitor software is depicted in Fig.5-5. The detail of converting a Matlab program into a Delphi usable module is later discussed in 6.4.3 of this dissertation.

In general, these tools are needed in the development of the system:

TinyOS, *nesC*: They are used to develop the software residing in each transmitter mote and the receiver mote. The program written in *TinyOS* and *nesC* are responsible for data acquisition, medium access control, and data transmission etc.

Matlab: It is used to develop and implement all the sophisticated data analysis and algorithms to generate critical vital parameters at the monitor software end.

Microsoft Visual C++: Integrates and compiles the c files generated by Matlab into DLL file so that the monitor software can directly call the functions perform data analysis.

Borland Delphi: It is used to develop the interface of the monitor software. The monitor software reads data from the receiver via the USB interface or gets data from the server via Internet (discussed in detail in 6.3), displays the data to the user graphically. The monitor software also performs sophisticated data analyses by calling routines developed in Matlab. The monitor software also provides the tool to remotely configure the motes and issue commands to the motes.

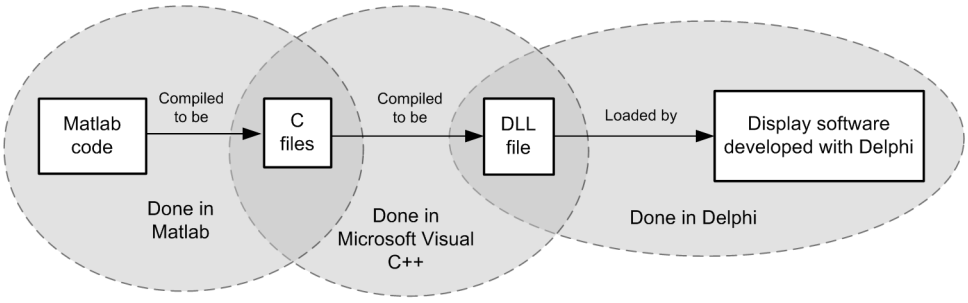


Fig. 5-5. To Make the Matlab program usable in the monitor software

5.3 Summary

The *Tmote Sky* is a general wireless transmission hardware platform that is compliant to IEEE 802.15.4 standard. The *Tmote Sky* circuit integrates the wireless transmission chip, CC2420, together with the ultra low-power consumption microcontroller MSP4301611. The *Tmote Sky* runs on *TinyOS* to provide a convenient hardware and software platform wireless transmission applications. We utilized SNT network architecture as the networking model of our system because a single receiver is designed to support multiple transmitters so that the cost of the system is reduced. The medium access is controlled and coordinated by the receiver. For any particular transmitter, it can send data only when a consent command is provided by the receiver. In addition to the implementation of standard protocols, we developed an application level protocol, termed C/D TP, to meet the specific requirements of our system. The C/D TP features a `REQUEST/RESPONSE` model, which results in better medium access control.

To expedite the development of our system, multiple programming tools are used in different part of the system. For sophisticated digital signal processing algorithms, the code was written in Matlab because Matlab is a very convenient tool for algorithm development. Algorithms developed by others are all written in Matlab. To translate the Matlab code into a module usable by the monitor software, Microsoft Visual C++ is employed to build DLL files based on the Matlab program. The exported DLL files are then loaded by the monitor software for data analysis. The monitor software is developed using Borland Delphi.

The embedded program residing in the transmitter and receiver are developed with *TinyOS* and *nesC*. *TinyOS* is an embedded operating system designed specifically for wireless mesh network. It utilizes a component-based and event-driven architecture. With a lot of system components available, the application development has been simplified by wiring these

components through standard interfaces. The source codes of these system components are also open to the public. The developer can modify the codes at will. To support features of *TinyOS*, *nesC* was invented. An application written in *nesC* typically consists of a configuration file and several module implementation files. The *nesC* complier translates *nesC* into C language file and then uses other C compliers (such as `gcc`) to generate objective files and then link these objective files to an executable binary file.

Chapter 6 System Implementation and Performance Assessment

6.1 Introduction

The system implementation is mainly about the software development at each individual transmitter, the receiver, and the monitor software in the monitoring center. The software development is conducted under the guidelines of the design as described in Chapter 5. For the development of the software at the transmitter, receiver, or the monitor software, different programming tools are used as discussed in Chapter 5.

At the transmitter, the functionality of the software mainly includes the digitizing analog signal from a analog signals (e.g. ECG); retrieving digital output from a module such as the SpO_2 circuit (MP506); understanding the commands issued by the receiver as well as packing and sending data to the receiver. The software at the transmitter end has three modules: Data collection module, C/D TP Parser, and Data Packing and Sending module.

The software residing in the receiver is of critical importance. All the transmitters are coordinated by the receiver. The software in the receiver not only relays data to the monitoring center and delivers commands to each transmitter but also facilitates the medium access control of the entire system. The receiver is responsible for authorization of the access to the radio medium for each particular mote in a particular time period.

The monitor software is developed for the monitoring center. The monitor software should be able to receive the data passed by the receiver and display the data in a real-time fashion. The monitor software is also equipped with real-time digital signal processing algorithms. These algorithms perform real-time data analysis to reveal more diagnostic-relevant information to facilitate medical alerts in a real-time manner.

The comprised system is also tested for its capacity and reliability. The test showed that a single receiver can coordinate 10 independent transmitter motes at a single physical frequency with each mote continuously sending the ECG data at 200 Hz sampling rate. If only a few processed parameters are sent instead of the raw ECG data, then the maximum number of the motes supported by a single receiver can be as high as 100 transmitter motes.

6.2 Software Development at the Wireless Transmitter End

The software residing in the transmitter has the following functional modules:

1. Data Collection Module

The system is built based on the ODT (On-command Data Transmission) scheme. When requested by the monitoring center, the transmitter sends raw data wirelessly instead of a few processed parameters. So, the transmitter mote should have the ability to directly access the output of an analog module, for example, the ECG circuit; digitize it and send it out. The software in the transmitter should implement such a module to achieve this goal. When sending raw data as needed, the module should be invoked. Analog data is not the only data format that the transmitter deals with. Some bio-sensor has digital output, for example the SpO_2 module we used in our system has the output of SpO_2 reading in a digital format. The SpO_2 reading is sent out by the MP506 via its on-board UART port. The communication protocol is specifically defined for the MP506. Thus, the transmitter software should also contain such a module to

correctly retrieve SpO_2 reading from MP506.

2. C/D TP Parser

At the application layer, we developed the C/D TP to facilitate the implementation of ODT scheme. The software in the transmitter should be able to understand the meaning of the received C/D TP message issued by the receiver mote. The transmitter mote is supposed to respond to the C/D TP message properly. A C/D TP parser is required to interpret the C/D TP message so that the recipient will take proper actions as specified by the command in the message.

3. Data Packing and Sending Module

The data and command sent by the transmitter (and receiver) is packed in a certain format according to the protocols discussed in Chapter 5 so that the message can be correctly interpreted and understood. The transmitter program implements a function module to pack data and commands in terms of the protocols. The data packing module can be considered as a functionally reversed procedure of the parser.

6.2.1 Data collection module

A primary function of the software at the transmitter end is to collect data from multiple bio-sensors. According to the property of the output data, the bio-sensors can be categorized into analog sensors and digital sensors. Collecting data from different types of sensors need quite different approaches. For analog signals, the sensor should be connected to the transmitter's AD convertor. The software residing in the transmitter controls the sampling rate and digitization precision of the analog signal. For digital signals, a transmitter retrieves the data through a digital input/output port, typically a USART port, by complying with a certain communication protocol.

1. Analog bio-sensor – ECG Module

The ECG module developed by our lab is a two-channel circuit. The circuit board for ECG data collection is shown in Fig. 6-1. One of the salient features of the ECG circuit is the low power consumption of the circuit, as it is powered by the direct current of 3 volts (2 AA-size batteries). We utilize two power circuits, however, to increase the DC power from 3 to 5 volts. A MAX1706 chip is used to convert 3 volts to 5 volts and a MAX635 chip is used to provide negative 5 volts power source for the ECG circuit. The primary reason for the use of these power circuits is to stabilize the power source.

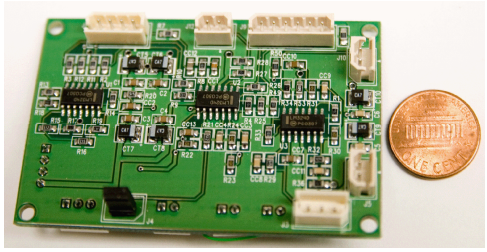


Fig. 6-1. ECG Module developed by our lab

Other features of the ECG circuit are amplifiers, filtering circuits, and a driven right leg circuit. There are two differential amplifiers in cascade to increase the gains 100 and 10 folds, respectively. Filtering circuits are comprised of a notch filter (to remove 60 Hz noise interference) and a low-pass filter with a cut-off frequency of 100 Hz. The driven right leg circuit was designed to further reduce instrumentation noise interference.

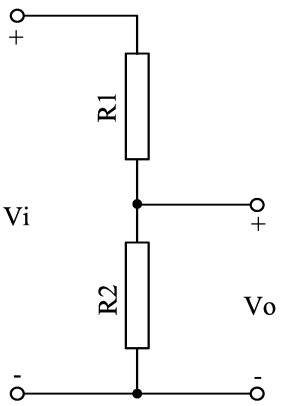


Fig. 6-2. Voltage Divider

In our system, the output of the ECG circuit has a range from 0 volt to 1 volt. The *Tmote Sky* AD input range is limited by the reference voltage, which is 1.5v and 2.v and is software selectable. So, it is safe to directly wire the ECG analog output to the *Tmote Sky*'s ADC input pin since the ECG voltage is below the reference voltage. If any additional analog circuit will be added to the system in the future that has output voltage higher than the reference voltage, a voltage divider should be used to bring down the voltage. A simple voltage divider is constructed by two resistors in series, as shown in Fig. 6-2. The transfer function of the divider is:

$$V_o = \frac{R_2}{R_1 + R_2} \bullet V_i \tag{6-1}$$

By choosing appropriate R_1 and R_2 values, the input voltage V_i can be rescaled down to a safe level within the limit of the ADC reference voltage.

An important issue regarding the analog signal collection is to control the sampling rate. In our system, we utilized a "Timer" to control the sampling rate. On hardware, the timer is a built-in circuit in the MSP430 microcontroller. *TinyOS* has abstracted the hardware Timer and offered a software component providing the `Timer` interface:

```
interface Timer {
      command result_t start(char type, uint32_t interval);
      command result_t stop()
      event result_t fired();
}
```

The `event result_t fired()` is the subroutine that is automatically called when the timer expires. This subroutine should be implemented by the application's program to take care of the expiration of the Timer. We can take advantage of the automatic call of this subroutine to sample the analog signal. Thus, controlling the expiration of the timer interval is equivalent to control the sampling rate.

The MSP430's built-in AD converter also has been abstracted by *TinyOS*. The AD converter can be accessed through the interface provided by *TinyOS*:

```
interface ADC {
      async command result_t getData();
      async command result_t getContinuousData();
      async event result_t dataReady(uint16_t data);
}
```

The AD conversion is completed in two steps. First, the `async command result_t getData()` is called to trigger the conversion. It typically takes a short time (measured in "clock" cycles, around several microseconds or more, depending on the configuration of the AD convertor) for the converted data to be stable. Once the converted data is ready, the subroutine `async event result_t dataReady(uint16_t data)` is automatically called. The application program should implement customized code in this routine to read the data out of the AD converter's register, and perform further processing, if needed.

By utilizing the `Timer` and `ADC` components in *TinyOS*, it is very convenient to construct the code to sample an analog signal at the desired sampling rate.

First, determine the timer interval in terms of the desired sampling rate. Suppose the

timer interval is T, measured in milliseconds, the sampling rate is S, measured in Hertz, then the following relationship hold:

$$T = \frac{1000}{S} \tag{6-2}$$

The `Timer` is invoked by calling: `Timer.start(TIMER_REPEAT, T)` in *TinyOS*, where T is a value written into the timer's register and, it must be an integer. Due to this requirement, there may be a round-off error in Eq. (6-2) for some arbitrary sampling rates. Besides, *TinyOS* 1.x divides one second into 1024 nominal milliseconds for hardware optimization reasons. This also causes the actual sampling rate to be slightly off the desired one. This small difference may not affect the sampled data too much, but it can cause a problem when the monitor software displays the data in real-time fashion. This problem is further discussed in 6.4.2, and a solution is provided.

The built-in AD converter on MSP430F1611 has 12-bit precision. This precision is high enough to preserve all morphologies of an ECG waveform.

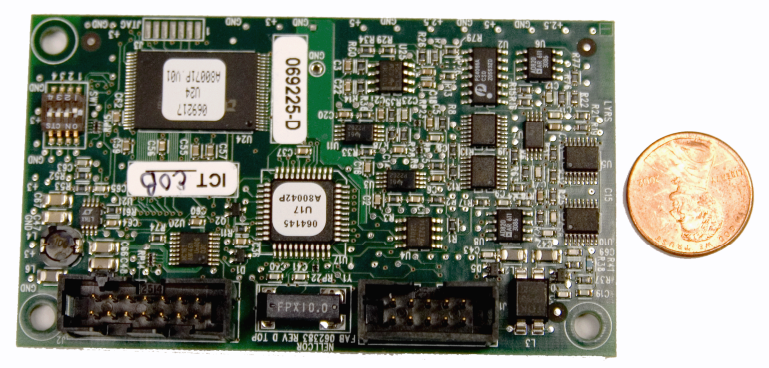
2. Digital bio-sensor – Pulse Oximeter

There are a few pulse-oximeter OEM (Original Equipment Manufacturer) modules commercially available. The Nellcor Oximetry MP506 is one of them. The MP506 uses a simple and flexible method of implementing non-invasive pulse oximeter. It is based on two physical principles: 1) the light absorbance of oxygenated hemoglobin is different from that of reduced hemoglobin, at the oximeter's two wavelengths, which include red and near infrared light; and 2) the absorbance of both wavelengths has a pulsatile component, which is due to the fluctuations in the volume of arterial blood between the source and the detector [64]. The two wavelengths employed in the MP506 are 660 nm for the red light and 890 nm for the infrared light, respectively. The MP506 is designed to interface with OxiMAX Sensors to provide the following patient data: 1) oxygen saturation (SpO_2); 2) pulse rate; 3) pulse waveform and pulse amplitude modulation (blip); 4) motion indicator*; 5) sensor disconnected indicator; 6) sensor off patient indicator*; 7) in sensor trend*; 8) sensor adjust*; and 9) sat-seconds alarm management*. ("*" indicates that this feature is available only in a certain operation mode.). The MP506 is capable of communicating with a host system via serial communication link. To facilitate a reliable communication, two protocols have been developed: Compatible Host Interface Protocol (CHIP), and Standard Host Interface Protocol (SHIP) by the company. The MP506 is compatible to both of these 2 protocols via a configuration switch selection. The data transmission rate is also optional, varying from 2,400 bps to 19,200 bps.

The MP506 printed circuit board is approximately 3.5 in. (89 mm) long, 2 in. (50 mm) wide, and 0.55 in. (14 mm) high. Its size is comparative to a credit card [65]. The small size provides great mobility. A sample of MP506 and the OxiMAX probe are shown in Fig.6-3(a) and (b), respectively.

Unlike the ECG module, which has an analog output, the MP506 module provides digitized output and the SpO_2 data should be read out through a serial communication link with the specified protocols (CHIP or SHIP) [65]. The pulse oximeter module is wired to the serial port on *Tmote Sky*. The *Tmote Sky* is programmed to periodically query the MP506 module to get SpO_2 readings. The query time interval determines the sampling rate of SpO_2, which can be easily preset or changed by the software.

(a)



(b)

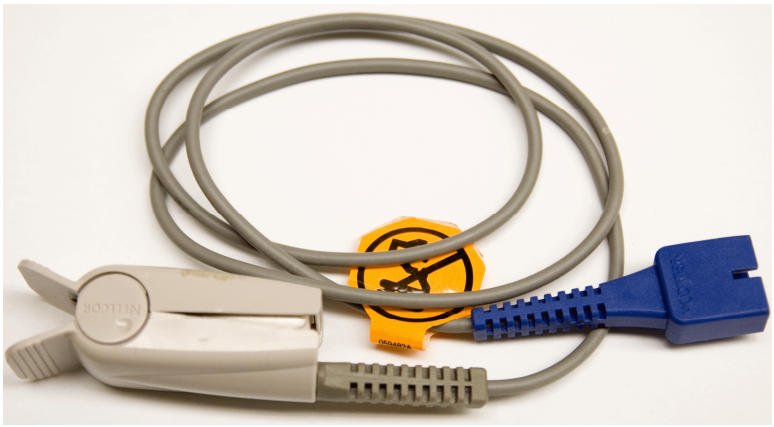


Fig. 6-3. MP506 circuit and probe: (a) the main circuit board; and (b) the OxiMAX probe

To understand the messages from the MP506, the *Tmote Sky* must be able to decode the message encoded by the MP506 communication protocol. In our system, we used the SHIP protocol supported by MP506. According to the MP506 technical document, the SHIP message has the frame structure shown in Fig. 6-4 [66]. All SHIP messages start with the hexadecimal value of `0x55` and end with `ETX` (hexadecimal value of `0x03`). The data field is further consisted of the message `KEY`, message `LEN`, and the message `VALUE`, as shown in Fig. 6-4.

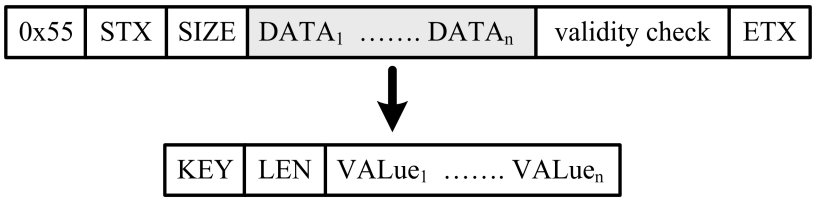


Fig. 6-4. SHIP message structure

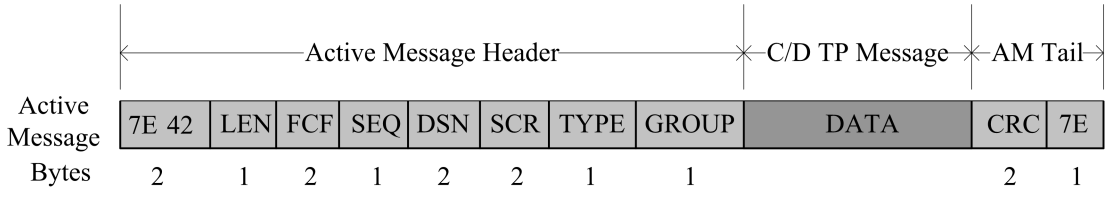
The most common way to implement a protocol is to utilize a Finite State Machine (FSM) [67, 68]. An FSM is a system behavior model. The model consists of finite system states, transitions between two states, and actions at the entry or exit of a state. The conditions leading to the state transition are also specified in the model. An FSM is best illustrated by a state diagram. In the diagram each state is represented by a circle tagged with the state name. The transition between two states is denoted by a straight line with an arrow to indicate the direction of the transition. The conditions triggering the transition are labeled on the line. In the implementation of communication protocol using FSM, the transition condition is simply the

receival of each valid byte.

The FSM method has been used multiple times in the work of this thesis. Besides the implementation of SHIP protocol, it also has been used in the R-wave detection algorithm (see 7.4.1) and sending C/D TP message by the additional microcontroller (see 7.3.2) etc. The usage of FSM can be seen not only in the embedded software, but also in the monitor software.

6.2.2 C/D TP parser

The communication between each transmitter and the receiver is mediated by a bunch of protocols, as described in 5.2.3. These communication protocols deal with different layers of the communication. At the communication layer, we developed the C/D TP protocol specifically for our system. In order to understand the commands issued by the receiver, the transmitter needs a specific software module to interpret the C/D TP messages so that the transmitter can take appropriate actions as response to the received command. The C/D TP Parser is such a software module to interpret the messages issued by the receiver mote. According to the protocols described in 5.2.3, a typical message exchanged between the receiver and the transmitter has the structure shown in Fig. 6-5.



LEN: Length of the Active Message;
FCF: Frame Control Field, specified by IEEE 802.15.4
SEQ: Message sequence number, required by IEEE 802.15.4
DSN: Destination address
SCR: Source address
TYPE: Message type, defined in *TinyOS*
GROUP: The sender's group ID
DATA: The AM data field
CRC: Cyclic Redundancy Check

Fig. 6-5. Structure of a typical Active Message

The starting point of the C/D TP Parser is the arrival of a valid Active Message. As shown in Fig. 6-5, a valid Active Message starts with `0x7E 0x42` and ends with `0x7E`. The reception of a valid AM message is handled by *TinyOS*. The receive component in *TinyOS* provides a receive interface defined as:

```
includes AM;
interface ReceiveMsg
{
      event TOS_MsgPtr receive(TOS_MsgPtr m);
}
```

When a valid Active Message is received, the receive component automatically trigger the event of `ReceivieMsg.receive()`. Notice that as shown in Fig. 6-5, the C/D TP message is embedded inside the data field of an Active Message, it is intuitive to implement the C/D TP parser in the custom subroutine automatically called when the `ReceivieMsg.receive()` event is generated. Since the integrity of the Active Message is guaranteed by the *TinyOS* system

component, the C/D TP parser starts with the assumption that the received Active Message does not have any corrupted data. With this guarantee, the flow of the C/D TP parser is simplified: first, the data field of the received Active Message is stripped out. Second, the first byte of the data field, which is the `ID` of the C/D TP message as shown in 5-4, is tested to see if the C/D TP message is a `REQUEST` or `RESPONSE` message, then the execution flow is branched by using the `switch` statement according to the C/D TP message `ID`:

```
if(_IS_REQUEST(tCDTPMsgRev.nId)){
      switch(tCDTPMsgRev.nId) {
            case REQUEST_GET_WODATA:
                  ……
                  break;
            case REQUEST_GET_WIDATA:
                  ……
                  break;
            case REQUEST_SCH_WODATA:
                  ……
                  break;
            case REQUEST_SCH_WIDATA:
                  ……
                  break;
            case REQUEST_CHG_SPRATE:
                  ……
                  break;
            ……
      }
} else {
      switch(tCDTPMsgRev.nId) {
            case RESPONSE_AVA_WODATA:
                  ……
                  break;
            case RESPONSE_REG_WODATA:
                  ……
                  break;
            case RESPONSE_REG_WIDATA:
                  ……
                  break;
            case RESPONSE_DER_WODATA:
                  ……
                  break;
            case RESPONSE_DER_WIDATA:
                  ……
                  break;
            ……
      }
};
```

Finally, the Channel Selection Word (CSW) is checked to find out which channel is affected by the received C/D TP message. For example, if the received message has a CSW of `0x02`, it means the command is applicable to ECG channel only (see 5.2.3 for the definition of each bit in CSW).

There is a trick in the use of the `switch` statement. At the transmitter end, the most

frequently received C/D TP message is the `REQUEST_GET_WODATA` message because the receiver keeps sending this message to the transmitter to assign a GTS for that transmitter mote so that it is authorized to access the radio frequency for data transmission. Thus, it is mostly likely that the received C/D TP message has the message `ID` of `REQUEST_GET_WODATA`. Obviously, it is more efficient to first test the C/D TP message `ID` against value `REQUEST_GET_WODATA`, as shown in the above code. The `REQUEST_GET_WODATA` is put at the first branching position of the `switch` statement. So in most cases, the rest comparisons between the C/D TP message `ID` and the other values do not need to be executed.

The C/D TP parser is not only implemented at the transmitter end, but also implemented at the receiver and in the monitor software (see 6.4). There is a slight difference among these implementations due to the functionality of the parser at different ends. Especially in the monitor software, there is no such system component as `ReceivieMsg.receive()` provided by *TinyOS* that guarantees the receival of a valid Active Message. As stated above, the starting point of the C/D TP Parser is the receival of a valid Active Message. So, the monitor software has to locate and identify a valid Active Message in the received byte flow by itself before the C/D TP Parser is bought into the front. In the development of the monitor software, the C/D TP Parser is embedded in a lower level subroutine that locates a valid Active Messages from the received byte flow. Combining the C/D TP Parser with the lower level subroutine brings some programming convenience and avoids some overhead.

From this C/D TP Parser example, we find that it is of much development convenience to modularize the general system functions. The transmitter and receiver have benefited from the support of *TinyOS*. The implementation of C/D TP Parser is much easier in the receiver and transmitter software than in the monitor software because the arrival of a valid Active Message is guaranteed by the *TinyOS* system modules. In the next step of this project, an additional microprocessor is added into the system to enhance the signal processing capability. The similar modularization principle is followed in the development of a software platform for embedding digital signal processing algorithms into an additional microcontroller, as discussed in Chapter 7. The software platform implements system functions in some stand-alone modules by providing encapsulated software components. Embedding DSP algorithms starts from these abstracted system components without being concerned of the low-level functionality. By this means, it is much convenient and efficient to embed more DSP algorithms into the system. Besides, modularization development method is also less error-prone.

6.2.3 Data packing and sending

The collected raw data or processed parameters should be sent to the monitoring center in a certain format. The data packing and sending module prepare the data according to the structure shown in Fig. 6-5 so that it can be correctly illustrated by the monitoring software. The packing is the reverse of the parsing procedure.

As the paring procedure starts from the “outer” of the message by stripping off the header and tail control information of the message, the packing initiates from the “inner” of the message. The data is first embedded in a C/D TP message by adding the C/D TP header and tail information. Then, the C/D TP message is inserted into the “data” field of an Active Message. The header and tail portion of the Active Message are added in accordance to the definition of the Active Message shown in Fig. 6-5. Messages generated in this way are standardized, so they can be correctly received and interpreted by the recipient.

One point should be clarified is that when an Active Message is being sent out over the

radio, the underlying hardware circuit will add more control sequence at the head and tail of the pay load [69]. These control sequence is important at the physical layer, but they will be automatically removed when the bit flow is received and verified by the recipient hardware. For the detail of the control sequence, the reader is referred to [69].

Packing messages is easy to carry out, but sending the messages is little complicated. As specified by the IEEE 802.15.4 standard, the maximum length of a message is limited by 128 bytes [69]. The Active Message header and tail themselves occupy 15 bytes as Fig. 6-5 shows. This overhead puts an efficiency issue on the transmission of data. Suppose the length of the Active Message is n bytes, then the pay load length is $n-15$. So the pay load efficiency of such a message is:

$$eff = \frac{n-15}{n} \times 100\%$$

If a single sampled value (2-byte) is transmitted by the message, the total length of the message is 17 bytes. Thus the eff is less than 12% ($(17-15)/17 \times 100\% \approx 11.8\%$). This means by average, for 100 transmitted bytes only 12 bytes are the actual sampled data while the others are just message control information facilitating the transmission. This is very inefficient since majority of the bandwidth is used to transmit the control information instead of payload. Obviously, large n will also increase the efficiency. However, the total length of the Active Message is limited by 128 bytes, so the efficiency also has an upper limit, which is:

$$eff_{\lim} = \frac{128-15}{128} \times 100\% \approx 88\%$$

Since the radio transmission is closely related to the power consumption of the system, it is not trivial to increase the transmission efficiency. In the future development of the system, it is of consideration to erase the Active Message protocol with the replacement of a mature C/D TP protocol so that higher efficiency can be reached.

The *TinyOS* has a system component that provides the send interface as:

```
includes AM;
interface BareSendMsg
{
      command result_t send(TOS_MsgPtr msg);
      event result_t sendDone(TOS_MsgPtr msg, result_t
      success);
}
```

Direct call of `BareSendMsg.send()` initiates the sending of an Active Message. The hardware and *TinyOS* take care of the detail of the sending protocol. When the complete Active Message is sent out successfully, the application program is automatically informed by the event `BareSendMsg.sendDone()`. The application program must respond to this event, as required by *TinyOS*.

6.3 Software Development at the Receiver End

The receiver plays a critical role in the system because all the command/data exchanged between each individual mote and the monitoring center is first passed to the receiver. So, one of the receiver's major functions is to behavior like a "bridge". This "bridge" relays data/command

to its specific destinations. For the data flow of "downwards" (from the monitoring center to each mote), the message is passed to the receiver via USB connection first and then delivered to its destination through a wireless radio. For the data flow "upwards" (from each mote to the monitoring center), the data is transmitted via a radio frequency first and then relayed to the monitoring center through a USB.

Another major function of the receiver is to control the medium access. In our system, a single receiver is designed to support multiple motes. It is of critical importance to coordinate these motes to send data in an organized way since all the motes are working on the same radio frequency, such that the transmission collision is minimized. A random medium access scheme results in an unacceptable collision rate in such a narrow bandwidth. We have tested such that without the coordination of the receiver; even two motes cannot send reliable data with the collision rate lower than an acceptable level. The software in the receiver facilitates the implementation of MAC, as mentioned in 5.2.1, by providing such coordination function.

The receiver also has a similar C/D TP parser as described in 6.2.2. Again, the parser is used to understand the command the receiver obtains so that a correct action are carried out by the receiver.

6.3.1 Relay command/data upwards and downwards

The receiver software has two different communication interfaces to deal with: USB and wireless radio. At hardware configuration aspect, the USB controller chip is connected to MSP430's USART1 port. Thus, sending or retrieving data from USB port is equivalent to the same procedure of a general USART port. In *Tmote Sky*, the USART1 port is configured in UART mode. Thus, UART communication protocol parameters have to be set up before the USART1 port can work properly, which involves baud rate, stop bit, odd/even check bit, and character bit. The *TinyOS* supports some commonly used baud rates: 9,600 bps, 19,200 bps, 38,400 bps, 115,200 bps and 262,144 bps. Based on the consideration that the bandwidth of the wireless radio channel is 250,000 bps, we chose 262,144 bps as the UART baud rate because this is the only baud rate that is higher than the wireless bandwidth. If choose a baud rate lower than this, the receiver may not be able to relay all the data it receives from the wireless radio to the monitor software. For other UART configuration values, we chose the following *TinyOS* default setting: 1 stop bit, no check bit, and 8-bit character.

TinyOS has already abstracted the communication through USART port by providing a system component. The system component exports a convenient interface `HPLUART` to facilitate the realization of UART communication in the application program. The `HPLUART` interface is defined as:

```
interface HPLUART {
      async command result_t init();
      async command result_t stop();
      async command result_t put(uint8_t data);
      async event result_t get(uint8_t data);
      async event result_t putDone();
}
```

Calling of `HPLUART.put()` sends a single byte to the UART port; when a valid byte is received by the UART port, an event of `HPLUART.get()` is generated. To process the received data, the application code should be put in the body `HPLUART.get()`. In the receiver program, the UART port is used to communicate with the monitor software, so a C/D TP Parser is

implemented in the `HPLUART.get()` body. As discussed earlier, the C/D TP Parser is implemented using an FSM.

6.3.2 Coordinating medium access

One of the most important functions of the receiver software is to control the medium access. The transmission medium of the radio frequency is the open air, thus any transmitter mote is able to access this medium freely. The access to the medium must be controlled to avoid transmission mishap.

