

CUSTOM BIOMEDICAL SENSORS FOR APPLICATION IN WIRELESS BODY AREA
NETWORKS AND MEDICAL DEVICE INTEGRATION FRAMEWORKS

by

KEJIA LI

M.S., Kansas State University, 2010

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Electrical & Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2012

Abstract

The U.S. health care system is one of the most advanced and costly systems in the world. The health services supply/demand gap is being enlarged by the aging population coupled with shortages in the traditional health care workforce and new information technology workers. This will not change if the current medical system adheres to the traditional hospital-centered model. One promising solution is to incorporate patient-centered, point-of-care test systems that promote proactive and preventive care by utilizing technology advancements in sensors, devices, communication standards, engineering systems, and information infrastructures.

Biomedical devices optimized for home and mobile health care environments will drive this transition. This dissertation documents research and development focused on biomedical device design for this purpose (including a wearable wireless pulse oximeter, motion sensor, and two-thumb electrocardiograph) and, more importantly, their interactions with other medical components, their supporting information infrastructures, and processing tools that illustrate the effectiveness of their data. The GumPack concept and prototype introduced in Chapter 2 addresses these aspects, as it is a sensor-laden device, a host for a local body area network (BAN), a portal to external integration frameworks, and a data processing platform. GumPack sensor-component design (Chapters 3 and 4) is oriented toward surface applications (e.g., touch and measure), an everyday-carry form factor, and reconfigurability. Onboard tagging technology (Chapters 5 and 6) enhances sensor functionality by providing, e.g., a signal quality index and confidence coefficient for itself and/or next-tier medical components (e.g., a hub).

Sensor interaction and integration work includes applications based on the GumPack design (Chapters 7 through 9) and the Medical Device Coordination Framework (Chapters 10 through 12). A high-resolution, wireless BAN is presented in Chapter 8, followed by a new physiological use case for pulse wave velocity estimation in Chapter 9. The collaborative MDCF work is transitioned to a web-based Hospital Information Integration System (Chapter 11) by employing database, AJAX, and Java Servlet technology. Given the preceding sensor designs and the availability of information infrastructures like the MDCF, medical platform-oriented devices (Chapter 12) could be an innovative and efficient way to design medical devices for hospital and home health care applications.

CUSTOM BIOMEDICAL SENSORS FOR APPLICATION IN WIRELESS BODY AREA
NETWORKS AND MEDICAL DEVICE INTEGRATION FRAMEWORKS

by

KEJIA LI

M.S., Kansas State University, 2010

A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Electrical & Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2012

Approved by:

Major Professor
Steve Warren

Copyright

KEJIA LI

2012

Abstract

The U.S. health care system is one of the most advanced and costly systems in the world. The health services supply/demand gap is being enlarged by the aging population coupled with shortages in the traditional health care workforce and new information technology workers. This will not change if the current medical system adheres to the traditional hospital-centered model. One promising solution is to incorporate patient-centered, point-of-care test systems that promote proactive and preventive care by utilizing technology advancements in sensors, devices, communication standards, engineering systems, and information infrastructures.

Biomedical devices optimized for home and mobile health care environments will drive this transition. This dissertation documents research and development focused on biomedical device design for this purpose (including a wearable wireless pulse oximeter, motion sensor, and two-thumb electrocardiograph) and, more importantly, their interactions with other medical components, their supporting information infrastructures, and processing tools that illustrate the effectiveness of their data. The GumPack concept and prototype introduced in Chapter 2 addresses these aspects, as it is a sensor-laden device, a host for a local body area network (BAN), a portal to external integration frameworks, and a data processing platform. GumPack sensor-component design (Chapters 3 and 4) is oriented toward surface applications (e.g., touch and measure), an everyday-carry form factor, and reconfigurability. Onboard tagging technology (Chapters 5 and 6) enhances sensor functionality by providing, e.g., a signal quality index and confidence coefficient for itself and/or next-tier medical components (e.g., a hub).

Sensor interaction and integration work includes applications based on the GumPack design (Chapters 7 through 9) and the Medical Device Coordination Framework (Chapters 10 through 12). A high-resolution, wireless BAN is presented in Chapter 8, followed by a new physiological use case for pulse wave velocity estimation in Chapter 9. The collaborative MDCF work is transitioned to a web-based Hospital Information Integration System (Chapter 11) by employing database, AJAX, and Java Servlet technology. Given the preceding sensor designs and the availability of information infrastructures like the MDCF, medical platform-oriented devices (Chapter 12) could be an innovative and efficient way to design medical devices for hospital and home health care applications.

Table of Contents

List of Figures	x
List of Tables	xvii
Acknowledgements.....	xviii
Chapter 1 - Introduction.....	1
Custom Biomedical Sensors	2
Wireless Body Area Networks	5
Medical Device Integration Frameworks	6
Chapter 2 - GumPack Platform.....	9
Goal and Objectives.....	10
Technology Overview.....	11
GumPack Concept Model.....	13
Chapter 3 - Surface Component Design	15
Processor SC	15
Motion Sensor SC.....	17
Electrocardiograph SC.....	19
ZigBee Coordinator SC	21
Generic Surface Component.....	23
Chapter 4 - Pulse Oximeter Sensor Design.....	26
Requirements and Device Layout.....	27
AC Extraction and Drift Resistance.....	30
Closed-Loop System.....	33
Removable Noise	34
Motion Artifact	35
MATLAB Interface	37
Device Prototype.....	38
Measurement Data	40
Chapter 5 - Onboard Tagging Technology.....	48
Technology Overview.....	49

Onboard and Real-Time Tags.....	50
Tag Type and Content.....	50
Tag Format and Indices	52
Chapter 6 - Application A: Feature Detection on a Pulse Oximeter.....	54
Algorithm Flow.....	56
Feature Selection.....	60
Observations and Hierarchical Decision-Making.....	65
Communication Protocol and MATLAB Interface	71
Downsampling and Compact Representation.....	72
Feature-Extraction and Decision-Making Performance	74
Chapter 7 - Sensor Integration and Interaction.....	76
Surface Substrate and Interconnection Interface	76
Cuboid Assembly.....	77
Software Architecture.....	79
GumPack Component Connection Framework	81
Chiclet Network and Device Interoperability Standards	84
Web Server and User Interface.....	86
Prototype and Initial Results.....	87
Chapter 8 - High Resolution WBAN.....	91
IEEE 802.15.4 MAC Layer	91
Issue 1: Lost and Wandering Frames.....	91
Issue 2: Timeline Distortion	92
Frame Field Definition.....	93
Timeline Restoration in MATLAB.....	94
Chapter 9 - Application B: Pulse Wave Velocity Estimation.....	96
Study Description and Experimental Setup	97
Basic Approach.....	98
Signal Synchronization	98
Signal Preprocessing.....	100
Pulse Travel Time Extraction	102
Count Differences	103

Pulse Wave Velocity Estimation	103
WBAN Approach	105
Decomposition of the Sensor Data Streams.....	105
Timeline-Recovered Waveforms	106
Defragmentation and Curve Fitting	106
Synchronization and Feature Extraction.....	108
Chapter 10 - Medical Device Coordination Framework.....	109
JMS and MDCF Architecture.....	109
Scenario Creation and Installation	111
Medical Device Class	113
Transformer Class.....	114
Display Class	117
Chapter 11 - Application C: HIIS	120
From MDCF to HIIS.....	120
ActiveMQ, AJAX, and MySQL Database.....	122
HIIS Interface and Consoles.....	127
Scenario Creation in the HIIS.....	131
Scenario APP Configuration and Execution in the HIIS.....	133
Running the HIIS on a Mobile Platform.....	137
Chapter 12 - Medical Platform-Oriented Device.....	139
Moving from Conventional Medical Devices to MPODs	139
Object-Oriented Model	142
APP Logic Pool	145
APP Library	147
Case 1: Parallel Processing.....	149
Case 2: Multiple Devices	150
Case 3: Closed-Loop Behavior	151
Chapter 13 - Conclusion	152
References.....	156
Appendix A – GumPack Hardware Interconnection Interface.....	164
Appendix B – GumPack Hardware Schematics and PCB Layouts	166

Appendix C – Screenshots of the GumPack Web Interface	171
Appendix D – Instructions to Install the JRE on the GumPack.....	176
Appendix E – Exporting the MDCF to the GumPack	179
Appendix F – Acronyms and Abbreviations	184

List of Figures

Figure 2.1. A prototype GumPack design and demonstrative Web interface (closest surface: single-lead electrocardiograph).....	11
Figure 2.2. GumPack cuboid conceptual model.	13
Figure 2.3. Wireless network architecture of a GumPack.	14
Figure 3.1. Gumstix Overo board (left) and custom wireless reflectance pulse oximeter (right).	16
Figure 3.2. Motion sensor surface component (top view).	18
Figure 3.3. Motion sensor surface component (bottom view) with two 70-pin connectors.	18
Figure 3.4. ECG surface component (top and bottom view).	20
Figure 3.5. ECG surface component with commercial electrodes.....	21
Figure 3.6. ZigBee coordinator surface component (top view).	22
Figure 3.7. ZigBee coordinator surface component (bottom view).	22
Figure 3.8. Diagram of a generic surface component (GSC) and the interface to the GumPack Component Connection Framework (GCCF).	24
Figure 3.9. Infusion pump design using GSC model fitting.	25
Figure 4.1. Circuit-level system layout. Coordinated by the microcontroller, signal baselines are digitally extracted as an alternative to conventional filtering.	29
Figure 4.2. A differential amplifier with gain G compares the first-stage PPG (S_1) to a DC reference voltage to obtain the second-stage PPG (S_2).	31
Figure 4.3. Palm PPG data before (blue) and after (red) compensation.	32
Figure 4.4. Pulse oximeter control flow that illustrates how the first-stage PPG (S_1) can be used to create the second-stage PPG (S_2), where both signals provide feedback to stabilize the acquisition process.	33
Figure 4.5. An example of removable noise: (a) compensated PPG corrupted by ambient noise and (b) frequency spectrum of these data sampled at 240 Hz.	35
Figure 4.6. (a) PPG severely corrupted by hand motion along three axes. (b) Frequency spectrum of the 28 seconds of data sampled at 100 Hz.	36
Figure 4.7. Three uncompensated PPGs acquired at $f_s = 240$ Hz under similar slight-motion conditions but with different parameter pairs (W, A).	37

Figure 4.8. Pulse oximeter MATLAB GUI. In this example, a series of calibration coefficients (lower right) is extracted from the current data (upper right).....	38
Figure 4.9. Top view of the wireless reflectance pulse oximeter.	39
Figure 4.10. Bottom view of the wireless reflectance pulse oximeter.....	39
Figure 4.11. Fingertip results: 25 seconds of near-infrared PPG data.....	41
Figure 4.12. Fingertip results: 25 seconds of red PPG data.....	42
Figure 4.13. Fingertip results: 25 seconds of near-infrared magnitude spectra.....	42
Figure 4.14. Fingertip results: 25 seconds of red magnitude spectra.....	43
Figure 4.15. Fingertip signal processing and digital volume pulse (DVP) analysis.....	44
Figure 4.16. Wrist PPGs corresponding to the placement locations in Figure 4.17.....	45
Figure 4.17. Pulse oximeter measurement locations on the left wrist.	45
Figure 4.18. Earlobe results: (a) near-infrared channel and (b) red channel.	46
Figure 4.19. Temple results: (a) time-domain PPG with respiration and swallowing motion and (b) the corresponding frequency-domain spectrum.	47
Figure 5.1. Tags embedded in an original data stream.	49
Figure 5.2. Data frame structure for a custom pulse oximeter that employs serial communication (R: red channel; IR: near-infrared channel; AC & DC: pulsatile and baseline samples).	52
Figure 5.3. Tag frame structure for a custom pulse oximeter. The frame length is N bytes containing K tags.....	53
Figure 6.1. Feature detection algorithm flow. The wireless reflectance pulse oximeter appears on the left.	57
Figure 6.2. Triangular structures identified for the “compact representation.”	58
Figure 6.3. Feature 1 (baseline variation count) statistics for 20 subjects.....	61
Figure 6.4. Feature 2 (rising count) statistics for 20 subjects.	63
Figure 6.5. Feature 3 (falling count) statistics for 20 subjects.....	63
Figure 6.6. Feature 4 (saturation index) statistics for 20 subjects.	64
Figure 6.7. Ratio of falling time to rising time statistics for 20 subjects.....	65
Figure 6.8. Three-step hierarchical decision-making approach.....	66
Figure 6.9. Histograms of Observation 3 for valid PPG data.	67
Figure 6.10. Histograms of Observation 3 for invalid PPG data.	68

Figure 6.11. Histograms of (a) Observation 1 for no motion and (b) Observation 2 for no saturation.....	70
Figure 6.12. Data frame structure used in wireless and serial communication.....	71
Figure 6.13. Tag frame structure used in wireless and serial communication.....	72
Figure 6.14. MATLAB interface developed to visualize the feature detection algorithm results. All algorithm-related parameters and decisions are ascertained on the wireless unit prior to transmission.....	72
Figure 6.15. Time-domain depiction of (a) an original PPG, (b) its downsampled representation, and (c) its “compact representation.”.....	73
Figure 6.16. A typical data set consisting of 33 three-second segments.....	74
Figure 6.17. Four features extracted from the 33 segments.....	75
Figure 7.1. Surface substrate circuit board (fully populated version).....	76
Figure 7.2. SolidWorks design of the GumPack inner cuboid.....	78
Figure 7.3. Machined GumPack inner cuboid.....	78
Figure 7.4. GumPack software architecture.....	80
Figure 7.5. Profile memory chip field definitions.....	83
Figure 7.6. GCCF finite state machine (FSM).....	84
Figure 7.7. GumPack Chiclet network and medical device interoperability.....	85
Figure 7.8. GumPack web server login page.....	86
Figure 7.9. GumPack prototype and Chiclets.....	87
Figure 7.10. Caspa camera board alongside a motion sensor SC.....	88
Figure 7.11. Unfiltered ECG sampled from an ECG SC.....	89
Figure 7.12. Frequency spectrum of the unfiltered ECG.....	90
Figure 8.1. An example star-topology network. Each frame sent by a sensor node carries a sequence # to address the issue of lost and wandering frames.....	92
Figure 8.2. Ticket production and consumption mechanism inside the Coordinator to address the timeline distortion issue.....	93
Figure 8.3. Frame field definitions for the Coordinator and sensor nodes on the application layer of the custom communication protocol.....	93

Figure 8.4. Original data stream with sequence #s in the received order (blue crosses) and the recovered time-aligned sequence (red dots). Placeholder frames are assigned negative sequence #s.	94
Figure 8.5. Linear regression – relative delay to recover the distorted timeline.	95
Figure 9.1. Measurement setup with one pulse oximeter (sensor) at the wrist and the other at the fingertip. The primary arterial tree of the hand is depicted to illustrate the pulse travel path. This figure is adapted from [88].	98
Figure 9.2. A MATLAB interface that simultaneously communicates with two pulse oximeters on different serial ports.	99
Figure 9.3. Punched raw PPG waveforms that are aligned (synchronized).....	100
Figure 9.4. Synchronized raw PPG waveforms with the punch marks removed.	101
Figure 9.5. Filtered PPG waveforms. The channels are still synchronized due to the same group delay imposed by the linear-phase lowpass filter.	101
Figure 9.6. First derivative (b) and second derivative (c) of a pair of filtered single-cycle PPGs (a). Three delays, d1, d2, and d3, are extracted according the count differences represented by the line at $y = 0$ in (b) and (c).....	102
Figure 9.7. The original frame sequence decomposed into two PPGs. Each frame carries two AC values, but only one is plotted here.....	106
Figure 9.8. Two PPGs with data points time-aligned.	107
Figure 9.9. Representative waveform for eight PPG cycles worth of discontinuous, discrete data.	107
Figure 9.10. Statistically synchronized, continuous PPGs from two sensor nodes.	108
Figure 10.1. MDCF architecture and example APP virtual machine (lower right).....	111
Figure 10.2. MDCF Development Environment (upper left), auto-generated files (upper right), and MDCF Administrative Console (lower right).	112
Figure 10.3. Snapshot of a Java code segment for the MD class in a pulse oximeter scenario..	113
Figure 10.4. Snapshot of a Java code segment for the MD class with a nested MDP class.	114
Figure 10.5. Java classes GenericDeviceMessage (left) and GUIMessage (right).....	115
Figure 10.6. The Java code segment for the Transformer class that describes the main transformation procedure.	115
Figure 10.7. PPG waveform visualized in a Display class.	117

Figure 10.8. Numeric physiological values.	118
Figure 10.9. Instantiation of a GenericDeviceMessage (left, partial content) and its corresponding GUIMessage (right, full content) processed by the Feature Detection transformer.	118
Figure 10.10. Numeric features visualized in a Display class.	119
Figure 10.11. Physiological values influenced by feature detection results.	119
Figure 11.1. MDCF architecture based on the MOM model and the publish-subscribe mechanism.	120
Figure 11.2. HIIS architecture utilizing ActiveMQ and AJAX.	121
Figure 11.3. HIIS architecture improved by automatic transformer assignment and a mobile device interface.	122
Figure 11.4. ActiveMQ startup information.	126
Figure 11.5. ActiveMQ WebConsole with topic information.	127
Figure 11.6. HIIS homepage with patient information, user identity, and administration tools.	128
Figure 11.7. HIIS homepage with medical device information.	128
Figure 11.8. HIIS homepage with clinical scenario information.	129
Figure 11.9. Patient record management console in the HIIS.	130
Figure 11.10. Scenario management console in the HIIS.	130
Figure 11.11. Scenario creation console in the HIIS.	131
Figure 11.12. Creation of an example scenario in the HIIS Scenario Creation Console.	132
Figure 11.13. Automatically created scenario script and Java code templates in the HIIS Scenarion Creation Console.	132
Figure 11.14. Upload of a new scenario to the server in the HIIS Scenario Creation Console.	133
Figure 11.15. Example patient console in the HIIS that displays patient information, connected medical devices, and active scenarios.	134
Figure 11.16. Selection and assignment of medical devices and components in the patient console.	135
Figure 11.17. Sequence to configure and start a scenario for a patient in the patient console.	136
Figure 11.18. APP display for the button and infusion pump scenario in the HIIS.	137
Figure 11.19. Access to the HIIS with a mobile device.	138
Figure 11.20. Patient scenario display on a mobile device.	138
Figure 12.1. Elements of a conventional medical device.	140

Figure 12.2. A custom reflectance pulse oximeter (left) and two representative PPGs acquired with the device (right).....	142
Figure 12.3. Object-oriented model for device-related MDCF components.....	143
Figure 12.4. An example component pool for PPG-oriented APPs. Transformer components are depicted as colored bubbles in the middle area (interconnections will be configured for specific clinical scenarios).....	148
Figure 12.5. System-level diagram for a pulse oximeter APP that employs parallel transformers.....	149
Figure 12.6. System-level diagram for a pulse wave velocity APP that employs multiple pulse oximeters (devices).....	150
Figure 12.7. System-level diagram for a cardiovascular parameter extraction APP that exhibits closed-loop behavior.....	151
Figure C.1. GumPack web interface - Dashboard tag. Data acquired from an accelerometer, gyroscope, pulse oximeter, and ECG sensor.....	171
Figure C.2. GumPack web interface - Devices tag. Data selection and analysis for a pulse oximeter.....	172
Figure C.3. GumPack web interface - Trackers tag.....	173
Figure C.4. GumPack web interface - File Repository page.....	174
Figure C.5. GumPack system resources allocated for the web server lighttpd.....	175
Figure D.1 Configuration file for Wi-Fi access.....	177
Figure D.2. Wireless interface configuration after connecting to the access point successfully.....	177
Figure E.1. Example scenario with an MDCF client running on the GumPack.....	179
Figure E.2. Exporting resources to a JAR file in Eclipse.....	180
Figure E.3. Choosing the launch configuration and file properties in the last step of a runnable JAR file export.....	181
Figure E.4. Original code in the MDCF when receiving a message regarding a publish topic..	182
Figure E.5. Modified code in the GumPack when receiving a message regarding a publish topic.....	182
Figure E.6. Original code in the MDCF when receiving a message regarding a subscribe topic.....	183

Figure E.7. Modified code on the GumPack when receiving a message regarding a subscribe
topic..... 183

List of Tables

Table 4.1. Wireless Reflectance Pulse Oximeter Design Requirements	28
Table 5.1. Tag Types Used in Onboard Tagging Technology.....	51
Table 6.1. Observation Description Given the Original Four Features	65
Table 6.2. Hypothesis Definition and P_d for Each Step.....	69
Table 6.3. Hierarchical Decision Rule Results	75
Table 9.1. PWV Results from a Ten-Second PPG Waveform Pair from One Subject.....	104
Table 10.1. Features Extracted from PPGs.....	116
Table A.1. GumPack Hardware Interface Pin Definition.....	164
Table F.1. Acronyms and Abbreviation.....	184

Acknowledgements

The author wishes to thank Dr. Steve Warren, Dr. Bala Natarajan, and Dr. John Hatcliff for their helpful advice and assistance throughout this program. This work was supported in part by the National Science Foundation under grants BES–0093916, BES–0440183, and CNS–0551626, as well as the 2011 CIMIT (Center for Integration of Medicine & Innovative Technology) prize for primary healthcare. Opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF or CIMIT.

Chapter 1 - Introduction

The U.S. medical system offers many of the most advanced technologies, treatments, and therapies in the world. It is also one of the main cradles for innovation in health care, from biotechnology to medical devices and systems, under the supervision of the U.S. Food and Drug Administration (FDA). On the other hand, this system is one of the most costly health care systems in the world. The average annual premium for family medical coverage through an employer reached \$15,073 in 2011, an increase of 9 percent relative to the prior year, according to a new study by the Kaiser Family Foundation [1]. The evolving aging population, driven by the wave of baby boomers that approach retirement [2], will impose significant pressure on a medical system with already crowded hospital rooms and long waiting lists for surgeries such as heart transplants. Increasing demands from more patients for more care services, coupled with shortages in the healthcare workforce, could eventually pose a crisis for current medical systems. Regarding physicians and nurses, the nation is looking at a future shortage of about 300,000 nurses and 35,000 to 44,000 adult-care generalist physicians practicing family medicine and general internal medicine [3]. Health information technology (HIT) workers present the other significant shortage, as technology tools such as electronic health records (EHRs) and health information exchanges (HIEs) begin to be widely adopted in the health care industry [4]. An additional 35,000 HIT workers are needed by 2018, as reported by the Bureau of Labor Statistics.

The healthcare system needs a transformation, given the long-unchanged and disappointing situation of rising medical care costs and rapidly increasing demands that exceed health care supplies. One feasible solution is to utilize technology advancements in sensors, devices, communication standards, engineering systems, information infrastructures, etc. and transit to bottom-to-top approaches that rely on the large consumer base. In order to realize a patient-centric, point-of-care system that focuses on proactive and preventive care, consumer medical devices should penetrate into the large user space, providing the infrastructure for a necessary healthcare transition.

There will be a 'healthcare war' in the near future involving all of us. To win this war, the 'technology weapons' need to be placed into the hands of the largest group affected by this war:

the patients and individuals who are concerned about their health and wellbeing. Hospitals should not be the only places where medical care is provided. Instead, the health care system should be distributed across homes or travel with individuals (i.e., mobile health care). Technology will help realize this transition by bringing medical devices and systems to the consumer and interconnecting them with the hospital information infrastructure.

Smart phones, as consumer digital devices, have already been vastly integrated into people's daily lives. Mobile health applications have begun to incorporate smart phone platforms, utilizing their increasing processing speed, friendly user interfaces, and communication capabilities. Efforts include (a) attaching biomedical sensors directly or indirectly to a smart phone to track a user's physiological indices, (b) using the components already integrated inside the phone, such as accelerometers and cameras, for medical purposes, and (c) the creation of a variety of health-related APPs. AliveCor's iPhone ECG case turns a phone into an ECG device by sliding a case with ECG electrodes onto the back of the phone [5]. Imec and Holst Centre, together with TASS, developed a wireless receiver on an Android phone to form a body area network (BAN) to monitor ECG, EEG, EMG, etc [6]. The integrated camera on the phone has been proven to obtain accurate heart rate measurements [7] and other physiological variables such as respiratory rate and oxygen saturation by analyzing PPG signals [8]. Thousands of APPs can be downloaded from APP stores in the categories of medicine, health, and fitness [9].

Custom Biomedical Sensors

The creation of consumer medical devices (e.g., smart-phone-based medical devices) will serve as the foundation for this health care transition. From a system-level engineering perspective, consumer electronics are embedded systems, consisting of basic components such as processors, memory, and input/output interfaces. In the context of health monitoring, the development of biomedical sensors and/or actuators is a challenge of utmost importance. Custom biomedical sensors optimized for home and mobile health care environments will be pivotal factors in this transition.

However, in this scenario, the user or operator of a medical device will not often be a trained caregiver, but rather a consumer who may have varied experience with consumer devices like smart phones and computers. Even with the availability of professional and technical

support, patients will be on their own in many circumstances, including times when they set up devices, acquire measurements, and read results. Hence, there are three basic requirements for the design of custom biomedical sensors in order to minimize usage complexity, ensure result reliability, and improve safety:

1. **Non-invasive measurements** will be the foremost product of biomedical devices used outside of hospitals. For example, to measure blood glucose, most glucose meters acquire blood samples by skin punctures. Non-invasive glucose monitoring technology [10] will provide a much easier way for diabetics to manage their care, dramatically or completely reducing pain and the chance for infection. Several methods are described in [10] to non-invasively acquire glucose readings, such as near infrared spectroscopy and transdermal measurement. An approach to use tears instead of minimally invasive blood samples has also been investigated [11]. The primary challenge for the use of most non-invasive biomedical sensors as indirect methods is to improve performance and obtain reliable results given common issues like baseline wandering, the influence of sweat, and surface artifacts. Non-invasive techniques may also initially involve invasive procedures, such as implanting medical components.

2. **Continuous monitoring** is important from the perspective of mobile applications. This does not always mean continuous data acquisition, but rather the capability to provide reliable performance during a change in measurement context (e.g., stepping outside of the house). The main idea is that wearing or using a biomedical sensor should minimally influence a user's normal activities while maintaining the ability to acquire valid data. Since these devices will be used outside of relatively controlled hospital environments, biomedical sensors designed for consumers should consider the more severe conditions of daily life, such as movement, sweating, and exposure to sunlight. For example, motion artifact is a common issue in electrocardiograph and pulse oximeter design. However, motion itself is a common parameter continuously recorded by motion sensors like accelerometers and gyroscopes, so this parameter can help a device to compensate for alterations in data due to real-world conditions. Identification of context change and accurate extraction of the target physiological parameters are of utmost importance.

3. **Surface biosensors**, a concept described in [12], will be prolific in future care scenarios. For instance, several types of wearable designs for ambulatory pulse oximeters exist. Some of these use ring form factors, and others use finger clips. These designs use

predominantly transmission-mode sensors. For broader use with wrist watches, head bands, socks, sensor ‘Band Aids’, and other wearable platforms that are unobtrusive and well suited for mobility, it makes sense to consider reflectance-mode layouts. This is especially true when one contemplates the immense potential of surface biosensors: medical sensors embedded in the surface of everyday consumer electronics such as handheld personal device assistants (PDAs), cell phones, smart phones, tablet PCs, head-mounted displays, etc.

More properties must be considered when designing custom biomedical sensors for home and mobile health care environments, including safety, durability, and reusability. There are generally two ways to categorized these devices: according to their functionality (i.e., the parameter(s) monitored) and their design principle (i.e., how each measurement is obtained). Functionality is the first item that draws a customer to a medical device. It covers the primary vital signs [13] (core body temperature, weight, pulse rate, blood pressure, and respiratory rate), other circulatory system parameters (e.g., blood oxygen saturation and blood glucose level), nerve system parameters (e.g., pain and mood), activity level (e.g., gait speed), and other parameters that help to predict and/or diagnose health conditions.

The following sensing methods are apt to be employed: optical (e.g., light-based sensor or imaging system), acoustic (e.g., ultrasound), and electrode-based (e.g., ECG, EMG, EOG, or bioimpedance). There are many design variations in each category, including contact versus non-contact. In this dissertation, several specific sensor designs are presented in Chapters 3 and 4. For example, in a reflectance pulse oximeter design, the sensor can employ a single small photodiode [14] as in most transmittance sensors. However, tissue is highly forward scattering, so the relative number of remitted photons detected in reflectance mode is low, yielding lower-quality PPGs [15]. Improved sensor configurations are therefore often adopted to better acquire the radial reflectance profile, including a ring-shaped photodiode design [16], [17], a photodiode array around the central LEDs [18], and conversely an LED array around a central photodiode [19]. These designs generally employ cascaded high pass and low pass filters to extract the PPGs [20]. Such analog filters inevitably alter and even distort the signals of interest. These alterations are visibly obvious in some papers, and cycle-to-cycle inconsistencies can be significant. For this reason, a filter-free design is desirable (see Chapter 4).

Another special category of custom biomedical sensor is single-use and disposable. The obvious advantage of single-use solutions for certain monitoring situations is that no sensor

cleaning and sterilization is needed. This reduces the risk of infection, may provide more accurate results, and can be more cost effective than traditional technologies. For example, a single-use pressure sensor could be cheaper than a sanitary pressure transducer that requires autoclaving and recalibration after each use [21].

Wireless Body Area Networks

Wireless body area networks have received significant attention in recent years due to their potential to enable patient-centered wellness monitoring, where the continuous acquisition of health indicators such as heart rate and activity level have garnered primary emphasis. Technical focus areas within this domain include sensor miniaturization [14], wearable sensors [22], ultra low-power circuits [23], communication protocols [24], and network topologies. General rules related to, e.g., low power and reliability, guide each WBAN development phase.

The physical components in a WBAN fall into two categories depending on their roles in the network: sensor (transmitter) and coordinator/hub/base-station (receiver). Note that in many applications, sensor nodes also receive data (e.g., commands) from coordinator nodes. This basic structure implies two fundamental roles of a WBAN: an integrated sensor collection and a health gateway (coordinator).

Various sensor configurations can constitute a WBAN. Motion sensors are used in [25] for computer-assisted physical rehabilitation. An 8-channel EEG system based on a WBAN is described in [26]. Two wearable sensors (a pulse oximeter and a blood pressure monitor) and a GPS module are employed in a wireless system for patients in a disaster scene in [27]. A wireless body sensor network that provides a finger photoplethysmogram (PPG), an electrocardiogram (ECG), and continuous cuffless blood pressure via a 3D accelerometer patch is presented in [28]. A BAN in [24] contains two accelerometers, an ECG sensor, a blood pressure sensor, and a pulse oximeter; it detects context change and determines the state of each sensor (e.g., data rate).

In these WBAN applications, the gateway role is primarily to coordinate sensor activities and stream data to a computer for storage, analysis, and display. Furthermore, these data could be forwarded to an external network such as a Medical Device Coordination Framework (MDCF) [29] introduced in the next section. In practice, the Internet would be a reasonable candidate for the 'external network.' Recently, eDevice announced the HealthGO platform which utilizes Freescale's Home Health Hub reference platform [30]. HealthGO was reported as the

first fully customizable data aggregation platform in industry, supporting remote patient monitoring. Its key tasks include universal connectivity for patients/devices and data delivery for remote device management through global communication network services (e.g., cellular and Ethernet). A similar technology from Qualcomm Life – 2net platform – provides a wireless health network that connects medical devices and a cloud-based system through a 2net Hub [31].

WBAN protocols contain a common component and an application-specific component. Since early 2008, the IEEE 802.15 Task Group 6 has addressed WBAN communication standards, which are likely to be based on the medium access control (MAC) layer of the current IEEE 802.15.4 standard [32]. Such a communication protocol should ensure high network capacity, energy efficiency, and adequate quality of service [33]. The protocol defines what data are transmitted and received, as well as when and how this occurs. To a large degree, these items determine the network performance, as they affect power consumption, data throughput, and packet loss rates. Data management techniques like adaptive duty cycles and message prioritization [24] can be considered if a monitoring scenario involves patient state/context changes and/or the coexistence of different types of sensor nodes.

Diverse and creative application-layer schemes can be designed and evaluated for various WBAN scenarios. A new wireless body area network is introduced in Chapter 8, which presents a WBAN that emphasizes high-resolution raw data, real-time operation, and time synchronization of intra-sensor data and inter-sensor waveforms. Utilizing this WBAN, Chapter 9 describes a new physiological use case for pulse wave velocity estimation.

Medical Device Integration Frameworks

Many medical devices on the market support some form of connectivity through serial ports, Ethernet, 802.11b/g/n or Bluetooth wireless, etc. Unfortunately, device interoperability is lacking due to the limited implementation of interfacing standards, and connectivity is usually only leveraged by Medical Device Data Systems (MDDSs) that uni-directionally transfer data/events from devices to composite monitors or EHRs. In contrast, electronic devices in other domains have widely adopted interfacing standards that allow “plug-and-play” interoperability. Many of these devices can be connected to computing platforms with flexible software frameworks that enable developers to easily build complex monitoring and data sharing applications, some of which employ closed-loop control algorithms. The realization of medical

systems with these capabilities has been slow, partially because of the need for careful device regulation coupled with issues such as cost, liability, infrastructure availability, and the existence of a user community (both patients and providers) that is not always technology-savvy – similar barriers that adopters of telemedicine technology have faced since the early 1990s. As a result, medical devices are predominantly stand-alone or networked in pre-arranged collections, where the number of possible device configurations and the resulting interactions between these devices are limited to a carefully controlled subset of possible embodiments. Device implementations are therefore static by nature so as to streamline the device regulation process and limit liabilities incurred.

In previous collaborative work with engineers from the U.S. Food and Drug Administration and researchers from the University of Pennsylvania [34], KSU researchers have implemented a medical computing and device integration platform called the Medical Device Coordination Framework (described in Chapter 10 and upgraded to a web-based Hospital Information Integration System in Chapter 11). The MDCF provides many of the capabilities called out in the Integrated Clinical Environment (ICE) platform [35] specified in the ASTM F2761-09 standard. An ICE-compliant implementation can be viewed as a computing platform (architecture, hardware, and software services) that allows heterogeneous medical devices to be integrated to create medical systems that support high-acuity patient care, similar in criticality and functionality to Integrated Modular Avionics. ICE provides services that expose data and control aspects of integrated devices to an ICE Supervisor Application. Individual medical devices that provide data and administer treatment are connected to a shared network substrate managed by the ICE Network Controller. This system can manage multiple devices simultaneously based on extensible clinical APPs. Each APP can combine input from multiple devices, synthesize it, display it, and possibly take action, instructing devices to alter their behavior based on feedback from other devices.

At issue is the desire to move toward intelligent device collections that include operational attributes similar to devices in the evolving consumer electronics domain. In that space, devices are fully networked and rely heavily on functional improvements through automated and user-initiated APP upgrades; updates that are currently manifested in software but can also be realized through hardware technologies such as FPGAs. Architecturally, this design space can be approached two ways:

1. **Complex highly-capable devices** that incorporate reconfiguration and decision-making capabilities onboard and then feed their data to the network with which they interact.

2. **Platform-oriented frameworks** where simple data-acquisition devices upload their data to a computing platform capable of performing complicated data processing operations with reconfigurable APPs made up of components, or transformers.

The first approach lends itself well to (a) some wireless scenarios where immediate assessments of data viability can avoid the needless transmission of data with limited diagnostic value, (b) applications where hardware optimizations for known scenarios would improve device operation, and/or (c) applications where the static nature of this functionality directly impacts the ability to regulate the associated devices. The onboard tagging technology (Chapter 5) falls into this category, and a feature detection application for a custom reflectance pulse oximeter is demonstrated in Chapter 6.

The second approach is appropriate for applications where (a) more flexible (or more numerous) data analyses are desired, (b) access to a broader developer based is appealing, and/or (c) verification and validation approaches for reconfigurable devices can be put in place to protect patients and providers from developer oversights. The work in Chapter 12 explores this approach and aims to develop a vision of what can be called a "medical platform-aggregated device" (MPAD). An MPAD is a composite medical device whose functionality is achieved by aggregating (a) medical platform-oriented devices (MPODs) – hardware components – such as sensors and actuators that are designed to be integrated (either by wire or wirelessly) with a computing platform that supports safety and security guarantees appropriate for medical devices and (b) applications (APPs) hosted on the computing platform that interact with platform-connected devices, implement data processing and control algorithms, and realize visual displays and control panels used by patients or caregivers.

Chapter 2 - GumPack Platform

Wearable and everyday carry devices that integrate with regional or hospital information networks promise to increase the quality of care rendered to individuals that desire mobility yet require frequent or continuous health monitoring [36], [37]. The maturation of interoperability standards such as IEEE 11073 – Personal Health Data (11073 PHD) [38], coupled with the proliferation of wireless communication standards, plug-and-play wired technologies, small medical sensors, and low-power computation and visualization tools (that are in large part driven by the smart phone industry) offer promise for the creation of small, effective portable medical devices that can be customized to meet the needs of the individual user. Sensor-laden devices that offer the connectivity of a cell phone and are small enough to attach to a keychain or be carried in a purse like lipstick, an inhaler, or a pocket knife are especially attractive, as such items would be culturally accepted, inconspicuous, and minimize the effects of the medical monitoring process on day-to-day existence.

Cell phone centric development (see examples in Chapter 1) seems at first glance to be an effective match for everyday-carry medical devices that promote the transition to patient-centered care. However, a smart phone is not natively designed to be a professional platform for medical applications. The physical interface to a phone is limited by its wired/wireless connections, each of which adds an additional communication layer and requires extra electronic parts, affecting the device cost, power consumption, and form factor. Further, standardization at the hardware level can be difficult because manufacturers define different interfaces for each embedded element. This inaccessibility to embedded hardware resources within a smart phone further extends to its firmware, whose development is constrained to the software development kits (SDKs) provided by the respective manufacturers. Finally, thread priorities in smart phones are optimized for voice communication rather than data gathering, which brings into question the reliability of physiological data that must be acquired through these external ports. In summary, a phone-based platform and its supporting development kit is far from an open embedded environment that can provide the freedom to design an optimized consumer medical device. Moreover, the integration of a reliable and efficient information framework within the current operating

systems that facilitates device reconfigurability and interoperability will be impossible without strong technical support from phone manufacturers.

This chapter addresses a new type of multi-sensor medical monitoring device (nickname: GumPack) whose functionality and architecture are notably different relative to current research and commercial devices. The design attempts to optimize the low-power versus high-performance tradeoff relevant to mobile embedded systems. One trend in wearable medical device design is to form a body area network with wearable sensors that connect to a smart phone. The phone then serves as a computation platform and information-forwarding hub, and the devices are assumed to be ‘dumb,’ computationally hobbled, and unable to coordinate with other nearby devices: their only role is to acquire and forward data to the hub. This wastes battery power and frustrates device creativity because hardware development kits for smart phones are generally unavailable, so the topology is fixed. Third-party designers cannot create, e.g., reconfigurable surface-sensor collections on these phones. The technology proposed here leapfrogs this design space by providing a small, reconfigurable, medical sensor platform that can be carried everyday and operate up to six hours on a single battery charge when transmitting device data continuously over a Wi-Fi connection. It will offer a computer-grade signal processor, support security functionality (user identification, device authentication, and secure data access), and encourage real-time interoperability.

Goal and Objectives

The overall goal of this GumPack effort is to prototype and evaluate a new type of multi-sensor medical monitoring device that offers a small size, the processing capabilities of a computer, the ability to host multiple biomedical sensors that can be reconfigured based upon patient need, and communication and networking resources that allow the device to upload data to a patient’s electronic health record via the Internet. A GumPack provides a platform for the research content documented in this dissertation, including custom biomedical sensor design, sensor integration and interaction, and medical device integration frameworks. Five technical objectives support the overall goal mentioned above:

1. Design a small medical system in the form of a cuboid, where each surface can host a different plug-and-play medical device, processing sub-system, or connectivity mechanism.

2. Overlay an information framework on the mobile system that promotes medical device reconfigurability and interaction with remote information frameworks.
3. Demonstrate the sensibility of this approach with a collection of snap-on surface devices.
4. Unveil the use of the GumPack as a body area network hub that can interface to wearable/nearby sensors (“Chiclets”) via ZigBee wireless telemetry.
5. Illustrate the ability of the GumPack to perform advanced system and data processing tasks often reserved for personal computers and to interact with remote systems using Wi-Fi.

Technology Overview

A GumPack is a new type of everyday-carry, multi-sensor medical monitoring device whose layout and functionality offer the potential to leapfrog the design space specified by mobile phones connected to dumb medical devices. A rectangular cuboid prototype for this device is depicted in Figure 2.1. As the GumPack is rotated each quarter turn, the user is presented with a different face of the cuboid, where a new device resides. These surface elements can be switched out in a plug-and-play manner to match a user’s specific care needs.

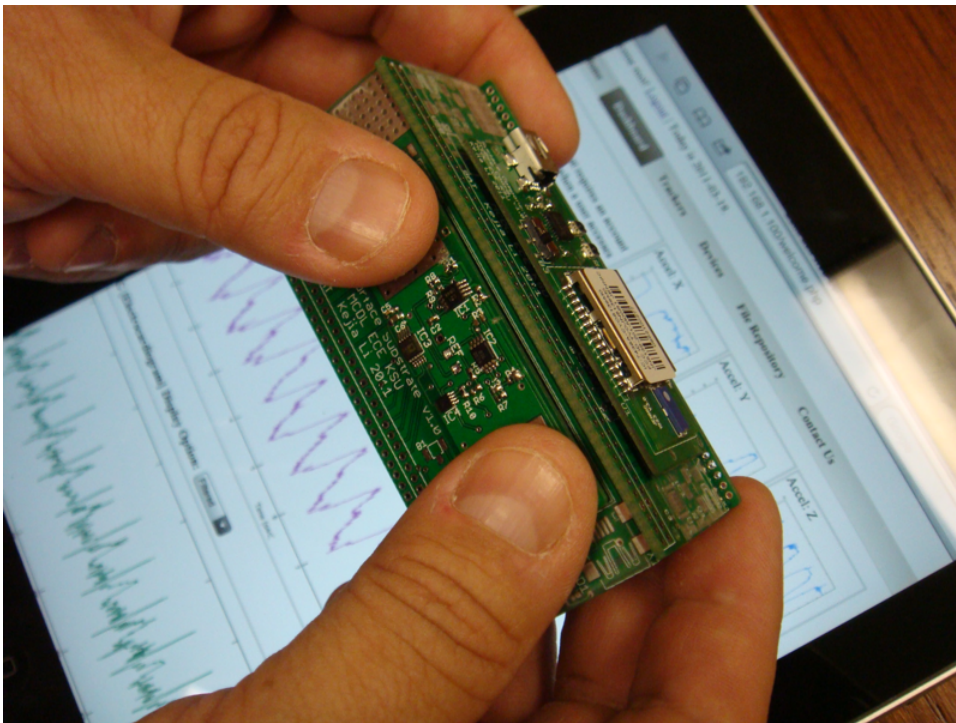


Figure 2.1. A prototype GumPack design and demonstrative Web interface (closest surface: single-lead electrocardiograph).

The everyday carry vital sign monitor is also intended to serve as a body area network hub to support additional wearable or nearby devices (Chiclets – described later in this dissertation). The monitor is designed to be one component that operates within a ubiquitous home infrastructure yet also offers patient mobility outside of the home. Operationally, it will supply continuous or processed data in real time or in store-and-forward mode. On-board storage will allow a PHR to travel with the user, and advanced computation capabilities will promote onboard signal processing operations such as filtering, parameter extraction, and spectral analysis.

From a connectivity viewpoint, the base platform will offer cell-phone, Wi-Fi, and lower-power wireless (e.g., Bluetooth and ZigBee) support, depending on whether the unit communicates with a remote system or a local body area network. USB serial communication and rechargeability are important features, as is Web interface support, which will allow nearby devices such as smart phones and tablets to view these sensor data, e.g., when a display component is absent on the GumPack.

Note that this design's resource collection makes it a good base platform for algorithms that address context/situation awareness, physiological models that assess patient health given multi-sensory inputs, intelligent agents, and on-board analyses for local decision making (including the ability to maximize battery life through onboard processing, minimizing the need for telemetry). Role-based operation and device-level security (e.g., user identification, device authentication, and secure data access) are also sensible targets for this design, which seeks to optimize the low-power versus high-performance tradeoff relevant to all mobile embedded systems.

At present, the GumPack does not provide a cell-phone-like user interface (e.g., touch screen). However, a user can alternatively utilize a phone, a tablet, Internet TV, or any device that supports Wi-Fi and Web browsing as an indirect access interface. While the data depicted on the iPad interface in Figure 2.1 are real measurement results collected from the GumPack and a network of Chiclet sub-sensors, they currently do not update in real-time, and the graphical interface itself is not needed for device operation. This demonstrative GumPack Web interface also provides early implementations of a health & wellness tracker, a device configuration panel, and file checkout using a registered GumPack account (see Chapter 7).

GumPack Concept Model

The rectangular cuboid object illustrated in Figure 2.2 is the initial GumPack concept model that consists of (a) up to four surface components, (b) four surface substrates, (c) a camera and a microphone, (d) wired connectors, and (e) a battery. Each surface component and substrate pair is connected by two 70-pin connectors. The Gumstix Overo board [39] (see Figure 3.1, left side) is a required surface component that supports core computing/control, component coordination, power management, and basic communication, while the configuration of the other surface components is flexible. A surface component can be a reflectance pulse oximeter, a two-thumb ECG, a sensor conditioning board, an expansion board, or even another Overo board. One type of expansion board, e.g., a ZigBee coordinator, can create a local low-power wireless network, e.g., a body area network, using Chiclets: small, wearable, low-power wireless sensors (see Chapter 3 for a detailed description of each surface component).

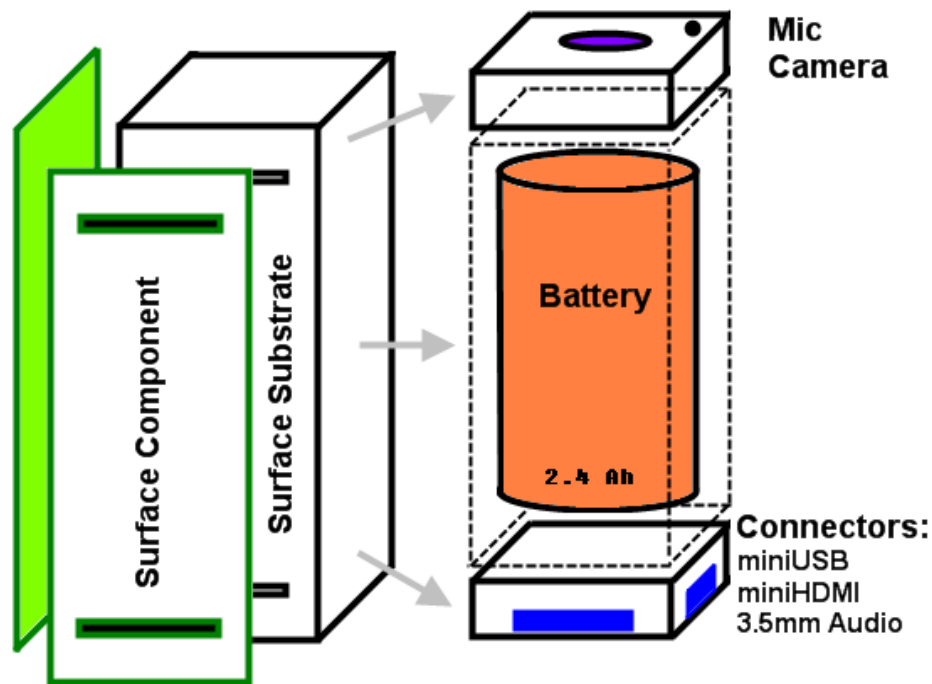


Figure 2.2. GumPack cuboid conceptual model.

Figure 2.3 illustrates the GumPack communication flexibility in terms of wireless network functionality. The Wi-Fi module, in managed mode, enables the GumPack to potentially connect to a Hospital Information System via an access point. In ad-hoc mode, the GumPack is a

local data server for handheld devices to access. For example, one could display processed PPG data on a nearby iPad. A local ZigBee network of Chiclets can coexist with a Wi-Fi mode. For a use case of, e.g., ‘motion pattern recognition,’ Chiclets could be accelerometers that send data from different body locations to the GumPack while the GumPack simultaneously runs other medical device components.

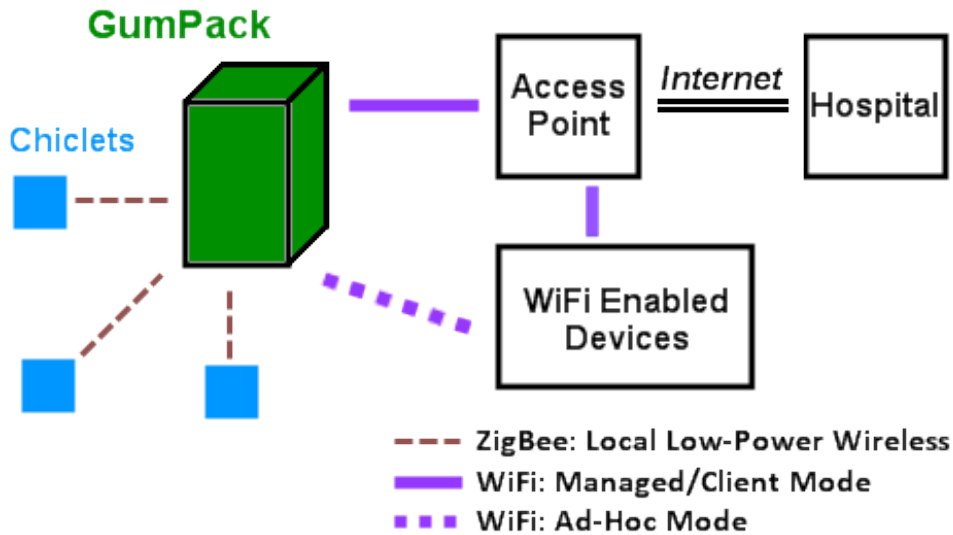


Figure 2.3. Wireless network architecture of a GumPack.

From the perspective of standards and interoperability, the modularized, configurable, and expandable GumPack design will require specifications for ‘surface component’ design and medical data streaming. Such standards will be important for team collaboration, device upgrades, and integration with other information systems. The ISO/IEEE 11073 standards (including the Personal Health Data subset) are being assessed as an interoperability complement to the transport standards (e.g., ZigBee, Wi-Fi, and USB) already supported by the GumPack hardware.

Chapter 3 - Surface Component Design

The surface component (SC) and Chiclet concept provided by the GumPack platform serves two purposes: (a) traditional custom biomedical sensor design and (b) sensor integration and interaction based on the uniform interface provided on a GumPack, which functions as a medical components hub. The first item is addressed by four example SC designs and a generic SC template in this chapter, as well as a pulse oximeter Chiclet design in the next chapter. The second item unfolds in Chapters 7 through 9.

A conventional way to upgrade a device typically involves redesigning the whole embedded system with new software support, such as utilizing a new CPU for a smart phone. With the GumPack platform, a more flexible and innovative approach is to plug in only the SCs that support the functionality a user really needs, such as a new biomedical sensor. This is one major benefit of the GumPack design. Four instantiations of GumPack SCs are described below.

Processor SC

The primary SC hosts a processor and other system resources such as memory, a network interface, etc. The processor SC is a compulsory element in the GumPack design, whereas the other SCs are optional and depend on patient needs. A processor SC plugged into an SS form a minimum system.

At present, this SC utilizes Texas Instruments' OMAP3530 high-performance applications processor. Several powerful subsystems housed in this chip include (a) an ARM Cortex-A8 microprocessor, (b) a TMS320DM64x+ digital signal processor, (c) an SGX graphics accelerator, and (d) a camera image signal processor. It supports numerous HLOS and RTOS solutions, including Linux and Windows CE, providing maximum flexibility over a wide range of end applications. More specifically, the Gumstix Overo board (see Figure 3.1 (left side)) is employed as a processor SC.

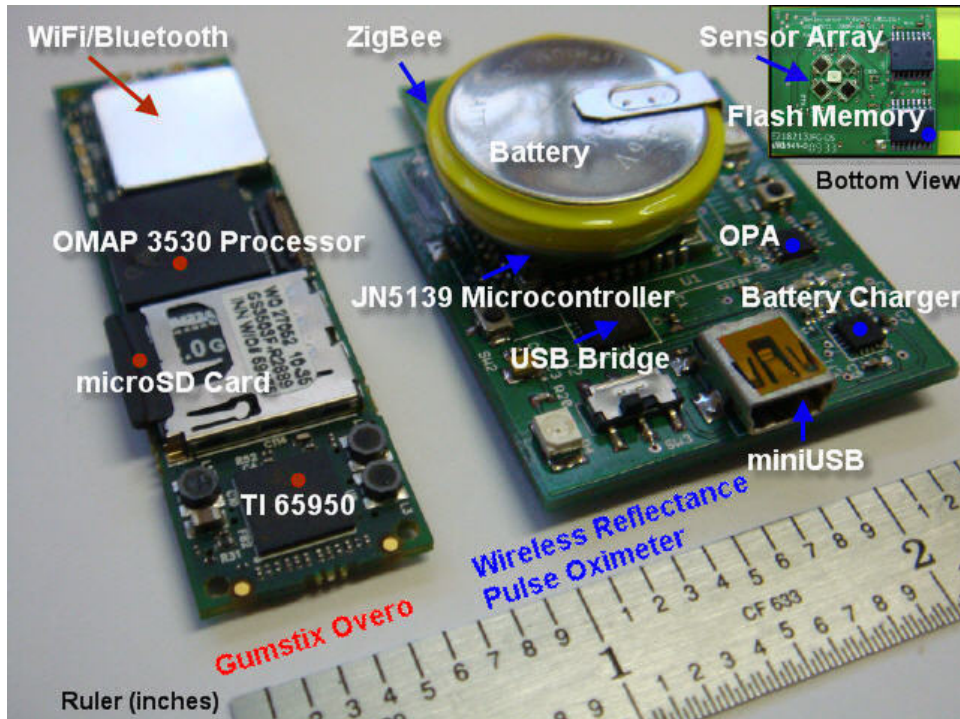


Figure 3.1. Gumstix Overo board (left) and custom wireless reflectance pulse oximeter (right).

The Overo board size is 58 mm by 17 mm, which constrains the smallest GumPack size to 58 mm x 17 mm x 17 mm. This SC provides more signals than are defined on the 44-pin interconnection interface, e.g., camera signals, display signals, memory bus signals, JTAG signals, etc., many of which are directly derived from the OMAP3530 processor. Future use of these signals will depend on the GumPack applications employed.

Other notable features of the Overo board include a Wi-Fi/Bluetooth module, which supports most GumPack wireless applications, and a Micro SD card module, which enables a fast switchover of the entire software system (even the boot loader), e.g., from onboard FLASH to the inserted Micro SD card. Current consumption is approximately 250 mA (500 mA with Wi-Fi active). Actual power consumption will depend on the GumPack configuration and system resource usage (e.g., CPU usage, Wi-Fi active time, etc.).

Motion Sensor SC

Motion sensors have been widely embedded in consumer electronics, since they provide useful information about device attitude and user activity. Applications based on motion sensors, especially accelerometers, include screen auto rotation, game controllers, and system auto wakeup (using an inertial interrupt function). In the biomedical domain, motion sensors are used to monitor patients' daily physical activity [40] or as a reference signal to reduce motion artifact in, e.g., ECGs [41] and PPGs [42].

Given its ability to support generic applications, the GumPack design can be viewed as both a consumer electronics platform and a collection of medical devices (sensors). The first natively designed GumPack-ready SC is the motion sensor board in Figure 3.2 (top view) and Figure 3.3 (bottom view). The board size is 58 mm by 17.5 mm. Two types of motion sensors are mounted on this SC board: one 3-axis accelerometer and two gyroscopes (1- and 2-axis). The relative x , y , and z directions are printed on the board (see the lower left corner in Figure 3.2). The accelerometer employs STMicroelectronics's LIS33DE unit, an ultra compact low-power (< 1 mW) three axis linear module. It sends the measured acceleration (including the direction of gravity) within the range of ± 8 g through the I²C serial interface with an address of 0x1d. The gyroscopes are STMicroelectronics's low-power LY510ALH (z-axis) and LPR410AL (x, y-axis) units. They provide angular rate data within the range of ± 100 dps (degrees per second) through an analog output (connected to the ADC pins of the GumPack).

A profile memory chip comes with all SCs and plays an important role in the GumPack Component Connection Framework (GCCF) introduced in Chapter 7. For the motion sensor SC, Microchip's 24AA01, a low-power (< 1 mA) EEPROM, is adopted. A GumPack can access its 1024 bits of memory space through the I²C serial interface with the address 0x50.

On the bottom side of the motion sensor SC, one can find the two 70-pin connectors (header), as in Figure 3.3, that match the 70-pin connectors (receptacle) on the surface substrate. When the SC is plugged into the system, it is automatically identified (on the I²C bus) and authenticated (with information in EEPROM). It can then begin to provide motion data, consistent with the functionality realized by the GCCF.

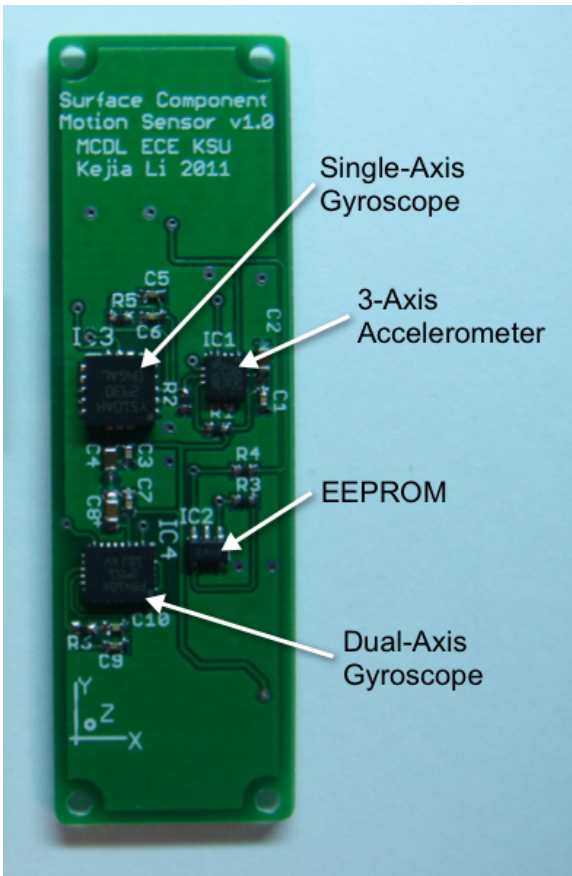


Figure 3.2. Motion sensor surface component (top view).

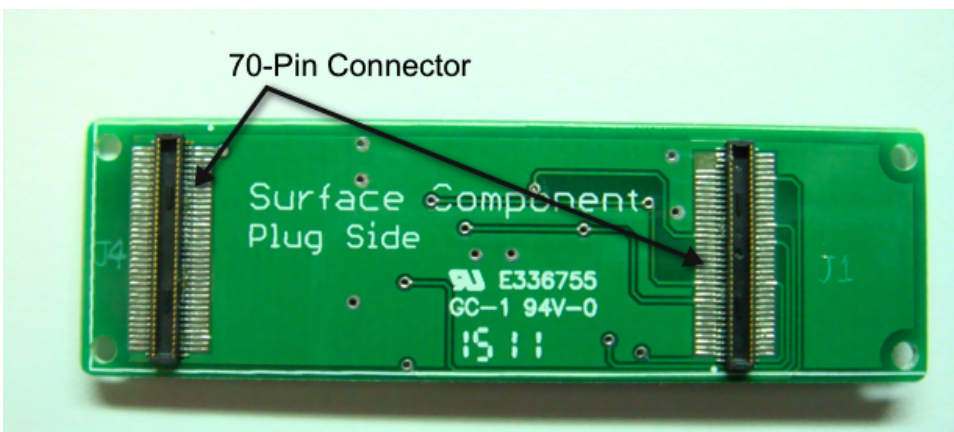


Figure 3.3. Motion sensor surface component (bottom view) with two 70-pin connectors.

