Homework 9: Software Design Considerations

Team Code Name: Vitalis  Group No. 13
Team Member Completing This Homework: Shantanu Joshi
E-mail Address of Team Member: joshi4@purdue.edu

Evaluation:

<table>
<thead>
<tr>
<th>SEC</th>
<th>DESCRIPTION</th>
<th>MAX</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Introduction</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2.0</td>
<td>Software Design Considerations</td>
<td>30</td>
<td>26</td>
</tr>
<tr>
<td>3.0</td>
<td>Software Design Narrative</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>4.0</td>
<td>Summary</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5.0</td>
<td>List of References</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>App A</td>
<td>Flowchart/Pseudo-code for Main Program</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>App B</td>
<td>Hierarchical Block Diagram of Code Organization</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>TOTAL</td>
<td>100</td>
<td>96</td>
</tr>
</tbody>
</table>

Comments:
Grader: George Toh

Excellent Work!
1.0 Introduction

Our project is to design a wrist-mounted wireless health monitoring system for both a hospital and residential setting. The system focuses on monitoring patient vital health parameters (such as pulse rate, SpO₂ and skin temperature) and transmitting them via Wi-Fi for secured remote web access. Other attractive features include an automatic alarm system in the case of anomalous readings, fall detection and a battery management system that displays battery life and allows recharging while the device is still in use. Multiple software modules are required to ensure proper behavior of the microcontroller and web application. Our device is designed to be portable and is thus battery powered, this makes it important to balance the safety and reliability of the device with methods for conserving power and prolonging the life of the device.

2.0 Software Design Considerations

2.1 Memory Utilization

The MCU has to complete a variety of tasks that include initialization of various modules (UART, I2C, ADC etc.), sampling sensors, updating data buffers and patient information (on OLED) and wirelessly transmitting data to the appropriate server when the buffer is full.

The ATmega1284 has 128kB of Flash, 4k of EEPROM and 16k of SRAM [1]. Our program does not tax the MCU memory system heavily as we will only be storing 3 relatively small data buffers, which will have readings for skin temperature, pulse and SpO₂. There will be a separate flag for when an emergency has been detected. Finally, a handful of variables, (six as of now) which are used to automatically infer an emergency complete the memory requirements for our design. There is no requirement for heap memory, as nothing needs to be allocated at run-time.

The program will be stored in Flash memory (128kB starting at 0x0000)[1] so that even if the battery dies out the program will still remain on the device and function correctly when the battery has been recharged. No other data (like personal information) will be stored in Flash since in a hospital setting this device can be used on any number of patients in its lifetime and it does not make sense to add any personalization touches or retain any patient information.

2.2 Peripherals and Code Organization

Organization

The main function of the MCU(Microcontroller Unit) is to sample the sensors and update the buffers. A natural choice to implement this functionality is a polling method. However, instead of taxing the MCU by including the sampling code inside an infinite while loop, it is much more efficient to use the Interrupt Driven polling method as this allows us, the designers, to precisely control the frequency at which the sensors need to be sampled and helps lower power consumption. We are still in the process of empirically determining the exact interrupt interval and buffer size(s)

Comment [GT1]: While this makes sense, it might be good to store info from the last 30-60s on flash, in case the device resets unexpectedly?

Comment [GT2]: Good reasoning.
that will balance safety and reliability required of the device with the desired outcome of conserving
the battery as continuously transmitting via WiFi will draw too much current too quickly from the
battery. Hence, the data is stored in a buffer will be transmitted once the buffer is full. The interrupt
driven polling method is implemented by using the timer module to trigger an interrupt when it
counts to a specific value stored in the OCR1A register [1]. The interrupt handler will then start the
process of sampling the sensors, updating the data-buffers and if necessary transmit the data over
WiFi.

**Peripherals**

*Accelerometer:* An accelerometer (ADXL 335)[2] located on the patients shoulder is used to sense if
the patient has suffered a fall and if so automatically raise an alarm. The ADXL 335 is a 3-axis
accelerometer with a sensing range of +/- 3g. It outputs analog data and thus has to be interfaced to
the microcontroller, using analog to digital conversion. Port A has 8 ADC (PA0-PA7) out of which
we require 3, (PA0-PA2) one for each axis. The ATmega1284 provides 2 modes of operation for
ADC 8 bit resolution and 10 bit resolution. Using 10 bit resolution places restrictions on how fast the
device can be clocked, whereas the 8 bit mode has no such restrictions [2]. We plan to detect a fall
by looking out for large changes in successive accelerometer readings. Therefore the 8-bit mode is
suitable as there is no need to have extremely accurate individual readings.