The receiver software facilitates the medium access control by providing a Time-Division Multiple Access (TDMA) scheme. In our system, the transmitter has to be registered by the receiver before it can access the radio medium. The receiver issues consent command only to the registered motes authorizing the access to the medium for that particular mote in a particular time period, called Guaranteed Time Slot (GTS). So, the radio frequency is time-divided. In any GTS, there is only one active transmitter sending the data via the radio frequency. By controlling the medium access in this organized way, the collision from other transmitters in the system is eliminated. This strategy not only minimizes the traffic congestion, but also uses the bandwidth very efficiently. As a consequence, more transmitters can share the same frequency channel. In our system, 10 transmitters are able to send ECG data sampled at 200 Hz on a single frequency channel due to the TDMA scheme.

To realize the TDMA scheme in the system, certain support from hardware and software are needed. On the hardware end, the wireless transmission chip, CC2420, has a built-in circuit that supports hardware addressing features. Each CC2420 can be assigned a hardware address, which is stored in the RAM address from `0x168` to `0x16B`. When the hardware receives a certain bit flow that complies with the IEEE 802.15.4 standard, the address information in the bit flow is compared to the value stored in the above RAM position by the hardware (not by software). The comparison is automatically performed by the digital logic circuit residing in the chip CC2420. Thus, for the *Tmote Sky* board, no assistance from the microcontroller is needed for the address recognition. This feature certainly saves the microcontroller time and simplifies the programming of *Tmote Sky*. The transmitter and receiver software takes advantage of this feature to implement the TDMA scheme. The commands sent by the receiver contain the destination address. Although the radio signal is received by all of the transmitters, only the particular mote specified by the address information in the message will accept the command message and pass it to the microcontroller for further processing.

The hardware address recognition feature can be turned on and off by the software. *TinyOS* has a radio control component, `CC2420ControlM`, which provides interface to manipulate the configuration of CC2420. Besides enabling or disabling the hardware address decoding, the interface also provides functions to tune the radio power or to select different radio channel. The `CC2420ControlM` component provides the `CC2420Control` interface as:

```
interface CC2420Control{
     command result_t TunePreset(uint8_t channel);
     command result_t TuneManual(uint16_t freq);
     command uint8_t GetPreset();
     command uint16_t GetFrequency();
     async command result_t VREFOn();
     async command result_t VREFOff();
     async command result_t OscillatorOn();
     async command result_t OscillatorOff();
```

```
        async command result_t TxMode();
        async command result_t TxModeOnCCA();
        async command result_t RxMode();
        command result_t SetRFPower(uint8_t power);
        command uint8_t  GetRFPower();
        async command result_t enableAutoAck();
        async command result_t disableAutoAck();
        async command result_t enableAddrDecode();
        async command result_t disableAddrDecode();
        command result_t setShortAddress(uint16_t addr);
}
```

A call of `CC2420Control.enableAddrDecode()` will enable the hardware address recognition function. Calling of `CC2420Control.SetTFPower()` will set the radio radiation power. This feature can be used for power control. If the transmitter is close to the receiver, it is possible to reduce the radio radiation power so that more battery power can be saved while the receiver still has very good signal reception.

As mentioned in the above paragraph, all the transmitters must be registered with the receiver so that the receiver has the knowledge of which transmitters are currently active. This information is mandatory for the receiver to implement TDMA scheme in order to avoid channel race. To achieve this goal, a linear data structure termed as linked list is used in the receiver software to maintain the registration of the transmitters. A linked list is one of the fundamental data structures in computer science. It consists of a sequence of nodes with each node containing a data field and a pointer field [70, 71]. The data field is the actual data for that node while the pointer field is the pointer pointing to the next logical node. The benefit of a linked list over a common array is that the logical order of the nodes in a linked list can be different from the physical storage order. This feature offers a linked list much flexibility of efficiently deleting or inserting a node at any position in the linked list. The structure of a linked list is illustrated in Fig. 6-6. A linked list is referenced by a single pointer pointing to the first node in the list. This pointer is named by convention as "head" pointer. Since each node in the list has the address information of where the next node is stored, giving the value of the head pointer guarantees the accessibility of all the nodes in the list. The most common operations on a linked list are traversal, insertion and deletion. Standard implementations have been developed for these operations in computer science. Thus, it is of much convenience to manipulate the linked list.

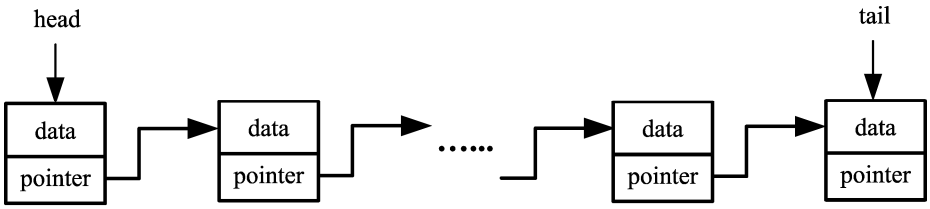


Fig. 6-6. Structure of a linked list

In our system, a single receiver is designed to support multiple motes. It is very possible that the number of simultaneously active motes may vary due to some reasons. For example, when a receiver is receiving data from 5 active motes, the monitoring center needs to add 2 more motes because 2 more patients are admitted. This involves the insertion of 2 motes into the receiver's registration list. On the other hand, the monitoring center may need to delete 3 motes from the receiver's registration list because 3 patients have been discharged from the system. This random insertion and deletion operation on a list is best implemented by a linked list data structure. In the receiver software, such a linked list is used to maintain the registered motes.

The linked list is initialized by the monitor software. At the start of the monitor software, the user is asked to select which motes are going to be monitored. Besides, the configuration information for each particular mote is also specified by the user. The configuration information includes what signal for that mote is going to be monitored, and what the sampling rate for the signal etc. This information is passed by the monitor software to the receiver so that the receiver can register the selected motes with the specified configuration. Further details of the monitoring software are provided in 6.4.

The data field of a node in the registration linked list is defined as a C structure containing the identification number of a mote and the Channel Selection Word (CSW, see 5.2.3) for a particular mote:

```
typedef struct TMote
{
      uint16_t id;
      uint16_t csw;
} TMote;
```

The `id` field in the `TMote` structure is the hardware address of the mote. Having this information, the receiver can send a command to a particular mote. The `csw` field indicates what signal should be sent by a particular mote. When a receiver completes the generation of the registration list, the program begins to send a consent command to each mote in the list starting from a mote pointed by the `head` pointer. Consequent motes are sequentially polled by using standard linked list traversal method. When the tail of the list is reached, the program restarts the polling procedure from the `head` pointer again. If any motes need to be added into the list, the monitor software sends a command to the receiver specifying the ID and CSW of the new mote. The receiver program inserts the new mote into the list by performing a standard linked list insertion procedure without discontinuing the polling process. Similarly, if any motes are desired to be removed from the list, the user can send a command through the monitor software to the receiver. The receiver deletes the motes from the list by using a standard linked list deletion operation.

In summary, the TDMA medium access control strategy is achieved by the support of the hardware and the development of the software. The user first selects and configures the motes that need to be monitored through the monitor software. The monitor software passes this information to the receiver. The receiver generates a linked list based on the information to manage the active motes. When a command is sent out to a particular mote, the hardware address is contained in the message. Although the radio signal can be received by any motes, due to the hardware address recognition feature only the particular mote having the same address as the message will pass the message to the microcontroller for further processing. So, if a C/D TP command `REQUEST_GET_WODATA` (request the target mote to send back data during the following GTS, see 5.2.3) is sent out by the receiver, only the mote that has the same address contained in the message will sent the data back to the receiver. The command message is silently ignored by other motes. In common condition, the receiver iteratively sends the `REQUEST_GET_WODATA` command to the registered motes in the linked list one by one, with the time interval of GTS duration. It is important to note that it is both the hardware address recognition feature and the `REQUEST/RESPONSE` mode of the C/D TP design that make the TDMA mechanism possible.

6.4 Monitor Software

The monitor software is developed for the monitoring center. Currently, the monitor software is developed based on the use of a personal computer (PC). Due to powerful processing capability of PCs, the monitor software provides a variety of functions. The basic function of the monitor software is to display the received signal in a real-time manner. The monitor software also features more sophisticated real-time digital signal processing analyses. Since the monitor software is mostly used by the health providers, it has been designed to incorporate user-friendly computer interface so that interactions between the user and the motes is simplified.

6.4.1 Overview of the monitor software functions

The current version of the monitor software has some basic functionality. The monitor software has a channel selection and configuration interface that allows the user to select which mote is going to be monitored and what signal on each mote should be monitored. By choosing the "`File→New…`" menu, the setup interface pops out, as shown in Fig. 6-7. The user not only can configure each mote, but also can choose the source of the data. Currently, the monitor software not only can retrieve the data from a receiver that is physically attached to the USB of the computer, but it can also get data from another computer running the same monitor software through the internet. The monitor software can also be configured as a "data server" that sends data out through the internet upon the user's choice. In the configuration interface shown in Fig. 6-7, the "`Data In`" option specifies the source of data; and the "`Data Out`" option determines if the data server in the monitor software should be on or off. The "`Channel Configuration`" part of the setup interface gives the detail information of the each mote. Unchecking of a mote will exclude that mote from being monitored. For each mote, different signal can be monitored simultaneously as the user can select only the interested signals separately for each mote on the setup interface. Currently, the system supports 10 motes for a single receiver, and for each individual mote the simultaneous monitoring of ECG signal and SpO_2 readings is supported.

If the "`Data In`" option is chosen "`From Serial Port`", when the new recording is started by choosing "`File→Start All`" the monitor software will automatically detect if a receiver mote is attached to the USB port. If the detection is successful, the monitor software will send the mote configuration information (see Chapter 6.3) to the receiver so that a registration procedure can be started. After a successful registration procedure, the receiver automatically starts its GTS timer and sends request messages to the motes on a continuous basis to coordinate data transmission. During this time, the monitor software queries the USB port to examine if any data has arrived, and if so, the monitor software decodes the received byte flow according to the protocols described in 5.2.3. The data are then continuously displayed in the computer screen. At the same time, the received data is also saved on the hard disk.

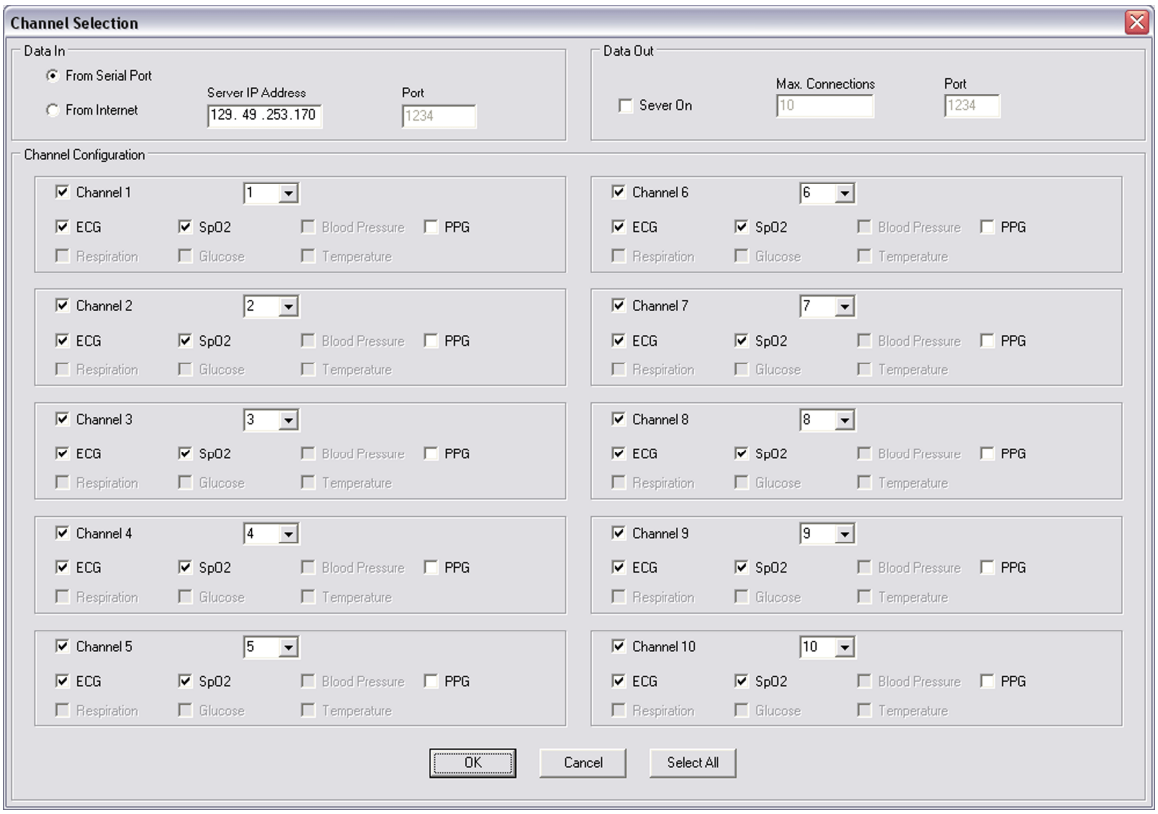
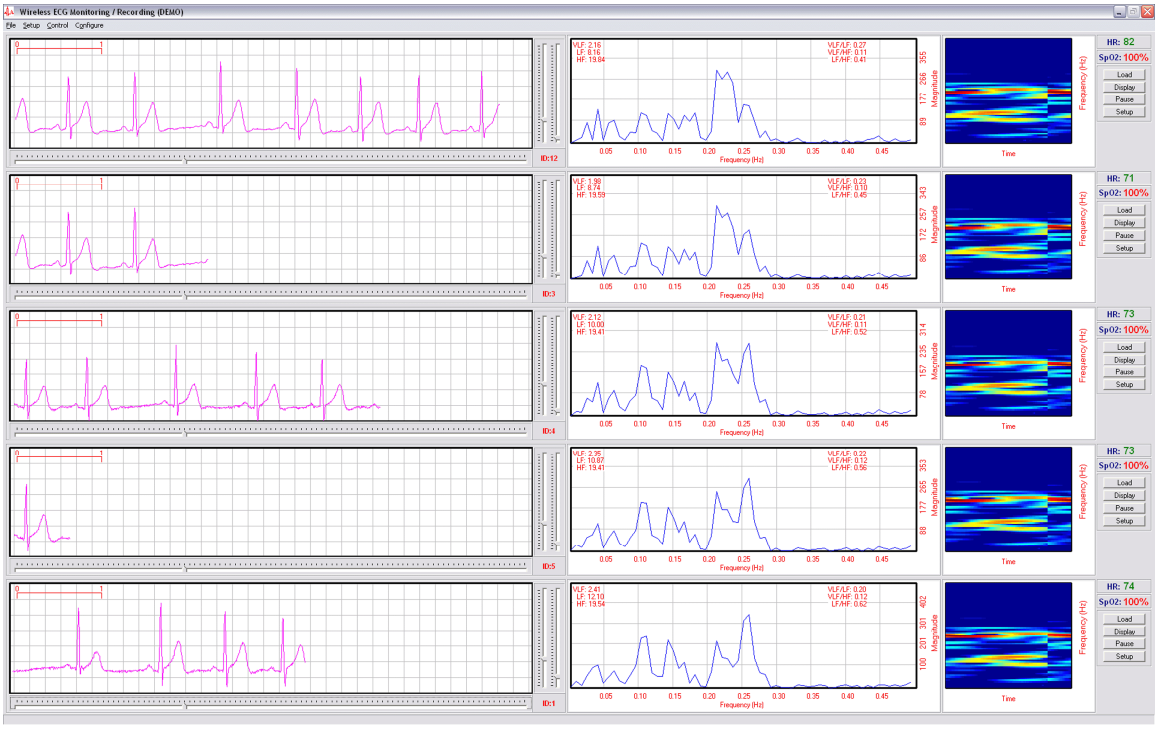


Fig. 6-7. The channel setup interface of the monitor software

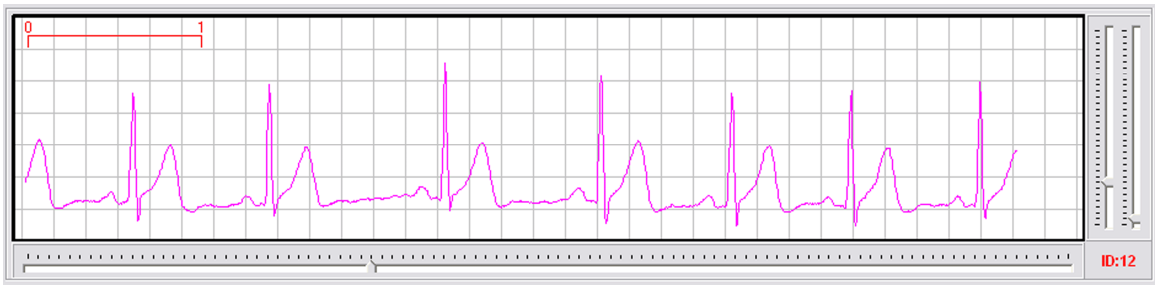
The expiration of the GTS timer inside the receiver is also reconfigurable. The monitor software has been designed to allow the user to specify the duration of a GTS as shown in Fig. 6-8. The default GTS value is 200 milliseconds which allows support 10 motes. In general, a larger GTS value introduces a longer display delay but it allows inclusion of more motes. We found that 200 milliseconds is long enough to support 10 motes and the display delay is around 2~3 seconds. Once a new GTS value is specified, the monitor software sends a C/D TP message to the receiver to change the GTS timer interval. The receiver returns a response message back to the monitor software when the attempt of changing GTS timer interval is completed. The response message indicates the result of the attempt, either success or fail. Thus, the `REQUEST/RESPONSE` rule of the C/D TP protocol is compliant (see 5.2.3).

Fig. 6-8. GTS Setup Interface

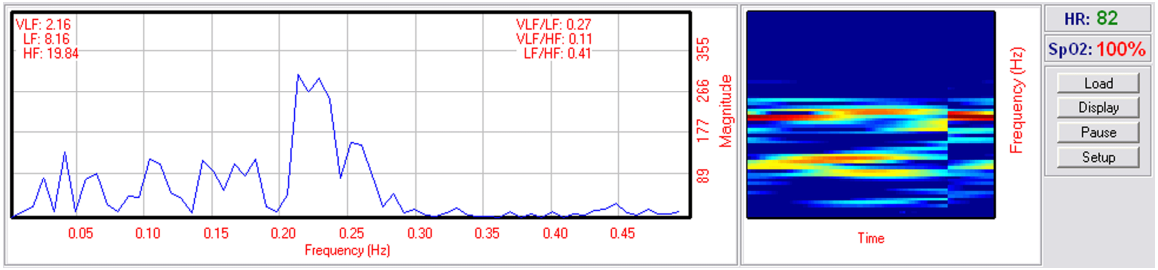
The monitor software also features real-time digital signal processing capabilities. The received ECG signal is first low-pass filtered by a built-in filter. R-wave detection is performed on the filtered signal and instantaneous heart rate is calculated according to the R-R interval. The monitor software consists of real-time analysis of both time-invariant and time-varying spectra of the detected RR interval data. By using the technique described in 6.4.3, the heart rate series is de-trended and its PSD is calculated based on the Welch periodogram method [72]. The monitor software also provides the estimates of the power contained in the low frequency (LF) and high frequency (HF) ranges as well as the power ratio between LF and HF ranges. Besides the instantaneous PSD analysis, the monitor software also features time-frequency spectrum analysis. The time-frequency spectrum is calculated based on the use of short-time Fourier transform with the segment size set to 128. It is updated every second.



(a)



(b)



(c)

Fig. 6-9. Monitor software. (a) main display interface; (b) displayed ECG signal of a single mote; and (c) displayed PSD and time-frequency spectrum of the heart rate series; shown together are frequency statistics of the PSD, instantaneous heart rate and SpO_2 reading.

Fig. 6-9 shows the display interface of the monitor software. Fig. 6-9(a) shows the main interface of the monitor software. Please note that our system is able to monitor 10 motes with a single receiver although only 5 display windows are shown in Fig. 6-9(a). The number of monitored motes is configurable as Fig. 6-7 shows. The number of display windows is automatically adjusted by the software according to how many motes are selected. Fig. 6-9(b) shows a window displaying a monitored ECG signal from a particular mote. Fig. 6-9(c) shows

the window that displays the PSD and the time-frequency spectrum. Also shown are the calculated heart rate, SpO_2 values and statistics derived from the PSD.

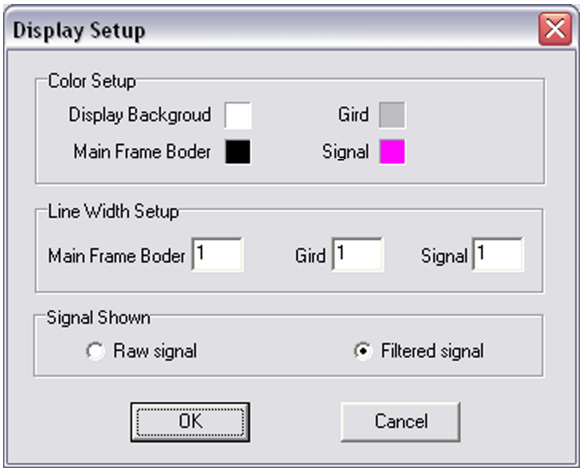


Fig. 6-10. Display Setup dialog

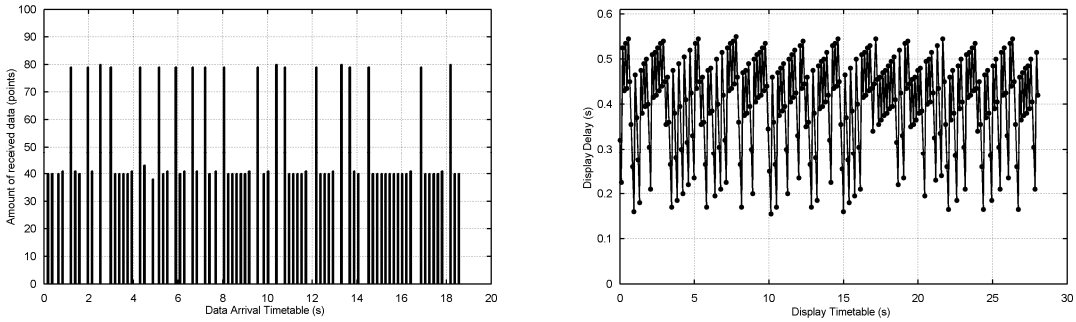
The display properties are also configurable. Fig. 6-10 shows the dialog for a display setup. The user can change the background and foreground colors, together with the line width of the signal and frame border. Particularly, a user can choose between the raw signal and low-pass filtered signal to be shown in the signal display window.

A fundamental function of the monitor software is to display the signal in a real-time fashion. Display buffering technique was used to achieve a smooth, and jutter-free real-time display of the signal. This is fully discussed in 6.4.2. Another important function of the monitor software is to perform real-time digital signal processing analyses. To take advantage of newly developed algorithms written in Matlab, special programming technique has been employed in our system to facilitate the adaption of these algorithms into the monitor software. This technique is presented in 6.4.3.

6.4.2 Buffering technique for real-time signal display

The data received by a computer is in a "pulse" fashion instead of continuous one, as Fig. 6-11(a) illustrates. This is due to the nature of the system that the receiver sends the consent command message to a particular mote at the beginning of each GTS. Only after the successful receival of the consent message can this mote be authorized to use the radio channel to send its data. From this particular mote's point of view, the radio channel is not always "open" for communication, but only open at a particular moment. Thus, the data transmitted from this mote is not continuous. A consequence of this is that a direct display of the transmitted data in the monitor software will make the signal look pulsatile. A solution to combat this problem was to have the signal delayed so that it is synchronized with the receiver. In particular, in order to display the signal smoothly and naturally, the buffering technique was employed in the monitor software. The received data are written into a display buffer on their arrival at the monitor software. The display subroutine in the monitor software reads a certain amount of data from the buffer and displays only this amount of data on the screen. The display subroutine is called periodically at a preset time interval. This time interval determines the update rate of the displayed signal; often known as Frame Per Second (FPS). To achieve a continuous display of the signal, a minimum value of 10 FPS is required. In the monitor software, we set the FPS as 12.5. Assuming the sampling rate of the signal is 200 Hz, the number of data points that should be read out from the buffer and displayed on the screen is calculated as: $200 \div 12.5 = 16$. The display subroutine maintains the position of the `read-pointer` of the buffer so that it knows the location from where the data should be read and displayed. The buffering technique separates the data that are to be displayed from the data that are received from a mote. So, this cheme restricts the pulsation-like appearance of the received data within the display buffer. The success of this solution relies on the synchronization and precision of the frame update timer residing in the monitor software and the timer set up in the transmitter mote for AD conversion. The *TinyOS*

1.x provides multiple timer components with different timing precisions, among which the best possible precision is 1 microsecond. It should be noted that *TinyOS* divides 1 second into 1024 "nominal" milliseconds. Therefore, 1 millisecond is wrapped to 1.024. Thus, in the calculation of how many data points should be displayed, this wrapping must be taken into account. The tests have shown that without this proper calibration, the data in the display buffer will continue to increase, which will result in further delays of the displayed signal. To fix this timing mismatch, the proper calibration should be carried. Since 1 real millisecond is wrapped into 1.024 nominal millisecond, 5 nominal milliseconds accounts for 5/1.024 actual milliseconds. So, the nominal 200 Hz sampling rate is actually 1024/5 = 204.8 Hz. As a consequence, $204.8 \div 12.5 = 16.384$ points should be displayed every time the monitor software updates the signal display. The data length in the display buffer reflects the display delay of the received signal. Since the data is read from a buffer, the length of the data in the buffer will exhibit a pulsatile pattern. Fig. 6-11(b) shows the pattern of display delay due to the pulsatile changes. The shown result is based 6 motes and 65 milliseconds GTS.



(a) (b)

Fig. 6-11. Pulsatile data arrival and display delay. (a) Data arrival amount vs. data arrival timetable; (b) Display time delay vs. display timetable

6.4.3 Importing Matlab program into the monitor software

Our laboratory has developed many digital signal processing algorithms that are applicable to analyzing physiological data. For example, the coherence analysis [36, 50], the Principle Dynamic Mode (PDM) analysis [57], and atrial fibrillation (AF) detection etc [73, 74]. However, these algorithms were original developed in Matlab for convenience. Although it is possible to convert the Matlab code into C code or Pascal code by manually rewriting the algorithms, it is also time-consuming. Fortunately, with the help of several tools the algorithms written in Matlab can be automatically converted into a Dynamic Link Library (DLL) which can then be accessed by the monitor software. The advantage of this development approach is obvious. First, the development of the algorithms and the interface is separate and can be carried out in parallel. Second, the development of the display interface and the algorithms is made as different modules. Modification of each module does not affect the other. And finally, the algorithms are encapsulated in the DLL files, thus the algorithms are kept confidential.

The monitor software is developed using Delphi with the aid of Objective Pascal programming language. However, the Matlab does not provide the support for Objective Pascal. Fortunately, Matlab provides a full support for C/C++ language. Our solution is to first convert the Matlab code into a C code; then modify the C code and compile the C code to generate DLL files. As well known, DLL files are language independent and they can be loaded almost by any high-level programming languages including Objective Pascal.

The Matlab complier provides an automatic way to convert M-files into C or C++ files.

This eliminates the time-consuming and error-prone manual translation process. These following features are offered by the delicate Matlab complier [75]:

- *Distribute C and C++ applications easily and freely to colleagues*
- *Include MATLAB math and graphics in end-user applications*
- *Generate, rotate, zoom, and scroll MATLAB 2-D and 3-D plot types*
- *Incorporate MATLAB GUI controls into your application*
- *Display lighting and shading on 3-D surfaces*
- *Compile toolbox M-files for inclusion in your stand-alone applications*
- *Protect proprietary algorithms to prevent users from modifying source code*
- *Compile, edit, and run your application from within Visual Studio*

The complier can be invoked by typing "`mcc`" in the Matlab command window. The typical usage of the complier is:

```
mcc [-options] fun [fun2 ...] [mexfile1 ...] [mlibfile1 ...]
```

For the detail usage of the complier, please be referred to the Matlab Complier manual [76].

The C code generated by the Matlab compiler uses a specific data type defined by Matlab, which is `mxArray`. The function arguments in the generated C code are passed to the functions. So, we directly compile the generated C code to export DLL files; the host program that uses the DLL must be able to provide the `mxArray` data type. Unfortunately, Objective Pascal does not have this kind of data type; Matlab does not provide a support for Objective Pascal as it does for C. Thus, it is not feasible to directly compile the generated C code to export DLL files. To solve this problem, we developed an "adaptor" project in C++ to seal the usage of `mxArray` type inside the DLL only by providing external interfaces that take regular type of arguments.

To explain how to convert an M-file into C code and make the appropriate modification, let's take a look at an example. Suppose in Matlab, a file named `myPSD.m` is written to calculate the PSD of a time series using the Matlab built-in FFT routine. The function `myPSD` is defined M-file as:

```
function Pxx = myPsd(x, overlap)
```

It takes two arguments: the first argument `x` is the time series; and the second argument `overlap` indicates how much overlap between two consecutive windows when calculating the PSD. The return value is the PSD, which is a function of frequency. By using the `mcc` compiler command, a `myPsd.c` file is generated according to the `myPsds.m` file. In the `myPsd.c` file, there are several defined functions that are different from its original M-file in which only a single function `myPsd()` is defined. Among these generated C functions, the one actually doing the PSD calculation is the `mlfMypsd()` function:

```
mxArray * mlfMypsd(mxArray * x, mxArray * noverlap) {
      ……
}
```

Notice that all the arguments type has been changed to `mxArray`! If this function were directly compiled to generate DLL file, the monitor software would not be able to correctly pass the data to this routine nor could the monitor software get correct result from this routine because the monitor software does not have `mxArray` data type.

