John Careaga

Timothy Chong

EC450 Final Project

Death Pit™ Proudly Presents

Bluetooth Heart Rate Monitor Report

**Objective**

The objective of this project was to build a working battery-powered bluetooth heart rate monitor.

**Project Design**

An independent battery regulated by an LM317T regulator powers a pulse sensor, an LM339N comparator, an MSP430 chip, and a bluetooth module. The comparator converts the pulse sensor’s voltage output to a noise-free square wave. This square wave is in turn fed into an MSP430, which then uses the alternating input signal to calculate the time interval between the user’s heartbeats. The MSP430’s data is transmitted to a computer over the bluetooth chip. An Mac OS application was written to use this data to continuously output the user’s heart rate in beats per minute. Once the computer program detects stabilization of heart rate, it halts the MSP430’s monitoring system and presents an estimated heart rate to the user. Additionally, the computer program communicates with the MSP to determine when the MSP should start and stop capturing data. All communication is done using the UART protocol.

All electronic components are installed onto one breadboard in a neat fashion.

**Hardware implementation**

**Power source**

The power drawn by the system is provided by a 9 volt battery. Two 1.5V battery in series have been tested to give a rather fluctuating voltage source when current was drawn, so an LM317T regulator was used to provide the system with a stable voltage supply. The 9V battery output is connected to the regulator’s input. The appropriate resistor values were chosen to ensure that the regulator’s output is at about 3.3v and the capacitors prevent oscillations in the voltage output (see schematics). The multimeter showed that the regulator’s output voltage was a constant 3.4v, an acceptable value.

Additionally, a switch was connected between the battery and the regulator to allow the user to turn the system on and off. This feature allows a longer battery life.

**Monitor - Comparator - LED**

A 3.3v pulse sensor was used to obtain heart rate signals. We found that the method of using Op-Amps and a speaker to sense pulses was an ineffective and unreliable way to capture heart rate signals. Thankfully, the pulse sensor found on Sparkfun was much more promising.

An LM339N comparator was used to convert the pulse sensor’s output to a noise-free square wave. The comparator (powered by 3.4v) outputs either Vcc or 0v depending on the input signal and the comparator threshold. 2.3V was found to be a good comparator threshold level to capture the peaks from the pulse sensor output signal. A voltage divider between Vcc and 0 is used to provide this threshold.

The square wave output of the comparator is connected to an LED, which allows the user to note when the pulse sensor’s signal is strong enough to generate the comparator output square wave. This way, the user can verify that the MSP430 is ready to capture heart rate signals.

**MSP - Bluetooth - Computer**

The comparator’s output is fed into the P1.3 input pin of the MSP430. The MSP430 is programmed and installed onto the breadboard and can be used anywhere so long as it is powered. The MSP430 and the computer application (see software section) calculate the user’s heart rate as with the data transmitted from the MSP430.

The sending and receiving of MSP430 data bytes is done through UART; therefore, the MSP430’s UART pins are connected to the bluetooth module. The MSP430’s transmit pin is connected to the bluetooth’s receive pin, and the MSP430’s receive pin is connected to the bluetooth’s transmit pin. Both the MSP and the Bluetooth are powered by the regulator’s output. The bluetooth module is paired with the computer and allows wireless transmission of bytes between the MSP and the computer.

**Software Implementation**

**MSP430**

**GPIO**

The GPIO programming section is very light. The GPIO setup is written in an initialization function that is run once in the main function. All that is required is to set P1.3 as an input. This is done by clearing the corresponding bit of P1DIR. No edge-triggered interrupts are used and no pullup resistors are needed. P1SEL and P1SEL2 are zero for the bits corresponding to P1.3, indicating that it serves as an IO pin. A macro definition, INBIT, is defined as the input pin number. This input pin is connected to the comparator’s output, resulting in the continuous change of P1IN3 between 1 and 0.

**UART**

The UART programming is more involved. The goal of the UART communication module is to provide a communication path between the MSP430 and the computer. The computer sends specific op-codes to the MSP that serve as instructions. If the computer sends the ‘START’ op-code to the MSP430, the MSP receives this byte, sends a byte back to the computer indicating that it has successfully read this instruction, and begins capturing heart rate data. Once data is acquired by the MSP430 (see WDT), an op-code byte is sent to the computer indicating that two bytes of heart rate data are available. Then the MSP proceeds to send these two data bytes to the computer. The computer takes in these data bytes and outputs an appropriate BPM value. This process of sending data-ready op-codes followed by two bytes of heart rate data continues indefinitely until the computer sends a STOP instruction byte to the MSP. When the MSP receives this instruction, it halts its data acquisition process, and sends a STOP confirmation byte to the computer, allowing the computer to verify that the process has ended. To start the heart rate monitoring process again, the computer simply sends the START instruction again.