Electrocardiograph SC

An electrocardiograph (ECG) is a standard clinical device that is also popular in mobile health care environments. It transduces tissue biopotential differences into millivolt-level electrical signals using differential electrode pairs. ECG variations are often distinguished by the number of leads employed (one ECG lead results from two electrodes), such as 3-, 5-, and 12-lead configurations. In the context of a wearable medical device, a single lead ECG [43] can help to minimize device size and power consumption.

An ECG SC is a single-lead ECG board natively designed for the GumPack, as in Figure 3.4. The board size is 58 mm by 17.5 mm. Two electrodes are integrated into the top side of the board in the form of conductive solder pads. During the measurement, two digits (e.g., thumbs) from different hands need to be carefully placed on the two onboard electrodes and kept still without touching other electronic parts. A single-lead ECG lacks a reference signal from a third electrode, which makes it different to manage relative to a regular 3-lead ECG system. The ECG SC design is based on the *Medical ECG Application Circuits* described in Texas Instruments' INA321 datasheet [44]. The TI INA321 is the instrumentation amplifier component, and the TI OPA2336 is the operational amplifier.

Two types of reference voltage schemes are available, as noted in Figure 3.4 (upper). The purpose of a reference voltage is to keep the ECG signal in the range of 0 to 3.3 V at all amplification stages, providing baseline wandering control. REF 1 is fixed at 1.65 V (or half of the power supply level). REF 2 is output by a digital-to-analog converter (DAC) from an Analog Devices AD5315 chip. The particular value of REF 2 is sent through the I²C serial interface at the address 0x0c. Given access to REF2, digital baseline control could lead to a full digital ECG circuit without an analog filter. Such a design could yield a high-fidelity, unfiltered (i.e., distortion-free) electrocardiogram. A similar scheme was employed in the reflectance pulse oximeter mentioned in Chapter 4.

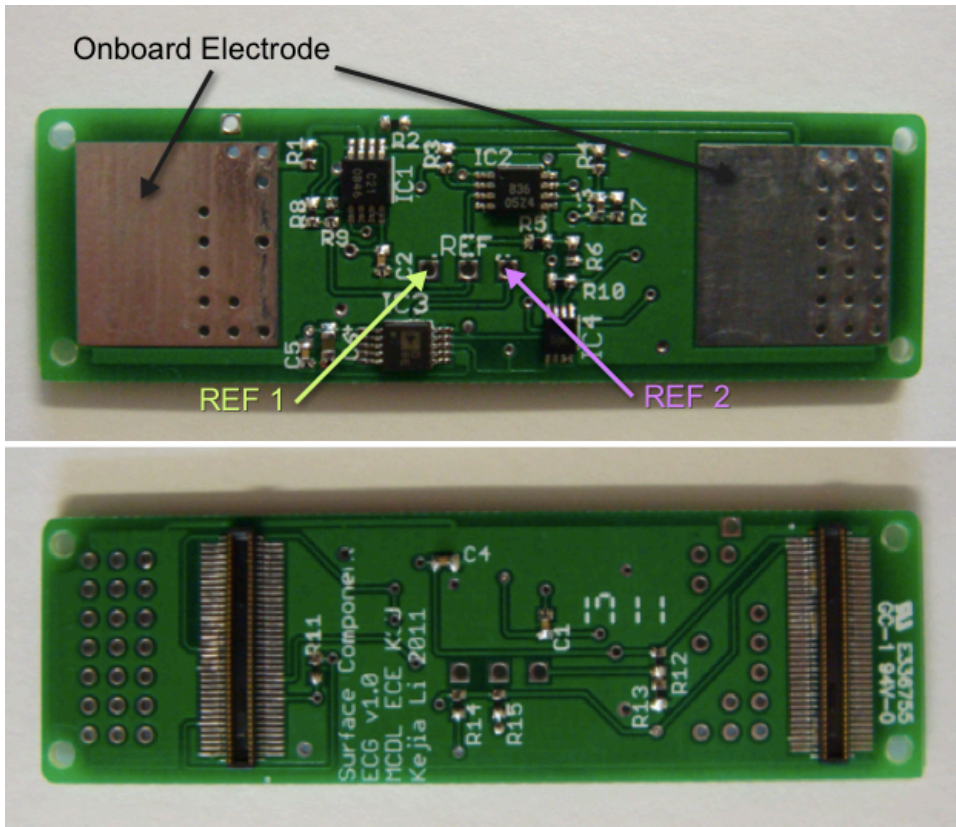


Figure 3.4. ECG surface component (top and bottom view).

Figure 3.5 shows an alternative usage mode for the ECG SC. The electrodes (sensors) are transported separately from the board and plugged in when needed. Commercial electrodes offer hands-free measurement and higher signal quality. This is a typical example of using accessories to improve the functionality of a surface component while keeping the GumPack unit in a compact form factor. This use case also indicates that the sensors need not be housed on the same SC at all times, since the board acts only as a signal conditioning circuit and interface to the GumPack. This concept will be elaborated upon in the section Generic Surface Component. Note that the jumper cap is also identified in Figure 3.5. In this case, REF 1 (half of the supply voltage) is selected.

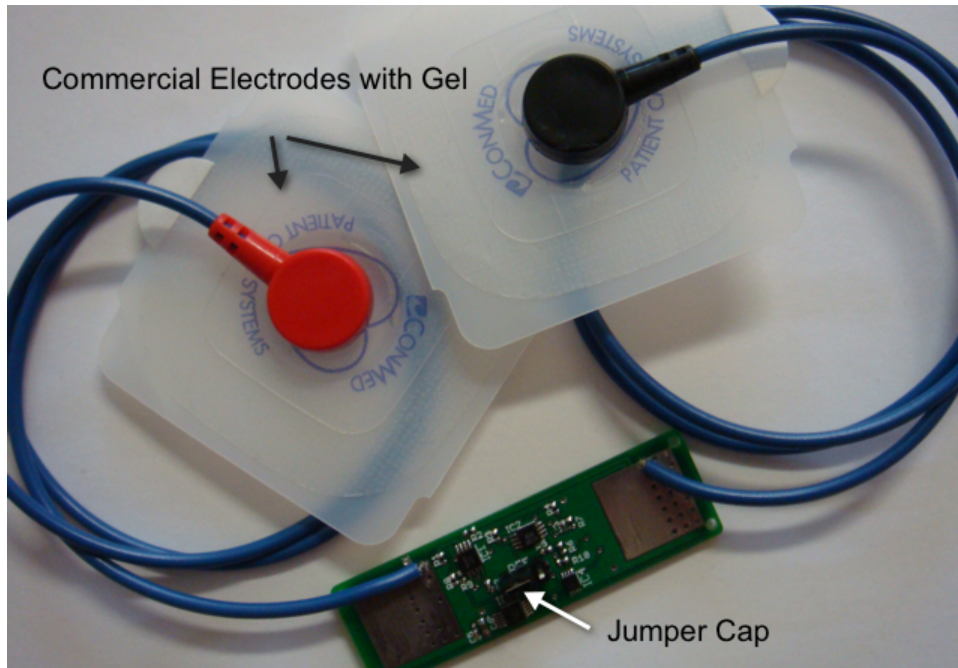


Figure 3.5. ECG surface component with commercial electrodes.

ZigBee Coordinator SC

The GumPack can host up to three biomedical devices or sensors in the form of SCs. To further extend its hosting capacity (e.g., to associate it with a body area network, where low-cost, low-power components are required), a ZigBee coordinator SC has been designed – see Figure 3.6 and Figure 3.7. The board size is 58 mm by 24 mm.

To simplify the design procedure, the same Jennic JN5139 wireless microcontroller is used in the ZigBee coordinator SC as was used in the wireless pulse oximeter in Figure 3.1 (right side). The JN5139 was designed for robust and secure low-power wireless applications. It integrates a 32-bit RISC processor with a 2.4 GHz IEEE 802.15.4 (ZigBee) transceiver, 192 kB of ROM, 96 kB of RAM, and a mix of analog and digital peripherals. The wireless link requires the most current of all of the optional SCs, with a TX (transmitter) current draw of 38 mA and an RX (receiver) current draw of 37 mA. The CPU consumes 7.75 mA at full speed, and the current required by the peripherals (ADC, DAC, UART, Timer, etc.) is less than 1 mA in aggregate. The JN5139 sleep current (with an active sleep timer) is only 2.6 μ A. When compared to the 250 mA current consumption of the Wi-Fi/Bluetooth module on the processor SC, the ZigBee coordinator

SC demonstrates its low-power advantage for a body area network (a 250 kbps data rate within a 10 m range).

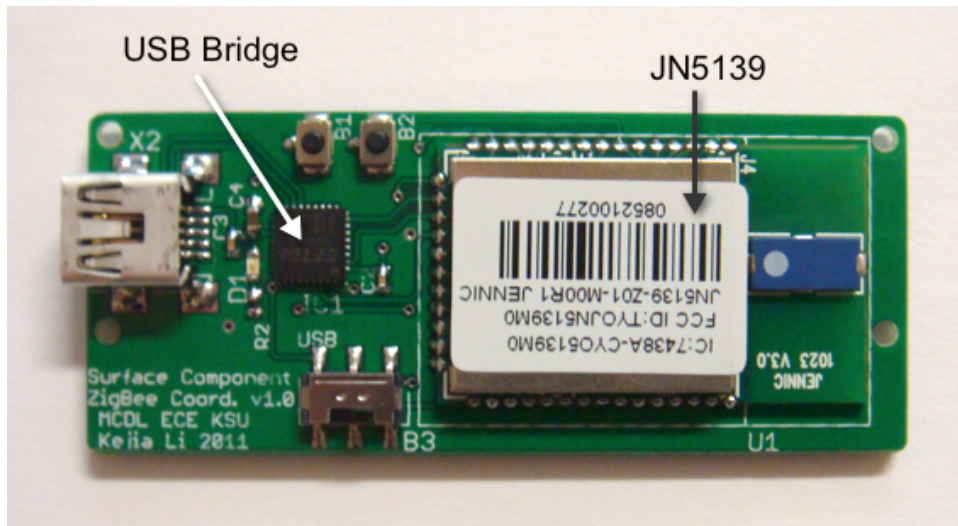


Figure 3.6. ZigBee coordinator surface component (top view).

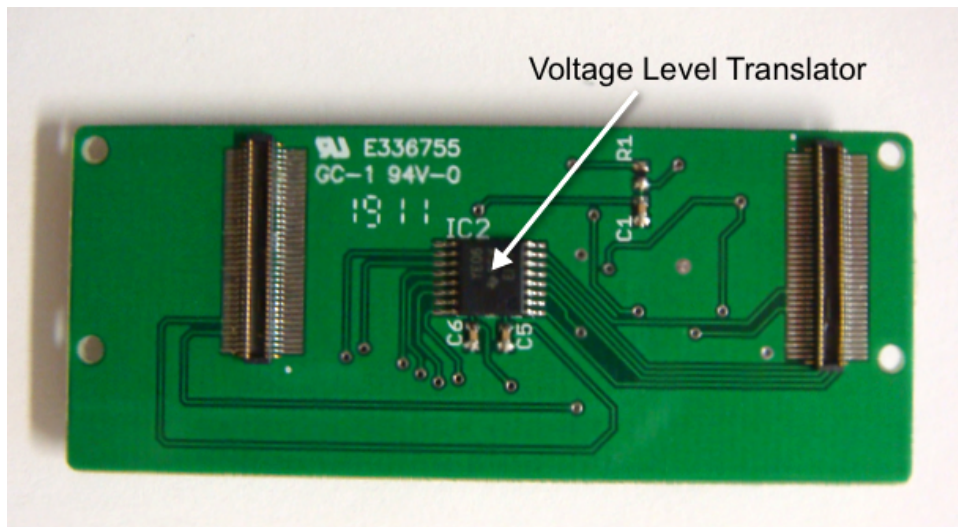


Figure 3.7. ZigBee coordinator surface component (bottom view).

Given the onboard USB bridge chip and mini-USB connector, the ZigBee coordinator SC can be programmed by and communicate with, e.g., a computer as a standalone device. These parts are needed because the firmware on the JN5139 module needs to be frequently updated in the development stage. Because of this feature, the GumPack could be the host to rewrite the

firmware on the JN5139 microcontroller. This is sensible, since a change in Chiclet functionality and topology warrants a corresponding change in the firmware of their coordinator SC (see Chapter 7).

At present, data communication between a ZigBee coordinator SC and its GumPack host occurs only through either the UART or Intelligent Peripheral interface. A UART serial communication interface is a regular mechanism to communicate with other devices, while the JN5139 features a high-speed, low-processor-overhead Intelligent Peripheral interface that functions as an SPI slave. One significant advantage of using the Intelligent Peripheral interface is that transmitted and received data are held in a dedicated area of memory without CPU intervention.

A surface component developer should note that the processor SC logic level is 1.8 V. Signals from other SCs need to be adjusted to match this level if they seek to communicate with one another. Since the JN5139 uses a 3.3 V power supply and its logic level is also 3.3 V, a voltage level translator is employed as shown in Figure 3.7. The TXB0106 is a 6-bit bidirectional voltage level translator from Texas Instruments. The 6 channels are used for SPI (4-bit) and UART (2-bit) data communication interfaces.

Generic Surface Component

Three native GumPack-ready surface components have been introduced. Design patterns can be extracted from them as guidance for a broader category of surface component. Figure 3.8 presents a generic surface component (GSC) model within the larger scope of the GumPack system. The GumPack Component Connection Framework (GCCF) provides a software interface to SC interconnections – this framework is unfolded in Chapter 7. Here, the SC block diagram on the left of Figure 3.8 is the focus.

The ‘Sensor’ block indicates a signal input module, including a sensor, conditioning circuit, and hardware interface. The ‘ID’ module is a memory chip storing SC identification and data communication information (e.g., data format). The ‘Actuator’ block is a module that receives information from the GumPack. It could be a real actuator, like a motor, or a feedback input on the sensor module.

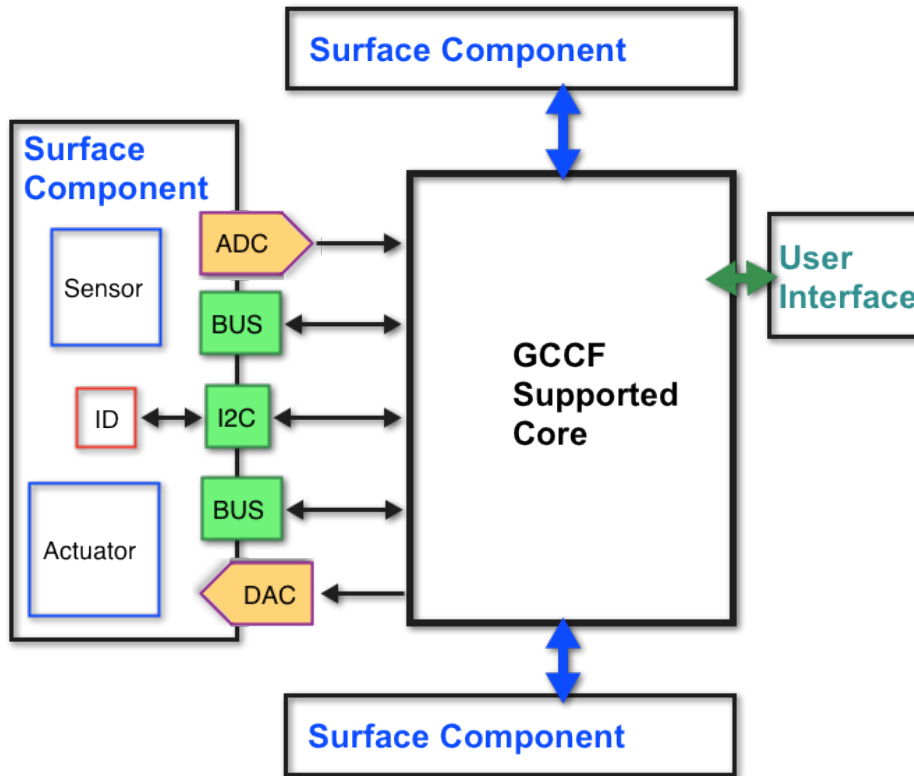


Figure 3.8. Diagram of a generic surface component (GSC) and the interface to the GumPack Component Connection Framework (GCCF).

The hardware interface to the GumPack core is both analog and digital. The analog interface includes ADCs and DACs, which are unnecessary on the SC side, while the digital interface includes GPIOs and buses, which typically require corresponding bus controllers on the SC side. For the ID module, an I²C interface is required by the GCCF; the address should be in the range of 0x03 – 0x77 and different from other I²C addresses that have already been allocated.

As noted for the ECG SC, neither the sensors nor the actuators are necessarily onboard. A system-level design of an infusion pump SC is introduced as an example to demonstrate how other medical devices could (partially) work within a GSC scheme. Information related to this discussion can be found in Texas Instruments' *Medical Application Guide – Diagnostic, Patient Monitoring and Therapy* [45]. An infusion system often consists of (a) a fluid reservoir, (b) a catheter system, and (c) a device that combines electronics and mechanics to generate and regulate fluid flow. An infusion pump usually refers to the third element. Sensors (part of a sensor module), that monitor the flow passing through the catheter system, and a pump (part of

an actuator module) that drives the flow are both attached to the catheter system. Here, after a GumPack SC is connected to these two modules, an infusion system could be built to manage the process. Figure 3.9 illustrates this idea. Conditioning circuits and ADCs collect data from upstream and downstream sensors (flow feedback). The pump driver controls the stepper motor or servo in the pump.

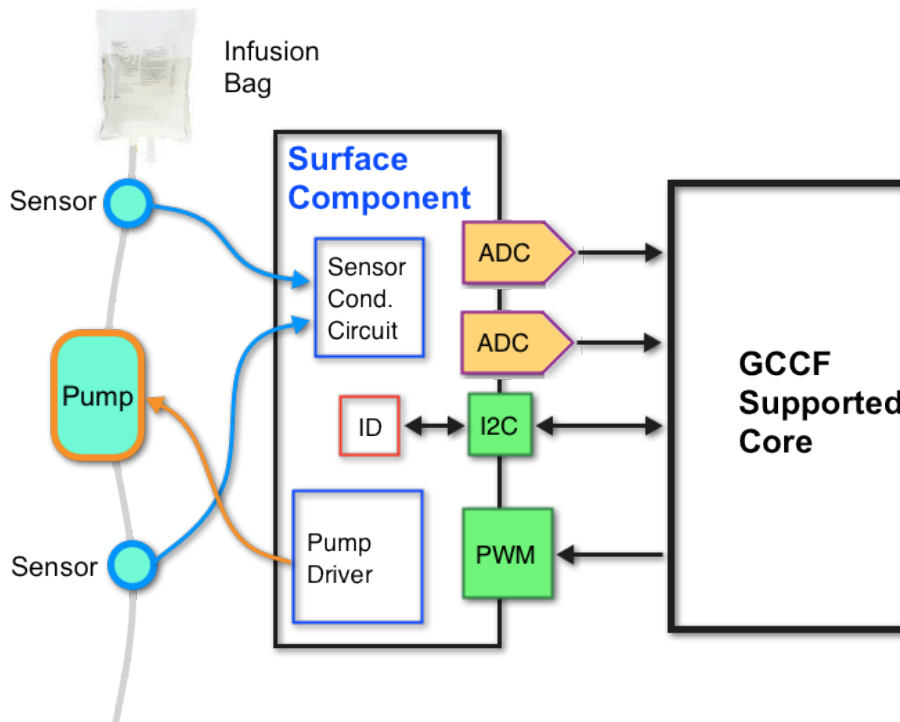


Figure 3.9. Infusion pump design using GSC model fitting.

Chapter 4 - Pulse Oximeter Sensor Design

Health problems such as cardiovascular disease, hypertension, diabetes, and congestive heart failure continue to plague society [46]. These conditions are primary drivers for the development of wearable and mobile health monitoring technologies that offer the potential to (a) increase the quality of life for individuals that already suffer from these health conditions and (b) prevent or mitigate the onset of disease in those that are at risk to acquire these health issues [47]. Of the array of medical devices that can be brought to bear for wearable/mobile applications that address these diseases, pulse oximeters offer significant relative promise because they provide two clinically relevant health parameters (heart rate (HR) and blood oxygen saturation (SpO_2)), they do not require electrical contact to tissue, and they can operate at very low power [23], [48]. Additionally, the pulsatile plethysmographic data offered by this light-based sensing technique (which are usually discarded by commercial pulse oximeters after being used to calculate the parameters for the front panel display) can help to ascertain hemodynamic information that is well-suited for the assessment of the disease states listed above [15], [49-51]. This information includes blood pressure [52], [53], arterial compliance [49], [54], [55], pulse wave velocity (PWV) [47], [56], stroke volume (and therefore cardiac output) [57], and other vascular parameters [50], [51], [58], [59]. Other relevant quantities include respiration rate [60], [61], patient motion [59], and even patient authentication [62-64].

However, low-cost pulse oximeter designs are unavailable that provide (a) quality, unfiltered PPGs ideally suitable for research and education toward the realization of new PPG diagnostics and (b) positional flexibility suitable for mobile and surface-based applications. While PPGs are often accessible from commercial desktop units via serial ports, these data have been filtered in proprietary ways to stabilize HR and SpO_2 calculations. Further, due to their clinical prevalence, pulse oximetry and PPG analysis deserve coverage in biomedical instrumentation laboratories offered in secondary education curricula, yet low-cost pulse oximeters that provide reasonable-quality PPGs are not a staple in off-the-shelf educational kits.

Regarding ambulatory pulse oximeters, it makes sense to consider reflectance-mode layouts for broader use with wrist watches, head bands, socks, sensor 'Band Aids', and other wearable platforms that are unobtrusive and well suited for mobility. This is especially true

when one contemplates the immense potential of surface biosensors (SBs). In this paradigm, physiological sensors will be accessible and signals will be easy to obtain, as human factors considerations for the overall product design will drive ease of use for the integrated biosensors. Additionally, each SB will utilize its host device's processor, memory, display, and wireless communication resources to provide user services typically unavailable in wearable platforms [65]. E.g., consider a reflectance pulse oximeter embedded on the back side of a cell phone alongside a built-in camera. As the user holds their finger against the reflectance sensor, the data will be processed by the microprocessor in the cell phone, and the LCD screen will display the signals and parameters.

In summary, the desire to extract additional physiological information from PPGs acquired with reflectance-mode sensors imposes design constraints with respect to signal quality. This chapter presents the design of a low-cost, wireless, reflectance-mode pulse oximeter suitable for these needs. It is initially housed on the surface of a printed circuit board but can be easily migrated to other surface-based applications. A unique filter-free circuit (that digitally extracts the PPG waveform) and a two-stage, feedback-loop-driven control system enable the acquisition of unfiltered PPGs with 2^{12} levels of precision from varied body locations. An optimized LED/detector configuration promotes surface use, and the device signal quality and cost enhance its potential for integration into SB-based consumer devices. This chapter presents a design for a filter-free, reflectance pulse oximeter that combines many desirable features into a single platform. A more detailed description of this design has been documented in the author's thesis [59].

Requirements and Device Layout

The design requirements are outlined in Table 4.1. Signal requirements include quality, unfiltered PPGs whose baselines are digitally removed, consistent with the prior discussion. The high sampling rate ensures that (a) primary signal and noise components are adequately sampled without aliasing and (b) secondary noise harmonics, e.g., 120 Hz up to several kHz from fluorescent lighting, are not aliased on top of the signal components of interest.

Table 4.1. Wireless Reflectance Pulse Oximeter Design Requirements

Signal	
Integrity	Unfiltered data with an optimal SNR
Precision	Thousands of peak-to-peak digitization levels
Sampling frequency	≥ 240 Hz to minimize PPG/noise aliasing
Baseline subtraction	Digital and filter-free
Data availability	Full access to all pulsatile/baseline data
Sensor	
LED/detector geometry	Radial arrangement, large area, and 3-5 mm source/detector separation
Ambient light operation	Adjustable gain and reference baseline
Functionality	
Communication	Wireless (10 m range) and USB
Local storage	Onboard flash memory
Battery	USB-rechargeable; Multi-day lifetime
Client Software	Visualization and control panel
Application	
Measurement sites	Multiple body locations; Various vascular profiles and perfusion levels
Wearability	Low-profile reflectance layout adaptable for wearable and SB applications
Cost	Low (~\$100)

Regarding sensor requirements, the photodetectors are ideally distributed radially around the central excitation LEDs to maximize the number of photons collected. Further, an LED/detector separation of 3 to 5 mm is appropriate at these wavelengths, as it maximizes the AC/DC ratio for each sensor channel, as verified experimentally [18], [66] and with Monte Carlo simulations [64]. In other words, reflectance photons that contain DC information from shallow, poorly perfused epidermal layers reflect near the central excitation LEDs and are undetected. Photons collected at greater radial distances are more likely to have traveled deeper into blood-perfused tissue and contain a greater percentage of AC data. Given the increased sensing area in

a large-area detector, the control circuitry must easily compensate for baseline changes due to ambient light, tissue perfusion, respiration depth, etc.

Figure 4.1 shows the block diagram for the pulse oximeter circuitry; a brief description was also included in [67]. The LED, sensor array, and operational amplifier (OPA) circuitry are coordinated by a Jennic JN5139 microcontroller. The intensity and timing of the bi-color LED are controlled by a digital-to-analog converter (DAC) and digital input/output ports (DIOs), respectively. A signal from the sensor array (four photodiodes surrounding the central bi-color LED) is first buffered and then fed to a differential OPA circuit. The buffered signal, designated here as the first-stage PPG signal (entire AC + DC contribution), is sampled by an analog-to-digital converter (ADC). Another ADC collects the second-stage PPG signal (the AC portion only) from the output of the differential OPA circuit that has a positive input from another microcontroller DAC.

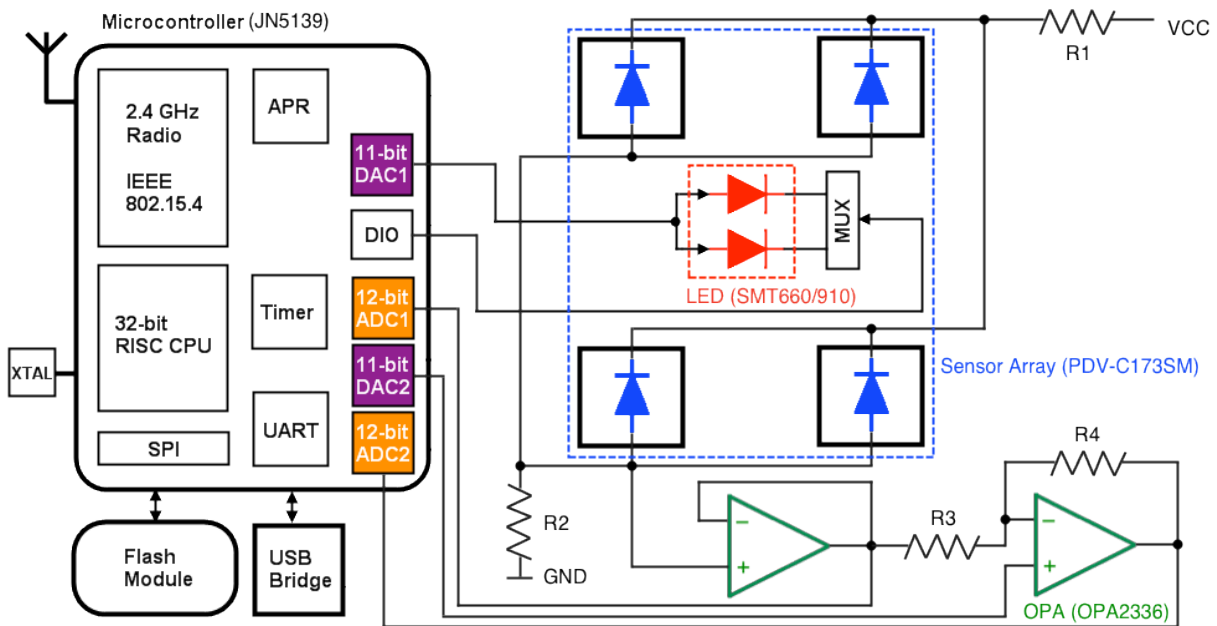


Figure 4.1. Circuit-level system layout. Coordinated by the microcontroller, signal baselines are digitally extracted as an alternative to conventional filtering.

No filters are used in the signal acquisition process, whose elements will be introduced in the section *Closed-Loop System* later in this chapter. The battery (unstable power source) is isolated from the PPG excitation and collection circuitry, since it is powered by the

microcontroller’s analog peripheral regulator (APR). Normally, the pulse oximeter uses a wireless link to communicate with a receiver on a PC, and data are stored on the PC through a MATLAB graphical user interface (GUI). A mini-USB connection can provide a wired interface to the PC while the battery is recharged. If neither the wireless link nor the USB connection is available, sampled data will be temporarily stored on the flash memory module (e.g., for store-and-forward applications).

AC Extraction and Drift Resistance

The first-stage PPG is characterized by a large DC portion and a small AC portion, as in Figure 4.2. The goal is to extract the second-stage AC signal by eliminating the DC component. (In many systems, a high pass filter extracts the AC signal.) If the DC portion instead remains, then obtaining hundreds to thousands of digitization levels in the AC portion over its small voltage range requires an ADC of very high precision (e.g., 16-bit), which is inappropriate for a low-cost, low-power-consumption device. This extraction, or DC removal, process is executed by the OPA unit. Its role is expressed as

$$S_2 = G \times (V_{ref} - S_1) \quad (1)$$

where S_1 and S_2 are the first-stage and second-stage signals, respectively, G is the gain of the OPA, and V_{ref} is a user-defined reference voltage that functionally equates to the DC signal level. To show an upward-oriented PPG peak during systole as with a blood pressure curve, V_{ref} is connected to the positive pin of the OPA, effectively inverting the AC signal amplitude prior to digitization.

S_1 is naturally unstable, as both its AC and DC levels are influenced by changes in intrinsic blood flow, extrinsic motion, respiration, background light, etc. These factors cause drifting in S_2 . The input voltage range, or digitization range, of the 12-bit ADC is set to [0, 2.4] V, so one digitization level is $2.4 \text{ V} / 4095 \text{ levels} = 0.586 \text{ mV}$. For example, given a gain $G = 30$ and a constant V_{ref} , one digitization-level increment in the DC signal results in a decrement of 30 digitization levels in S_2 according to (1). As in Figure 4.2, S_2 may drift 0.3 V (512 digital levels) in 10 seconds, which is unacceptable because the signal will eventually clip at the lower bound of the sampling range, and clipped data mean signal corruption. To address this issue, (1) implies that one can adjust one or more elements on the right side to adjust the value of S_2 on the left. In

this effort, a V_{ref} adjustment is employed to resist S_2 drifting, since V_{ref} is an output of the DAC and can be easily updated. V_{ref} is defined as

$$V_{ref} = MA(t) + V_+ \quad (2)$$

where $MA(t)$, the estimator of the DC component, V_{DC} , is a W -point (e.g., $W = 256$) moving average of the first-stage signal over the time interval that ends at t . V_+ (the adjustable term) is added to $MA(t)$ to ensure that V_{ref} makes S_2 in (1) positive.

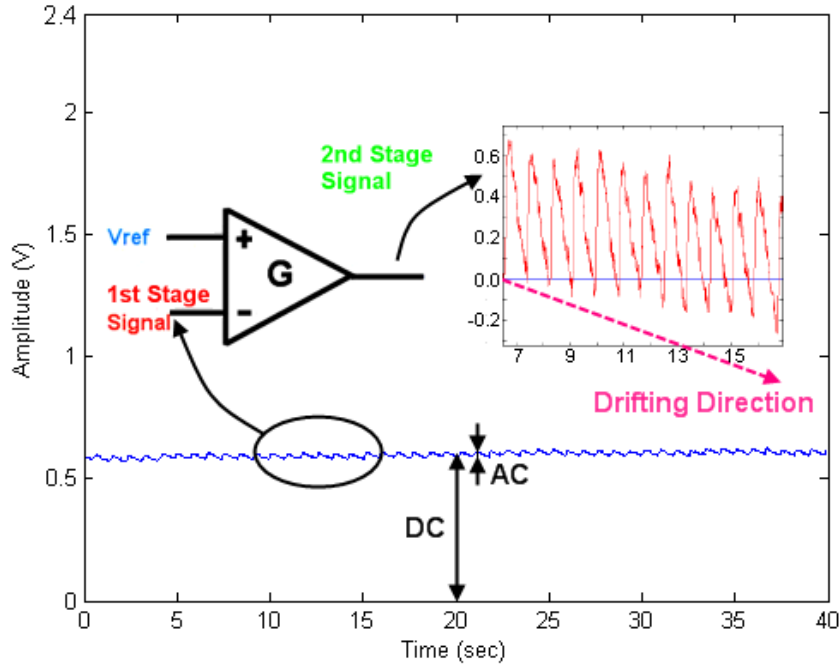


Figure 4.2. A differential amplifier with gain G compares the first-stage PPG (S_1) to a DC reference voltage to obtain the second-stage PPG (S_2).

V_{ref} usually varies slowly (several seconds per digitization level change, in an environment with minimal motion and ambient noise), and the V_{ref} adjustment leads to a discontinuity in S_2 . Hence, the V_{ref} data must also be transmitted or stored along with the digitized second-stage data in order to restore the original PPGs, a process called “compensation.”

Figure 4.3 shows a data set from the palm. Collected data are compensated to remove discontinuities caused by V_{ref} jumps, or immediate value changes, in the pulsatile waveform using the following method. Inserting V_{ref} from (2) into (1) and then rearranging the result isolates the first-stage signal, S_1 :

$$S_1 = MA(t) + V_+ - \frac{S_2}{G} \quad (3)$$

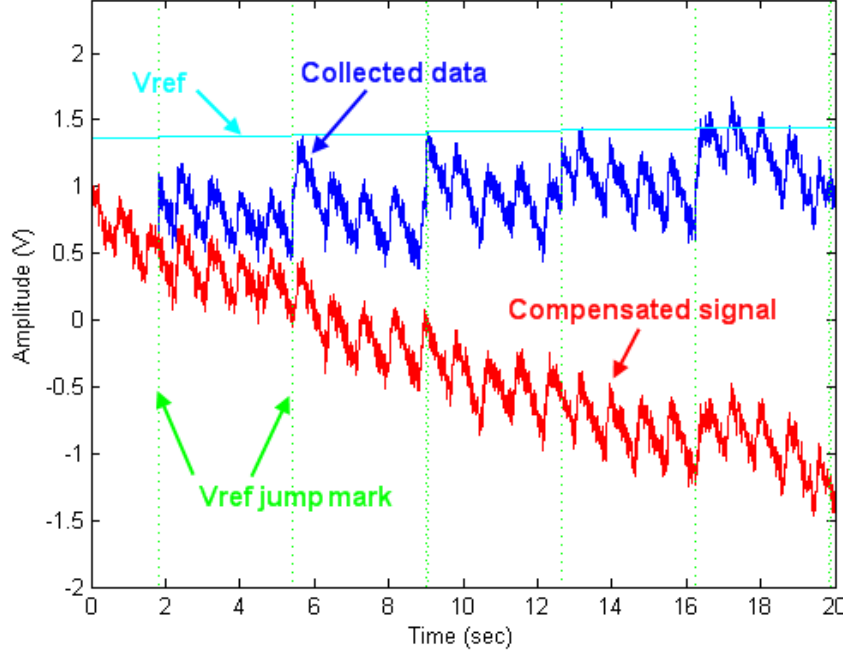


Figure 4.3. Palm PPG data before (blue) and after (red) compensation.

The compensated second-stage signal, \hat{S}_2 , can be represented as

$$\hat{S}_2 = G \times (V_{DC} + V_+ - S_1) \quad (4)$$

where V_+ is added to V_{DC} to ensure a positive \hat{S}_2 . Substituting (3) into (4) yields

$$\hat{S}_2 = S_2 - G \times (MA(t) - V_{DC}) \quad (5)$$

Typically, V_{DC} is an unknown constant, but it is sensible to initially set $V_{DC} = MA(t_0)$ at time t_0 and define $V_{jump} = MA(t) - MA(t_0)$ at time t ($t > t_0$) so that (5) becomes

$$\hat{S}_2 = S_2 - G \times V_{jump} \quad (6)$$

With this method, each PPG can be restored as long as the second-stage signal is unsaturated. The V_{ref} adjustment effectively resists first-stage-signal drifting. For example, in Figure 4.3, the compensated signal drifts below 0 V after 6 seconds. If no V_{ref} adjustment occurs, the subsequent signal is sampled as 0 V. To calculate blood oxygen saturation, V_{ref} is usually considered equal to V_{DC} .

Closed-Loop System

The V_{ref} adjustment mechanism not only helps to realize the AC extraction task; it also results in resilience in the PPG signal. In the control system, as illustrated in Figure 4.4, two closed loops provide stability for the whole data acquisition process. The closed loop in the lower left maintains the S_1 value in a predetermined range, which is set by the Intensity Regulator that controls the led intensity via a DAC. The physical function of this control loop is to maintain the number of reflected photons at an optimal level within the active range of the photodiode, independent of a subject's vascular and perfusion profiles [67]. The closed loop in the upper right prevents S_2 from saturation, since the compensation method described in (6) requires an unsaturated second-stage signal. Upon detecting saturation onset, the Saturation Inhibitor adjusts the V_+ component of V_{ref} , which leads to a corresponding change in S_2 according to (1).

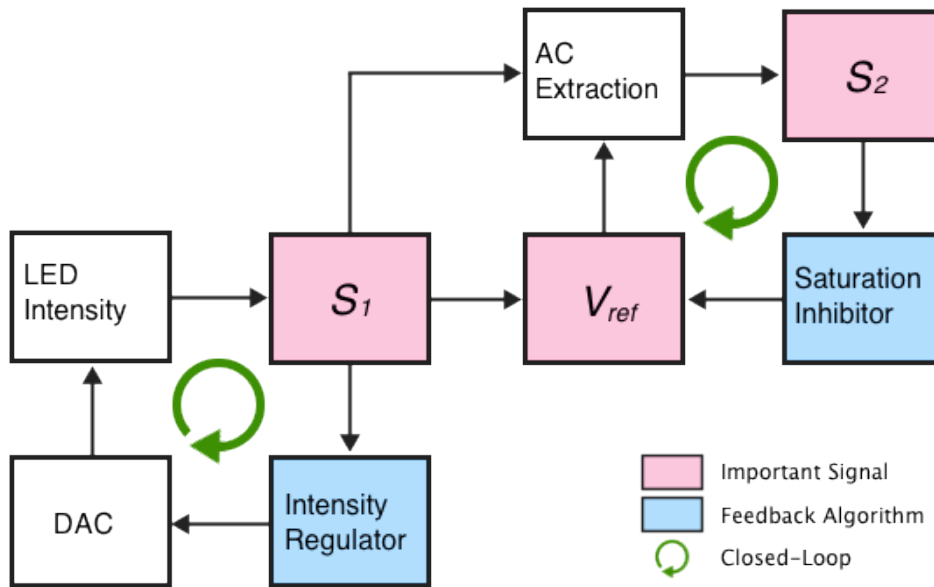


Figure 4.4. Pulse oximeter control flow that illustrates how the first-stage PPG (S_1) can be used to create the second-stage PPG (S_2), where both signals provide feedback to stabilize the acquisition process.

To maintain signal quality, the Intensity Regulator sensitivity should be minimized. When the regulator affects changes in LED excitation level, the influence on the first-stage

signal converted by the photodiode sensor array will be hard to predict because the blood-perfused tissue between the LED and the sensor is an unknown system. Conversely, the sensitivity of the Saturation Inhibitor should be set high to ensure a rapid response to signal drift. Since this adjustment only influences V_{ref} , the native PPG is uncontaminated, and the second-stage signal can be compensated using (6).

In a controlled scenario, ambient noise variations can be ignored. If the desired signal intensity increases as the LED intensity increases, the signal-to-noise ratio (SNR) will improve. However, this implies a saturation risk due to a second-stage signal with too large of a magnitude within a fixed digitization range, in spite of the aforementioned drift-resistant method. Additionally, a more intense LED consumes more power. So, an optimized intensity level should be empirically predetermined as the Intensity Regulator reference.

Removable Noise

In the U.S.A, ambient light often includes a 60 Hz component and the associated harmonic noise, e.g., 120 Hz flicker from full-wave-rectified fluorescent room lights plus higher-frequency harmonics. Most physiological information in a PPG resides in the range of 0-20 Hz. From the Nyquist-Shannon sampling theorem, the lowest sampling frequency, f_s , should then be 40 Hz, but to prevent ambient noise aliasing, sampling frequencies of at least 240 Hz are needed.

Figure 4.5 depicts the magnitude spectrum of a PPG containing ambient noise. The heart rate component is 1.329 Hz, and its harmonics dominate in the frequency band below 20 Hz. At greater frequencies, noise is apparent at 60.02 Hz, 84.43 Hz (unclear source), and 119.9 Hz. Most of this noise is removable by post-processing as long as the sampling frequency is high enough that these noise components do not alias into the frequency range of the signal components of interest. Note that the raw signal exhibits a low SNR compared to PPGs from pulse oximeters that employ filters, but all signal components are intact and many can be removed to create a high SNR (see Figure 4.15).

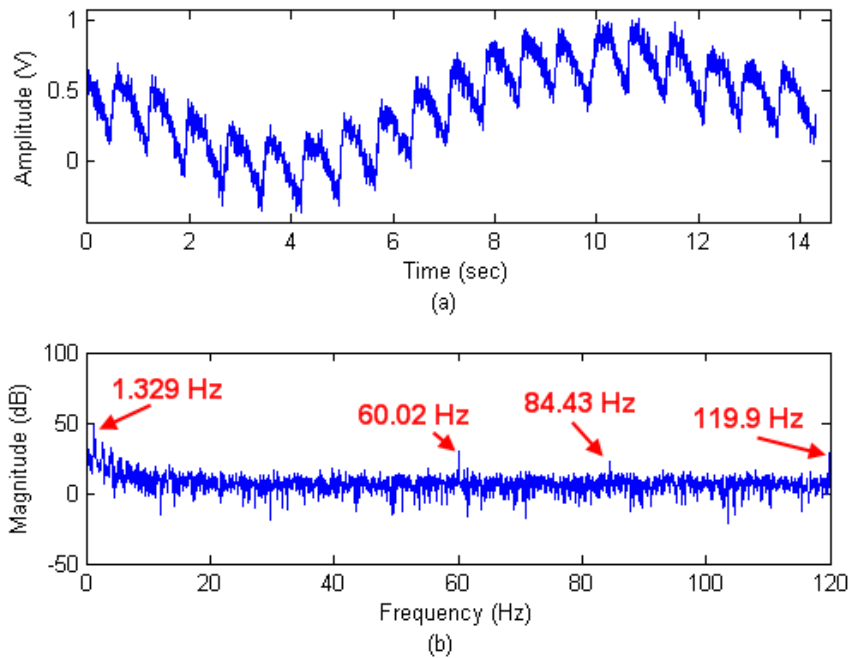


Figure 4.5. An example of removable noise: (a) compensated PPG corrupted by ambient noise and (b) frequency spectrum of these data sampled at 240 Hz.

Motion Artifact

Motion artifact is an issue for a pulse oximeter, especially in reflectance mode [15]. Existing literature focuses on signal processing to reduce motion artifact and restore PPGs [68]. Most methods assume that enough information exists in the corrupted signal for PPG recovery. However, if motion is severe, saturation occurs frequently and lasts for some time, leading to data loss. With this in mind, this development considered motion artifact to be a type of signal drift that can be partially addressed with a drift-resistant method (V_{ref} adjustment); the design does not address motion extraction.

Motion artifact can be classified into two categories: slight and severe. Figure 4.6 demonstrates the severe condition characterized by three axes of hand motion, where the sensor is taped to the finger. Movements are within a 10 cm range and occur at a rate of ~ 1 Hz. The PPG is severely corrupted (the fundamental frequency is 1.028 Hz), and it is clipped at the upper and lower bounds of the digitization range; many AC segments are lost and unrecoverable.

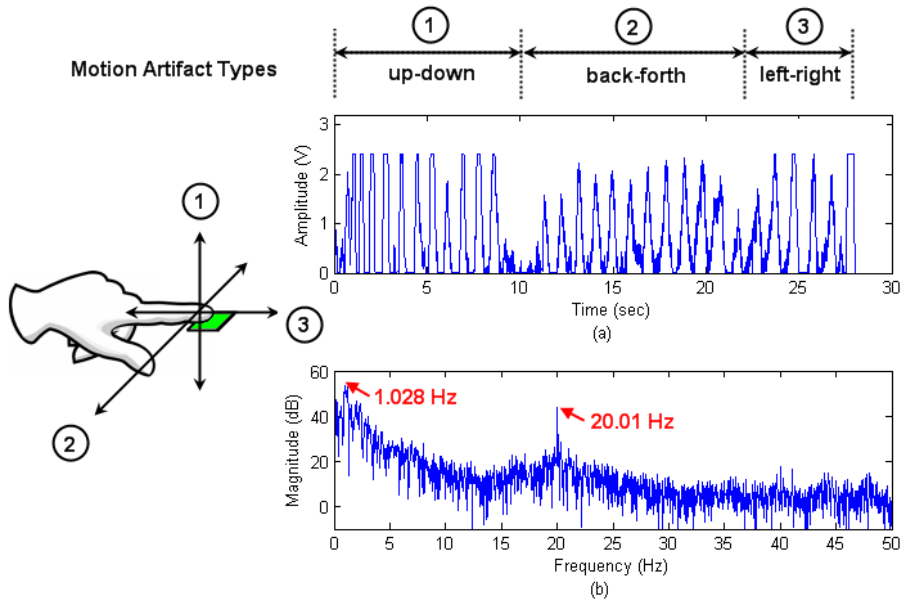


Figure 4.6. (a) PPG severely corrupted by hand motion along three axes. (b) Frequency spectrum of the 28 seconds of data sampled at 100 Hz.

The slight condition refers to, e.g., slow body movements, where a PPG retains its general shape but contains spurious components relative to a still condition. To counteract this type of artifact, the shift-resistant method is promising and relies on the setting of an optimal assignment rate and window width for V_{ref} adjustment. The DAC assigns the V_{ref} value to the positive amplifier pin, and that voltage remains constant until the next V_{ref} assignment to the DAC. The window size of the moving average filter (the DC estimation time delay, or count) and the rate of assigning V_{ref} to the DAC (not the rate of V_{ref} variation) influence the second-stage signal. An extreme case occurs when the window size $W = 1$ data point and the rate of assigning V_{ref} to the DAC is $A = f_s$: motion will never influence the signal since $MA(t) \equiv S_1$ and consequently $S_2 = G \times V_+$ according to (1) and (2).

As an illustration of slight motion response, Figure 4.7 shows three experimental records acquired under similar conditions (exaggerated deep respiration activity), where a different moving-average window width, W , and V_{ref} assignment rate, A , is employed in each case. Only subplot (a) offers a reasonable representation of the PPG. A lower assignment rate (b) or wider window (c) causes the signal to drift severely, and some segments are nearly saturated. The empirical parameter pair ($W = 256$, $A = f_s$) was adopted for V_{ref} adjustment.

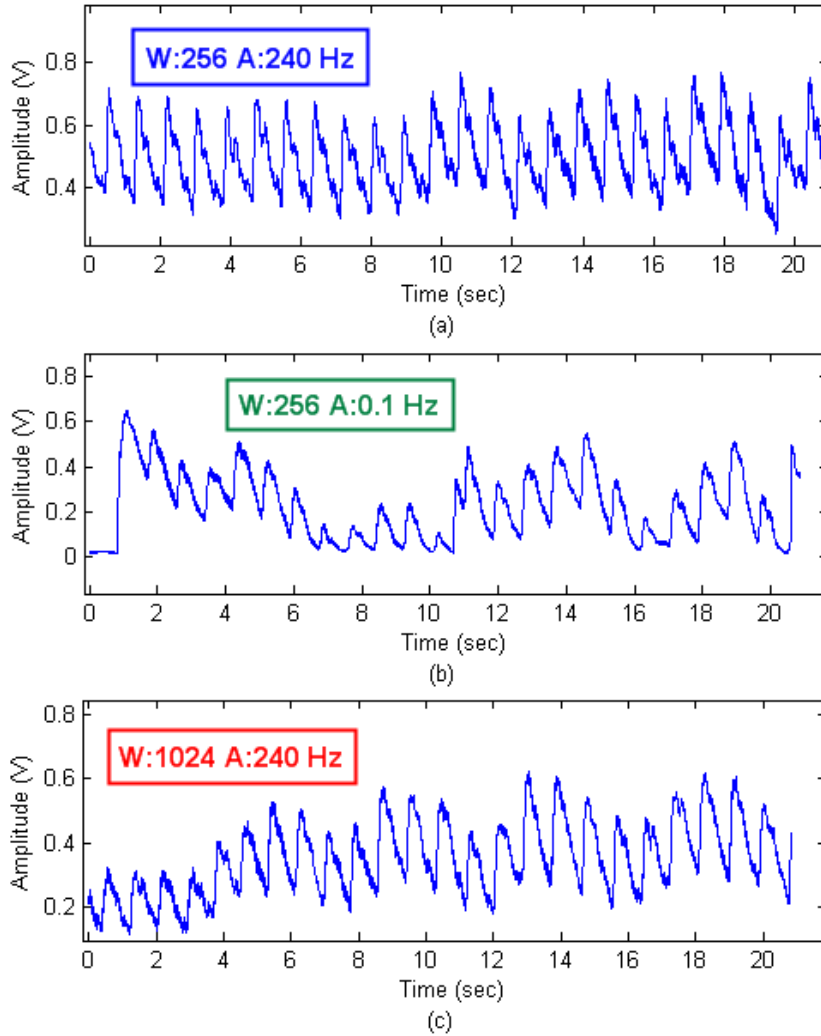


Figure 4.7. Three uncompensated PPGs acquired at $f_s = 240$ Hz under similar slight-motion conditions but with different parameter pairs (W , A).

MATLAB Interface

A MATLAB GUI allows a user to set/view communication parameters, visualize PPGs, process these data in real-time (e.g., digitally filter a signal with a linear-phase filter), and store raw data to files, making it a helpful development tool (see [59] for a full description). Figure 4.8 illustrates an example data set obtained by this GUI, where acquisition options (e.g., Serial Port, Sampling Rate, Signal Channel, and Signal Processing Type) are specified on the left control panel. The upper axes display the raw PPG and baseline for the near-infrared channel, whereas the lower axes show the real-time calibration coefficient, R , calculated from the magnitudes of

the fundamental red/infrared frequency components using a Fourier transform method [64]. R is updated every 0.5 seconds using the previous 4 seconds of PPG data. An overall SpO_2 value is achieved by calculating the median or mean of 40 consecutive R values (in a 20-second segment) and inserting the result into a pre-determined linear calibration equation.

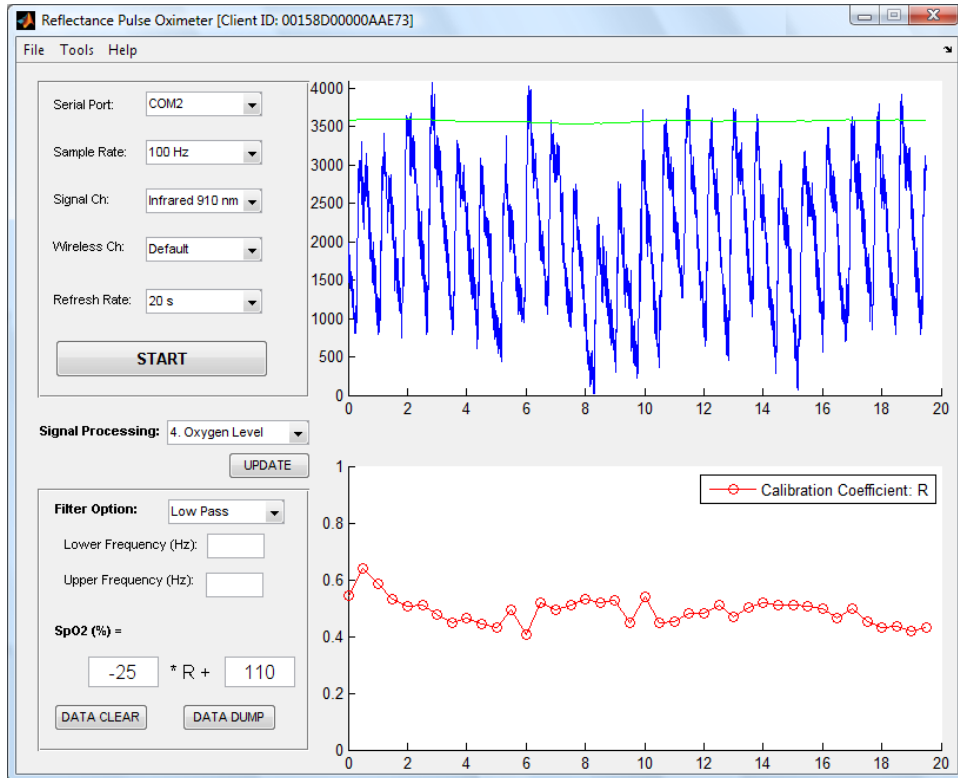


Figure 4.8. Pulse oximeter MATLAB GUI. In this example, a series of calibration coefficients (lower right) is extracted from the current data (upper right).

Device Prototype

Figures 4.9 and 4.10 contain top and bottom views of the pulse oximeter prototype, which consists of four main modules: microcontroller module, excitation LED module, signal sampling module, and power management module. The main printed circuit board is 41 mm by 36 mm, excluding the antenna board. This hardware combines functionality from the Jennic JN5139-EK020 development kit with lessons learned from an earlier reflectance pulse oximeter design [66].

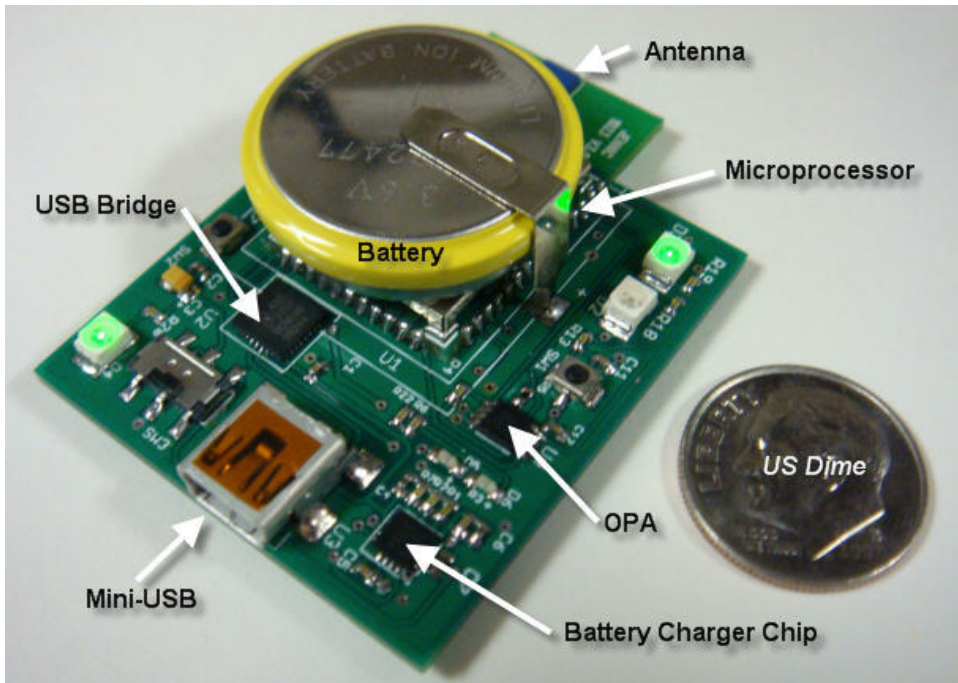


Figure 4.9. Top view of the wireless reflectance pulse oximeter.

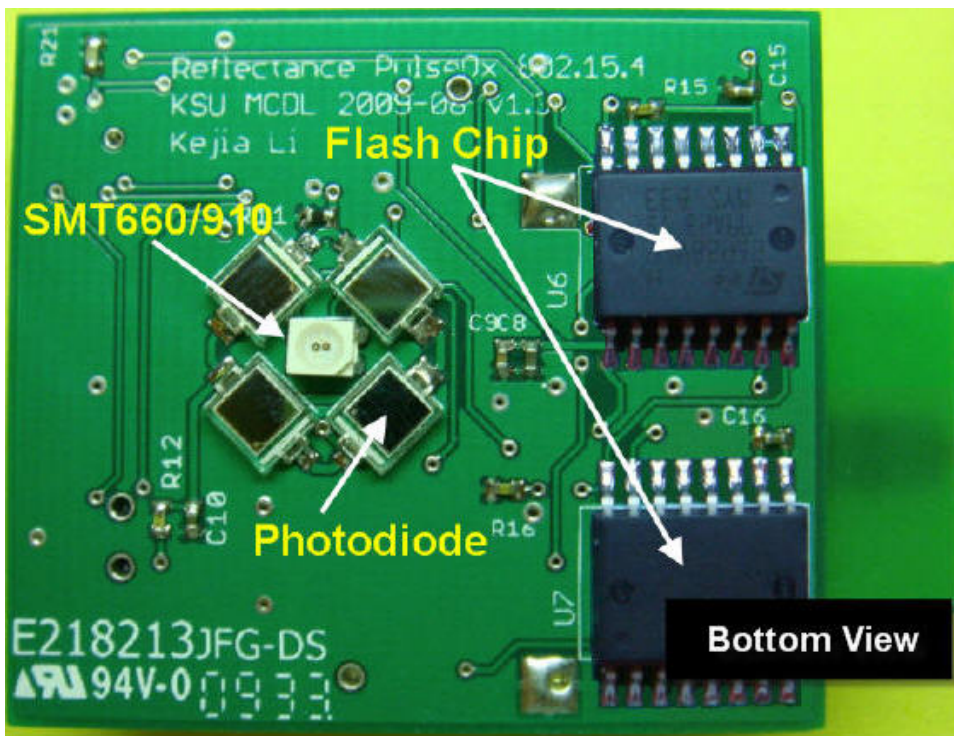


Figure 4.10. Bottom view of the wireless reflectance pulse oximeter.

A microcontroller module is the prototype kernel. The JN5139 wireless module, designed for robust and secure low-power wireless applications, integrates a 32-bit RISC processor with a

2.4 GHz IEEE 802.15.4 (ZigBee) transceiver, 192 kB of ROM, 96 kB of RAM, a mix of analog and digital peripherals (including four 12-bit ADCs and two 11-bit DACs), and up to 21 DIO ports. The wireless link requires the most current, with a TX (transmitter) current draw of 38 mA and an RX (receiver) current draw of 37 mA. The CPU consumes 7.75 mA at full speed, and the current required by the peripherals (ADC, DAC, UART, Timer, etc.) is less than 1 mA in aggregate. The JN5139 sleep current (with an active sleep timer) is only 2.6 μ A.