The "adaptor" project provides an alternative interface as a replacement of the routine

mflMypsd(). First, in the header file of the project, the following statement was added:

```
extern "C" _declspec(dllexport) void _cdecl
matPSD(double *x, int len, int* m, double* psd);
```

This statement tells the C compiler that in the exported DLL file, there is a function named matPSD(). It takes four arguments as specified by the argument list. x is a regular pointer pointing to a double array. It is the time series that needs to be analyzed. len is a regular integer argument specifying the length of x. m is a returned integer value storing the length of the resultant PSD; and psd is another double pointer indicating where the resultant PSD is stored. In the source file of the "adaptor", the definition of the matPSD() function converts the regular data type in the argument list into mxArray data type and calls mlfMypsd() to perform the actual calculation of PSD:

```
extern "C" _declspec(dllexport) void _cdecl matPSD(double*
x, int len, int* m, double* psd){
      mxArray* s = NULL;
      mxArray* p = NULL;
      double* ptmp = NULL;
      InitializeModule_mypsd();
      mlfAssign(&s, mlfDoubleMatrix(len, 1, x, NULL));
      mlfAssign(&p, mlfMypsd(s, mlfScalar(0)));
      ptmp = mxGetPr(p);
      *m = mxGetM(p);
      memcpy(psd, ptmp, sizeof(double)*(*m));
      TerminateModule_mypsd();
      mxDestroyArray(s);
      mxDestroyArray(p);
}
```

The library function mlfAssign() provided by Matlab converts an argument of regular double type into mxArray type In the second call of mlfAssign() shown above, the mlfMypsd() is called to calculate the PSD. To convert an mxArray argument into regular double type, the C standard system routine memcpy() is used. Now we can see in the above code that the DLL provides the matPSD() interface taking regular type of arguments but using Matlab generated function mlfMypsd() to calculate the PSD. From the host program point of view (which is the monitor software in our system), the function mlfPsd() is invisible but it can be used by the host program through the interface of matPSD(). The following example code extracted from the monitor software shows how to use the DLL to calculate the PSD of a time series:

```
type TPSD = procedure(p: pointer; l: integer; m:
pointer; res: pointer); cdecl;
FuncPSD: TPSD;
Module: THandle;
FuncAddress: TFarProc;
……
Module := LoadLibrary('Adaptor.dll');
FuncAddress := GetProcAddress(Module, 'matPSD');
FuncPSD := TPSD(FuncAddress);
……
FuncPSD(HR, NFFT, @PSD_len, HRPSD);
```

The `type` statement defines a new data type which is a `procedure` with four arguments. Notice that the definition of this new type is consistent with the declaration of the `matPSD()` function in the header file of the "adaptor" project. This consistency must be absolutely guaranteed; otherwise the result will be catastrophic. The variable `FuncPSD` is declared as the `TPSD` type, that is, a `procedure` with four arguments. Before the `FuncPSD` can be called to do the PSD analysis, it must be initialized first. The initialization of `FuncPSD` is to "link" `FuncPSD` to the actual executable code segment inside the DLL file. Be aware that there may be many functions embedded in the DLL file, so the `FuncPSD` must be directed to the specific one (`matPSD()`) by some means. This is achieved by calling the standard Windows Application Programming Interface (API) function `GetProcAddress()`. Once the `FuncPSD` is successfully initialized, it can be called like a regular function, as demonstrated by the example code.

With the assistance of the "adaptor" project, importing a Matlab program into the monitor software is made as a template. First, use the Matlab compiler `mcc` to convert M-file into C file. Then declare the external interface by taking regular type of arguments in the header file of the adaptor project. Finally, in the definition of the declared external interface, make the conversion between regular data type and `mxArray` type by calling `mlfAssign()` function and call the `mlf*` function to do the actual processing. The DLL file generated in this way can be loaded by the host program just like a regular DLL.

6.5 Performance Assessment

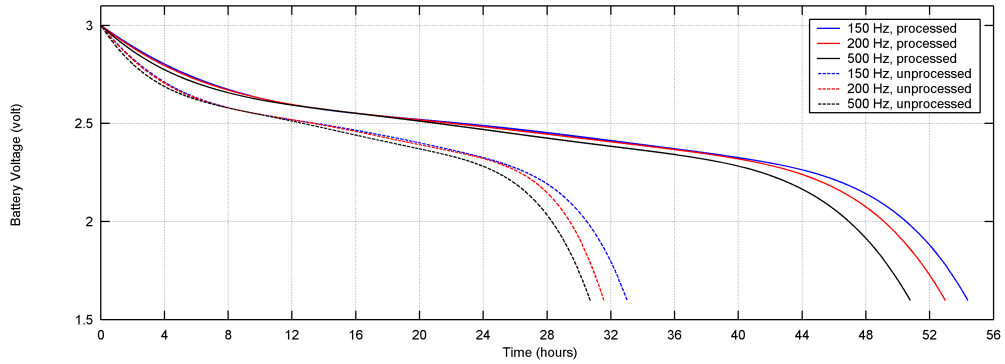
Preliminary tests have been administrated on the current version of the system. The tests involve the battery life experiments and the validation of the collected signal. The battery experiment is to investigate the power consumption of the system in the scenarios of transmitting raw data versus transmitting a few processed parameters under different sampling rates. The signal validation experiment compares the ECG signal recorded by our system against the signal obtained by a commercially available ECG machine. The comparison not only involves comparing the ECG waveform, but also involves the comparisons of extracted parameters from the ECG signals.

6.5.1 Power consumption

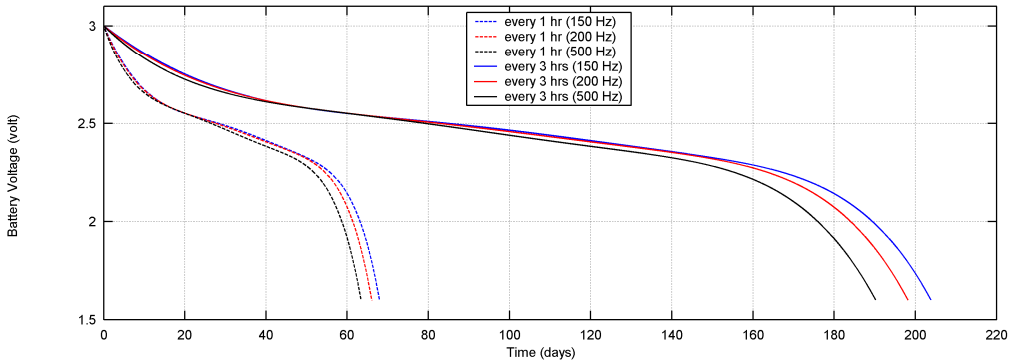
Power consumption is a critical issue in any battery-powered system. In our system, majority of the power is consumed by the wireless transmission. For example, the *Tmote Sky* consumes very little power when wireless transmission is not active. In this mode, the power consumption is only 1.8 mA; while when the wireless radio is active, the power consumption increases to 21.8 mA in receiving mode and 19.5 mA in transmitting mode [77]. To use the battery power more economically, we developed the ODT (On-command Data Transmission) smart sampling rate data transmission scheme combined with localized data processing. By transmitting only a few processed parameters instead of raw data, battery power can be dramatically preserved, as shown in the following battery lifetime experiments.

The device is powered by 2 AA batteries. The ECG signal is sampled at 150, 200, and 500 Hz, respectively. For each sampling rate, the battery life of sending the raw data and sending only processed parameters every second is compared. Based on this information, the battery life of sending the processed parameters very 1 and 3 hours is also compared.

Fig.6-12(a) shows the comparison of battery lifetimes for three different scenarios: (1) transmission of unprocessed ECG data at 150Hz, 200Hz and 500 Hz; (2) continuous transmission of 4 derived parameters computed over two minutes of R-R interval recordings; (3) transmission of the derived parameters every 1 or 3 hours. Please note in Fig. 6-12, different lines indicate the result of transmitting raw data (dashed line) or processed parameters (solid line); and different colors of the line indicate varying sampling rates (blue: 150 Hz, red: 200 Hz, and black: 500Hz). As shown in Fig.6-12(a), the first scenario results in a battery lifespan of about 28 to 30 hours, while the second scenario results in a lifespan of 50 to 55 hours. The third scenario of transmitting the derived parameters (still over 2 minutes of R-R interval recordings) once every 1 or 3 hours is depicted in Fig. 6-12(b). We see that if we send derived parameters once in an hour, then the battery lifespan is around 65 days. The battery lifespan can be further extended to more than several months if the derived parameters are sent every 3 hours. We predicted the third scenario by turning the mote and the associated ECG circuit off during the silent period based on analysis over initial data. The device is turned on every 1 or 3 hours, when it starts to record and process the data for a short amount of time. For continuous monitoring of subjects in nursing homes or individual homes, updating low and high frequency spectral power values every 1-3 hours is sufficient, as these values are not likely to change much within this time frame. For all these three scenarios, the difference in power consumption with different sampling rates is primarily due to difference in R-wave peak detection processing load. In these experiments, we have assumed battery power supply cutoff at 2 volts, which is the minimum needed for a reliable collection and transmission of ECG data.



(a)



(b)

Fig. 6-12. Battery life span for sampling rates of 150 (blue lines), 200 (red lines) and 500 Hz (black lines). (a) Transmission of unprocessed ECG signal (dashed lines) vs. transmission of 4 locally derived parameters (VLF, LF, HF, and LF/HF) over 2 minutes of RR interval recordings (solid lines). (b) Transmission of the 4 derived parameters once every 1 hour (dashed lines) and once every 3 hours (solid lines).

6.5.2 Signal validation

To validate the fidelity of ECG signal obtained by our system, we used a commercially available ECG monitoring machine, Hewlett Packard 78354A, as a reference device. The comparison involved not only the raw ECG signal, but also the extracted R-R interval time series. The most important temporal and frequency parameters for heart rate variability analysis are also compared. Such parameters are LF/HF ratio, the standard deviation of normal-to-normal R-R intervals (SDNN), and the root-mean square of the successive difference of R-R intervals (RMSSD).

The experiment was conducted on a healthy subject in the supine position. 10 trials of 5-minute ECG signal were recorded both by the HP 78354A and our device at sampling rate of 200 Hz. In this section, only the results of a single subject are shown. Although, the results of a single subject do not have statistic significance, it still provides some basic information about the validity of the system. Since the system is still under development, more experiments and comparison will need to be conducted to verify the performance of the system.

The ECG waveform comparison is performed visually. Fig. 6-13 shows the comparison of two representative ECG waveforms. As shown, the two ECG waveforms are similar to each other. The most clinically important waves, P-wave, QRS complex, and T-wave are well preserved in both ECG waveforms. Inspection of the total 10 trials reveals consistent similar morphology of the ECG waveforms.

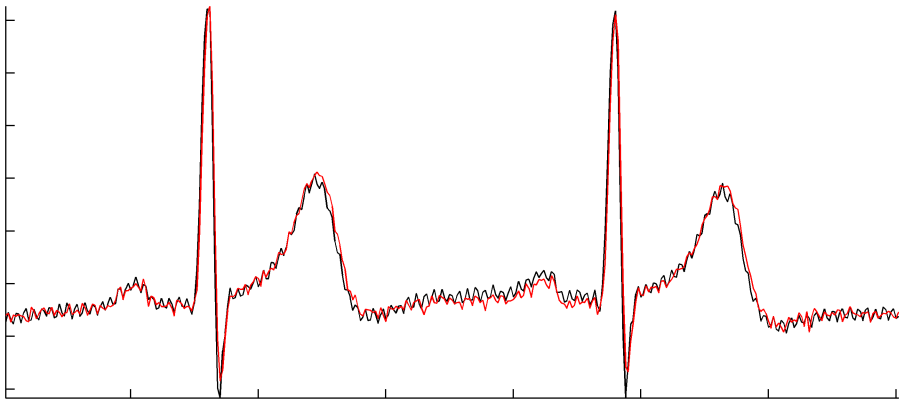


Fig. 6-13. ECG waveform comparison between the ECG signal recorded by the HP ECG machine (black line) and the ECG signal recorded by our system (red line).

In addition to the ECG waveform, some statistics of the R-R interval are of much more clinical interests and they can be used to generate medical alerts [78-81]. Thus it is very important to test the system if the fidelity of these parameters is also preserved. In this section, we compared the three most important parameters pertaining to heart rate variability analysis.

The first parameter we compared is the ratio between the low-frequency power and the high-frequency power, termed as LF/HF. The low frequency is defines as the range of 0.04 ~ 0.15 Hz and high frequency is 0.15 Hz ~ 0.4 Hz [53]. The ratio between the powers of these two frequency bands reflects activities of the Autonomic Nervous System (ANS). Specifically, it indicates the balance between the sympathetic and parasympathetic regulatory on the cardiovascular system since the low frequency is dominated by sympathetic activities and high

frequency is dominated by parasympathetic activities [82-84][1]. For example, in the supine position parasympathetic regulation is elevated while sympathetic is suppressed when compared to the upright position. Thus, the LF/HF ratio has smaller value in supine position than in upright position [53, 57]. By examining the ratio between low- and high-frequency powers, an approximation of the ANS balance can be obtained. Table 6-1 shows the comparison of LF/HF ratio based on the R-R interval obtained from our ECG signal and that from the HP ECG machine. It is shown that for all 10 trials, the LF/HF ratios obtained from two systems are similar. The forth column of the table shows that the absolute difference between the ratios is less than 0.013, which is only approximately 1.5% of the actual value. Note that the fifth column of the table shows that majority of the difference between the two systems in terms of a percentage is less than 1%, indicating that the ratios obtained from these two ECG signals are very similar to each other. Student-t test ($p < 0.05$) also shows that there is no difference between the ratios in the second and third column of the table. This suggests our system provides nearly identical results as the HP ECG machine when the LF/HF ratio parameter is used.

Table 6-1 Comparison of the LF/HF ratio between the R-R interval obtained by HP ECG machine and the R-R interval obtained by our system.

Segment	*HP ECG Signal*	*Our ECG Signal*	*Difference*	*Difference Percentage*
1	0.5584	0.5539	-0.0045	-0.8073 %
2	0.3422	0.3429	0.0007	0.2001 %
3	0.7164	0.7121	-0.0042	-0.5888 %
4	0.8704	0.8629	-0.0075	-0.8646 %
5	1.3195	1.3295	0.0100	0.7613 %
6	0.9465	0.9560	0.0095	1.0036 %
7	1.0477	1.0423	-0.0054	-0.5174 %
8	0.8186	0.8310	0.0124	1.5113 %
9	0.8555	0.8492	-0.0063	-0.7373 %
10	0.8274	0.8200	-0.0075	-0.9053 %

The second parameter we compared is the SDNN parameter. SDNN is the standard deviation of normal-to-normal R-R interval, measured in millisecond (ms). SDNN reflects all the cyclic components responsible for variability. SDNN is mostly used in the analysis of long-term ECG recording such as the 24h ECG monitoring. We compared the SDNN values between the two devices, and the result is shown in Table 6-2. Similar to the LF/HF ratio values, the SDNN values obtained from the two devices are similar. The difference between them is less than 0.3 ms for all 10 trials, as shown in the forth column of Table 6-2. The difference between the SDNN values is even smaller than 0.2%, suggesting that not only the absolute difference is small, but the relative difference is also very small. Statistic test (Student-t test, $p < 0.05$) shows that there is no difference between the two devices.

1 Some research also shows that it may be not accurate to assume sympathetic is in the LF and parasympathetic is in the HF. There might be overlap and interaction between these two regulation mechanisms. See reference [85-90].

Table 6-2 Comparison of the SDNN parameter (ms) between the R-R interval obtained by HP ECG machine and the R-R interval obtained by our system.

Segment	*HP ECG Signal*	*Our ECG Signal*	*Difference*	*Difference Percentage*
1	97.0522	97.2347	0.1825	0.1880 %
2	120.2294	120.2996	0.0702	0.0584 %
3	131.7386	131.9486	0.2101	0.1595 %
4	115.6353	115.7055	0.0701	0.0606 %
5	128.8386	128.6324	-0.2062	-0.1600 %
6	143.2694	143.0579	-0.2114	-0.1476 %
7	140.2753	140.5557	0.2805	0.2000 %
8	127.2618	127.0382	-0.2236	-0.1757 %
9	148.6754	148.6815	0.0062	0.0041 %
10	167.2009	167.2729	0.0720	0.0431 %

The last parameter we compared is the RMSSD of the R-R interval. RMSSD measures the square root of the mean squared differences of successive normal-to-normal R-R intervals. It estimates short-term components of HRV and has important clinical significance [53]. Table 6-3 shows the comparison result of the RMSSD from the two devices. Again, the forth column of the table shows the absolute difference between the RMSSD values are very small for each trial as evidenced by the fact that the maximum absolute difference is less than 0.6. The difference percentage shown in the fifth column also manifests very small values. Student-t test has also confirmed that the RMSSD values are not different between the two devices.

Table 6-3 Comparison of the RMSSD parameter (ms) between the R-R interval obtained by HP ECG machine and the R-R interval obtained by our system.

Segment	*HP ECG Signal*	*Our ECG Signal*	*Difference*	*Difference Percentage*
1	104.8550	105.3625	0.5075	0.4840 %
2	133.7174	134.0500	0.3326	0.2487 %
3	143.9766	144.4746	0.4980	0.3459 %
4	106.1838	106.4178	0.2340	0.2204 %
5	138.9509	138.6123	-0.3386	-0.2437 %
6	153.4519	152.8703	-0.5817	-0.3791 %
7	146.1559	146.7493	0.5935	0.4061 %
8	132.8272	132.4808	-0.3464	-0.2608 %
9	119.8913	120.0129	0.1216	0.1015 %
10	133.3119	133.4834	0.1714	0.1286 %

The above preliminary signal validation tests suggest that our system not only can preserve the fidelity of the ECG waveform, but also can faithfully provide accurate results on those clinical important parameters extracted from the ECG signal.

6.6 Summary

Various software tools have been developed for accurate transmission and receival of data. In addition, data display software has been also developed. Specifically, for transmitting data, we had to overcome issues related to A/D conversion, sampling rate, and data packing and temporary storage of data. For the proper receivable of data among many different motes, we used the TDMA scheme which enabled us to collect ECG data from 10 different motes simultaneously at a sampling rate of 200 Hz. Finally, a data display monitoring software was

developed where the objective was to display the signal in a real-time manner. In addition, we developed and incorporated into the display software, several signal processing algorithms that can be used as potential diagnostic parameters. We have also examined and devised an approach to save battery life, and performed experiments to verify that our devised plan of processing the data at the mote significantly curtails the battery usage. While in certain cases, it is necessary to transmit the raw data, but in most cases, it may suffice to transmit only the important processed physiological parameters which in turn significantly save battery life. Finally, we performed experiments to examine the fidelity of the collected ECG signal using our device. This was done by comparing our device against a commercially available device. It was found that both devices provided nearly identical results in terms of both unprocessed and processed data.

Chapter 7 On-board Localized Digital Signal Processing

7.1 Introduction

According to our preliminary tests, the single microcontroller residing in the *Tmote Sky* cannot perform more sophisticated data analysis because the microcontroller has to run *TinyOS* to support the wireless transmission. In order to implement more sophisticated digital signal processing at the device end, an additional microprocessor is certainly needed. Incorporating one more microcontroller will surely increase power consumption. However, this amount of additional power consumption is trivial compared to the power saved by transmitting only a few physiologically relevant processed parameters. With two microcontroller configuration, the microcontroller on the *Tmote Sky* is mainly responsible for the medium access control, data collection, and wireless transmission; while the other microcontroller is solely devoted to real-time signal processing.

The main purpose of integrating another additional microcontroller is to preserve battery power by sending only a few processed parameters. It is important to coordinate the two microcontrollers so that they can work properly and efficiently. In our system, we developed a wireless transmission module and DSP module separately as two independent subsystems. Each subsystem has its own clock source. With this design, the two microcontrollers can be put in "awake" or "sleep" mode without affecting each other. The two subsystems are physically connected via a UART port by hardware and they coordinate via the software residing at either end.

7.2 The Additional Microprocessor and Software Development Tools

7.2.1 The additional microcontroller – MSP430F449

As discussed in the previous section of this dissertation, the localized on-board signal processing solution can dramatically save both battery power and the signal bandwidth. With ever increasing novel advances in the semiconductor industry, high performance DSP chips with low-cost and ultra-low power consumption can be purchased off-the-shelf for the purpose of portable devices development. The MSP430F449 is one of such a chip. It belongs to the TI's MSP430x44x product family.

The TI MSP430x44x mixed signal microcontroller family consists of several devices. Each device features different set of peripherals tailored to different applications. The architecture of the devices is combined with five low-power modes, making them especially suited for battery-powered applications. The MSP430F449 main features include [91]:

- *Low Supply-Voltage Range 1.8V ~ 3.6 V*
- *Five Power Saving Modes*
- *Wake-Up from Standby Mode in less than 6 microsecond*
- *Main Clock Frequency up to 8M*
- *12-Bit A/D Converter with internal reference voltage*
- *16-Bit Timer_B with three or seven capture/compare-with-shadow registers*

- *16-Bit Timer_A with three capture/compare registers*
- *Two serial communication Interface (USART)*
- *Serial Onboard Programming*
- *60KB+256B Flash Memory, 2KB RAM*
- *Built-in Hardware Multiplier*
- *These features make the MSP430F449 much suitable for our system.*
- *Low Supply-Voltage Range 1.8V ~ 3.6 V*

The low supply-voltage range makes it easy to use only two normal AA batteries to power the microcontroller. The wide range of supply-voltage grantees that the microcontroller will still function properly even when the batteries are nearly depleted.

Five Power Saving Modes

This feature allows the microcontroller to be put into a sleep mode, or to shutdown only selected parts of its peripheral circuits to conserve power. In a total sleep mode, the microcontroller only consumes 0.1uA; in a standby mode, it consumes 1.1 uA current, while in an active mode, the power consumption is 280uA.

Wake-Up From a Standby Mode in less than 6 microsecond

In our system, when the additional microcontroller is off-duty, it is either shutdown or put into a sleep mode to save battery power. Thus, it is very critical to put the microcontroller back online when new data appear or a new command is issued to the microcontroller. The MSP430F449 can be activated from a sleep mode inless than 6 microseconds. This feature ensures that the microcontroller can quickly respond to an event or a command.

12-Bit A/D Converter with internal reference voltage

The MSP430F449 has a built-in A/D converter core. The core is connected to 8 external analogy input pins via a multiplexer. Thus, this AD converter supports up to 8 external analog inputs. The A/D module has separate configure and data storage registers for each analog input. So, this facilitates configuration of the A/D converter to sample each channel on a different sampling rate and store the sampled data temporarily without corrupting each other. The conversion accuracy is as high as 12-bit. Accompanying the AD converter is two reference voltages, which are provided by the on-chip voltage regulator. The reference voltage is also software reconfigurable for each analog input. The on-chip voltage regulator provides 1.5V and 2.5V reference voltages.

16-Bit Timer_A (Timer_B)

The on-chip timers are used to control the sampling rate in our system. The Timer_A and Timer_B are 16-bit timer/counter with several capture/compare registers. The clock source of the timer is software configurable. Together with the 16-bit counting register, the timer can be configured to generate a wide range of time interval interrupts with a scalable precision.

Two serial communication Interface (USART)

The two USART interfaces on the MSP430F449 device are designed to function independently and each is configurable. Either of them can be separately configured in UART model or the SPI mode. In our system, both of the two USART are configured to the UART mode. One of them is used to communicate with a pulse oximeter module, MP506, which also has a standard UART interface. The other UART is connected to the microcontroller's UART port on the wireless transmission module. The wireless transmission module issues commands or obtains processed results from the additional microcontroller through this UART connection.

Built-in Hardware Multiplier

The built-in hardware multiplier is very important feature of the MSP430F449. We integrated the MSP430F449 into our system as a co-processor, which is to perform real-time digital signal processing of the data. The real-time DSP analyses imposes heavy demand on the microcontroller's processing capability. For the implementation of DSP algorithms, the multiplication operation is inevitable. For example, to perform an N-point FFT, at least $2N\log_2 N$ multiplications are needed. Multiplication and division are two of the most time-consuming operations. For a single addition or subtraction operation, the time can be as short as one to several CPU clock cycles. However, for either multiplication or division operation, it can take as much as several hundreds of clock cycles. The hardware multiplier is specially designed circuit for performing multiplication operation. It has its own registers. The multiplication is performed by the hardware circuit when such an operation command is triggered. With the support of the hardware multiplier, the multiplication operation can be carried out almost as fast as addition operation.

7.2.2 MSP430 IAR Embedded Workbench® IDE

The IAR Embedded Workbench® IDE is a very powerful Integrated Development Environment (IDE) provided by IAR Company. This programming platform allows the user to develop and manage complete embedded application projects. This IDE allows seamless integration of all necessary tools for the embedded programming tasks such as the following [92]:

- *The highly optimizing MSP430 IAR C/C++ Compiler*
- *The MSP430 IAR Assembler*
- *The versatile IAR XLINK Linker*
- *The IAR XAR Library Builder and the IAR XLIB Librarian*
- *A powerful editor*
- *A project manager*
- *A command line build utility*
- *IAR C-SPY debugger, a state-of-the-art high-level language debugger*

The IAR Embedded Workbench® targets a large number of microprocessors and microcontrollers. The IAR Embedded Workbench® is designed with the concept of "Different Architecture, One Solution". It provides an easy-to-learn and highly efficient development environment with maximum code inheritance capabilities.

The highly optimizing MSP430 IAR C/C++ Compiler

This MSP430 IAR C/C++ Compiler is designed to take advantage of the MSP430 specific features, as well as to provide a state-of-the-art compiler that can handle either C or C++ programming language. Two features of this complier are very important for our system. First, generic and MSP430-specific optimization techniques offered by this compiler can produce a very efficient machine code. Not only the code size is reduced but the execution of the code is also accelerated. The microcontroller we chose for our system has limited memory (60 KB ROM and 2 KB RAM) and a relatively slow CPU clock frequency (up to 8M), thus, both the code size and execution time are critical. Another feature offered by this complier is that the object code it generates can be linked together with assembler routines. It is a must to write some software routines in the assembler language. For example, the start-up code and exit code must be written assembler language because they deal with the lowest level hardware initialization and finalization. For efficiency purpose, some mathematic operation may also be written in the

assembler language. By taking advantage of this feature, the developer has the flexibility to choose either the assembler language for certain portion of the project while C/C++ language for remaining portion of the project.

The IAR XAR Library Builder and the IAR XLIB Librarian

A library is a single file that contains a number of relocatable modules. Each of these modules can be loaded independently from other modules as it is needed. The IAR XAR Library Builder allows the developer to build and publish customized library so that a common software code can be shared by others. With this feature, we can package some of the commonly used routines in our system, for example, signal filtering or ECG R-wave detection algorithm, into a library after these programs have been thoroughly tested. In this way, the library can be shared by other developers, and yet it can be isolated from other modules. This is very important in keeping the consistency of the code and will expedite the development of the project. The IAR XLIB Librarian enables the developer to manipulate the library object files produced by the IAR. With the help of IAR XLIB Librarian, the user can combine other modules into a library file, remove or replace the modules, and change the modules between program and different library types.

IAR C-SPY debugger, a state-of-the-art high-level language debugger

Debugging process is essential in most task of computer programming. While debugging is especially challenging for the embedded systems because they do not have convenient input/output devices that the programmer can rely on. For these reasons, the development of the embedded systems follows the host-target cross-development mode. The host is typically a personal computer that has a programming/debugging Integrated Development Environment (IDE), such as the IAR Embedded Workbench® IDE. The program is written, complied and linked in the IDE at the host end. The target is the embedded system board. Specifically, it refers to the microcontroller together with some other peripheral devices. The host and target are connected by a special device often called "programmer". Fig. 7-1 shows the FET programmer/debugger. The prototype development of the system is based on the MSP-TSPZ100 Target Socket Module, as shown in Fig. 7-2. The cross-debugging is achieved with the support of the programmer and corresponding software, the IAR C-SPY debugger, which works together with the FET programmer for cross-debugging. There is an emulator driver inside the IAR IDE to drive the FET programmer so that the IAR IDE can access the memory and registers of the target microcontroller when the program is being executed on the target. The developer can set breakpoints at the host end, modify the contents of the memory and registers; and inspect the hardware status.

Fig. 7-1. The MSP-FET430UIF Debugger

The IAR C-SPY Debugger also has a driver for the software simulator residing in the host computer. The simulator can simulate part of the behavior and run-time environment of the target device. By debugging the code on the simulator at the host end, the user does not need to write the program to the target device every time. This is useful when the program being debugged has little relevance to the hardware. For example, in our system signal processing algorithms can be debugged on the simulator at the host end. However, the simulator can only partially simulate the target behavior. For example, it cannot simulate the hardware interrupt.

For this type of program, it has to be debugged directly on the target device. In our system, this type of program includes the sampling rate control, AD conversion and UART communication etc.

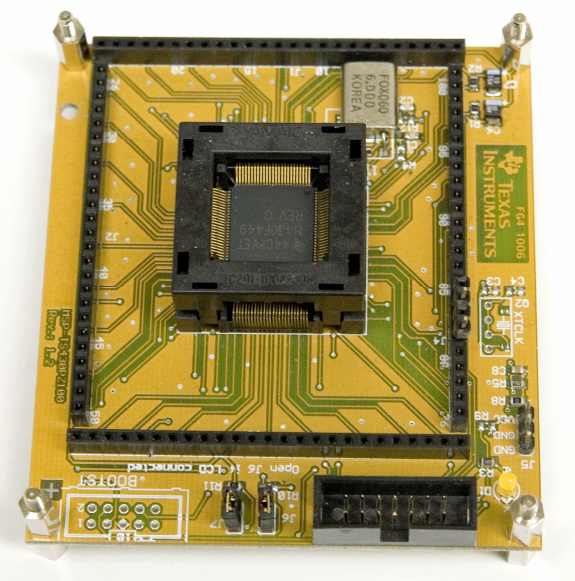


Fig. 7-2. The MSP-TSPZ100 Target Socket Module

7.2.3 The programming language: Embedded C++

C++ is a general-purpose programming language that has both high-level and low-level capabilities. So it is sometimes regarded as a "mid-level" language. C++ has many unique features that C does not have. C++ is an object-oriented programming (OOP) language. It allows the programmer to create classes, which is similar in concept to the structure in C language, but it contains more advanced features. With the introduction of classes in C++, more features are created in C++ such as the function overloading, operator overloading, class inheritance, polymorphism and templates [93, 94]. With the aid of these features, C++ provides safer data protection. Due to the optimization of many C++ compilers, the overhead of introducing these additional features has been minimized. In addition, C++ program is as efficient as its C counterpart.

7.3 Software Development

Embedding the DSP algorithms into a microcontroller needs a supporting software platform that is able to sample the analog signal and manage the embedded DSP algorithms. The supporting software platform has similar functions as a real-time operation system (RTOS). Although there are some RTOSs available for free such as the freeRTOS and velOSity RTOS [95, 96], they are developed as a general purpose RTOS and they typically require several KB RAM to run. For example, the velOSity RTOS requires at least 3 KB RAM memory [96]. The microcontroller we chose, MSP430F449, has only 2 KB RAM. So, it is not possible to run a general purpose RTOS using our microcontroller. Besides, a general purpose RTOS has many additional features that are not needed in our system. These features will introduce more

unnecessary system overhead in our system as they consume additional power and the system's processing capability. Based on these considerations, instead of adopting an available RTOS, we developed a RTOS-like software platform, termed as Software Platform for Digital Signal Processing (SP-DSP), which is specifically tailored to our system.

The principles of SP-DSP development are to keep the system as simple (with the restriction of providing required functions), as specific (to take the advantage of the specific hardware features), and portable as possible (to minimize the re-programming when a hardware upgrade is desired in the future). In keeping these features, the SP-DSP is developed at two separated levels: low-level and high-level, as discussed in detail in 7.3.1 and 7.3.2, respectively.

7.3.1 Low-level implementation: Hardware Abstraction

The low-level programming is essential for any embedded application development. The purpose of the low-level programming is to isolate the high-level application algorithms from the particular hardware platform so that the high-level code is independent of the hardware. In future, if a upgrade of the hardware is needed, only the low-level code should be changed. While the code implementing DSP algorithms at high-level can be directly adopted with minor modifications.

The low-level implementation is essentially to develop drivers for the hardware. The driver abstracts hardware by providing software components with interfaces. The interfaces are called by the high-level code to invoke the functionality provided by the hardware module. Although the detailed implementation of a certain hardware function needs to be modified if the hardware is upgraded, the calling semantics in the high-level code remains the same.

A hardware component is controlled by its own registers. To abstract the hardware component involves the configuration of those registers according to the requirements of an application. It is also desired by the high-level program that the hardware may be reconfigured at run-time, so the low-level driver program needs to provide some interfaces to access those registers.