*Temperature Sensor:* The TMP102 [3] is used to report Skin temperature to the MCU. It has a great
resolution of 0.0625 degrees Celsius and is extremely robust and can operate accurately between -25
degrees Celsius to 85 degrees Celsius. It is interfaced to the microcontroller using I2C (TWI )
communication protocol. The MCU will be the master and the TMP102 has one of 4 slave addresses
depending upon whether the ADDR pin is connected to VCC, GND, SDA or SCL. This was a design
choice by the makers of the sensor to allow 4 different sensors on the same bus. The TMP102 sensor
also has inbuilt pull-up resistors(1K-ohm) to pull both the SDA and SCL lines high [3]. The SCL and
SDA lines of the MCU are located in PC0 and PC1 respectively.

*Pulse-Oximeter Sensor:* We are building our own pulse-oximeter sensor using the TSL230 Light to
Frequency optical sensor [4]. The concept behind the pulse-oximeter sensor is to find the ratio
between the frequency of light absorbed using a Red LED and an infra-red LED when the light is
passed through the patients finger. An external interrupt, associated with PD0, PD1 or PD2 is used to
discern the frequency output from the TSL230. A detailed explanation of the working of the pulse-
oximeter sensor can be found in the next section.

Comment [GT3]: How frequently will you take readings in order to not “miss” a fall?
OLED Screen: The OLED [5] screen mounted on the patient’s wrist provides a summary of all the important information that the device monitors namely: temperature, pulse, SpO2, battery-status and emergency flag status. The novel function of our device is to transmit the patient data wirelessly to a central server to facilitate remote monitoring of multiple patients by a doctor at his/her convenience. However, the OLED provides an important redundancy as in the event of a failure in the WiFi chip the device is still useful and poses no risk to the patient. The OLED is updated via command sent using UART located in PD2 and PD3 on the MCU.

Fuel Gauge: The LiPo Fuel Gauge connects directly to the battery and uses a sophisticated algorithm to detect relative state of charge and direct A/D measurement of battery voltage. The Fuel Gauge, like the TMP102 sensor is interfaced to the MCU using I2C. Since, we need to read from the fuel gauge [6], it will be operated in slave mode.

WiFly Shield: The RN-171 [7] is a complete plug and play WiFi solution. It is configurable through commands sent over UART. What attracted us to this specific product is that it is an ultra low power device (4 μA sleep, 40mA Rx and 180mATx at 10 dBm) and has a built in HTML client which can be used to directly issue POST requests to our web-server.

NFC Tag: We would like to do away with the traditional paper records kept at the foot of the patient’s bed for the doctor to check on his rounds with an NFC version of the same. We have attached an NFC tag to our device, which can be read by an NFC capable smart phone or tablet. In order to ensure security we have created our own Android App, which requires a user to login before he/she is redirected to the web-app, which displays a patient’s vital information.

2.3 Debugging

Initially, we utilized the UART for debugging purposes, much like ‘printf’ statements in C. Now, we use the AVR dragon and utilize its JTAG interface to program our microcontroller. This is extremely convenient as it provides us with on-chip debugging capabilities, which makes our development process much easier. Spotting errors and verifying the correctness of our code becomes much easier when we can set breakpoints and observe the values of variables in memory during run time.

3.0 Software Design Narrative

Pulse-Oximeter [Successfully Tested with PDIP package]:

We are building our own pulse-oximeter sensor using the TSL230 light to frequency converter [4]. The concept behind a pulse-oximeter is that oxygen present in the blood absorbs different frequencies
of red and infra-red light, the ratio of which can be used to calculate the oxygen saturation (SpO2) and counting the peaks obtained in the data lets us determine the pulse.

When light hits the TSL230 it outputs the frequency of the incident light as a pulse train. The MCU is configured such that each pulse triggers an external interrupt which updates a particular counter based on whether it was the red light or the infra-red led whose light was incident on the sensor. A timer is used to measure the frequencies for a specified amount of time before we find the ratio to determine the SpO2 reading.

In addition to determining the frequency a peak detection algorithm is implemented to measure the pulse. As of now, we have a simple peak detection, which detects a peak when the incoming readings, which have been consistently increasing, start decreasing. However, due to the contraction and expansion of the heart we obtain two peaks thus the final result has to be divided by 2 (or left shifted 1 bit) to obtain the correct result.

**Timer Interrupt [Successfully Tested with PDIP package]**:

The timer interrupt is used to implement the interrupt driven polling method of code organization. When the timer reaches a certain value an interrupt is triggered and all the sensors are sampled in its corresponding ISR.

Writing a 1 to the OCIE1A bit in the TIMSK1 register enables the timer interrupt. The TCCR1B register is used to select the appropriate pre-scalar to generate the clock used by the Timer module. The TCCR1A register is used to clear the counter when it reaches the value it is supposed to count up to (specified in 16 bit OCR1A register) to trigger the timer interrupt. Finally, setting the OCF1A bit in the TIFR1 register clears the flag that is set when OCR1A reaches the value it is supposed to count to [1].