The excitation LED module uses a low-cost Marubeni SMT660/910 bi-color LED with a typical forward current of 20 mA and forward voltages of 1.9 V and 1.3 V for the 660 nm and 910 nm sources, respectively. The 11-bit DAC output (0-2.4 V) provides excitation signal modulation by managing the power supply for the excitation LED module.

The signal sampling module consists of OPA circuitry connected to the sensor array. Four API PDV-C173SM high-speed photodiodes are connected in parallel; their responsivity to wavelengths above 650 nm is more than 0.3 A/W. The photodiodes are arranged radially around the central LEDs and maintain a source/detector separation of 3-5 mm. The OPA chip contains two amplifier units. The sensor array signal is buffered at the first unit and amplified by the second unit.

The power management module includes two chips: (a) a Silicon Labs CP2102 USB-to-UART bridge that powers the pulse oximeter when the USB connection is detected and bridges data communication to the host and (b) an STMicroelectronics L6924D battery charger system with an integrated power switch for lithium-ion batteries which charges the battery when the USB connection is detected. An LIR2477 3.6 V lithium-ion rechargeable button cell with a capacity of 180-200 mAh serves as the power source when the USB connection is absent.

Memory chips, indicators, and buttons are also housed on the board. Two Numonyx M25PX64 64-Mbit flash memory chips with SPI bus interfaces provide storage space when the pulse oximeter works in offline mode; each consumes 20 mA of current while being accessed.

Measurement Data

The pulse oximeter prototypes were used to acquire hundreds of PPG records from 48 different subjects that are 20 to 64 years old. Experimental results in this section were acquired in an indoor environment utilizing the prototype pulse oximeter. The results are categorized according to conventional location (fingertip) versus other locations (wrist, earlobe, temple, etc.).

Figures 4.11 through 4.14 illustrate 25 seconds of representative fingertip data from a 24-year-old subject. Both channels of PPG data, red and near-infrared, are uncompensated. The AC values of the near-infrared channel (Figure 4.11) offer 1.2 V peak-to-peak (i.e., 2048 digitization levels), and the AC values of the red channel (Figure 4.12) offer fewer digitization levels: about half compared to the near-infrared channel. Even without the use of analog or digital filters, the signal demonstrates distinguishable period and amplitude information useful for HR and SpO₂ determination. The SNRs of the raw near-infrared and red PPGs are 8.0, and 3.0, respectively. As shown in Figure 4.13 and Figure 4.14, up to seven harmonics reside in the spectrum of the near-infrared data (the inset shows frequency components above 5 Hz), and six distinguishable harmonics reside in the spectrum of the red data. Additionally, the PPG information and noise components (e.g., 60 Hz and 120 Hz grid noise) are clearly separated in the frequency domain. To further refine the signal, a properly designed digital band pass filter can be applied.

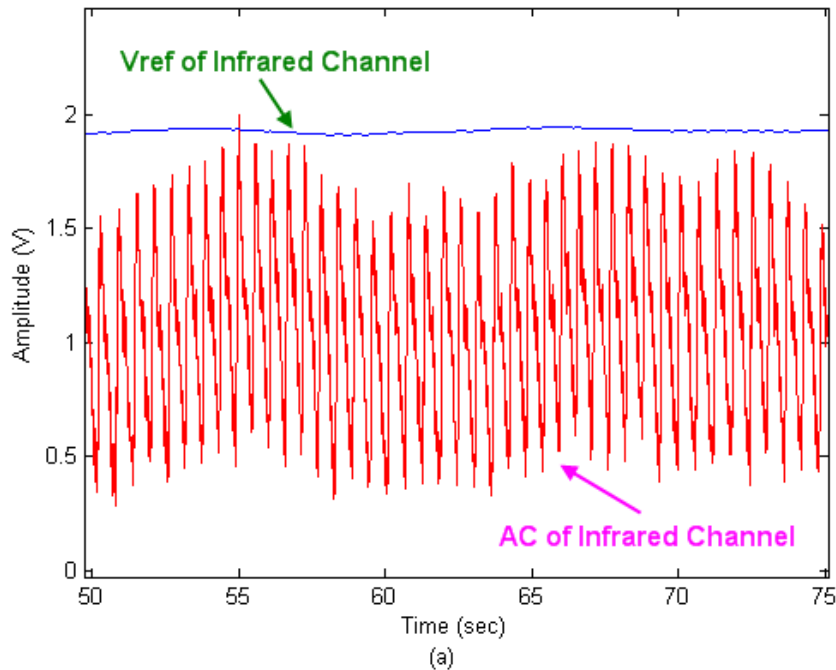


Figure 4.11. Fingertip results: 25 seconds of near-infrared PPG data.

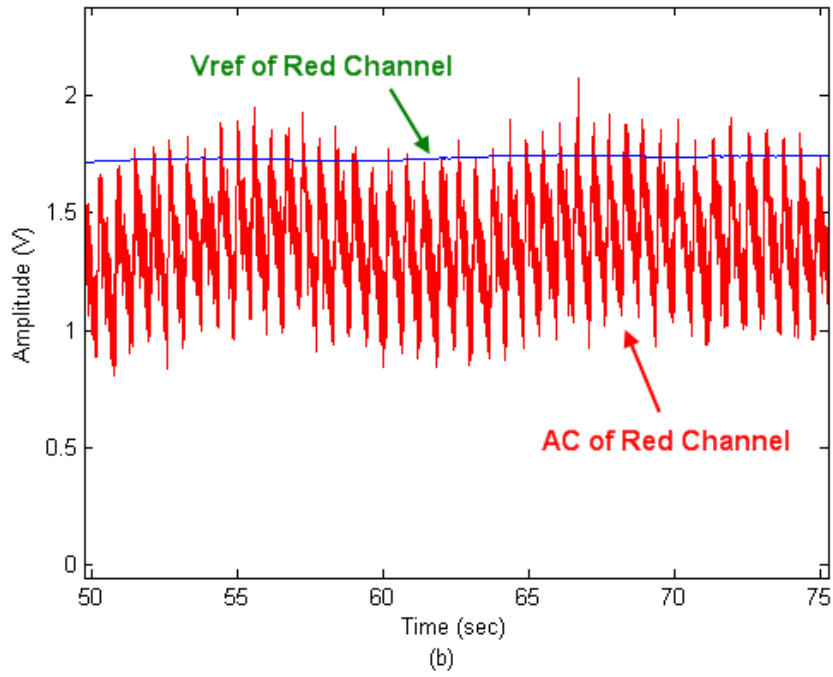


Figure 4.12. Fingertip results: 25 seconds of red PPG data.

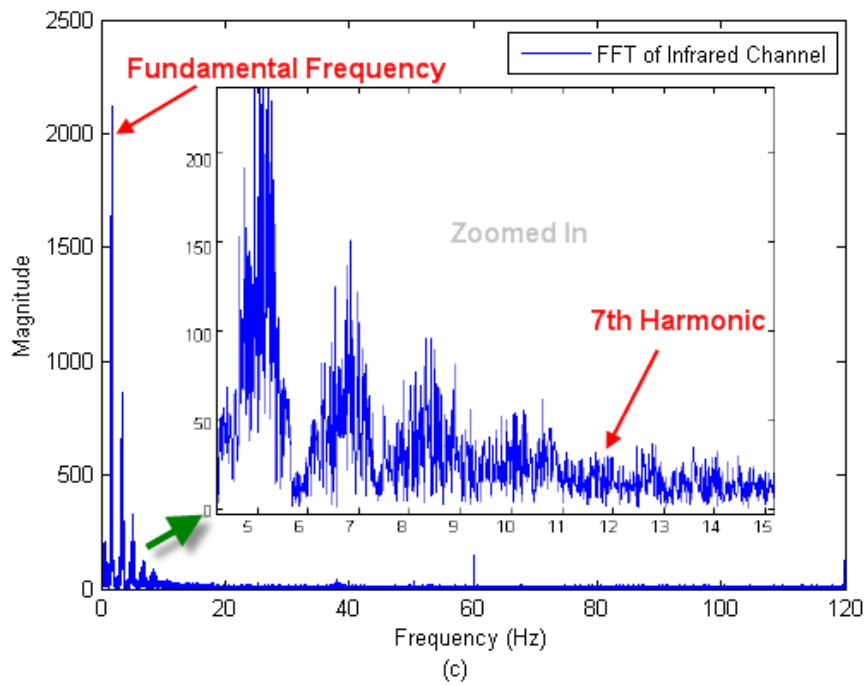


Figure 4.13. Fingertip results: 25 seconds of near-infrared magnitude spectra.

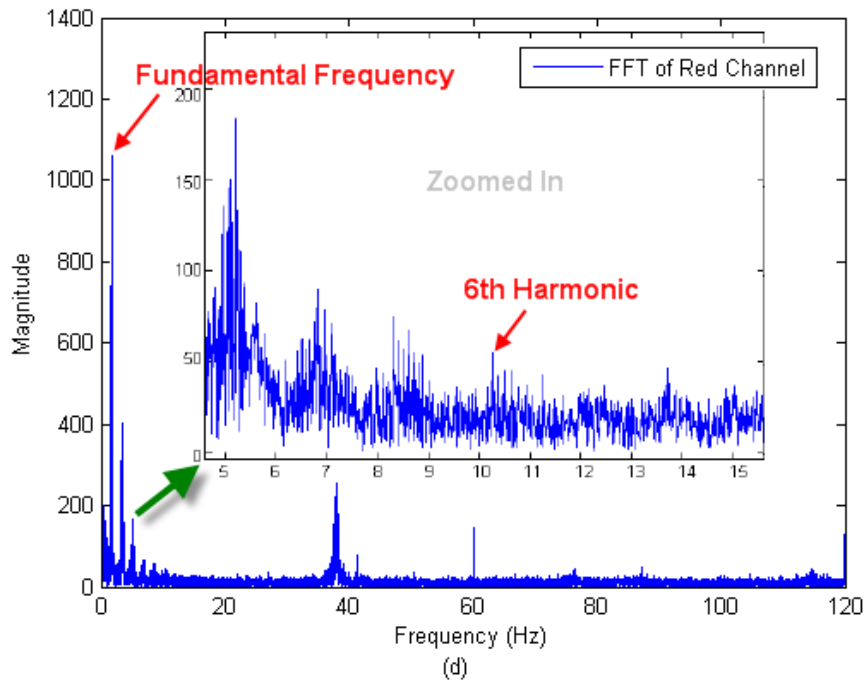


Figure 4.14. Fingertip results: 25 seconds of red magnitude spectra.

Figure 4.15 displays another short segment of an experimental fingertip data set, where the raw near-infrared PPG is accompanied by its real-time filtered form within a MATLAB GUI. The filter is a 200th-order low pass filter with a 10 Hz cut-off frequency realized by the MATLAB function `firls()`: a linear-phase FIR filter that uses least-squares error minimization. This high order filter causes a time delay of $t_d = (n-1)/(2f_s) = 0.414$ seconds, where $n = 200$ and $f_s = 240$ Hz, where the time delay helps to visually separate the original and filtered waveforms. Since the peak-to-peak noise of the filtered signal is too small to be recognizable (< 1 digitization level) the SNR is assumed to be $> 2048/1$ if the signal amplitude is 1.2 V.

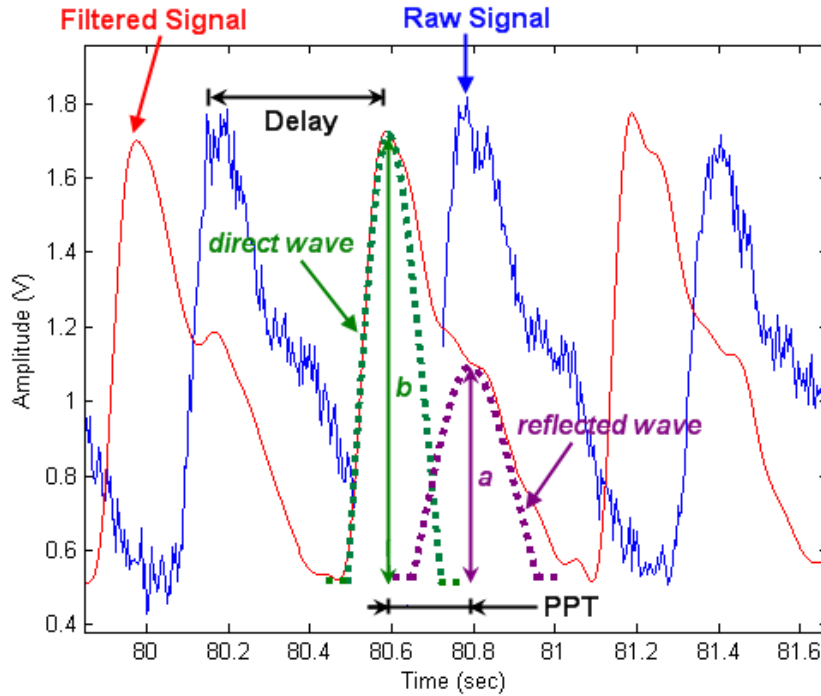


Figure 4.15. Fingertip signal processing and digital volume pulse (DVP) analysis.

Digital Volume Pulse (DVP) Analysis. Cardiovascular parameters other than HR and SpO₂ can be accurately extracted [47], [56], [58] given the quality of this DVP waveform. For example, the peak-to-peak time (PPT), as marked in Figure 4.15, can be used to calculate pulse wave velocity, which correlates to arterial stiffness, and “a” and “b” are used to calculate the reflectance index, which correlates to endothelial function. Additionally, as noted in the Introduction, these unfiltered PPG waveforms could potentially lead to improved assessments of blood pressure and stroke volume via light.

Figure 4.16 displays experimental data from the wrist at the three placement locations depicted in Figure 4.17. The signal quality in location 2 is obviously lower relative to the SNR in locations 1 and 3, but all three are suitable PPGs. At present, it is difficult to consistently obtain high quality PPG data from the wrist; that often requires the application of pressure to bring the optical sensor closer to the major arteries [64]. An operation to achieve the same effect (i.e., bending the wrist at about a 45° angle), was usually employed if the PPG had a low SNR. Since subjects demonstrate a variety of different arterial locations and depths at the wrist, sensor placement flexibility is essential to acquire commendable data sets at this body location, which was also noted in [69].

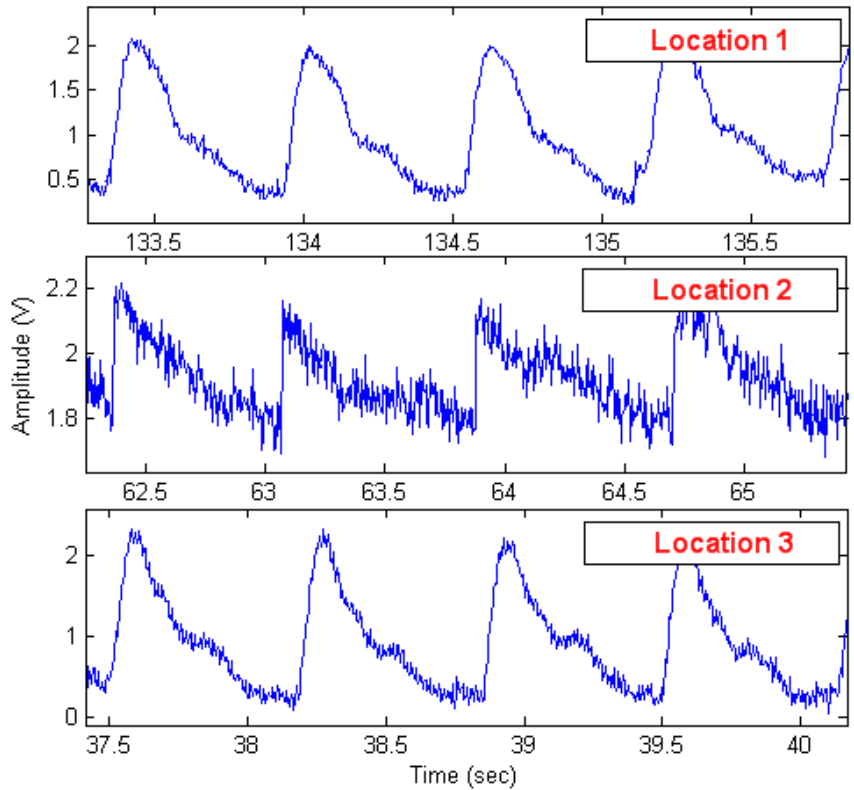


Figure 4.16. Wrist PPGs corresponding to the placement locations in Figure 4.17.

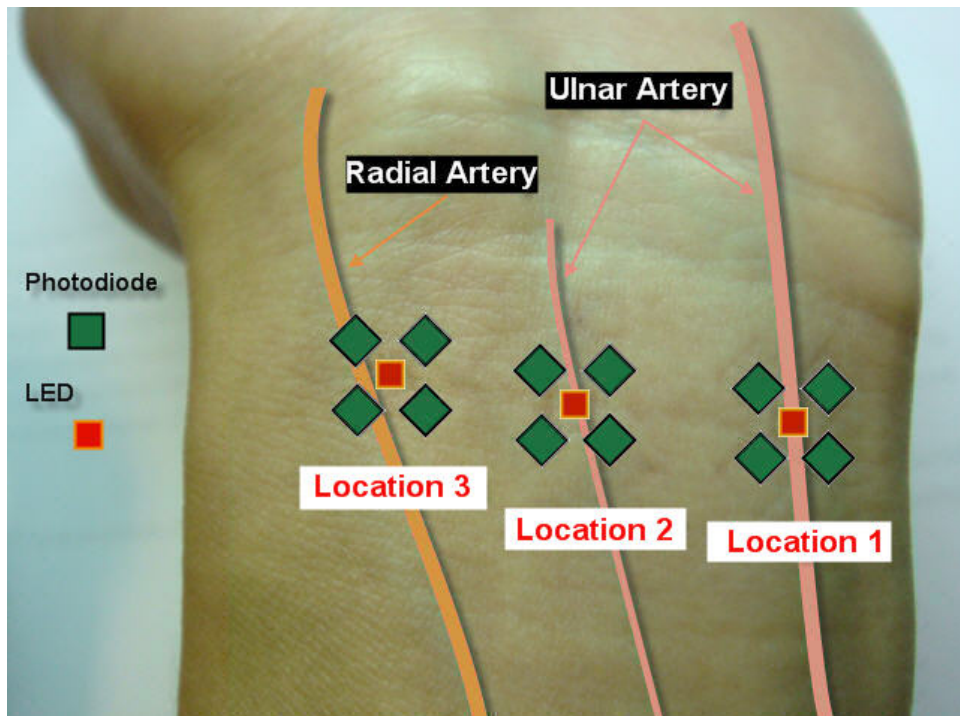


Figure 4.17. Pulse oximeter measurement locations on the left wrist.

New Pulse Wave Velocity (PWV) Estimation Approach. Given the capability to acquire quality PPG data from the wrist (where the SNR dramatically improves with post-filtering), a new approach to estimate PWV has been evaluated by the authors [70]. This approach compares near-infrared PPGs from two synchronized pulse oximeters placed at the fingertip and wrist of the same hand. PWVs can be estimated from several time differences/delays extracted from corresponding features on the two PPGs.

Figure 4.18 displays two channels of data acquired from the earlobe. The near-infrared channel has an SNR of 5.7 and a peak-to-peak range of 1.0 V (1706 digital levels); the red channel has a much lower SNR of 1.8 and a peak-to-peak range of 0.6 V (1024 digital levels).

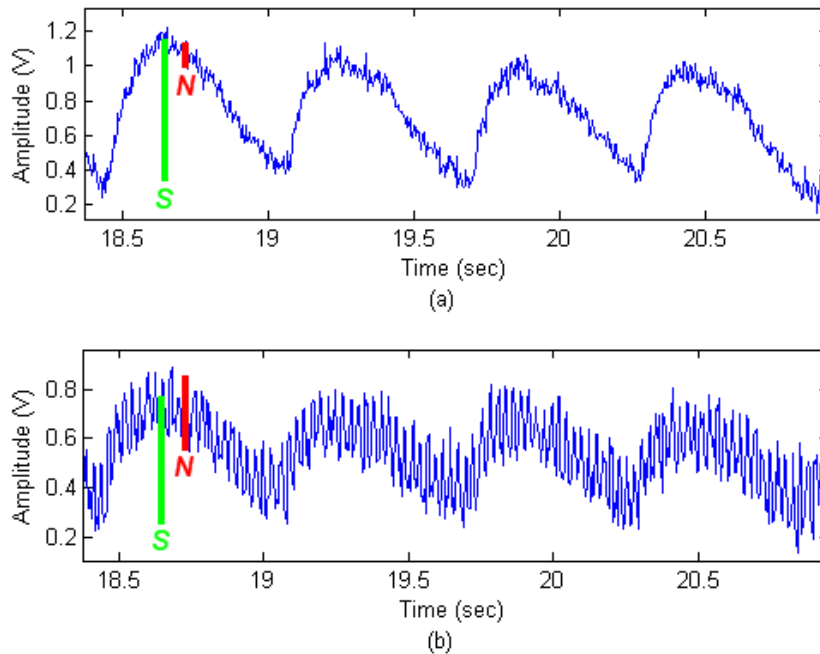


Figure 4.18. Earlobe results: (a) near-infrared channel and (b) red channel.

Respiration Activity Analysis. Figure 4.19 (a) displays 120 seconds of experimental data from the temple that include respiration activity and a swallowing motion. There are 33 respiration cycles present during the 120-second recording time (i.e., the respiration rate is 0.275 Hz). An FFT was applied to ascertain the visibility of these events in the magnitude spectrum, as noted in Figure 4.19 (b). The peak at 1.679 Hz corresponds to the subject's heart rate (100.7 bpm) and the 0.266 Hz frequency component is likely the respiration rate.

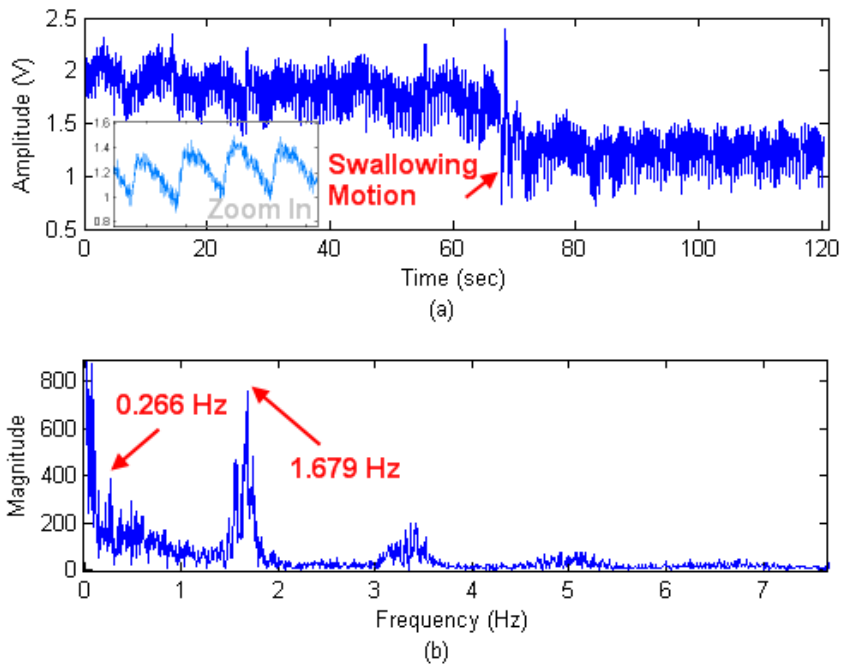


Figure 4.19. Temple results: (a) time-domain PPG with respiration and swallowing motion and (b) the corresponding frequency-domain spectrum.

Chapter 5 - Onboard Tagging Technology

Medical devices can be categorized according to their ‘intelligence quotient.’ Most medical devices are ‘dumb:’ they leave data interpretation to a clinician or host system, although they may locally display these data and provide notifications that, e.g., indicate whether parameters go out of range or whether measurements have been correctly acquired. These devices are easier to regulate because their functional state spaces are limited and predictable.

Alternatively, a ‘smart’ device might make contextual decisions based upon acquired data, including changes in how those data are processed or alterations to its operational state. A smart device might also change its modes given remote commands. Such devices would be clinically useful but difficult to verify and regulate since their operational state spaces would be significantly larger and more complex. Devices that control other devices add a further layer of complexity and are not a feature of most systems slated for FDA regulation. Note that devices that simply offer more features are not necessarily ‘smart’ (though they may be marketed that way) – contextual decisions must play a role.

To further the dialogue regarding how the medical community might move from dumb to smart devices, it is valuable to specify basic issues that drive the use of dumb devices. These include (a) hardware limitations (e.g., a low-power wearable device may host a microcontroller but offer limited processing, storage, and communication resources), (b) software and algorithm hurdles (e.g., besides the obvious resource limitations, a clinically-effective expert system on a small medical device requires broad collaborations), and (c) the need for clinical verification and validation, including FDA approval.

Efforts have been made to improve the intelligence level of some formerly dumb medical devices. For instance, most commercial pulse oximeters can indicate the presence of motion artifact through an alarm. In that case, front panel readouts for heart rate and blood oxygen saturation will not update until valid data recommence. Various research efforts have focused on the detection and reduction of motion artifact, primarily within the context of the viability of the pulsatile photoplethysmogram (PPG) [68]. However, most of these algorithms are computationally complex relative to the normal physiologic parameter extraction process and would be a challenge to implement on a resource-limited device.

Even a standardized ‘motion detected’ indicator in a pulse oximeter data stream is a good step forward towards a ‘smart’ pulse oximeter. While some manufacturers utilize this feature, it is customized for their device and often used only internally in the machine. This kind of indicator is referred to as a ‘tag’ in this dissertation, as tags have wide use in other daily contexts. For example, a price tag describes a commodity’s price, manufacturer, category, etc. Blogging or video blogging services such as YouTube use tags on entries to classify, search, and share information.

The term ‘tag’ as defined here similarly provides concise but meaningful information to a medical device as well as to other devices that receive its data. Any meaningful tag must be sensible and keep the device’s original functionality and data intact. A tag should update with newly acquired data, meaning it should only be valid for a specific data segment (see Figure 5.1). Such properties help to ensure, within reason, that a new device which employs tags is “substantially equivalent” to, e.g., a formerly approved device, allowing a 510(k) mechanism for U.S. device approval [71].

This chapter discusses onboard tagging technology, which if standardized can improve medical devices and the healthcare services they provide. For devices with limited hardware and battery resources, onboard tagging promises advantages due to its light-weight computational requirement and its potential to optimize transmission time as well as the data that are sent to a host system.

Technology Overview

This section addresses the need for onboard/real-time tags, the types of tags one might employ, the information they can convey, and some suitable tag formats. The assumption is that tags mark the original data stream provided by a medical device, as in Figure 5.1. Once data frames accumulate to form a data segment with a predetermined length, a tag will be appended to the end of the segment.



Figure 5.1. Tags embedded in an original data stream.

Onboard and Real-Time Tags

An ‘onboard’ tag implies that the tagging procedure is performed on the device and does not depend on external resources (human or machine) such as the receiver. Onboard tagging should be emphasized for these reasons:

1. The receiver or host system cannot be guaranteed to be active, available, or capable enough to provide assistance, especially in the case of mobile devices.
2. A device should know its data best, and a data point or segment is ideally processed right after being sampled, e.g., even prior to being packed into a data frame.
3. A device is immediately improved if it can indicate characteristics of its current data and/or make independent decisions based on these tags, e.g., automated sleep control.
4. For low-power, wearable/mobile devices, wireless data transmissions are the primary consumers of battery power. Tags can help a device to (a) process its own data with a goal of sending processed parameters instead of raw data over the telemetry link and/or (b) decide when transmissions should be avoided, such as cases where invalid data are undesired and their transmission would unnecessarily reduce battery life.

New tags can be created or updated as physiologic data are acquired and therefore should be attached directly to these data streams by the medical devices that provide the real-time data processing and transmission. This clearly requires that onboard tagging also be performed in real time. The following are other reasons why onboard tagging can or should be accomplished in real time:

Tags are a condensed information set. They require few processing and storage resources compared to the routine tasks of a medical device.

1. Old tags become irrelevant when new data emerge.
2. Delayed tags offer limited correlation value when compared against current data segments.

Tag Type and Content

While medical devices may record numerous pieces of information in tag form, tags fall into three broad categories (Table 5.1). Clearly, the stakeholders in column two may change depending upon the care scenario and device type.

Table 5.1. Tag Types Used in Onboard Tagging Technology

Type	Interested Group	Primary Content of Interest
I	Device Designer	Hardware, algorithm, and data states
II	Signal Analyzer	Signal character and statistics
III	End User	Alerts, physiologic indices, & data quality

Type I tags indicate whether the device is functioning as designed. Such a tag could describe an internal system error, a system variable, a data sampling state, a control flow state, etc.; a role much like the information provided to a firmware developer when debugging a prototype. System level tags within this group could serve multiple usage roles. For example, a tag that represents a hardware failure state could also serve as a warning for an end user or clinician.

Type II tags focus on the signals themselves. Devices such as 12-lead ECGs have multiple signal channels, and each channel can have separate tags. Typical tags in this category may mark easy-to-discern statistical features related to one data segment, e.g., extreme values (valley and peak), amplitude (peak-to-valley excursion), number of cycles (amplitude swings), rising time (a counter may increment by one when a current data point is larger than the previous point), falling time (counterpart of rising time), etc. Looking at the extreme values as an example, if a tag indicates a certain number of lower-bound and/or upper-bound values for the given sampling range (e.g., a 10-bit ADC has the sampling range of [0, 1023] digitization levels), then the tagged signal segment is saturated to some level.

Type III tags speak to clinical data viability and are a user friendly version of type I and type II tags. For an end user that has little or no professional knowledge regarding device design or signal interpretation, type III tags make it easy for them to understand what is happening to the device and the measurement data. For example, if a power source voltage goes below a threshold and generates a type I tag, then a second type III tag like “low battery” could be created as an alert. In another instance, a type II cycle-count tag may change abruptly compared to prior tags, presenting an apparent inconsistency, so a type III tag may be used to note that the current value has low reliability, the current signal quality is poor, or a longer measurement time is required.

A type III tag offers a higher level description of type I and II tags. It should arguably not be directly attached to the original data stream since it carries indirect information. A digital event log would be a more appropriate repository for these tags. As implied earlier, a transformation mechanism on the device would be responsible for presenting Type III tags to a common user. If the transformation process equates to a task such as the interpretation of raw data to create a physiologic index, then a prudent verification and validation procedure should be performed in advance.

Tag Format and Indices

Since tags accompany the data stream, and new tags apply only to the current data segment, it is sensible to look at the data frame structure first. To give this discussion context, the custom pulse oximeter that employs serial communication is used as an example throughout this section. For this device, a data frame that contains one data point for each of four channels is laid out in Figure 5.2. The frame length is 18 bytes, with the first 8 bytes assigned to a unique MAC address, the next 8 bytes assigned to the four signal channels, and the last two bytes appended for frame integrity.

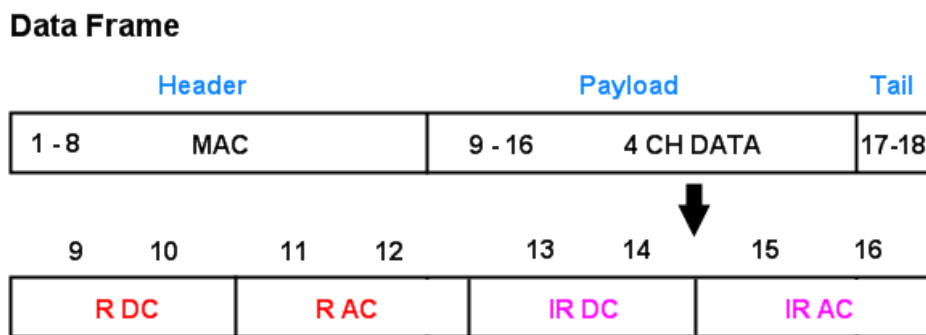


Figure 5.2. Data frame structure for a custom pulse oximeter that employs serial communication (R: red channel; IR: near-infrared channel; AC & DC: pulsatile and baseline samples).

A similar structure is adopted for a tag frame (see Figure 5.3). A tag frame header is an eight-byte MAC address, except it is not the real device MAC address. A preset virtual MAC

(VMAC) is uniformly assigned to all tag frames. When the system detects a VMAC sequence, it knows the frame is a tag frame that holds tags for the current data segment.

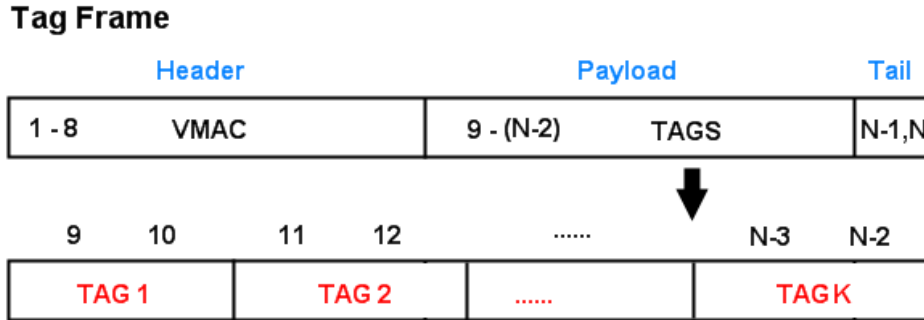


Figure 5.3. Tag frame structure for a custom pulse oximeter. The frame length is N bytes containing K tags.

The tag frame length of N bytes is not fixed – it depends on how many tags the frame conveys. If we use a uniform tag size of 2 bytes, a tag frame holding K tags has a length of

$$N = 2K + 10 \tag{7}$$

bytes, where $2K$ bytes is the payload size and the remaining 10 bytes hold the header (8 bytes) and the tail (2 bytes).

One issue regards the length of time a tag is active and the rate at which tags are assigned. Two indices, *tag active time* and *tag density* are introduced here. Tag active time denotes the duration for which a tag is active – usually one data segment. E.g., if a tag frame is yielded after each three-second data segment, *tag active time* = 3 seconds and *tag delay* = 1.5 sec (the average reporting delay over a three-second moving segment). Tag density is defined as

$$\text{Tag Density} = \frac{\text{Tag Frame \#}}{\text{Data Frame \#}} \tag{8}$$

For example, if the data frame rate (sampling frequency) is 240 Hz and the tag active time is 3 seconds, then the *tag density* = $1/720 = 0.14\%$. Tag density is important. A high tag density can lead to over-reporting and unnecessarily high bandwidth requirements, whereas a low tag density can increase the tag delay and make it more difficult for a system to respond to data anomalies in ‘real time.’

Chapter 6 - Application A: Feature Detection on a Pulse Oximeter

Wireless, wearable medical devices promise a new degree of independence to those who need frequent health monitoring. The competent design of such devices can save resources and increase the integrity of the electronic health records that they populate. For instance, when corrupt data are locally processed and/or sent over a wireless link, computation cycles and battery power are wasted because these data are minimally useful for clinical decision making, with the caveat that corrupted data used for medical decision making can lead to high false-alarm rates and other serious consequences.

Intelligent algorithms that run on devices and assess signal quality prior to wireless transmission and storage can help to mitigate these issues at a time when wireless devices and electronic patient records are proliferating. Such algorithms can (a) alert users/clinicians to the presence of poor data so that device adjustments can be made or (b) better inform devices that must autonomously determine follow-on tasks. Consistent with the prior chapter on tagging, this chapter uses pulse oximeter PPGs as a context to illustrate how intelligent algorithms can be implemented on a device intended for mobile applications.

The idea of intelligent pulse oximeter algorithms is not new; other efforts have focused, e.g., on motion artifact detection and reduction using various methods [68], [72-75]. However, the complexity of these algorithms (Fourier transform [72], independent component analysis [68], [73], wavelet transform [74], and adaptive filtering [75]) makes them unwieldy for microcontroller implementation (i.e., onboard application). Further, most of these approaches do not deal with signal classification prior to motion removal, which is meaningless in, e.g., a saturated PPG. This emphasis on motion artifact is sensible, as motion is the primary corruptor of PPG data and offers a substantive signal processing challenge, since the desired PPG signal and the undesirable motion artifact occupy the same frequency range and are not independent [76]. A comprehensive approach to data quality assessment in wireless devices would address motion specification, real-time processing, and information fidelity.

1. **Motion Specification.** Motion artifact is defined in a limited sense in the earlier literature, where motion-corrupted PPGs often retain the basic information in a ‘clean’ PPG waveform and are unsaturated. The term ‘severe’ used to describe motion in prior literature [68]

may not always refer to prominent motion, and PPGs designated as ‘clean’ [72] may carry mild motion artifact due to, e.g., respiration and normal tremors. In a real application, the motion type and degree should be considered prior to the use of processing algorithms, since different types of motion influence PPG data in different ways. These motion types include user activities (e.g., sensor placement/adjustment, walking, task movements, gestures, and conversation) and platform movement due to vehicle motion/vibration, furniture movement, and sensor/clothing interaction.

2. Real-Time Processing. Onboard processing is needed on a wireless, wearable medical device if the designer (a) does not wish to waste battery power by sending corrupt or nonessential data and (b) hopes to preserve a complete record of the processed data when the wireless receiver is unavailable. Further, processing must occur in real-time and not interfere with the data acquisition process so as to avoid missing data, data backlogs, and reporting delays. The notion of ‘real-time’ is context dependent, addressing the continuity of the signal processing approach as well as reporting delays imposed by the algorithms. The code implementation method and memory requirement also affect algorithm feasibility. Given a device that reports only heart rate and blood oxygen saturation, an ever-present time delay of a few seconds satisfies the notion of ‘real-time.’ Here, a 1.5-second delay (the average reporting delay over a 3-second moving segment) is tolerated.

3. Information Fidelity. Motion data are typically discarded (e.g., filtered) by PPG processing methods [74], [75], yet these data might help to track user activity types and levels if properly isolated. Ambient noise is removed for granted, yet these data might help to estimate the environment condition, e.g., room light intensity. The formerly annoying information could be used as inputs to autonomous-device algorithms that control device functionality such as auto-sleep and excitation LED intensity controls.

This chapter presents an onboard feature-extraction and decision-making algorithm for PPGs acquired with a custom pulse oximeter sensor. This algorithm affirms the absence of motion, the absence of signal saturation, and the presence of clean PPG data that remain. The algorithm run time is evenly distributed across the firmware cycle and does not diminish the functionality of the original pulse oximeter. Specifically, the algorithm first extracts four waveform features from the latest 3-second data segment and analyzes those features within a hierarchical decision tree that utilizes predetermined thresholds. The algorithm then calculates

physiological parameters from a “compact representation” structure. A MATLAB interface on a personal computer helps to visualize the features extracted, the algorithm flow, and the decision results, where all algorithm-related parameters and decisions are ascertained on the wireless unit prior to transmission.

Most pulse oximeters use similar excitation/collection principles and ratiometric oxygen saturation calculations [77], but implementation methods for circuitry, firmware, sampling, and algorithms differ widely. Such variants of a base design imply algorithm transplantation issues between devices. Therefore note that the algorithm presented here was initially implemented on the pulse oximeter described in Chapter 4 – a nonconventional reflectance-mode device that provides completely unfiltered PPGs. The following sections briefly describe this design to help the reader better understand the new feature-extraction and decision-making algorithm.

Algorithm Flow

Figure 6.1 depicts the feature detection algorithm flow. Four data channels that are customary for pulse oximeter design are continuously sent to a MATLAB interface. The algorithm on the wireless pulse oximeter first intercepts a 3-second segment of the latest four-channel data set, where the PPG data are sampled at 240 Hz since these data may include 120 Hz components from full-wave-rectified room lights. However, 240 Hz is overkill for the feature-extraction process and slows it down, so the algorithm downsamples the signal at a decimation of 8, yielding PPGs sampled at 30 Hz, which still meets the Nyquist criterion for the fundamental frequency and secondary harmonics expected in a quasi-periodic PPG [78].

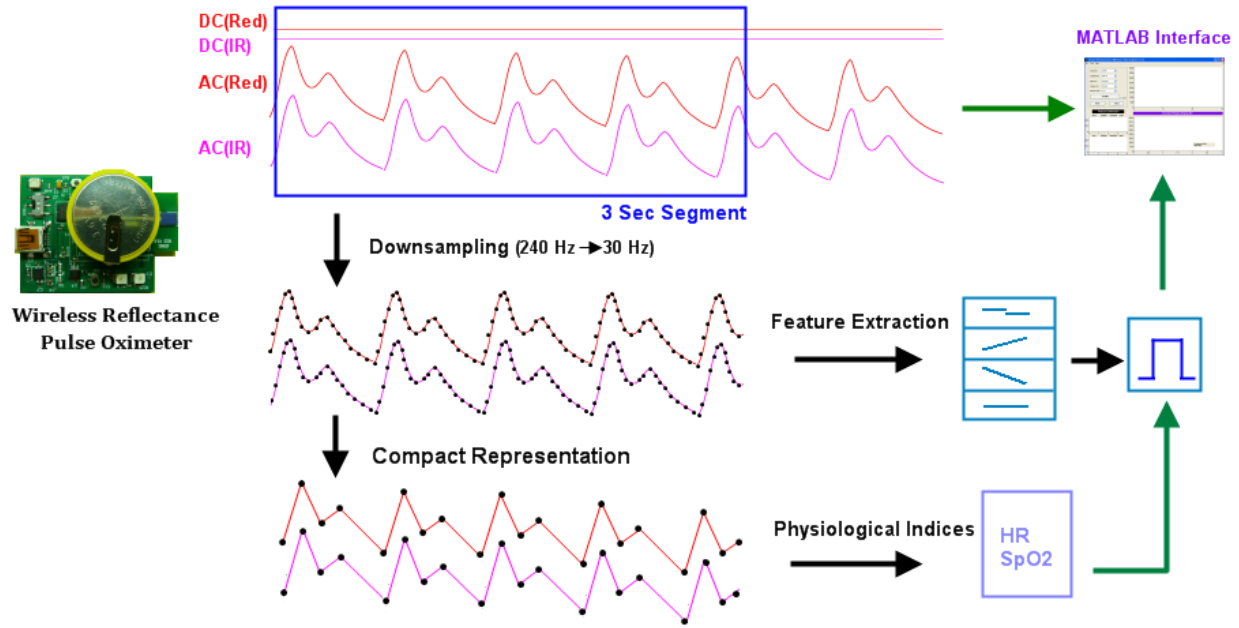


Figure 6.1. Feature detection algorithm flow. The wireless reflectance pulse oximeter appears on the left.

Once downsampled data are available, features are extracted at each sample time and summarized at the end of each 3-second segment. Rule-based decision results [79] are sent to the MATLAB interface. Decisions address (a) the degree of motion detected, (b) the signal saturation status, and (c) PPG quality.

Finally, a “compact” signal representation is created, where the goal is to further simplify each downsampled PPG to obtain the least number of samples that represent its period and amplitude. This simplification is helpful if one wishes to implement such an algorithm on a microprocessor with limited resources. The process starts with triangular structure identification in the downsampled data set, where each set of three consecutive samples is evaluated to see if it creates a triangle with the middle point as a peak or valley, as in Figure 6.2. If the PPG is unsaturated, then two triangular structures are of primary interest as they are interlaced in the “compact representation.” For the left (blue) triangle in Figure 6.2, the middle point is identified as a local minimum, as it is less than the two points on either side of it. Analogously, for the right (red) triangle, the middle point is identified as a local maximum.

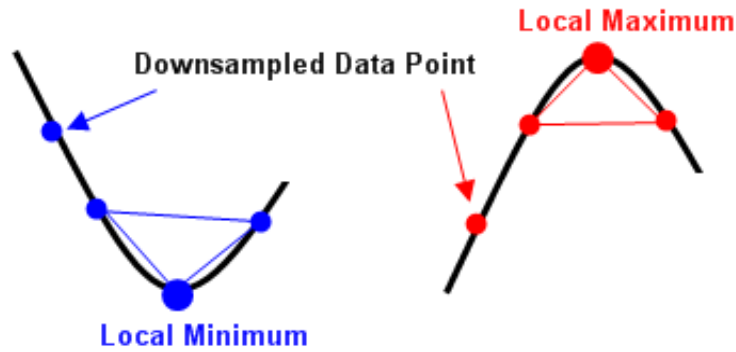


Figure 6.2. Triangular structures identified for the “compact representation.”

If the decision based on the extracted features is favorable, i.e., a valid pulsatile signal exists, then the physiological parameters of heart rate (HR) and blood oxygen saturation (SpO₂) calculated from the “compact representation” are also stored with the raw PPG data. Additionally, they can be used as decision-making inputs, since calculated HR and SpO₂ values should be in a reasonable range.

Signal processing steps are distributed across each 3-second segment, yielding “streaming signal processing.” This makes the best use of resources that remain after the data acquisition, storage, and transmission functions are performed. The feature-extraction algorithm is noted below in pseudo code.

```

While (New raw data point is sampled)
{
    *Update features with the new data point;
    Update downsampled data point to X;
    If (X is ready)
    {
        Update features with X;
        Detect triangular structure with new X;
        If (Triangle detected)
        {
            Add peak/valley to compact representation;
        }
    }
    If (Current segment is done)
    {
        Initialize for next segment;
        Notify main() function;
    }
}

```

*Feature 1 (baseline variation – introduced in the next section) is the only feature updated each time a new raw data point is available.

This code segment is included in the timer interrupt routine of the microcontroller firmware where the normal pulse oximeter control code resides. A “flag” is set at the end of each 3-second segment to be detected by the `main()` function, where the “compact representation” code is implemented using the data stored in a `CR[]` array. The `CR[]` memory space should be released prior to being accessed by the code in the timer interrupt routine. The pseudo code for this part of the algorithm is described below.

```

While (Notified)
{
    Copy compact representation to a new array A;
    For (i = 1 to length of A)
    {
        If ((A(i)+Tolerance) < A(i+1))
        {
            Amplitude = A(i+1) - A(i);
            Cycle count ++;
        }
    }
    Repeat the code for the other channel;
    Calculate HR from cycle count;
    Calculate SpO2 from Amplitudes;
}

```

Feature Selection

Features must be efficient and effective, but highly effective features, e.g., higher order statistics and bi-spectrum analyses as in [80], can require complex calculations. This efficiency versus effectiveness tradeoff challenges the feature selection process. After evaluating many features in light of this tradeoff, four PPG features were selected: (a) baseline variation count, (b) rising time, (c) falling time, and (d) saturation index.

To statistically validate these features, ‘clean’ PPGs were first collected from 20 subjects of ages 20 to 64 years (signal quality like Figure 6.15). Each sequence was longer than 30 seconds to ensure the acquisition of at least 10 3-second segments, yielding 200 segments in aggregate. Features were extracted from each segment, and the mean and standard deviation of each feature were calculated for each person.

Regarding the extraction of Feature 1 (variation of baseline, V_{ref}), one can note that V_{ref} is adjusted to prevent signal saturation and that motion is the predominant cause of saturation and severe baseline fluctuations. Hence, frequent baseline value changes suggest that either motion is present or the signal amplitude is too large, i.e., already saturated. The latter condition is usually

temporary and soon disappears via the LED intensity autoregulation mechanism. (The related saturation index (Feature 4) will be introduced momentarily.)

Feature 1 is the only feature directly obtained from the 720 points in a raw data segment (3 seconds of data sampled at 240 Hz). When the baseline value varies either up or down, Feature 1 is incremented. Figure 6.3 illustrates the Feature 1 statistics from 20 subjects, where the height of each bar is the average number of baseline fluctuations over 10 of each individual's 3-second segments, and the line at the top of each bar is the corresponding standard deviation. All bar heights are less than 120, and the mean of Feature 1 is 47.

A similar chart for motion-corrupted PPGs would be hard to create, as it would require a clear method to quantify motion degree and a sensible means to identify the minimum degree of acceptable motion; these issues commonly arise in the literature that addresses PPG motion artifact. Here, we circumvent it by defining a motion degree index based on the assumption that Feature 1 is proportional to motion severity.

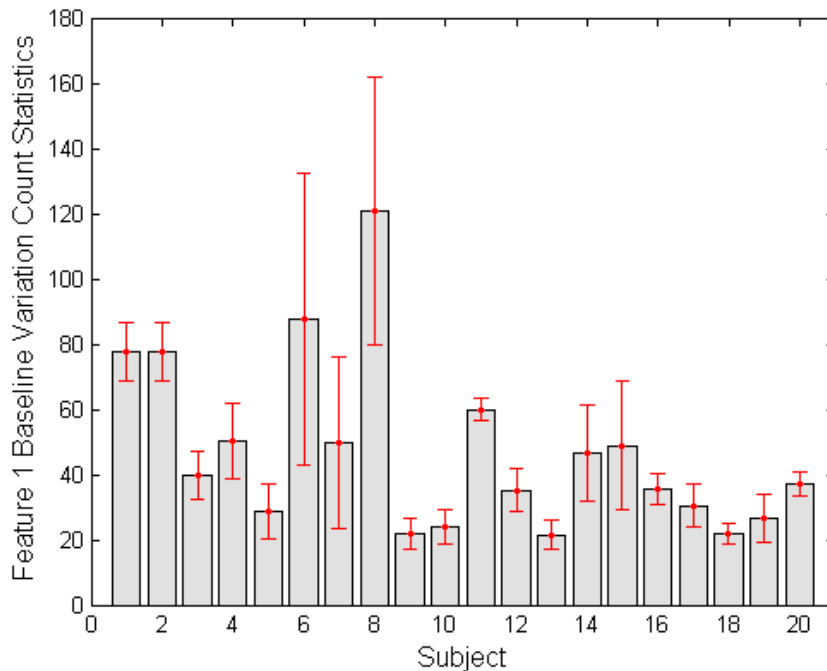


Figure 6.3. Feature 1 (baseline variation count) statistics for 20 subjects.

Features 2, 3, and 4 are complementary, as they track signal trends in 90 points of downsampled data in a 3-second segment at 30 Hz. This pseudo-code applies:

```
Copy 90 points of downsampled data to an Array D;
```

```
For (i = 1 to 89)
{
    If (D(i+1) > D(i))
    {
        Rising count ++;
    }
    If (D(i+1) < D(i))
    {
        Falling count ++;
    }
    If (Both D(i+1), D(i) == 0 or 4095)
    {
        Saturation index ++;
    }
}
```

Figures 6.4 through 6.6 depict feature statistics given the same data used for Figure 6.3. Feature 2 (rising count) has a mean of 28, Feature 3 (falling count) has a mean of 60, and Feature 4 (saturation index) has a mean less than 1. Count fluctuations are slight, as the standard deviation for each is less than 4 for these subjects.

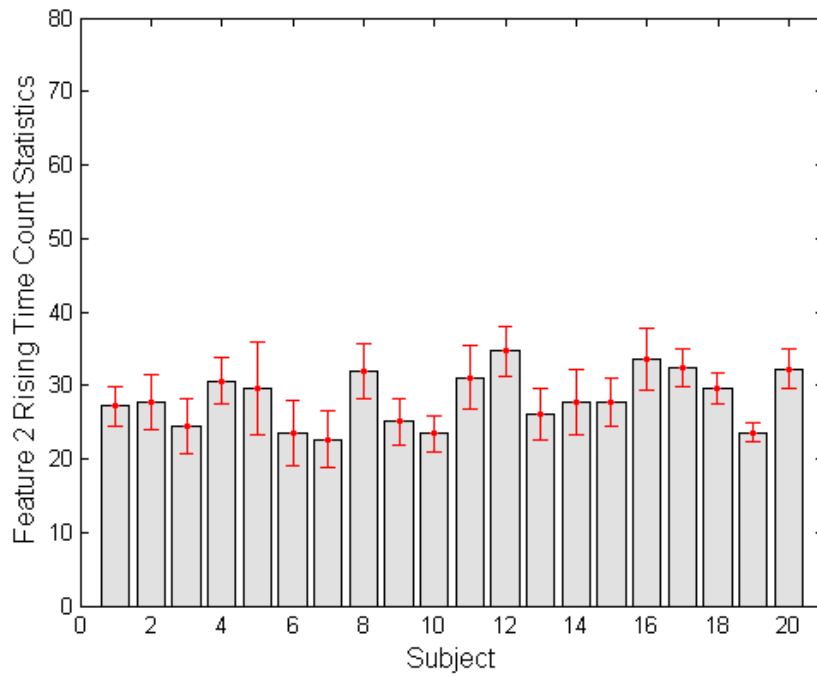


Figure 6.4. Feature 2 (rising count) statistics for 20 subjects.

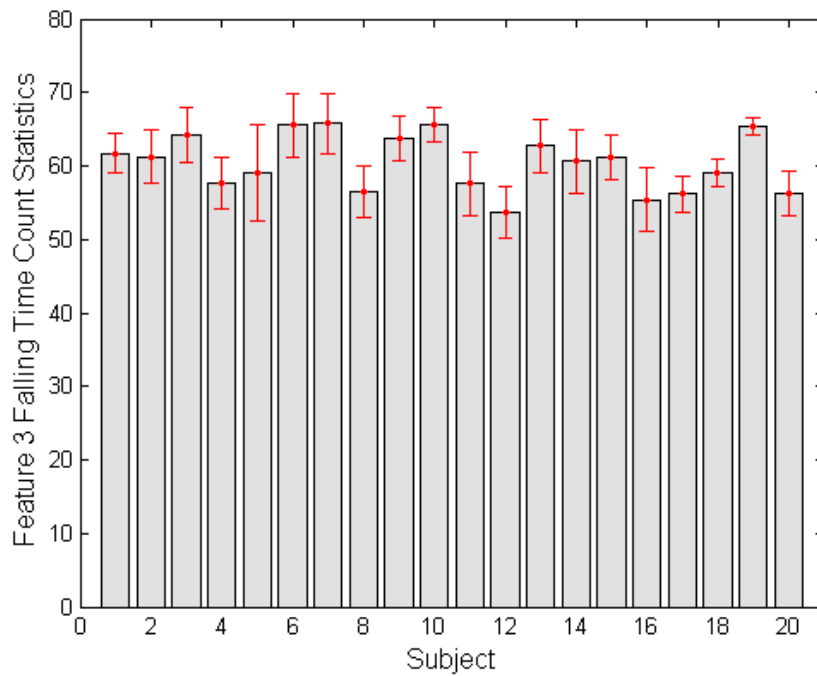


Figure 6.5. Feature 3 (falling count) statistics for 20 subjects.

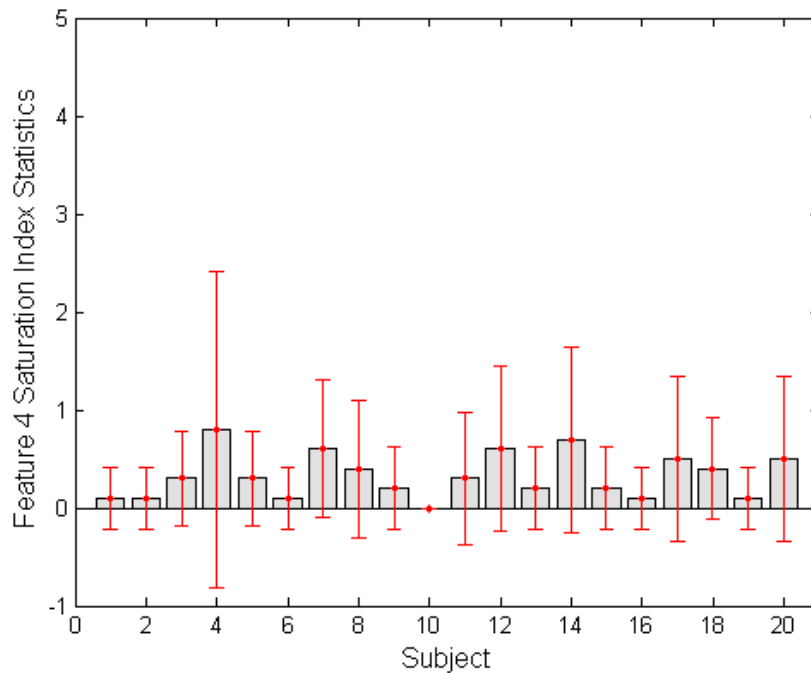


Figure 6.6. Feature 4 (saturation index) statistics for 20 subjects.

The Feature 3 to Feature 2 ratio in Figure 6.7 warrants attention. The mean of 2.2 sensibly represents the asymmetry in each PPG cycle (see Figure 6.15), which features a steep rising slope and a mild falling slope. In other words, the diastolic interval is roughly two times longer than the systolic interval. This special ratio is seldom created by normal motion and ambient noise. For instance, when the pulse oximeter is removed from the patient and laid on a desktop to merely receive ambient light, the resulting signal resembles a sinusoid with a falling/rising ratio close to 1. If motion is simulated, it also usually creates a sinusoid-like signal; periodic motion with a falling/rising ratio similar to that of a normal PPG is unnatural.

Data from this first group of 20 subjects were supplemented with data from 27 subjects to complete model training and validation. In each session, an investigator handed a reflectance pulse oximeter to a subject, who was instructed on how to place the device. Data were recorded during the whole process (lasting for minutes), and these data typically consisted of a relatively large portion of data corrupted with random motion/operation versus a relatively small portion of valid PPG data (see Figure 6.16 for an illustrative data set).

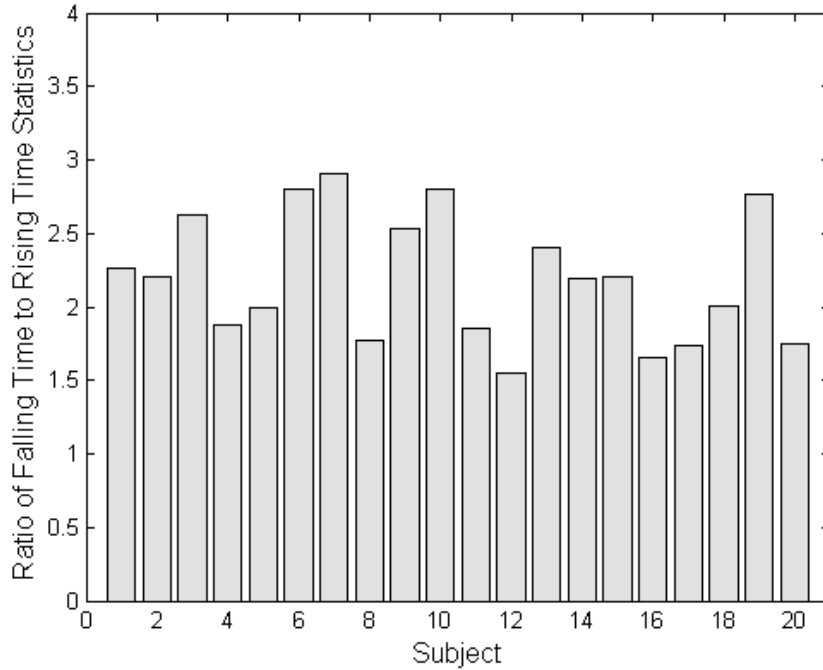


Figure 6.7. Ratio of falling time to rising time statistics for 20 subjects.

Observations and Hierarchical Decision-Making

The four selected features are represented by three observations for use with the decision-making algorithm, as listed in Table 6.1. Inspired by the hierarchical clustering method introduced in the statistical signal classification approach in [81], we use a three-step hierarchical decision-making approach with these four features. The decision hierarchically walks through the following assessments: (a) motion status, (b) saturation status, and (c) signal quality, as in Figure 6.8.