In the current version of SP-DSP, the following hardware modules have been abstracted: Timer, A/D Converter and UART Port.

The Timer

MSP430F449 has two independent timer circuits: Timer_A3 and Timer_B7. They are both 16-bit timer/counter with multiple capture/compare registers. For example, Timer_A3 has three such registers and Timer_B7 has seven. Each register can be considered as a stand-alone hardware timer. Thus, all together, MSP430F449 can provide 10 independent hardware timers, among which each can be individually started, stopped, or configured into different modes. The Timer_A3 and Timer_B7 share the following common features:

- *Asynchronous 16-bit timer/counter with four operating modes*
- *Selectable and configurable clock source*
- *Three or seven configurable capture/compare registers*
- *Interrupt vector register for fast decoding of all Timer's interrupts*

The abstraction of the timers involves initialization, configuration, and interrupt handling. In current SP-DSP, the timers are restricted to be initialized only once at run-time. The initialization occurs when the program is started. However, the timers can be reconfigured at run-time to specify a new timer expiration interval. This hardware-related operation is invisible from

the high-level programming aspect. The timer hardware modules are abstracted as a `Timer` class in the SP-DSP:

```
class Timer {
      private:
      public:
            void Start(const int* val);
            void Stop();
            void SetCallback(void (*p) ());
}
```

To discriminate the abstracted timer class from the actually hardware timer, I will refer the abstracted timer as "software timer" hereafter. The high-level program can define up to 10 software timers corresponding to the 10 hardware timer registers. Each software timer is internally linked to a distinguished hardware timer by the low-level driver program. From the high-level program point of view, the 10 software timers are exactly identical, and no knowledge about the hardware is required to use these software timers. The following example code shows how to use the abstracted software timer:

```
{
      Timer t1;
      t1.SetCallback(&foo);
      t1.Start(5);
}
```

The statement `Timer t1` defines a software timer `t1`. `t1` is automatically linked to an available hardware timer by the driver program when it is defined. `t1.SetCallback(&foo)` specifies that every time `t1` expires, the routine `foo()` is automatically called. `t1.Start(5)` is the actual code to start the timer. This statement specifies that timer `t1` expires every 5 milliseconds. The high-level program implements the `foo()` routine in response to the timer expiration, for example, to trigger an AD conversion. The semantics of the above call of the software timer will remain the same even the hardware and its driver is upgraded.

The AD Converter

The ADC12 module built-in the MSP430F449 supports fast, 12-bit analog-to-digital conversions. The conversion-and-control buffer allows up to 16 independent ADC samples to be converted and stored without any CPU intervention. Selected ADC12 core features include:

- *Greater than 200 ksps maximum conversion rate*
- *Conversion initiation by software*
- *Software selectable on-chip reference voltage generation (1.5 V or 2.5 V)*
- *Eight individual configurable external input channels*
- *Four selectable conversion modes*
- *Interrupt vector register for fast decoding of 18 ADC interrupts*

The ADC reading is dependent on the reference voltages. The relationship between the reference voltages and the AD reading is given by:

$$N_{ADC} = 4095 \times \frac{V_{in} - V_{R-}}{V_{R+} + V_{R-}}$$

The V_{in} is the analog input. V_{R+} and V_{R-} are reference voltage levels. They are both software programmable. In most applications, the V_{R-} is grounded, and V_{R+} is connected to the

internal reference voltage.

The abstracted ADC class has the following interfaces (only public interfaces are listed):

```
class ADC {
      private:
      public:
            static void Initialize();
            bool Start();
            unsigned char Get();
            void SetCallback(void (*p)());
            bool SelectCH(const unsigned char& ch);
            static bool EnableCore();
            static bool DisableCore();
            static void SelectRef1_5v();
            static void SelectRef2_5v();
}
```

Like the timer circuit, the ADC12 core must be correctly initialized so that it can work properly. The high-level program invokes the initialization by calling `ADC.Initialize()` in the abstracted interface. In this process, the AD conversion clock source is specified. The reference voltage is also selected during initialization. In the current SP-DSP, the internal 1.5 volt is selected as the reference voltage. The reference voltage for AD conversion is of critical importance. A tiny ripple of the reference voltage can result in significant noise in the AD conversion results. The internal reference voltage is provided by an on-chip voltage regulator, which persistently stabilizes the voltage at a constant level. Thus, choosing the internal reference voltage reduces noise in the final AD readings. In addition, the sample and conversion timing is also configured in initialization process. An AD conversion can either be automatically started by Timer_A/Timer_B via internal hardware connections, or can be initiated by setting the `ADC12SC` bit in the control register by a software instruction. In the current SP-DSP, the Timer_A and Timer_B are used as the underlying hardware support for the software timers, so we chose the software triggering method to initiate an AD conversion.

The interrupt generation and handling mechanism of the AD converter is little complicated. The ADC12 core supports up to 8 external input channels. But these channels share a single interrupt vector. In another word, no matter which channel finishes the completion of an AD conversion and generates an interrupt, the same program segment, called Interrupt Service Routine (ISR), is executed. So, the 8 external analog inputs are undistinguishable in terms of interrupt vector because they all trigger the execution of the same ISR. However, the ADC12 core has an `ADC12IV` register, which is used to store the index of the analog channel that generates the current interrupt. In the ISR for the AD interrupt, the `ADC12IV` register is accessed by the driver code to identify which channel generated the current interrupt so that the execution is branched for each channel. This process has been encapsulated in the abstracted `ADC` class. By using the public interface `ADC.SetCallback(void (*p)())`, the high-level program can specify a distinguished interrupt handler pointed by `*p` for each ADC channel; as if each ADC channel had its own interrupt vector and ISR. The benefit of this is to make the high-level programming independent from the hardware architecture. If the hardware is replaced by a new different chip, the high-level code using `ADC` class does not need to be changed.

The typical usage of the abstracted `ADC` class is like the following:

```
{
      ADC ecgADC
```

```
        ecgADC.SetCallback(&ecgADC_foo);
        ecgADC.SelectCH(ADC_A0);

        /*  in timer expiration ISR  */
        {
             ecgADC.Start();
        }

        /*  the ecgADC ISR  */
        void ecgADC_foo() {
             int ecgvalue = ecgADC.Get();
        }
}
```

It is not necessary to explicitly call the `ADC::Initialize()` to initialize the hardware. Call to this routine is automatically made at the time when the first `ADC` object is defined. The call of `ADC::SelectCH()` binds a defined `ADC` object to an external analog input. For example, in the above code, the call `ecgADC.SelectCH(ADC_A0)` selects the A0 pin as the analog input of `ADC` object `ecgADC`. An AD conversion is initiated by calling `ecgADC.Start()` as shown in the example code. The ADC12 core needs a short time period to complete a single conversion. Once a conversion is completed, the ADC12 core generates an interrupt. The interrupt causes the automatic call to the `ecgADC_foo()` routine in the above example code. The conversion result is stored in one of the ADC registers. To read out the data, a call to `ADC::Get()` must be made, as demonstrated in the above `ecgADC_foo()` routine.

Due to the fact that the ADC12 core needs a short time to complete a conversion, there is a potential collision problem for multiple channel conversions. Although the ADC module supports 8 external analog inputs, there is only a single ADC12 core in the module performing the actual conversion. The ADC12 core is connected to the 8 external analog input through a multiplexer directed by the control register. The software selects different input channel by specifying the `CSTARTADDx` bits in the control register. This may cause a collision problem as shown in Fig. 7-3. Suppose the ADC12 core is performing the conversion on input A0, a request of sampling input A1 is imposed by another piece of code, say, the expiration of the sampling timer that controls the sampling rate of A1. If the `CSTARTADDx` bits are changed immediately to connect the ADC12 core to input A1, the conversion on A0 will be interrupted, resulting in a corrupted data. So, a mechanism must be developed to arbitrate the race of the ADC12 and solve the collision problem. A reasonable solution is to hold the request of sampling A1 for a short time, giving the ADC12 core enough time to finish the current conversion on A0. In the current SP-DSP, a queuing technique was employed to solve this collision problem.

A "queue" is a line-up of a series of similar elements. It has the "first in, first out" (FIFO) priority setting. The element firstly pushed into the queue is popped out first to be processed. Such a conversion request queue is maintained by the ADC driver program to avoid conversion collision. The call of `ADC::Start()` causes the conversion request being pushed into the queue instead of immediate change of the `CSTARTADDx` bits. The underlying ADC driver polls the queue every time a conversion is completed to check if there are any pending requests unprocessed. If so, the request is popped out and the ADC12 core is switched to the specified analog input and a new conversion is started. Again, the potential collision is handled by the low-level ADC driver. From the high-level programming point of view, it is safe to assume that the call of `ADC::Start()` initiates a new conversion immediately and all the external analog inputs are independent from each other.

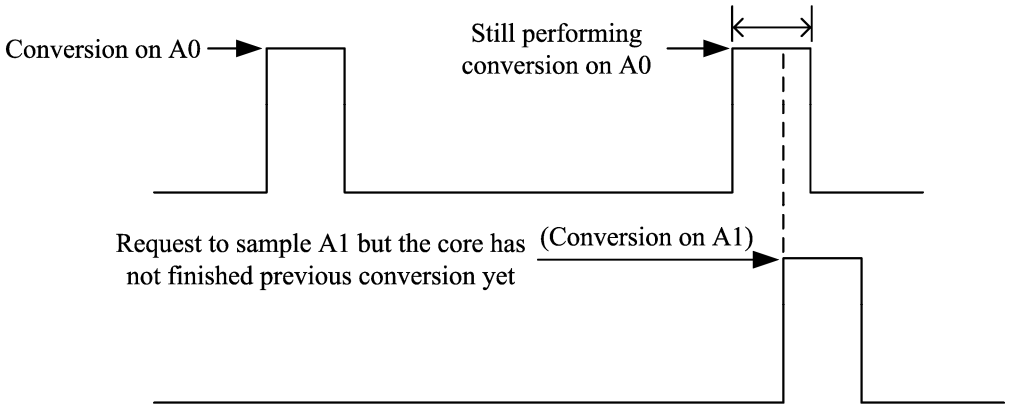


Fig. 7-3. Potential conversion collision when multiple analog inputs are sampled.

The queue technique is also used in the design of the system core to manage data analyses tasks. This is discussed in 7.3.2 and 7.4.4.

USART Port

The physical connection between the additional MSP430F449 and the *Tmote Sky* is established through the USART circuit. The USART circuit at both sides can be either configured in UART mode or in Serial Peripheral Interface (SPI) mode. In current version of SP-DSP, we utilized the USART in UART mode. In the future, if higher data transmission rate between the additional MSP430F449 and the *Tmote Sky* is desired, it is possible to configure the USART port in SPI mode since SPI is much faster. For example, if the system clock is 8 MHz, the SPI can reach a transmission rate as high as 4 Mbps; while UART cannot afford this high rate otherwise it will suffer from unbearable error rate. In *TinyOS* 1.x, the maximum UART baud rate is 262,144 bps, which is only one fifteenth of the maximum SPI speed. However, using SPI mode will slightly increase the power consumption and slightly increase software overhead.

The communication protocol between the *Tmote Sky* and the additional MSP430F449 is C/D TP. Unlike the communication between the wireless receiver and transmitter, where the Active Message protocol has been introduced to facilitate a reliable peer-to-peer transmission, the *Tmote Sky* and the additional MSP430F449 have already been "peer-to-peer" connected by hardware. Thus there is no need to incorporate AM protocol here.

The wire connection between the *Tmote Sky* and the additional MSP430F449 through UART port is illustrated in Fig. 7-4. Only three wires are needed as shown in Fig. 7-4: Rx (Receival), Tx (Transmission) and GND (Ground). The wiring has the "cross" pattern. That is, the Tx terminal of one end should be wired to the Rx terminal of the other end, and vice versa.

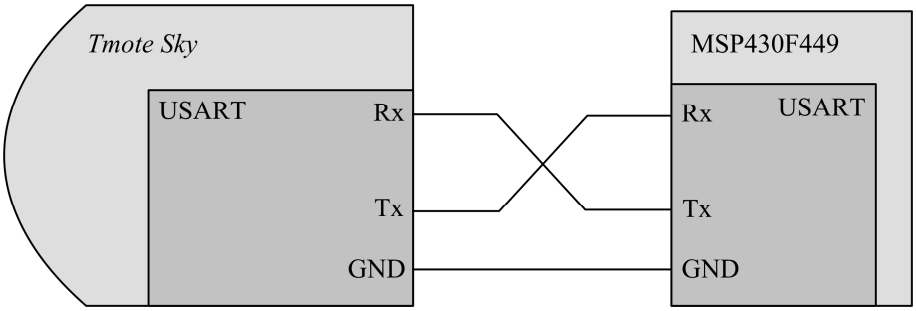


Fig. 7-4. The UART hardware wiring between the *Tmote Sky* and the additional MSP430F449

The abstracted USART interface is the class `UART` in SP-DSP. It provides the following public interfaces that are used by the high-level program:

```
class UART {
      private:
      public:
            bool Bind(const unsigned char& p)
            void SetBaudrate(const int& br);
            void SetRxCallback(void (*p)())
            void Send(const unsigned char& txbyte)
            void SendCDTP(const CDTPMsg& msg)
            unsigned char Receive()
            void SetTxCallback(void (*p)())
            void EnableRx()
            void EnableTx()
            void DisableRx()
            void DisableTx()
}
```

Like other hardware components, the USART also has a bunch of registers to be configured and manipulated to make it work properly. Most of the configuration occurs in the initialization process and is encapsulated in the low-level UART driver program.

In the initialization process, the UART clock source is selected by specifying the `SSELx` bits in the USART transmit control register. Selecting an accurate clock source is of crucial importance for UART communication because the baud rate generator relies heavily on the clock to generate an accurate baud rate. Mismatched baud rates between two UART talkers can lead to a complete failure of the UART communication. In current version of SP-DSP, we chose the external 8 MHz crystal as the UART clock source because it is more accurate and reliable than the internal crystal oscillator. In order to configure the baud rate generator to get a desired baud rate, appropriate values should be written to the baud rate control register and the modulation control register. Determining the values for these register is a dynamic and complicate process. Some microcontroller manufacturer provides the developer a look-up table to determine the these values. There are also some online programs available to calculate this value. For the detail of how to determine the modulation register value, the reader is referred to [97]. In our system, the value is calculated by the online program [98].

The typical usage of the abstracted `UART` class starts with the definition of a `UART` object followed by the call to `UART::Bind()`, which connects the `UART` object to an actual USART hardware module. In the MSP430F449, there are two USART modules. Thus, up to two `UART` objects can be defined in the high-level program. After successfully binding the defined UART object to the USART port, consequent call of `UART::SetBaudrate()` specifies the baud rate of the USART port. The argument passed to this interface is the desired baud rate, measured in bit per second (bps). The baud rate cannot be an arbitrary value. In the current SP-DSP version, only the *TinyOS* compatible baud rates are supported and they are: 9,600 bps, 19,200 bps, 38,400 bps, and 262,144 bps.

Except the baud rate, which can be alternated by the high-level program at run-time, other UART parameters are specified and fixed in the underlying initialization code. These parameters include stop bit, parity check bit and character format. Currently, these parameters are initialized as 1 stop bit, no parity check and 8-bit character.

The ISRs for the UART receive and transmission are encapsulated in `UART::SetRxCallback()` and `UART::SetTxCallback()`, respectively. Call of these

interfaces specifies the user customized routines that are automatically executed when a valid byte of data is received or transmitted. The `UART::Send()` function takes a byte value and sends it out through the USART port. If the USART circuit is not busy, the value is written into the transmit buffer register and sent out at once. The counter-part of the `UART::Send()` is the `UART::Receive()`. If a valid byte data is received by the USART hardware, the received data is stored in the receive buffer register and an interrupt is generated (if the receival interrupt is enabled). Calling of the `UART::Receive()` retrieves the data from the receive buffer register. The high-level program must make this call in the ISR (specified by the call of `UART::SetRxCallback()`) to read out the received data so that the overflow of the USART port is avoided.

In order to expedite the high-level programming, the abstracted `UART` class also provides an auxiliary interface to send a C/D TP message over a UART connection. A call to `UART::SendCDTP()` initiates the transmission of a C/D TP message through the UART connection at the preset baud rate. The C/D TP message is specified by the function argument of this interface. For the high-level program, there is no need to specify the ISR if a C/D TP message is sent. The interrupt service is carried out by the underlying driver code. The `UART::SendCDTP()` is implemented by a Finite State Machine. So, a simple call of `UART::SendCDTP()` reliably sends out the C/D TP message.

7.3.2 High-level implementation: The System Software Components

The high-level implementation of the SP-DSP is designed to be independent from the low-level implementation. The high-level implementation is logically isolated from its low-level counterpart so that any changes in the low-level driver code do not affect the high-level system software components. By this design, once the system components remain stable and unchanged no matter what modifications are going to be made on the low-level driver due to hardware upgrade. It is possible, however, to expand the functionality of the software system compoenents by adding more features in the future.

In this section, the available functions of the current system components are discussed. The data structure and programming techniques to implement these functions are also discussed.

The system components provide some implemented classes for the convenience of high-level programming. The basic framework for the execution of multiple DSP tasks is also defined and implemented. The currently available system components are: `Buffer`, `IntBuffer`, `DBuffer`, `Queue`, `Task`, `CDTPMsg`, and `FFT`.

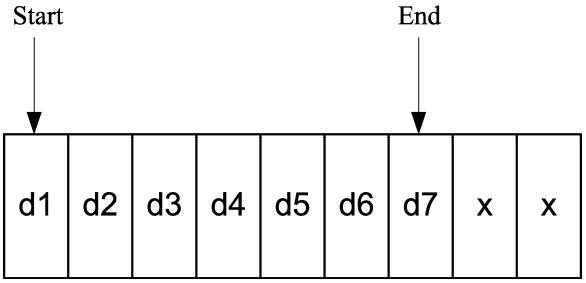
`Buffer`

Buffering is a technique that has been widely used in computer programming. It is so fundamental that not even a simple piece of code can be written and executed without it. The basic idea of buffering is to temporally store whatever data in a small piece of memory so that the data can be processed shortly after. The “processing” here is not limited to a specific kind of mathematic processing. It can be any manipulations performed on the data; e.g. to display the data; to send the data over UART; or to filter the data. The buffering technique plays a more important role in real time systems because real time systems are more prone to event interruption. When the execution flow is toggled between the tasks and events, buffering is certain needed to protect data. Besides, real time systems have deadlines for every task. This increases the complexity of real time systems and the possibility of data race. Buffering technique is one of the solutions of the data race problem.

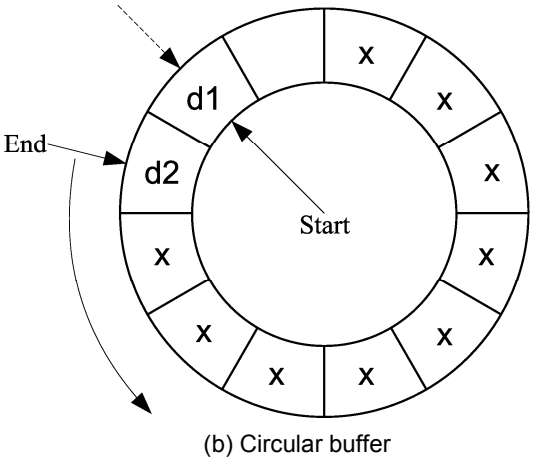
The buffering technique has been utilized in the monitor software for smooth and jutter-free display of ECG signal as described in 6.4.2. In general, buffer can be cataloged into two types: linear buffer and circular buffer (also called ring buffer). In linear buffer, the elements are logically sequenced according to their physical positions. The index pointer linearly increases or decreases, as shown in Fig. 7-5(a). The `Start` pointer indicates the first element in the buffer. Both the logical and physical positions of the `Start` pointer are fixed at the creation of the buffer. The `End` pointer points to the last element in the buffer and its value is logically not less than the `Start` pointer. The only scenario that these two pointers have the same value is when the buffer is created and no data has been written into it, yet. Since the `End` pointer can only either linearly increases or decreases, there is a potential overflow if the `End` pointer goes out the physical scope of the buffer. So, every time a new data is written to or read out of the linear buffer, boundary check should be carried out to make sure the operation is legal.

The linear buffer is used thoroughly in the SP-DSP. For example, it is used for data packing and message synthesis.

The circular buffer is conceptually different from its linear counterpart. Although the data is stored in linear memory as it is in the linear buffer, the way to access the data in a circular buffer is different from that of the linear buffer. As shown in Fig. 7-5(b), the `End` pointer increases circularly. When it reaches the end boundary of the buffer memory, the `End` pointer is rolled back to the beginning address of the buffer memory. But logically the buffer is not overflowed in this case. The rolling back is treated as a seamlessly legal operation of "buffer-grow". From the user's point of view, the `End` pointer could always keep increase and was never bounded by a physical memory address. So, there is no physical overflow of the circular buffer. However, there is a logical overflow of the circular buffer. The `End` pointer is logically bounded by the `Start` pointer. If the `End` pointer hits the `Start` pointer after an increase, the circular buffer is logically said "overflow". In another word, if the `End` pointer and `Start` pointer point to the same element, the circular buffer overflows because the new data will overwrite the old data pointed by the `Start` pointer. This causes a problem for the circular buffer. At the creation of the buffer, or at the reset of the `Start` and `End` pointers, before any data is written to the buffer, both of the pointers point to the same position in the circular buffer. Obviously, the buffer is not overflowed in this case even the two pointers point to the same position. So, how do we differentiae this creation/reset scenario from an overflow? Multiple solutions have been proposed. First, we can maintain a counter variable for the circular buffer, which counts how many data has been written to the buffer. At reset of the two pointers, the counter is set zero since no data in the buffer at all; while in the overflow case, the counter should have a non-zero value. By checking the value of the counter variable, we can tell if the



(a) Linear buffer



(b) Circular buffer

Fig. 7-5. Linear buffer and Circular buffer

buffer is overflowed or not. Another solution is to keep at lease on free space in the circular buffer. At reset, the two pointers do not point to the same element, but to two adjacent ones. Thus, the only scenario for the two pointers to point to the same position is the overflow condition. The `Start` pointer of a circular buffer is typically an indicator a "read" operation. This operation fetches the stored data out of the buffer for processing or transmission. On the other hand, the `End` pointer is often denoted as a "write" operation. The new data is written to the "end" position by some data collecting subroutine. For example, at the transmitter end in our system, such data collection module is the ISR of the AD converter (see 7.3.1). In the monitor software, two circular buffers are used: one is for receiving data from the receiver via a USB port and the other is used to buffer the ECG data for real-time display. Circular buffer is very important in a real-time data collection system since there is no way to know *a prior* how many data will be collected. Using linear buffer in such systems will eventually encounter the "out of memory" problem and crash the system.

The SP-DSP provides a `Buffer` class that can be used either as linear buffer or circular buffer. The interfaces it provides are:

```
class Buffer {
      private: ……
      public:
            unsigned short Read(unsigned char *p, const
            unsigned short& n);
            unsigned short Write(unsigned char *p, const
            unsigned short& n);
            unsigned short CopyTo(Buffer& destbuf);
            unsigned short CopyFrom(Buffer& srcbuf);
            unsigned short Size() const;
            bool SetSize(const unsigned short& sz)
            void SetPos(const unsigned short& pos)
            bool ReadByte(unsigned char& b);
            bool WriteByte(const unsigned char& b);
            const unsigned short& WritePos() const;
            const unsigned short& WritePos(const unsigned
            short& pos);
            const unsigned short& ReadPos() const;
            const unsigned short& ReadPos(const unsigned
            short& pos);
}
```

As shown by the code, the `Buffer` class manipulates the data byte by byte. This is especially useful to implement the transmission protocols since all these protocols are defined on a byte basis (see 5.2.3).

`IntBuffer`

The class `IntBuffer` is designed the same way as the `Buffer` class. It also can be used as linear and circular buffer. The difference is the data type they are designed for. The `IntBuffer` class deals with integer data; while the `Buffer` class works on byte data. The actual size of an integer is hardware dependent. For MSP430F449, an integer occupies 2 bytes memory. The benefit of using `IntBuffer` class instead of `Buffer` is that the read and write operation is faster when an integer number is pushed in or popped out of the buffer. In our system, the AD convertor has 12-bit precision. This requires an integer (16-bit) type of buffer to store the result. If the `Buffer` class were used, the program had to write the high byte and low

byte of the sampled data separately using two writing cycles. When the data is read out of the buffer, it puts more obstacles in the process. First, the low byte is read out followed by reading out the high byte. Then, the two bytes need to be stitched together to get the original value. This process not only involves two reading cycles, but also introduces the overhead of stitching two bytes together. Since the MSP430F449 is a 16-bit microcontroller, it can read/write a 16-bit data in a single instruction cycle. The class `IntBuffer` takes advantage of this fact and expedites the operation of integer numbers.

`DBuffer`

`DBuffer` stands for Double Buffer. The `DBuffer` class is designed as:

```
class DBuffer {
      Buffer *pbuf[2];
      unsigned char indexw, indexr;
      unsigned short size;

public:
      void Swap();
      bool ReadByte(unsigned char& b);
      bool WriteByte(const unsigned& b);
      Buffer* ReadBuffer();
      Buffer* WriteBuffer();
}
```

As shown from the code, the `DBuffer` class has a 2-element `Buffer` array. One of them is assigned as a "read-only" buffer and the other one is "write-only" buffer. The double buffer design separates the read and write operation to avoid data race condition in a real time system. For example, in our system, the sampling module collects the data from analog input periodically. Once a certain amount of data is collected, or a command requiring data transmission is received, the sending module packs the data and sends them out. During this process, the sampling procedure should be still working normally to avoid loss of data. In this case, it is possible that the packing and sending process may be preempted by the sampling module when the reading of data is in progress. If a single buffer were used to temporary store the collected data, both the sending module and the sampling module would try to access the same buffer. This asynchronous access to the same resource is the origin of data race problem. The preempted process needs the data remain untouched until it is resumed to execute from being preempted; while the preempting processing has the potential to alternate the data without letting the preempted process be aware of this. If the resource is somehow changed by the preempting process, the preempted process, when resumed, has a wrong assumption of integrity of the resource. This often causes a strange and unpredictable system behavior at run-time and it is very difficult to locate such kind of error. The double buffer technique is used here to avoid the data race problem introduced by asynchronous read/write operation. The sampling module is restricted to access the "write-only" buffer; while the sending module can only access the "read-only" buffer. In this way, even if the sending process is preempted by the sampling process, there is not data race because the two processes work on different buffers. The class members `indexw` and `indexr` are indices indicating which buffer is the read-only buffer and which is the write-only buffer. The two pointers are mutex to each other. At any given time, the relationship between these two pointers is: `indexw + indexr = 1`. The read-only and write-only buffers are not fixed to a particular one, but are swapped from time to time. The swap occurs when the write-only buffer is full. This is accomplished by an atomic (cannot be preempted by any process) call of `DBuffer::Swap()`. During the process of swap, the values

of the indexw and indexr indices are exchanged. The provided public interfaces DBuffer::ReadByte() and DBuffer::WriteByte() reads one data from the read-only buffer and writes one data to the write-only buffer, respectively. While the DBuffer::ReadBuffer() and DBuffer::WriteBuffer() return a memory pointer to the read-only and write-only buffer, respectively.

Queue

Queue is another widely used data structure in computer sciences. It is a linear collection of many similar elements. The elements in a queue are sequenced in the First-In-First-Out order when they are added into the queue. The operations on a queue involve adding a new element only at the rear position of the queue and removing an old element only from the head position of the queue. By this configuration, the queue data structure naturally has a fixed linear priority setting. The element at the head position, which is also the element that has been staying in the queue for the longest time, has the highest priority. The priority linearly decreases from the head of the queue to the tail. Those typical operations on a queue like to add an element at rear terminal; to remove an element from the head; and to delete an arbitrary element etc, are provided by the public interfaces of the Queue class:

```
class Queue {
private:
public:
      bool Initialize(const unsigned char& sz)
      unsigned char Size();
      unsigned char Head();
      unsigned char Tail();
      bool Push (void *p);
      void* Pop();
      void Delete(void *p)
      unsigned char& Count();
}
```

The Queue::Initialize() interface is called when a queue object is created. This procedure sets the maximum size of the queue. It determines how many elements can be store in the queue. The Queue::Push() operation adds a new element to the rear position and Queue::Pop() remove the head element of the queue. The removed element is served by other module or subject to some processing depending on what the queue is created for. The Queue::Count() interface allows other module to inquire how many elements are currently in the queue.

The queue has been used in the AD convertor driver code to avoid the potential problem of conversion collision (see 7.3.1). It is also used in the main loop of SP-DSP to coordinate the execution of multiple tasks as discussed later in this chapter.

Task

The SP-DSP uses the “task + event” concurrence model as many real time operating systems do. A task is the basic execution unit in the main loop. Tasks follow the “run to completion” pattern. A task cannot be preempted by another task, but can be preempted by any interrupts. The class Task is an abstracted code execution model. The data analyses algorithms embedded in the SP-DSP should be encapsulated in and executed as tasks. The design of the Task class is more complicated than any other classes. The type of the routine executed as a task varies a lot depending on the application. It can be a global regular function or a sealed member function of different classes. To accommodate this, we utilized the template feature

supported by the Extended Embedded C++ to design the Task class. The template feature provides the flexibility to seamlessly deal with many different types of classes. By designing the Task class in this way, executing different types of routines can be initiated by calling the same standard interfaces of the Task class. The polymorphism feature of C++ guarantees the proper routine is executed. Designing the Task class in this way requires the routine to be a member function of a certain class. If a global function needs to be executed as a task, it should be encapsulated in a dummy class as a member function. The declaration of the Task class is as the following:

```
class BasicTask {
      public:
            virtual void Execute() = 0;
            virtual ~BasicTask() {};
};

template <class T>
class Task:public BasicTask {
            unsigned char state;
            T *pT;
            void (T::*pFunc)();
      public:
            Task();
            Task(void (T::*pf)());
            Task(const Task& tk);
            ~Task();
            void Set(T *p);
            void Set(T t);
            void Set(T *p, void(T::*pf)());
            void Set(T t, void(T::*p)());
            void Execute();
            void State(const unsigned char& st);
            const unsigned char& State();
};
```

The Task class is inherited from the BasicTask class. The execution of a certain routine is started by calling BasicTask::Execute(). The BasicTask::Execute() is declared as a pure virtual function to mandatorily require the inherited Task class to overload it. This prevents the mistaken call of BasicTask::Execute() before it is instantiated. Making the BasicTask::Execute() virtual takes the advantage of the C++ polymorphism feature so that the calling semantics of any task is identical but the actual execution of the task is automatically differentiated by the C++ compiler within the scope of Task class. The call of the BasicTask::Execute() is implicitly and automatically directed to the call of the actual routine pointed by the function pointer Task::pT. The overloaded member function Task::Set() sets the value of the function pointer Task::pT to specify the actual routine of the task.