**ADC [Successfully Tested with PDIP package]**:

The Analog to Digital Conversion module is used to interface the Accelerometer with the MCU. The ADC is initialized by writing a 1 to the ADLAR bit to the ADMUX register, which sets the ADC to 8bit mode. There is no pre-scalar required as we are operating in 8bit mode. Next, setting the ADIE bit to 1 in the ADSCRA register enables the interrupts. Finally, ADEN bit is set in the ADSCRA register to turn the ADC module on. Setting the ADCSC bit high in the ADSCRA register starts the conversion. The ISR (Interrupt Service routine) reads the final digital value stored in ADCH and changes the ADC channel sampled using a switch case statement which ensures that all three channels ADC0 to ADC2 are sampled before returning to the Timer ISR to sample the next sensor [1].

**Two Wire Interface (I2C) [Successfully Tested with PDIP package]**:
The I2C communication protocol [1] is used to interface the temperature sensor and Fuel gauge to the MCU. I used a library provided by Sparkfun to initialize all the correct registers and abstract away all the low level functions. The library provides high-level functions like Start, Stop, SendByte, WaitForCompletion GetReceivedByte etc. Using, these abstractions it was very easy to set up the TWI with the MCU as Master and the TMP102 sensor as a slave. I connected the ADDR pin to VCC and thus the temperature sensor had a slave address of 0x93. The TMP102 sensor returns a 16bit value, which is acquired by reading twice in succession. The two results are then combined by shifting the higher order 8 bits to the left by 8 bits and then performing a bitwise or of the result with the lower order 8 bits.

[Reference to Code] A similar approach is used to read the battery status from the fuel gauge.

UART [Successfully Tested with PDIP package]

The UART communication [1] is used to send data and commands to both the μOLED and the WiFi module. The USART initializations are pretty straightforward and taken straight from the datasheet. The baud rate is set to 9600. It is a 16bit value that has to be bit shifted into UBR0H and UBR0L registers respectively. Shifting a 1 into the RXEN0 and TXEN0 bits of the UCSR0B register enables the receiver and transmitter. Setting the stop bit to 1 and specifying the number of data bits to 8 finishes the USART initialization.

While transmitting a character, UCSR0A and UDRE0 are used to test if the transmit buffer is empty of not. If it is empty then the new character can be added to the transmit buffer. From here it is a simple matter to write a function that transmits strings over UART.

Android Application [Successfully Tested with the help of SW emulation]:

We wrote a simple Android application that would read the NFC tag and upon proper authentication would redirect the user to the web-application and display relevant patient information. It is coded in Java and makes use of a lot of high-level API’s that makes the development extremely straightforward. This app has been tested by software emulation of the NFC and it currently redirects to google.com because our web-app is still being tested using localhost.

Web Application [In progress, 80% done]

The web-application is written in Node.js [8], which is a highly scalable asynchronous single-threaded non-blocking framework for writing both web servers and clients. It is relatively new tool which has been actively adopted by the industry and hence quite a supportive community. The server I have written interacts with a MongoDB [9] database and is capable of adding, deleting and modifying the records of existing patients. Also stored in the database are credentials of doctors to ensure there is no unauthorized access.
Once a doctor or primary care giver navigates to the relevant URL to view the patients data the client makes a request to the server which responds with an array of the patients latest readings which are then plotted using a charting library called Smoothie.js [10].

The data is received when the WiFi module transmits (makes a POST request) on another route (URL) to the server. The exact protocol of the format of the data and the time-interval at which it will be sent is still being worked out. However, I have tested the web-app using localhost and manually making the POST requests to the server using a test page.

**Emergency Flag [Finished Pseudocode]**

The MCU continuously monitors the incoming data and can immediately alert the patient and doctor incase of an emergency. There is also a provision for the user to press an emergency button as a call for attention.

Fall detection is based on the successive readings of the accelerometer and if there is a large spike then it is assumed to correspond to a fall, which will set the emergency flag. Pulse, SpO2 and Temperature above and below certain thresholds (yet to be determined) will also set the emergency flag. Once the emergency flag is set then the data in the buffers is immediately sent along with the emergency flag to the web server so that the doctors can immediately resolve the emergency.

**4.0 Summary**

In conclusion, our wireless biometric sensor will utilize an accelerometer, temperature sensor, fuel gauge and a custom pulse-oximeter sensor to provide useful information to both the patient and the doctor. Most of our code has been prototyped (Accelerometer, I2C, UART, pulse-oximeter) on a PDIP package and in the coming weeks we will be testing all the different components together. Finally, the most critical aspect of our project, the wireless communication between the device and our web app needs to be completed and tested thoroughly so that we can accurately determine the timer interrupt interval and data buffer size.
5.0 List of References


Appendix A: Flowchart/Pseudo-code for Main Program

Figure 1: Flowchart for Main function and module Initializations

Comment [GT6]: After initialization, the microcontroller has nothing to do until an interrupt? Low power mode?
Figure 2: Flowchart for Timer interrupt and associated code modules
Appendix B: Hierarchical Block Diagram of Code Organization

Figure 3: Code Hierarchy and Code Organization