Table 6.1. Observation Description Given the Original Four Features

Feature	Observation
Feature 1: Baseline Variation Count	Observation 1
Feature 4: Saturation Index	Observation 2
Feature 2, 3: Ratio of Falling Count to Rising Count	Observation 3

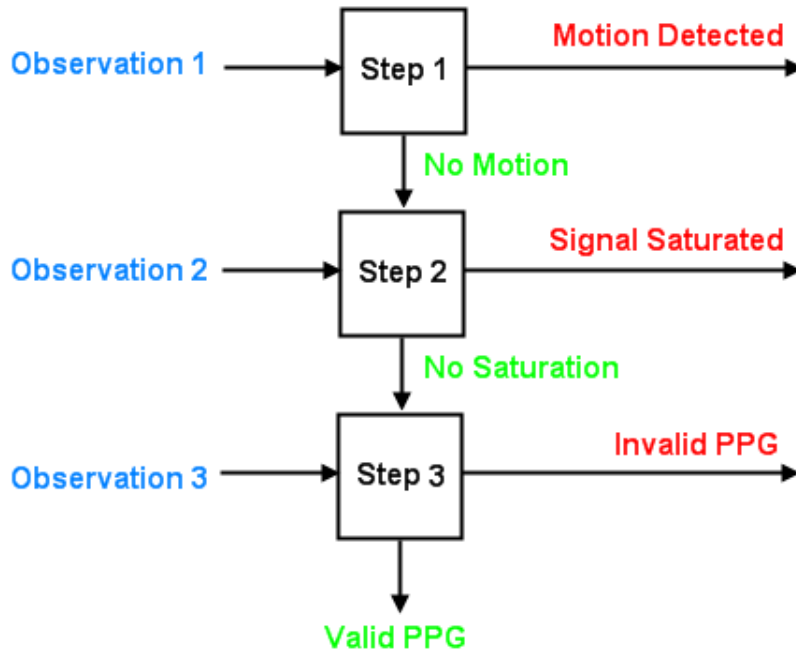


Figure 6.8. Three-step hierarchical decision-making approach.

In Step 1, the motion condition is detected based on a single input: Observation 1. (Decision-rule details are laid out in the next section.) If a decision result indicates that no motion is present, Observation 2 is used to decide the signal saturation status. The decision rule in Step 2 couches the decision result in terms of whether the signal can pass through to the next decision step. For example, a signal of pure ambient noise can easily walk through the first two steps; however, the decision rule for Step 3 that concludes signal quality from Observation 3 (Feature 3 to Feature 2 ratio) will reject the noise signal and disqualify other non-PPG signals. PPG data passing through the entire hierarchy are ‘free’ of motion, ‘free’ of saturation, and valid.

A binary Bayesian hypothesis testing method [79] is used to create the three decision rules in Figure 6.8. Each hierarchical step contains two possible hypotheses, H_0 (null hypothesis) and H_1 (alternative hypothesis), corresponding to two probability density functions, p_0 and p_1 . Taking Step 3 for example, H_0 denotes a valid PPG, whereas H_1 denotes an invalid PPG. Under each hypothesis, an observation x is distributed as

$$\begin{aligned}
 H_0 : x &\sim p_0 \\
 H_1 : x &\sim p_1
 \end{aligned}
 \tag{9}$$

where p_i is normally assumed to be a Gaussian distribution, consistent with the Figure 6.9 and Figure 6.10 histograms. Note that 200 and 1148 segments were used for the histograms in Figure 6.9 and Figure 6.10, respectively.

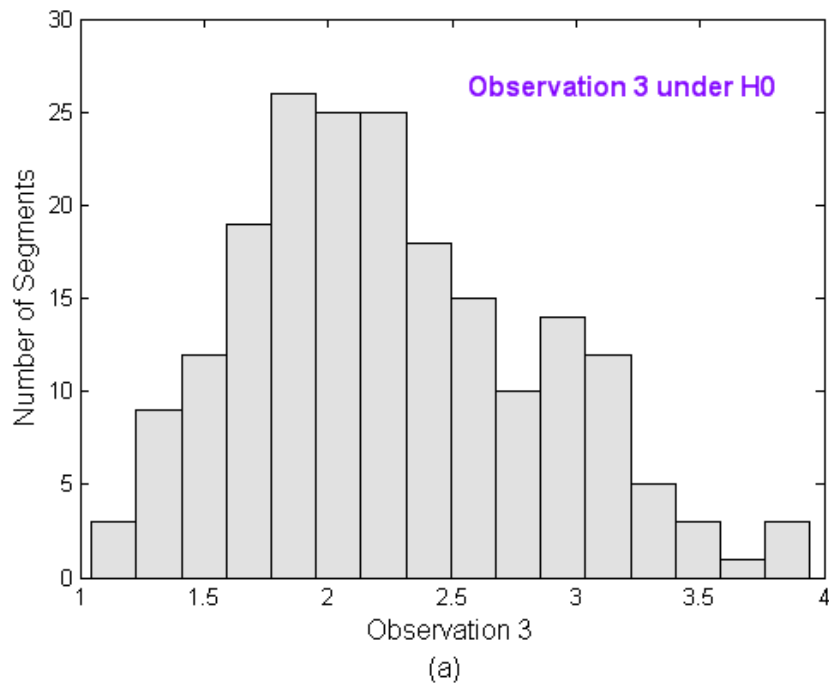


Figure 6.9. Histograms of Observation 3 for valid PPG data.

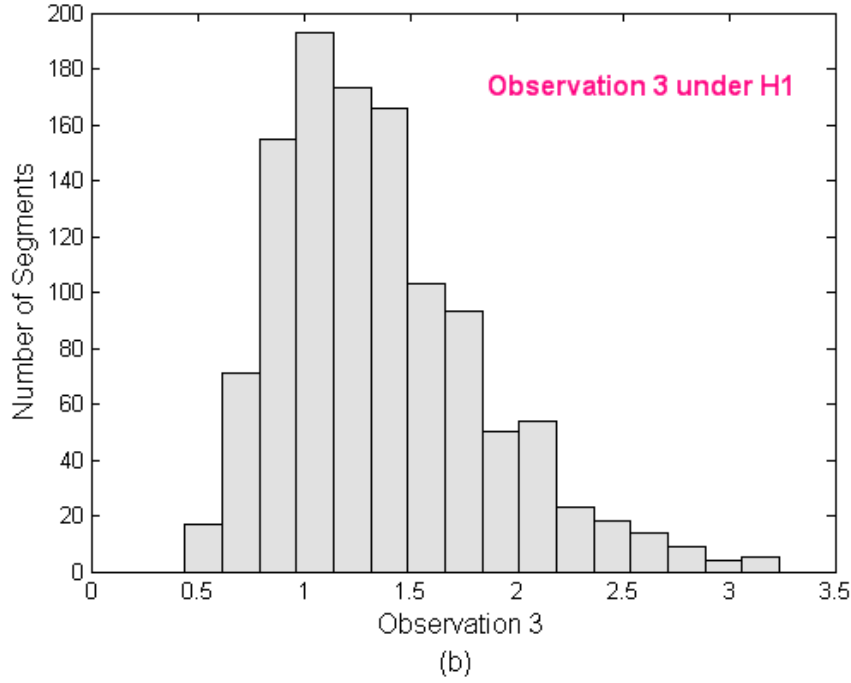


Figure 6.10. Histograms of Observation 3 for invalid PPG data.

The Bayesian decision rule is given by

$$\delta = \begin{cases} 1 & \text{if } L(x) \geq \tau \\ 0 & \text{if } L(x) < \tau \end{cases} \quad (10)$$

where the likelihood ratio, $L(x)$, is defined as

$$L(x) = \frac{p_1(x)}{p_0(x)} \quad (11)$$

and the hypothesis threshold, τ , is defined as

$$\tau = \frac{\pi_0(C_{10} - C_{00})}{\pi_1(C_{01} - C_{11})} \quad (12)$$

where, π_i is the prior of the two hypotheses and C_{ij} is the cost incurred by choosing H_i when H_j is true. Although uniform costs and equal priors ($\tau = 1$) are initially adopted here, these variables can be further utilized in machine learning development. For example, the prior of H_0 increases if valid PPG data have been continuously acquired for a long time, and the cost of falsely accepting corrupted data as valid PPG data can be considered higher than the cost of falsely rejecting valid PPG data as corrupted.

To evaluate the performance of the hypothesis testing method, two indices, P_D (probability of detection) and P_F (probability of a false alarm), are employed for each step, as in Table 6.2. P_D is probability of deciding H_1 when H_1 is true; P_F is the probability of deciding H_1 when H_0 is true. 200 segments were used for each observation under each H_0 ; 1148 segments were used for each observation under each H_1 .

Table 6.2. Hypothesis Definition and P_d for Each Step

Step	H_0	P_D
1	No Motion	Motion correctly detected
2	No Saturation	Saturation correctly detected
3	Valid PPG	Invalid signal correctly detected

H_1 is the opposite of H_0 .

Figure 6.11 depicts the occurrence of Observations 1 and 2 under H_0 . For example, Figure 6.11 (a) plots the number of segments as a function of baseline variation with the goal to set the threshold high enough to achieve a zero P_F , where a signal with no motion will never be tagged as motion corrupted, and a signal that is unsaturated will never be tagged as saturated.

An extra 891 segments from the 27 subjects (33 from each) were available for model validation. If all of the data segments with valid PPG data that are embedded in a generally corrupted data set could be accurately detected, then the validity of the feature detection algorithm could be demonstrated. Note that model training and validation processes are implemented on a PC; training results (hypothesis thresholds) are programmed into the pulse oximeter module for onboard application.

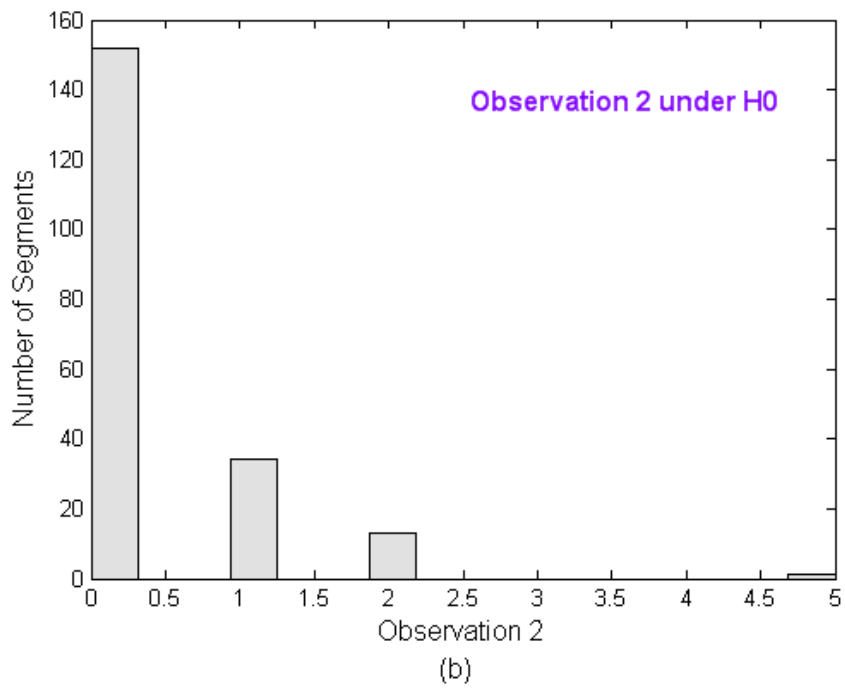
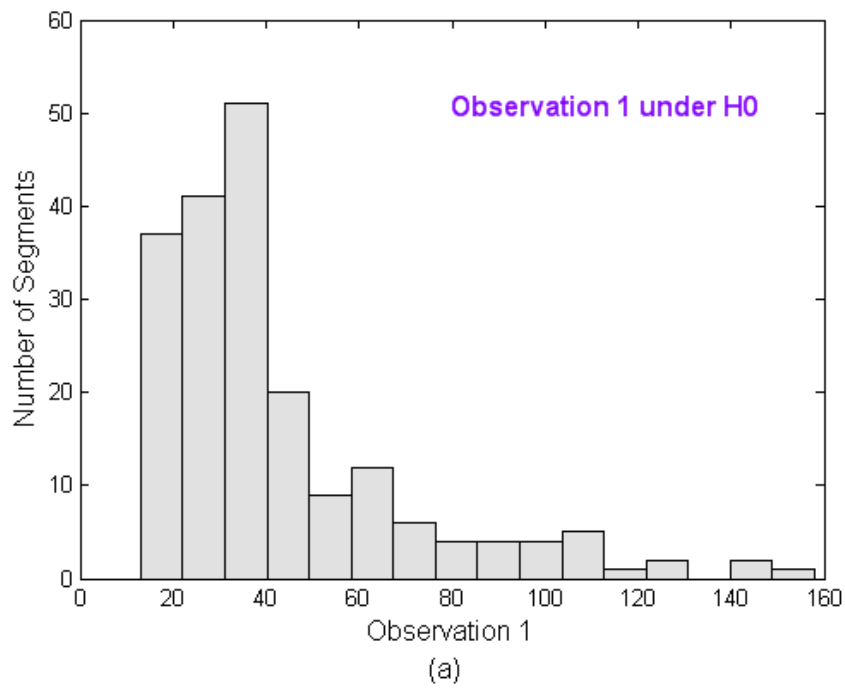


Figure 6.11. Histograms of (a) Observation 1 for no motion and (b) Observation 2 for no saturation.

Communication Protocol and MATLAB Interface

Wireless communication (between the pulse oximeter and a receiver board on a PC) and serial communication (between the receiver board and the MATLAB interface) utilize a frame structure as illustrated in Figure 6.12 and Figure 6.13. The data frame (Figure 6.12) length is 18 bytes, with the first 8 bytes assigned to a unique MAC address, the next 8 bytes assigned to the four channels of signal data, and the last two bytes appended to ensure frame integrity. At the end of a 3-second segment (720 data frames), an extra tag frame (Figure 6.13) is transmitted or received that contains extracted features and decision-making results. A tag frame begins with a pre-specified universal virtual MAC address to be identified by the MATLAB interface. Its 8-byte payload includes Feature 1 (2 bytes), Feature 2 (1 byte), Feature 3 (1 byte), Feature 4 (1 byte), three binary hierarchical decision results (1 byte), heart rate (1 byte), and SpO₂ (1 byte).

Figure 6.14 illustrates the MATLAB visualization interface. Its purpose is to visualize the algorithm implementation process and manually verify decision results. Note that no signal processing algorithms are implemented on this interface; it merely displays signals, signal features, decision results, and HR and SpO₂ values transmitted by the pulse oximeter. The interface also incorporates a control panel to assign communication parameters and control raw data storage (Figure 6.14, upper left). The two small plots (Figure 6.14, lower left) display four features that update at a 3-second interval. The plot on the upper right depicts the original PPG, and the plot on the lower right depicts the “compact representation” for the latest 3-second segment. An information bar between the upper and lower plots displays the decision result: “Severe Motion Detected,” “Valid Pulse,” “Signal Saturated,” etc.

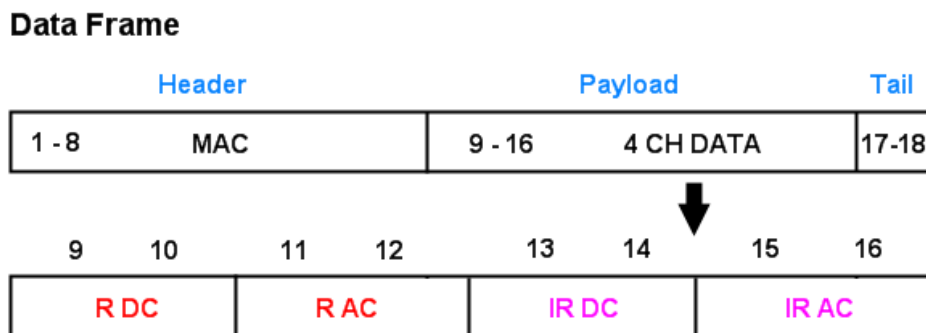


Figure 6.12. Data frame structure used in wireless and serial communication.

Tag Frame

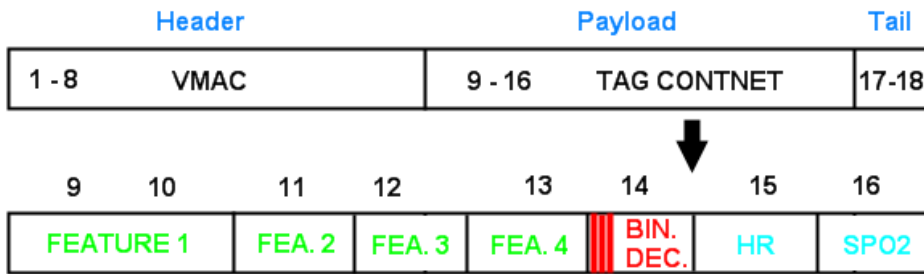


Figure 6.13. Tag frame structure used in wireless and serial communication.

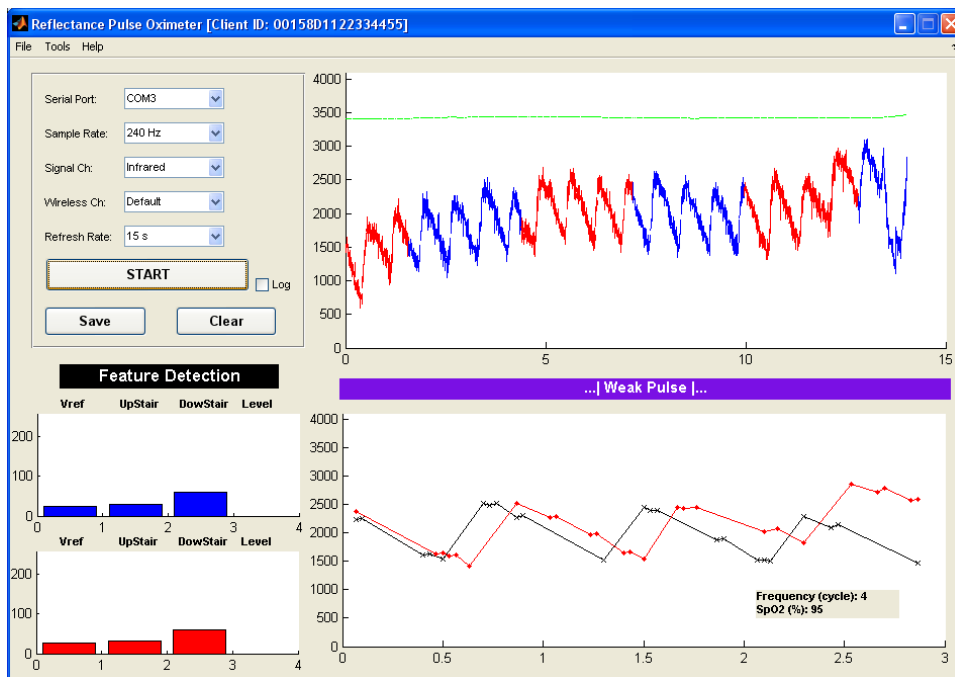


Figure 6.14. MATLAB interface developed to visualize the feature detection algorithm results. All algorithm-related parameters and decisions are ascertained on the wireless unit prior to transmission.

Downsampling and Compact Representation

Figure 6.15 illustrates an example of the downsampling and “compact representation” process. The downsampled signal in Figure 6.15 (b) presents the basic shape/trends of the raw signal in Figure 6.15 (a). The “compact representation” in Figure 6.15 (c) obtained from the downsampled data also retains the basic period and amplitude characteristics of the raw signal

required for HR and SpO₂ calculation, respectively. These three waveforms are time aligned, presenting the same number of full waveform cycles and amplitudes; hence, the data reduction process is an effective way to lessen the algorithm’s computational burden. Additionally, the “compact representation” also attempts to keep the dicrotic notch on the down slope of each cycle, which offers the opportunity to further extract physiological parameters such as reflectance index [58] and stiffness index [47], [56].

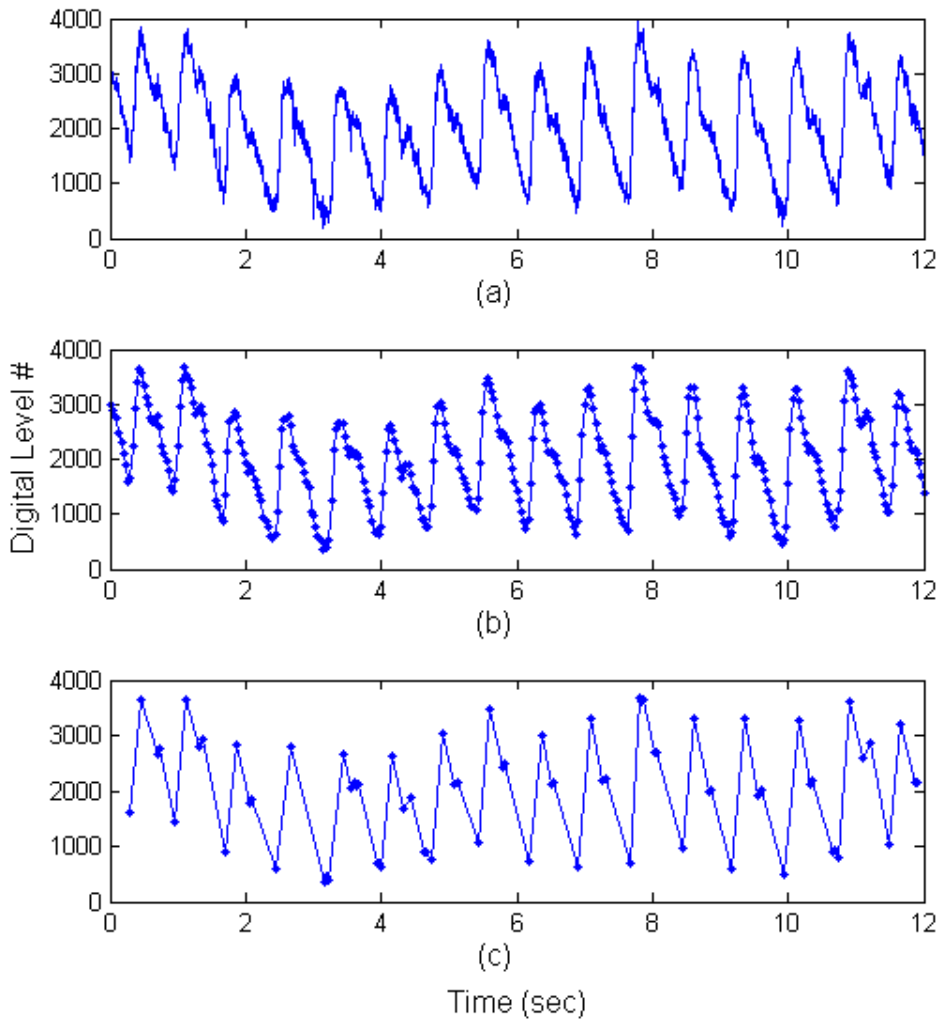


Figure 6.15. Time-domain depiction of (a) an original PPG, (b) its downsampled representation, and (c) its “compact representation.”

Feature-Extraction and Decision-Making Performance

An excerpt collected from one of the 27 additional subjects is depicted in Figure 6.16. The 99-second set consists of 33 3-second segments, notated as segment 1 to 33. The feature-extraction algorithm yields four features for each of the 33 segments, as displayed in the bar charts in Figure 6.17. The three hierarchical decision rules and their corresponding detection and false-alarm probabilities are listed in Table 6.3.

Based on the features in the bar charts in Figure 6.17 and the thresholds in Table 6.3, segments 1, 16-20, 24-26, and 30-33 in Figure 6.16 are tagged as “No Motion” (marked with the symbol ■). Of these segments, segments 1, 17-19, 24-25, and 30-33 are tagged as “No Saturation” (marked with the symbol ●) and are pushed to Step 3 in the decision hierarchy. After Step 3, segments 17 and 30-32 are eventually tagged as “Valid PPG” (marked with the symbol v).

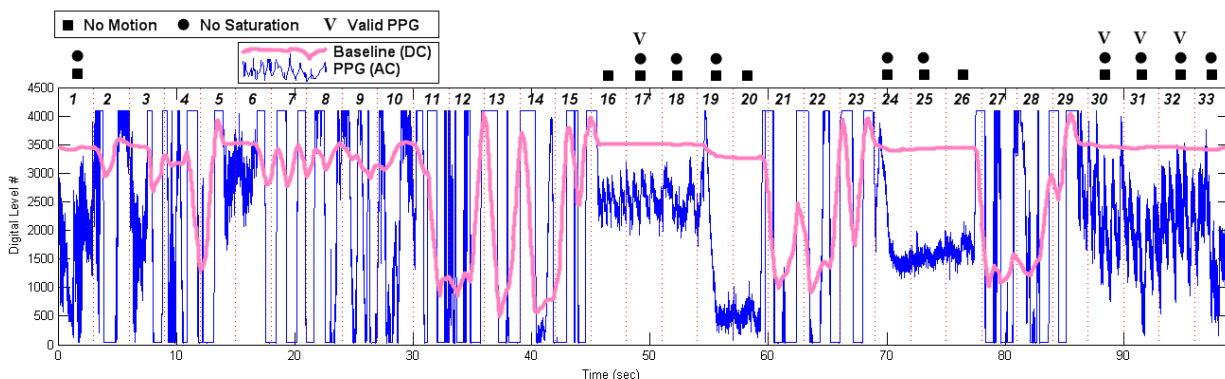


Figure 6.16. A typical data set consisting of 33 three-second segments.

These results coincide perfectly with the classifications determined by manual waveform observation. In a statistical validation experiment with 891 randomly selected segments, a decision success level of 99% was achieved, where the remaining 1% of incorrect decisions resulted primarily from the presence of a state transition segment such as segment 19 or 24 in Figure 6.16. An additional feature to indicate the consistency between consecutive segments may mitigate this type of error.

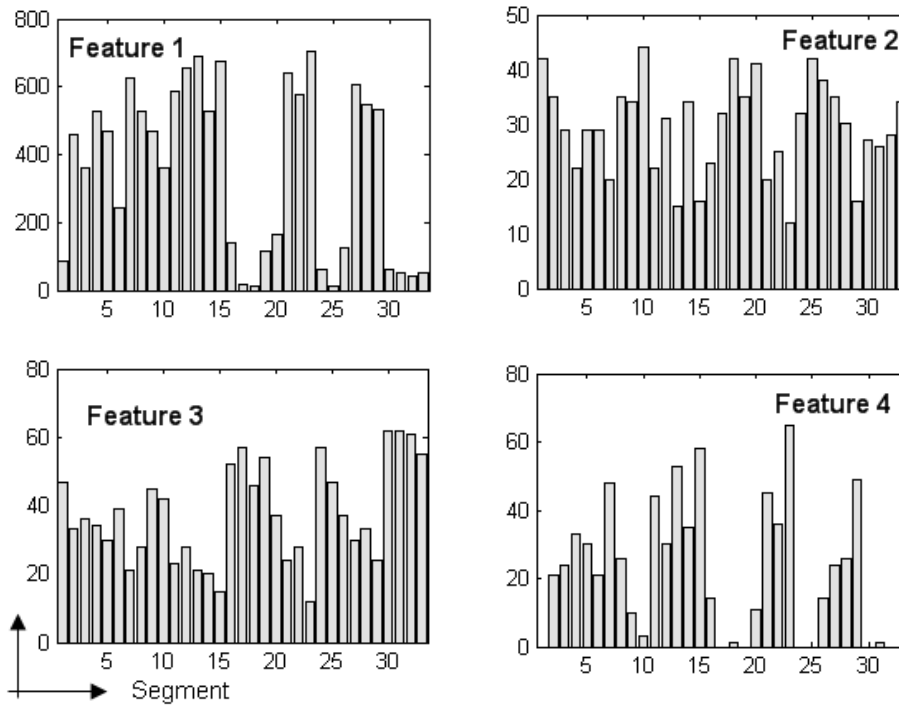


Figure 6.17. Four features extracted from the 33 segments.

Table 6.3. Hierarchical Decision Rule Results

Step	Threshold	P_D	P_F
1	180	> 99.9%	< 0.1%
2	10	> 99.9%	< 0.1%
3	1.78	81.0%	21.5%

Note: A decision fusion [82] result is unavailable since each hierarchy is based on different hypotheses.

Chapter 7 - Sensor Integration and Interaction

The remaining chapters will focus on multiple biomedical sensors, medical devices, and other medical components in small or large scale health care systems. This chapter addresses a low level sensor integration and interaction framework based on the GumPack platform that includes SCs and Chiclets.

Surface Substrate and Interconnection Interface

The GumPack model is fundamentally composed of four surface substrates (SSs) and four surface components (SCs). Each SS provides connection support for a SC that is snapped on. Besides two 70-pin connectors between each paired SS and SC, the connection support includes the interconnections between the four SS-SC pairs.

Figure 7.1 shows the surface substrate prototype. The board size is 84.4 mm by 32.8 mm. A “Surface Component Place Here” region is marked on the SS in between two mounted 70-pin connectors (the receptacle). The interconnection interface region is also denoted in Figure 7.1. It consists of two rows of vias. The first row receives connections from the previous SS, and the second row offers connections to the next SS.

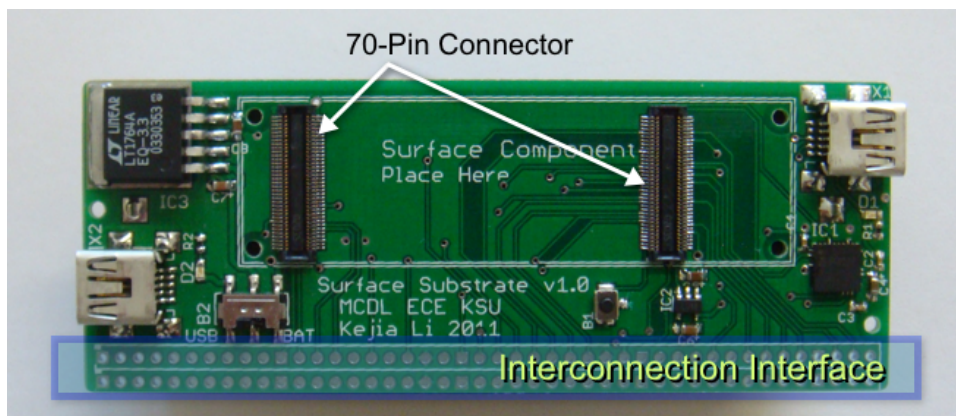


Figure 7.1. Surface substrate circuit board (fully populated version).

A standard SS board with only the 70-pin connectors populated is intended for use with a regular SC such as a biomedical sensor board. For a SC with a core processor, e.g., a Gumstix Overo board, a fully populated version is required as in Figure 7.1. These extra parts include (a)

a 3.3 V voltage regulator, (b) a 1.8 V voltage regulator, (c) a USB bridge, (d) mini-USB connectors, and (e) switches and LEDs. This SS provides power selection (USB or battery) and supply, USB connection to other devices (e.g., a personal computer), and a GumPack reset. Typically, the interconnection interface chain starts with a fully populated SS. The interconnection interface is a subset of the 140 pins (2 x 70 pins) plus several power supply lines. Each row on the interface has 44 vias, designated as pin 1 to pin 44 (left to right in Figure 7.1). The pins apply to these functions:

- Power supply (3.3 V, 1.8 V, GND)
- GPIO (General Purpose Input/Output)
- 1-Wire bus
- I²C bus (address range 0x03 – 0x77)
- SPI bus (two chip selection)
- UART (Universal Asynchronous Receiver/Transmitter)
- PWM (Pulse Width Modulation)
- ADC (10-bit analog-to-digital converter).

A full pin assignment/definition is described in *Appendix A – GumPack Hardware Interconnection Interface*. Note that a pin can be multiplexed, as most buses can be configured to GPIO. For example, pin 3 (GPIO171_SPI1_CLK) can be used as either a GPIO or a clock signal in SPI bus 1.

Cuboid Assembly

The GumPack cuboid conceptual model was illustrated in Figure 2.2. The main parts (surface substrates (SSs) and surface components (SCs)) were introduced in the previous text. A solid cuboid structure is yet needed to support up to four SSs that allow four SCs to plug in. Figure 7.2 illustrates the inner cuboid design created with SolidWorks CAD software. The physical part (see Figure 7.3) was machined with the assistance of the KSU Department of Mechanical & Nuclear Engineering. Its dimensions are 84.8 mm x 28.9 mm x 28.9 mm. This inner cuboid shell provides a rigid structure for SS circuit boards and can hold a battery and two antennas via two cylindrical voids.

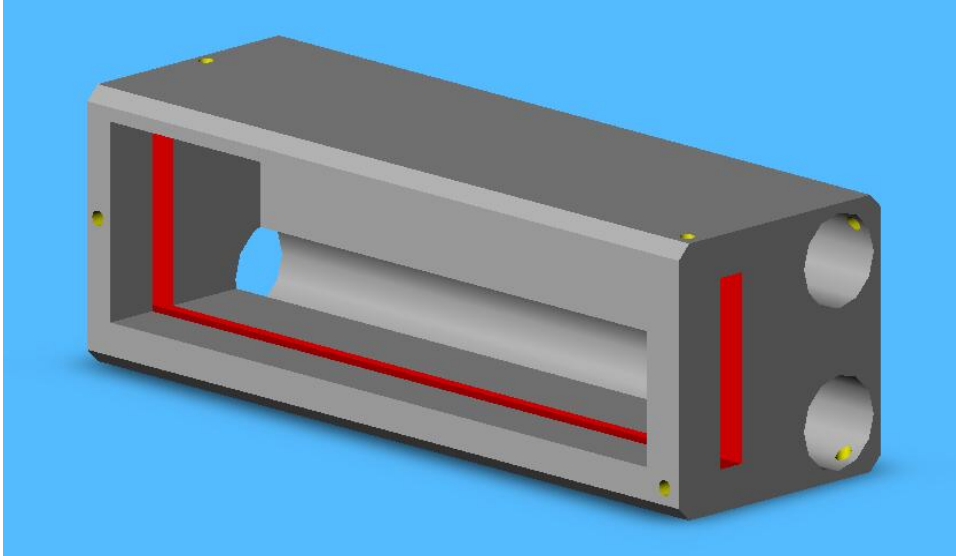


Figure 7.2. SolidWorks design of the GumPack inner cuboid.



Figure 7.3. Machined GumPack inner cuboid.

A type 18650 3.7 V Li-ion rechargeable battery (size: 64.9 mm by 18.3 mm) powers the GumPack. The UltraFire battery shown in Figure 7.3 has a capacity of 2400 mAh. UltraFire also provides a 3600 mAh battery (the 18650), which could power a GumPack with a continuously transmitting Wi-Fi connection up to 6 hours, assuming the current consumption is 600 mA on

average. When fully charged, the battery voltage is around 4.0 V, and the GumPack processor SC works normally until the battery voltage drops below 3.2 V. A fully assembled GumPack prototype will be presented in Figure 7.9 later.

Software Architecture

The following sections address the software architecture and component framework envisioned for the GumPack. Software supplements will focus on the following:

1. Field definition of the profile/memory chip on each surface component that is consistent with hardware and software level standards.
2. Finite state machine for the interaction between surface components and the GumPack core.
3. Implementation and evaluation of each state while considering efficiency (system resource optimization) and security.
4. Automation and optimization of the processes of component connection and disconnection (plug-and-play).
5. Application interface definition and integration of the GumPack app library for the automation of app installation and execution.

From a software architecture perspective., the GumPack, through the OMAP3530 processor, supports several operating systems, including Android, Linux, and Windows CE. To accelerate software development, Angstrom, a commonly used Linux distribution for a variety of embedded devices, is employed as the software platform for the GumPack. Angstrom is also well supported by the OpenEmbedded development environment. OpenEmbedded utilizes BitBake to cross compile an embedded Linux. In the case of the GumPack, a personal computer with an Ubuntu operating system and an OpenEmbedded build system is used to develop another operating system Angstrom for the GumPack.

Figure 7.4 shows the enormous software architecture that is built upon the GumPack hardware. It is by convention divided into (a) kernel space and (b) user space. In this section, a brief description is provided for each block of the software architecture. The following sections elaborate on some of the more important components.

The kernel space deals with the operating system kernel, driver, and boot loader. U-Boot is used as the boot loader for the GumPack – the first piece of software that can be programmed.

Pin multiplexing is typically done by U-Boot. For example, pin 3 (*GPIO171_SPI1_CLK*) on the SS board is defined as a clock signal in SPI bus 1 rather than a GPIO. To use pins in the user space (e.g., switch an LED on via a GPIO), U-Boot should explicitly export them to the kernel, where the kernel further exports them to the user space. A script can be created to do boot time configuration of the board, do low-level testing, or set up the U-Boot environment. Linux kernel (device driver) programming typically uses kernel modules for the GumPack, which allows a fast installation of device drivers into the kernel without rebuilding the kernel.

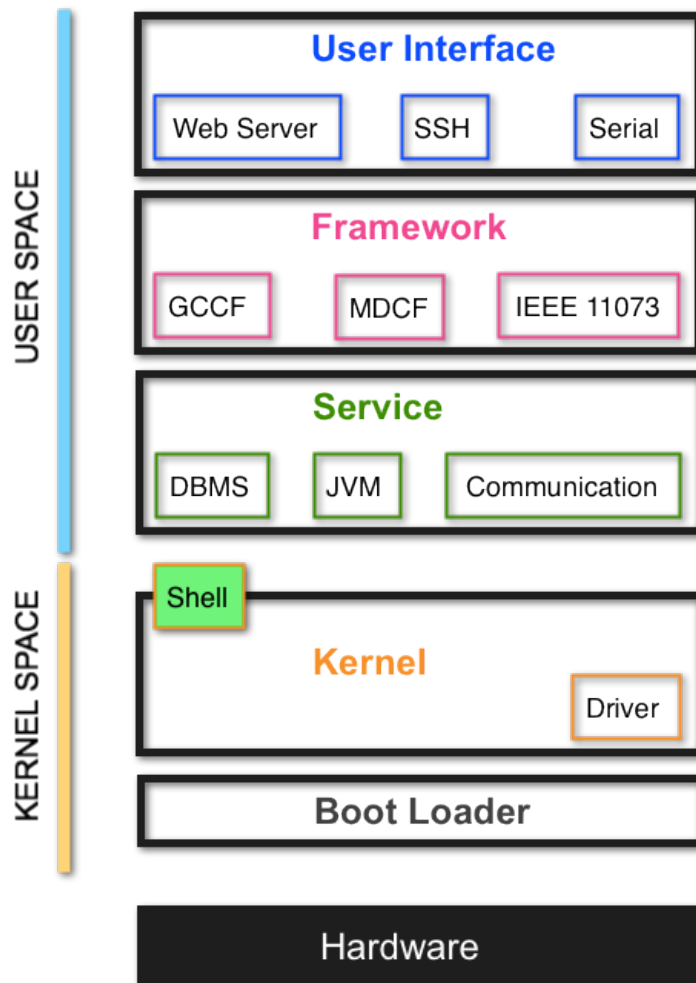


Figure 7.4. GumPack software architecture.

A shell is a command line interpreter that is frequently used in the GumPack. It links the kernel and user space for typical uses such as these:

1. System configuration (e.g., to set up the network interface in the file `/etc/network/interfaces`)
2. Package installation (e.g., to install a Java Virtual Machine (JVM) using `opkg install cacao`)
3. Implementation of all kinds of system commands and user applications (e.g., `./myApp`)

The user space has a three-layer architecture (from bottom to top as in Figure 7.4): Service, Framework, and User Interface. The Service layer provides three basic services. (a) The Database Management System (DBMS) service records and manages all incoming user data, e.g., aggregated sensor data. Data can be organized in the file system within hierarchical folders or maintained by commercial DBMS software like MySQL. (b) The JVM service provides a platform for Java applications (e.g., the MDCF client is fully implemented in a JVM). (c) The Communication service controls wireless network access (as a client) or creation (as a host). An example would be the creation of an ad-hoc network for other handheld devices to locally access a GumPack web server. The Framework layer provides interconnection and interoperability standards for surface components (e.g., GCCF and IEEE 11073) and the GumPack itself in a large medical device cooperation environment (e.g., MDPnP, MDCF, or IEEE 11073). The User Interface layer supports three means by which a user can access the GumPack. (a) A Secure Shell (SSH), a remote shell, and (b) a Serial communication link are aimed at developers who need real time access to the system. (c) A Web Server is a user-friendly channel to access the PHR maintained by the GumPack.

A particular application (APP), e.g., for a motion sensor SC, links to all three layers. Its position can be considered parallel to all three layers and on top of the kernel. A motion sensor SC is 'connected' by the GCCF framework, its data are stored by the DBMS service, and a user reads a motion waveform through the Web Server interface, which utilizes the Communication service.

GumPack Component Connection Framework

The GumPack design allows one to integrate biomedical components in an efficient and flexible way. New device functionality is added to a GumPack by plugging in or switching in the

corresponding surface component. Such a means to design medical devices may attract developers' attention, resulting in a large collection of SCs available to GumPack users. This section addresses the means to regulate these third-party components and automatically call the corresponding APPs without user intervention.

The GumPack Component Connection Framework (GCCF) is a software interconnection protocol based on the hardware interconnection interface on each surface substrate. Its primary functions are (a) SC authentication and (b) SC APP automation, both realized by the GCCF and a profile chip on a SC. The profile chip is a memory chip with an I²C interface, and the address should be in the range of 0x03 – 0x77 and different from other I²C addresses that have already been registered in the system. Chips with I²C interfaces usually have configurable I²C addresses. It is recommended that SC designers leave such options to users (e.g., the LSB (least significant bit) of the address can be selected by a switch).

The profile chip contents include its identification information (for an SC authentication function) and interface and data format information (for an SC APP automation function). The memory size is at least 1 kbit. Figure 7.5 shows a 128-bit x 8-row allocation structure for a profile memory space. The first row is the 128-bit ID or Key field. This unique sequence is only authorized for an SC that has already been verified by certain standards and regulations. The authentication algorithm of the GCCF checks the SC ID and decides if further communication will proceed. The second 128 bits are evenly divided into four fields of 32 bits each that contain other SC ID related information: type, address (up to four different I²C addresses), option, and checksum for the entire ID section. The interface and data format sections span from the third row to the end. The Pin Request field designates which pins are used. E.g., the 1st bit in this field corresponds to pin 1 (GND) on the hardware interconnection interface. It is set to 1 if pin 1 is utilized on the SC board. The Interface 1 field occupies the first 64 bits of the fourth row. It addresses the interface type (e.g., GPIO, ADC, or SPI), direction (input, output, or bidirectional), data rate (e.g., 100 Hz), data width (8 bits), etc. Given this information, the GCCF calls the corresponding APP from the APP library or online APP store.

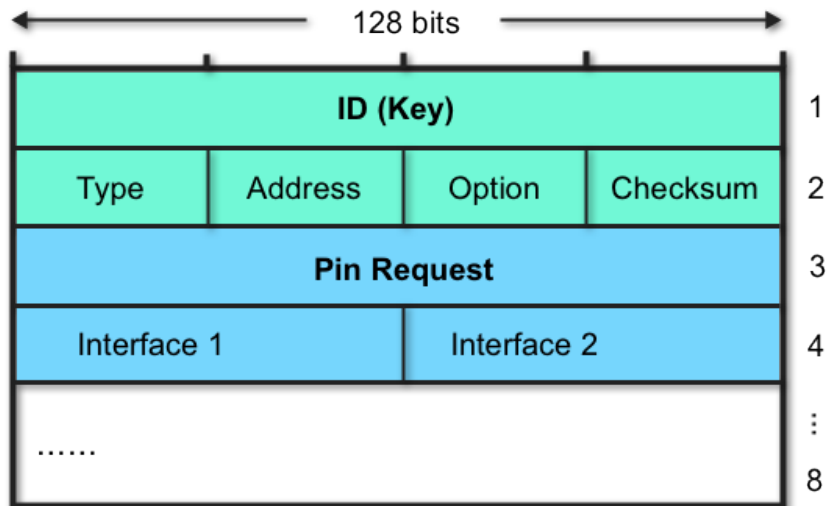


Figure 7.5. Profile memory chip field definitions.

Figure 7.6 illustrates the GCCF finite state machine (FSM). The starting state is Bus Scan. The GCCF periodically checks the I²C interface, which is designated for profile chip communication, from address 0x03 to 0x77. If there is no response for a particular address, it moves to the next address. A successful acknowledgement (ACK) triggers the GCCF to read the SC ID (the first 16 bytes) via the bus. An authentication process is implemented based on the 128-bit ID. If a negative result occurs, the FSM goes back to the Bus Scan state. If a positive result occurs, the GCCF reads interface information by accessing the same I²C address. Retrieved data are from the 33rd byte up to the last interface field (see Figure 7.5). The interface data are interpreted to check required resources (e.g., a bus driver), install the necessary APP from the APP library, and eventually run the APP. If any error is encountered during the state Read Interface up to the state App Serving, the GCCF returns the FSM to the Idle state. Potential errors include (a) a resource request fail, e.g., a required GPIO is used by another APP, (b) an interpretation error, e.g., a data format is undefined, (c) a profile chip communication error, and (d) a data communication error.

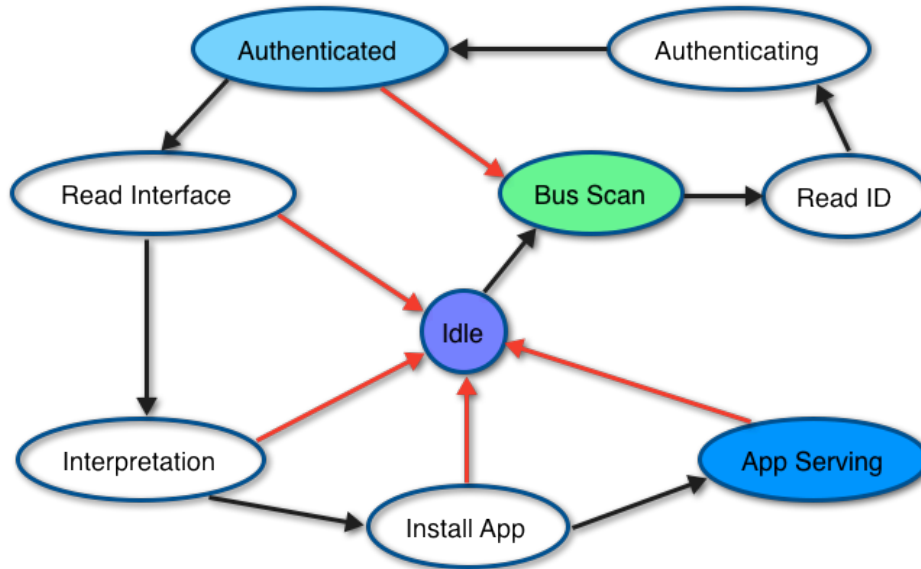


Figure 7.6. GCCF finite state machine (FSM).

If the SC is designed using the GSC model, the GCCF could also automatically generate APP code (via an APP template) for developers with the information that is to be written into the profile chip. This approach can minimize, e.g., APP development time and data interpretation errors.

Chiclet Network and Device Interoperability Standards

Chiclets are an important GumPack supplement for applications that involve wearable devices/sensors. They are off-board ‘surface components’ that wirelessly connect to the GumPack rather than rely on the onboard 70-pin connectors. The communication protocol could be Bluetooth, ZigBee, or other short distance but low-power solutions such as Ant. ZigBee is preferred for the time being, since the first Chiclet design will be the wireless reflectance pulse oximeter. Additionally, multiple network topologies are well supported in ZigBee. ZigBee functionality is enabled by attaching the ZigBee Coordinator SC to a GumPack.

A Chiclet network is formed between one or more Chiclets and a GumPack. It is like a private local network rather than a public network in the context of interoperability between conventional medical devices. This relationship is illustrated in Figure 7.7. The lower left area depicts a Chiclet network based on a ZigBee infrastructure. A tree topology is used to expand the diameter of the network coverage as a flexible topology example. Using a tree topology, the end

nodes are usually the sensors, and the middle nodes (routers) are merely data transporters between end nodes and the GumPack (ZigBee Coordinator SC). For a typical body-area-network use case, a star topology would be preferable since the communication range is within 10 m. A star topology also alleviates time synchronization issues for multiple Chiclets.

In the broader context of a medical device network (see Figure 7.7), a GumPack can be seen as a single terminal on an enterprise service bus (ESB), though its role is not fixed (e.g., it may be a vital signs monitor or a patient communicator), which makes the GumPack a competitive candidate for KSU’s MDCF (Medical Device Coordination Framework) project. MDCF clients were previously ‘mock’ devices: JVMs run on a computer or real devices with JVM adapters, connecting to the MDCF bus maintained by the MDCF server (bus manager). The GumPack is the first native MDCF-ready medical device that can successfully run the whole MDCF client on itself with JVM service support (see Figure 7.4).

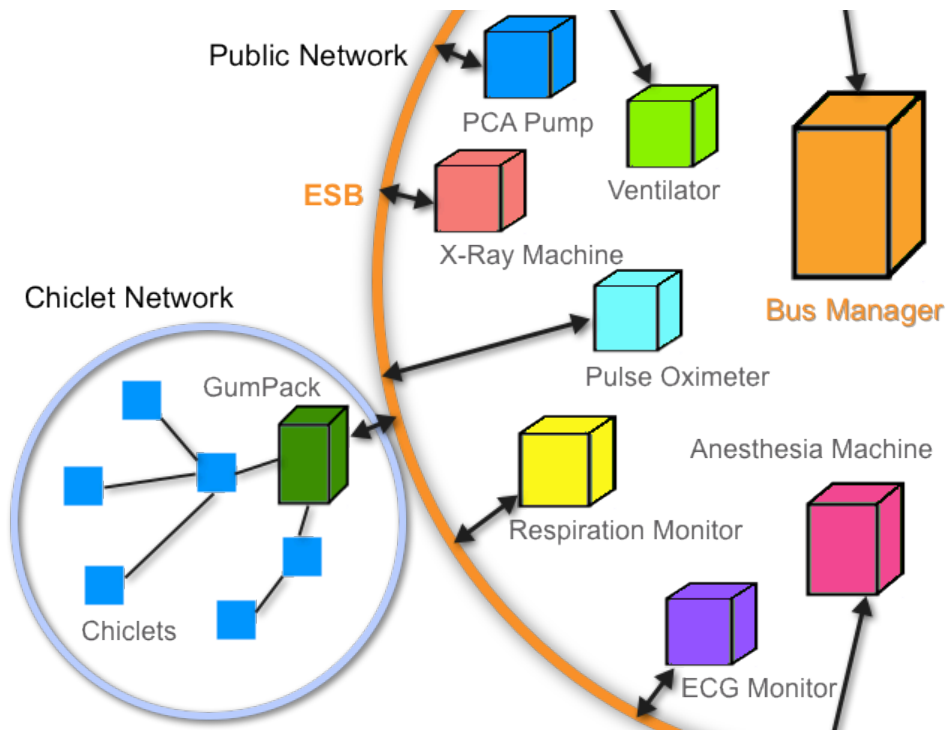


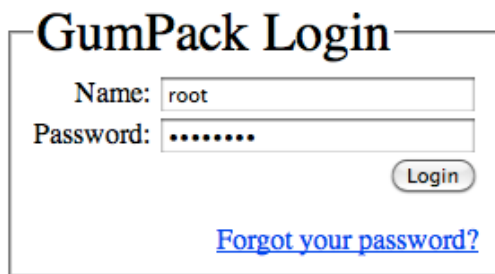
Figure 7.7. GumPack Chiclet network and medical device interoperability.

This MDCF scenario is an example of integrating the GumPack into a cooperative medical device network. Other interoperability frameworks such as MDPnP will be evaluated in

the future. ISO/IEEE 11073 standards are also to be assessed for Chiclet network and GCCF operation.

Web Server and User Interface

At present, the GumPack does not provide a cell-phone-like user interface. However, a user is able to utilize a phone, a tablet, Internet TV, or any device that supports Wi-Fi and web browsing as an indirect access interface. This capability is realized by the web server maintained on the GumPack.



The image shows a web browser window with the title "GumPack Login". Inside the window, there are two input fields: "Name:" containing the text "root" and "Password:" containing masked characters ".....". To the right of the password field is a "Login" button. Below the input fields is a blue hyperlink that reads "Forgot your password?".

Figure 7.8. GumPack web server login page.

A lightweight web server `lighttpd` is employed on the GumPack instead of the more popular Apache. `Lighttpd` is a secure, fast, compliant, and flexible web server that has a small memory footprint compared to other web servers and takes care of CPU load. CPU usage of `lighttpd` is less than 1.0% for a single client. It is configured to run PHP modules. PHP provides an ODBC (Open Database Connectivity) interface to database and file operation functions (e.g., to read and write a file on the GumPack file system). Hence, PHP is used to develop the GumPack web site and web pages. Figure 7.8 shows the login page that requires an account name and a password when a user accesses the GumPack server. The GumPack server supports multiple user accounts, which can be configured after a user is logged in. Four categories of information are accessible to logged-in users:

- Dashboard, e.g., for signal waveform display
- Trackers, e.g., for user input to a PHR
- Devices, e.g., for SC configuration
- File Repository, e.g., to check out files.

Demonstrative web pages are shown in Appendix C. They were captured by an iPad that was connected to the same local wireless network as the GumPack.

Prototype and Initial Results

Figure 7.9 shows the first version of the assembled GumPack prototype surrounded by three Chiclets. The size of this prototype is 84.4 mm x 32.8 mm x 32.8 mm, constrained by the size of each surface substrate board. The four mounted surface components are the aforementioned processor SC, a motion sensor SC, an ECG SC, and a ZigBee coordinator SC that coordinates the three Chiclets (reflectance pulse oximeters). A Wi-Fi antenna is placed inside of the cuboid structure and connected to the processor SC board, which needs to be redesigned to a better form factor that can be embedded inside the GumPack cuboid. The interconnection wires are not fully populated at the current debugging stage; they are envisioned to wrap around the four surfaces and go through the space between each SS-SC pair. A flexible printed circuit board (cable) will likely replace the wrapping wires, or a new inner cuboid design will provide interconnection sockets at each corner.

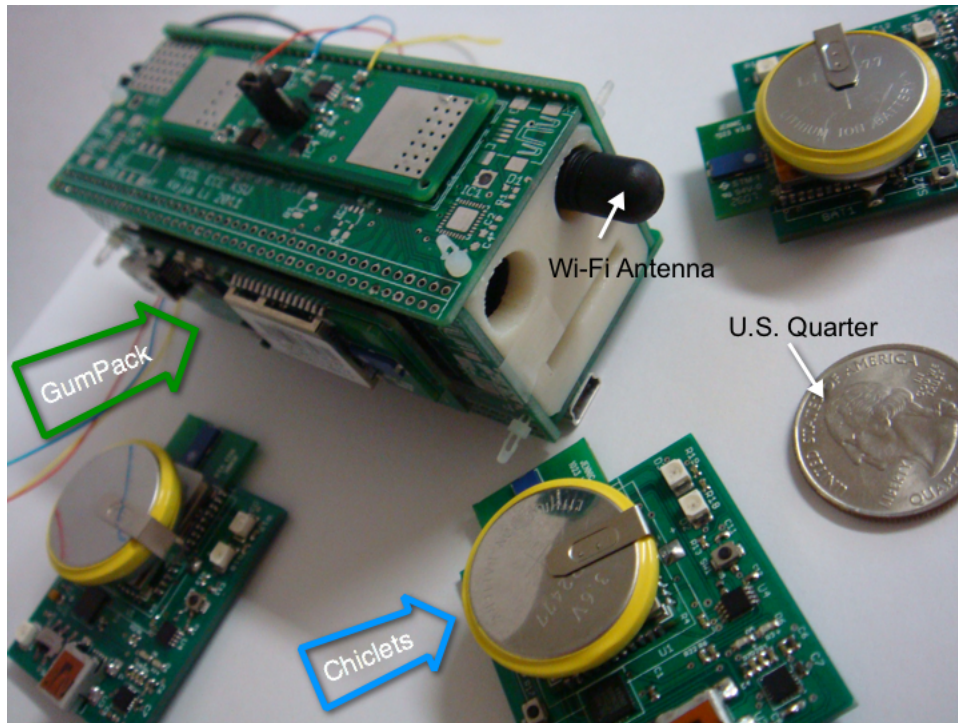


Figure 7.9. GumPack prototype and Chiclets.

As implied by the GumPack conceptual model in Figure 2.2, camera and audio support will be integrated into the GumPack design in the future. For example, Figure 7.10 shows a Caspa camera that was originally an open source design from Pixhawk and is now supported by Gumstix's Overo board. A Bluetooth headphone, keyboard, GPS, etc. are all on the waitlist. Consistent with the "GumPack" theme, more customizable biomedical surface components are also needed.

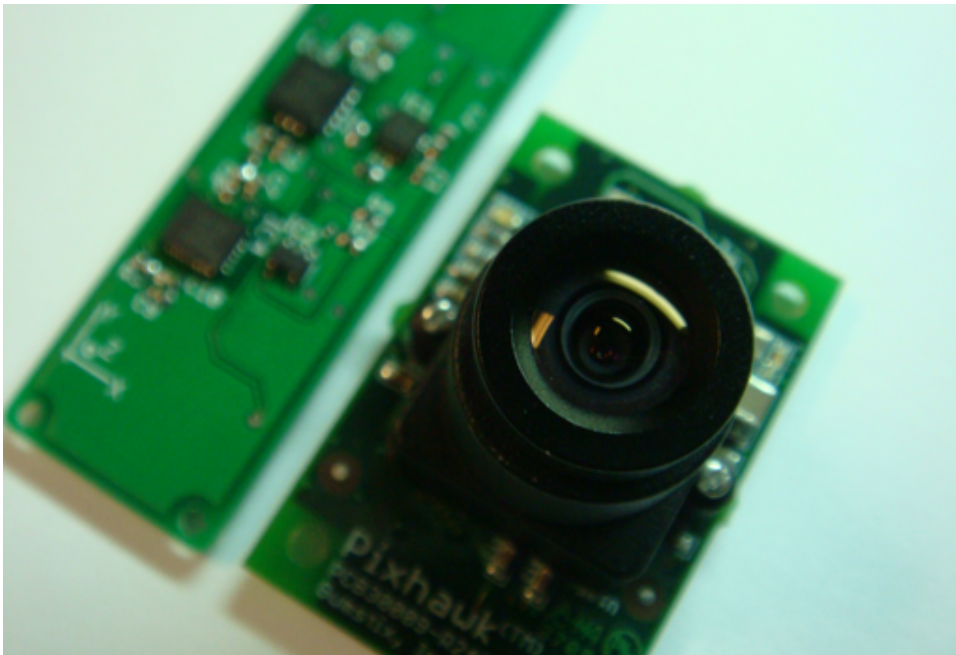


Figure 7.10. Caspa camera board alongside a motion sensor SC.

Although demonstrating the acquisition of a sensor signal is not the centerpiece of the GumPack idea (for purposes of this chapter), the concept of obtaining an 'unfiltered signal' is worth briefly noting. The Chapter 4 figures have already depicted the unfiltered PPGs that can be obtained by a reflectance pulse oximeter Chiclet. Figure 7.11 displays an unfiltered single-lead ECG obtained with careful baseline and amplification control. Such signals are usually accompanied by different levels of 60 Hz grid noise (in the U.S.) and appear to be low quality. However, they represent raw data with all signal details (e.g., frequency components) unaltered and exposed. These data are therefore extremely useful for education and signal processing comparison studies.