When the SP-DSP is started, a task queue is created. In the main loop, a piece of code continuously polls the task queue to see if any unexecuted tasks remaining in the queue. If so, the task at the queue head position is popped out and executed; if not, the program simple continues to the next start of the loop and polls the task queue again. The following code is a simplified version of the main loop, showing how the task queue is polled and a task is executed:

```
Queue TaskQ;
BasicTask *pCurrentTask = 0;

while(1) {
      if(TaskQ.Count()) {
            if(pCurrentTask = (BasicTask*)TaskQ.Pop()){
                  pCurrentTask->Execute();
      }
}
```

The task queue is polled by the statement `if(TaskQ.Count())`. If the return value is zero, the `while(1)` is executed and the `if(TaskQ.Count())` is executed again. If the return value is not zero, it indicates there are some unexecuted tasks in the queue. So the task that has the highest priority is popped out at the call of `pCurrentTask = (BasicTask*) TaskQ.Pop()`, and is executed right away at the call of `pCurrentTask->Execute()`.

`CDTPMsg`

To facilitate the successive software development at the application level, the `CDTPMsg` class was designed. The `CDTPMsg` class integrates all the elements of a C/D TP message and provides public interfaces to manipulate these elements such as setting the C/D TP command ID of a message; configuring the CSW field of a C/D TP message, and modifying the data field of the C/D TP message. The declaration of the `CDTPMsg` class is:

```
class CDTPMsg{
      unsigned char  nId;
      Buffer* pData;
      unsigned short wCSW;
      public:
            CDTPMsg(): pData(0), nId(0), wCSW(0) {};
            CDTPMsg(Buffer& buf):pData(0), nId(0), wCSW(0);
            ~CDTPMsg() {};
            CDTPMsg& operator=(const CDTPMsg& src);
            unsigned char CDTP_ID(const unsigned char& id);
            unsigned char CDTP_ID() const unsigned short
            CDTP_CSW(const unsigned short& csw);
            unsigned short CDTP_CSW();
            const unsigned char Length();
            bool SetSize(const unsigned short& sz);
            bool ReadByte(unsigned char& b) const;
            bool WriteByte(const unsigned& b);
            unsigned short WritePos() const;
            unsigned short WritePos(const unsigned& pos);
            unsigned short ReadPos() const;
            unsigned short ReadPos(const unsigned& pos);
            Buffer* SetBuffer(Buffer& buf);
            Buffer* SetBuffer(Buffer* pbuf);
}
```

The detail functionality of the interfaces is described in Appendix II.

`FFT`

FFT is a very common tool in digital signal processing field. In our system, many algorithms have FFT involved. For example, the calculation of the power spectrum density of the heart rate series; the estimation of coherence function between tow signals; and the extraction of respiratory rate from photoplethysmography (PPG). Based on the consideration that the real-time FFT may be performed on different signals for frequency related analysis, it is of great convenience to encapsulate the FFT algorithm in a class so that the common code can be shared by multiple FFT analyses instances. Besides, the task mechanism in SP-DSP also requires that a routine to be posted as a task must be a member function of a certain class as described in the above section. The declaration of the `FFT` class is quite straightforward:

```
#define NFFT  128

class FFT {
      private:
            short Re[NFFT];
            short Im[NFFT];
            void DoFFT();

      public:
            void SetValue(short* data, int n);
            Task<FFT> *FFTTask()
};
```

The `FFT` class has two private member data `Re[NFFT]` and `Im[NFFT]`. They are used to store the real and imaginary portion of the FFT result, respectively. They are both initialized to zeros when an `FFT` instance is created. The public member function `FFT::SetValue()` copies the signal data to `FFT::Re[NFFT]` before the call of `FFT::DoFFT()`, which starts the calculation of FFT. Since the `FFT::DoFFT()` is declared as a private member function, the call of `FFT::DoFFT()` must be made inside the `FFT` class. The `FFT` class provides a public member function, `FFT::Task<FFT> *FFTTask()`, to export a task object. This task object can be pushed into the task queue and popped out to execute when it moves to the head position of the queue. This implementation connects the specific execution of FFT, `FFT::DoFFT()`, to standardized call of an abstracted task: `BasicTask::Execute()`. At this point, the FFT, like many other digital signal processing algorithms, has the uniform calling semantics: `BasicTask::Execute()`. Making the calling semantics uniform for every DSP algorithm is mandatory to facilitate the usage of the task queue mechanism.

The actual code calculating FFT is implemented in `FFT::DoFFT()`. In embedded systems, implementation of FFT algorithm is quite different from the implementation of FFT on other hardware platform like PCs. This difference is discussed in detail in 7.4.2 and the actual code is listed in Appendix III.

7.4 Embedding Real-Time DSP Algorithms into the Microcontroller

To embed DSP algorithms into the microcontroller is the primary objective of the development of SP-DSP. As shown in the introduction and 6.5.1, transmitting only a few processed parameters significantly saves power and bandwidth. Currently, the system has only one channel ECG signal, so the following discussion of signal processing is mainly based on ECG signal. The real-time ECG processing includes low-passing filtering and R-wave detection. However, the principles of embedding DSP algorithms into the microprocessor are similar for any

other DSP algorithms. These principles involve the considerations of execution time, calculation precision, and efficient memory usage. In this section, the embedded R-wave detection algorithm and FFT are discussed to show how to embed a DSP algorithm into the microcontroller.

7.4.1 Embedding real-time R-wave detection algorithm

Since most of the algorithms (e.g. the algorithms developed in the first part of this dissertation, as well as other algorithms, like Principle Dynamic Mode (PDM) analysis [57]) deal with the heart rate (HR) series, the basic processing on raw ECG data is to extract R-R interval. Thus, R-wave detection is one of the basic embedded DSP algorithms in our system. Although, many R-wave detection algorithms have been proposed [99-104], we have implemented a real-time R-wave detection algorithm similar to the most popular one that was developed by Pan etc. in 1985 [105]. The preprocessing is similar to that of the algorithm in reference [105], but our algorithm looks into the instantaneous variance of the derivative signal; unlike the algorithm in reference [105] , which just uses the square of the derivative signal. By taking the square of the mean out, our method preserves the same effective dynamic range as the method in reference [105] but has a smaller absolute value range. This is very important to prevent the results from overflow since the algorithm will be implemented on a 16-bit hardware platform. Using double precision or long integer type will certainly provide larger absolute value range, but this will also significantly increase the consumption of memory, power, and calculation time. The R-wave detection algorithm is a point-based algorithm, which means it is executed every time a new data is sampled, its effectiveness is as important as its correctness. We have optimized the R-wave detection algorithm to increase its effectiveness without sacrificing its performance. For the optimization in detail, please see 7.4.3. After being optimized, the typical execution time of the R-wave detection algorithm is less than 0.1 milliseconds and the longest execution time is less than 0.32 milliseconds (see 7.4.4 for detail).

The idea of the R-wave detection algorithm is based on the fact that R-wave exhibits high frequency characteristic and has a local maximum value, together with a large local slope. So, theoretically the first derivative of the ECG signal should have 0 values at the point corresponding to R-peak. Instead of detecting R-peak directly as being done in reference [105], we first detect those nearby regions surrounding 0 values on the first derivative signal as a rough estimation of the location of R-peaks. This step is archived by comparing the instantaneous variance of the derivative signal to an adaptive threshold. Typically, this step will restrict the position of the R-peak within a range of around 3 ~ 7 points. The R-peak position is then refined by finding the local extreme value of the ECG signal among these points. The algorithm is roughly depicted in Fig. 7-6. The shadow area is the 0-region determined by the threshold of the instantaneous variance.

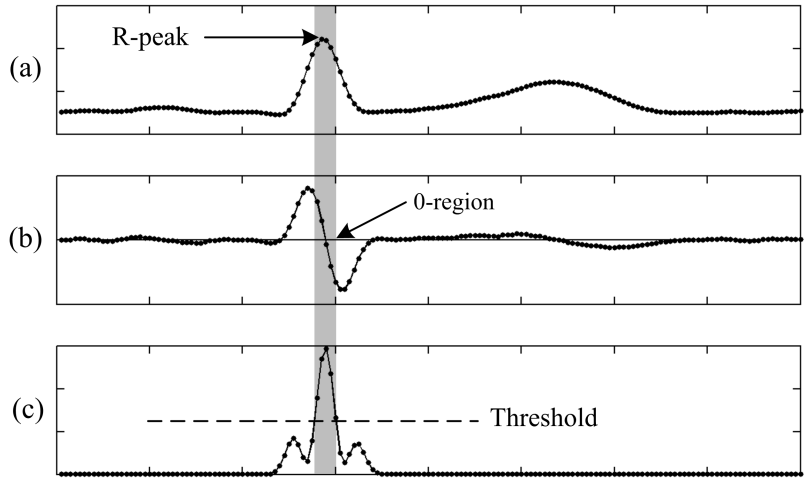


Fig. 7-6. Real-time R-wave detection. (a) low-pass filtered ECG signal. Shadow area covers the R-peak; (b) first derivative signal. Shadow area is the 0-region; and (c) the instantaneous variance. Shadow area is determined by the threshold.

The first step of processing ECG signal is low-pass filtering to eliminate the high frequency noise. We adopted the low-pass filter proposed in reference [105] into our system, but with the modification of canceling out the $z=1$ zero and pole to ensure the low-pass filter has minimum phase. The cancelation facilitates the optimization of the code, as discussed in 7.4.3 later. Besides, the original implementation of the low-pass filter has a 12-point delay as specified in reference [105]; while our version of the filter only has 6-point delay. The modified low-pass filter is given by the following equation:

$$y(n)=\sum_{i=0}^{10} h(i)x(n-i), \quad h=[1,2,3,4,5,6,5,4,3,2,1] \tag{7-1}$$

The cutoff frequency of the filter is 11 Hz and the gain is 36. Shown in Fig. 7-6(a) is the filtered ECG signal.

To calculate the first derivative, a simple 2-point difference is used to approximate the differential:

$$\Delta y(n)=y(n)-y(n-1) \tag{7-2}$$

Fig. 7-6(b) is the first derivative signal calculated from the filtered ECG signal. As shown in the figure, the derivative signal changes abruptly from a maximum positive value to a minimum negative value, intercepting time axis (0 values) at the position of R-peak. This feature can be quantified and detected by calculating the local variance of the derivative signal within a moving window (abrupt change from a maximum value to a minimum value is best evidenced by a huge local variance of that short segment). The optimal window width is determined by the sampling rate. In our system, the sampling rate is 200 Hz, and we found that a window width of 7 is capable to catch the abrupt increase of the instantaneous variance, as shown in Fig. 7-6(c).

In the embedded R-wave detection algorithm, there two thresholds involved: threshold TH_s to detect the R-peak and threshold TH_n to detect the presence of a valid signal. The threshold TH_n reflects the energy level of the system noise. Only the energy of the signal is greater than the noise energy, a valid signal is assumed. The value of TH_n is fixed after the

system is calibrated. Threshold TH_s, however, is adaptive in the detection process. Once a valid signal is detected, the algorithm uses 2 ~ 3 seconds of the signal to initialize TH_s. The threshold is updated every time an R-peak is detected to follow the signal amplitude variation, which may be due to respiration or body movements.

The R-wave detection procedure can be implemented by a Finite State Machine (FSM). The state diagram of the R-wave detection FSM is depicted in Fig. 7-7.

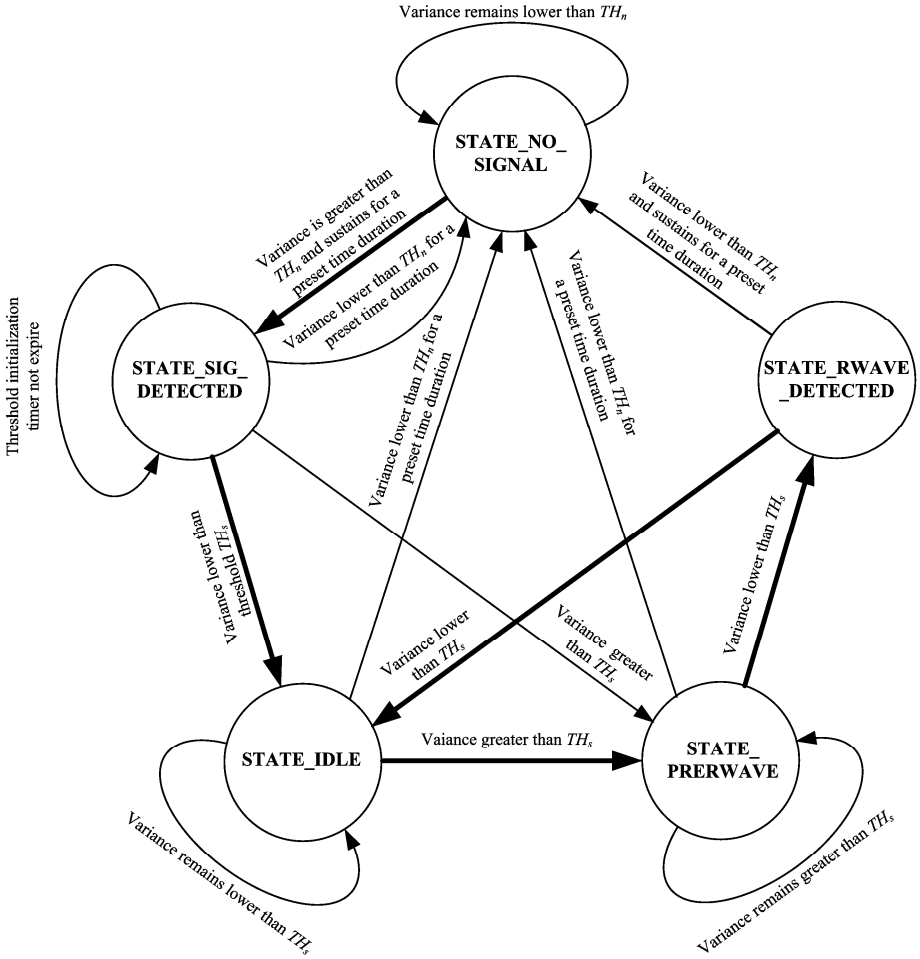


Fig. 7-7. Finite State Machine of the real-time R-wave detection algorithm

The meaning of the states and the transition conditions are illustrated briefly in the following.

`STATE_NO_SIGNAL`: The energy of the signal is lower than the threshold TH_n indicating these is no valid ECG signal on the analog input. This is the initial state of the R-wave detection FSM. Every time the system is started or reset, the FSM is put on this state. This state will also be entered from any other states if the energy of the signal on the analog input pin is persistently lower than the noise threshold TH_n for a preset time (3 seconds).

`STATE_SIG_DETECTED`: This state is a temporary state. When the energy of the signal is higher than the noise threshold, the FSM is switched to this state. The FSM stays on this state for 2 ~ 3 seconds. During this time, the FSM assumes the signal on the analog input is a valid ECG signal and it uses the first 2 ~ 3-seconds ECG signal to initialize the adaptive threshold

TH_s. After the initialization of the threshold, the current instantaneous variance is compared to the threshold. Depending on the comparison result, the FSM either enters the STATE_IDLE state (less than the threshold TH_s), or the STATE_PRERWAVE state (greater than the threshold TH_s).

STATE_IDLE: This state denotes the period with presence of a valid ECG signal but not including the QRS complex duration. During this period, the instantaneous variance is not only less than the threshold TH_s, but also less than the noise threshold TH_n. Obviously, this state should be distinguished from the STATE_NO_SIGNAL although the instantaneous variance is less than the noise threshold TH_n, and it is referred as STATE_IDLE.

STATE_PRERWAVE: With the presence of a valid ECG signal, if the instantaneous variance is greater than the threshold TH_s, the FSM is switched to this state. Within the shadowed period shown in Fig. 7-6(c), the instantaneous variance is persistently greater than the threshold TH_s. Thus, the FSM remains on the STATE_PRERWAVE state during this period. Falling below the threshold triggers the exit of this state. Before the FSM leaving this state, the algorithm searches back in the shadowed duration to locate the exact position of the R-peak. After that, the FSM is switched to STATE_RWAVE_DETECTED state.

STATE_RWAVE_DETECTED: This state is also a transition state. In this state, the algorithm updates the threshold by taking into account the instantaneous variance value of the detected R-peak. The new threshold is set as half of the averaged value over the past five R-peaks. The FSM exits the STATE_RWAVE_DETECTED state after updating the threshold and typically enters the STATE_IDLE state. These three states: STATE_IDLE, STATE_PRERWAVE, and STATE_RWAVE_DETECTED form a typical detection cycle of the R-wave detection algorithm.

7.4.2 Embedding real-time Fast Fourier Transform (FFT)

Surprisingly, the general idea of FFT was first discovered by the famous German mathematician and scientist Gauss around year 1805 but he did not ever publish it. The FFT was later rediscovered by Cooley and Tukey in their independent research in 1965. When Cooley and Tukey published their break-though paper in 1965 [106], the fast way to calculate Discrete Fourier Transform (DFT) was suddenly made world-widely aware. Tremendous calculation time has been saved by using FFT, which kicked away the obstacle preventing DFT being applied to many fields as seen today.

The general idea of FFT is to recursively break down a DFT of any composite size $N = N_1 N_2$ into many smaller DFTs of sizes N_1 and N_2. Specifically, the most known use of this idea is to factorize the transformation into two pieces of $N/2$ in each stage. The factorization can be repeated until each of the small segments has a length of 2-point, 4-point, or 8-point etc. This factorization method is often referred as "butterfly algorithm". Depending on the segment size, the corresponding FFT algorithm is called radix-2, radix-4, or radix-8 FFT. The factorization reduces the number of multiplications from the order of $O\left(N^2\right)$ down to the order of $O\left(N\right)$. Despite the slight difference of calculation time among different radix-x FFT, the butterfly algorithm generally has an approximate linear calculation complexity of $O\left(N \log_2 N\right)$. The detail of butterfly FFT algorithm can be found almost in any digital signal processing books.

Implementation of FFT in an embedded system is worth of careful investigation. Different ways to implement it can lead to significant different performance regarding the execution time, the needed storage memory and calculation accuracy. In this section, how the FFT was implemented in our system is discussed. Since our system is designed for low-power consumption, real-time digital signal processing with limited system resources (2K RAM) and vital parameters monitoring, implementing FFT in such systems imposes specific requirements regarding execution time, memory usage, and numerical accuracy.

Most of the calculation time of FFT algorithm is spent on multiplications. Although, the microprocessor, MSP430F449, does have a hardware multiplier, the hardware multiplier is designed for integer multiplication. It still takes long time for the hardware multiplier to perform double type multiplication. Besides, more memory is also needed to hold a double value. For example, in MSP430F449, 2-byte is used to store an integer, but 4 bytes or more are used to store a double value. If the FFT is implemented based on double type, both the execution time and memory storage will be greatly increased and it will not be affordable for our system. A better choice is to implement the FFT based on integer type by taking advantage of the microcontroller's hardware multiplier.

The idea of implementing FFT based on integer type is to digitize the values of data and the FFT basis functions $(\cos\omega t, \sin\omega t)$. As well known, the range of either $\cos\omega t$ or $\sin\omega t$ is $[-1,1]$. To hold any value within this range, a float-point type is certainly needed. Digitization of this range involves dividing this range into many much smaller ranges (called "layers" hereafter), and the value is represented by the index of the layer which the value falls into. As an example, a sine wave before and after digitization is shown in Fig. 7-8. Left panel of Fig. 7-8 is the original continues sine wave, and the right panel is the digitized sine wave. The range of $[-1,1]$ is divided into 256 layers in this example. The index of each layer is defined as an integer within $[-127,127]$. By this definition, the conversion between the real double value and the corresponding integer index value is given by:

$$\begin{cases} V_{integer} = round\left(V_{double} \times 127\right) \\ V_{double} = V_{integer} / 127 \end{cases} \tag{7-3}$$

where $V_{integer}$ is the integer index value and V_{double} is the real double value. Notice that an arbitrary double value in the range of $[-1,1]$ has been digitized to an integer value in the range of $[-127,127]$, as represented by each dot in the right panel of Fig. 7-8. To hold an integer value within this range, an 8-bit (1 byte) memory unit is sufficient. By this conversion, the double multiplication in FFT has been reduced to integer multiplication. This reduction saves both calculation time and storage memory. We can also take advantage of the hardware multiplier on MSP430F449, to further expedite the calculation.

Digitizing the float-point values of a sine wave into integer values certainly saves time and memory, but it causes the numerical accuracy problem. When a float-point value is converted into an integer value, its precision is reduced to one over the full scale of the integer range. In the above example, the precision is limited by $1/127$ after digitization. This precision loss cannot be taken back by converting the integer value back to double. So, to increase the precision and reduce the round-off error, more digitization layers, thus larger integer scale, is desired. For example, if 10-bit is used to digitize a float-point value, a precision of $1/511$ can be archived, at the price of 2-byte storage for each integer value and a little bit longer calculation time. Choosing how many bits should be used to digitize a float-point value depends on the

specification of the system. More bits, higher precision but larger memory requirement and longer calculation time; less bits, faster execution and smaller memory storage, but worse accuracy.

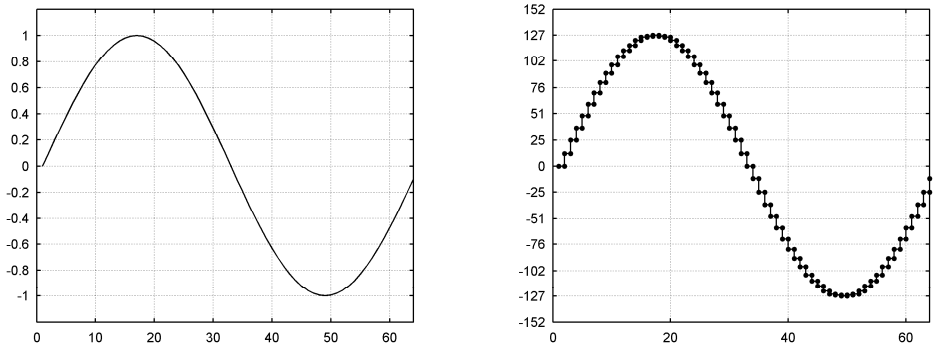


Fig. 7-8. Digitization of a sine wave. Left panel: continuous values ranging from -1 to 1; Right panel: Sine wave digitized by 256 layers.

To find out how many bits should be used in our system, a series of tests have been run based on different digitization bits. However, before presenting the results of the tests, we should take a look at the application of FFT in our system in order to make the tests results justified.

In our system, the primary usage of FFT is to perform the spectrum analysis on R-R interval series. The sampling rate of ECG in our system is set as 200 Hz. Thus the finest time resolution of R-R interval is 5 milliseconds ($1000\ \text{ms}/200 = 5\ \text{ms}$). The real R-R interval measured in millisecond has the following relationship with the number of sampling points between two consecutive R-R peaks:

$$T_{\text{R-R}} = 5\ \text{milliseconds} \times N_{\text{R-R points}} \tag{7-4}$$

where $T_{\text{R-R}}$ is the R-R interval measured in milliseconds; while $N_{\text{R-R points}}$ is the number of sampling points of the corresponding R-R interval.

Since "5" is a constant and it does not change the result of FFT except rescaling the magnitude, it is unnecessary to carry this constant when doing FFT. So, the FFT can be actually performed on $N_{\text{R-R points}}$ series, the number of sampling points between two consecutive R-R peaks. What's more, a closer observation of series $N_{\text{R-R points}}$ reveals that its value varies only within a certain range. If we assume the heart rate we are measuring is within the range of $[41, 270]$, the corresponding range of $N_{\text{R-R points}}$ is $[300, 45]$. A value within this range needs two bytes to hold it since one byte can only hold a value up to 255. But we notice that there is a baseline in the $N_{\text{R-R points}}$ range. The true dynamic range of the $N_{\text{R-R points}}$ series is actually only 255 ($300 - 45 = 255$), which is the range a single byte can hold. So, we can preset a mean value for the $N_{\text{R-R points}}$ series, and only store the offset to this mean value instead of directly store the original $N_{\text{R-R points}}$ values. For example, let's choose a preset mean value of 173. If the R-wave detection algorithm finds the current R-R interval is 300 sampling points, we don't store the number 300 but store the number $300 - 173 = 127$. Similarly, if the detected R-R interval is 45 sampling points, we store the number $45 - 173 = -128$ instead of 45. By this means, the range of the values that we should store is $[127, -128]$, corresponding to the true heart rate of $[41, 270]$.

The range of $[127, -128]$ is exactly the range of a signed 8-bit integer. Thus, one single byte (8 bit) is sufficient to hold the equivalent R-R interval series in contrast to use 2-byte to store a value in range $[300, 45]$. Performing FFT on such series does not lose any information including the DC component. The DC component is just split into two parts, one is the preset mean and the other is the mean of the offset series. By this manipulation, the memory consumption has been reduced to half of that it seems to be needed. As a consequence, less byte are needed to store the temporary results of the FFT as well. This further saves more memory. Besides that, this manipulation also expedites the execution of the FFT because microprocessor takes less time to do mathematical calculations on shorter data type.

Through the above analysis, we concluded that given the assumption of 200 Hz sampling rate and $[41, 270]$ heart rate range, doing FFT on R-R interval is equivalent to doing FFT on a time series taking any possible integer values in the range of $[127, -128]$. The tests for choosing appropriate digitization bit were performed on these preconditions. In a single test, an integer time series was generated by randomly taking 128 values in the range $[127, -128]$. The values were taken according to uniform distribution. The PSD of the test data was calculated based on normal double precision FFT and digitized integer FFT separately. Surely, there would be slight difference between these two PSDs because FFT with integer precision is just an approximation of FFT with double precision, and the error may be different at each frequency bin as well. We took the maximum absolute error between the two PSDs as the "PSD Error" for that particular test data. By doing this, we exaggerated the error by considering "the largest possible error" introduced by the digitization. The absolute PSD Error value was then normalized by the total energy of the PSD and is reported as a percentage. Formula expression of the PSD Error is given by:

$$\text{PSD Error} = \frac{\max\left|\text{PSD}_{\text{double}}(f) - \text{PSD}_{\text{integer}}(f)\right|}{\text{sum}\left(\text{PSD}_{\text{double}}(f)\right)} \times 100\% \tag{7-5}$$

For each digitization bit, the test was repeated 1000 times. The mean and standard deviation of the PSD Error were calculated, as reported in Fig. 7-9. It's clear to see that when only 8-bit is used, the mean PSD Error can be as high as 0.28%. And the standard deviation is also high compared to other digitization bits. When more bits are used, the mean value of the PSD Error consistently decreases as expected. So does the standard deviation. However, when the bit is increased to 12, both the mean value and standard deviation are almost the same as those of bit 13 to bit 15. Since more bits introduce longer computation time without gaining much improvement in accuracy, we chose 12 bit to digitize the float-point value of $\cos\omega t$ and $\sin\omega t$. The precision is of $1/4095$ at 12-bit digitization. With this precision, the mean PSD Error is only 0.01% and the standard deviation is only 0.005%, as shown in Fig. 7-9.

Using integer FFT as a replacement of float-point FFT can be considered as part of the optimization process. Other more general optimization techniques to speed up the calculation are discussed in 7.4.3. The calculation time of FFT is discussed in 7.4.4, where a general method to measure the execution time of any subroutine is also demonstrated.

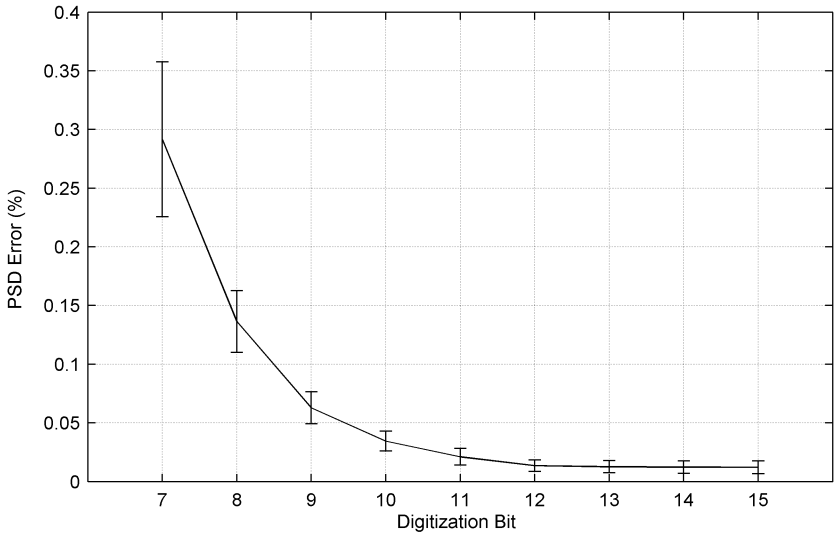


Fig. 7-9. Numerical accuracy of integer FFT. At each digitization bit, 1000 tests were performed. The PSD of each test data was calculated in double precision and integer precision. The maximum error between these two PSDs was taken as the error for that particular test. The standard deviation was calculated based on the 1000 tests. As shown in the figure, when digitization bit is increased from 12 bit to 15 bit, both the absolute error value and the standard deviation of the error are significantly lower than fewer bits.

7.4.3 Code optimization

In the development of embedded systems, it is very important to optimize program code to enhance the performance of the system. Embedded systems have very limited resources such as small memory and weak processing ability. Code optimization techniques are developed to either reduce the code size so that more memory can be saved, or to simplify the complexity of the program to shorten the execution time. However, it is often contradictory to both reduce memory usage and speed up execution. Typically, larger code size corresponds to shorter execution time and vice versa.

The optimization can take place in different phases of the program development. The most direct and simplest way is to take advantage of the complier optimization feature. Nowadays, almost all compliers provide a certain level of code optimization. The complier optimization can be carried out with respect to size or time upon configuration. For example, the IAR Embedded Workbench® IDE provides three optimization levels: low, medium and high. Although, complier optimization is very easy to carry out, a developer should never fully rely on the complier optimization. Not only the complier optimization has limited impact on the improvement of the performance, but sometimes it is also not safe. Erroneous results may be generated by the complier optimization. For example, considering the following code segment:

```
{
      int state;
      ……
      state = 1;
      Led.Off();
      if(state == 1)
            Led.On();
}
```

If the code is optimized by a complier, the complier most likely will skip the comparison of variable `state` to value `1`, as specified by the statement `if(state == 1)`. When the complier analyzes the code, it detects the fact that between the statement `state = 1` and `if(state == 1)`, there is no other instruction to change the value of the variable `state`. Thus, the complier concludes that variable `state` is always one when the statement `if(state == 1)` is reached. So, there is no need to actually execute this comparison since the result of the comparison is always "true". By skipping the actual comparison, the complier will probably generate the following "optimized" code to replace the original one:

```
{
      int state;
      ……
      state = 1;
      Led.Off();
      Led.On();
}
```

However, in an embedded real-time system, it is very possible that the value of variable `state` is alternated by other interrupt service routine during the call of `Led.Off()`. The `state` value may not be `1` before the call of `Led.On()`. Under such circumstance, the original code correctly skips the call of `Led.On()` by faithfully comparing the `state` value to 1; while the optimized code falsely makes a call of `Led.On()` since it has the wrong assumption that the value of `state` remains as `1` and skips the comparison for optimization purpose. This certainly gives a wrong indication of the `state` value. When this happens, the optimized code does not have identical behavior as its original version. The system will behave unexpectedly if it has such kind problems. For this particular example we do have a fix. That is, to declare `state` as `volatile int state`. The keyword `volatile` actually tells the compiler not to do any optimization on this variable. By this means, however, we have avoided the problem, but paid the price of no optimization.