The UART setup is written in an initialization function that is run once in the main function. The MSP, bluetooth, and computer share a common Baud rate (115200) to allow correct transmissions between the three devices.

To receive bytes from the computer, the receive interrupt handler is enabled. The receive handler is called each time the MPS’s receive buffer is filled with a new byte. If the byte received corresponds to the START instruction, the handler switches the MSP state to heart rate data acquisition mode and sends a START confirmation byte back to the computer. If however, the received byte is the STOP instruction, the handler sets the MSP state to inactive mode where heart rate data is no longer being acquired, and sends a STOP confirmation byte to the computer.

To send a byte of data, a function called “send\_byte” is called. This function first ensures the UCA0 module is not busy. It then copies a byte into the transmit register to be sent to the computer. The byte to be sent is determined by a switch statement. Within each case of the switch statement, the variable that determines which case to enter is updated appropriately. For example, after the first byte is copied to the transmit buffer, the transmission state variable is changed so that the next time the function is called it will send the second byte instead. Additionally, the START successful and STOP successful cases configure the MSP’s heart rate data acquisition mode appropriately.

**WDT**

The Watchdog Timer is used to monitor the input signal and record usable data to be sent to the computer. The WDT is configured to interval mode, and configured to set an interrupt flag every (512)/(8M) = 64μs. The initialization is written in a function that is run once in the main function. See the code comments under WDT section to follow each line. When the MSP’s receive reads a START instruction, its interrupt enables WDT interrupts. When the MSP’s receive reads a STOP instruction, its interrupt disables WDT interrupts.

The WDT interrupt handler is responsible for detecting signal edges, calculating data, and calling the “send\_byte” function to send the data. While the WDT interrupt is enabled, it monitors the input signal and increments a global variable i\_count (interrupt counter) with each interrupt. When a rising edge is detected, the number of interrupts that occurred while the signal was low is copied into a global variable ‘down\_count.’ The interrupt counter is reset and increments until a falling edge is detected. When the falling edge is detected, half the number of interrupts that occurred in that period while the signal was high is copied into the global variable ‘half\_count’. A local variable is created that consists of the sum the current half\_count value, the current down\_count value, and the previous half\_count value. Thus, this local variable estimates the number of WDT interrupts that occurred over the last pulse. The interrupt handler then sends the two bytes of data that represent the approximate number of WDT interrupts to the bluetooth. At every signal edge, the interrupt counter i\_count is reset to zero. See the commented code under WDT handler to see the detailed implementation of this algorithm. It is worth noting that the UART handlers and functions reset the WDT interrupt counters when a STOP instruction is received, thus allowing readiness of new data capturing.

**Main**

The Main function calibrates the SM clock to be used for the MSP, sets up each module initial configuration, and then enters low power mode and enables general interrupts. The CPU is off in low power mode and is only activated when interrupts handlers are called.

**Mac Application**

A Mac OS application was written to convert the Watchdog Timer intervals sent from the bluetooth module to sensible data to be presented to the user. OSSerialPort framework was used to allow serial communication between the application and the bluetooth module. When data is sent to the bluetooth module, the didReceiveData function is called, with the data bytes store inside the variable “data”. The program parses that data, and presents the appropriate information to the user depending on the state the the system is in.

When the application launches, it continuously looks for a serial communication channel with the bluetooth module with the identifier “HRM-RNI-SPP”. Once the the communication channel is found (bluetooth module paired), it switches the view button to allow the user to begin heart rate acquisition. When heart rate acquisition has been initiated, the application listens for pulse receive (198) command, which attaches with two bytes after for the watch dog timer interval recorded from MSP430 (see WDT software implmentation). Experimental testing has shown that the two bytes attached after the 198 commands are sometimes sent in the same package as the command, and sometimes otherwise, so the updatePulseStateWithData function was written to convert bytes in different package to the valid interval data needed. The 2 byte WDT time interval is converted to beats per minute with formula: Heart rate = 15625 / WDT count between two peaks.