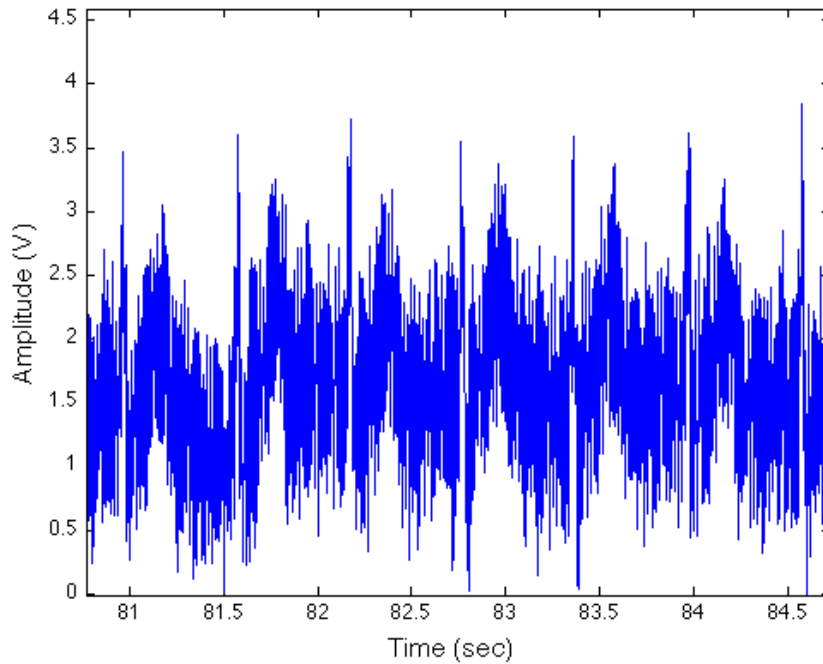


Figure 7.11. Unfiltered ECG sampled from an ECG SC.

Figure 7.12 presents the raw ECG's magnitude spectrum from 0 – 15 Hz. The fundamental frequency, 1.63 Hz, is the heart rate (97.8 bpm), and its nine harmonics are distinguishable in the spectrum. A 0.2 Hz frequency component is also recognized, which is likely the respiration rate. The 60 Hz noise component has a large magnitude, though it is not shown here.

Signal quality (e.g., signal-to-noise ratio) is immediately improved by digital post-processing, which can be efficiently realized in the GumPack due to its processing capability, which will be further improved by exploiting the digital signal processor core. A filter toggle button/option is provided on the web user interface, giving a user the view of both the 'clean' waveform and the raw waveform. The goal is to allow a user to click the waveform to see, e.g., extracted parameters, frequency spectra, and other analysis results.

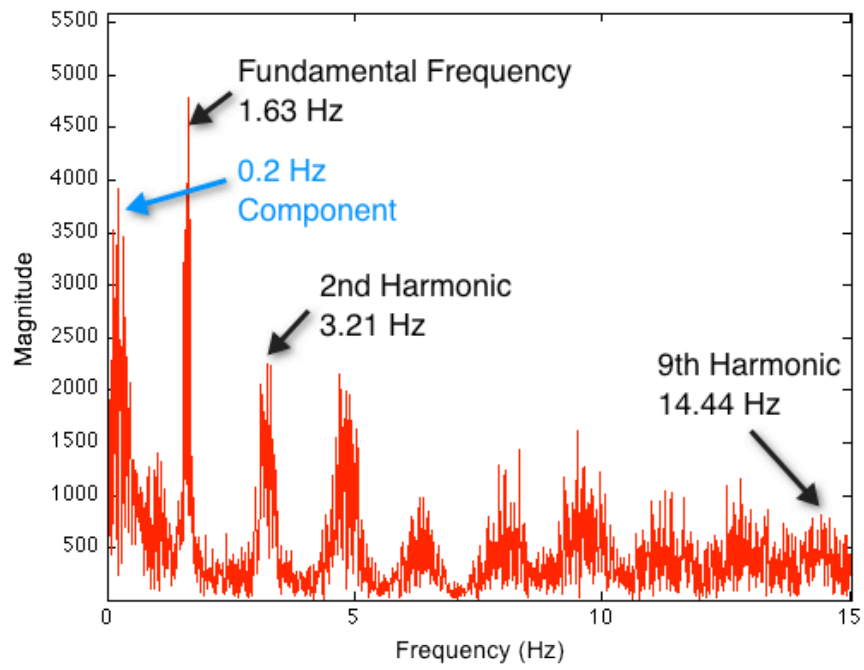


Figure 7.12. Frequency spectrum of the unfiltered ECG.

Chapter 8 - High Resolution WBAN

Diverse and creative application-layer schemes can be designed and evaluated for various WBAN scenarios. This chapter addresses a WBAN that emphasizes high-resolution raw data, real-time operation, and time synchronization of intra-sensor data and inter-sensor waveforms. The WBAN requires a high-speed wireless connection that is always alive over the desired data acquisition period, as opposed to most WBAN applications, which allow periodic sleep modes. This type of WBAN transfers large amounts of raw data to a base station, where heavy signal processing is implemented.

IEEE 802.15.4 MAC Layer

The carrier sense multiple access with collision avoidance (CSMA/CA) method is adopted in the IEEE 802.15.4 MAC layer. The CSMA approach requires transmitting nodes to first listen to a channel and then transmit only if the channel is idle; the collision avoidance mechanism improves CSMA performance by dividing the communication channel equally among all nodes in the collision domain. Utilizing this method in a WBAN with only one channel ensures that each sensor node has an equal opportunity to upload data, and each successfully transmitted data segment can be synchronized with the lost data segments from the other nodes. Since ZigBee (the wireless standard utilized by the custom pulse oximeters) is based on IEEE 802.15.4, this approach is employed here.

Issue 1: Lost and Wandering Frames

Figure 8.1 illustrates a star-topology ZigBee network with two sensor nodes (S1 and S2) and one receiver (Coordinator). Sensor-node frames are received by the Coordinator with equal probability. Each sensor has a unique MAC address, which can be identified by the Coordinator, so the frames sent by S1 and S2 are illustrated differently. Each frame is originally assigned a sequence # to help address the issue of lost and wandering frames (see Figure 8.1). Lost frames are unrecoverable because each sensor node only attempts to transmit a frame once to ensure real-time operation. Wandering frames can be rearranged by a computer, which receives frames from the Coordinator via a serial port.

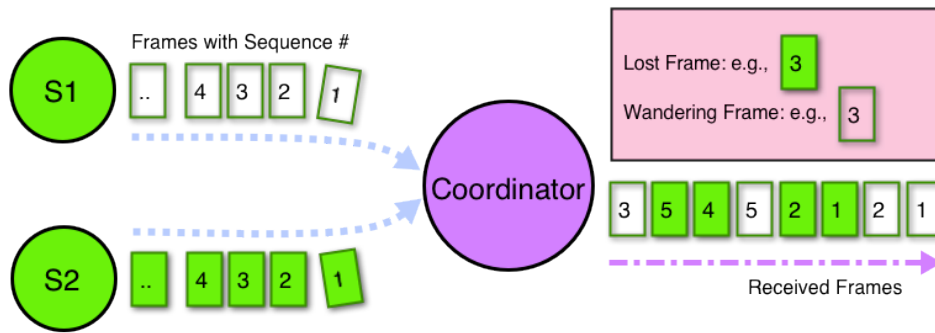


Figure 8.1. An example star-topology network. Each frame sent by a sensor node carries a sequence # to address the issue of lost and wandering frames.

Issue 2: Timeline Distortion

The second issue is caused by the asynchronization of the frame sending and receiving rates. For example, if S1 and S2 each competitively transmit 240 frames per second, the Coordinator does not necessarily receive a sum of 240 frames in one second. Experiments indicate that the number of received frames depends on the frame length.

To address this issue, the inner timers in the sensor nodes and the Coordinator are identical. Such timers trigger events like transmitting a frame or generating a ticket, as shown in Figure 8.2. A Ticket Counter inside the Coordinator increases by 1 when the Ticket Generator produces a ticket and decreases by 1 when a received frame consumes a ticket. This Ticket Counter number is appended to each frame to assist the follow-on timeline restoration process. Continuing the thought, if the Ticket Counter value decreases by 10 in one second (the tickets are over-consumed), the Coordinator has received an extra 10 frames in one second, which means the timeline will be expanded given the assumption that 240 frames corresponds to one second. A correction factor of 0.96, calculated from $240/(240+10)$, should then be applied to the received-frame timeline.

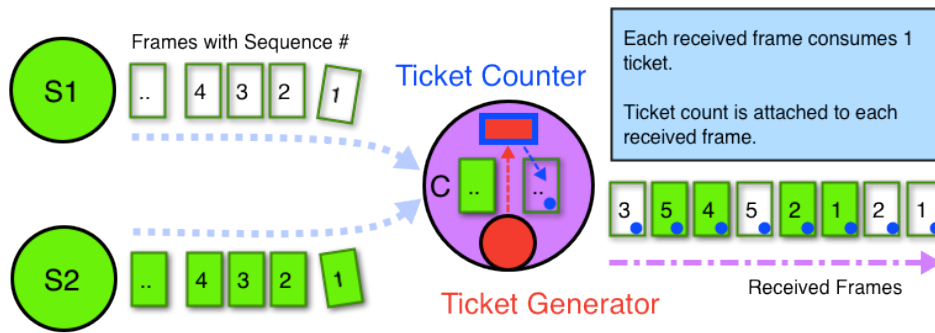


Figure 8.2. Ticket production and consumption mechanism inside the Coordinator to address the timeline distortion issue.

Frame Field Definition

Frames are assembled on the application layer of the wireless communication protocol. Figure 8.3 contains the frame field definitions for the sensor nodes (lower) and the Coordinator (upper). Each sensor frame carries data for two samples: their AC values and the corresponding DC value for the pair. The frame begins with the acknowledgement value returned from the previous transmission function and ends with a checksum. When the Coordinator receives a sensor frame, it adds an ID (mapped to the unique MAC address), a Ticket Count, and a new checksum.

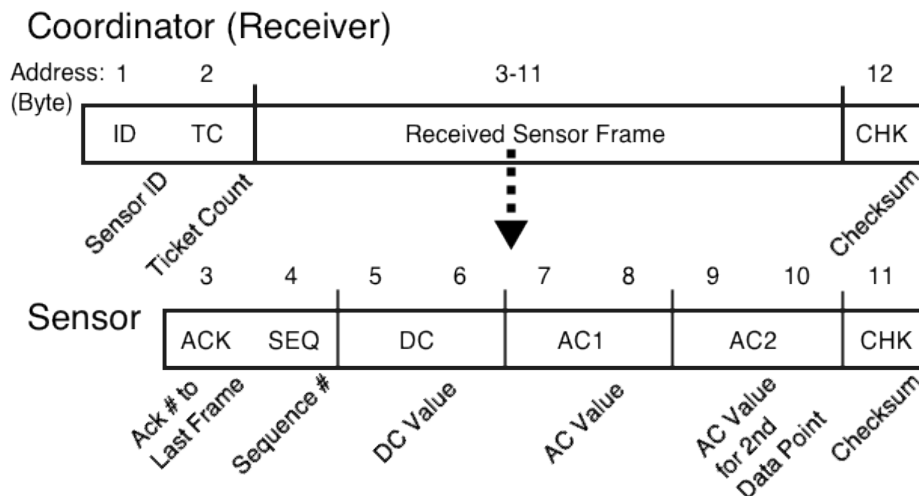


Figure 8.3. Frame field definitions for the Coordinator and sensor nodes on the application layer of the custom communication protocol.

Timeline Restoration in MATLAB

The frames finally stream to a MATLAB interface on a computer, where all the “heavy” signal processing tasks are implemented. Task 1 is to arrange the frames from each sensor node in ascending order and fill the gaps with ‘placeholder’ frames – the positions that correspond to lost frames. Since a sequence # is defined to be one byte long (see Figure 8.3), the frame search window is assumed to be within $[-127, 128]$.

Figure 8.4 shows the frame sequence #s in the received order (blue crosses) and the time-aligned sequence #s (red dots) for one sensor. The original sequence illustrates that frame wandering occurs frequently. The number of placeholder frames (assigned with sequence # = -1) between data segments is the length of data segments from other sensor nodes in the original sequence, and it will be equal to the actual number of lost frames in the recovered sequence if the frame search method did not miss any wandering frame.

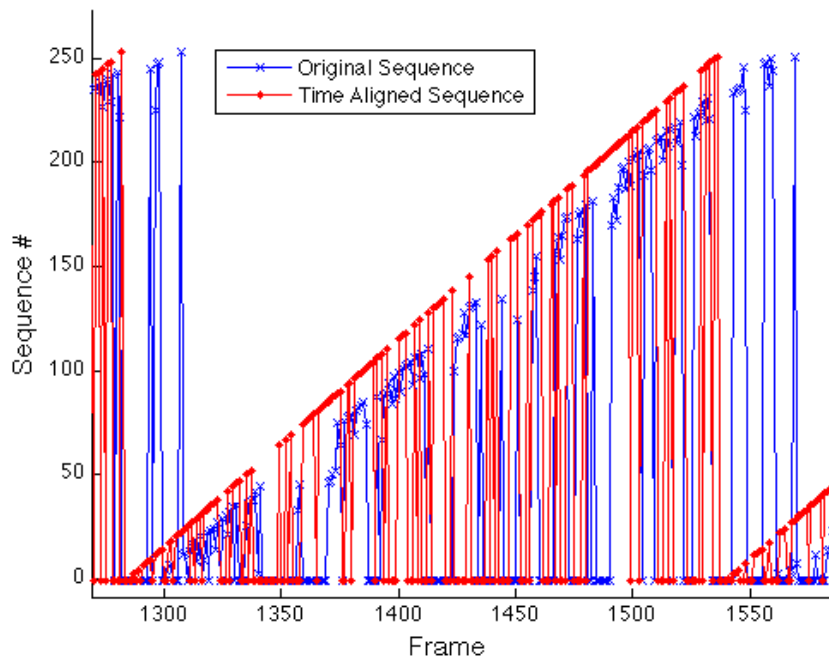


Figure 8.4. Original data stream with sequence #s in the received order (blue crosses) and the recovered time-aligned sequence (red dots). Placeholder frames are assigned negative sequence #s.

In Figure 8.4, the recovered sequence has obviously deviated from the trend of the original sequence at around the 1300th frame (placeholder frames are counted). This phenomenon relates to timeline distortion. In other words, in this example, the timeline is expanded, or tickets are over-consumed. One way to recover the timeline is to use the correction factor calculated from the Ticket Count. Another method is to perform a linear regression on the delay relative to the expected sequence #, as illustrated in Figure 8.5.

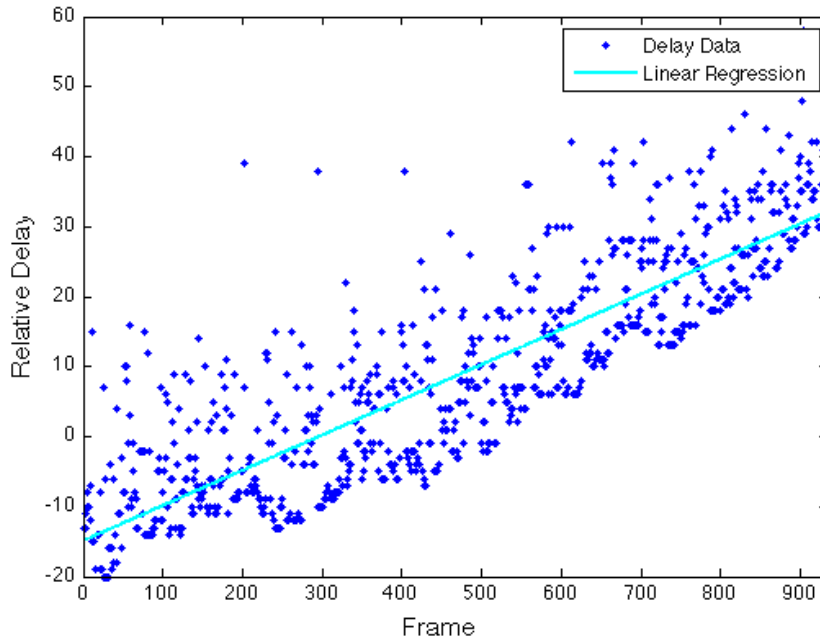


Figure 8.5. Linear regression – relative delay to recover the distorted timeline.

A reference frame has the expected sequence #; it could be a placeholder frame. The expected sequence # increases by 1 for each time step, returning to 0 after reaching the maximum. This approach estimates the relative delay of every frame, especially the reference frame, which is used to adjust the synchronization of the frame streams from different sensor nodes in a statistical manner.

Chapter 9 - Application B: Pulse Wave Velocity Estimation

Most pulse oximeters estimate arterial oxygen saturation using baseline-normalized photoplethysmograms (PPGs) acquired with several wavelengths of excitation light [83]. The periodicity in these pulsatile waveforms is intuitively related to heart rate since arterial blood volume and flow (and therefore optical absorption due to hemoglobin) track the cardiac cycle. Related research notes that physiologic parameters such as blood pressure, respiration rate, and stroke volume can also be derived from PPGs [49], [51].

Pulse wave velocity (PWV) has attracted attention as an arterial elasticity indicator. PWV methods that utilize PPGs include (a) pulse transit time (PTT) extracted from time-correlated ECGs and PPGs [84], (b) transit time acquired from dual-channel (finger and toe) PPGs [85], and (c) peak-to-peak time (PPT) from a single digital volume pulse (DVP) waveform [47] (also tested in our early efforts [12], [86]). Other PWV estimation modalities include ultrasonic Doppler [87].

A direct way to calculate average wave velocity is the distance between two locations divided by the wave transit time. Obstacles for using this method to calculate PWV with PPGs include:

1. At least one of the two locations on the same arterial segment makes PPG acquisition difficult with a transmittance-mode sensor (e.g., one sensor on the fingertip and the other at the wrist).
2. PWV is much faster than blood flow, requiring a pulse oximeter to sample at a high rate to ascertain the transit time for a PPG waveform acquired at two locations separated by a short distance.
3. The two pulse oximeters or PPG acquisition devices must be synchronized.
4. PPG signal must be high quality (e.g., thousands of digitization levels, high signal-to-noise ratio, and undistorted), especially when considering that PWV varies as vessel walls dilate and constrict during each heart cycle.

This chapter demonstrates an initial study on PWV estimation using a pair of low-cost, custom wireless reflectance pulse oximeters at two measurement locations on the same hand. While larger scale tests are pending and the results are not verified against commercial medical

equipment, this short investigation offers insight into the design issues related to an economical single-hand approach, including changes to downstream versus upstream signal shape caused by intra-cycle changes in arterial pressure.

Study Description and Experimental Setup

PWV can be expressed as a function of arterial elasticity via the Moens-Korteweg equation [55]:

$$PWV = \sqrt{\frac{hE_{inc}}{2r\rho}} \quad (13)$$

where h is the arterial wall thickness, E_{inc} is the incremental elastic modulus, r is the lumen radius, and ρ is the blood density. When PWV is measured with one of the aforesaid methods, arterial parameters such as arterial elasticity could be estimated by the reverse solution of (13). For instance, a high relative PWV in the aorta may suggest a stiffer arterial wall in some individuals. As implied in the beginning, PWV is not constant even in the same vessel segment. Rather, it can be derived as a monotonically increasing function of pressure. I.e., PWV is different when measured at different stages of the cardiac cycle [55].

Custom wireless reflectance pulse oximeters were employed to simultaneously acquire PPGs from the wrist and fingertip (middle finger). Each pulse oximeter yields four PPG channels: separate pulsatile (AC) and baseline (DC) sample sequences for the red and near-infrared excitation wavelengths. The sampling frequency is $f_s = 240$ Hz to avoid aliasing, e.g., 120 Hz flicker from full-wave-rectified fluorescent room lights. Each channel is unfiltered and has a precision of 4096 digitization levels that correspond to a (0, 2.4) V range.

Since (a) blood oxygen saturation is not the focus of this effort and (b) each near-infrared channel exhibits better signal quality than its red counterpart, only the near-infrared AC and DC channels of the two devices are utilized for PWV calculations. The DC signal (also referred to as the ‘baseline’ in this dissertation) is not a constant in this model. It is controllable and may jump (change value instantly) to resist saturation of AC signal; the influence of the jump on the AC signal is removed later. More details are described in Chapter 4.

When synchronized PPG waveforms from two pulse oximeters placed at the wrist and the fingertip (see Figure 9.1) are ready, time differences can be extracted from point pairs on the corresponding curves. A typical PPG cycle has a steep rising slope and a mild falling slope consistent with the systolic and diastolic durations in the cardiac cycle. For this study, three point

pairs were selected at different waveform features: (a) the valley (foot), (b) the inflection point on the rising slope and (c) the time corresponding to the rising-slope peak. This offers the opportunity to calculate and compare three PWV values at different intra-arterial pressure levels. In other words, this work was initiated to see whether the estimated PWVs are consistent with PWVs obtained using other methods, such as PPT from DVP, and whether PWV exhibits measurable dependency on pressure.

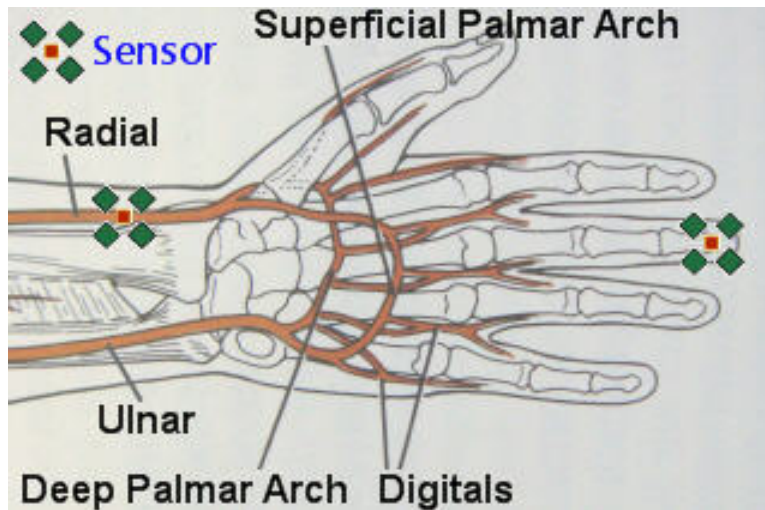


Figure 9.1. Measurement setup with one pulse oximeter (sensor) at the wrist and the other at the fingertip. The primary arterial tree of the hand is depicted to illustrate the pulse travel path. This figure is adapted from [88].

Basic Approach

Two approaches are explored in this chapter to retrieve synchronized PPG streams. The first method was presented in [70]. The two pulse oximeter sensors talk to their respective receivers, which are either two serial ports or ZigBee coordinators.

Signal Synchronization

A MATLAB interface (Figure 9.2) collected, through serial communication, near-infrared data acquired by two pulse oximeters. The respective communication parameters (serial port, sample rate, signal channel, wireless channel, and refresh rate) can be set on the left panel. Note that the displayed near-infrared waveforms in the two sub-windows on the right are visualized in a ‘first ready, first displayed’ unsynchronized manner. However, the PPG signals

generated by the two pulse oximeters must be synchronized so as to accurately extract the relative time delay on the waveform acquired from the distal sensor, where the sensor-to-sensor distance is known. Hence, an interface feature was developed that ‘punches’ the data sets when they are dumped from two separate serial buffers at the same time. This ‘punch’ method ensures that the punched points on the two waveforms are time synchronized.

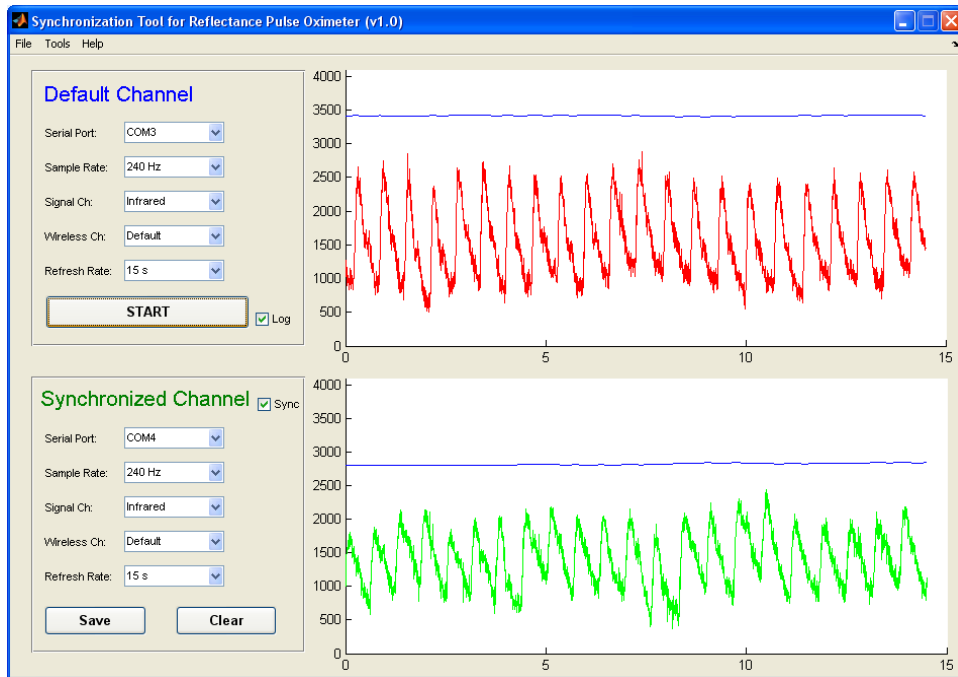


Figure 9.2. A MATLAB interface that simultaneously communicates with two pulse oximeters on different serial ports.

Figure 9.3 illustrates two PPG waveforms that are synchronized by aligning the corresponding punched points in their data sets. The punch dots on these AC channels have a uniform value of 2048. The DC/baseline channels are also punched (not shown here) to further confirm the punch dots.

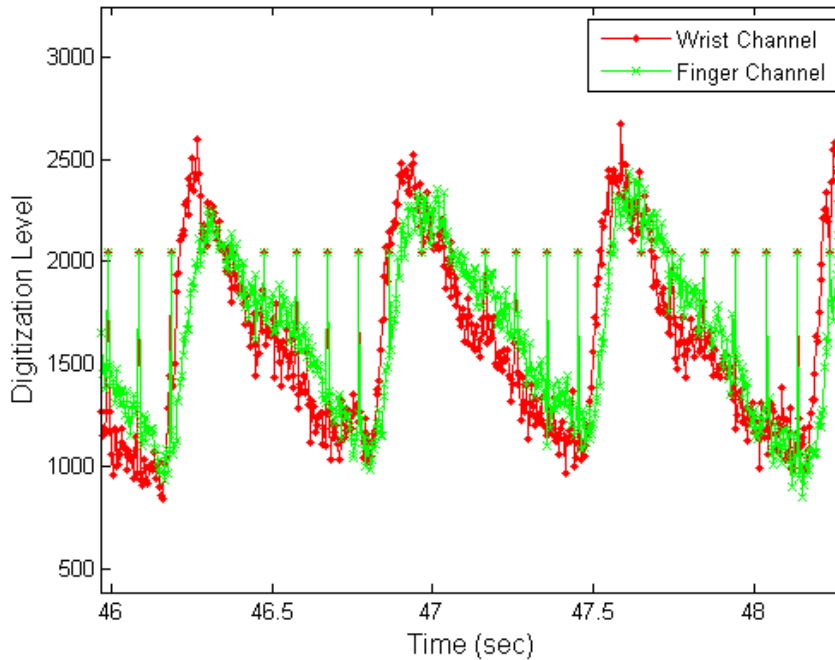


Figure 9.3. Punched raw PPG waveforms that are aligned (synchronized).

Signal Preprocessing

Removing the punched points from the synchronized signals yields raw PPGs (see Figure 9.4) for further processing. The second preprocessing step is to filter the raw PPGs. The filter is a 240th-order lowpass filter with a 5 Hz cutoff frequency realized with a MATLAB function `firls()`, a linear-phase FIR filter that uses least-squares error minimization. This high-order filter causes a time delay of $t_d = (n-1)/(2f_s) = 0.498$ seconds, where $n = 240$ and $f_s = 240$ Hz. The group time delay equally pushes all cycles (including each of their individual frequency components) to the right as seen in Figure 9.5 when compared to Figure 9.4 without phase distortion. The cutoff frequency is set to the relatively low value of 5 Hz to ensure smooth derivative curves (see Figure 9.6). A higher cutoff frequency would require a method such as curve fitting to be employed when extracting the associated delays from these derivative data.

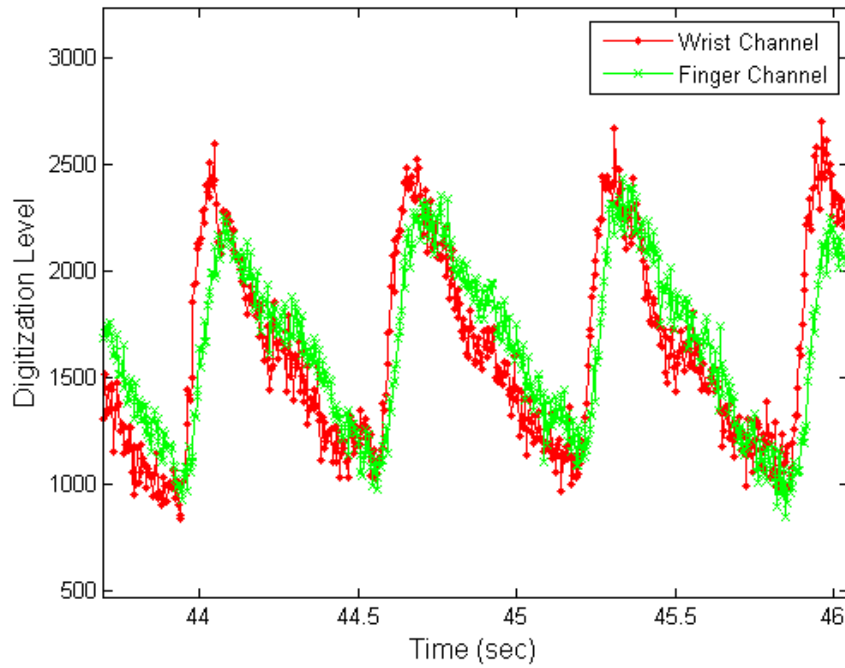


Figure 9.4. Synchronized raw PPG waveforms with the punch marks removed.

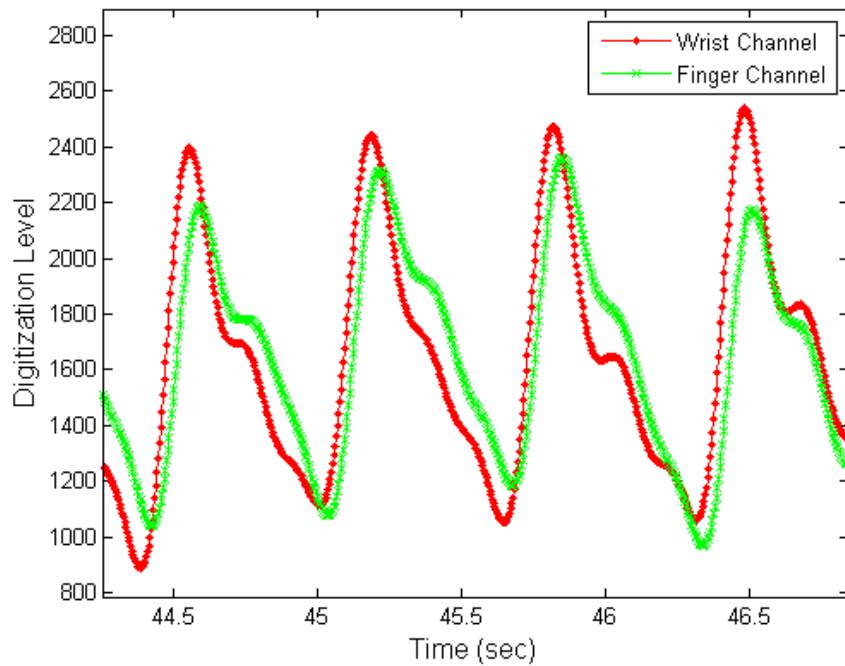


Figure 9.5. Filtered PPG waveforms. The channels are still synchronized due to the same group delay imposed by the linear-phase lowpass filter.

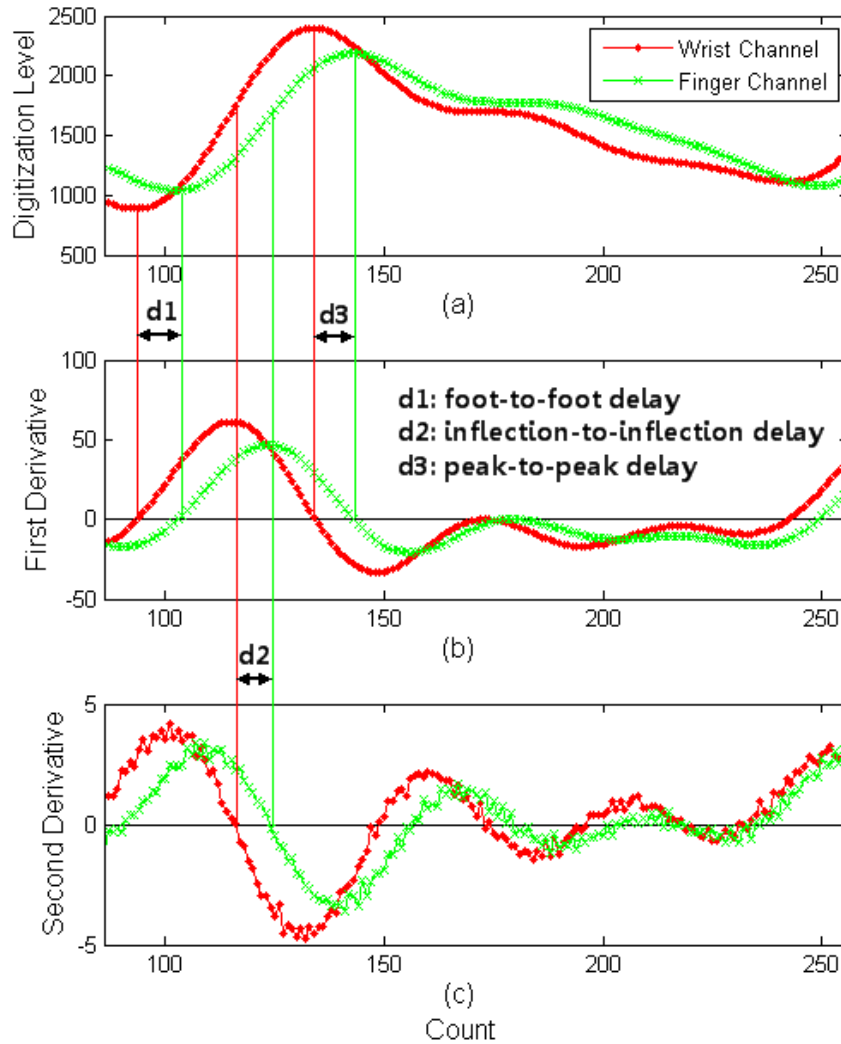


Figure 9.6. First derivative (b) and second derivative (c) of a pair of filtered single-cycle PPGs (a). Three delays, d1, d2, and d3, are extracted according the count differences represented by the line at $y = 0$ in (b) and (c).

Pulse Travel Time Extraction

The time difference between the two PPG waveforms becomes clearer after filtering. Because of the discrete signal nature, the time axis is really a count axis. Figure 9.6 (a) depicts this idea with a pair of filtered, single-cycle PPGs, each containing ~ 150 data points. The time duration between consecutive samples is $1/f_s = \sim 0.417$ milliseconds, the best time resolution available. To identify the corresponding PPG feet and peak locations, the first derivative was

employed (see Figure 9.6 (b)). The second derivative was used to locate inflection points on the rising slope (see Figure 9.6 (c)). Count delays were subsequently extracted using the zero values of the appropriate differentiated data, as illustrated in Figure 9.6.

Count Differences

A ten-second pair of near-infrared PPG segments was chosen, where the selection criterion was minimal variation in the baseline, which is preferred since severe baseline fluctuations normally imply motion. Each segment contains 2400 ($10 f_s$) data points and 15 whole PPG cycles (rising slopes), which means that 15 sets of time differences/PWVs can be extracted and averaged for each waveform feature. The results are detailed in Table 9.1. Taking the first cycle as an example, as in Figure 9.6, the foot indices for the wrist and fingertip PPGs are 94 and 103, respectively, so the FTF (foot-to-foot) delay is $103 - 94 = 9$ counts. Likewise, the ITI (inflection-to-inflection) delay is $124 - 116 = 8$ counts, and the PTP (peak-to-peak) delay is $143 - 134 = 9$ counts.

Averaging these values over 15 cycles yields FTF = 7.1 counts, ITI = 6.5 counts, and PTP = 8.3 counts. The corresponding standard deviations are as high as 1.5 counts, which is relatively large. Though the pulse oximeter has a 240 Hz sampling rate, only 6-to-8 counts exist between similar features on the two waves: PWV is much faster than blood flow. Considering that the red channel does not contribute to PWV calculation, we assume that the removal of those code blocks from the firmware could improve the sampling frequency to 480 Hz.

Pulse Wave Velocity Estimation

Count differences can be converted into time differences in seconds by dividing by f_s . If the pulse travel length (TL in Table 9.1) between the sensors is measured, the three variations on PWV can be calculated from the time differences. The distance between two measurement locations is 0.224 m; however, the arterial distance is slightly longer when one considers the indirect paths in the arterial tree of the hand (see Figure 9.1.). A subjective factor of 1.25 was therefore used to get $TL = 0.224 * 1.25 = 0.280$ m. The PWVs using this estimated TL are listed in Table 9.1.

Table 9.1. PWV Results from a Ten-Second PPG Waveform Pair from One Subject

Cycle	Wrist Foot	Finger Foot	Wrist Peak	Finger Peak	Wrist Inflect.	Finger Inflect.	FTF (count)	ITI (count)	PTP (count)	TL (m)	FTF (m/s)	ITI (m/s)	PTP (m/s)	
1	94	103	134	143	116	124	9	8	9	0.280	7.47	8.40	7.47	
2	242	249	286	293	266	272	7	6	7		9.60	11.20	9.60	
3	396	403	437	445	419	425	7	6	8		9.60	11.20	8.40	
4	556	562	596	603	579	584	6	5	7		11.20	13.44	9.60	
5	711	719	752	760	734	740	8	6	8		8.40	11.20	8.40	
6	870	877	910	918	892	898	7	6	8		9.60	11.20	8.40	
7	1037	1045	1077	1087	1059	1067	8	8	10		8.40	8.40	6.72	
8	1205	1211	1244	1250	1226	1232	6	6	6		11.20	11.20	11.20	
9	1365	1370	1406	1415	1388	1392	5	4	9		13.44	16.80	7.47	
10	1527	1533	1566	1576	1548	1555	6	7	10		11.20	9.60	6.72	
11	1690	1698	1731	1740	1712	1719	8	7	9		8.40	9.60	7.47	
12	1847	1855	1888	1896	1870	1875	8	5	8		8.40	13.44	8.40	
13	2000	2009	2042	2052	2024	2032	9	8	10		7.47	8.40	6.72	
14	2156	2160	2197	2204	2178	2185	4	7	7		16.80	9.60	9.60	
15	2311	2320	2353	2362	2334	2342	9	8	9		7.47	8.40	7.47	
AVG = Average							AVG	7.13	6.47	8.33		9.91	10.80	8.24
STD = Standard Deviation							STD	1.50	1.24	1.23		2.57	2.36	1.30

FTF = foot-to-foot; ITI = inflection point to inflection point; PTP = peak-to-peak. TL = travel length of pulse wave at the hand.

As noted in the previous section, PWV should increase as pressure increases, which occurs along the rising slope of a PPG. Hence, PWVs calculated from the feet, inflection points, and peaks of the rising slopes should be ordered from least to greatest. These early results do indicate an increase in average PWV from the foot (9.9 m/s) to the inflection point (10.8 m/s) but a decrease at the peak (8.2 m/s). The next two paragraphs contain thoughts in that regard.

First, a PPG as in Figure 9.6 is composed of two waveforms that correspond to different time delays: (a) the systolic (or direct) component that results from the direct transmission of the systolic pressure wave from the aorta to the arm and (b) the diastolic (or reflected) component that results from the reflection of the pressure wave from the peripheral arteries to the aorta and then back to the arm [55], [86]. The peak of the rising slope therefore experiences dispersion caused by these wave reflections, which also result in minor peaks that appear on the falling PPG slope. Effects of reflections on PWV values are also noted in [55].

Second, two small arteries converge in the fingertip: one from the superficial palmar arch and the other from the deep palmar arch. These arches bridge the radial artery (where the

other device is placed) and the ulnar artery. This structure makes the fingertip pulse waveform more complex than the waveform one would model for a simple vessel segment.

WBAN Approach

The second approach utilized the high resolution WBAN introduced in Chapter 8. The WBAN method [89] is similar to the previous study but improves on the approach by

1. Forming a star-topology WBAN with a single receiver and a single communication channel,
2. Doubling the sensor sampling frequency to 480 Hz, theoretically doubling the resultant resolution, and
3. Using a curve fitting method over several cardiac cycles to build a more representative PPG segment, since PPG behavior statistically repeats over time.

The pulse oximeter sensor nodes transmit their own samples continuously (in real time), letting the CSMA/CA algorithm decide which sensor communicates to the receiver during a certain time slot. Data streams received from pulse oximeter sensor nodes are therefore fragmented, but these fragmented data segments are ‘geared’ together in a real-time manner. Note that since this approach aims to extract features from pulsatile high-resolution signals, the missed segments can be rebuilt afterwards.

Decomposition of the Sensor Data Streams

The PWV application involves two sensor nodes placed at the wrist (node 1) and the corresponding fingertip (node 2). PWVs are calculated from the time differences between these two PPGs. The first step is to separate the two sensor data streams based on their IDs. Figure 9.7 plots two example waveforms in their received order; the data points with negative values represent placeholder frames.

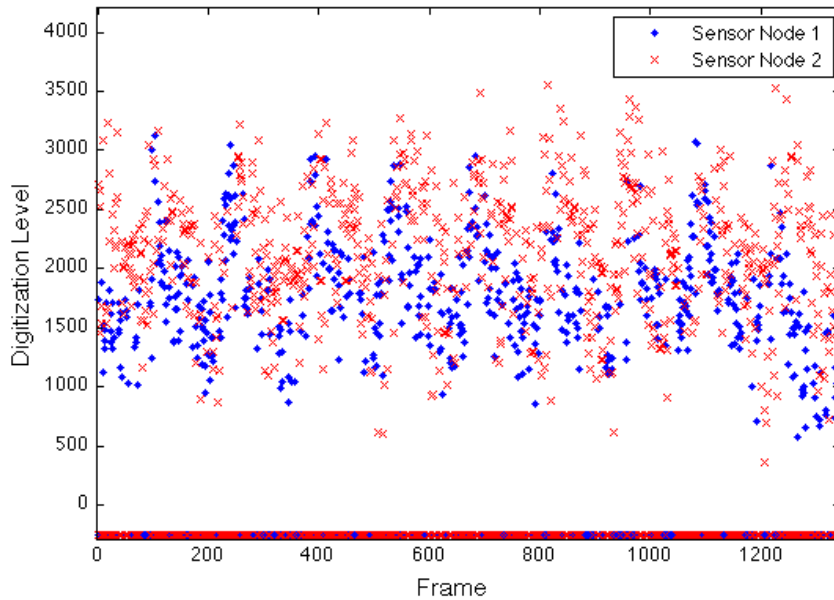


Figure 9.7. The original frame sequence decomposed into two PPGs. Each frame carries two AC values, but only one is plotted here.

Timeline-Recovered Waveforms

Sequence #s are utilized to time-align data points for each sensor node. The rearranged results are depicted in Figure 9.8. This time, data points with negative values are from placeholders corresponding to lost frames. Here, synchronization has not been addressed yet.

Defragmentation and Curve Fitting

Although the PPG data points for node 1 (blue dots) are connected with lines in Figure 9.8, they are actually fragmented or composed of small data segments. I.e., the sequence # is not continuous, which is also indicated by the points with negative values. To eliminate the discontinuities, a peak detection algorithm is employed to identify individual PPG cycles, and these cycles are then ‘stacked’ together; cycles are aligned at the systolic peak position. Figure 9.9 shows a representative PPG (blue circles, including at least one whole cycle) averaged from eight cycles worth of discrete, discontinuous data. Note that the x -axis is changed to Count, since each frame contains two data points. The sampling frequency = 480 Hz. A 10th-order polynomial curve was fitted to these data to estimate the continuous PPG.

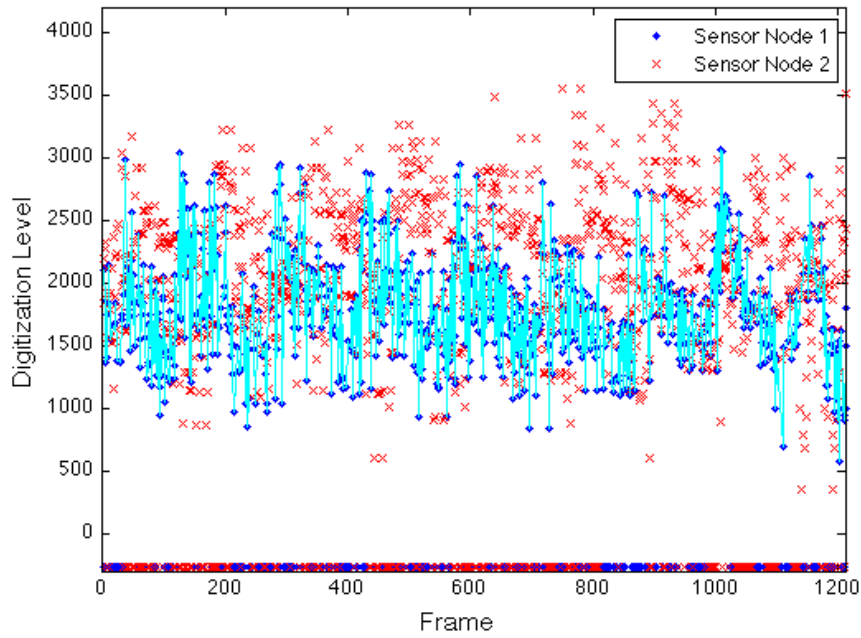


Figure 9.8. Two PPGs with data points time-aligned.

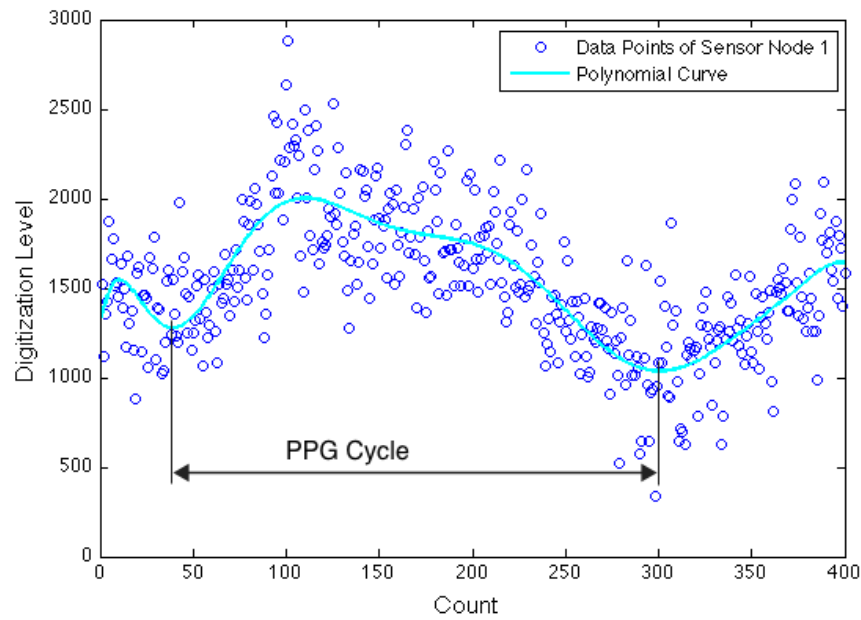


Figure 9.9. Representative waveform for eight PPG cycles worth of discontinuous, discrete data.

Synchronization and Feature Extraction

After the polynomial curves for both nodes are calculated, they need to be placed appropriately, or synchronized, relative to one another, in order to complete the PWV calculation. The positioning process depends on two factors related to both nodes: (a) the corresponding peak position (the first peak of the eight cycles) in the time aligned sequence and (b) the delay relative to the expected sequence # based on the results of the linear regression method in Chapter 9. Figure 9.10 shows the relative position of two polynomial curves after statistical synchronization.

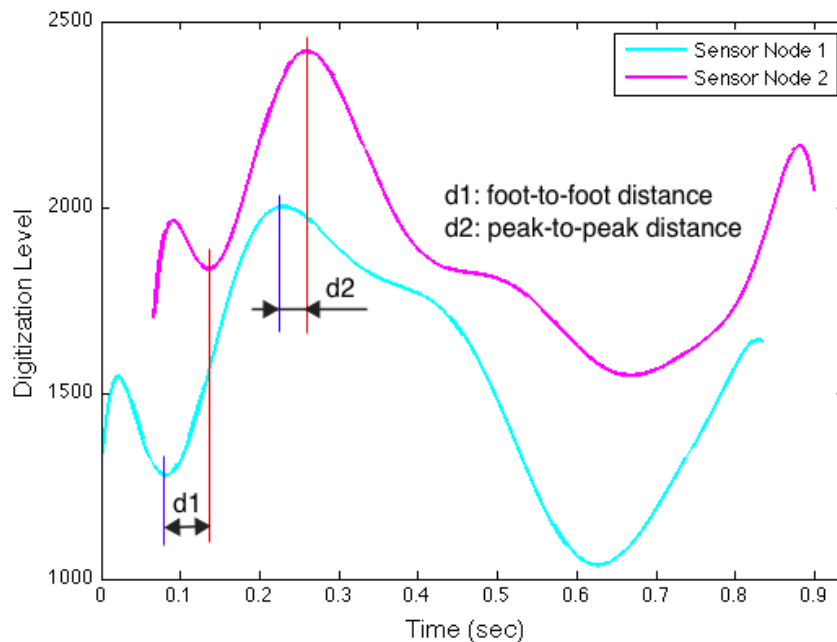


Figure 9.10. Statistically synchronized, continuous PPGs from two sensor nodes.

To calculate PWV, corresponding features on the rising slope of the two waveforms could be identified toward the use of the time difference extraction approaches used in [70], which involve taking derivatives of the PPGs. For example, the foot-to-foot and peak-to-peak differences are determined by first finding the roots of the first derivatives and then performing a subtraction (see Figure 9.10). In this example, three PWV values are estimated as 5.12 m/s at the foot, 5.51 m/s at the inflection point, and 9.47 m/s at the peak.

Chapter 10 - Medical Device Coordination Framework

The MDCF is an open-source project with a goal of exploring means to design, implement, verify, and certify systems of medical devices [34], [90]. Existing MDDS systems are constrained by FDA regulations to only forward data from devices to remote displays or hospital information systems without transforming those data. The MDCF effort was initiated by the U.S. Food and Drug Administration in partnership with Kansas State University, then funded by the National Science Foundation, as an open test bed to allow academics, industry, and government regulators to explore engineering and safety issues involved in removing MDDS restrictions to allow multiple devices to be integrated as a system, where the behavior of the system is determined by the execution of one or more APPs that can combine/transform device data and control the actions of devices. This work is informed by related efforts pursued by the Medical Device “Plug-and-Play” Interoperability Program (MDPnP) [91], which is developing standards and prototypes for systems of cooperating devices. MDPnP has demonstrated a number of interesting applications of this concept, including (a) implementations of more effective safety interlocks and smart alarms that span multiple devices and (b) automated clinical workflows that previously proved problematic due to manual errors [92].

JMS and MDCF Architecture

The message-oriented middleware (MOM) architecture of MDCF is implemented based on Java Message Service (JMS). The features JMS provides to MDCF include a) flexible and dynamic information flow, b) many message providers and consumers (e.g., fan-in and fan-out structure), and c) time-critical and scalable performance [34]. OpenJMS is an open source implementation of JMS API 1.1 specification [93]. It provides five steps to set up a JMS and send and receive messages over JMS.

1. Creating a connection factory from a JNDI (Java Naming and Directory Interface) InitialContext object which specifies where the data store is (e.g., tcp://localhost:61616, analogous to the root of a directory tree for a file system).
2. Creating a connection from the connection factory.

3. Creating a session from the connection.
4. Retrieving a destination from a session. A destination can be topic or queue.
5. Sending messages to or receiving messages from the destination.

In MDCF, these five steps are repacked into three steps (one line of code for each step) when set up a JMS communication in a component. Taking sending messages for example, two classes `SenderFactory` and `IMDCFSender` are utilized from MDCF core package.

1. Creating a `IMDCFSender` object for the `SenderFactory`
2. Connecting the `IMDCFSender` object with a topic.
3. Sending messages from the `IMDCFSender` object.

Figure 10.1 depicts the general structure of the MDCF in two principle parts: an MDCF server and an MDCF client. The MDCF Core is the fundamental layer of this framework; it provides a message-oriented-middleware bus (which can be instantiated as an implementation of the Java Message Service (JMS) [34] or a light-weight TCP socket service) that supports a publish-subscribe communication paradigm. Message senders and receivers can connect to a particular MDCF *channel* (similar to a JMS *topic*), and senders can publish messages to a channel that are delivered asynchronously to all receivers connected to that channel. Medical devices and hospital information systems such as an EHR are MDCF *clients* and communicate with the MDCF using channels. For example, heart rate and blood oxygen saturation data streams from a conventional pulse oximeter would be communicated to the MDCF from the device using a separate outgoing channel for each data stream (the device acts as the sender). An MDCF client device typically will also have an incoming channel that carries commands to which it responds. Channel connections on the client side are realized using a lightweight communications library that can either be incorporated into the software/firmware of the medical device (an approach taken with other experimental devices on which we are working that include the ability to run embedded Linux) or implemented in a dongle/adaptor that converts signals on the device's native interface to MDCF-compliant messages (a strategy similar to the approach taken in many MDDS systems, and the approach taken with the pulse oximeter discussed later in this chapter).

The MDCF server contains several essential manager and console modules that allow clinicians and hospital IT staff to maintain and supervise MDCF operation. For example, the Device Manager takes care of device authentication and registration when a new device is plugged into the MDCF and assigns topics/channels to the device for data streaming based on its declared MDCF interface.

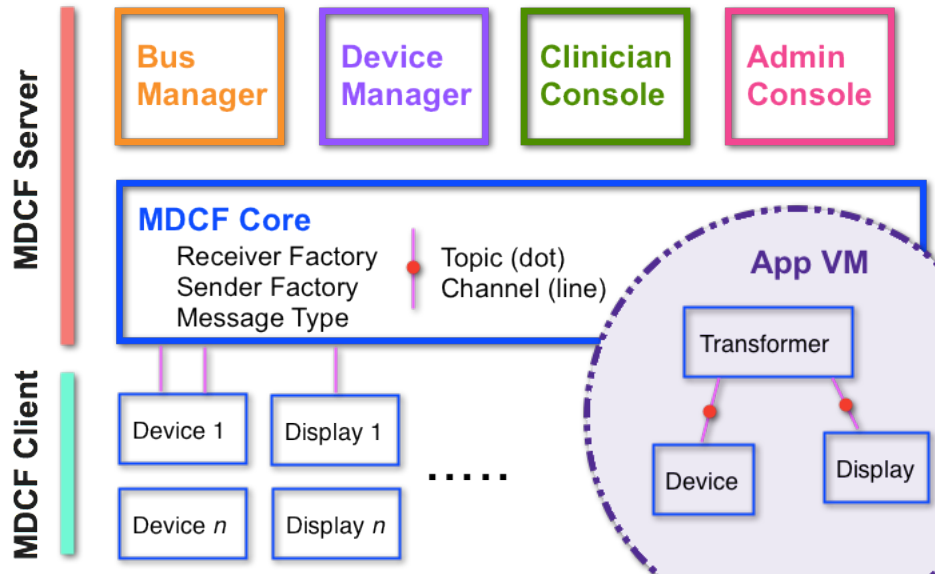


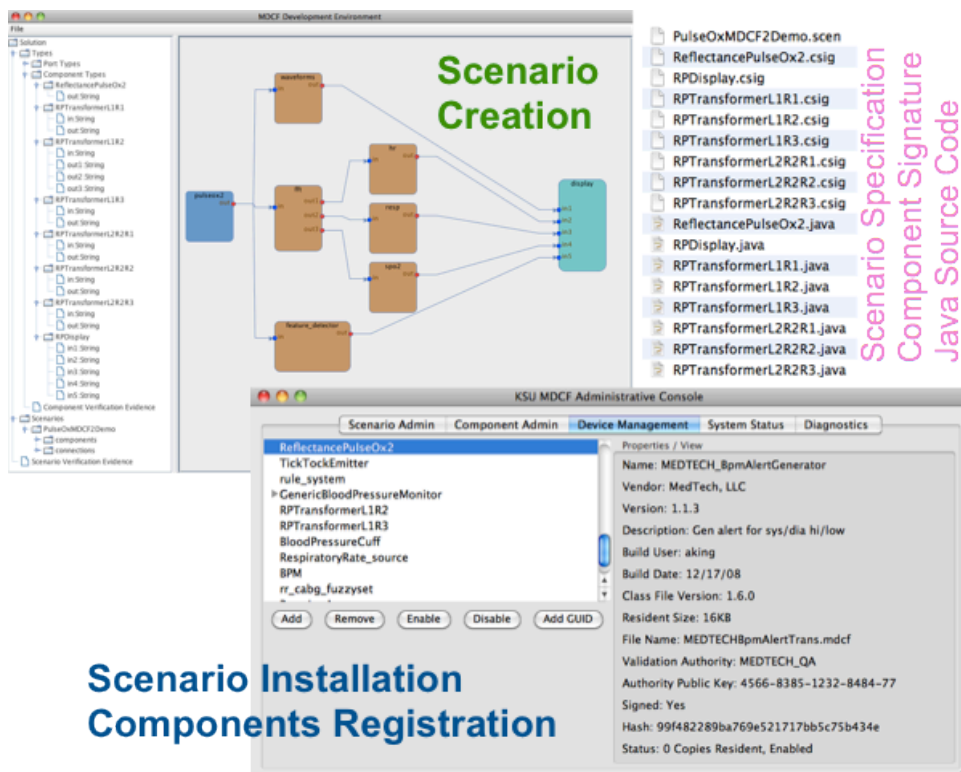
Figure 10.1. MDCF architecture and example APP virtual machine (lower right).

A key module of the MDCF server is the APP virtual machine that executes APPs built from libraries of reusable components with message ports for channel-based communication. An APP is assembled in the MDCF APP development environment by wiring together the message ports of component instances. Three primary component types exist: device (data input), transformer (data processing), and display (data output). Typically, a collection of at least three components forms an MDCF APP.

Scenario Creation and Installation

The previous section addressed high-level descriptions of medical devices employed in the MDCF. In the following sections, a walkthrough of building an APP in the MDCF presents a step-by-step tutorial for building such a medical device from scratch using the custom MPOD pulse oximeter and the tools provided by the MDCF.

A clinical scenario typically involves several medical devices or APPs. For example, in an MDCF APP for closed-loop control of a patient-controlled analgesia (PCA) pump [90], the APP includes components to control the PCA pump and other vital sign monitors such as a pulse oximeter and a respiration rate monitor. The APP controls the PCA pump according to the physiologic indices received. Since the focus of this section is to demonstrate the feasibility of reconfigurable APPs for a single device as applied to, e.g., PPG processing (within the context of the MDCF), rather than create a multiple-device-coordination example, we illustrate our approach with a forward structure APP in Figure 10.2.