The above problem and other similar ones caused by compiler optimization can only be detected at run-time. And, to make things worse, it is very hard at run-time to even recognize what exactly the problem is, not to say to locate the problem. The only clue we have is that the system behaves weirdly after optimization. It often takes a long time and a frustrating procedure to finally locate this kind of bugs. To save our lives, it is wiser to put the optimization under the control of our own hands.

The better way to optimize the code is in the development phase of the program. Especially in a real-time digital signal processing system, tremendous mathematical operations are involved and the computation time is critical. By carefully analyze the algorithms, alternative implementation can be used to achieve the same mathematical results but with very high efficiency.

The following paragraphs describe some guidelines of optimizing code in the development phase. As examples, the applications of the optimization techniques in our system are also discussed to show the efficiency of these techniques.

Reform the algorithm in recursive format

Many digital signal processing algorithms have the nature of recursiveness. Rewriting the expression of the DSP algorithm in its equivalent recursive format often brings surprising reduction of the computation complexity. Recursiveness is also one of the natures of a real-time system because in real-time system, new data comes into the system in a one-by-one fashion. Recursive implementation of the DSP algorithms not only reduces the computation complexity,

but also processes the new data right upon its arrival – the system does not need to stay in idle state for a long time waiting for a bunch of data. The recursive processing strategy actually distributes the computation load evenly according to time.

In the embedded R-wave detection algorithm discussed in 7.4.1, the first step is to low-pass filter the raw ECG signal. Given the description of the low-pass filter by Eq. 7-1, the direct way to implement it is to perform the following mathematic operation:

$$\begin{aligned} y(n) &= x(n-10)+x(n-9)\times 2+x(n-8)\times 3+x(n-7)\times 4+x(n-6)\times 5 \\ &\quad +x(n-5)\times 6+x(n-4)\times 5+x(n-3)\times 4+x(n-2)\times 3+x(n-1)\times 2+x(n) \end{aligned} \tag{7-6}$$

where $x(n-i), i=0.....10$ is the raw ECG signal and $y(n)$ is the low-pass filtered signal. In this implementation, nine multiplications and ten additions need to be carried out as Eq. 7-6 shows. But there is a much conciser way to implement it. If we look at the previous value of the filtered signal $y(n-1)$, we notice that:

$$\begin{aligned} y(n-1) &= x(n-11)+x(n-10)\times 2+x(n-9)\times 3+x(n-8)\times 4+x(n-7)\times 5 \\ &\quad +x(n-6)\times 6+x(n-5)\times 5+x(n-4)\times 4+x(n-3)\times 3+x(n-2)\times 2+x(n-1) \end{aligned} \tag{7-7}$$

Subtracting $y(n-1)$ from $y(n)$ yields:

$$\begin{aligned} y(n)-y(n-1) &= -x(n-11)-x(n-10)-x(n-9)-x(n-8)-x(n-7)-x(n-6) \\ &\quad +x(n-5)+x(n-4)+x(n-3)+x(n-2)+x(n-1)+x(n) \end{aligned} \tag{7-8}$$

Let:

$$\begin{cases} S_s(n) = x(n-11)+x(n-10)+x(n-9)+x(n-8)+x(n-7)+x(n-6) \\ S_a(n) = x(n-5)+x(n-4)+x(n-3)+x(n-2)+x(n-1)+x(n) \end{cases} \tag{7-9}$$

Then:

$$y(n) = y(n-1)+S_a(n)-S_s(n) \tag{7-10}$$

The value of $y(n)$ can be recursively calculated by Eq. (7-10) based on previous value $y(n-1)$. Notice that no multiplication operation is involved in Eq.(7-9) and Eq. (7-10), thus the calculation is certainly speeded up.

The right term in Eq. (7-10) can be further made recursive. Notice that both $S_a(n)$ and $S_s(n)$ are the summations of six consecutive retro data points, the $S_a(n)$ and $S_s(n)$ themselves can be updated in a recursive way:

$$\begin{cases} S_s(n+1) = x(n-10)+x(n-9)+x(n-8)+x(n-7)+x(n-6)+x(n-5) \\ \qquad\quad = S_s(n)+x(n-5)-x(n-11) \\ S_a(n+1) = x(n-4)+x(n-3)+x(n-2)+x(n-1)+x(n)+x(n+1) \\ \qquad\quad = S_a(n+1)+x(n+1)-x(n-5) \end{cases} \tag{7-11}$$

Initially, all the values of $y(n)$, $S_s(n)$ and $S_a(n)$ are set to 0. As new data comes in one by one, these values are recursively updated by Eqs. (7-11) and (7-10). Comparing Eqs. (7-11) and (7-10) to Eq. (7-6), we find that the original nine multiplications and ten additions have been taken over

by only three additions and three subtractions. Not surprisingly, the complexity of the low-pass filtering has been greatly reduced and large amount of computation time has been saved.

A similar reforming was also manipulated on the calculation of the instantaneous variance in the R-wave detection algorithm. The recursive method further reduces the computation complexity and speeds up the processing.

Avoid multiplication and division

The key point of optimizing the digital signal processing algorithms is to avoid multiplications and divisions as possible as it can be. No matter whether the microprocessor has hardware multiplier or not, it always takes longer time to do multiplications than additions or subtractions. A division operation even takes longer time than a multiplication. So it is best to avoid the multiplication and division by using alternative mathematical operations. The above example is a very good demonstration of replacing multiplications with additions or subtractions.

Another way is to replace multiplications and divisions with bit manipulations. Multiplication can be replaced by left bit-shift if one of the multiplier is the power of 2. In general, shifting an integer one bit left is equivalent to multiplying the number by 2. However, bit shifting is not a general alternative of multiplication. It can be done only when at least one of the multipliers is the power of 2.

Division takes longer calculation time than any other mathematic operations. And it is also the hardest operation to avoid. The only possible substitution to division is right-bit-shift. Shifting an integer one bit right is equivalent to divide the number by 2. However, this operation is further restricted that the divisor must be the power of 2. In addition to this, the right-bit-shift operation also introduces round-off error. Shifting one bit right always truncates the result to the nearest integer towards 0 for a positive integer. For example, shifting one bit right of number 5 (0101b) yields 2 (0010b). This round-off error should be compensated to obtain more accurate results. Otherwise, this error will accumulate in a recursive calculation. If that happens, the result will be totally wrong even just after several steps of recursion. In 7.4.2, we concluded that to save memory space and expedite the FFT analysis, we digitized the float-point values of $\cos\omega t$ and $\sin\omega t$ with 12-bit precision. Thus, in each stage of the butterfly algorithm, the results are amplified by 4096 times (since value "1" is mapped to $4096 = 2^{12}$). After the multiplications are done, the results have to be rescaled back by dividing 4096. Otherwise, the magnitude would be accumulatively amplified by 4096 times at every stage of the butterfly algorithm. The results will be ridiculously large and will be certainly out of the range of any integer type. For example, in a 128-point FFT, there are 7 ($\log_2 128 = 7$) stages of the butterfly algorithm. Thus, the result magnitude will be amplified $\left(2^{12}\right)^7 = 2^{84}$ times without rescaling! The division of 4096 for rescaling purpose can be accomplished via a right-shifting the results by 12 bit, but the above round-off error must be somehow compensated. In our system, we added a compensation term to those positive values before doing the right-shifting. The compensation term is the half scale of the full digitization range, which is 2048 ($4096 \div 2 = 2048$) for 12-bit digitization. See the complete code of the FFT listed in Appendix III. The PSD Error shown in 7.4.2 has already included the round-off error compensation.

Trade time with memory

If the system's responsiveness is of critical, as it is in most real-time system, it is wise to sacrifice a little memory space to store some predetermined values instead of calculate these values repeatedly at run-time. A good example is the FFT algorithm. Once the length of the FFT is chosen, the values of the basis functions $\left(\cos\omega t, \sin\omega t\right)$ are determined at each sampling

time. It is quite unnecessary to calculate the values of the triangle functions over and over again every time a FFT is performed because it takes much calculation time. Since these values are pre-determined, the typical way is to calculate these values based on the FFT length in the initialization phase of the system when the system is started. The values are then stored as a look-up table in the memory. Thus, the values can be retrieved by a very fast memory access instruction. Building up the look-up table in this way not only saves a lot of computation time, but also has a certain level of flexibility. If the FFT length is changed due to some reason at run time, the program can recalculate the values and update the table entries. This update is needed only once every time the FFT length is changed. The gain of this method is the fast access to the triangle function values and the flexibility of changing these values at run-time, but the price paid is some amount of RAM memory. Since the values are calculated at run-time (not at compile time), they have to be stored in RAM memory instead of ROM memory. But RAM memory is typically more expensive than ROM and is more limited than ROM in embedded systems. For example, if the FFT length is 128 and each value of $\cos \omega t$ and $\sin \omega t$ is stored in 2 bytes, then totally 512 bytes are needed to store this look-up table. This is sometimes intolerable. In the MSP430F449, the RAM is limited by 2048 bytes. It's probably not a good idea to store the 512-byte look-up table in such a crowded RAM.

There is another alternative way, though. The on-chip RAM is quite limited but the on-chip ROM is often plentiful because ROM is much cheaper than RAM. ROM is used to store the executable code and other constants that cannot be modified at run-time. The alternative way is to move the look-up table from RAM to ROM. To do this, the look-up table should be built in the source code at compile time instead of at run-time (see the FFT source code in Appendix III.). In the source code, the table is declared as constants. When the complier generates the executable code, the table is automatically put in the ROM by the complier. If desired, we can also specify at what exact memory address we want to put this table by informing the compiler with directive commands.

In the implementation of FFT, three tables are needed: the $\cos \omega t$ table, the $\sin \omega t$ table and the bit-reversal table, as shown in the FFT source code in Appendix III. These three tables occupy 640 bytes in ROM, which is trivial compared to the total amount of the on-chip ROM (60 KB). The only drawback of this method is that the tables cannot be updated at run time. If the FFT length needs to be changed, the tables have to be updated in the source code and reload to the microcontroller.

Other general optimization methods

There are also some other optimization techniques to make the program run fast and more reliable. Such techniques include loop unrolling, using register variables, using global variables and function inlining etc [107].

7.4.4 Timing aspect of the embedded algorithms

In embedded system, the processing capability of the microprocessor is often limited. This limit arises from the fact that the microprocessor in embedded system often has much slower clock frequency than that of a personal computer. Especially in real-time digital signal processing system, this limit can significantly affect the system design and implementation because many digital signal processing algorithms are so complicated that they occupy most of the processing capability of the microprocessor. If the microprocessor is kept so busy running the DSP algorithms, the interrupt latency will be significantly increased and the event generating such interrupt may not be handled in time as it is expected. In the worst case, the system crashes due to congested tasks and interrupts. At this point, it is important to estimate the execution time

of each task so that the robustness of the system is assured.

Benchmarking the execution time of a subroutine in an embedded system is a little challenging because the time is measured in a very small time scale like milliseconds or even microseconds. Given the fact that there is no direct method neither on hardware nor on software to measure such a short time period, an indirect approach was used to achieve this goal.

The method used in this dissertation is to utilize the Input/Output (I/O) port of the microcontroller as the indirect indicator of the execution time of a task. The I/O port of the microcontroller can be set at logic high level or logic low level by the software. The output of the I/O port responds to the software instruction very fast. The typical responding time is on the same order as the system clock. For example, if the microprocessor is running at an 8 MHz clock, the I/O port responding time is less than 1 microsecond. The idea is that before a task is to be executed, use software instruction to set a certain I/O port to logic high level. Since the task features in "run to completion" (see 7.3.2), the execution flow of the microcontroller is "blocked" during the execution of a certain task (ISRs, however, are not blocked from task.). Thus, the I/O port stays at high level until it is explicitly changed by another software instruction that is imposed after the task. At the completion of the task, the software instruction pulls the I/O port down to logic low level. Thus, the logic high time duration on this I/O port is corresponding to the execution time of the task. By measuring the time duration of logic high on the port, a good estimation of the task execution time can be obtained. The following example code can help to understand the usage of this method:

```
#define     _EXEC_TIMER1_HIGH()     P1OUT |= BIT6
#define     _EXEC_TIMER1_LOW()      P1OUT ^= ~BIT6

{
      _EXEC_TIMER1_HIGH();
      pCurrentTask->Execute();
      _EXEC_TIMER1_LOW();
}
```

The instruction `P1OUT |= BIT6` sets the pin 6 (pin #81 on MSP430F449) of the P1 I/O port to high and the instruction `P1OUT ^= ~BIT6` sets the same pin to low. As shown above, the code that needs to be measured must be embraced by `_EXEC_TIMER1_HIGH()` and `_EXEC_TIMER1_LOW()`. According to the example code, the duration of high level on pin #81 of the microcontroller is the execution time of task `pCurrentTask`.

A typical waveform on the I/O port pin is a periodical square wave. In order to measure the duration of high level in the square waveform, we recorded the waveform with a very high sampling rate. In this dissertation, the PowerLab/4SP provided by ADInstruments Company was used to record the square waveform. The PowerLab/4SP provides a sampling rate up to 100k Hz. At this sampling rate, the time resolution is 10 microseconds (0.01 milliseconds). Since all the digital signal processing algorithms in our system have the execution time on the order of milliseconds, this time resolution is high enough to get a good estimate of the execution time.

Execution time of the embedded R-wave detection algorithm

The real-time R-wave detection algorithm is a point-by-point algorithm. Every time a new data is sampled, the algorithm should take the data and go through the detection procedure described in 7.4.1. If the sampling rate of the ECG signal is 200 Hz, the R-wave detection algorithm is executed repeatedly 200 times in a second. This is equivalent to a time interval of 5 milliseconds between two consecutive executions of the R-wave detection algorithm. If the R-wave detection algorithm were unable to complete a single execution in less than 5 milliseconds,

the system would crash down. Depending on the data that the algorithm is working on and the current state of the algorithm state machine, the execution time of the R-wave detection algorithm also varies. As discussed in 7.4.1, once a valid ECG signal is detected, the FSM of the R-wave detection algorithm goes through a typical cycle periodically in these three states: `STATE_IDLE`, `STATE_PRERWAVE`, and `STATE_RWAVE_DETECTED`. Thus, it is not coincident that the execution time of the R-wave detection algorithm falls into three distinguished ranges, as shown in Fig. 7-10. In Fig. 7-10, roughly 12000 times of execution of the detection algorithm are shown. Each dot in the figure represents the completion time of one single execution. The time shown here is based on 8 MHz CPU clock.

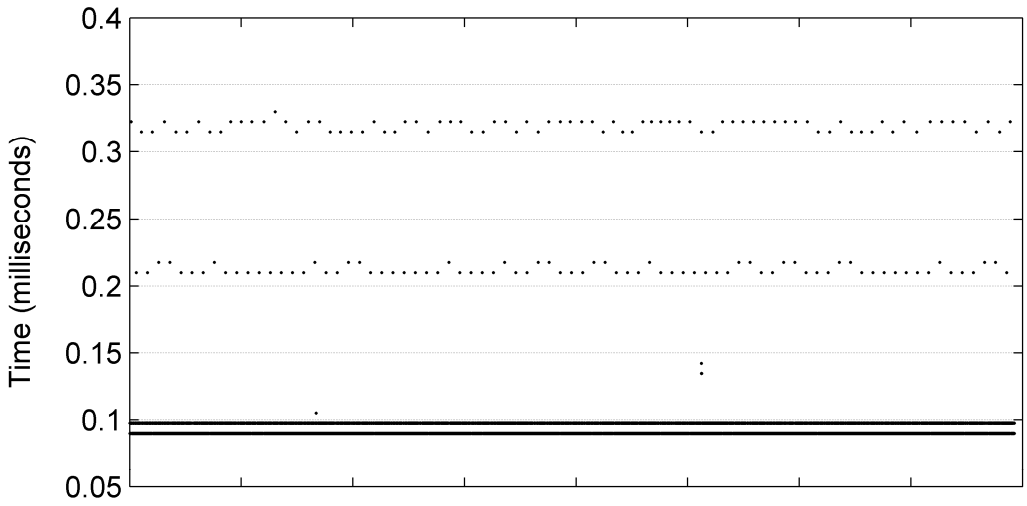


Fig. 7-10. Execution time of the embedded R-wave detection algorithm (8 MHz CPU clock)

The first range is from 0.09 milliseconds to 0.1 milliseconds. This execution time corresponds to the `STATE_IDLE` state. The execution time in this state accounts for the time of low-pass filtering process, calculation of the derivative signal and variance of the derivative signal in a preset window. Due to the recursive and other optimization techniques discussed in 7.4.3, this preprocessing stage is made very short and fast. The preprocessing is performed every time a new data is sampled no matter what state the FSM is. Thus, this execution time of the preprocessing stage is the minimal time needed by the R-wave detection algorithm. After the preprocessing, a simple comparison between the calculated variance and the threshold is performed. If the variance is less than the threshold, the detection subroutine exits and the FSM remains in `STATE_IDLE` state. If not, the FSM will shits to a new state. The state transition from `STATE_IDLE` to `STATE_PRERWAVE` contributes to the second time range shown in Fig. 7-10, which is from 0.21 to 0.22 milliseconds. This state transition needs additional operation besides the low-pass filtering and calculation of the variance. So, the execution time is a little longer. The third range in Fig. 7-10 denotes the possible longest execution time, which occurs when an R-peak is detected. When an R-peak is detected, the R-R interval between the current peak and the previous one is calculated and stored in a buffer. Besides, the adaptive threshold is also updated and the FSM is switched to `STATE_RWAVE_DETECTED` state. These additional operations further increase the execution time. It is roughly 0.31 to 0.32 milliseconds, as shown in Fig. 7-10. From the above analysis we can conclude that in the detection of each individual R-peak, only one point needs a processing time in the 2nd range and another point needs a processing time in the 3rd range because the longer time occurs only at the state transition while the state transition occurs only at a single particular point. Majority of the processing time is less

than 0.1 milliseconds. Besides, among the three time ranges, even the longest one (0.3 milliseconds) is far less than the sampling interval (5 milliseconds). Thus there is a lot of margin of the processing time available for other DSP algorithms.

Execution time of embedded FFT

The same technique was used to measure the execution time of the embedded FFT algorithm. In the current version of SP-DSP, the FFT length is fixed at 128 points. Thus, the execution time referred here is specific for 128-point FFT. Since the FFT complexity is approximate proportional to its length, the calculation time can be estimated for other length of FFT, if desired. Besides, as described in 7.4.2, the FFT is implemented with 12-bit accuracy and is carried out based on fix-point type. The FFT code also has been optimized by the techniques described in 7.4.3.

The system is under normal running conditions when the test is performed. This means other system functions are also open and active. These system functions include the ECG acquisition subroutine sampling the ECG signal at 200 Hz sampling rate. The R-wave detection task is also active and is posted every 5 milliseconds. The FFT task is executed every one second. One minute continuous running of the system is recorded and analyzed. So, totally 60 times of the FFT execution are recorded. The averaged time over the 60 executions is 27 milliseconds, and the standard deviation is very small. It is even smaller than the time resolution (10 microseconds) of the measuring method, indicating the FFT is completed within a very consistent time frame.

After getting the knowledge of the execution time of the embedded digital signal processing algorithms like the FFT, it is worth to revisit the task management mechanism discussed in 7.3.2. In that section, we designed the DSP algorithms as tasks that are run by the microcontroller in a different mode from the execution of ISR. These tasks are maintained in a system task queue. If a DSP algorithm needs to be called in an ISR, the ISR pushes the request of running the DSP algorithm into to the task queue instead of calling the algorithm directly. By this means, the execution time of ISR is kept very short. Making ISRs as short as possible is a golden requirement for the development of real-time system. It will be a disaster if an ISR takes too long time to complete. Noticing the fact that the interrupts are prioritized, those ISRs with lower priorities are blocked from execution when a higher priority ISR is being run. This will certainly increases the interrupt latency for those lower priority interrupts. In many cases, the interrupt requests of those lower priority interrupts keep being generated even though the previous requests have not been served yet due to the long execution time of a higher priority ISR. This causes unexpected behavior of the system and may also crash the system.

Considering the FFT example, we now know the execution time of an FFT is about 27 milliseconds. Given the sampling rate of 200 Hz, there will be at least 5 times of the timer expiration that requests an AD conversion in this time duration. The microcontroller is committed to response the requests in time by informing the AD converter to initiate the conversions. However, if the FFT subroutine is executed from a call made in an ISR instead of from the task queue, the execution of the caller ISR is hanged up until the completion of the FFT after 27 milliseconds. During the hang-up of the ISR, the requests of AD conversion are blocked because the interrupt of AD conversion request has lower priority than the ISR that invokes the FFT. The microcontroller is kept busy on doing FFT but has no opportunity to inform the AD converter as it is supposed to. As a consequence, the data in the execution of FFT has been lost.

Instead of calling the FFT subroutine directly in the ISR, pushing the request of running FFT into the task queue is a fundamental solution to the aforementioned problem. In the ISR, only several simple instructions are needed to push the request of running FFT into the task queue. After that, the ISR quickly returns, releasing the microcontroller to deal with any other priority

interrupts. When the microcontroller has no interrupts to deal with, it keeps polling the task queue to check if there are unexecuted tasks remaining in the queue. The task is popped out of the queue and executed when the microcontroller is free from dealing with interrupts. Since tasks can be preempted by any interrupts, as described in 7.3.2, the microcontroller will not miss the requests of AD conversion during the execution of FFT task.

Fig. 7-11 illustrates the execution flow when FFT is run as a task and how the contents of the system task queue change. The execution flow starts with a single FFT task in the task queue. Since the microcontroller is free, the FFT task is fetched out and started to execute. At sometime (5 ms), the timer expires to demand an AD conversion. The microcontroller responses to the timer interrupt immediately by putting the FFT task on hold for a very short time. The execution flow is branched to the timer's ISR. Recall that the R-wave detection algorithm should be applied to every sampled point. Thus, when ISR gets a new sampled data, it pushes the call of R-wave detection subroutine at the rear of the task queue, as shown in Fig. 7-11. The pushing process is much faster than the R-wave detection subroutine itself, so the microcontroller will not be stuck in the ISR. The execution flow then returns from the ISR to the main loop and resumes the FFT execution. From the FFT task's point of view, the execution of FFT is just paused for a very short time (the execution time of the ISR, typically several microseconds). At time 10 ms, the timer expires again. So the above steps are repeated. Notice that the FFT task has not been completed yet, the previous call of R-wave detection subroutine is still in the task queue because a task (the R-wave detection task [1]) cannot exempt another task (the FFT task). Thus pushing another call of the R-wave detection subroutine causes the task queue to grow. Now, there are two unexecuted tasks in the queue. Before the FFT task is completed, the task queue keeps growing since more and more calls of R-wave detection are pushed in, as shown in Fig. 7-11. Following the completion of the FFT, the firstly pushed-in R-wave detection task is popped out and executed; and so are the others consequently. Notice that one single execution of the R-wave detection only takes less than 0.32 milliseconds; the microcontroller can finish all the R-wave detection tasks remaining in the queue within a very short time.

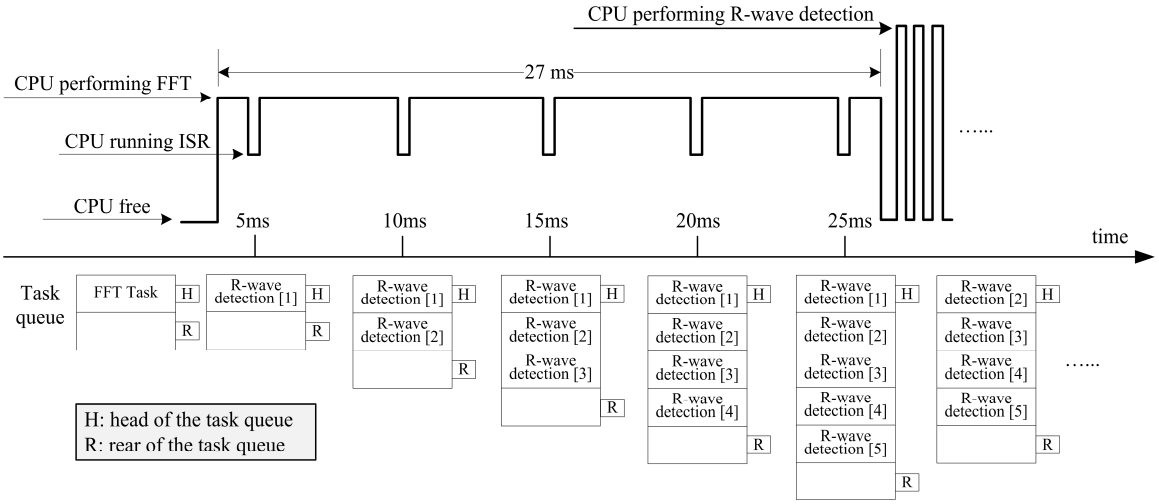


Fig. 7-11. Changes of the system task queue contents during the execution of FFT

7.5 Summary

The *Tmote Sky* has already incorporated an ultra-low power consumption microcontroller MSP430FG1161 on board. But since the microcontroller has to run *TinyOS* to support the

wireless data transmission, there is no much margin of its processing ability available for more real-time DSP algorithms. In order to enhance the on-board digital signal processing ability, an additional microcontroller with low-power consumption has been integrated into the system. The additional MSP430F449 microcontroller belongs to the MSP430 family, which provides a series of microcontrollers with ultra-low power consumption so that they are especially suitable for battery-powered systems.

Introducing an additional microcontroller also brings up a demand of supporting software for it. To meet this demand, a software platform, SP-DSP, was developed specifically for the system. The design of SP-DSP separates the hardware-related programming from those hardware-independent programming such as the DSP algorithms. The hardware-related programming, noted as low-level implementation, is to write driver program for the hardware components residing in the additional microcontroller. At this time, the drivers for the timer, AD converter, and USART port have been developed. They are abstracted as three classes in the SP-DSP. The low-level implementation of the SP-DSP handles the hardware specifications, thus in the case hardware is upgraded, this part of code should be updated.

The high-level implementation of SP-DSP employs the "task + event" concurrence model. The DSP algorithms are designed as "tasks" running in the background when the microcontroller is free. The data collection subroutine is implemented as "events" so that the signal is sampled instantaneously without being deferred. The tasks are managed by a task queue, which is polled all the time in the main loop. The task at the head position of the queue will be popped out and executed if the microcontroller is free. The tasks have the "run to completion" characteristic. A task cannot be preempted by another task. So, at any time, there is at most one task being run. The events, on the other hand, have the highest priority in the system. They can preempt any tasks. The events themselves are prioritized so that they also preempt each other.

With the support of the developed SP-DSP, some DSP algorithms have been successfully embedded into the additional microcontroller. To achieve fast real-time responses of the embedded DSP algorithms and preserve more battery power from computation, code optimization techniques have been applied to the embedding of these algorithms. These optimization techniques include deriving recursive version of the algorithm; using fixed-point data type instead of float-point type; avoiding multiplications and divisions; and using look-up table instead of computing the constants at run-time. The experiments have shown the optimized R-wave detection typically takes less than 0.1 ms to complete and the maximum execution time is no longer than 0.32 ms. The execution time of the optimized 128-point FFT is around 27 milliseconds and this value is very consistent every time FFT runs.

Chapter 8 Future Development

As the aging of population is becoming a world-wide reality, health care delivered to the elder people has become a growing society problem as well as an industrial market. The long term health monitoring is the first step of providing the elder people better health and medical service. Driven by these specific needs, the development of vital parameters monitoring devices will follow these trends:

1. More Portable.

It is obvious that the more mobility the device is, the more people will like it and use it; so the more the device can benefit people and the society. More mobility of the monitoring device is achievable only with the development of hardware and software. To miniature the device, integrated circuit technique is playing an important role. For the device developed in this work, the wireless transmission module (*Tmote Sky*) has a microcontroller (MSP430F1611) that controls another separate wireless transmission chip (CC2420). It is now, however, possible to integrate the two chips into a single one. Texas Instruments has just issued the new generation of System on Chip (SoC) solution for the wireless mesh network – CC2430 and CC2431. The new CC2430 and CC2431 have remarkable feature and decent design to meet the specific requirements of long-term low-power consumption of wireless monitoring and data collection systems. Inside the tiny 7 mm × 7 mm chip, the wireless transceiver, CC2420, has been integrated with another industrial-standard enhanced microcontroller 8051. Together integrated are 32/64/128 KB flash memory, 8 KB RAM, and other peripheral hardware components such as timer, ADC, and DMA controller. The CC2431 even also integrates a hardware Location Engine that computes the location of the node in the mesh network in real-time. With so many components integrated and so powerful function it provides, the size of the new generation wireless device is kept incredible small.

As a general solution of wireless mesh network, CC2430 and CC2431 are providing a successful SoC solution. But for the development of health monitoring device, the step of seeking SoC solution should not stop here. In future technology advances, the bio-sensors themselves can be integrated into a single chip as well, offering a true and complete SoC solution specifically for vital parameters monitoring. Even further, the bio-sensors, the wireless transceiver and the microcontroller can be integrated into a single chip all together to further reduce the device size. At the technique aspect, it will not be a problem to produce a low-cost, low-battery power, watch-sized device that provides multiple vital parameters monitoring and sophisticated real-time signal processing together with wireless data transmission. The most concern is the maturity of the market.

Most progress in putting a bio-sensor into a single chip has been achieved in miniaturing ECG sensor. Quite a few of researches have been attemped and some products are now available [108-111] by following the concept of "ECG in an electrode" or "ECG on chip" [108]. The ECG sensor profile is being reduced to an incredible small size. For example, the CHEFREN, provided by Aurelia Microelettronia, has a small size of 5.8 × 5.1 mm^2 [110]. Another ECG chip being developed even has a much smaller size of 3.4 × 2.1 mm^2 [108]. The small size of ECG sensor will significantly reduce the mechanical volume of the monitoring device. Currently, miniaturing other bio-sensor is rarely reported. In future development, it is possible to integrate more bio-sensors, such as blood pressure sensor, breath sensor, SpO_2 sensor, and other physiology measurement sensors, into a single integrated circuit (IC) to provide multiple parameters monitoring simultaneously via a single chip.

Miniaturing the bio-sensors not only reduces the size, but also greatly decreases the power consumption of the sensor. For example, the integrated ECG sensor reported in [109] only

consumes 3 mA power; while the integrated ECG chip shown in [108] has a power consumption as low as 0.18 mA. The low-power consumption is of critical importance for portable monitoring devices because these devices are powered by battery. If the power consumption of the device can be reduced to a certain low level, the device can even be powered by button battery, which in turn further miniatures the device.

2. More Real-time

Majority of the current monitoring devices just record the data and save the data somewhere for off-line review. For example, many portable monitoring devices record the data on a memory card. The card has to be handed in to the doctor for review. Other web-based implementation of such system stores the data in a database through some wireless connection like the GPRS. The doctor can download and review the data from the database at a local computer. In this case, the monitoring is still "off-line" and not real-time because when the data is being reviewed, the physiology state of the patient has already changed. The real-time monitoring system collects the data from the patient and transmits the data to the monitoring center and gets the data reviewed in real-time no matter what the geometry distance between the patient and the monitoring center is. The real-time monitoring has invaluable benefit and advantage. It quickly reveals the current state of the patient. This is critical for those patients that have such kind of disease requiring immediate medical care. For example, most heart attack starts slowly with mild symptoms. The patient undergoing heart attack is often unware of it, or not sure about what is happening. To rescue the patient from the threatening of death, the first 60 minutes . The real-time monitoring system can detect this life-threatening situation and alert the medical carer to take proper action. The real-time monitoring feature also expands the applicable scenarios of such kind of system. For example, in battle fields, triaging certainly needs a real-time monitoring system.