The application would automatically send an end command (183) to the system when the heart rate stabilizes. This happens when it has occurred in seven consecutive times that the beat average of the most recent seven beats is within 1BPM of the previous average.

The application also incorporated a heart beating animation. The text showing the heart rate beats according to the heart beating of the user.

**Success Assessment and Difficulties Encountered**

**Success**

After much debugging and circuit building, the system performs **flawlessly**. The switch is turned on, powering all components installed on the breadboard. Once the user places his/her finger on the pulse sensor interface, the pulse signal calibrates and begins to output a periodic signal strong enough to generate the comparator square wave. We ensure to begin data capturing once the LED indicates signal stability. Once the bluetooth module pairs with the Mac application, we press the button on the screen to start data acquisition. The application continuously receives data wirelessly from the MSP and presents accurate numbers on the screen. Once the Mac application notices stability in the user’s heart rate, it halts the MSP, averages the stable data, and presents a final number. No bugs or glitches have been detected as of yet.

To ensure accurate numbers, we connected a function generator to the input of the MSP and observed the system’s results. We found high precision and high accuracy for a wide range of frequencies (0.1Hz - 20Hz). We also tested the system with multiple users including an MIT professor (30BPM) who admired the neatness of the system. Our monitor was able to capture data accurately for all tested users thus far.

**Difficulties**

**Independent Powering**

We wanted to power our project independently so we had to figure out how to provide a stable 3.3V to the components. Originally, we tried powering the components with two AA batteries directly. We found that the data was unreliable. After researching and learning about voltage regulators, we decided to buy a 9v battery and an LM317T regulator from RadioShack to obtain a consistent 3.4V Vcc.

**Pulse Sensor over Speaker/OpAmp**

Obtaining a strong pulse signal proved to be a challenge. At first, we tried the speaker/OpAmp method to try to obtain a noticeable signal. However, we found this method to be ineffective, noisy, and unreliable. In addition, the Op-Amps we had required high +Vcc and -Vcc in order to operate - a problem if we wanted to use as few batteries as possible. We decided to take the plunge and order the pulse sensor from Sparkfun. It proved to be a crucial decision: the pulse sensor was effective.

**0 - Vcc Rail Comparator**

We needed to convert the pulse sensor output to a noise-free square wave that was either 0v or 3.4v to ensure the MSP input was stable. In order to do so, we needed to use a comparator. However, the typical comparators we were used to were the Op-Amps that required +Vcc and -Vcc. After talking with Professor Freedman, who teaches EC410, we learned that we could use a rail to rail comparator that only required Vcc and ground. We found the LM339 comparator at RadioShack and made the purchase. Fortunately, it worked as expected.

**iPhone Bluetooth**

Initially, our plan was to make an iPhone application in conjunction with the MSP and bluetooth. However, after purchasing the bluetooth module, we realized that Apple requires the enrollment of products in their MFi program in order to allow communication between iPhone and the bluetooth module. After consideration, we decided instead to make a Mac application instead.

**Next Steps**

**Exploring GPIO**

An idea that occurred to us was to use multiple GPIO inputs to monitor different signals. We could for example, implement a system that would alert the user when the pulse sensor signal is strong enough to allow heart rate measuring. As of now, we use the LED to verify whether the comparator is outputting the desired square wave pattern. Additionally, other multiplexing input signals could be implemented to control operation modes, such as motion sensor mode (the pulse sensor detects when your hand moves above it).

**iPhone Implementation / Computer Program Features**

Additional features could be added to the computer software program, such as data saving, comparing, etc. The GUI would have more buttons and a menu of operation modes. The software could also be written for a mobile phone application allowing the user to measure heart rate anywhere.

**Circuit Board Improvements**

As of now, our heart rate monitor system is neatly installed on a breadboard. However, it would be desirable to reduce the size of the system and enclose the components in a small easy-to-use device.

**Commercialization**

If the the size of the device is reduced and is easily usable, we could potentially produce more of our systems and sell the product, allowing the buyer to download the mobile phone/computer application for free.

**Work Distribution**

**John Careaga**

* MSP430 Coding
* Written Report
* Circuit Implementation
* Aesthetics (Code and Physical System)
* Purchasing components

**Timothy Chong**

* Mac Application Coding
* MSP430 Coding
* Written Report
* Circuit Implementation
* Component Purchases on Sparkfun

The work distribution on this project is spread very evenly between the two of us.