**Scenario Installation
Components Registration**

**Scenario Specification
Component Signature
Java Source Code**

Figure 10.2. MDCF Development Environment (upper left), auto-generated files (upper right), and MDCF Administrative Console (lower right).

The specification of each APP component, as well as the assembly of the APP itself, is created in the MDCF Development Environment as illustrated in Figure 10.2 (upper left). First, each component interface is defined in terms of its input/output channel ports. Then, MDCF code-generation facilities generate most of the component implementation, leaving the developer to only fill in the business logic (e.g., specific transformation algorithm) for each component.

Components are then wired together into an assembly as illustrated in Figure 10.2. MDCF code-generation facilities generate executable APP code to be installed and executed in the MDCF APP virtual machine.

Medical Device Class

Most of the Java code for a Medical Device (MD) class is auto-generated by the MDCF Development Environment during the scenario creation session (see the class created for the pulse oximeter in Figure 10.3). This code addresses: (a) topic assignment/naming, (b) channel establishment, (c) initial communication with the bus manager, (d) implementation of the interactions with the middleware for message passing, etc. The major part of the code that requires supplemental attention is the definition of the data to send and the method to pack the data into a particular type of message.

```
public class ReflectancePulse0x2 extends BaseMachine{

    XStream xstream = null;

    IMDCFSender outSender = null;

    InputStream inputStream = null;

    public ReflectancePulse0x2(){
        UUID = "1306964681175"; //modify as required, must be set
        xstream = new XStream();
        setupChan = ReceiverFactory.createReceiver();
        setupChan.connect("setupChan");
        setupChan.regAsynch(new setupChanListener());
        toAdmin = SenderFactory.createSender();
        toAdmin.connect("toAdmin");
        toAdmin.sendMessage("connect,ReflectancePulse0x2,"+UUID);
    }
    ...
}
```

Figure 10.3. Snapshot of a Java code segment for the MD class in a pulse oximeter scenario.

A Medical Device Port (MDP) class will be required to nest inside the MD class, as it talks to the real pulse oximeter device. Since the custom pulse oximeter uses serial communication, RXTX [94], an MDP object was built using a native library that provides serial and parallel communication for the Java Development Toolkit (JDK) (see another code segment in Figure 10.4). The MDP class typically extends the Java Thread class, so the lower level communication is not blocked by, e.g., the malfunction of the MD class. Consistent with the

object-oriented model, the received data from the MDP are packed into a standard message class for transmission. A GenericDeviceMessage is currently used for this purpose. Figure 10.4 also illustrates that the medical data and other device related information (e.g., the sampling frequency) are packed into the message object using its own methods. This message is converted from a Java object to text/XML via the XStream library [95].

```

...
deviceInfo.setMAC(getHexString(ID));
deviceInfo.setSampleRate("240 Hz");
deviceInfo.setWireless("NA");
deviceInfo.setChannels("4");
deviceData.loadChannel("RED_DC", redDCList);
deviceData.loadChannel("RED_AC", redACList);
deviceData.loadChannel("IR_DC", irDCList);
deviceData.loadChannel("IR_AC", irACList);
GenericDeviceMessage mesgObj = new GenericDeviceMessage(deviceInfo, deviceData);
String xml = xstream.toXML(mesgObj);
if (outSender != null) {
    outSender.sendMessage(xml);
}
...

```

Message Packing

Message Transmission

```

private class PulseOximeterDriver extends Thread {
    CommPortIdentifier portId;
    Enumeration portList;
    SerialPort serialPort;
    ...
}

```

MDP Class

Figure 10.4. Snapshot of a Java code segment for the MD class with a nested MDP class.

Transformer Class

In the MDCF autogenerated transformer template, the major section that needs to be filled out by the developer addresses the processing of the received message (GenericDeviceMessage) and the repacking of the results into a new message type for the next component. Figure 10.5 displays a small fragment of the Java source code that defines the classes GenericDeviceMessage and GUIMessage (e.g., required by the following display class). Both Java classes contain the ‘get’ and ‘set’ methods for all of the data fields. For example, when a transformer receives a GenericDeviceMessage in text/XML format, it uses the XStream library to transform it into a GenericDeviceMessage object, where ‘get’ methods are used to extract the raw data loaded by the device component. After the signal processing procedure is complete, the results are ‘put’ into a new GUIMessage object, and the message to send is again transformed to text/XML format by the XStream library.

```

public class GenericDeviceMessage {
    private DeviceInfo DeviceInfo =
    private DevicePayload DevicePayl

    public GenericDeviceMessage(Devi
        this.DeviceInfo = DeviceInfo
        this.DevicePayload = Data;
    }

    public DeviceInfo getDeviceInfo(
        return this.DeviceInfo;
    }

    public DevicePayload getDevicePa
        return this.DevicePayload;
    }
}

public class GUIMessage {
    /* Field */
    private String Timestamp = nu
    private String Topic = nu
    private HashMap<String, String>
    private HashMap<String, ArrayLis
    private DeviceInfo DeviceInfo =
    /* Method */
    public GUIMessage() {}
    // set
    public void setTimestamp(String
    public void setTopic(String temp
    public void loadNumeric(String to
    public void loadWaveform(String
    public void setDeviceInfo(Device
    // get
    public String getTimestamp() {}
}

```

Figure 10.5. Java classes `GenericDeviceMessage` (left) and `GUIMessage` (right).

This process is also described by the code segment in Figure 10.6. A method, e.g., `dataTransform`, typically represents the functionality of a transformer and needs to be added into the template.

```

class inListener implements MessageListener{
    GUIMessage dataMesg = null;
    public void onMessage(Message message){
        //TODO: implement recv logic
        TextMessage tMsg = (TextMessage)message;
        try {
            String msgStr = tMsg.getText();
            Object msgObj = xstream.fromXML(msgStr);
            if (msgObj instanceof GenericDeviceMessage) {
                dataMesg = dataTransform((GenericDeviceMessage)msgObj);
                String xml = xstream.toXML(dataMesg);
                if (outSender != null) {
                    outSender.sendMessage(xml);
                }
            }
        }
    }
}

```

Figure 10.6. The Java code segment for the `Transformer` class that describes the main transformation procedure.

The waveform transformer is designed to generate a PPG signal suitable for display. A number of possible instantiations could be used here, including various forms of waveform smoothing. In this case, the goal is to view the raw PPG, so the transformation implemented is

essentially an “identity” transformation. The transformer ‘gets’ channel data from the GenericDeviceMessage object and ‘puts’ them into the waveform field of a GUIMessage object, functioning as an MDDS. Four channels of raw PPG data are visualized with the display component.

Fourier transformation, which maps a time-domain signal into a complex frequency-domain spectrum, has been used widely for biomedical signal processing. Thus, a real-time fast Fourier transform (FFT) implementation provides a good means to evaluate the signal processing performance of the MDCF transformer component. It also provides useful information about primary signal frequencies (e.g., heart rate and respiration rate as frequencies that correspond to fundamental harmonics) and magnitudes (e.g., the amplitude of a pulsatile PPG). An FFT transformer is a classic example of a deterministic signal processing unit.

In the Feature Detection transformer, a statistical approach is adopted for a decision-making application. It can identify three items: (a) PPGs corrupted by motion artifact, (b) data affected by signal saturation, and (c) clean, usable PPGs versus non-saturated data that do not exhibit meaningful pulsatile wave shapes [67]. Such results promote data quality (e.g., by maximizing the integrity and usability of the associated electronic health records) and system performance (e.g., by releasing system resources when selectively pausing some transformers). These decisions are made based on four features extracted from the PPGs, as listed in Table 10.1. The feature extraction method was designed for onboard application (on a microcontroller), so it requires minimal MDCF system resources.

Table 10.1. Features Extracted from PPGs

Feature	Content	Relation
1	Baseline (*DC channel) variation count	Motion
2	PPG extreme value count	Saturation
3	PPG rising time count	Shape
4	PPG falling time count	Shape

The DC level is assumed to be constant in some pulse oximeter designs or adjustable according to the subject’s vascular profile and perfusion level, as in the custom pulse oximeter employed here.

Display Class

The Display class is a visualization interface for transformer results that may be merged with a user interface (e.g., to set HR and SpO₂ alarm thresholds; alarm signals would be generated by adding additional transformer components to assess the range of the Level 2 transformer outputs). Alternatively, the output data streams can be captured and stored for off-line processing. Here we simply implemented Display instances to visualize the data streams from the aforementioned transformers.

Figure 10.7 and Figure 10.8 illustrate the visualization of waveform and numeric data, respectively. These depictions are implemented by the nested classes Waveform and Numeric in the Display class and can have customized appearance and behavior (e.g., timeline length and refresh rate).

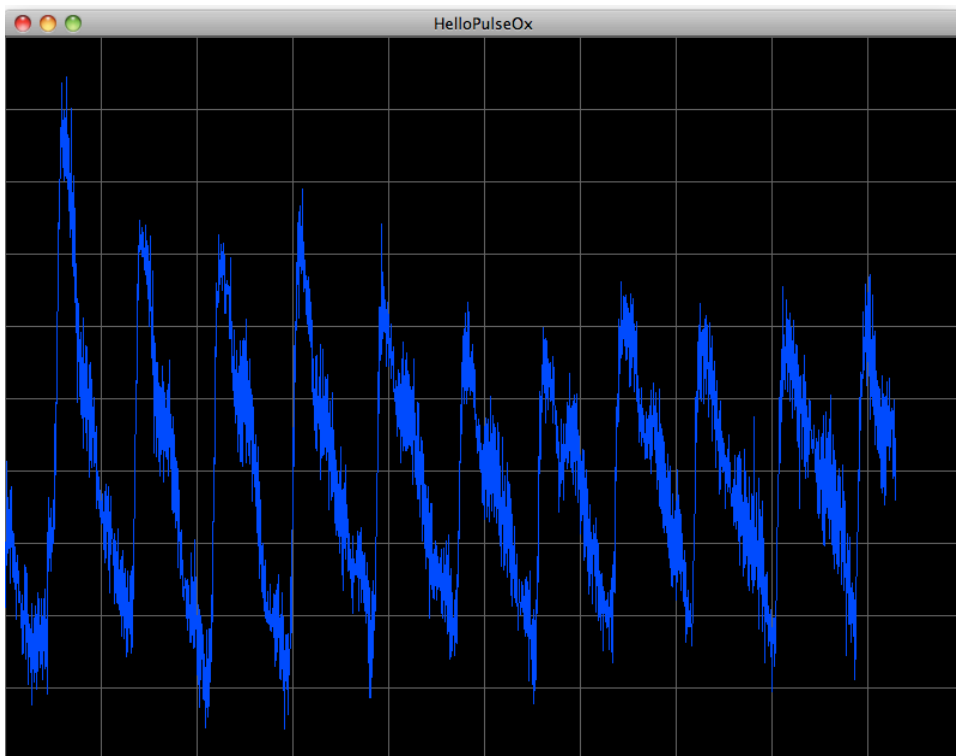


Figure 10.7. PPG waveform visualized in a Display class.



Figure 10.8. Numeric physiological values.

Figure 10.9 shows the two message types printed in their original XML format. The GenericDeviceMessage is packed and transmitted by the device/source component, which consists of a DeviceInfo segment and a DevicePayload segment. Some basic information about the pulse oximeter’s manufacturer, MAC address, etc. could be added under the DeviceInfo tag. Four channels of data are loaded under the DevicePayload tag; each channel has (equally) ~48 data points (48 is the retrieval threshold for the serial buffer). The blue box marked in Figure 10.9 (left) is part of the DC data for the infrared channel.

```

<ksu.mcdl.message.GenericDeviceMessage>
  <DeviceInfo>
    <Manufacturer>ksu.ece.mcdl</Manufacturer>
    <MAC>00158d1122334455</MAC>
    <SampleRate>240 Hz</SampleRate>
    <Channels>4</Channels>
    <Wireless>NA</Wireless>
  </DeviceInfo>
  <DevicePayload>
    <channelMap>
      <entry>
        <string>IR_DC</string>
        <list>
          <string>3364</string>
          <string>3364</string>
          <string>3366</string>
          <string>3366</string>
          <string>3366</string>
          <string>3366</string>
          <string>3366</string>
          <string>3366</string>
          <string>3366</string>
        </list>
      </entry>
    </channelMap>
  </DevicePayload>
</ksu.mcdl.message.GenericDeviceMessage>

<ksu.mcdl.message.GUIMessage>
  <Numerics>
    <entry>
      <string>Feature_1</string>
      <string>65</string>
    </entry>
    <entry>
      <string>Feature_2</string>
      <string>0</string>
    </entry>
    <entry>
      <string>Feature_3</string>
      <string>32</string>
    </entry>
    <entry>
      <string>Feature_4</string>
      <string>57</string>
    </entry>
  </Numerics>
  <Waveforms/>
</ksu.mcdl.message.GUIMessage>

```

Figure 10.9. Instantiation of a GenericDeviceMessage (left, partial content) and its corresponding GUIMessage (right, full content) processed by the Feature Detection transformer.

The GUIMessage shown in Figure 10.9 (right side) is generated by the Feature Detection transformer, where four features (name-key pairs) are loaded under the Numerics tag. The message updates every ~ 0.2 sec (48/240) and represents the most recent three seconds of PPG data; it locally maintains a three-second data buffer. Using the same training thresholds from the Bayesian hypothesis testing in [67], this particular GUIMessage indicates that the most recent 3 seconds of data incorporate (a) no motion, (b) no saturation, and (c) a valid PPG signal. The feature values can also be easily visualized with the numeric components of the Display class (see Figure 10.10).

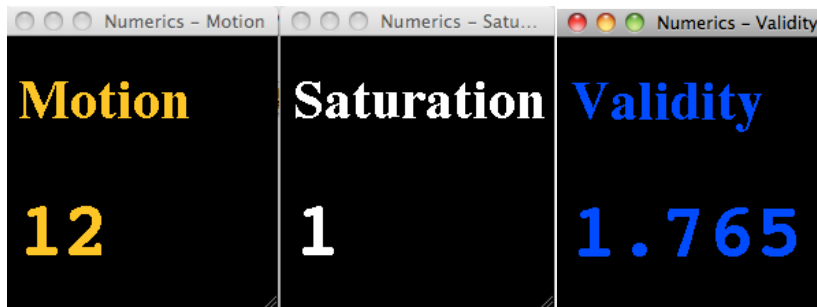


Figure 10.10. Numeric features visualized in a Display class.

Feature detection results (e.g., Motion, Saturation, and Validity) could be used for decision-making applications, such as mapping warnings to physiological parameter thresholds (see Figure 10.11). Such information could also be useful for other transformers. For example, the HR updating algorithm uses $HR_{\text{report}} = (1-a) \times HR_{\text{old}} + a \times HR_{\text{new}}$. A relatively large weight, a , is used for the HR_{new} calculated from the message that is identified to carry a high quality signal.

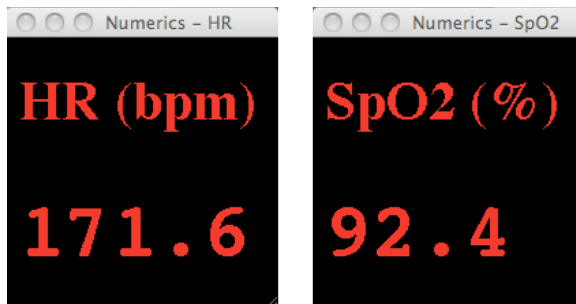


Figure 10.11. Physiological values influenced by feature detection results.

Chapter 11 - Application C: HIIS

HIIS stands for Hospital Information Integration System. It was inspired by the earlier medical device coordination framework concept. The new system is an extension or upgrade of the original MDCF that employs new technologies including a database, AJAX (Asynchronous JavaScript and XML) [96], and Java Servlets [97]. Besides the original message-oriented middleware (MOM) architecture, HIIS internally incorporates database functionality to store, e.g., patient and device information, and externally provides a standard web-based user interface.

From MDCF to HIIS

Figure 11.1 depicts the original MDCF system diagram. All of the components are written in Java and run in a Java runtime environment (JRE). The three basic components – Medical Devices, Transformers, and Displaying Devices – form a basic scenario APP which is managed by the Scenario Console. There are also other managers, and the console coordinates the active components within the framework. JMS provides MOM functionality in this case.

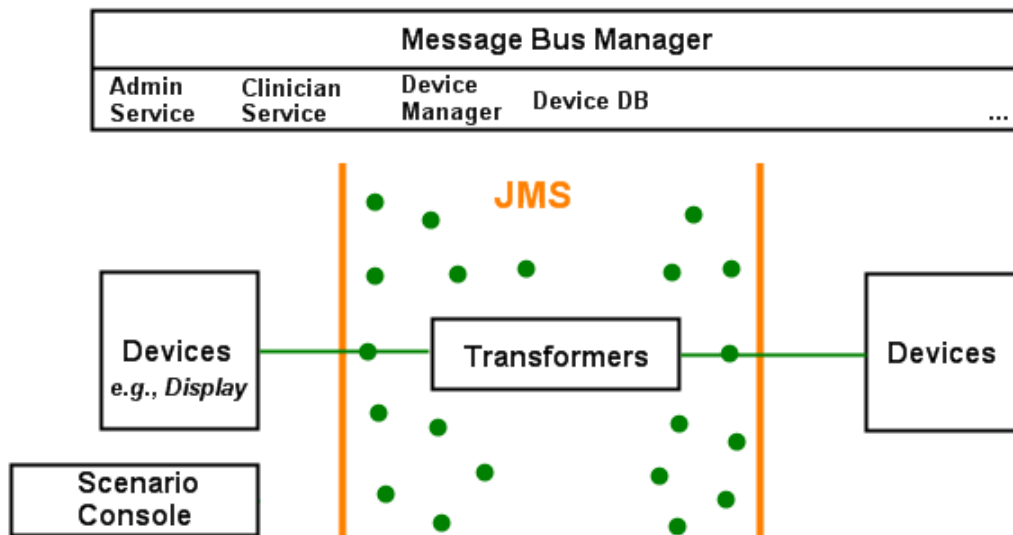


Figure 11.1. MDCF architecture based on the MOM model and the publish-subscribe mechanism.

HIIS primarily extends the MDCF architecture in two ways by (a) expanding the JMS message bus outside of the JRE and (b) integrating database features into the architecture core. Figure 11.2 depicts the new expanded framework, using a pulse oximeter as an example device. The former Display component becomes a web client in the HIIS but is still able to send and receive messages over JMS. The technical tools and details, including ActiveMQ and AJAX, that realize these functions will be introduced in the next section.

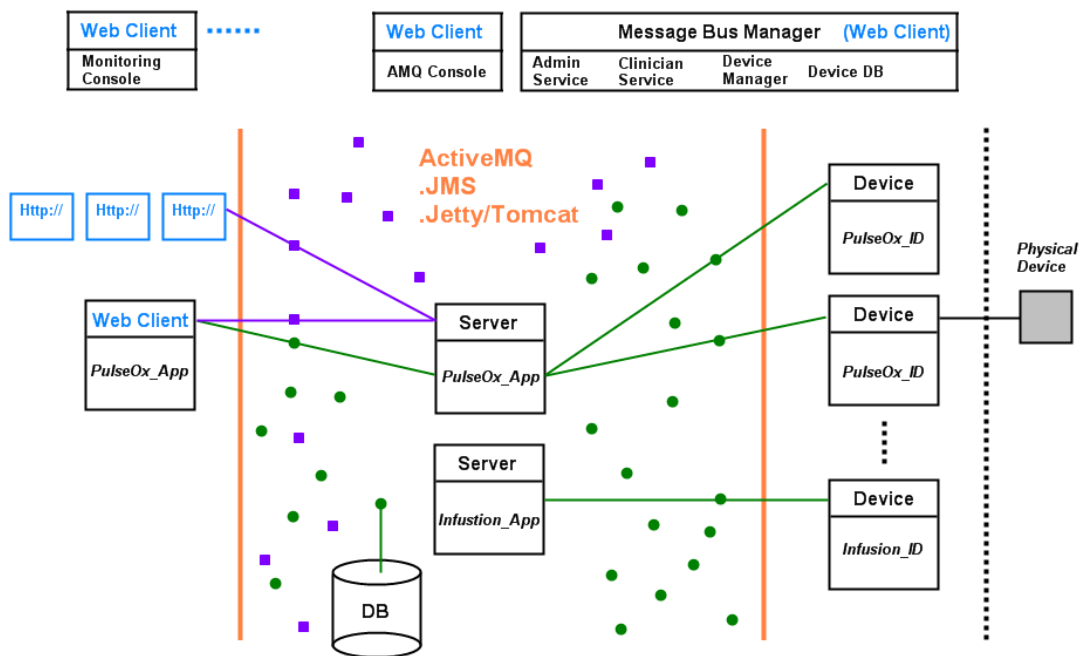


Figure 11.2. HIIS architecture utilizing ActiveMQ and AJAX.

In Figure 11.2, the server or servlet plays the role of a Transformer in the MDCF and communicates with the database component via a topic. An improved HIIS architecture has been designed as shown in Figure 11.3. The server automatically creates the Transformer for the corresponding Device and Display in a particular APP and has direct access to the database, increasing the efficiency and performance of the entire framework. The APP specification, including its transformer code, could be submitted through a web application or manually installed by a server administrator. Since the web client is based on a web browser, it also incorporates an interface for a mobile device such as an iPhone.

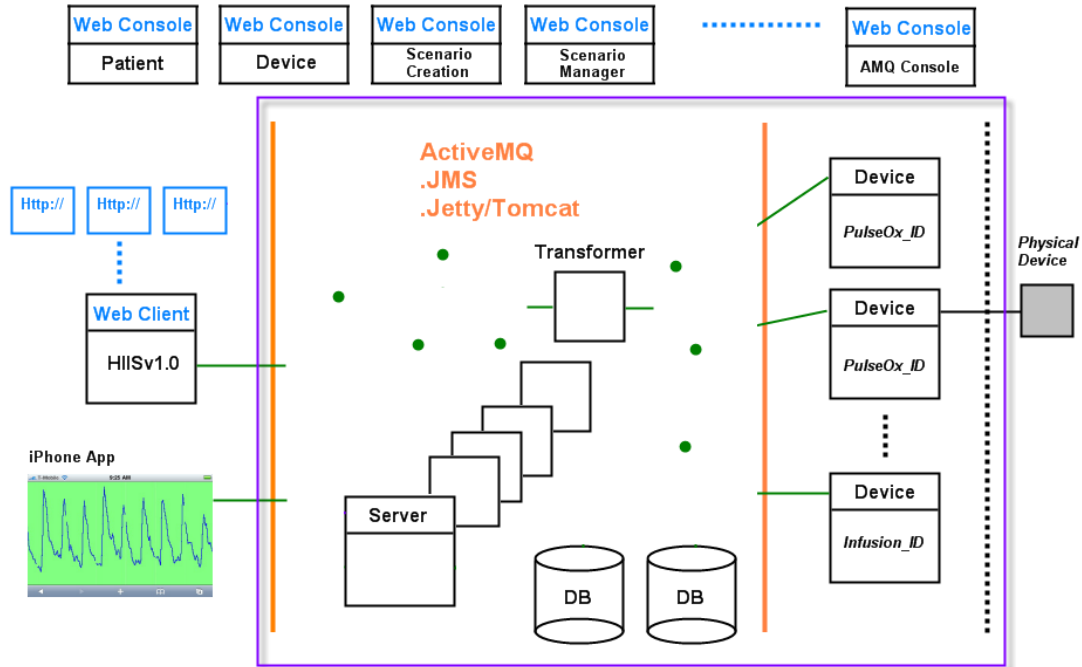


Figure 11.3. HIIS architecture improved by automatic transformer assignment and a mobile device interface.

ActiveMQ, AJAX, and MySQL Database

Apache ActiveMQ is open source software that provides messaging and integration patterns services [98]. The main features utilized by the HIIS are JMS and AJAX support. AJAX supports web streaming, allowing web applications to be a part of the messaging fabric. Instead of using the Java Servlet Jetty included in ActiveMQ, Apache Tomcat is employed, which is an open source software implementation of Java Servlet and JavaPages Server technologies [99].

AJAX makes it possible for a real-time web application to take full advantage of the JMS publish-subscribe mechanism inside ActiveMQ. The realization of this capability is accomplished in two steps. First, one can build Java Servlets that run in the JRE where JMS resides. On the other side, Web Applications, one can then configure Servlets in Tomcat and use the JavaScript API amq.js in the web pages that send and/or receive messages over JMS. To send a JMS message from JavaScript, the following method is called:

```
amq.sendMessage("topic://MCDL.PULSE", mymsg);
```

Here, "topic://MCDL.PULSE" is the topic name, and mymsg is a message in string format. To receive JMS messages, a listener is defined in terms of a topic name and a message handler.

```
amq.addListener('PULSE', 'topic://MCDL.PULSE', rcvMessage);  
var rcvMessage = function(message) {  
    if (message != null) {  
        }  
};
```

Here, 'PULSE' is a string identifier that can be used for a later call to remove the listener:

```
amq.remove('PULSE', 'topic://MCDL.PULSE');
```

All of the message streaming between a web application and the JRE occurs through portals that are Servlets. The following are the Servlets configured in the Tomcat file WEB-INF/web.xml and correspond to services in the HIIS.

- AjaxServlet: The required ActiveMQ AJAX Servlet to support JMS over AJAX.
- TabContent: Refreshes the list for patient, device, scenario, etc. by retrieving information from the corresponding database.
- AjaxSigninServlet: Manages an HIIS user account (administrator).
- PatientsServlet: Provides access to patient data.
- ScenarioServlet: Provides access to scenario data.
- DeviceServlet: Provides access to device data.
- JMSServlet: Enables the addition of a new topic or the decoupling of an existing topic in the TOPICREG database.

The Servlets interact heavily with the database that holds the information for the patients, devices, scenarios, JMS topics, and their relationships. In the HIIS, the open source database MySQL is utilized and is connected to the JRE via the JDBC driver "com.mysql.jdbc.Driver" on the connection "jdbc:mysql:///AJAX?user=ajax&password=ksuecehiis." The following are the SQL commands to create a database and two tables in the HIIS.

```
CREATE DATABASE AJAX;  
USE AJAX;
```

```
CREATE TABLE PATIENTS(  
    ID VARCHAR(32) PRIMARY KEY,  
    AVATAR VARCHAR(32),  
    FIRST_NAME VARCHAR(32),  
    LAST_NAME VARCHAR(32),  
    GENDER VARCHAR(32),  
    BIRTH DATE,  
    ROOM VARCHAR(32),  
    JOINED_DATE DATE,  
    JOINED_TIME TIME,  
    LAST_CHECK TIME,  
    DESCRIPTION VARCHAR(128),  
    NEXT_CHECK TIME,  
    DOCTOR VARCHAR(32),  
    TEL VARCHAR(32),  
    EMAIL VARCHAR(32),  
    ADDRESS VARCHAR(32),  
    CITY VARCHAR(32),  
    STATE VARCHAR(2),  
    ZIPCODE VARCHAR(5),  
    DEVICES VARCHAR(256),  
    TOPICS VARCHAR(1024),  
    MTOPICS VARCHAR(1024));
```

```
CREATE TABLE TOPICREG(  
    NAME VARCHAR(32) PRIMARY KEY,  
    PATIENT VARCHAR(32),  
    DEVICE VARCHAR(32),
```

```
JOINED_DATE DATE,  
JOINED_TIME TIME,  
COUPLE VARCHAR(32));
```

The TOPICREG table is accessed when a new device is connected to the system, where the device is typically associated with a patient ID. The new device will register the topic name that receives messages from consoles or other components in the JMS. Meanwhile, the new TOPICREG record will contain the patient ID associated with the device, which is indicated by a COUPLE flag. When a device is connected to the system, it is not yet coupled with its specified patient, although it has submitted its topic registration information. A nurse or a physician would see a device list associated with a particular patient in a patient console by accessing the PATIENTS table. When a heart beat message is sent from the patient console to the associated device with a registered topic, the status of that device becomes coupled. If there is no response from the device, that associated device will be removed from the device list in the corresponding record in the PATIENTS table.

In conclusion, the HIIS is built upon ActiveMQ (JMS, AJAX), Tomcat (Java Servlet and web server), and MySQL (database). These tools should be appropriately configured and started up prior to HIIS system initialization. For example, Figure 11.4 shows the startup screen of ActiveMQ version 5.3.2. The JRE version is 1.6.0, and the JMS message broker has been started successfully. The transport connection URL is `tcp://localhost:61616`, and the ActiveMQ Console can be accessed at `http://localhost:8161/admin`.

```
kejia@ubuntu: ~
File Edit View Terminal Help
Java Runtime: Sun Microsystems Inc. 1.6.0_21 /usr/java/jdk1.6.0_21/jre
Heap sizes: current=60736k free=59148k max=466048k
JVM args: -Xmx512M -Dorg.apache.activemq.UseDedicatedTaskRunner=true -Djava.
util.logging.config.file=logging.properties -Dcom.sun.management.jmxremote -Dact
ivemq.classpath=/home/kejia/Downloads/apache-activemq-5.3.2/conf; -Dactivemq.hom
e=/home/kejia/Downloads/apache-activemq-5.3.2 -Dactivemq.base=/home/kejia/Downlo
ads/apache-activemq-5.3.2
ACTIVEMQ_HOME: /home/kejia/Downloads/apache-activemq-5.3.2
ACTIVEMQ_BASE: /home/kejia/Downloads/apache-activemq-5.3.2
Loading message broker from: xbean:activemq.xml
INFO | Using Persistence Adapter: org.apache.activemq.store.kahadb.KahaDBPersis
tenceAdapter@1342a67
INFO | Replayed 1 operations from the journal in 0.01 seconds.
INFO | ActiveMQ 5.3.2 JMS Message Broker (localhost) is starting
INFO | For help or more information please see: http://activemq.apache.org/
INFO | Listening for connections at: tcp://ubuntu:61616
INFO | Connector openwire Started
INFO | ActiveMQ JMS Message Broker (localhost, ID:ubuntu-41605-1334848487817-0:
0) started
INFO | Logging to org.slf4j.impl.JCLLoggerAdapter(org.mortbay.log) via org.mort
bay.log.Slf4jLog
INFO | jetty-6.1.9
INFO | ActiveMQ WebConsole initialized.
INFO | Initializing Spring FrameworkServlet 'dispatcher'
INFO | ActiveMQ Console at http://0.0.0.0:8161/admin
INFO | Initializing Spring root WebApplicationContext
INFO | Successfully connected to tcp://localhost:61616
INFO | Camel Console at http://0.0.0.0:8161/camel
INFO | ActiveMQ Web Demos at http://0.0.0.0:8161/demo
INFO | RESTful file access application at http://0.0.0.0:8161/fileserver
INFO | Started SelectChannelConnector@0.0.0.0:8161
```

Figure 11.4. ActiveMQ startup information.

The ActiveMQ console is a useful tool to observe and manage the topics or the queues in the JMS. For example, assume an infusion pump is registered in the database and adds a listener to its registered topic MCDL.INFUSION.1282681389904. That topic also shows up in the ActiveMQ console as in Figure 11.5. The console shows how many messages are consumed by, and queued in, that topic, and the console can be used to send a new message directly to that topic for debugging purposes.

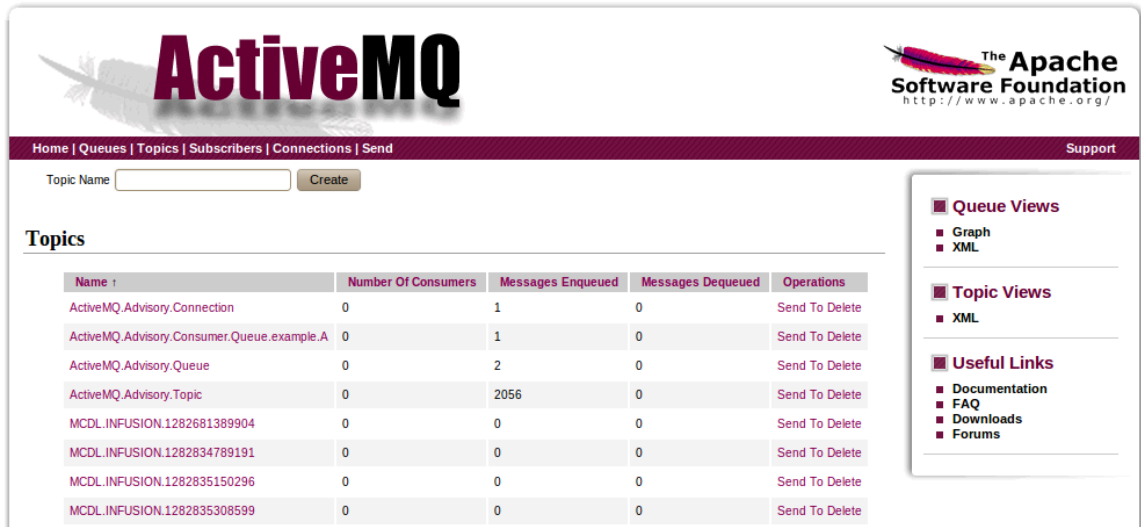


Figure 11.5. ActiveMQ WebConsole with topic information.

HIIS Interface and Consoles

The HIIS user interface is a set of web applications. From the perspective of an HIIS user, it is a web server connected to hospital information databases (e.g., EHRs), physical medical devices, and medical supplies. Figure 11.6 illustrates the homepage after a user logs in with an administrator account. Different types of accounts are granted different levels of accessibility. An administrator has full access and control at present, including the ability to retrieve information from the database and utilize all types of consoles to manage, e.g., patient, device, and scenario records. The homepage shows the patients tab that lists all of the current patients in the hospital; each entry contains a unique patient ID, patient name, room number, time to be checked, alert status, and brief symptom description. A user can click on a patient entry to open that patient's console, as introduced in the next section.

The basic function of the HIIS interface is to list records of patients, devices, and clinical scenarios. For example, a nurse or a doctor could look at each patient's personal and medical information, the room where they are located, and whether an alarm is triggered or attention is needed. Device information can be viewed in the same way under the Medical Devices tab (see Figure 11.7). Each entry is a specific device with device name, manufacturer, device type, function, and inventory status.

Hospital Information Integration System

Patients | Medical Devices | Scenarios

Patient ID	First Name	Last Name	Room NO.	Last Check Time	Next Check Time	Alarm Level	Description
001234	Optimus	Prime	R-023	01:01:01	02:02:02		Rusty
001235	Bumblebee		R-024	00:00:00	00:00:00		Allergy
001237	Ratchet		R-025	00:00:00	00:00:00		Anemia
001238	Jazz		R-026	00:00:00	00:00:00		Blood clot
001239	Ironhide		R-027	00:00:00	00:00:00		Brain tumor
001245	Arcee		R-028	00:00:00	00:00:00		Cold
001247	Jet	Fire	R-029	00:00:00	00:00:00		Diabetes
001249	Mudflap		R-030	00:00:00	00:00:00		Heart attack
001250	Skids		R-031	00:00:00	00:00:00		Cancer
001256	Sideswipe		R-032	00:00:00	00:00:00		H1N1
001267	Jolt		R-101	00:00:00	00:00:00		Early pregnancy
001268	Inferno		R-102	00:00:00	00:00:00		High blood pressure
001271	Weeie		R-108	00:00:00	00:00:00		Depression
001319	Megatron		R-123	11:14:49	00:00:00		Kidney stone

Administrator's Toolbox:

[HIIS User Console](#) registration open now! [Patient Console](#) updates patient EHR information, hospital status, and device association, adds or removes patient entry, etc. [Device Console](#) manages medical device database, defines device interface, etc. [Scenario Creation Console](#) creates new clinician scenario. [Scenario Management Console](#) switches scenario status, deletes scenario, etc. [ActiveMQ Message Broker](#) Apache ActiveMQ is the most popular and powerful open source messaging and Integration Patterns provider.

Figure 11.6. HIIS homepage with patient information, user identity, and administration tools.

Hospital Information Integration System

Patients | Medical Devices | Scenarios

Device Name	Type	Manufacturer	Description	Count	Active
Reflectance PulseOx	Pulse Oximeter	MCDL	HR, Spo2 mornitor	10	Active
Ajax Web Client	GUI	MCDL	Information display	1023	Active
FXS 160.25	X-Ray Machine	Microfocus	Real-Time X-ray system	10	Active
RE950	Pulse Oximeter	Edgepark	HR, Spo2 mornitor	312	Active
BP4900	Blood Pressure Monitor	Braun ExactFit	Blood pressure monitor	341	Active
M Scan 4000	Surgical Camera	Cabot	Surgical video camera	12	Active
iVent201	Ventilator	MSEC	Medical respiratory ventilator	32	Active
VE-3001	Anesthesia Machine	Anesthesia	Medical Products Direct	9	Active
Alarm Button	Button	MCDL	Trigger a alarm message	1023	Active
PLUM A+	IV Infusion Pump	ABBOTT	Infusion intravenous IV pump system	98	Active
Corometrics 172	Fetal Monitor	Fetal monitor	Corometrics	54	Active

Administrator's Toolbox:

[HIIS User Console](#) registration open now! [Patient Console](#) updates patient EHR information, hospital status, and device association, adds or removes patient entry, etc. [Device Console](#) manages medical device database, defines device interface, etc. [Scenario Creation Console](#) creates new clinician scenario. [Scenario Management Console](#) switches scenario status, deletes scenario, etc. [ActiveMQ Message Broker](#) Apache ActiveMQ is the most popular and powerful open source messaging and Integration Patterns provider.

Figure 11.7. HIIS homepage with medical device information.

The clinical scenarios currently used in the hospital are listed under the Scenarios tab, as illustrated in Figure 11.8. Each scenario entry contains the scenario name, a scenario description, and its active/inactive status. Only an active scenario can be applied to patients. The operations of adding a new scenario or altering the active status are managed in their corresponding consoles, which typically requires greater authentication privileges in the HIIS account.

Welcome [root](#) signed in as an administrator | Detected OS: [Linux i686](#) [sign out](#)

Hospital Information Integration System

[Patients](#) | [Medical Devices](#) | [Scenarios](#)

Scenario Name	Description	Active
Patient Alarm	Display alarm information sent from patient rooms	Inactive
Infusion Pump	A closed loop control medical device example	Active
Body Temperature Display	Body temperature display	Inactive
Blood Pressure Monitor	Blood pressure Monitor	Inactive
Reflectance Pulse Oximeter	System real-time test with physical device from MCDL	Inactive
Button Controlled Infusion Pump	Device Coordination Example Button Controlled Infusion Pump	Active

Administrator's Toolbox:

[HIIS User Console](#) registration open now! [Patient Console](#) updates patient EHR information, hospital status, and device association, adds or removes patient entry, etc. [Device Console](#) manages medical device database, defines device interface, etc. [Scenario Creation Console](#) creates new clinician scenario. [Scenario Management Console](#) switches scenario status, deletes scenario, etc. [ActiveMQ Message Broker](#) Apache ActiveMQ is the most popular and powerful open source messaging and Integration Patterns provider.

Figure 11.8. HIIS homepage with clinical scenario information.

For each database (e.g., patient, device, and scenario), the HIIS web interface provides a corresponding console to manage the records. To illustrate this concept, Figure 11.9 displays the patient record management console, where a new patient record can be added and an existing record can be modified by a user with proper permissions. Other convenient functions are to clear the patient alarm and update the check time. A similar record management console exists for medical devices. New devices can be added into the system, the information regarding current devices can be updated, or obsolete devices can be removed or deactivated in the hospital database.

Scenario management is more complicated, since scenarios can be created on the HIIS interface. (An example is presented in the next section.) Moreover, this type of feature represents an important functional deviation from traditional hospital information infrastructures. Scenario APP concepts highlight medical device integration and medical component interoperability. In

the HIIS, scenario management falls into two categories: scenario creation and scenario control. Figure 11.10 shows the console to toggle a scenario's active status.

Patient Console of HIIS

Patient List

Patient ID	First Name	Last Name	Room NO.	Last Check Time	Next Check Time	Description	
001234	Optimus	Prime	R-023	01:01:01	02:02:02	Rusty	
001235	Bumblebee		R-024	00:00:00	00:00:00	Allergy	
001237	Ratchet		R-025	00:00:00	00:00:00	Anemia	
001238	Jazz		R-026	00:00:00	00:00:00	Blood clot	
001239	Ironhide		R-027	00:00:00	00:00:00	Brain tumor	
001245	Arcee		R-028	00:00:00	00:00:00	Cold	
001247	Jet	Fire	R-029	00:00:00	00:00:00	Diabetes	
001249	Mudflap		R-030	00:00:00	00:00:00	Heart attack	
001250	Skids		R-031	00:00:00	00:00:00	Cancer	
001256	Sideswipe		R-032	00:00:00	00:00:00	H1N1	
001267	Jolt		R-101	00:00:00	00:00:00	Early pregnancy	
001268	Inferno		R-102	00:00:00	00:00:00	High blood pressure	
001271	Weelie		R-108	00:00:00	00:00:00	Depression	
001319	Megatron		R-123	11:14:49	00:00:00	Kidney stone	

Patient Entry:

Avatar: First Name: Last Name:

Gender: Date of Birth:

Room: Next Check Time:

Doctor: Description:

Tel: Email:

Address: City: State: Zipcode:

Figure 11.9. Patient record management console in the HIIS.

Scenario Manager of HIIS

Scenario List

Scenario Name	Description	Active	Operation
Patient Alarm	Display alarm information sent from patient rooms	Inactive	<input type="button" value="toggle"/> <input type="button" value="remove"/>
Infusion Pump	A closed loop control medical device example	Active	<input type="button" value="toggle"/> <input type="button" value="remove"/>
Body Temperature Display	Body temperature display	Inactive	<input type="button" value="toggle"/> <input type="button" value="remove"/>
Blood Pressure Monitor	Blood pressure Monitor	Inactive	<input type="button" value="toggle"/> <input type="button" value="remove"/>
Reflectance Pulse Oximeter	System real-time test with physical device from MCDL	Inactive	<input type="button" value="toggle"/> <input type="button" value="remove"/>
Button Controlled Infusion Pump	Device Coordination Example Button Controlled Infusion Pump	Active	<input type="button" value="toggle"/> <input type="button" value="remove"/>

Scenarios Entry:


Figure 11.10. Scenario management console in the HIIS.

Scenario creation was implemented in the MDCF Development Environment as introduced in the previous chapter. In the HIIS, the same thing can be done through the Scenario

Creation Console web application. Figure 11.11 illustrates the original web page showing the available devices/medical components and step-by-step procedures.

Scenario Creation Console of HIIS

Device List Type_0 ▾ Reload



Device Cart & Bus Connection Configuration

Interface / Port Type	Node ID	Device Type	Operation

Scenario Factory

Select a scenario type: Simple Forward ▾ Make

Scenario Uploader

Scenario name:

Description:

Output-Port Device(s):

Input-Prot Device(s):

Transformer(s): Browse... Upload

[View Scenario Configuration](#) | [Upload Scenario Configuration](#)

Figure 11.11. Scenario creation console in the HIIS.

Scenario Creation in the HIIS

Three steps are required to create a scenario APP in the HIIS. First, the medical devices or components that are involved in the scenario must be chosen from the Device List and dragged onto the Device Cart, as depicted in Figure 11.12. A good example is the creation of a button-controlled infusion pump. Each device has its own data visualization window. The four components (Button, IV-Infusion-Pump, and two GUIs) are dragged onto the scenario cart. Each

component has input and output ports. These ports should be properly configured under the Interface/Port Type column by specifying the corresponding component's Node ID. The check button can be used to clear basic logic errors.

Device Cart & Bus Connection Configuration





Interface / Port Type	Node ID	Device	Type	Operation
<input checked="" type="checkbox"/> Input Port Connect to ... ▼	1334937857017		GUI	<input type="checkbox"/> check <input type="button" value="remove"/>
<input checked="" type="checkbox"/> Output Port Connect to ... ▼				
<input checked="" type="checkbox"/> Output Port Connect to ... ▼	1334937855758		Button	<input type="checkbox"/> check <input type="button" value="remove"/>
<input checked="" type="checkbox"/> Input Port Connect to ... ▼	1334937854510		GUI	<input type="checkbox"/> check <input type="button" value="remove"/>
<input checked="" type="checkbox"/> Output Port Connect to ... ▼				
<input checked="" type="checkbox"/> Input Port Connect to ... ▼	1334937853086		IV-Infusion-Pump	<input type="checkbox"/> check <input type="button" value="remove"/>
<input checked="" type="checkbox"/> Output Port Connect to ... ▼				

Figure 11.12. Creation of an example scenario in the HIIS Scenario Creation Console.

The next step of the Scenario Factory requires the selection of the connection pattern and a click on the “make” button. If the connection is sensible (e.g., a connection to only one component), the build results are printed as in Figure 11.13. The display lists each connection between two components and automatically assigns a Transformer in between those components.

Scenario Factory

Select a scenario type: ▼

Devices w/ Output Port	Message	Transformers	Message	Devices w/ Input Port
1334937853086	IV-Infusion-Pump	Transformer[2].java	GUI	1334937854510
1334937857017	GUI	Transformer[1].java	IV-Infusion-Pump	1334937853086
1334937855758	Button	Transformer[0].java	GUI	1334937857017

Message defines the message/data interface of each device.
 Other Messages may be used [GenericCMDMessage](#), [DeviceInfo](#), [DevicePayload](#).
 Define device message in [Device Console](#).
Transformer performs message interpretation and repacking.

Figure 11.13. Automatically created scenario script and Java code templates in the HIIS Scenario Creation Console.

A Java template file can be downloaded from the server using each blue, underlined link in the Scenario Factory. The Java codes for the existing medical devices have already been stored in the HIIS. Hence, in most cases, the only files that need to be uploaded to the server are Transformers, along with appropriate scenario names and descriptions (see Figure 11.14). A Transformer typically receives one type of message, interprets its content, transforms it to a different kind of information, and repacks the results to another type of message that is compatible with the next component. These message specification files are also provided in the Scenario Factory.

Scenario Uploader

Scenario name:

Description:

Output-Port Device(s):

Input-Prot Device(s):

Transformer(s):

[View Scenario Configuration](#) | [Upload Scenario Configuration](#)

Figure 11.14. Upload of a new scenario to the server in the HIIS Scenario Creation Console.

Scenario APP Configuration and Execution in the HIIS

After a new scenario is uploaded to the HIIS server and approved by an administrator, it appears in the scenario management console (see Figure 11.10). The new scenario APP can be used for a patient once its status is active and it is selected in that patient's console. Figure 11.15 illustrates a patient console obtained by clicking an entry in the Patient tab in Figure 11.6. The patient console displays detailed patient information (e.g., connected with an EHR), connected devices, and currently active scenarios.

The connected devices that are listed are associated with the Device List field of the patient's record in the database. One or more of them might not be physically connected or may have connection trouble after the relationship between the patient and the device is established. To address this issue, the care provider could open the Patient Device Setup page to check the connectivity of each listed device or simply click Reload.

In the Activated Scenarios segment, the device component placeholders need to be filled up before a scenario can be initiated. When the cursor is moved over the placeholder, it displays the required device name and ID. When the cursor is moved over the device icon in the Connected Device segment, it also displays its ID. A matched ID is required before a device icon can be dragged onto the placeholder in an activated scenario.

	ID	001234	Name	Optimus Prime	Gender	Male	Date of Birth	1900-02-02
	Room	R-023	Check In Time	2010-07-29 01:01:01	Doctor	Dr. Lee	Status	normal
	Contacts	785-320-1234 optimus@gmail.com		Home Address	X, Earth, KS 66502			
	Sympton Description:	Rusty more						

Connected Devices [Patient Device Setup](#) | [Reload](#)



Activated Scenarios









Infusion Pump	A closed loop control medical device example	Active	<input checked="" type="checkbox"/>
	 		
Button Controlled Infusion Pump	Device Coordination Example Button Controlled Infusion Pump	Active	<input checked="" type="checkbox"/>
	     		

Figure 11.15. Example patient console in the HIIS that displays patient information, connected medical devices, and active scenarios.

Two virtual devices for an infusion pump and a button can be designed in Java to illustrate how to execute the button-controlled infusion pump APP. The button is clicked at a random time, and the resulting event is transmitted as a JMS message. The infusion pump stops for several minutes after receiving the command. In this example, the role of the transformer is simply to convert a click message to a stop-command message.

As a continuation of this example, the infusion pump and the button are associated with patient ID 001234 after they are started. If the Patient Device Setup page of that patient is opened, those two devices are listed with their unique topic names as shown in Figure 11.16. Clicking the check button sends out a heart beat message to the device. If the device responds successfully, the couple status changes to positive in the aforementioned TOPICREG table in the database. Other medical components can also be added, such as the GUI web component to visualize the infusion pump and the button data. The manually-added components will also be assigned unique topic names after submission; these topics names are appended with a symbol "V" to distinguish them from regular topic names.

Patient 001234 Device Setup

Device List Type_0 ▾ Reload








Patient Device Setup Cart Submit | check all

Avatar	Device	Type	Serial NO. / Topic Name	Operation
	Ajax Web Client	GUI	<input type="text"/>	check remove
	Ajax Web Client	GUI	<input type="text"/>	check remove
	Alarm Button	Button	<input type="text" value="MCDL.REG.1334936773337"/>	check remove
	PLUM A+	IV Infusion Pump	<input type="text" value="MCDL.REG.1334936657106"/>	check remove

Figure 11.16. Selection and assignment of medical devices and components in the patient console.

At this point, the user can return to the patient console and add the connected device icons to the corresponding places in the active scenario, as illustrated in Figure 11.17. The scenario name can be clicked to launch the APP.

	ID	001234	Name	Optimus Prime	Gender	Male	Date of Birth	1900-02-02
	Room	R-023	Check In Time	2010-07-29 01:01:01	Doctor	Dr. Lee	Status	normal
	Contacts	785-320-1234 optimus@gmail.com		Home Address	X, Earth, KS 66502			
	Symptom Description:	Rusty more						
	Connected Devices Patient Device Setup Reload							

Activated Scenarios







Button Controlled Infusion Pump	Device Coordination Example Button Controlled Infusion Pump	Active	
	  	<input checked="" type="checkbox"/>	
	  		

Figure 11.17. Sequence to configure and start a scenario for a patient in the patient console.

The scenario APP display is shown in Figure 11.18. The top segment includes the patient and scenario information along with a simple command-input area where commands can be directly send to a device from the drop-down list box. Below that segment are the web GUI gadgets for the button device and the infusion pump. Each GUI gadget contains the device icon, ID, current numerical value, and waveform plot. The simulated button was clicked around time 60, and the infusion pump stopped for a while right after receiving the button's message.

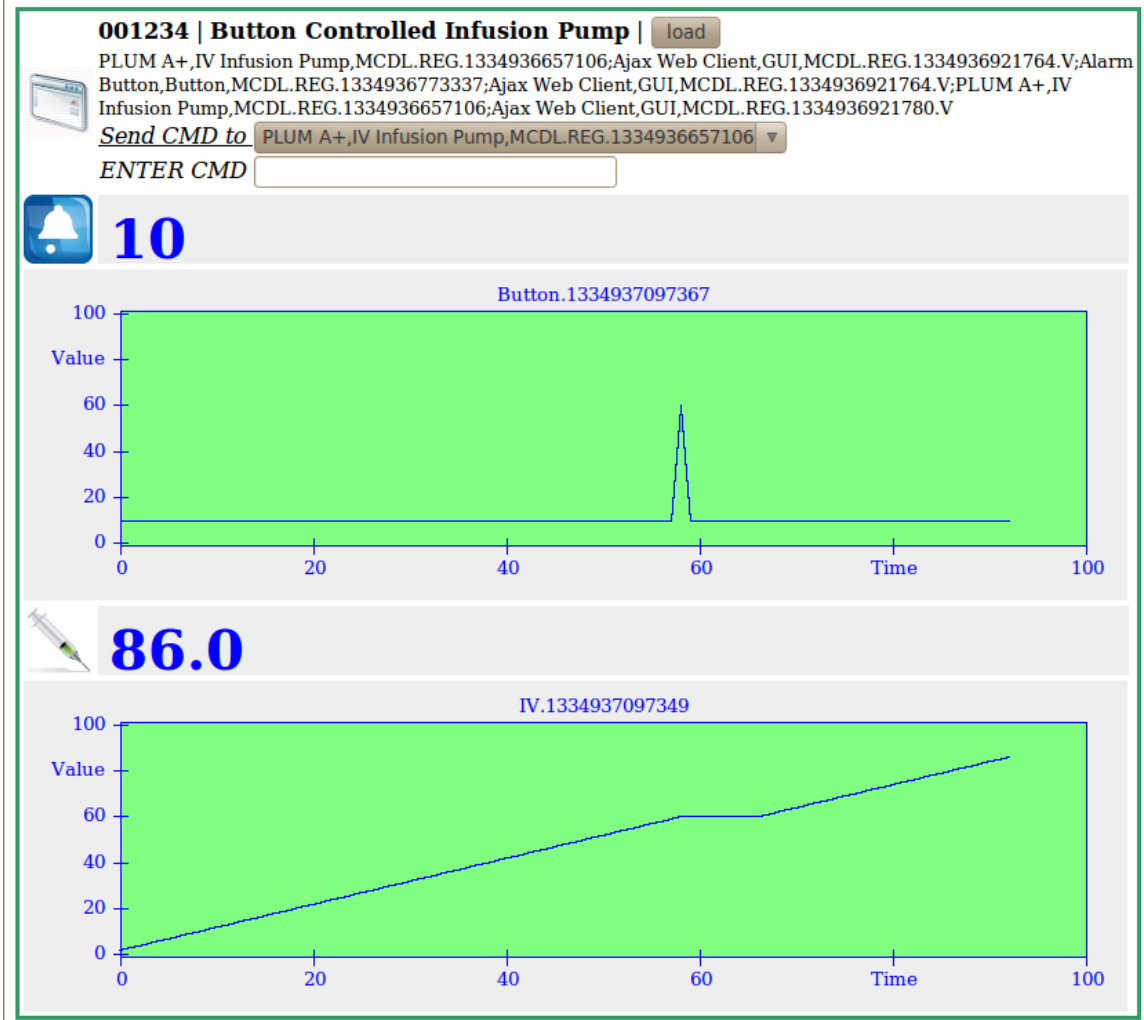


Figure 11.18. APP display for the button and infusion pump scenario in the HIIS.

Running the HIIS on a Mobile Platform

The aforementioned HIIS interface and consoles, including the scenario creation and execution web applications, are all accessible by a regular web browser with JavaScript support, which makes it possible to use HIIS on a mobile platform. For example, a mobile phone user could be granted permission to log into the HIIS system with the same account used on a computer, as shown in Figure 11.19. The interface can also stream vital sign data over the Internet in real time from any device connected to the HIIS message bus. Figure 11.20 illustrates a mobile phone receiving real-time PPGs from a patient who is wearing a pulse oximeter that feeds data to the HIIS. Since waveform drawing is a relatively challenging operation in a web

browser, the mobile platform offers great potential for the development of more features to extend the HIIS infrastructure.



Figure 11.19. Access to the HIIS with a mobile device.

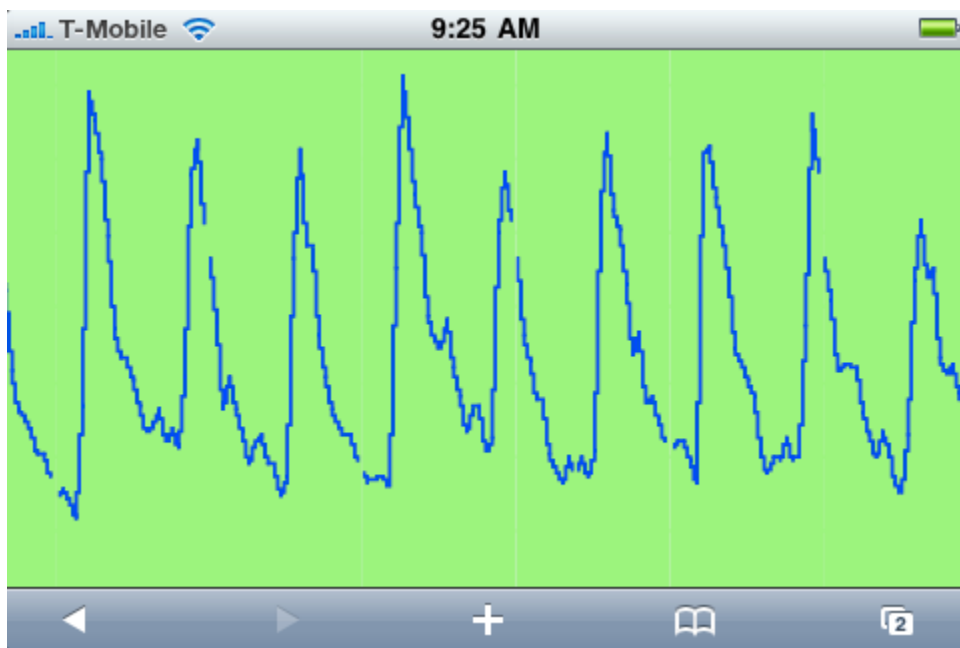


Figure 11.20. Patient scenario display on a mobile device.

Chapter 12 - Medical Platform-Oriented Device

This chapter illustrates the nature and benefits of the platform-oriented device concept by demonstrating how the MDCF can be used to realize a highly leverageable framework for Medical Platform-Oriented Devices (MPODs) [100] such as pulse oximeters that utilize photoplethysmograms (PPGs). Specifically, PPGs acquired by a wireless reflectance pulse oximeter are processed by a reconfigurable set of MDCF-based software components. In aggregate, these components support the construction of a flexible set of PPG post-processing applications that can extract a variety of meaningful clinical parameters and assess the viability of the PPGs for the intended applications. Using software engineering terminology, this work provides the foundation for a product family or software product line of MPOD pulse oximeters and other PPG-based devices, where product instances are configured according to desired features (e.g., clinical parameters and waveform processing tasks). These device instances can be used in isolation or in the context of larger multi-device APPs. As noted earlier, one benefit of this approach is that it opens up the data post-processing dimension to enable numerous instantiations that can be tailored to particular care delivery scenarios or APP contexts.

Sub-topics addressed here include the hardware features of the MPOD devices, components, and APP design in support of the desired application scenarios, and experimental results that speak to the viability of this platform-oriented approach to medical device design. Although this chapter does not address verification & validation (V&V) or regulatory issues, the work in this chapter feeds into a broader effort [101] to design V&V, third-party certification, and regulatory concepts suitable for assuring the safe and effective use of platform-oriented medical devices.

Moving from Conventional Medical Devices to MPODs

Before focusing on the instantiation of medical platform-oriented devices (MPODs) natively designed for the MDCF environment, it is helpful to first address the idea of converting a conventional medical device to an MPOD profile. As an illustration, the diagram in Figure 12.1 depicts typical elements of a stand-alone medical device (e.g., a bench top pulse oximeter). With the exception of the sensor and its support hardware (e.g., battery, control hardware, case, etc.)

(and likewise an actuator for a device such as an infusion pump), many of the functional and user-interface elements could be potentially mapped to the MDCF infrastructure, allowing for flexible APP configurations/updates and remote device control. The advantages of such an approach are clear; some of these are explained in more detail in the rest of the chapter, where pulse oximeter applications focus the discussion.