In hospital environment, the physiology parameters monitoring tends to be part of the whole hospital network. Some research have shown that it is possible to utilize the WiFi technique to establish a wireless monitoring system within the frame of the whole hospital Local Area Network (LAN) [112]. Embedding the real-time vital parameters monitoring into the hospital LAN can bring more advantages far beyond simple "monitoring". Since the bandwidth of the hospital LAN is much broader than other wireless connection such as Bluetooth, PAN, or GRPS, much more information is able to be transmitted over the network in addition to vital parameters. Such auxiliary information can include video and audio. For example, in the Intensive Care Unit (ICU), nurses are employed to frequently monitor the state of the patients. In the case of emergency, the nurse needs to call the doctor for urgent medical care through a separate paging system. It often takes some time for the doctor to be on the scene so that proper action can be conducted to stabilize the patient's state. This obviously may miss the most critical time to save the patient. It is possible in future to integrate the paging system and the vital parameters monitoring system into the hospital LAN, together with an ICU video system that monitors the patient physical reaction. By this solution, the call to the doctor is made instantaneously and as simple as push of a button. Besides, not only the vital parameters can be delivered to the doctor's hand-held device, but the live video of the ICU is made real-time accessible for the doctor. The doctor then can direct the nurse in the ICU to take proper medical actions even before he/she arrives at the scene.

In the case of telemedicine, such monitoring system can be part of the infrastructure of such system. The telemedicine system is a large scale system that cannot be built based on a mono network structure [113]. In the short range of data transmission, wireless solution is much preferred because it provides great mobility. As a consequence of such mobility, data transmission is guaranteed to be continuous even the patient is being transported among different hospital laboratories.

3. More Intelligent

Instead of an easy real-time wireless monitoring system, more intelligent systems will be developed for variety of health care applications. The intelligence of the system relies on the processing of the obtained vital signals. Dedicatedly developed digital signal processing algorithms will contribute the most to the birth of intelligence in health monitoring systems. These algorithms are able to detect emergency event of the patient and alert the health carer or the people by the patient's side. The currently most successful example is probably the heart rate monitor with automatic arrhythmia detection. These built-in algorithms at the device end can also extract more physiology parameter from a single signal so that extract sensors are eliminated from the system. For example, it has been shown that respiratory rate can be extracted either from ECG signal [114, 115] or from the PPG signal [116-119]. By this means, the device does not need an extra respiratory sensor but still be able to provide respiratory rate. This will further reduce the size of the device and preserve more battery power. Development of sophisticated digital signal processing algorithms will also expedite the processing of the recorded data. Such kind of device is intended to be used for long-term monitoring, thus the volume of the recorded data can be overwhelming. Developed algorithms will automatically analyze the huge amount of data and identify suspicious abnormal events. For example, developed digital signal processing algorithms are now able to automatically detect atrial fibrillation (AF) with pretty high accuracy [120-122]. Extraction of the health information embedded inside a single physiology signal even can further reveal valuable signs of multiple diseases. Only with long-term monitoring and dedicated digital signal processing methods is this made possible. For example, researches have shown that there is significant association between ANS and the development of diabetes. With more clinical study in this field, the diabetes might be able to be prevented based on the prediction of the extracted indices from HRV data [123-126]. In future, with more biomedical signal processing algorithms are developed and tested on clinical trials, they can be integrated into the monitoring system to unfold health signs preventing diseases from development.

With more biomedical signal processing algorithms embedded, the monitoring device will become smarter and smarter. The function of the device will be far beyond monitoring. It will be part of the diagnosis process with its given intelligence. The device can compare the monitored data with its expert database and the patient's medical history to evaluate the situation of the patient. Based on the evaluation results, the device will be able to give its own medical justification of the patient and provide valuable information upon the request of the doctor as if it were a smart medical assistant.

BIBLIOGRAPHY

[1] P. Rubel, J. Fayn, G. Nollo, D. Assanelli, B. Li, L. Restier, S. Adami, S. Arod, H. Atoui, M. Ohlsson, L. Simon-Chautemps, D. Telisson, C. Malossi, G.L. Ziliani, A. Galassi, L. Edenbrandt, and P. Chevalier, "Toward personal eHealth in cardiology. Results from the EPI-MEDICS telemedicine project," *J Electrocardiol*, vol. 38 Suppl, pp. 100-6, 2005.

[2] I. Sachpazidis, A. Stassinakis, D. Memos, S. Fragou, S. Nachamoulis, A. Vamvatsikos, A. Stavropoulou, M. Fonseca, R. Magalhaes, B. Valente, A. D'Aquila, M. Fruscione, J. Ferreira, and C. Aguiar, "@HOME is a new Eu-Project in Tele Home care," *Biomed Tech (Berl)*, vol. 47 Suppl 1 Pt 2, pp. 970-2, 2002.

[3] "http://www.americanheart.org/presenter.jhtml?identifier=3053," in, 2006, Accession no. Accession Number| Available rom Database Provider|.

[4] "http://www.wirelessecg.com/about_lifesync/index.html," in, 2006, Accession no. Accession Number| Available rom Database Provider|.

[5] "http://www.wirelessecg.com/support/pdf/SUM_86083_RevB1.pdf," in, 2006, Accession no. Accession Number| Available rom Database Provider|.

[6] "http://www.vitelnet.com/HM/vtc_home_monitoring.htm," in, 2006, Accession no. Accession Number| Available rom Database Provider|.

[7] "http://www.cardguard.com/site/products-list.asp?id=17," in, 2006, Accession no. Accession Number| Available rom Database Provider|.

[8] J. Becker, D. Gebauer, L. Maier-Hein, M. Schwaibold, J. Schochlin, and A. Bolz, "The wireless monitoring of vital parameters: a design study," *Biomed Tech (Berl)*, vol. 47 Suppl 1 Pt 2, pp. 851-3, 2002.

[9] K. Hung and Y.T. Zhang, "Implementation of a WAP-based telemedicine system for patient monitoring," *IEEE Trans Inf Technol Biomed*, vol. 7, (no. 2), pp. 101-7, Jun 2003.

[10] M.F. Rasid and B. Woodward, "Bluetooth telemedicine processor for multichannel biomedical signal transmission via mobile cellular networks," *IEEE Trans Inf Technol Biomed*, vol. 9, (no. 1), pp. 35-43, Mar 2005.

[11] R. Zou and K.H. Chon, "Robust algorithm for estimation of time-varying transfer functions," *IEEE transactions on bio-medical engineering*, vol. 51, pp. 219-228, 2004.

[12] R. Zou, H. Wang, and K.H. Chon, "A robust time-varying identification algorithm using basis functions," *Annals of biomedical engineering*, vol. 31, (no. 7), pp. 840-53, Jul-Aug 2003.

[13] R. Coifman and M. Wickerhauser, "Entropy-based algorithms for best basis selection," *IEEE Trans. Infor. Theory*, vol. 38, pp. 713-718, 1992.

[14] P. Bodin, T.O.e. Silva, and B. Wahlberg, "On the construction of orthonormal basis functions for system identification," in Book On the construction of orthonormal basis functions for system identification, vol. I, *Series On the construction of orthonormal basis functions for system identification*, Editor ed.^eds., City: World Congress, 1996, pp. 369-374.

[15] P. Bodin, L.F. Villemoes, and B. Wahlberg, "An algorithm for selection of best orthonormal rational basis," in Book An algorithm for selection of best orthonormal rational basis, *Series An algorithm for selection of best orthonormal rational basis*, Editor ed.^eds., City, 1997.

[16] S. Lu, K.H. Ju, and K.H. Chon, "A new algorithm for linear and nonlinear ARMA model parameter estimation using affine geometry," *IEEE transactions on bio-medical engineering*, vol. 48, (no. 10), pp. 1116-24, Oct 2001.

[17] A. Just and W.J. Arendshorst, "Dynamics and contribution of mechanisms mediating renal blood flow autoregulation," *Am J Phyisol Regul Integr Comp Physiol*, vol. 285, pp. R619-31, 2003.

[18] R. Zou, W.A. Cupples, K.P. Yip, N.H. Holstein-Rathlou, and K.H. Chon, "Time-varying properties of renal autoregulatory mechanisms," *IEEE transactions on bio-medical engineering*, vol. 49, (no. 10), pp. 1112-20, Oct 2002.

[19] Y.S. Cho, S.B. Kim, and E.J. Powers, "Time-varying spectral estimation using AR Models with variable forgetting," *IEEE Trans. Signal Processing*, vol. 39, (no. 6), pp. 1422-1425, 1991.

[20] Y. Xu, S. Haykin, and R.J. Racine, "Multiple window time-frequency distribution and coherence of EEG using Slepian sequences and Hermite-functions," *IEEE transactions on bio-medical engineering*, vol. 46, pp. 861-865, 1999.

[21] E.G. Lovett and K.M. Ropella, "Time-frequency coherence analysis of atrial fibrillation termination during procainamide administration," *Ann. Biomed. Eng.*, vol. 48, pp. 1116-1124, 1997.

[22] M. Arnold, W.H.R. Miltner, H. Witte, R. Bauer, and C. Braun, "Adaptive AR modeling of nonstationary time series by means of Kalman filtering," *IEEE transactions on bio-medical engineering*, vol. 45, (no. 553-562), 1998.

[23] O. Braddick, J. Atkinson, and J. Wattam-Bell, "Normal, and anomalous development of visual motion processing: motion coherence and dorsal-stream vulnerability," *Neuropsychologia*, vol. 41, pp. 1769-1784, 2003.

[24] J.N. Caviness, C.H. Adler, M.N. Sabbagh, D.J. Connor, J.L. Hernandez, and T.D. Lagerlund, "Abnormal corticomuscular coherence is associated with the small amplitude cortical myoclonus in Parkinson's disease," *Mov. Disord*, vol. 18, pp. 1157-1162, 2003.

[25] J.G. Semmler, K.W. Kornatz, and R.M. Enoka, "Motor-unit coherence during isometric contractions is greater in a hand muscle of older adults," *J. Neurophysiol*, vol. 90, pp. 1346-1349, 2003.

[26] K.H. Chon, Y.M. Chen, N.H. Holstein-Rathlou, D.J. Marsh, and V.Z. Marmarelis, "On the efficacy of linear system analysis of renal autoregulation in rats.," *IEEE transactions on bio-medical engineering*, vol. 40, pp. 8-20, 1993.

[27] W.A. Cupples, P. Novak, V. Novak, and F.C. Salevsky, "Spontaneous blood pressure fluctuations and renal blood flow dynamics," *Am J Phyisol*, vol. 270, (no. F82-9), 1996.

[28] N.H. Holstein-Rathlou, A.J. Wagner, and D.J. Marsh, "Tubuloglomerular feedback dynamics and renal blood flow autoregulation in rats.," *Am J Phyisol*, vol. 260, pp. F53-68, 1991.

[29] K.P. Yip, D.J. Marsh, and N.H. Holstein-Rathlou, "Evidence of low-dimensional chaos in renal blood flow control in genetic and experimental hypertension," *Physica D*, vol. 80, pp. 95-104, 1995.

[30] K.H. Chon, Y.M. Chen, N.H. Holstein-Rathlou, and V.Z. Marmarelis, "Nonlinear system analysis of renal autoregulation in normotensive and hypertensive rats," *IEEE transactions on bio-medical engineering*, vol. 45, pp. 342-353, 1998.

[31] K.H. Chon, Y.M. Chen, V.Z. Marmarelis, D.J. Marsh, and N.H. Holstein-Rathlou, "Detection of interactions between myogenic and TGF mechanisms using nonlinear analysis," *Am J Physiol*, vol. 267, (no. 1 Pt 2), pp. F160-73, Jul 1994.

[32] N.H. Holstein-Rathlou, A.J. Wagner, and D.J. Marsh, "Tubuloglomerular feedback dynamics and renal blood flow autoregulation in rats," *Am J Physiol*, vol. 260, (no. 1 Pt 2), pp. F53-68, Jan 1991.

[33] V.Z. Marmarelis, K.H. Chon, Y.M. Chen, D.J. Marsh, and N.H. Holstein-Rathlou, "Nonlinear analysis of renal autoregulation under broadband forcing conditions," *Ann Biomed Eng*, vol. 21, (no. 6), pp. 591-603, Nov-Dec 1993.
[34] M. Walker III, L.M. Harrison-Bernard, A.K. Cook, and L.G. Navar, "Dynamic interaction between myogenic and TGF mechanisms in afferent arteriolar blood flow autoregulation," *Am J Physiol Renal Physiol*, vol. 279, pp. F858-65, 2000.
[35] E. Moller, B. Schack, N. Vath, and H. Witte, "Fitting of one ARMA model to multiple trials increases the time resolution of instantaneous coherence," *Biol Cybern*, vol. 89, (no. 4), pp. 303-12, Oct 2003.
[36] H. Zhao, S. Lu, R. Zou, K. Ju, and K.H. Chon, "Estimation of time-varying coherence function using time-varying transfer functions," *Ann Biomed Eng*, vol. 33, (no. 11), pp. 1582-94, Nov 2005.
[37] A. Porta, R. Furlan, O. Rimoldi, M. Pagani, A. Malliani, and P. van de Borne, "Quantifying the strength of the linear causal coupling in closed loop interacting cardiovascular variability signals," *Biol Cybern*, vol. 86, (no. 3), pp. 241-51, Mar 2002.
[38] A. Papoulis, *Signal Analysis*: McGraw-Hill Company, 1977.
[39] T. Schreiber and A. Schmitz, "Surrogate time series," *Physica D*, vol. 142, (no. 3-4), pp. 346-382, 2000.
[40] J. Theiler, S. Eubank, A. Longtin, B. Galdrikian, and J. Farmer, "Testing for nonlinearity in time series: the method of surrogate data," *Physica D*, vol. 58, pp. 77-94, 1992.
[41] K.H. Chon, H. Zhao, R. Zou, and K. Ju, "Multiple time-varying dynamic analysis using multiple sets of basis functions," *IEEE Trans Biomed Eng*, vol. 52, (no. 5), pp. 956-60, May 2005.
[42] X. Wang and W.A. Cupples, "Interaction between nitric oxide and renal myogenic autoregulation in normotensive and hypertensive rats," *Can J Physiol Pharmacol*, vol. 79, (no. 3), pp. 238-45, Mar 2001.
[43] F.S. Wright and J. Schnermann, "Interference with feedback control of glomerular filtration rate by furosemide, triflocin, and cyanide," *J Clin Invest*, vol. 53, (no. 6), pp. 1695-708, Jun 1974.
[44] A. Just and W.J. Arendshorst, "Dynamics and contribution of mechanisms mediating renal blood flow autoregulation," *Am J Physiol Regul Integr Comp Physiol*, vol. 285, (no. 3), pp. R619-31, Sep 2003.
[45] T. Schreiber and A. Schmitz, "Improved surrogate data for nonlinearity tests," *Phys. Rev. Lett.*, vol. 77, (no. 4-22), pp. 635-638, 1996.
[46] D.J. Miller, N. Stergiou, and M.J. Kurz, "An improved surrogate method for detecting the presence of chaos in gait," *J Biomech*, Dec 1 2005.
[47] L. Faes, G.D. Pinna, A. Porta, R. Maestri, and G. Nollo, "Surrogate Data Analysis for Assessing the Significance of the Coherence Function," *IEEE transactions on bio-medical engineering*, vol. 51, (no. 7), pp. 1156-1166, 2004.
[48] A. Porta, R. Furlan, O. Rimoldi, M. Pagani, A. Malliani, and P.v.d. Borne, "Quantifying the strength of the linear causal coupling in closed loop interacting cardiovascular variability signals," *Biological Cybernetics*, vol. 86, (no. 3), pp. 241-251, 2004.
[49] L. Faes, G.D. Pinna, A. Porta, R. Maestri, and G. Nollo, "Surrogate data analysis for assessing the significance of the coherence function," *IEEE Trans Biomed Eng*, vol. 51, (no. 7), pp. 1156-66, Jul 2004.
[50] H. Zhao, W.A.Cupples, K. Ju, and K.H. Chon, "Time-varying causal coherence function and its application to renal blood pressure and blood fow data," *IEEE transactions on bio-medical engineering*, vol. 54, (no. 12), pp. 2142-2150, 2007.
[51] T. Schreiber and A. Schmitz, "Surrogate time series," *Physica. D*, vol. 142, pp. 346-382, 2000.

[52] M. Elstad, K. Toska, K.H. Chon, E.A. Raeder, and R.J. Cohen, "Respiratory sinus arrhythmia: opposite effects on systolic and mean arterial pressure in supine humans," *J Physiol*, vol. 536, (no. Pt 1), pp. 251-9, Oct 1 2001.
[53] "Heart rate variability. Standards of measurement, physiological interpretation, and clinical use. Task Force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology," *Eur Heart J*, vol. 17, (no. 3), pp. 354-81, Mar 1996.
[54] A.M. Bianchi, L.T. Mainardi, C. Meloni, S. Chierchia, and S. Cerutti, "Continuous monitoring of the sympatho-vagal balance through spectral analysis," *IEEE Eng Med Biol Mag*, vol. 16, (no. 5), pp. 64-73, Sep-Oct 1997.
[55] C.C. Yang and T.B. Kuo, "Assessment of cardiac sympathetic regulation by respiratory-related arterial pressure variability in the rat," *J Physiol*, vol. 515 (Pt 3), pp. 887-96, Mar 15 1999.
[56] A. Zaza and F. Lombardi, "Autonomic indexes based on the analysis of heart rate variability: a view from the sinus node," *Cardiovasc Res*, vol. 50, (no. 3), pp. 434-42, Jun 2001.
[57] Y. Zhong, H. Wang, K.H. Ju, K.M. Jan, and K.H. Chon, "Nonlinear analysis of the separate contributions of autonomic nervous systems to heart rate variability using principal dynamic modes," *IEEE transactions on bio-medical engineering*, vol. 51, (no. 2), pp. 255-62, Feb 2004.
[58] Chipcon, "Chipcon AS *SmartRF*^{R} CC2420 Preliminary Datasheet (rev 1.2)," 06-09 2004.
[59] "Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)," *IEEE std. 802.15.4*, 2003 2003.
[60] "MoteIV Corporation, "Tmote Sky: Quick Start Guide,""2005.
[61] R.P. Croke, K.V. Bulchandani, W.R. Jacobs, and H.S. Loeb, "Letter: Pseudoarrhythmia due to defective infusion pump and ECG monitor," *Jama*, vol. 235, (no. 7), pp. 705-6, Feb 16 1976.
[62] P.L. David Gay, David Culler, Eric Brewer, "NesC 1.1 Language Reference Manual," vol. May, 2003.
[63] D. Gay, P. Levis, R.v. Behren, M. Welsh, E. Brewer, and D. culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems," *http://nescc.sourceforge.net*, 2005.
[64] V. Kamat, "Pulse-Oximeter," *Indian J. Anaesth*, vol. 46, pp. 261-268, 2002.
[65] "MP506 Engineeing Product Specification," *Nellor Puritan Bennett*, vol. Rev. B/1, 2004.
[66] "MP506 Host Interface Specification," *Nellor Puritan Bennett*, 2001.
[67] R.C. Martin, "UML Tutorial: Finte State Machines," *http://www.objectmentor.com/resources/articles/umlfsm.pdf*, 1998.
[68] F. Wagner, R. Schmuki, T. Wagner, and P. Wolstenholme, *Modeling Software with Finite State Machines: APractical Approach*: CRC Press, 2006.
[69] "Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)," *IEEE std. 802.15.4*, pp. http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf, 2003 2003.
[70] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*: MIT Press, 1990.
[71] M.J. Atallah, *Algorithms and Theory of Computation Handbook*: CRC Press LLC, 1999.
[72] P.D. Welch, "The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms," *IEEE Trans. Audio Electroacoustics*, vol. AU-15, pp. 70-73, 1967.
[73] B. Schmidt, S. Asbach, O. Schweika, M. Zehender, C. Bode, and T.S. Faber, "Atrial fibrillation reduces the atrial impedance amplitude during cardiac cycle: a novel detection

algorithm to improve recognition of atrial fibrillation in pacemaker patients," *Europace*, vol. 9, (no. 9), pp. 812-6, Sep 2007.
[74] R.B. Shouldice, C. Heneghan, and P. de Chazal, "Automated detection of paroxysmal atrial fibrillation from inter-heartbeat intervals," *Conf Proc IEEE Eng Med Biol Soc*, vol. 1, pp. 686-9, 2007.
[75] Mathworks, "C/C++ Compiler Suite 2.1."
[76] Mathworks, "Matlab Complier Users' Guide," http://www.cc.gatech.edu/ccg/people/helene/matlab/compiler2.pdf.
[77] Moteiv, "Tmote Sky Datasheet."
[78] N.Y. Belova, S.V. Mihaylov, and B.G. Piryova, "Wavelet transform: A better approach for the evaluation of instantaneous changes in heart rate variability," *Auton Neurosci*, vol. 131, (no. 1-2), pp. 107-22, Jan 30 2007.
[79] J.F. Brosschot, E. Van Dijk, and J.F. Thayer, "Daily worry is related to low heart rate variability during waking and the subsequent nocturnal sleep period," *Int J Psychophysiol*, vol. 63, (no. 1), pp. 39-47, Jan 2007.
[80] E. Scharrer, H. Hessel, A. Kronseder, W. Guth, B. Rolinski, R.A. Jorres, K. Radon, R. Schierl, P. Angerer, and D. Nowak, "Heart rate variability, hemostatic and acute inflammatory blood parameters in healthy adults after short-term exposure to welding fume," *Int Arch Occup Environ Health*, vol. 80, (no. 4), pp. 265-72, Feb 2007.
[81] V. Vuksanovic and V. Gal, "Heart rate variability in mental stress aloud," *Med Eng Phys*, vol. 29, (no. 3), pp. 344-9, Apr 2007.
[82] A. Blasi, J.A. Jo, E. Valladares, R. Juarez, A. Baydur, and M.C. Khoo, "Autonomic cardiovascular control following transient arousal from sleep: a time-varying closed-loop model," *IEEE transactions on bio-medical engineering*, vol. 53, (no. 1), pp. 74-82, Jan 2006.
[83] G. Merati, M. Di Rienzo, G. Parati, A. Veicsteinas, and P. Castiglioni, "Assessment of the autonomic control of heart rate variability in healthy and spinal-cord injured subjects: contribution of different complexity-based estimators," *IEEE transactions on bio-medical engineering*, vol. 53, (no. 1), pp. 43-52, Jan 2006.
[84] F. Togo, K. Kiyono, Z.R. Struzik, and Y. Yamamoto, "Unique very low-frequency heart rate variability during deep sleep in humans," *IEEE transactions on bio-medical engineering*, vol. 53, (no. 1), pp. 28-34, Jan 2006.
[85] B. Pomeranz, R.J.B. Macaulay, M.A. Caudill, I. Kutz, D. Adam, D. Gordon, K.M. Kilborn, A.C. Barger, D.C. Shannon, R.J. Cohen, and H. Benson, "Assessment of autonomic function in humans by heart rate spectral analysis," *Am. J. Physiol. Heart Circ. Physiol*, vol. 248, (no. 1), pp. H151-H153, 1985.
[86] D.C. Randall, D.R. Brown, R.M. Raisch, J.D. Yingling, and W.C. Randall, "SA nodal parasympathectomy delineates autonomic control of heart rate power spectrum," *Am. J. Physiol. Heart Circ. Physiol*, vol. 260, (no. 3), pp. H985-H988, 1991.
[87] A. Skyschally, H.M. Breuer, and G. Heusch, "The analysis of heart rate variability does not provide a reliable measurement of cardiac sympathetic activity," *Clin. Sci.*, (no. 91), pp. 102-104, 1996.
[88] M.S. Houle and G.E. Billman, "Low-frequency component of the heart rate variability spectrum: a poor marker of sympathetic activity," *Am. J. Physiol. Heart Circ. Physiol*, vol. 276, (no. 1), pp. H215-H223, 1999.
[89] K.H. Chon, Y. Zhong, H. Wang, K. Ju, and K.M. Jan, "Separation of heart rate variability components of the autonomic nervous system by utilizing principal dynamic modes," *Nonlinear Dynamics Psychol Life Sci*, vol. 10, (no. 2), pp. 163-85, Apr 2006.
[90] Y. Zhong, Y. Bai, B. Yang, K. Ju, K. Shin, M. Lee, K.M. Jan, and K.H. Chon, "Autonomic nervous nonlinear interactions lead to frequency modulation between low-

and high-frequency bands of the heart rate variability spectrum," *Am J Physiol Regul Integr Comp Physiol*, vol. 293, (no. 5), pp. R1961-8, Nov 2007.
[91] T. Instruments, "MSP430x43x, MSP430x44x Mixed Signal Microcontroller Datasheet," 2004.
[92] "MSP430 IAR Embedded Workbench IDE User Guide."
[93] B. Eckel, *Thinking In C++*, Upper Saddle River, NJ: Prentice Hall, 2000.
[94] B. Stroustrup, *The C++ Programming Language*: Addison Wesley, Inc., 2000.
[95] "http://www.freertos.com/."
[96] "http://www.ghs.com/products/velosity.html."
[97] *MSP430x4xx Family User's Guide*: Texas Instruments, 2006.
[98] "http://www.daycounter.com/Calculators/MSP430-Uart-Calculator.phtml."
[99] I.K. Daskalov and Christov, II, "Electrocardiogram signal preprocessing for automatic detection of QRS boundaries," *Med Eng Phys*, vol. 21, (no. 1), pp. 37-44, Jan 1999.
[100] J.W. Lee, K.S. Kim, B. Lee, and M.H. Lee, "A real time QRS detection using delay-coordinate mapping for the microcontroller implementation," *Annals of biomedical engineering*, vol. 30, (no. 9), pp. 1140-51, 2002.
[101] Y. Chen and H. Duan, "A QRS Complex Detection Algorithm Based on Mathematical Morphology and Envelope," *Conf Proc IEEE Eng Med Biol Soc*, vol. 5, pp. 4654-7, 2005.
[102] X. Tian, C. Yan, Y. Yu, and T. Wang, "R-wave detection of ECG signal by using wavelet transform," *Sheng Wu Yi Xue Gong Cheng Xue Za Zhi*, vol. 23, (no. 2), pp. 257-61, Apr 2006.
[103] Q. Xiong, Z.X. Fang, H.L. Song, and X.M. Wu, "Method and implementation of real-time QRS-waves detection based on wavelet transform," *Zhongguo Yi Liao Qi Xie Za Zhi*, vol. 31, (no. 4), pp. 242-4, 270, Jul 2007.
[104] F. Zhang and Y. Lian, "Electrocardiogram QRS Detection Using Multiscale Filtering Based on Mathematical Morphology," *Conf Proc IEEE Eng Med Biol Soc*, vol. 1, pp. 3196-9, 2007.
[105] J. Pan and W.J. Tompkins, "A real-time QRS detection algorithm," *IEEE transactions on bio-medical engineering*, vol. 32, (no. 3), pp. 230-6, Mar 1985.
[106] J.W. Cookey and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, 1965.
[107] M. Barr and A. Massa, *Programming Embedded Systems with C and GNU Development Tools*, Sebastopol: O'Reilly Media, Inc., 2006.
[108] R. Abacherli, F. Braun, L. Zhou, M. Kraemer, J. Felblinger, and H.J. Schmid, "Electrocardiogram on a chip: overview and first experiences of an electrocardiogram manufacturer of medium size," *J Electrocardiol*, vol. 39, (no. 4 Suppl), pp. S36-40, Oct 2006.
[109] K. Lasanen and J. Kostamovaara, "A 1-v COMS preperocessing chip for ECG measurements," *Biomedical Circuits and Systems, IEEE International Workshop on*, pp. S1/2 - S1/4, 2004.
[110] "CHEFREN," *Aurelia Microeletronica S. p. A., http://www.Aurelia-micro.it*.
[111] "ECG ASIC," *Welch Allyn Company, http://www.welchallyn.com/*.
[112] N. O'Donoughue, S. Kulkarni, and D. Marzella, "Design and implementation of a framework for monitoring patients in hospitals using wireless sensors in ad hoc configuration," *Conf Proc IEEE Eng Med Biol Soc*, vol. 1, pp. 6449-52, 2006.
[113] M. Ackerman, R. Craft, F. Ferrante, M. Kratz, S. Mandil, and H. Sapci, "Telemedicine/telehealth: an international perspective. Telemedicine technology," *Telemed J E Health*, vol. 8, (no. 1), pp. 71-8, Spring 2002.
[114] A. Bianchi, G. Pinna, M. Croce, M.L. Rovere, R. Maestri, E. Locati, and S. Cerutti, "Estimation of the Respiratory Activity from Orthogonal ECG Leads," *Computers in Cardiology*, (no. 30), pp. 85-88, 2003.

[115] A. Travaglini, C. Lamberti, J.B. De, and M. Ferri, "Respiratory signal derived from eight-lead ECG," *Computers in Cardiology*, (no. 25), pp. 65-68, 1998.
[116] P.A. Leonard, J.G. Douglas, N.R. Grubb, D. Clifton, P.S. Addison, and J.N. Watson, "A fully automated algorithm for the determination of respiratory rate from the photoplethysmogram," *J Clin Monit Comput*, vol. 20, (no. 1), pp. 33-6, Feb 2006.
[117] L. Nilsson, A. Johansson, and S. Kalman, "Respiratory variations in the reflection mode photoplethysmographic signal. Relationships to peripheral venous pressure," *Med Biol Eng Comput*, vol. 41, (no. 3), pp. 249-54, May 2003.
[118] A. Johansson and P.A. Oberg, "Estimation of respiratory volumes from the photoplethysmographic signal. Part I: Experimental results," *Med Biol Eng Comput*, vol. 37, (no. 1), pp. 42-7, Jan 1999.
[119] A. Johansson and P.A. Oberg, "Estimation of respiratory volumes from the photoplethysmographic signal. Part 2: A model study," *Med Biol Eng Comput*, vol. 37, (no. 1), pp. 48-53, Jan 1999.
[120] N. Varma, B. Stambler, and S. Chun, "Detection of atrial fibrillation by implanted devices with wireless data transmission capability," *Pacing Clin Electrophysiol*, vol. 28 Suppl 1, pp. S133-6, Jan 2005.
[121] V. Kuhlkamp, V. Dornberger, C. Mewis, R. Suchalla, R.F. Bosch, and L. Seipel, "Clinical experience with the new detection algorithms for atrial fibrillation of a defibrillator with dual chamber sensing and pacing," *J Cardiovasc Electrophysiol*, vol. 10, (no. 7), pp. 905-15, Jul 1999.
[122] D. Cubanski, D. Cyganski, E.M. Antman, and C.L. Feldman, "A neural network system for detection of atrial fibrillation in ambulatory electrocardiograms," *J Cardiovasc Electrophysiol*, vol. 5, (no. 7), pp. 602-8, Jul 1994.
[123] M.R. Carnethon, R.J. Prineas, M. Temprosa, Z.M. Zhang, G. Uwaifo, and M.E. Molitch, "The association among autonomic nervous system function, incident diabetes, and intervention arm in the Diabetes Prevention Program," *Diabetes Care*, vol. 29, (no. 4), pp. 914-9, Apr 2006.
[124] M.R. Carnethon, S.H. Golden, A.R. Folsom, W. Haskell, and D. Liao, "Prospective investigation of autonomic nervous system function and the development of type 2 diabetes: the Atherosclerosis Risk In Communities study, 1987-1998," *Circulation*, vol. 107, (no. 17), pp. 2190-5, May 6 2003.
[125] D. Liao, J. Cai, F.L. Brancati, A. Folsom, R.W. Barnes, H.A. Tyroler, and G. Heiss, "Association of vagal tone with serum insulin, glucose, and diabetes mellitus--The ARIC Study," *Diabetes Res Clin Pract*, vol. 30, (no. 3), pp. 211-21, Dec 1995.
[126] M.A. Pfeifer, C.R. Weinberg, D.L. Cook, A. Reenan, J.B. Halter, J.W. Ensinck, and D. Porte, Jr., "Autonomic neural dysfunction in recently diagnosed diabetic subjects," *Diabetes Care*, vol. 7, (no. 5), pp. 447-53, Sep-Oct 1984.

APPENDIX

I. TVOPS Algorithm

The TV-ARMA process is represented by the following equation:

$$y(n)=\sum_{i=1}^{P}a(i,n)y(n-i)+\sum_{j=0}^{Q}b(j,n)x(n-j)+e(n) \tag{A-1}$$

where $a(i,n)$ and $b(j,n)$ are the time-varying AR and MA coefficients to be determined, respectively, and are functions of time. Indices P and Q are the maximum model orders of the AR and MA models, respectively. Variables $y(n)$ and $x(n)$ represent output and input signals, respectively. We expand the TV coefficients $a(i,n)$ and $b(j,n)$ onto a set of basis functions $\pi(n)$:

$$\begin{aligned} a(i,n)&=\sum_{k=0}^{V}\alpha(i,k)\pi_k(n)\\ b(j,n)&=\sum_{k=0}^{V}\beta(j,k)\pi_k(n) \end{aligned} \tag{A-2}$$

where $\alpha(i,k)$ and $\beta(j,k)$ represent the expansion parameters with V as the maximum number of basis sequences. Substituting Eq. (A-2) into Eq. (A-1), we obtain the following:

$$y(n)=\sum_{i=1}^{P}\sum_{k=0}^{V}\alpha(i,k)\pi_k(n)y(n-i)+\sum_{j=0}^{Q}\sum_{k=0}^{V}\beta(j,k)\pi_k(n)x(n-j)+e(n) \tag{A-3}$$

The next step is to select proper basis functions so that TV coefficients can be estimated. We have experimented with three different types of basis functions: Legendre polynomial, discrete prolate spheroidal sequence (DPSS), and Walsh. Our experience has shown that different basis functions show their own unique tractability and accuracy. Legendre polynomial and DPSS basis functions perform well if the coefficients are smoothly changing with time, for example, sinusoidal. Walsh functions, on the other hand, behave well for piece-wise stationary TV coefficients.

Once proper basis functions, $\pi(n)$, have been chosen, we create new variables:

$$\begin{aligned} y_k(n-i)&=\pi_k(n)y(n-i)\\ x_k(n-j)&=\pi_k(n)x(n-j) \end{aligned} \tag{A-4}$$

Substituting Eq. (A-4) into Eq. (A-3) results in the following expression:

$$y(n)=\sum_{i=1}^{P}\sum_{k=0}^{V}\alpha(i,k)y_k(n-i)+\sum_{j=0}^{Q}\sum_{k=0}^{V}\beta(j,k)x_k(n-j)+e(n) \tag{A-5}$$

Eq. (A-5) shows that the TV-ARMA model can now be considered to be a TIV-ARMA model since $\alpha(i,k)$ and $\beta(j,k)$ are not functions of time. Thus, the task simplifies to solving for parameters $\alpha(i,k)$ and $\beta(j,k)$, which can be more effectively estimated using the OPS algorithm [16].

The algorithm of the TVOPS follows the similar two-step procedure of OPS. There are two main differences, however. The first is that we arrange the pool of candidate vectors in the following matrix form:

$$\mathbf{M} = \left(\mathbf{Y_0(n-1)},\ldots,\mathbf{Y_V(n-1)},\mathbf{X_0(n)},\ldots,\mathbf{X_V(n)},\mathbf{Y_0(n-2)},\ldots,\mathbf{Y_V(n-2)},\ldots\ldots\right) \qquad \text{(A-6)}$$

where

$$\mathbf{Y_k(n-i)} = \left[\pi_k(N)y(N-i)),\cdots,\pi_k(2)y(2-i),\ \pi_k(1)y(1-i)\right]^{\mathrm{T}}$$
$$\mathbf{X_k(n-j)} = \left[\pi_k(N)x(N-j),\cdots,\pi_k(2)x(2-j),\pi_k(1)x(1-j)\right]^{\mathrm{T}}$$
$$k=0,\cdots,V; \quad i=1,\cdots,P; \quad j=0,\cdots,Q$$

Note that bold and unbold letters represent vectors (or matrices) and scalars, respectively. Among the pool of $P\times(V+1)+(Q+1)\times(V+1)$ candidate vectors, linearly independent candidate vectors are selected to form a new linearly independent vector pool such that:

$$\mathbf{W} = \left(\mathbf{w_1},\mathbf{w_2},\ldots,\mathbf{w_S}\right)$$

where S is the number of linearly-independent candidate vectors.

With the new candidate pool of linearly-independent vectors, least-squares analysis is performed:

$$\mathbf{Y} = \mathbf{W\Theta} + \mathbf{E} \qquad \text{(A-7)}$$

where $\mathbf{\Theta} = [\theta_1,\theta_2,\cdots,\theta_S]^{\mathrm{T}}$, $\mathbf{Y}$ is the signal vector and $\mathbf{E}$ is the prediction error vector.

In Eq. (A-7), θ_i is the coefficient estimate of the ARMA model. The objective is to minimize the equation error, $\mathbf{E}$, in the least-squares sense using the criterion function defined as follows:

$$J_N(\mathbf{\Theta}) = [\mathbf{Y - W\Theta}]^{\mathrm{T}}[\mathbf{Y - W\Theta}] \qquad \text{(A-8)}$$

The criterion function in Eq. (A-8) is quadratic in $\mathbf{\Theta}$, and can be minimized analytically with respect to $\mathbf{\Theta}$, yielding the following well-known equation:

$$\hat{\mathbf{\Theta}} = \left[\mathbf{W}^{\mathrm{T}}\mathbf{W}\right]^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{Y} \qquad \text{(A-9)}$$

The next stage of the algorithm involves further pruning of insignificant linearly-independent candidate vectors obtained from the first stage of the algorithm (Eqs. (A-4- A-6)). To determine which of the candidate vectors are significant, we calculate the following:

$$C_m = \frac{1}{N}\hat{\theta}_m{}^2\mathbf{w}_m^{\mathrm{T}}\mathbf{w}_m \qquad m = 1\sim S \qquad \text{(A-10)}$$

where S represents the number of linearly-independent candidate vectors and $\hat{\theta}_m$ represents the coefficient obtained from Eq. (A-9). Note that the vector $\mathbf{w}_m$ is calculated using Eq. (A-11):

$$\mathbf{w}_m = \left[\pi_k(N) y(N-i), \cdots, \pi_k(2) y(2-i), \pi_k(1) y(1-i) \right]^{\mathrm{T}}$$

or

$$\begin{aligned} &\mathbf{w}_m = \left[\pi_k(N) x(N-j), \cdots, \pi_k(2) x(2-j), \pi_k(1) x(1-j) \right]^{\mathrm{T}} \\ &k = 0,...,V; \;\; i = 1,...,P; \;\; j = 0,...,Q; \;\; n = 1,...,N; \;\; m = 1,...,S \end{aligned} \tag{A-11}$$

Note that the operation of Eq. (A-10) involves the adding of all C_m together for $\mathbf{w}_m$ with the same $y(n-i)$ or $x(n-i)$ but with different π_k. A magnitude value of each C_m, as estimated from Eq. (A-10), is then arranged in descending order. We only retain $\mathbf{w}_m$ terms that significantly reduce the estimation residuals. An approach that can be adopted to determine the significance of C_m terms is to calculate $(C_m - C_{m+1})/C_m$ and plot it as a function of m. The significant term can be characterized by the first maximum value in the plot. This operation has the added benefit of also discarding insignificant basis sequences, π_k, at the same time. The final procedure of the algorithm is to estimate ARMA model terms $\alpha(i,k)$ and $\beta(j,k)$ of Eq. (A-3) using the least-squares method and the calculation of TV coefficients $a(i,n)$ and $b(j,n)$ using Eq. (A-2).

II. SP-DSP Classes References

```
class ADC {
            unsigned short indexMCTL;
            void (*pCB)();
            unsigned short reading;
            static ADC* ADC_CH[ADC_MEMO];
            static Queue ADCQ;
            static bool Initialized;
      public:
            ADC();
            ~ADC();
            friend __interrupt void ADCISR(void);
            static void Initialize();
            bool Start()
            unsigned short Get()
            void SetCallback(void (*p)())
            bool SelectCH(const unsigned char& ch)
            static bool EnableCore()
            static bool DisableCore()
            static void SelectRef1_5v()
            static void SelectRef2_5v()
};
```

Class Member	*Description*
ADC::indexMCTL:	Value of the AD converter control register.
ADC::pCB:	The address of the interrupt service routine of the AD converter.
ADC::ADC_CH:	Map of ADC channel to ADC memory register.
ADC::Queue:	AD conversion requests queue.
ADC::Initialized:	ADC initialization status. Set to true after ADC is initialized.
ADC::ADC():	ADC constructor.
ADC::~ADC():	ADC destructor.
friend Initialize():	Initialize ADC12 convertor core.
ADC::Start():	Start an AD conversion.
ADC::Get():	Get the AD conversion readings from the associated memory register.
ADC::SetCallback():	Set the call back function when the AD reading is ready.
ADC::SelectCH():	Select an AD channel. Can be the analog input pins and internal channels.
ADC::EnableCore():	Enable the ADC12 convertor core.
ADC::Disable():	Disable the ADC12 convertor core.
ADC::SelectRef1_5v():	Select the on-chip 1.5 v reference voltage.
ADC::SelectRef2_5v():	Select the on-chip 2.5 v reference voltage.

```
class Buffer {
          unsigned char* data;
          unsigned short IndexR;
          unsigned short IndexW;
          unsigned short IndexP;
          unsigned short size;
          Buffer& operator=(const Buffer&);
     public:
          Buffer();
          Buffer(const unsigned short& sz);
          ~Buffer();
          unsigned short Read(unsigned char *p, const
          unsigned short& n);
          unsigned short Write(unsigned char *p, const
          unsigned short& n);
          unsigned short CopyTo(Buffer& destbuf);
          unsigned short CopyFrom(Buffer& srcbuf);
          unsigned short Size() const;
          bool SetSize(const unsigned short& sz);
          unsigned char Peek() const;
          unsigned char* Data();
          bool ReadByte(unsigned char& b);
          bool WriteByte(const unsigned char& b)
          const unsigned short& WritePos() const
          const unsigned short& WritePos(const unsigned
          short& pos);
          const unsigned short& ReadPos() const;
          const unsigned short& ReadPos(const unsigned
          short& pos);
}
```

Class Member	*Description*
`Buffer::data:`	Pointer to the memory of the data.
`Buffer::IndexR:`	Read-pointer of the buffer.
`Buffer::IndexW:`	Write-pointer of the buffer.
`Buffer::IndexP:`	Current operation position pointer of the buffer.
`Buffer::size:`	The size of the buffer.
`Buffer::operator=:`	Prevent "=" operation.
`Buffer::Buffer():`	Constructor, overloaded.
`Buffer::~Buffer():`	Destructor.
`Buffer::Read():`	Read `n` bytes from the buffer and store in memory `p`. `IndexR` is increased by `n`. Return value is how many are actually read.
`Buffer::Write():`	Write `n` bytes from the buffer from memory pointed by `p`. `IndexW` is increased by `n`. Return value is how many are actually written.
`Buffer::CopyTo():`	Copy the entire buffer to another buffer. Return value is how many are copied to.
`Buffer::CopyFrom():`	Copy the data from another buffer. Return value is how many are copied from.
`Buffer::Size():`	Return the size of the buffer.
`Buffer::SetSize():`	Set the size of the buffer.
`Buffer::Peek():`	Return the value at the current position indicated by `IndexP`.
`Buffer::Data():`	Return the pointer pointing to the data memory.
`Buffer::ReadByte():`	Read a single byte at `IndexR` position. `IndexR` is increased by 1.
`Buffer::WriteByte():`	Write a single byte at the `IndexW` position. `IndexW` is increased by 1.
`Buffer::WritePos():`	Return the value of `IndexW`. Overloaded version is to set the value of `IndexW`.
`Buffer::ReadPos():`	Return the value of `IndexR`. Overloaded version is to set the value of `IndexR`.

```
class CDTPMsg {
          unsigned char  nId;
          Buffer* pData;
          unsigned short wCSW;
     public:
          CDTPMsg();
          ~CDTPMsg();
          CDTPMsg(Buffer& buf);
          CDTPMsg& operator=(const CDTPMsg& src);
          unsigned char CDTP_ID(const unsigned char& id);
          unsigned char CDTP_ID() const;
          unsigned short CDTP_CSW(const unsigned short&
          csw);
          unsigned short CDTP_CSW() const;
          unsigned char Length() const;
          bool SetSize(const unsigned short& sz);
          bool ReadByte(unsigned char& b) const;
          bool WriteByte(const unsigned& b);
          unsigned short WritePos() const;
          unsigned short WritePos(const unsigned short&
          pos);
          unsigned short ReadPos() const;
          unsigned short ReadPos(const unsigned short& pos)
          Buffer* SetBuffer(Buffer& buf)
          Buffer* GetBuffer(Buffer* pbuf)
};
```

Class Member	*Description*
`CDTPMsg::nId:`	`ID` field of the C/D TP message.
`CDTPMsg::pData:`	Data field of the C/D TP message, stored in a `Buffer`.
`CDTPMsg::Wcsw:`	Channel Selection Word field of the C/D TP message.
`CDTPMsg::CDTPMsg():`	Constructor. Overloaded to create a C/D TP message from a `Buffer`.
`CDTPMsg::~CDTPMsg():`	Destructor.
`CDTPMsg::operator=:`	Create a new C/D TP message from another C/D TP message.
`CDTPMsg::CDTP_ID():`	Set the `CDTPMsg::nId` value. Overloaded version is to inquire this value.
`CDTPMsg::CDTP_CSW():`	Set the `CTMsg::wCSW` value. Overloaded version is to inquire this value.
`CDTPMsg::Length():`	Return the length of the C/D TP message.
`CDTPMsg::SetSize():`	Set the buffer size of the `CDTPMsg::pData`.
`CDTPMsg::ReadByte():`	Read a single byte form `CDTPMsg::pData`.
`CDTPMsg::WriteByte():`	Write a single byte to `CDTPMsg::pData`.
`CDTPMsg::WritePos():`	Return the write position of `CDTPMsg::pData`. Overloaded version is to set the write position of `CDTPMsg::pData`.
`CDTPMsg::ReadPos():`	Return the read position of `CDTPMsg::pData`. Overloaded version is to set the read position of `CDTPMsg::pData`.
`CDTPMsg::SetBuffer():`	Assign the `CDTPMsg::pData` to the source `Buffer`.
`CDTPMsg::GetBuffer():`	Return the value `CDTPMsg::pData` as a pointer to `Buffer`.

```
class DBuffer {
           Buffer *pbuf[2];
           unsigned char indexw, indexr;
           unsigned short size;
      public:
           DBuffer();
           DBuffer(unsigned short sz);
           ~DBuffer();
           void Swap();
           bool ReadByte(unsigned char& b);
           bool WriteByte(const unsigned& b);
           Buffer* ReadBuffer();
           Buffer* WriteBuffer();
};
```

Class Member	*Description*
`DBuffer::pbuf[2]:`	Pointers to two swappable Buffers.
`DBuffer::indexw:`	Index of the write-buffer.
`DBuffer::indexr:`	Index of the read-buffer.
`DBuffer::size:`	Size of the read- and write-buffer.
`DBuffer::DBuffer():`	Constructor. Overloaded version is to create a new `DBuffer` with the specified buffer size.
`DBuffer::~DBuffer():`	Destructor.
`DBuffer::Swap():`	Swap the two read- and write-buffer.
`DBuffer::ReadByte():`	Read one byte from the read-buffer.
`DBuffer::WriteByte():`	Write on byte to the write-buffer.
`DBuffer::ReadBuffer():`	Return a `Buffer` pointer pointing to the read-buffer.
`DBuffer::WriteBuffer():`	Return a `Buffer` pointer pointing to the write-buffer.

```
class FFT {
            short Re[NFFT];
            short Im[NFFT];
            short n;
            void DoFFT();
      public:
            FFT();
            ~FFT();
            short Size();
            void Setvalue(short* p, const int n);
            Task<FFT> *FFTTask()
}
```

Class Member	*Description*
`FFT::Re[]:`	Real part of the FFT result.
`FFT::Im[]:`	Imaginary part of the FFT result.
`FFT::n:`	FFT length.
`FFT::DoFFT():`	The function that actually performs the FFT calculation.
`FFT::FFT():`	Constructor.
`FFT::~FFT():`	Destructor.
`FFT::SetValue():`	Set the value of the `FFT::Re[]`. This function should be called to copy the data to the `FFT` object before the `FFT::DoFFT()` is executed.
`FFT::FFTTask():`	Return a `Task` pointer representing the particular `FFT` object. The returned pointer can be pushed into a `Task` queue.

```
class Queue {
            unsigned char size;
            unsigned char n;
            void **pList;
            unsigned char head, tail;
        public:
            Queue();
            Queue(const unsigned char sz);
            ~Queue();
            bool Initialize(const unsigned char& sz);
            unsigned char Size();
            unsigned char Head();
            unsigned char Tail();
            bool Push (void *p);
            void* Pop();
            void Delete(void *p);
            unsigned char& Count();
};
```

Class Member	*Description*
`Queue::size:`	Size of the queue.
`Queue::n:`	Number of elements in the queue.
`Queue::pList:`	Pointer pointing to the starting address of the memory that stores the pointers of the elements in the queue.
`Queue::head:`	The head position offset from `Queue::pList.`
`Queue::tail:`	The tail position offset from `Queue::pList`.
`Queue::Queue():`	Constructor. Overloaded version creates a new `Queue` with specified size.
`Queue::~Queue():`	Destructor.
`Queue::Initialize():`	Initialize the Queue with specified size.
`Queue::Size():`	Return the value of `Queue::size`.
`Queue::Hea():`	Return the value of `Queue::head`.
`Queue::Tail():`	Return the value of `Queue::tail`.
`Queue::Push():`	Push an element pointed by the augment pointer at the tail of the `Queue`.
`Queue::Pop():`	Pop out the element at the head position of the `Queue`. Return value is a pointer pointing to the popped element.
`Queue::Delete():`	Delete an arbitrary element specified by the argument pointer from the `Queue`. Deleting an element causes the elements after it move on position towards the head direction.
`Queue::Count():`	Return how many elements are currently in the `Queue`.

```
class BasicTask {
      public:
            virtual void Execute() = 0;
            virtual ~BasicTask() {};
};

template <class T>
class Task: public BasicTask {
      unsigned char state;
      T *pT;
      void (T::*pFunc)();
      public:
            Task()
            Task(void (T::*pf)())
            Task(const Task& tk)
            ~Task() {};
            void Set(T *p)
            void Set(T t)
            void Set(T *p, void(T::*pf)())
            void Set(T t, void(T::*p)())
            void Execute()
            void State(const unsigned char& st)
            const unsigned char& State()
};
```

Class Member	*Description*
`BasicTask::Execute()`:	Call this function to execute the task. This is a pure virtual function. So overloading from the derived class is mandatory.
`BasicTask::~BasicTask()`:	Virtual destructor.
`Task<T>::state`:	The state of the task. Should be one of: `TASK_STATE_POSTED`, `TASK_STATE_PENDING`, and `TASK_STATE_IDLE`.
`Task<T>::pT`:	Pointer to the object of `T`, for which the Task is created.
`Task<T>::pFunc`:	Function pointer pointing to a member function of `T` type.
`Task<T>::Task()`:	Constructor. Overloaded versions are to create a `Task` from a specified member function pointer, or from another `Task`.
`Task<T>::Set()`:	Set the value of `Task<T>::pT`. Overloaded versions take different types of arguments.
`Task::Execute()`:	Overloaded version of the virtual function in `BasicTask`.
`Task<T>::State()`:	Set the state of the task. Overloaded version is to inquire the task state.

```
class Timer {
            int interval;
            __no_init unsigned short volatile *pCTLx, *pCCRx;
            void (*pCB) ();
            static bool TAInited;
            static bool TBInited;
            static Timer* TAInt[TIMERA_NO];
            static Timer* TBInt[TIMERB_NO];
      public:
            friend __interrupt void TimerAISR(void);
            friend __interrupt void TimerBISR(void);
            static void InitializeTA();
            static void InitializeTB();
            Timer();
            ~Timer();
            void Start(const int& val);
            void Stop();
            void SetCallback(void (*p) ());
};
```

Class Member	*Description*
`Timer::interval:`	The expiration time interval of the Timer object.
`Timer::pCTLx:`	Pointer to the address of the Control Register of the hardware timer.
`Timer::pCCRx:`	Pointer to the address of the Compare/Capture Register.
`Timer::pCB:`	Function pointer pointing to the call back function (ISR-like) in response to the `Timer` object expiration.
`Timer::TAInited:`	Indicate if Timer_A has been initialized.
`Timer::TBInited:`	Indicate if Timer_B has been initialized.
`Timer::TAInt[]:`	The map between the Timer_A registers and the created Timer objects.
`Timer::TBInt[]:`	The map between the Timer_B registers and the created Timer objects.
`friend TimerAISR():`	Interrupt Service Routine of the Timer_A.
`friend TimerBISR():`	Interrupt Service Routine of the Timer_B.
`Timer::InitializeTA():`	Initialize Timer_A. `Timer::TAInited` is set after completion.
`Timer::InitializeTB():`	Initialize Timer_B. `Timer::TBInited` is set after completion.
`Timer::Timer():`	Constructor.
`Timer::~Timer():`	Destructor.
`Timer::Start():`	Start the `Timer` with specified time interval (in milliseconds).
`Timer::Stop():`	Stop the `Timer`.
`Timer::SetCallBack():`	Set the value of `Timer::pCB`.

```
class UART {
          static unsigned char port;
          static UART *pUART0, *pUART1;
          unsigned char port_id;
          void (*pRxCB) ();
          void (*pTxCB) ();
          unsigned char RxByte;
          volatile unsigned char *pTxBUF;
          unsigned char state;
          unsigned char ToSend;
          bool Escaped;
          const CDTPMsg* pSend;
          void SendCDTPMsg();
     public:
          UART();
          ~UART();
          friend __interrupt void UART0RxISR(void);
          friend __interrupt void UART0TxISR(void);
          friend __interrupt void UART1RxISR(void);
          friend __interrupt void UART1TxISR(void);
          friend void SendCDTPMsg();
          bool Bind(const unsigned char& p);
          void SetBaudrate(const int& br);
          void SetRxCallback(void (*p)());
          void Send(const unsigned char& txbyte);
          void SendCDTP(const CDTPMsg& msg);
          unsigned char Receive();
          void SetTxCallback(void (*p)())
          void EnableRx();
          void EnableTx();
          void DisableRx();
          void DisableTx();
};
```

Class Member	*Description*
UART::port:	Indicate the occupancies of USART0 and USART1. 0x00: both USARTs are not bound; 0x01: USART0 is bound; 0x20: USART1 is bound; 0x03: both USARTs are bound.
UART::pUART0:	Pointer to the hardware address that is assigned to the first UART object.
UART::pUART1:	Pointer to the hardware address that is assigned to the second UART object.
UART::pRxCB:	Function pointer pointing to the callback function responding to the Rx event of the UART object.
UART::pTxCB:	Function pointer pointing to the callback function responding to the Tx even of the UART object.
UART::RxByte:	The byte that is received by the UART object.
UART::pTxBUF:	Pointer pointing to the receive buffer of the UART object.
UART::state:	The state of Finite State Machine sending a C/D TP message.
UART::ToSend:	The byte needs to be sent out.
UART::Escaped:	Indicate if the byte being sent is reserved byte of the Peer-to-Peer Protocol. If yes, a 0x7e will be inserted into the outgoing byte stream.
UART::pSend:	The pointer pointing to the C/D TP message that needs to be sent.
UART::SendCDTPMsg():	Sending out a C/D TP message in background using FSM.
UART::UART:	Constructor.
UART::~UART():	Destructor.
friend UART0RxISR():	ISR of the USART0's Rx interrupt.
friend UART0TxISR():	ISR of the USART0's Tx interrupt.
friend UART1RxISR():	ISR of the USART1's Rx interrupt.
friend UART1TxISR():	ISR of the USART1's Tx interrupt.
UART::Bind():	Bind the UART object to a particular USART hardware port.
UART::SetBaudrate():	Set the baud rate of the UART object.
UART::SetRxCallback():	Set the value of UART::pRxCB.
UART::SetTxCallback():	Set the value of UART::pTxCB.
UART::Send():	Send out one single byte through the UART.
UART::SendCDTP():	Start the sending of a C/D TP message.
UART::Receive():	Read the received byte from the receive buffer and return it.
UART::EnableRx():	Enable the receiving function of the UART object.
UART::EnableTx():	Enable the transmitting function of the UART object.
UART::DisableRx():	Disable the receiving function of the UART object.
UART::DisableTx():	Disable the transmitting function of the UART object.

III. The 128-point Integer FFT Code

The FFT.h is listed here.

```
      /*   head file of FFT        */
      /*  by He Zhao               */
      /*  created on 9/29/2007     */
      /* last modified: 10/29/2007 */

#ifndef FFT_H
#define FFT_H
#include "Task.h"

#define NFFT  128

class FFT {
     private:
       short Re[NFFT];    // real part
       short Im[NFFT];    // imaginary part
       short n;

     public:
       FFT(): n(NFFT) {
           int i;
           for(i=0; i<NFFT; i++) {
               Re[i] = i;
               Im[i] = 0;
           }
       };

       ~FFT() {
         if(Re) delete []Re;
         if(Im) delete []Im;
       };

       short Size() {
           return(n);
       };

       void Setvalue() {
           int i;
           for(i=0; i<n; i++) {
               Re[i] = i;
               Im[i] = 0;
           }
       };

       void DoFFT();

       Task<FFT> *FFTTask() {
         Task<FFT> *p = new Task<FFT>;
         p->Set(this, &(FFT::DoFFT));
```

```
            return(p);
          }
};
#endif  // FFT_H
```

FFT.cpp file is listed as the following.

```
/****************************************/
/* by He Zhao                           */
/* Created:  9/29/2007                  */
/* Last modified: 10/30/2007            */
/****************************************/
#include "FFT.h"
// the following tables were built for 128-point FFT
// change of FFT point needs to change these tables as well
const unsigned char bit_rev[] = {0, 64, 32, 96, 16, 80, 48, 112,
8, 72, 40, 104, 24, 88, 56, 120, 4, 68, 36, 100, 20, 84, 52, 116,
12, 76, 44, 108, 28, 92, 60, 124, 2, 66, 34, 98, 18, 82, 50, 114,
10, 74, 42, 106, 26, 90, 58, 122, 6, 70, 38, 102, 22, 86, 54,
118, 14, 78, 46, 110, 30, 94, 62, 126, 1, 65, 33, 97, 17, 81, 49,
113, 9, 73, 41, 105, 25, 89, 57, 121, 5, 69, 37, 101, 21, 85, 53,
117, 13, 77, 45, 109, 29, 93, 61, 125, 3, 67, 35, 99, 19, 83, 51,
115, 11, 75, 43, 107, 27, 91, 59, 123, 7, 71, 39, 103, 23, 87,
55, 119, 15, 79, 47, 111, 31, 95, 63, 127 };

const short W_re[] = {4095, 0, 2896, -2896, 3784, -1567, 1567, -
3784, 4017, -799, 2276, -3406, 3406, -2276, 799, -4017, 4076, -
401, 2598, -3166, 3612, -1931, 1189, -3920, 3920, -1189, 1931, -
3612, 3166, -2598, 401, -4076, 4091, -201, 2751, -3035, 3703, -
1751, 1380, -3857, 3973, -995, 2106, -3513, 3290, -2440, 601, -
4052, 4052, -601, 2440, -3290, 3513, -2106, 995, -3973, 3857, -
1380, 1751, -3703, 3035, -2751, 201, -4091 };

const short W_im[] = {0, -4095, -2896, -2896, -1567, -3784, -
3784, -1567, -799, -4017, -3406, -2276, -2276, -3406, -4017, -
799, -401, -4076, -3166, -2598, -1931, -3612, -3920, -1189, -
1189, -3920, -3612, -1931, -2598, -3166, -4076,-401, -201, -4091,
-3035, -2751, -1751, -3703, -3857, -1380, -995, -3973, -3513, -
2106, -2440, -3290, -4052, -601, -601, -4052, -3290, -2440, -
2106, -3513, -3973, -995, -1380, -3857, -3703, -1751, -2751, -
3035, -4091, -201 };

enum {
      FFT_DIGIT = 12,  // W_re and W_im accuracy
      FFT_HALFSCALE = 1 << (FFT_DIGIT - 1)
};

void FFT::DoFFT(){
      long retmp, imtmp, wretmp, wimtmp;
      short i, j, k, m, p, q, Sum_re, Sum_im;
```

```
    for (m=n; m>=2; m=m>>1){
        j = m >> 1;
        for (p=0,k=0; p<n; p+=m){
            wretmp = W_re[k];
            wimtmp = W_im[k];
            for (q=p; q<p+j; q++){
                retmp = wretmp*Re[q+j] - wimtmp*Im[q+j];
                if(retmp > 0)
                    retmp = (retmp + FFT_HALFSCALE)>>FFT_DIGIT;
                else
                    retmp = retmp >> FFT_DIGIT;
                imtmp = wretmp*Im[q+j] + wimtmp*Re[q+j];
                if(imtmp > 0)
                    imtmp = (imtmp + FFT_HALFSCALE)>>FT_DIGIT;
                else
                    imtmp = imtmp >> FFT_DIGIT;
                Sum_re = Re[q];
                Sum_im = Im[q];
                Re[q] = Sum_re + (short)retmp;
                Im[q] = Sum_im + (short)imtmp;
                Re[q+j] = Sum_re - (short)retmp;
                Im[q+j] = Sum_im - (short)imtmp;
            }
            ++k;
        }
    }

    for (i=0; i<n; i++){
         if (bit_rev[i] <= i) continue;
         Sum_re = Re[i];
         Sum_im = Im[i];
         Re[i]= Re[bit_rev[i]];
         Im[i]= Im[bit_rev[i]];
         Re[bit_rev[i]]= Sum_re;
         Im[bit_rev[i]]= Sum_im;
    }
}
```