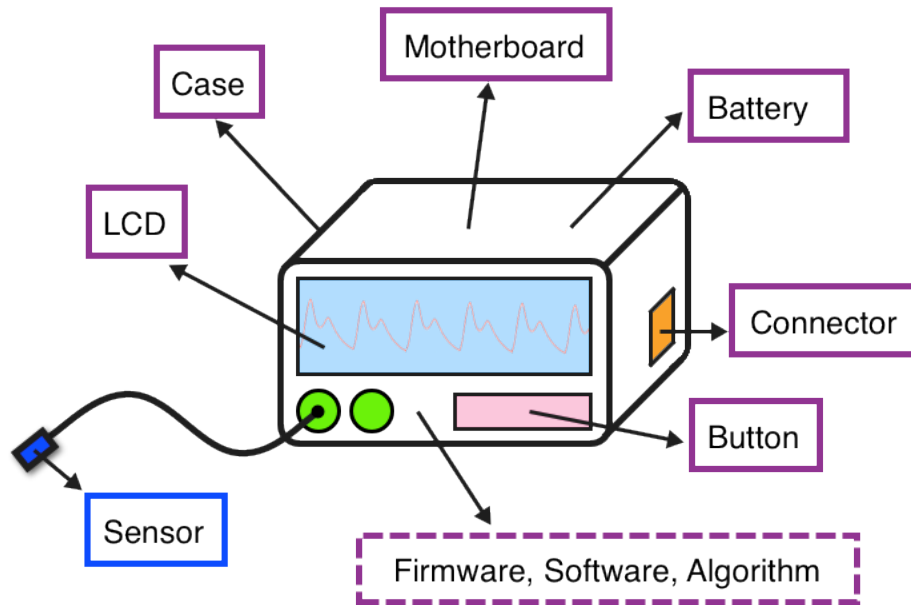


Figure 12.1. Elements of a conventional medical device.

Most commercial pulse oximeters are configured to be stand-alone devices. A typical hospital-grade pulse oximeter is comprised of a swappable sensor, an LCD screen, user interface buttons, a motherboard, etc. and is designed to report heart rate (HR) and blood oxygen saturation (SpO_2) as determined by embedded firmware. When compared to sophisticated but streamlined pulse oximeters supported by distributed frameworks or platforms that host additional processing and display elements, commercial-module limitations are clear:

1. Commercial stand-alone pulse oximeters are generally large and expensive, although some multi-parameter vital signs monitors have been designed to reduce the size and cost of device collections that incorporate pulse oximeters.

2. Pulse oximeter sensors (e.g., finger or ear clips) must support various proprietary interfaces/connectors to the host.

3. Most pulse oximeters employ circuitry and algorithms that process PPG data in unknown ways prior to parameter display.

4. On-board algorithms and firmware are relatively fixed; sometimes upgradable but not reconfigurable.

5. Internal raw sensor data are typically not exported, and exported PPGs are filtered in unknown ways. PPG waveforms are primarily used for display purposes only – not diagnostics.

In contrast, an MPOD as envisioned here emphasizes the delivery of high-quality raw data, where waveform processing, extraction of clinically meaningful parameters, and many other algorithms (e.g., for user identification) are implemented ‘off-device:’ on the computing platform. Such devices may be ‘headless’ (i.e., not include a full clinical display), because the display (e.g., customizable dashboard) is realized through the interoperability platform. Because displays and control panels may be removed from such a device, an MPOD can potentially offer a much smaller form factor, which in many cases may allow more flexible deployment of the device (e.g., a wearable device).

A physical pulse oximeter MPOD has already proved helpful when designing and evaluating the MPOD concept as well as the associated computing framework that supports reconfigurable APPs. This type of device is not an entire pulse oximeter when compared to a conventional product. Rather, it might gather data but be unable to provide clinically useful information when disconnected from the framework. The main parts of an MPOD would typically be a sensor, a signal conditioning and/or control module, a communication module, a case, simple user controls, and possibly a battery. The sensor would acquire raw data under the control of the conditioning module, and then the data would eventually be streamed to the server, either in real time or in store-and-forward mode (assuming onboard storage is available). Ideally, all sensor parts would be designed to fit within the conventional sensor form factor, e.g., a finger clip.

In the previous work, the custom reflectance pulse oximeter (see Figure 12.2 or Chapter 4) has the flavor of an MPOD. The device is lightweight, low cost, and most importantly streams four channels of raw PPG data to the receiver/host. Figure 12.2 depicts example high-quality infrared PPGs obtained at the finger and wrist (PPG data are also available at other body locations such as the palm and forehead, due to its reflectance-mode operation). These signals are acquired using a digital baseline-subtraction process that results in unfiltered, distortion-free

PPGs that offer thousands of peak-to-peak digitization levels and are sampled at 240 Hz to avoid aliasing of signal and noise components. In short, the data streamed to the MDCF are signals without circuit- and algorithm-level filtering. Hence, a variety of other information in addition to HR and SpO₂, such as motion and ambient light intensity, will also be available in these PPGs, potentially encouraging a large collection of MPOD pulse oximeter APPs.

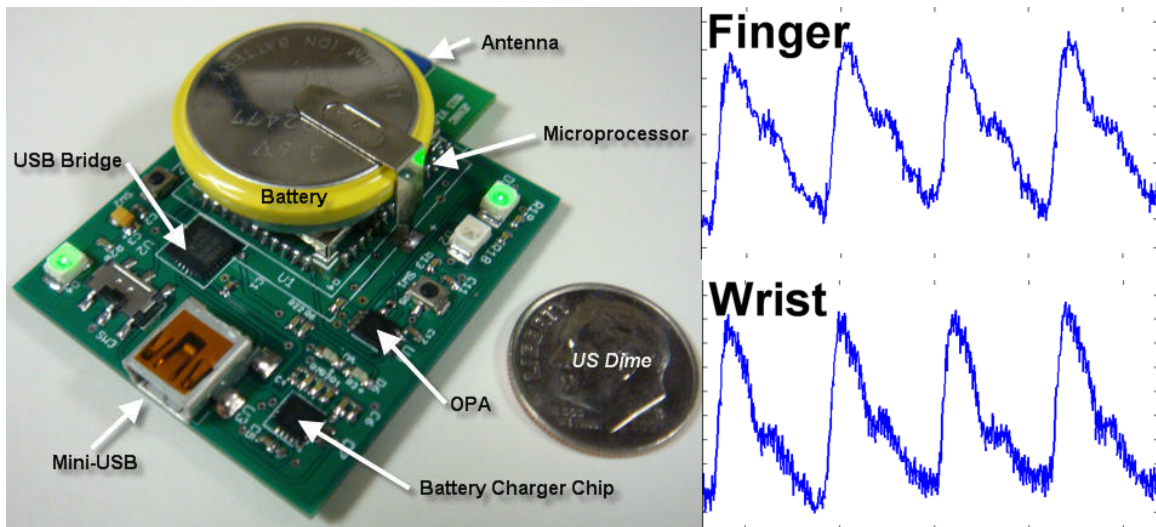


Figure 12.2. A custom reflectance pulse oximeter (left) and two representative PPGs acquired with the device (right).

Similar benefits of leveraging raw data streams could be obtained with conventional pulse oximeters if manufacturers could be persuaded to expose raw PPGs on their connectivity interfaces. Interestingly, previous MDPnP device coordination demonstrations based on the ICE encouraged ventilator vendors to add interface command functionality to make their devices more amenable to medical interoperability platform APPs [92].

Object-Oriented Model

When connected to the MDCF infrastructure, an MPOD pulse oximeter will stream measurement data through either a wired USB or wireless ZigBee connection. From the perspective of the OSI model, medical devices utilize different protocols on the transport layer, including serial, Bluetooth, Wi-Fi, and ZigBee protocols. An object called a Medical Device Port (MDP) serves as a terminal at the application layer whose subclasses correspond to the different types of communication protocols.

An MDP is an essential component in the MDCF – it represents a physical medical device. As an object class, it offers methods and properties. The methods include configure, get, set, start, pause, and stop; the properties include name, protocol type, port or channel number, buffer size, and last active time. From the perspective of the operating system, each MDP class includes device drivers for a specific communication protocol that depends on the operation system). Other MDCF objects that contribute to a medical device application are typically platform-agnostic, considering the MDCF project is written in Java. The object-oriented model in Figure 12.3 depicts the classes needed to build a medical device (application) in the MDCF.

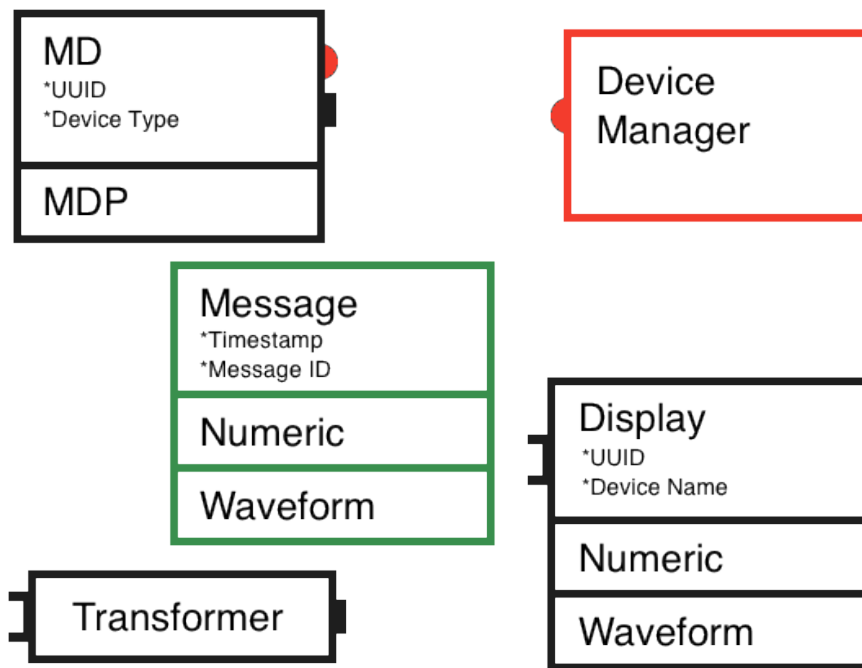


Figure 12.3. Object-oriented model for device-related MDCF components.

A Medical Device (MD) class serves as a logic device component that represents an abstract medical device in the MDCF. It could be (a) a logic wrapper around an MDP class that allows it to communicate with a real medical device or (b) just a mock device without MDP support. An MD class also offers methods and properties such as a Universally Unique Identifier (UUID) and a device type. An important idea is that methods and properties are associated with channels, as introduced earlier in Chapter 10. When an MD object is instantiated, it notifies the MDCF Device Manager via the admin channel, which is public to all newly created MDs.

Device authentication, registration, association, etc. are completed through the admin channel. A data channel is then created under the supervision of the Device Manager that allows communication with other components introduced thereafter. The information streamed over the data channel is typically raw data from the physical medical device.

The MD data sent over a data channel are first fed into an MDCF component called a Transformer class. In the context of the MDCF, a transformer is synonymous with a processor or neuron, where information is processed and repacked. For example, HR and SpO₂ parameters are extracted from the PPG data streamed from a pulse oximeter and then packed into a new packet in a transformer class. Each transformer has a specific processing capability given a particular type of input data. From a signals and systems perspective, a transform is functionally represented by a system, where input data in a pre-specified format are transformed into the desired output data. A collection of functions that can be applied to medical data are therefore mapped to a corresponding number of transformer classes, some of which (e.g., most mathematical functions) are useful for various types of medical data. This concept will be elaborated in the next section.

The medical data and/or transformed results are finally visualized by users (e.g., an ECG waveform displayed on the front screen of a conventional vital signs monitor). This functionality is realized by the MDCF Display class. Properties like a UUID or a device name are important when identifying a display for a particular application or data source. The Display class can be utilized to provide a uniform interface and data visualization approach for all medical devices integrated into the MDCF. For example, the nested classes Numeric and Waveform are two window/frame components designed to respectively display a numeric value (e.g., one HR number) and an array (a PPG waveform segment).

The final element in Figure 12.3 is the Message class, which streams data between two components, e.g., an MD and a transformer, using a standardized protocol and format. A generic Message class is defined as a superclass with properties like a timestamp (when the message was created), a message ID, a nested Numeric class, a nested Waveform class, etc. From the perspective of medical device specifications, a Message subclass could be specified for a particular medical device.

The object-oriented model brings the components of a conventional medical device to an abstract layer in the MDCF. More classes could be potentially integrated depending on the

desired functionality of the medical device components and the infrastructure that these medical devices require. E.g., a Database class would represent a storage role on either a device or an EHR in a hospital system.

APP Logic Pool

The MDCF ‘transformer’ concept is similar to the ‘function’ concept in most programming languages (e.g., C):

1. A function defines its input and output parameters and implements the transitional logic.
2. A collection of functions with related purposes forms a library, such as a math library, which contains commonly used math functions.
3. One function can be called from within another function.

In this chapter, a pulse oximeter is selected as the type of object to explain the concept of an APP logic pool (a collection of transformers) and an APP library – see the next section. PPGs acquired by the light-based sensor of a pulse oximeter potentially offer many other clinical parameters in addition to the HR and SpO₂ values reported by a conventional pulse oximeter:

- Additional cardiopulmonary and cardiovascular parameters:
- Systolic, diastolic, and mean blood pressure (BP)
- Stroke volume (SV)
- Cardiac output (CO)
- Respiration rate (RR)
- Peak-to-peak time (PPT)
- Pulse wave velocity (PWV)
- Arterial elasticity (AE)
- Stiffness index (SI)
- Reflection index (RI)
- Perfusion index (PI)
- Other parameters:
- Patient activity/motion
- Patient identity
- Ambient light information

Given the same PPG data, different parameters could be obtained by applying alternative algorithms/methods (see the colored bubbles in Figure 12.4), where each method corresponds to the functionality of a transformer. A transformer collection could form a pulse oximeter APP logic pool. Using the similar idea of “function,” several transformers might be combined in a serial cascade to form a single upper-level transformer that fits into a general device-transformer-display topology, which is called an APP (see the next section). On the other hand, a single bubble, e.g., Waveform Smoothing in Figure 12.4, could be realized in a variety of ways, including calls to other transformers that implement lower-level filtering functionality.

One potential opportunity relates to an enhanced pulse oximeter that can replace other medical devices that report physiological indices, such as a PPG-based indicator for continuous BP that provides an alternative to an unwieldy cuff-based device. These types of device substitutions could contribute to the decreased cost of medical monitoring equipment and the associated infrastructures. However, confidence in PPG-acquired parameters is still relatively low when compared to conventional methods to ascertain those parameters. Most of these PPG-based parameter extraction algorithms are still being developed and tested in laboratory settings and therefore presented as research papers; clinical validation will be required prior to broad use. A lower risk approach is to use these parameters as estimators or references for parameters acquired from dedicated medical devices. For example, when an inconsistency occurs between the BP values reported by a pulse oximeter and a BP monitor, a care provider would arrive to ensure that the BP cuff is appropriately placed, especially when the BP waveform displayed on the monitor becomes unstable or irregular. Other usage contexts include third-world areas that experience limited device availability. In such situations, an MPAD device collection built on a laptop with an accompanying MPOD PPG generator could yield a variety of useful parameters.

Parameters that are not usually reported by standard medical equipment, such as patient motion and identity, will lead to new medical device applications. For example, a pulse oximeter with user authentication functionality can help to prevent the situation where one patient’s medical data are mistakenly streamed to another patient’s record. Extra information extracted from PPG data is primarily assistive at the present time, since critical medical decisions require robust V&V.

APP Library

A collection of transformer components relevant to PPG-based APPs was illustrated in Figure 12.4, where the devices on the left provide the raw PPGs and the display on the right presents these data to the patient or clinician. In the current pulse oximeter APP library, the parameter extraction process is usually accomplished with two serialized (cascaded) transformers: (a) a PPG-processing algorithm to extract certain parameters (e.g., PI, SpO₂ Algorithm bubble in Figure 12.4) followed by (b) a parameter reporting algorithm such as a moving average computation (e.g., the SpO₂ bubble). Typically, a variety of options exists for the parameter extraction algorithm (the first transformer). Taking SpO₂ as an example, the existing methods to determine a coefficient linearly related to SpO₂ include

1. signal amplitude (valley-to-peak & baseline) extraction from the PPGs followed by a ratiometric calculation based on the results,
2. Fourier transformation of the PPGs to extract the peak of each fundamental harmonic [64], which provides equivalent amplitude information as in item 1 that can then be used in a ratiometric calculation, and
3. independent component analysis (ICA) to extract a peak-to-valley ratio directly [102], e.g., the Discrete Saturation Transform used by Masimo Corporation [103].

In commercial products, the actual implementations of these transformers are proprietary and vary across manufacturers. In our APP architecture, the first transformer could be instantiated to address a number of different methods, each of which receives PPG data and generates parameters. The second general-purpose transformer would then receive these newly calculated parameters and report a parameter that is likely further refined, e.g., with a moving average filter.

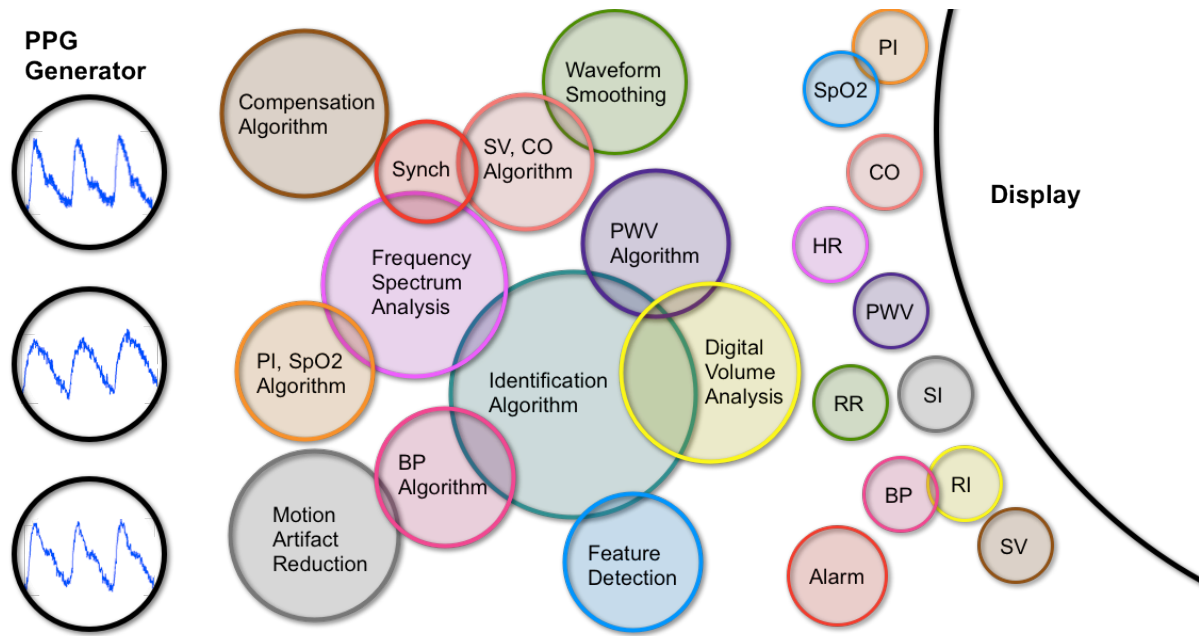


Figure 12.4. An example component pool for PPG-oriented APPs. Transformer components are depicted as colored bubbles in the middle area (interconnections will be configured for specific clinical scenarios).

Note that flavors of transformers will vary widely within such a framework. While some will, as in the prior examples, relate directly to physiological parameter extraction/processing (e.g., a motion artifact reduction transformer for a PPG signal), others will be more generic (e.g., a frequency spectrum analysis tool) and also be useful in, e.g., an ECG-based APP library. Further, transformers can exist at different levels of scope so as to allow transformers within transformers as in typical component- or object-based frameworks.

The beauty of this approach is that a considerable APP logic pool and a corresponding pool of higher-level, PPG-based APPs can be developed by a large developer community as long as the general topology of the APPs is well-defined. We are currently exploring the extent to which a rigorous architecture may allow APPs to be configured in a semi-automated manner when tied to feature models [104] that explicitly highlight the product-line nature of the APP library in terms of *variability points*. Correct composition of components into APPs is facilitated by defining a pool of message types that the transformer- and display-component inputs must support. For example, a transformer can only be allowed to interpret a particular type of message

sent from the previous component and repack the processed result into a message type required by the next component.

The following three sections present APP design examples that illustrate the reconfigurability of PPG-based APPs that employ a flexible set of data transformers.

Case 1: Parallel Processing

Figure 12.5 lays out the system-level diagram for the first PPG-based APP: a wireless reflectance pulse oximeter whose original embodiment is described in Chapter 4. Here, the physical MPOD device in Figure 12.2 performs a subset of its original operations by providing raw PPGs as input signals to the APP. As a demonstration of the feasibility of this APP-driven framework, the transformers within the APP then implement the remainder of the device functionality (processing and parameter extraction) that was formerly implemented in the device firmware.

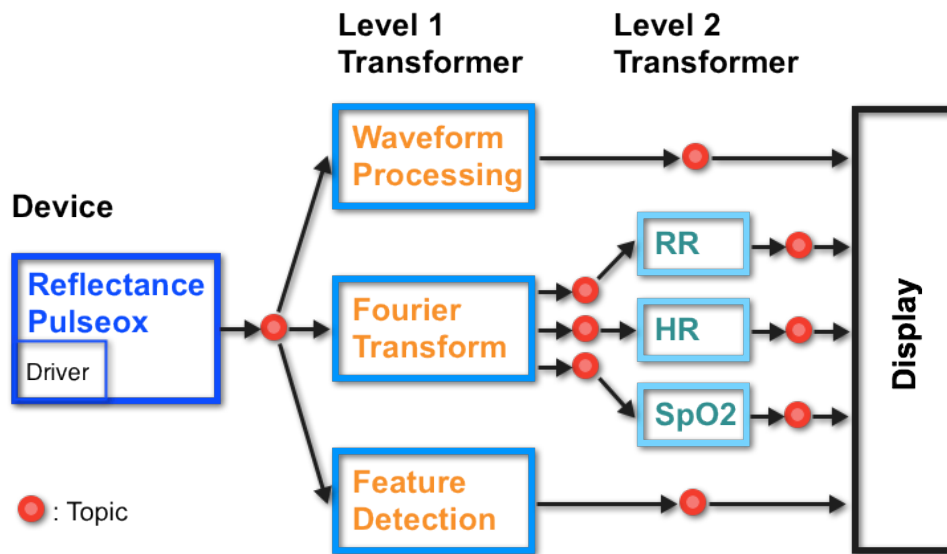


Figure 12.5. System-level diagram for a pulse oximeter APP that employs parallel transformers.

The Device block in Figure 12.5 is realized as an MDCF client running on a separate off-device computer. The client wraps a driver, where the driver links the operating-system-level driver to the physical device. Six transformers picked from the APP logic pool in Figure 12.4 are assigned at two levels. The three Level 1 transformers parallel-process the PPG waveform

data sent from the device – see Chapter 10 for more detailed descriptions of their individual functionality and realizations. The three Level 2 transformers report the physiological parameters of respiration rate, heart rate, and SpO₂ after performing, e.g., a moving average calculation on the corresponding messages from the Fourier Transform transformer. All results (signal waveforms, physiological parameters, and feature tags) are streamed to the Display (see Chapter 10).

Case 2: Multiple Devices

The second example focuses on a situation where multiple devices provide raw PPGs as APP inputs. Figure 12.6 lays out the system-level diagram for an application that extracts pulse wave velocity (PWV) given near-infrared PPGs provided by two reflectance sensors placed at the wrist and finger of the same hand [70]. Three transformers are connected in serial (cascade) to provide the PWV result. The Waveform Synchronization transformer aligns the PPGs received from the two devices using timestamps in the messages. The synchronized waveforms are then streamed to the central PWV Algorithm transformer, which internally employs a linear-phase lowpass filter and signal differentiation method as in [70]. The PWV transformer reports the final PWV value, e.g., after applying a moving average filter.

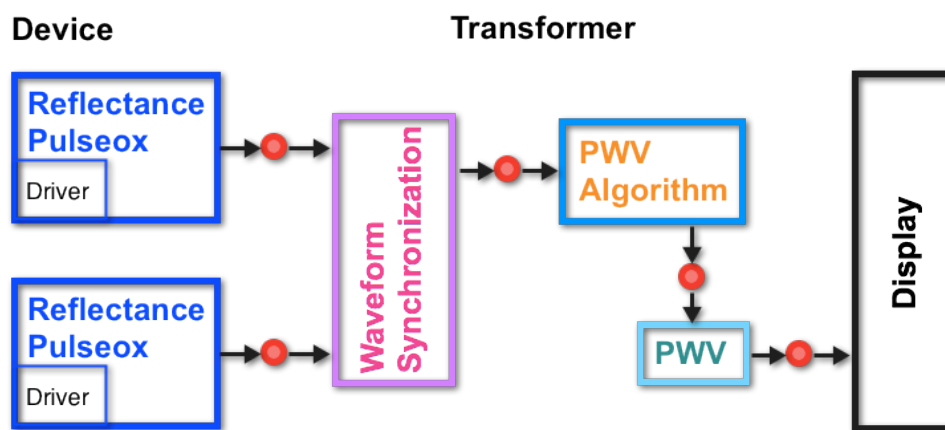


Figure 12.6. System-level diagram for a pulse wave velocity APP that employs multiple pulse oximeters (devices).

Case 3: Closed-Loop Behavior

The third example presents closed-loop APP behavior, as in Figure 12.7, where feedback channels are depicted as dashed lines. Except for the Feature Detection transformer, the other four transformers either extract or report one of two physiological parameters: BP and SV. In practice, the raw PPG data provided by the device are not always quality signals; they exhibit different levels of motion artifact and ambient noise. The resulting corrupted source data could lead to BP and SV values with little meaning, trigger nuisance alarms, and/or waste system resources. The Feature Detection transformer indicates the viability of these PPG data to the other transformers. For example, when a BP Algorithm transformer receives the detection result of “invalid PPG,” it will pause waveform processing until a valid signal appears. During the interval of the invalid signal, special-purpose error records can be forwarded to the display.

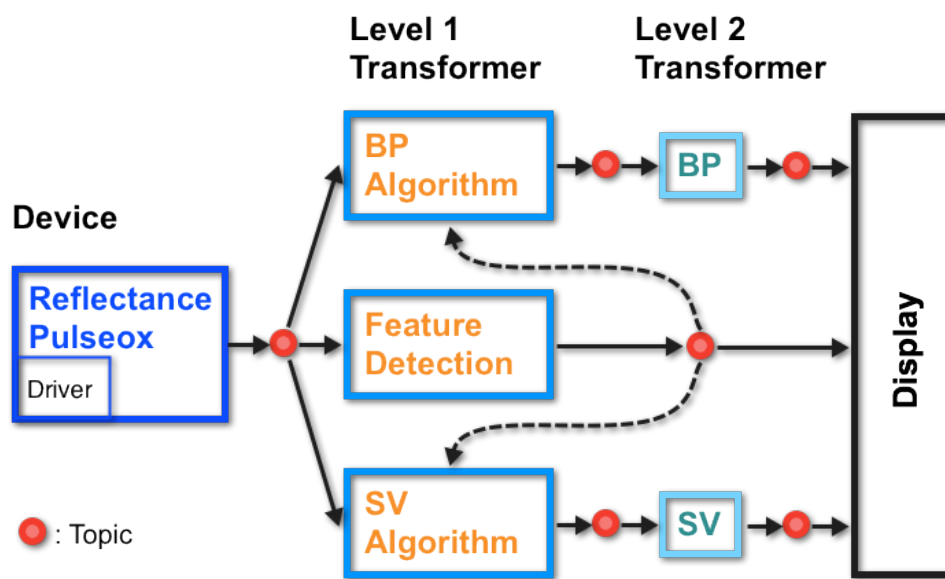


Figure 12.7. System-level diagram for a cardiovascular parameter extraction APP that exhibits closed-loop behavior.

Chapter 13 - Conclusion

This dissertation investigated three aspects of consumer biomedical devices that will be pivotal for medical systems' transitions to patient-centered architectures: (a) custom biomedical sensor designs that are optimized for home and mobile health care contexts, (b) sensor integration based on patient needs or in the home environment, and (c) medical device integration and interaction with hospital infrastructures or medical systems at different scales. The GumPack concept introduced in Chapter 2 provides an innovative hardware and software platform for research work addressing those fields. Three application chapters presented the potential use of the research results in the real world.

The concept design for a GumPack is described as a new type of everyday-carry, multi-sensor medical monitoring device that offers computer-grade processing, the ability to host multiple biomedical sensors that can be reconfigured in a plug-and-play manner based upon patient need, the means to serve as a host for wearable/nearby medical components, and integration into hospital-level medical device connectivity and coordination frameworks. The physical form factor is an innovative 3D hardware layout that does not stack circuit boards in layers like some 'adaptable' tools. Rather, its cuboid form factor provides a housing for multiple customizable surface components while still maintaining a small everyday carryable size. In its current structure, a GumPack supports four reconfigurable surfaces, where GCCF-compatible SCs can be easily added or removed. More than a sensor-laden device, a GumPack provides the computational capability and connectivity of a contemporary smart phone. The GumPack concept leapfrogs cell-phone-centered medical sensor systems that employ dumb devices and are targeted for ambulatory or wearable applications, including those intended for ubiquitous home care environments. The design not only provides a new topology between a sensor and a hub (e.g., for data aggregation, processing, and communication), but the GumPack offers a fully open development platform for hardware and software developers. Interoperability is considered and tested for the GumPack as a single conventional medical device. Since it is also a collection of local SCs and/or Chiclets as an independent system, the GCCF offers component identification, authentication, interface interpretation, and APP automation.

A high-performance wireless reflectance pulse oximeter was designed as an instantiation of the Chiclet concept for the GumPack platform, which included (a) functional features desired for research and education that are either unavailable or hard to find on commercial units and (b) design optimizations based on lessons learned from previous work or published in the pulse oximetry literature. These primary features include

- A unique filter-free circuit design,
- Full access to unfiltered PPG data,
- Many digitization levels in the pulsatile PPG signals that demonstrate a sampling frequency up to 240 Hz,
- A feedback mechanism to allow sensor operation in normal ambient room light,
- A large-area reflectance sensor that speaks to the promise of surface-infused biosensors and enables sensor placement at many body locations while optimizing the resistance of the sensor to stray photons and motion artifact,
- Onboard flash memory,
- ZigBee wireless support, and
- Mini-USB connectivity for data transfers and battery recharging.

The associated MATLAB GUI makes signal acquisition, visualization, restoration, and post-processing convenient. High-integrity PPGs acquired from 48 human subjects over a wide range of ages (20 to 64 years old) indicate the device's potential as a research and teaching platform in support of the extraction of new physiological parameters from time-domain PPGs. Finally, the size, cost, layout, and design of the sensing platform speak to its suitability for wearable applications and scenarios where medical sensors are connected to or embedded in consumer electronics such as smart phones and tablet PCs.

Onboard tagging technology was presented as a means to improve sensor intelligence by embedding information about medical device hardware (type I tag), signal samples (type II tag), and signal viability (type III tag) in the data stream itself. A use case addressing a custom pulse oximeter demonstrated that type II tags can be embedded in a data stream and then be used to calculate type III tags that are also embedded in the data stream. Further, these tags can be inserted seamlessly in the firmware flow without incurring a computation load that affects device performance. Application A addressed a means for classifying, or tagging, segments of PPG data acquired with a wireless reflectance pulse oximeter, where the presence of motion, saturation,

invalid data, or a valid PPG were of interest. To that end, a set of decision rules was established that evaluated four features related to signal behavior: baseline variation, rising count, falling count, and saturation index. The algorithm was then implemented on the portable, embedded device in a way that kept the normal pulse oximeter functionality and PPG data intact, illustrating the idea that smart tagging algorithms can be implemented onboard as feature-extraction and decision-making techniques that can improve the viability of data forwarded to electronic patient records while at the same time reducing the power needs of the portable devices. In early validation assessments, hierarchical decision rules based on training data from 47 subjects were able to achieve a decision success level of 99%.

To illustrate a sensor integration and interaction network, Chapter 8 presented an application layer scheme for WBANs with sensor nodes that compete for high-speed, real-time communication with the receiver. The lost and wandering frame issue was addressed by assigning a sequence # to each frame and a time-alignment process. The timeline distortion phenomenon was addressed by a ticket generation and consumption mechanism. A linear regression method provided the relative delay when rearranging the frames that provided information for the statistical synchronization of each sensor. As a demonstrative application for such a WBAN, PWVs were successfully extracted from the received frames. Application B presented a method to determine PWV in the hand using two reflectance-mode, wireless pulse oximeters placed at the wrist and fingertip. PWV values obtained at different features of the corresponding PPGs (foot, inflection point, and rising-slope peak) support the assertion that PWV depends on arterial pressure. PPG foot and inflection-point calculations show an increase in PWV as arterial pressure increases, but peak-based calculations are inconsistent with expectations, primarily because the effects of wave reflections on PPG shape are not well modeled and warrant further investigation. Nonetheless, the two-oximeter approach was shown to be feasible, so the next logical step is to compare these PWVs with values obtained from commercial devices employing other modalities. In other research areas such as digital volume pulse (DVP) analysis [47], the estimated PPGs would help to improve precision, especially in a WBAN scenario.

This dissertation also illustrated how medical device integration platforms such as the MDCF and the HIIS can allow developers to build new “platform-based” medical devices that are customized to different usage contexts. The growing trend toward networked clinical

systems (evidenced by the increased use of MDDSs and integrated EHRs) suggests a possible evolution of medical devices toward the MPOD concept – devices that are oriented towards being used primarily with an integration platform. The APP component library concept and lightweight MPOD for PPG processing has significant utility in its own right. We are working to build out the component library to support additional PPG examples. For the future integration of additional medical devices that require signal processing, the evaluation of a large pool of published signal processing modules would be worthwhile, as they would serve as a general transformer development library. A significant challenge is developing V&V and regulatory guidelines that will allow these types of frameworks to be brought to market. To this end, local team members are involved in collaborative work with MDPnP, several other universities, device and clinical information system manufacturers, certification labs, and government agencies such as the FDA and NIST. A preliminary vision for a component-based certification approach is presented in [101], and the examples and broader APP framework described in this dissertation will provide additional challenge problems to drive that effort.

References

- [1] R. Abelson, "Study Shows Health Insurance Costs Rising Sharply This Year," *The New York Times*, Sept. 27, 2011.
- [2] AFP, "U.S. Braces for Baby Boom Retirement Wave," Dec. 24, 2007, <http://afp.google.com/article/ALeqM5g-W4yeMsbNpknzoffLTFxOgKI2A>.
- [3] Chris Fleming, "Health Care Workforce: Nurse And Physician Shortages, Health Affairs Blog," June 14th, 2010, <http://healthaffairs.org/blog/2010/06/14/health-care-workforce-nurse-and-physician-shortages/>.
- [4] Walt Zywiak, "U.S. Health Care Workforce Shortages: HIT Staff," 2011, http://www.himss.org/Content/Files/CSC_US_Healthcare_Workforce_Shortages_HIT.pdf.
- [5] AliveCor, "AliveCor iPhone ECG," 2012, <http://alivecor.com/>.
- [6] Imec, "Monitoring Your Health with Your Mobile Phone," 2010, <http://www2.imec.be/>.
- [7] E. Jonathan and M. J. Leahy, "Cellular Phone-Based Photoplethysmographic Imaging," *Journal of Biophotonics*, vol. 4, pp. 293-296, May 2011.
- [8] C. G. Scully, J. Lee, J. Meyer, A. M. Gorbach, D. Granquist-Fraser, Y. Mendelson, and K. H. Chon, "Physiological Parameter Monitoring from Optical Recordings With a Mobile Phone," *IEEE Transactions on Biomedical Engineering*, vol. 59, pp. 303-306, Feb. 2012.
- [9] Apple Inc., "App Store," 2012, <http://itunes.apple.com/>.
- [10] John L. Smith, "The Pursuit of Noninvasive Glucose," 2011, http://www.mendosa.com/noninvasive_glucose.pdf.
- [11] Christian Nordqvist, "Using Tears To Monitor Diabetes Blood Sugar Levels," Nov 12, 2011, <http://www.medicalnewstoday.com/articles/237537.php>.
- [12] K. Li and S. Warren, "A Wireless Reflectance Pulse Oximeter with Digital Baseline Control for Unfiltered Photoplethysmograms," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, pp. 269-278, June 2012.
- [13] Wikipedia, "Vital Signs," 2012, http://en.wikipedia.org/wiki/Vital_signs.
- [14] S. Rhee, B. H. Yang, and H. Asada, "Artifact-Resistant, Power-Efficient Design of Finger-Ring Plethysmographic Sensors," *IEEE Transactions on Biomedical Engineering*, vol. 48, pp. 795-805, July 2001.

- [15] H. Asada, P. Shaltis, A. Reisner, S. Rhee, and R. Hutchinson, "Mobile Monitoring with Wearable Photoplethysmographic Biosensors," *IEEE Engineering in Medicine and Biology Magazine*, vol. 22, pp. 28-40, 2003.
- [16] S. B. Duun, R. G. Haahr, K. Birkelund, and E. V. Thomsen, "A Ring-Shaped Photodiode Designed for Use in a Reflectance Pulse Oximetry Sensor in Wireless Health Monitoring Applications," *IEEE Sensors Journal*, vol. 10, pp. 261-268, 2010.
- [17] P. Branche and Y. Mendelson, "Signal Quality and Power Consumption of a New Prototype Reflectance Pulse Oximeter Sensor," *31st Annual Northeast Conference of the IEEE Bioengineering*, 2-3 April 2005, 2005, pp. 42-43.
- [18] Y. Mendelson and C. Pujary, "Measurement Site and Photodetector Size Considerations in Optimizing Power Consumption of a Wearable Reflectance Pulse Oximeter," *25th Annual Conference of the IEEE EMBS*, 2003, pp. 3016-3019.
- [19] M. Nogawa, T. Kaiwa, and S. Takatani, "A Novel Hybrid Reflectance Pulse Oximeter Sensor with Improved Linearity and General Applicability to Various Portions of the Body," *20th Annual International Conference of the IEEE EMBS*, 29 Oct-1 Nov, 1998, pp. 1858-1861.
- [20] D. Guowei, T. Xiaoying, and L. Weifeng, "A Reflectance Pulse Oximeter Design Using the MSP430F149," *2007 IEEE International Conference on Complex Medical Engineering*, 23-27 May, 2007, pp. 1081-1084.
- [21] J. Furey, "Single-Use Noninvasive Sensing Technology," *Genetic Engineering and Biotechnology News*, vol. 29, Aug. 1 2009.
- [22] R. G. Haahr, S. B. Duun, M. H. Toft, B. Belhage, J. Larsen, K. Birkelund, and E. V. Thomsen, "An Electronic Patch for Wearable Health Monitoring by Reflectance Pulse Oximetry," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, pp. 45-53, 2012.
- [23] M. Tavakoli, L. Turicchia, and R. Sarpeshkar, "An Ultra-Low-Power Pulse Oximeter Implemented With an Energy-Efficient Transimpedance Amplifier," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 4, pp. 27-38, 2010.
- [24] T. O'Donovan, J. O'Donoghue, C. Sreenan, D. Sammon, P. O'Reilly, and K. A. O'Connor, "A Context Aware Wireless Body Area Network (BAN)," *3rd International Conference on Pervasive Computing Technologies for Healthcare*, London, April 1-3, 2009, pp. 1-8.
- [25] E. Jovanov, A. Milenkovic, C. Otto, and P. Groen, "A Wireless Body Area Network of Intelligent Motion Sensors for Computer Assisted Physical Rehabilitation," *Journal of Neuro Engineering and Rehabilitation*, vol. 2, 2005.
- [26] H. Chen, W. Wu, and J. Lee, "A WBAN-Based Real-Time Electroencephalogram Monitoring System: Design and Implementation," *Journal of Medical Systems*, March 2009.

- [27] T. Gao, D. Greenspan, M. Welsh, and R. R. Juang, "Vital Signs Monitoring and Patient Tracking over a Wireless Network," *27th Annual International Conference of the Engineering in Medicine and Biology*, September, 2005, pp. 102-105.
- [28] J. Espina, T. Falck, J. Muehlsteff, Y. Jin, M. A. Adán, and X. Aubert, "Wearable Body Sensor Network towards Continuous Cuff-less Blood Pressure Monitoring," *5th International Workshop on Wearable and Implantable Body Sensor Networks*, June, 2008, pp. 28-32.
- [29] Kansas State University, "Medical Device Coordination Framework (MDCF)," 2012, <http://mdcf.santos.cis.ksu.edu>.
- [30] Steven Dean, "eDevice HealthGO Platform Gets Integrated Connectivity with Freescale Home Health Hub Reference Platform," 2012, <http://blogs.freescale.com/2012/05/07/edevice-healthgo-platform-integrated-connectivity-freescale-home-health-hub-reference-platform/>.
- [31] Qualcomm Life, "Introducing the 2net™ Platform," 2012, <http://www.qualcomm.life.com/wireless-health>.
- [32] "IEEE 802.15 WPAN Task Group 6 Body Area Networks," 2012, <http://www.ieee802.org/15/pub/TG6.html>.
- [33] M. Chen, S. Gonzalez, A. Vasilakos, H. Cao, and V. C. M. Leung, "Body Area Networks: A Survey," *Mobile Networks and Applications*, vol. 16, pp. 171-193, 2010.
- [34] A. King, S. Proctor, D. Andresen, J. Hatcliff, S. Warren, W. Spees, R. Jetley, P. Jones, and S. Weininger, "An Open Test Bed for Medical Device Integration and Coordination," *ICSE*, Vancouver, Canada, May 16-24, 2009.
- [35] "Essential Safety Requirements for Equipment Comprising the Patient-Centric Integrated Clinical Environment (ICE) – Part 1: General Requirements and Conceptual Model " 2009, <http://mdpnp.org/ICE.html>.
- [36] S. Warren and B. Natarajan, "Wireless Communication Technologies for Wearable Systems," in *Wearable Monitoring Systems*, 1st ed, A. Bonfiglio and D. D. Rossi, Eds. New York: Springer, 2010, pp. 51-80.
- [37] J. Yao, R. Schmitz, and S. Warren, "A Wearable Point-of-Care System for Home Use that Incorporates Plug-and-Play and Wireless Standards," *IEEE Transactions on Information Technology in Biomedicine*, vol. 9, pp. 363-371, Sept. 2005.
- [38] "ISO/IEEE 11073 Personal Health Data (PHD) Standards," 2011, <http://www.11073.org/>.
- [39] Gumstix, "Gumstix Overo Fire COM," 2012, <http://www.gumstix.com/>.
- [40] C. V. C. Bouten, K. T. M. Koekkoek, M. Verduin, R. Kodde, and J. D. Janssen, "A Triaxial Accelerometer and Portable Data Processing Unit for the Assessment of Daily

- Physical activity," *IEEE Transactions on Biomedical Engineering*, vol. 44, pp. 136-147, 1997.
- [41] D. A. Tong, K. A. Bartels, and K. S. Honeyager, "Adaptive Reduction of Motion Artifact in the Electrocardiogram," *24th Annual Conference of the IEEE EMBS*, Houston, TX, Oct. 23-25, 2002, pp. 1403-1404.
- [42] H. Han, M.-J. Kim, and J. Kim, "Development of Real-Time Motion Artifact Reduction Algorithm for a Wearable Photoplethysmography," *29th Annual Conference of the IEEE EMBS*, 2007, pp. 1538-1541.
- [43] "IEEE Draft Standard for Health informatics - Personal health device communication - Device specialization - Basic Electrocardiograph (ECG) (1 to 3-lead ECG)," *IEEE P11073-10406/D07*, February 2011, pp. 1-69, 2011.
- [44] Texas Instruments, "INA321 Datasheet," 2011, <http://focus.ti.com/lit/ds/symlink/ina321.pdf>.
- [45] Texas Instruments, "Medical Application Guide," 2011, <http://focus.ti.com/>.
- [46] American Heart Association, "Heart Disease and Stroke Statistics - 2009 Update At-a-Glance," 2009, <http://www.americanheart.org/>.
- [47] S. R. Alty, N. Angarita-Jaimes, S. C. Millasseau, and P. J. Chowienczyk, "Predicting Arterial Stiffness From the Digital Volume Pulse Waveform," *IEEE Transactions on Biomedical Engineering*, vol. 54, pp. 2268-2275, Dec. 2007.
- [48] K. N. Glaros and E. M. Drakakis, "Trade-offs for Low Power Integrated Pulse Oximeters," *IEEE Biomedical Circuits and Systems Conference*, Nov. 26-28, 2009, pp. 245-248.
- [49] A. Reisner, P. A. Shaltis, D. McCombie, and H. H. Asada, "Utility of the Photoplethysmogram in Circulatory Monitoring," *Anesthesiology*, vol. 108, pp. 950-958, May 2008.
- [50] W. B. Murray and P. A. Foster, "The Peripheral Pulse Wave: Information Overlooked," *Journal of Clinical Monitoring*, vol. 12, pp. 365-377, Sept. 1996.
- [51] K. H. Shelley, "Photoplethysmography: Beyond the Calculation of Arterial Oxygen Saturation and Heart Rate," *Anesthesia & Analgesia*, vol. 105, pp. S31-S36, Dec. 2007.
- [52] S. C. Millasseau, F. G. Guigui, R. P. Kelly, K. Prasad, J. R. Cockcroft, J. M. Ritter, and P. J. Chowienczyk, "Noninvasive Assessment of the Digital Volume Pulse : Comparison with the Peripheral Pressure Pulse," *Hypertension*, vol. 36, pp. 952-956, 2000.
- [53] J. Allen and A. Murray, "Modelling the Relationship Between Peripheral Blood Pressure and Blood Volume Pulses Using Linear and Neural Network System Identification Techniques," *Physiological Measurement*, vol. 20, pp. 287-301, 1999.

- [54] K. H. Shelley, W. B. Murray, and D. Chang, "Arterial-Pulse Oximetry Loops: A New Method of Monitoring Vascular Tone," *Journal of Clinical Monitoring*, vol. 13, pp. 223-228, 1997.
- [55] N. Westerhof, N. Stergiopoulos, and M. I. M. Noble, *Snapshots of Hemodynamics*, 1st ed. Boston: Springer, 2004.
- [56] S. C. Millasseau, R. P. Kelly, J. M. Ritter, and P. J. Chowienczyk, "Determination of Age-Related Increases in Large Artery Stiffness by Digital Pulse Contour Analysis," *Clinical Science (London)*, vol. 103, pp. 371–377, Oct. 2002.
- [57] J. Li, L. Yang, S. Zhang, and Y. Yang, "Computation of Cardiac Output by Pulse Wave Contour," *1st International Conference on Bioinformatics and Biomedical Engineering*, Wuhan, 2007, pp. 1088-1090.
- [58] P. J. Chowienczyk, R. P. Kelly, H. MacCallum, S. C. Millasseau, T. L. G. Andersson, R. G. Gosling, J. M. Ritter, and E. E. Änggård, "Photoplethysmographic Assessment of Pulse Wave Reflection: Blunted Response to Endothelium-Dependent Beta2-Adrenergic Vasodilation in Type II Diabetes Mellitus," *Journal of the American College of Cardiology*, vol. 34, pp. 2007–2014, 1999.
- [59] K. Li, "Wireless Reflectance Pulse Oximeter Design and Photoplethysmographic Signal Processing," M.S. Thesis, Kansas State University, Manhattan, KS, 2010.
- [60] K. Nakajima, T. Tamura, and H. Miike, "Monitoring of Heart and Respiratory Rates by Photoplethysmography Using a Digital Filtering Technique," *Medical Engineering & Physics*, vol. 18, pp. 365-372, July 1996.
- [61] 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," *Journal of Clinical Monitoring and Computing*, vol. 20, pp. 33-36, 2006.
- [62] L. C. Ludeman and M. I. Chacon, "Use of Blood Pulse Signature for Identity Verification," *7th International Conference on Signal Processing Applications and Technology*, Boston, Massachusetts, October 7-10, 1996, pp. 1608-1612.
- [63] L. C. Ludeman and M. I. Chacon, "Evaluation of Blood Pulse Signature for Potential Application in a Multisensor Biometric Identity Verification System," New Mexico State University, Final report to Sandia National Laboratories under contract AM-5506, Sept. 1995.
- [64] J. T. Love, S. Warren, G. R. Laguna, and T. J. Miller, "Personal Status Monitor," Sandia National Laboratories SAND97-0418, UC-706, unlimited release, Feb. 1997.
- [65] M. Locke, "Android and RTOS Together: The Dynamic Duo for Today's Medical Devices," *Embedded Computing Design*, pp. 36-38, April 2010.

- [66] D. Thompson and S. Warren, "A Small, High-Fidelity Reflectance Pulse Oximeter," *2007 Annual Conference and Exposition of the ASEE*, Honolulu, HI, June 24-27, 2007.
- [67] K. Li, S. Warren, and B. Natarajan, "Onboard Tagging for Real-Time Quality Assessment of Photoplethysmograms Acquired by a Wireless Reflectance Pulse Oximeter," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, pp. 54-63, February 2012.
- [68] R. Krishnan, B. Natarajan, and S. Warren, "Two-Stage Approach for Detection and Reduction of Motion Artifacts in Photoplethysmographic Data," *IEEE Transactions on Biomedical Engineering*, vol. 57, pp. 1867-1876, 2010.
- [69] E. Geun, H. Heo, K. C. Nam, and Y. Huh, "Measurement Site and Applied Pressure Consideration in Wrist Photoplethysmography," *23rd ITC-CSCC*, 2008, pp. 1129-1132.
- [70] K. Li and S. Warren, "Initial Study on Pulse Wave Velocity Acquired from One Hand Using Two Synchronized Wireless Reflectance Pulse Oximeters," *33rd Annual International Conference of the IEEE EMBS*, Boston, MA, Aug. 30-Sept. 3, 2011, pp. 6907-6910.
- [71] U.S. Food and Drug Administration (FDA), "Premarket Notification (510k)," 2011, <http://www.fda.gov/MedicalDevices>.
- [72] K. A. Reddy, B. George, and V. J. Kumar, "Use of Fourier Series Analysis for Motion Artifact Reduction and Data Compression of Photoplethysmographic Signals," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, pp. 1706-1711, May 2009.
- [73] B. S. Kim and S. K. Yoo, "Motion Artifact Reduction in Photoplethysmography Using Independent Component Analysis," *IEEE Transactions on Biomedical Engineering*, vol. 53, pp. 566-568, March 2006.
- [74] J. Y. A. Foo, "Comparison of Wavelet Transformation and Adaptive Filtering in Restoring Artefact-Induced Time-Related Measurement," *Biomedical Signal Processing and Control*, vol. 1, pp. 93-98, Jan. 2006.
- [75] K. W. Chan and Y. T. Zhang, "Adaptive Reduction of Motion Artifact from Photoplethysmographic Recordings Using a Variable Step-Size LMS Filter," *2002 IEEE Proceedings on Sensors*, 2002, pp. 1343-1346.
- [76] J. Yao and S. Warren, "A Short Study to Assess the Potential of Independent Component Analysis for Motion Artifact Separation in Wearable Pulse Oximeter Signals," *27th Annual Conference of the IEEE EMBS*, 2005, pp. 3585-3588.
- [77] J. G. Webster, *Design of Pulse Oximeters*, 1st ed. New York: Taylor & Francis, 1997.
- [78] H. W. Lee, J. W. Lee, W. G. Jung, and G. K. Lee, "The Periodic Moving Average Filter for Removing Motion Artifacts from PPG Signals," *International Journal of Control, Automation, and Systems*, vol. 5, pp. 701-706, Dec. 2007.

- [79] H. V. Poor, *An Introduction to Signal Detection and Estimation*, 2nd ed. New York: Springer-Verlag, 1994.
- [80] R. Krishnan, B. Natarajan, and S. Warren, "Analysis and Detection of Motion Artifact in Photoplethysmographic Data Using Higher Order Statistics," *2008 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Las Vegas, Nevada, March 31-April 4, 2008, pp. 613-616.
- [81] G. J. Miao and M. A. Clements, *Digital Signal Processing and Statistical Classification*, 1st ed. Norwood: Artech House, 2002.
- [82] C. C. Lee and J. J. Chao, "Optimum Local Decision Space Partitioning for Distributed Detection," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 25, pp. 536-544, 1989.
- [83] K. K. Tremper, "Pulse Oximetry," *Chest*, vol. 95, pp. 713-715, April 1989.
- [84] J. Sol'a, O. Ch'etelat, C. Sartori, Y. Allemann, and S. F. Rimoldi, "Chest Pulse-Wave Velocity: A Novel Approach to Assess Arterial Stiffness," *IEEE Transactions on Biomedical Engineering*, vol. 58, Jan. 2011.
- [85] H.-T. Wu, C.-S. Ho, J.-S. Weng, W.-C. Tsai, and M.-C. Wang, "A Novel Method for Measurement of Pulse Wave Velocity by Dual-Channel Photoplethysmography," *2004 IEEE International Workshop on Biomedical Circuits & Systems*, 2004, pp. S2/6-8-10.
- [86] K. Li and S. Warren, "A High-Performance Wireless Reflectance Pulse Oximeter for Photo-Plethysmogram Acquisition and Analysis in the Classroom," *2010 Annual Conference and Exposition of the ASEE*, Louisville, KY, June 20-23, 2010.
- [87] N. Chubachit, H. Kanait, R. Muratat, and Y. Koiwa, "Measurement of Local Pulse Wave Velocity in Arteriosclerosis by Ultrasonic Doppler Method," *1994 IEEE Proceedings on Ultrasonics Symposium*, Cannes, France, Nov. 1-4, 1994, pp. 1747-1750.
- [88] A. P. Spence, *Basic Human Anatomy*, 1st ed. Menlo Park: The Benjamin/Cummings Publishing Company, Inc., 1982.
- [89] K. Li and S. Warren, "High Resolution Wireless Body Area Network with Statistically Synchronized Sensor Data for Tracking Pulse Wave Velocity," *34th Annual International Conference of the IEEE EMBS*, San Diego, CA, Aug. 28-Sept. 1, 2012, 2012.
- [90] A. King, D. Arney, I. Lee, O. Sokolsky, J. Hatcliff, and S. Procter, "Prototyping Closed Loop Physiologic Control with the Medical Device Coordination Framework," *SEHC*, Cape Town, South Africa, May 3-4, 2010.
- [91] "Medical Device 'Plug-and-Play' Interoperability Program," 2012, <http://mdpnp.org/>.

- [92] D. Arney, S. Fischmeister, J. M. Goldman, I. Lee, and R. Trausmuth, "Plug-and-Play for Medical Devices: Experiences from a Case Study," *Biomedical Instrumentation & Technology*, vol. 43, pp. 313-317, July-August 2009.
- [93] OpenJMS, "Java Message Service API 1.1 Specification," 2012, <http://openjms.sourceforge.net/>.
- [94] "RXTX Library," 2011, <http://rxtx.qbang.org/>.
- [95] "XStream Library," 2011, <http://xstream.codehaus.org/>.
- [96] Wikipedia, "AJAX (Programming)," 2012, <http://en.wikipedia.org/wiki/AJAX>.
- [97] Wikipedia, "Java Servlet," 2012, http://en.wikipedia.org/wiki/Java_Servlet.
- [98] Apache, "ActiveMQ," 2012, <http://activemq.apache.org/>.
- [99] Apache, "Tomcat," 2012, <http://tomcat.apache.org/>.
- [100] K. Li, S. Warren, and J. Hatcliff, "Component-Based App Design for Platform-Oriented Devices in a Medical Device Coordination Framework," *2nd ACM SIGHT IHI Symposium*, Miami, FL, Jan. 28-30, 2012, pp. 343-352.
- [101] J. Hatcliff, E. Vasserman, S. Weinenger, and J. Goldman, "An Overview of Regulatory and Trust Issues for the Integrated Clinical Environment," *Proceedings of the 2011 High-confidence Medical Device Systems and Software Workshop*, Chicago, IL, April, 2011.
- [102] T. Jensen, S. Duun, J. Larsen, R. G. Haahr, M. H. Toft, B. Belhage, and E. V. Thomsen, "Independent Component Analysis Applied to Pulse Oximetry in the Estimation of the Arterial Oxygen Saturation (SpO₂) - a Comparative Study," *31st Annual International Conference of the IEEE EMBS*, Minneapolis, Minnesota, Sept. 2-6, 2009, pp. 4039-4044.
- [103] J. M. Goldman, M. T. Petterson, R. J. Kopotic, and S. J. Barker, "Masimo Signal Extraction Pulse Oximetry," *Journal of Clinical Monitoring and Computing*, vol. 16, pp. 475-483, 2000.
- [104] K. Czarnecki, S. Helson, and U. Eisenecker, "Formalizing Cardinality-Based Feature Models and Their Specialization," *Software Process : Improvement and Practice*, vol. 10, pp. 7-29, 2005.
- [105] Gumstix, "Setting up a Serial Connection," 2012, <http://gumstix.org/connect-to-my-gumstix-system.html>.

Appendix A – GumPack Hardware Interconnection Interface

Table A.1. GumPack Hardware Interface Pin Definition

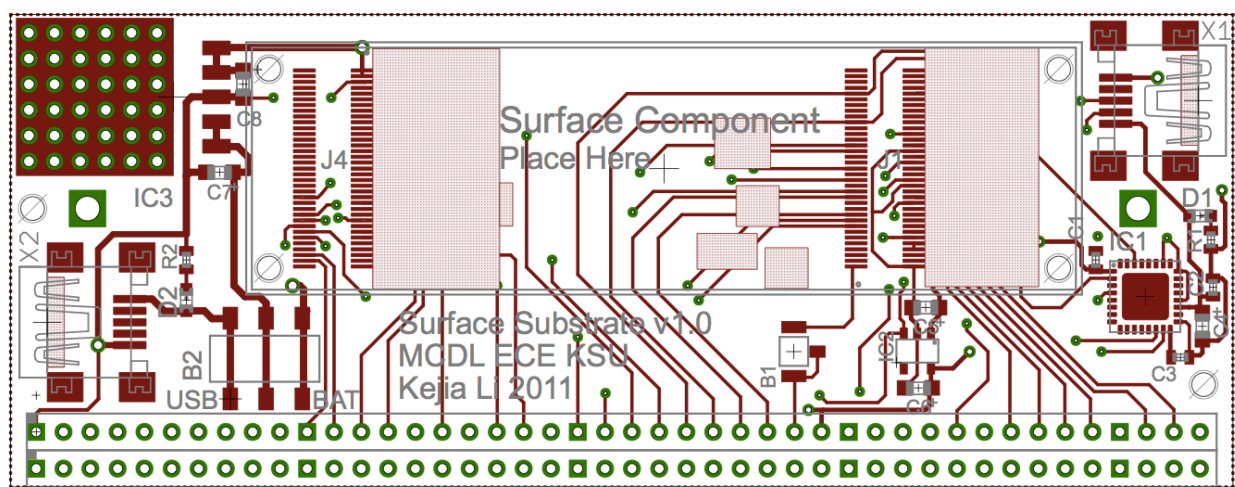
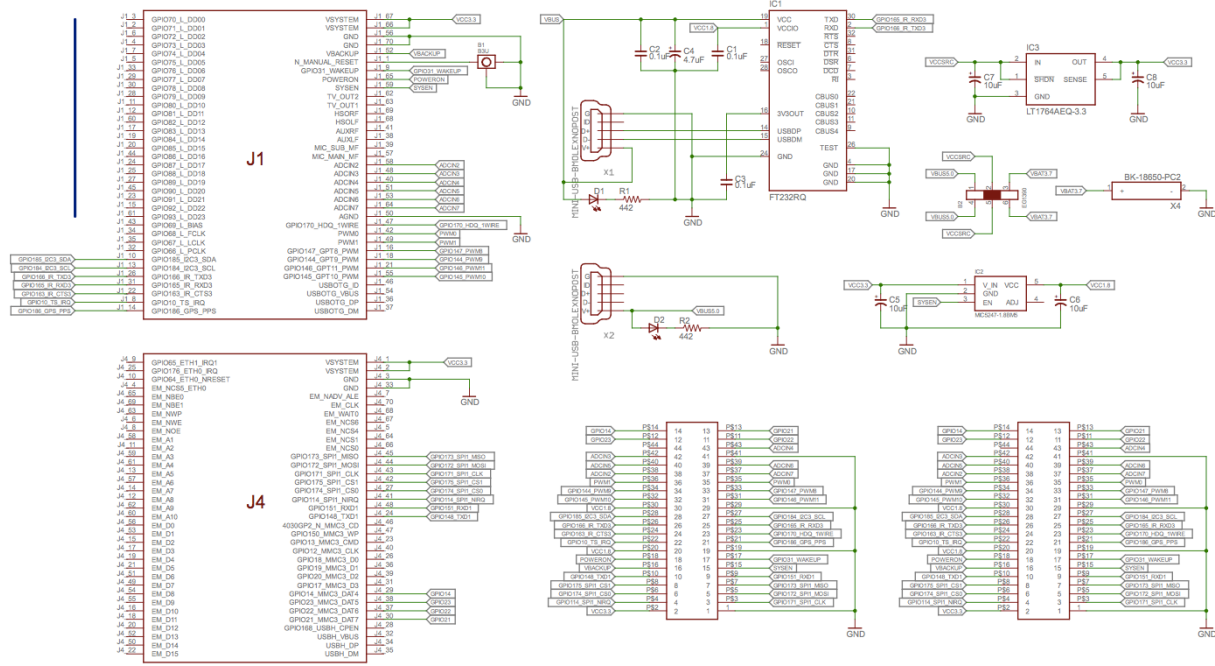
Pin #	Function	Motion Sensor	ECG	ZigBee Coord.
1	GND			
2	VCC3.3			
3	GPIO171_SPI1_CLK			
4	GPIO114_SPI1_NIRQ			
5	GPIO172_SPI1_MOSI			
6	GPIO174_SPI1_CS0			
7	GPIO173_SPI1_MISO			
8	GPIO175_SPI1_CS1			
9	GPIO151_RXD1			
10	GPIO148_TXD1			
11	GPIO22			
12	GPIO23			
13	GPIO21			
14	GPIO14			
15	SYSEN			
16	VBACKUP			
17	GPIO31_WAKEUP			
18	POWERON			
19	GND			
20	VCC1.8			
21	GPIO186_GPS_PPS			
22	GPIO10_TS_IRQ			
23	GPIO170_HDQ_1WIRE			
24	GPIO163_IR_CTS3			
25	GPIO165_IR_RXD3			
26	GPIO166_IR_TXD3			

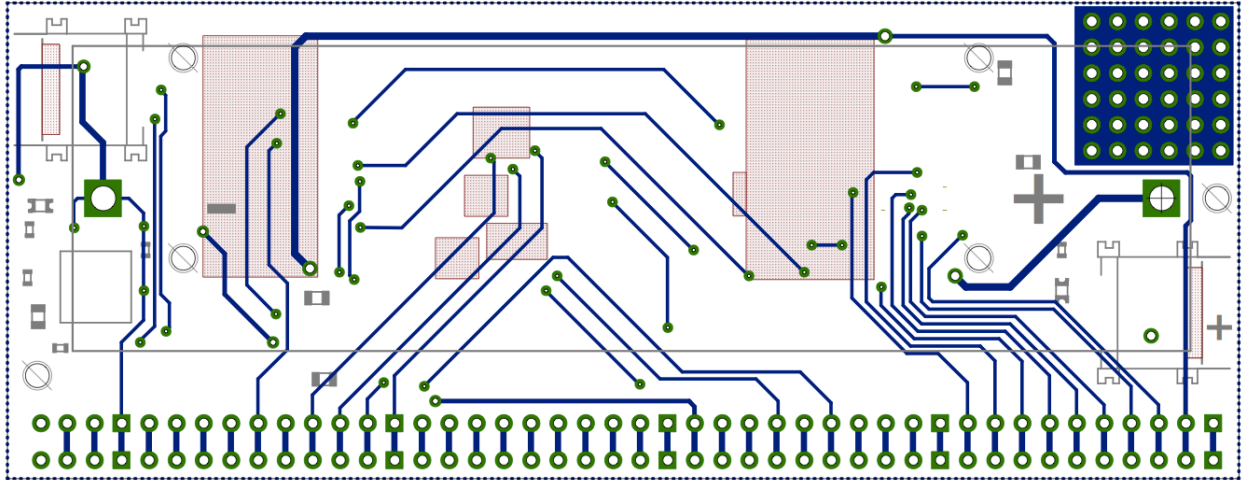
Pin #	Function	Motion Sensor	ECG	ZigBee Coord.
27	GPIO184_I2C3_SCL			
28	GPIO185_I2C3_SDA			
29	GND			
30	VCC1.8			
31	GPIO146_PWM11			
32	GPIO145_PWM10			
33	GPIO147_PWM8			
34	GPIO144_PWM9			
35	PWM0			
36	PWM1			
37	ADCIN7			
38	ADCIN2			
39	ADCIN6			
40	ADCIN5			
41	GND			
42	ADCIN3			
43	ADCIN4			
44	NC			

Pin utilization for the three example SC boards is indicated by color-filled cells.

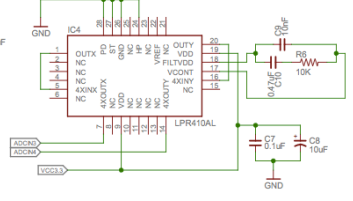
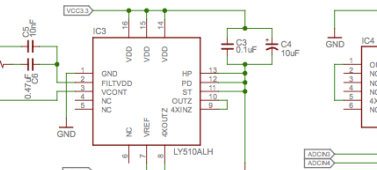
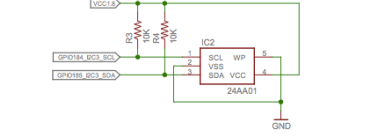
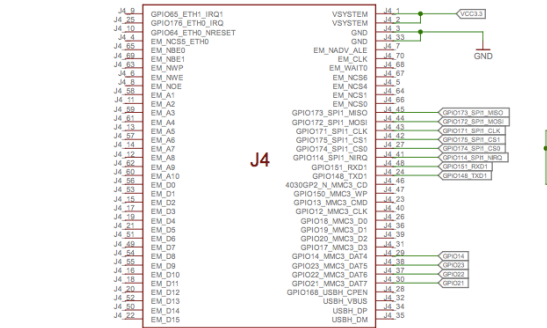
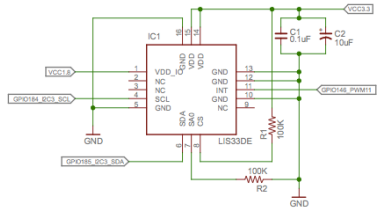
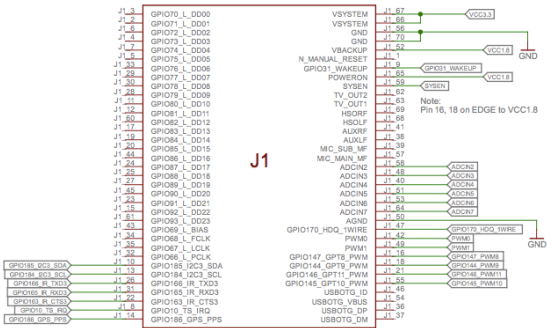
Appendix B – GumPack Hardware Schematics and PCB Layouts

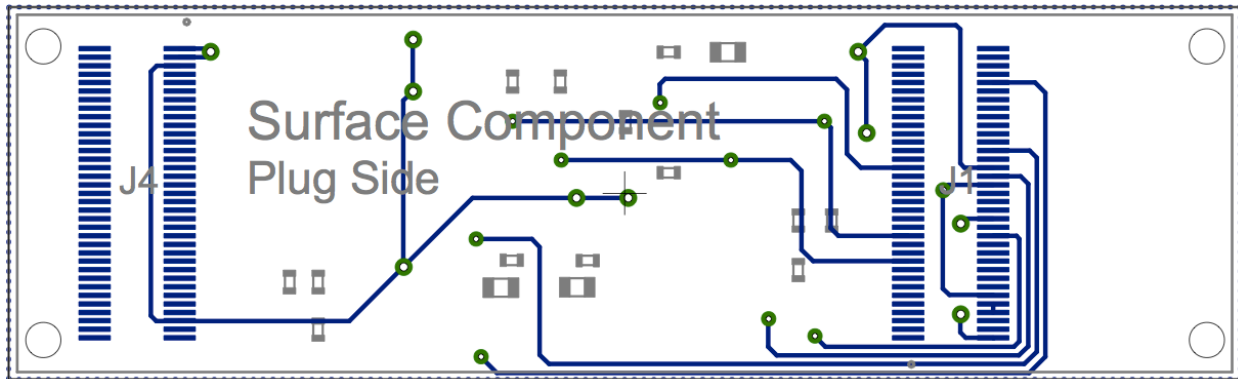
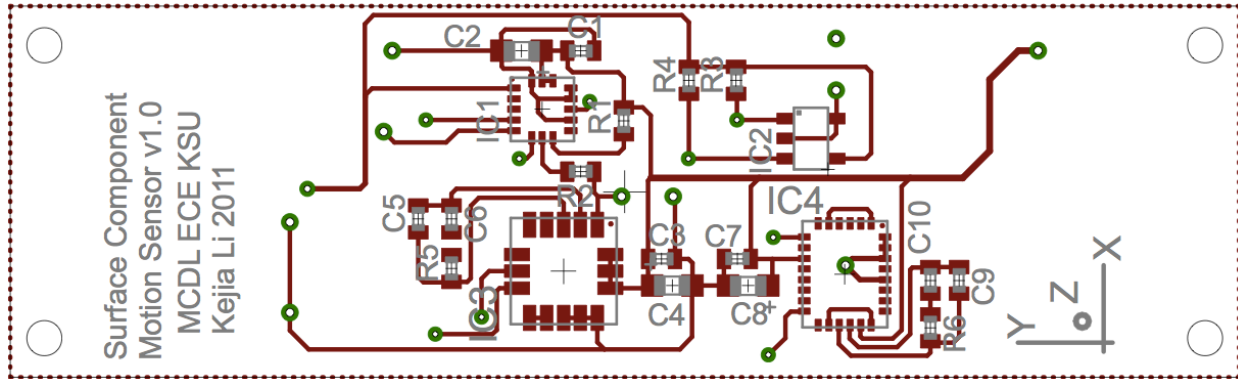
GumPack Surface Substrate



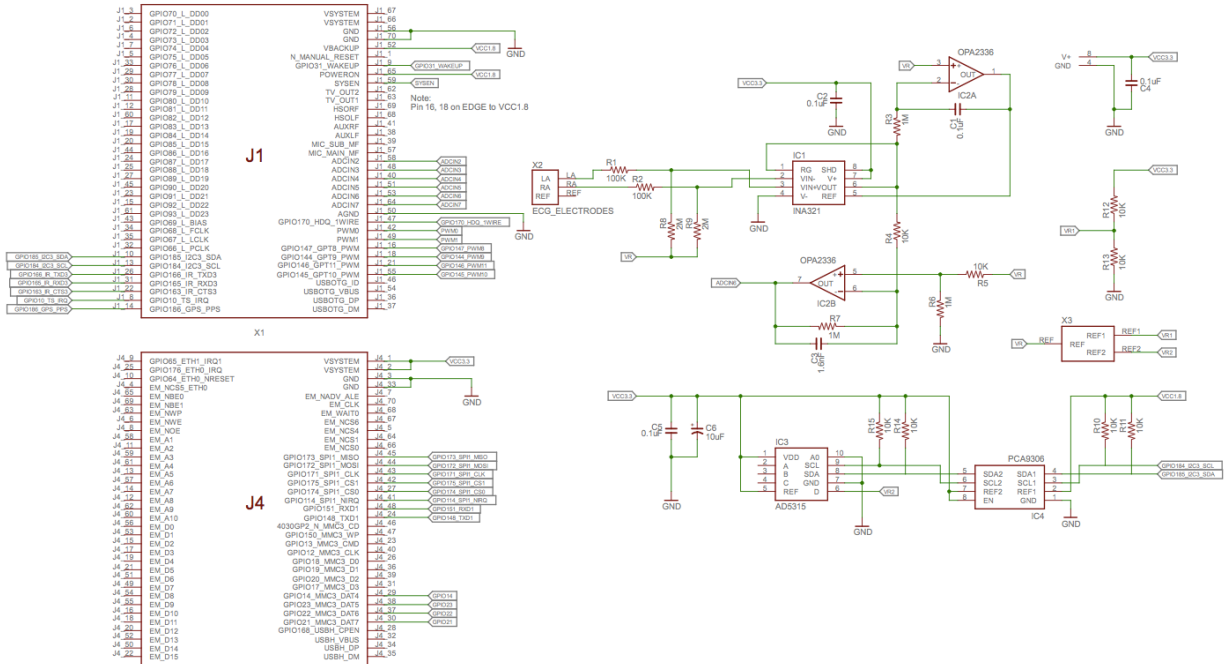


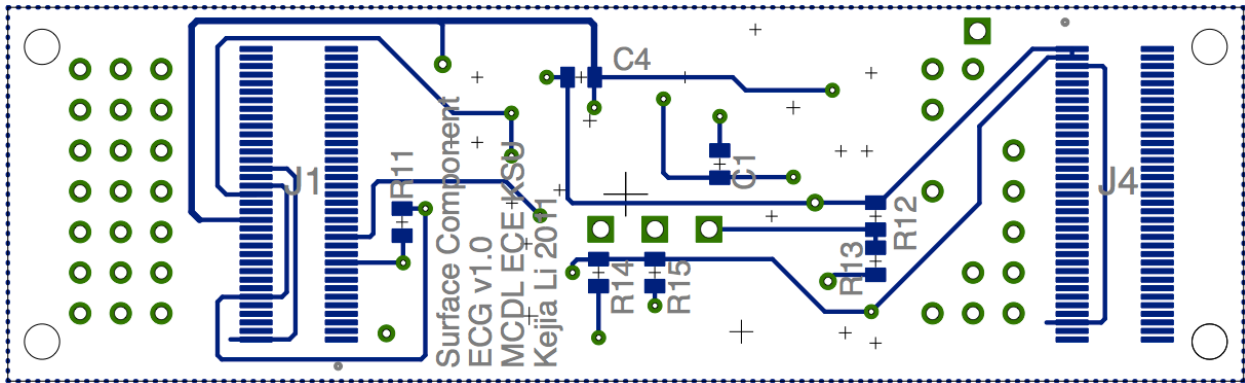
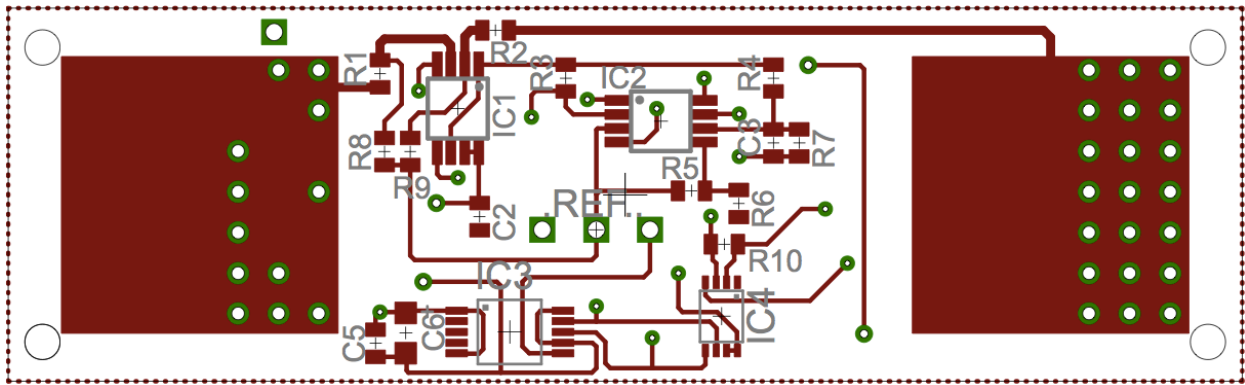
GumPack Surface Component: Motion Sensor



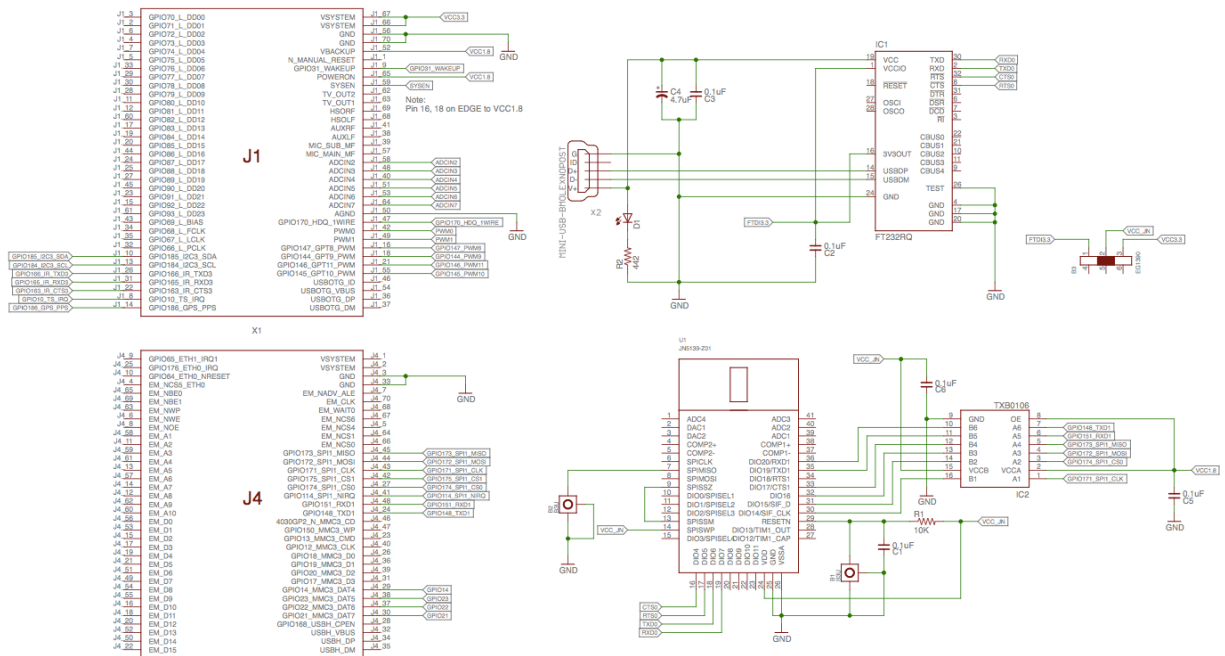


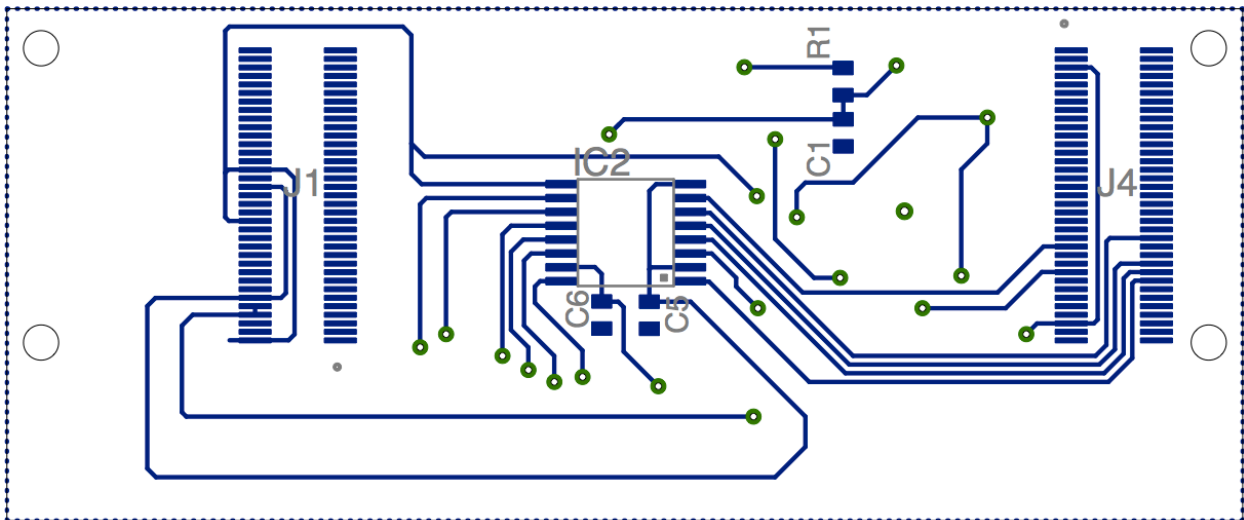
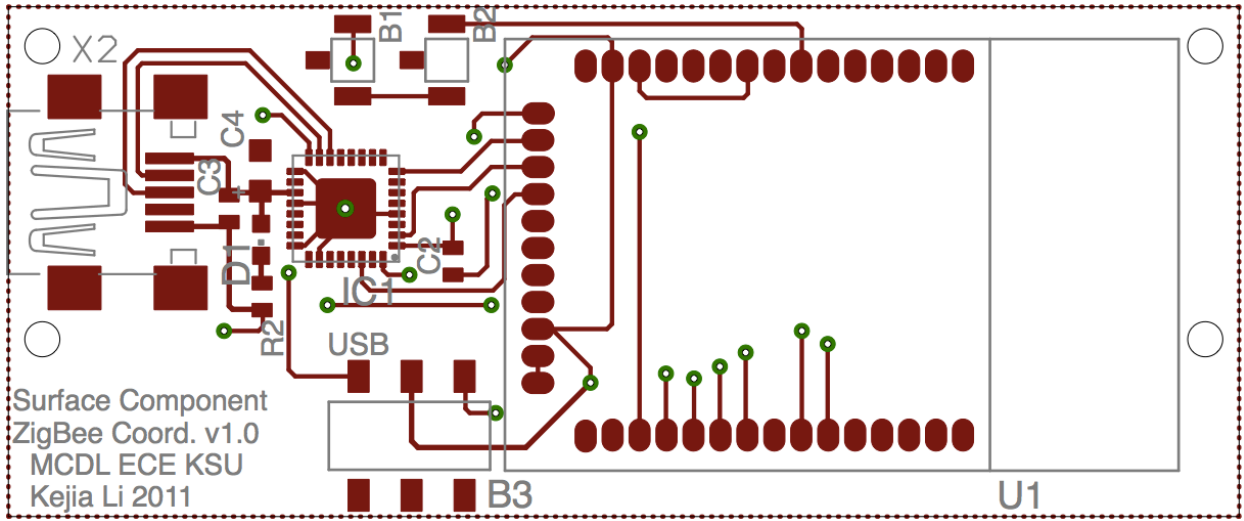
GumPack Surface Component: ECG





GumPack Surface Component: ZigBee Coordinator





Appendix C – Screenshots of the GumPack Web Interface

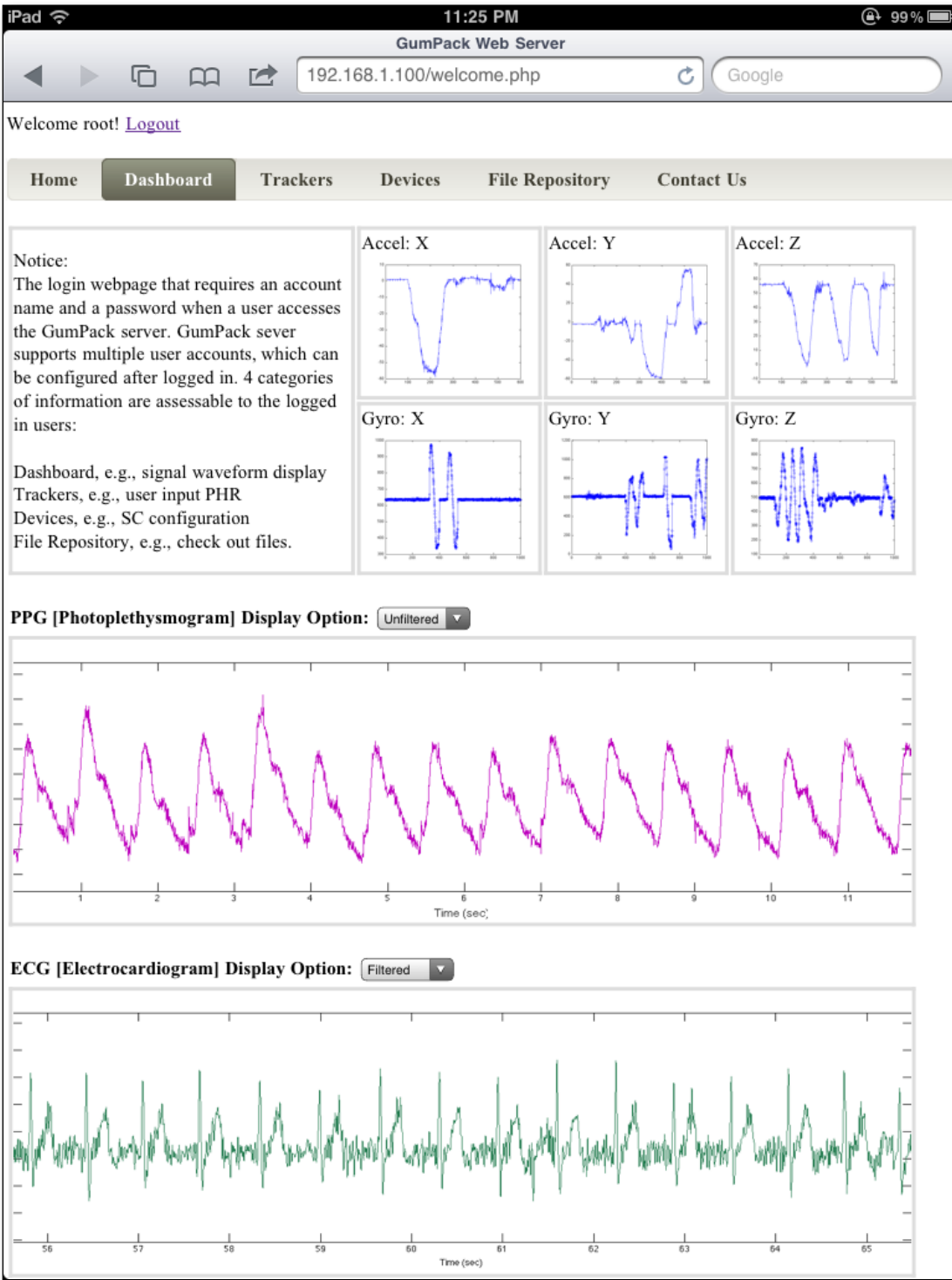


Figure C.1. GumPack web interface - Dashboard tag. Data acquired from an accelerometer, gyroscope, pulse oximeter, and ECG sensor.

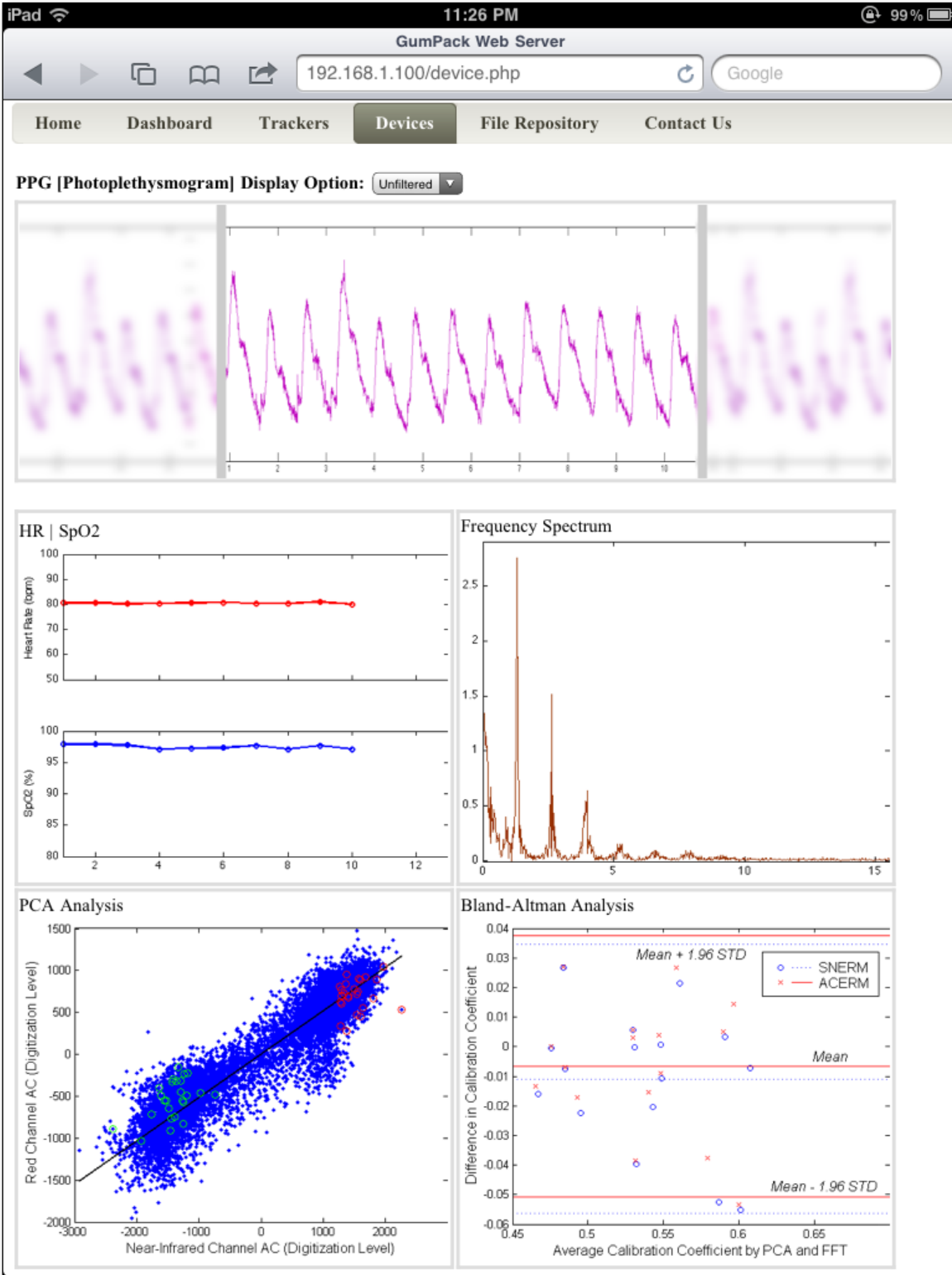


Figure C.2. GumPack web interface - Devices tag. Data selection and analysis for a pulse oximeter.

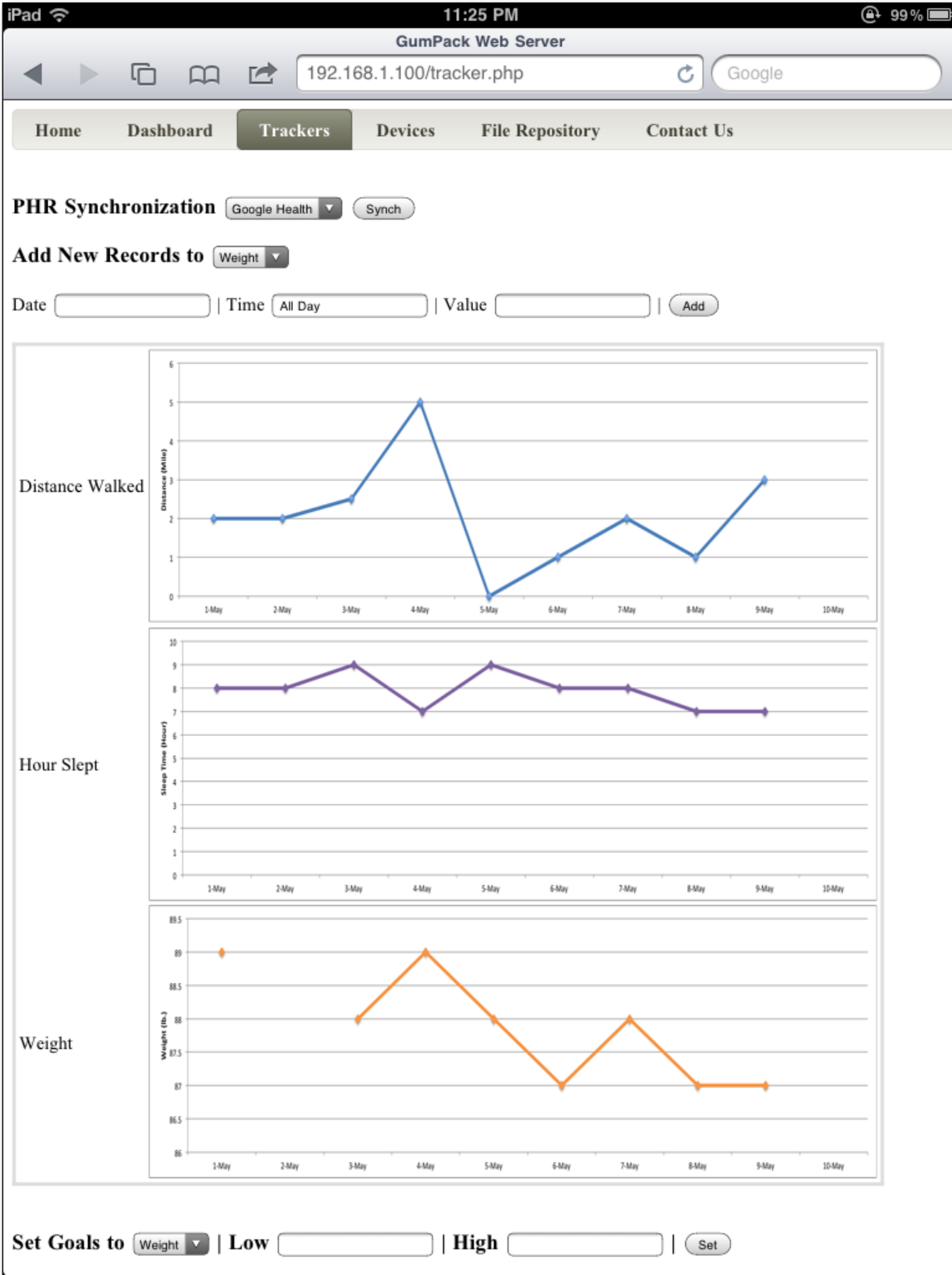
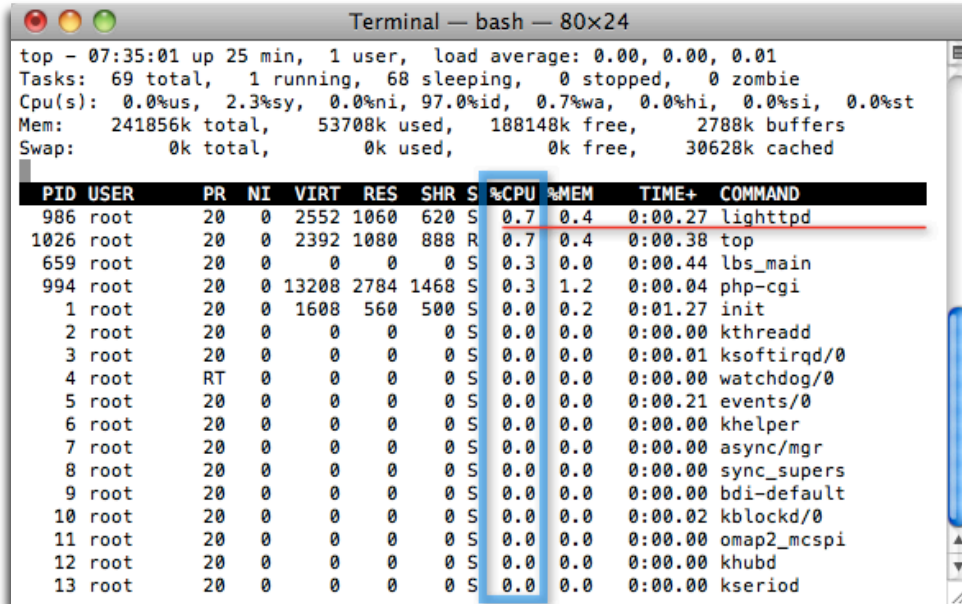


Figure C.3. GumPack web interface - Trackers tag.



Figure C.4. GumPack web interface - File Repository page.

Figure C.5 notes the CPU usage on the GumPack when an iPad accesses the web server (lighttpd). Its peak CPU usage is typically < 1.0%.



```
Terminal — bash — 80x24
top - 07:35:01 up 25 min, 1 user, load average: 0.00, 0.00, 0.01
Tasks: 69 total, 1 running, 68 sleeping, 0 stopped, 0 zombie
Cpu(s):  0.0%us,  2.3%sy,  0.0%ni, 97.0%id,  0.7%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   241856k total,  53708k used, 188148k free,  2788k buffers
Swap:    0k total,    0k used,    0k free,  30628k cached

  PID USER   PR   NI  VIRT  RES  SHR  S  %CPU  %MEM  TIME+  COMMAND
  986 root    20    0  2552 1060  620  S   0.7   0.4   0:00.27 lighttpd
 1026 root    20    0  2392 1080  888  R   0.7   0.4   0:00.38 top
   659 root    20    0     0     0     0  S   0.3   0.0   0:00.44 lbs_main
   994 root    20    0 13208 2784 1468  S   0.3   1.2   0:00.04 php-cgi
     1 root    20    0  1608  560  500  S   0.0   0.2   0:01.27 init
     2 root    20    0     0     0     0  S   0.0   0.0   0:00.00 kthreadd
     3 root    20    0     0     0     0  S   0.0   0.0   0:00.01 ksoftirqd/0
     4 root    RT    0     0     0     0  S   0.0   0.0   0:00.00 watchdog/0
     5 root    20    0     0     0     0  S   0.0   0.0   0:00.21 events/0
     6 root    20    0     0     0     0  S   0.0   0.0   0:00.00 khelper
     7 root    20    0     0     0     0  S   0.0   0.0   0:00.00 async/mgr
     8 root    20    0     0     0     0  S   0.0   0.0   0:00.00 sync_supers
     9 root    20    0     0     0     0  S   0.0   0.0   0:00.00 bdi-default
    10 root    20    0     0     0     0  S   0.0   0.0   0:00.02 kblockd/0
    11 root    20    0     0     0     0  S   0.0   0.0   0:00.00 omap2_mcspi
    12 root    20    0     0     0     0  S   0.0   0.0   0:00.00 khubd
    13 root    20    0     0     0     0  S   0.0   0.0   0:00.00 kseriod
```

Figure C.5. GumPack system resources allocated for the web server lighttpd.

Appendix D – Instructions to Install the JRE on the GumPack

Software development on the GumPack is largely based on the Java language, including the MDCF-related work. Appendix D addresses how to install a GumPack-supported JRE, from connecting the device to a computer to running a Java program on the GumPack.

First, a serial connection is required to access the GumPack console – a native user interface to the processor SC (Gumstix Overo board). The USB console port on the GumPack must be connected to a USB port on a computer. Serial communication tools used on computers vary among the different operating systems. The Mac OS X (Terminal) was used for this tutorial. As to other solutions such as Kermit on Linux and Putty on Windows, more information is available in [105].

For a first-time connection, install the FTDI chip driver that can be downloaded from their website. Open the Terminal program and use the following commands to establish a connection.

```
$ screen /dev/tty.usbserial-A8005CU9 115200 8N1
```

Here, `screen` is the name of the screen program and `/dev/ tty.usbserial-A8005CU9` is the name of the connection controller on the GumPack under the directory `/dev`. The token `115200` specifies that the connection rate is 115200 bps, and the token `8N1` specifies the use of 8 data bits, no flow control, and 1 stop bit,

```
$ screen -r
```

The token `-r` identifies the need to reattach to the communication console, e.g., after the Terminal program is closed.

Power on the GumPack; its boot-up information will be shown in the screen program. Log in with an administrator account, such as the default account “root.” Since the current console connection is not configured to transfer files/packages to the GumPack system, there are two options to do so: SD card (off-line) or Wi-Fi (on-line). Here the Wi-Fi method is introduced, which requires an extra step to configure Wi-Fi on the GumPack.

Execute the commands in Figure D.1, or write an executable script to connect to a Wi-Fi access point with the name of TP-LINK_DE455E. In the example, the WEP password is “1234567890,” the gateway is 192.168.1.1, and the IP address that the GumPack requires is 192.168.1.101.

```
kejalali — screen — 80x24
wlist wlan1 scan
ifconfig wlan1 down
ifconfig wlan1 192.168.1.101 netmask 255.255.255.0 broadcast 255.255.255.255
iwconfig wlan1 essid "TP-LINK_DE455E"
iwconfig wlan1 key 1234567890
ifconfig wlan1 up
route add default gw 192.168.1.1
~
~
```

Figure D.1 Configuration file for Wi-Fi access.

If the operation succeeds, the wlan1 configuration in the GumPack should look like the listing in Figure D.2. Ping the gateway to ensure the Wi-Fi connection is alive:

```
$ ping -c3 192.168.1.1
```

```
kejalali — screen — 80x24
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 58.838/82.937/103.028/18.262 ms
root@overo:~# ifconfig wlan1
wlan1    Link encap:Ethernet  HWaddr 00:19:88:3D:1E:38
         inet addr:192.168.1.101  Bcast:255.255.255.255  Mask:255.255.255.0
         inet6 addr: fe80::219:88ff:fe3d:1e38/64  Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:44 errors:0 dropped:0 overruns:0 frame:0
         TX packets:137 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:4456 (4.3 KiB)  TX bytes:21258 (20.7 KiB)

root@overo:~# iwconfig wlan1
wlan1    IEEE 802.11bg  ESSID:"TP-LINK_DE455E"
         Mode:Managed  Frequency:2.412 GHz  Access Point: 00:25:86:DE:45:5E
         Bit Rate=5.5 Mb/s
         Retry long limit:7  RTS thr:off  Fragment thr:off
         Encryption key:1234-5678-90
         Power Management:on
         Link Quality=70/70  Signal level=-31 dBm
         Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
         Tx excessive retries:0  Invalid misc:0  Missed beacon:0

root@overo:~# █
```

Figure D.2. Wireless interface configuration after connecting to the access point successfully.

Note that using Wi-Fi and/or Bluetooth double the power consumption of the GumPack and significantly increase the system temperature. Write the following script into the file `/etc/profile` to control the switch:

```
# disable wifi
echo 0 > /sys/class/gpio/gpio16/value
# disable bt
echo 0 > /sys/class/gpio/gpio164/value
```

If Internet access is available, the latest versions of the JRE-related packages can be directly downloaded from the repository. First, add the angstrom repository using the following command:

```
$ echo 'src/gz angstrom-base
http://www.angstrom-distribution.org/feeds/unstable/ipk/gli
bc/armv7a/base' > /etc/opkg/angstrom-base.conf
```

Install the latest version of the JRE on the GumPack. Choose either JamVM or Cacao.

```
$ opkg update
$ opkg install cacao
```

If you need graphical class support, install the GTK package.

```
$ opkg install classpath-gtk
```

Check the Java version, and execute a Java program.

```
$ java -version
$ java -jar archive_name.jar
```

Appendix E – Exporting the MDCF to the GumPack

In Appendix D, the JRE was set up on the GumPack. Appendix E addresses how to export the Java code in the original MDCF code base to the GumPack and establish a wireless connection between a GumPack and other MDCF components, e.g., an MDCF server on a computer.

The first step is to set up the address of the MDCF message bus. In this example, the server on a computer is designated the IP address of 192.168.100. Locate the classes TCPRecvClient and TCPSndrClient in the package mdcf.messagebus.transport.tcp of the MDCF code base and modify the following code.

```
private String hostname = "192.168.1.100";
```

Figure E.1 illustrates an example scenario where an MDCF client is exported to a GumPack. Two clients are used: one to publish a message (source), and the other to receive a message from a console running on the MDCF host (sink).

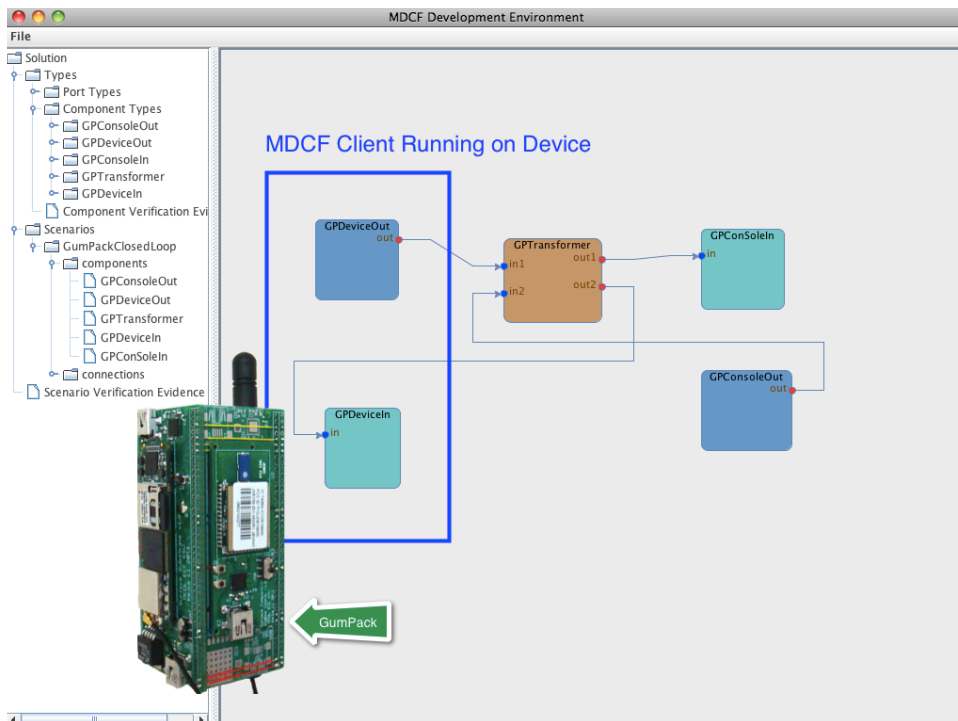


Figure E.1. Example scenario with an MDCF client running on the GumPack.

In Eclipse, choose File -> Export, as shown in Figure E.2. Select Runnable JAR file and click Next.

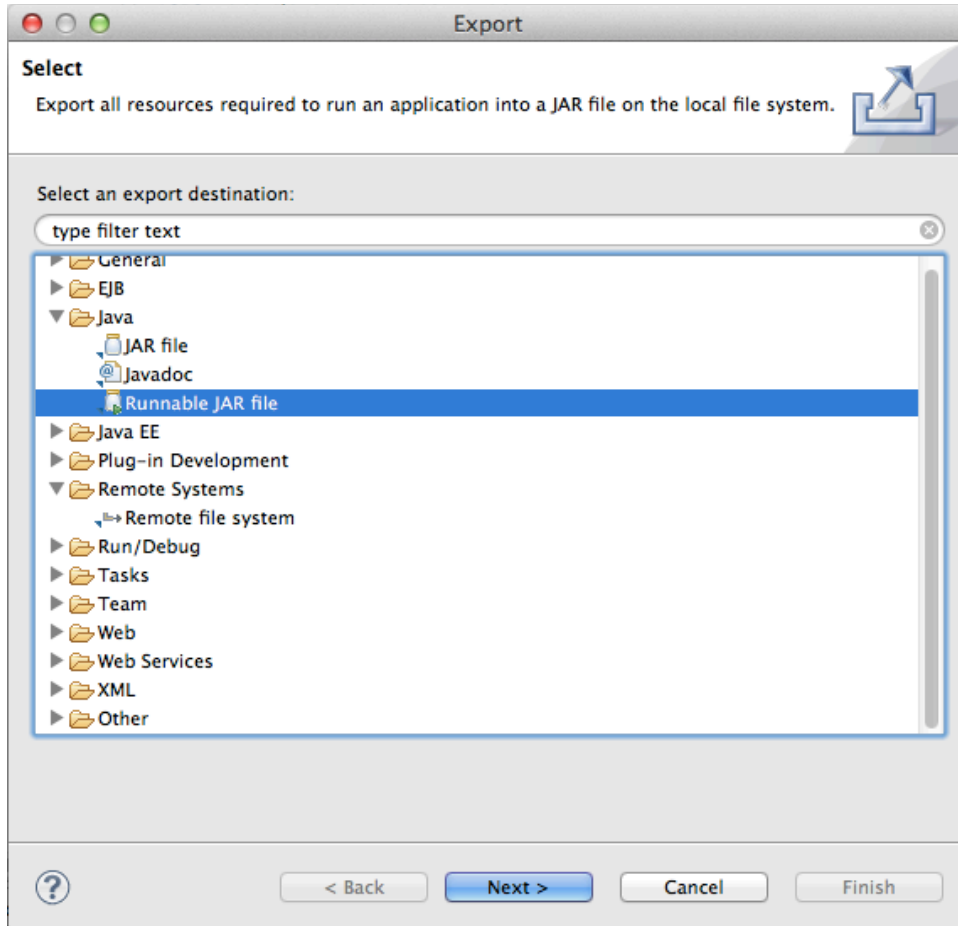


Figure E.2. Exporting resources to a JAR file in Eclipse.

Select the correct launch configuration that has been run on the server side at least once, although it will be exported to the GumPack side. Specify the export destination where a JAR archive is to be created. Note that one must also extract the required libraries into a generated JAR since those libraries are currently unavailable in the JRE of the GumPack.

Figure E.3 illustrates the final step to create the target JAR file. Upload this file to the GumPack either via an SD card or Wi-Fi. Make sure the GumPack is in the same wireless domain as the MDCF server, e.g., by connecting to the same wireless router. Execute the MDCF client on the GumPack after the message bus is established on the server side as a regular routine.

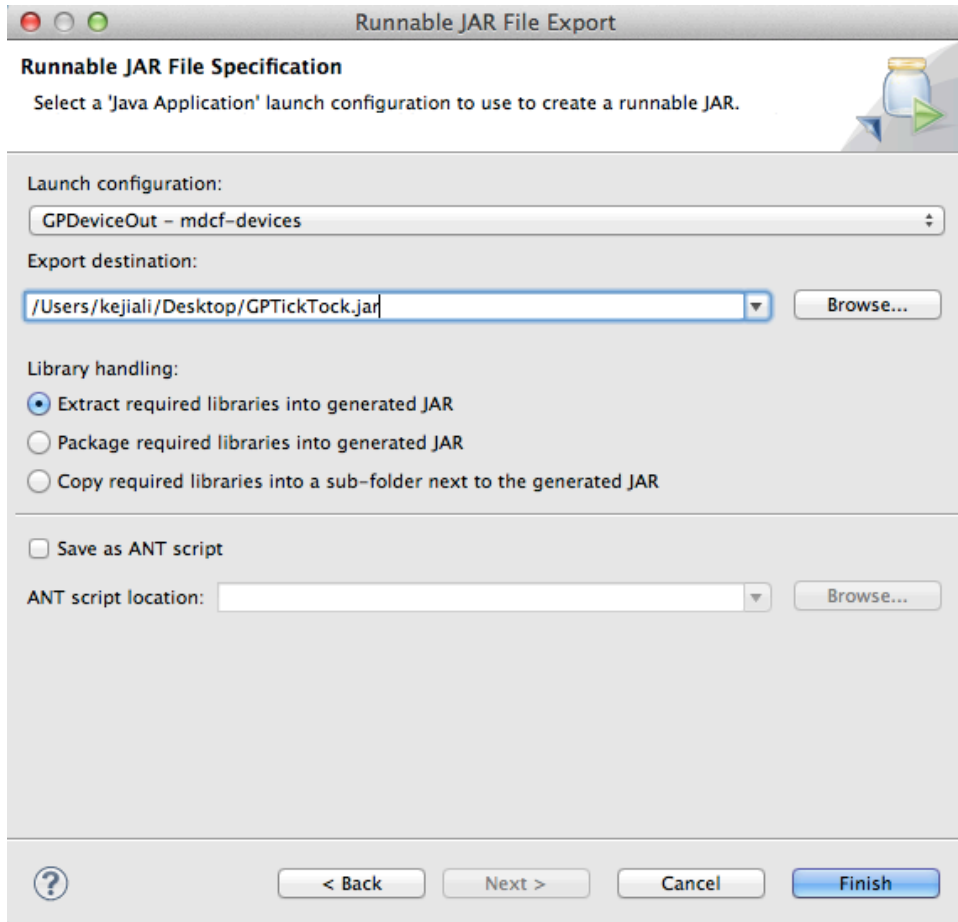


Figure E.3. Choosing the launch configuration and file properties in the last step of a runnable JAR file export.

One significant difference in the Java code for the GumPack is that its JRE currently does not support the XStream library which transform a Java object to/from XML. The SAX XML parser is adopted to address this issue. Import the following libraries to replace XStream and replace the original codes with new codes using the new libraries (see Figures E.4 through E.7).

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
```

```

Object msg = xstream.fromXML(messageText);
if(msg instanceof PubTopicAssignmentMsg){
    PubTopicAssignmentMsg pubAssign = (PubTopicAssignmentMsg)msg;
    outSender = SenderFactory.createSender();
    outSender.connect(pubAssign.topicMap.get("out"));
}
else if(msg instanceof SubTopicAssignmentMsg){
    SubTopicAssignmentMsg subAssign = (SubTopicAssignmentMsg)msg;
}
}

```

Figure E.4. Original code in the MDCF when receiving a message regarding a publish topic.

```

System.out.println(messageText);
InputSource is = new InputSource(new StringReader(messageText));
try {
    dom = dbuilder.parse(is);
} catch (SAXException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
Element docEle = dom.getDocumentElement();
System.out.println(docEle.getNodeName());
String strPubTopicAssignmentMsg = "mdcf.core.messages.mgmt.PubTopicAssignmentMsg";
if (docEle.getNodeName().equals(strPubTopicAssignmentMsg)) {
    String guid = null;
    String scenarioId = null;
    Map<String, String> topicMap = new HashMap<String, String>();

    NodeList nlGUID = docEle.getElementsByTagName("GUID");
    if (nlGUID.getLength() > 0) {
        guid = nlGUID.item(0).getFirstChild().getNodeValue();
        System.out.println("guid = " + guid);
    }
    NodeList nlSNR = docEle.getElementsByTagName("scenarioId");
    if (nlSNR.getLength() > 0) {
        scenarioId = nlSNR.item(0).getFirstChild().getNodeValue();
        System.out.println("scenarioId = " + scenarioId);
    }

    NodeList nlMap = docEle.getElementsByTagName("topicMap");
    if (nlMap.getLength() > 0) {
        NodeList nlEntry = ((Element) nlMap.item(0)).getElementsByTagName("entry");
        for (int i = 0; i < nlEntry.getLength(); i++) {
            NodeList nlString = ((Element) nlEntry.item(i)).getElementsByTagName("string");
            String key = nlString.item(0).getFirstChild().getNodeValue();
            System.out.println(key);
            String value = nlString.item(1).getFirstChild().getNodeValue();
            System.out.println(value);
            topicMap.put(key, value);
        }
    }
    System.out.println("topicMap = " + topicMap);

    PubTopicAssignmentMsg pubAssign = new PubTopicAssignmentMsg(guid, topicMap, scenarioId);
    outSender = SenderFactory.createSender();
    outSender.connect(pubAssign.topicMap.get("out"));
}
}

```

Figure E.5. Modified code in the GumPack when receiving a message regarding a publish topic.


```

Object msg = xstream.fromXML(messageText);
if(msg instanceof PubTopicAssignmentMsg){
    PubTopicAssignmentMsg pubAssign = (PubTopicAssignmentMsg)msg;
}
else if(msg instanceof SubTopicAssignmentMsg){
    SubTopicAssignmentMsg subAssign = (SubTopicAssignmentMsg)msg;
    if(subAssign.topicMap.get("in") != null){
        inReceiver.connect(subAssign.topicMap.get("in"));
        inReceiver.regAsynch(new inlistener());
    }
}
}

```

Figure E.6. Original code in the MDCF when receiving a message regarding a subscribe topic.

```

InputSource is = new InputSource(new StringReader(messageText));
try {
    dom = dbuilder.parse(is);
} catch (SAXException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
Element docEle = dom.getDocumentElement();
System.out.println(docEle.getNodeName());
String strPubTopicAssignmentMsg = "mdcf.core.messages.mgmt.SubTopicAssignmentMsg";
if (docEle.getNodeName().equals(strPubTopicAssignmentMsg)) {
    String guid = null;
    String scenarioId = null;
    Map<String, String> topicMap = new HashMap<String, String>();

    NodeList nlGUID = docEle.getElementsByTagName("GUID");
    if (nlGUID.getLength() > 0) {
        guid = nlGUID.item(0).getFirstChild().getNodeValue();
        System.out.println("guid = " + guid);
    }
    NodeList nlSNR = docEle.getElementsByTagName("scenarioId");
    if (nlSNR.getLength() > 0) {
        scenarioId = nlSNR.item(0).getFirstChild().getNodeValue();
        System.out.println("scenarioId = " + scenarioId);
    }

    NodeList nlMap = docEle.getElementsByTagName("topicMap");
    if (nlMap.getLength() > 0) {
        NodeList nlEntry = ((Element) nlMap.item(0)).getElementsByTagName("entry");
        for (int i = 0; i < nlEntry.getLength(); i++) {
            NodeList nlString = ((Element) nlEntry.item(i)).getElementsByTagName("string");
            String key = nlString.item(0).getFirstChild().getNodeValue();
            System.out.println(key);
            String value = nlString.item(1).getFirstChild().getNodeValue();
            System.out.println(value);
            topicMap.put(key, value);
        }
    }
    System.out.println("topicMap = " + topicMap);

    SubTopicAssignmentMsg subAssign = new SubTopicAssignmentMsg(guid, topicMap, scenarioId);
    inReceiver.connect(subAssign.topicMap.get("in"));
    inReceiver.regAsynch(new inlistener());
}
}

```

Figure E.7. Modified code on the GumPack when receiving a message regarding a subscribe topic.

Appendix F – Acronyms and Abbreviations

A list of commonly-used acronyms and abbreviations in the dissertation follows.

Table F.1. Acronyms and Abbreviation

Acronym/Abbr.	Name
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
APP	Application
BAN	Body Area Network
DOM	Document Object Model
DVP	Digital Volume Pulse
ECG	Electrocardiogram
EEG	Electroencephalogram
EHR	Electronic Health Record
EKG	Electrocardiogram
EMG	Electromyogram
EMR	Electronic Medical Record
EOG	Electrooculogram
GCCF	GumPack Component Coordination Framework
GUI	Graphical User Interface
HIIS	Hospital Information Integration System
ICE	Integrated Clinical Environment
JDBC	Java Database Connectivity
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JRE	Java Runtime Environment
MDCF	Medical Device Coordination Framework
MDDS	Medical Device Data System

MDPnP	Medical Device Plug and Play
MPAD	Medical Platform-Aggregated Device
MPOD	Medical Platform-Oriented Device
MOM	Message-Oriented Middleware
PHR	Personal Health Record
PPG	Photoplethysmogram
PWV	Pulse Wave Velocity
SB	Surface Biosensor
SC	Surface Component
SDK	Software Development Kit
SS	Surface Substrate
WBAN	Wireless Body Area Network