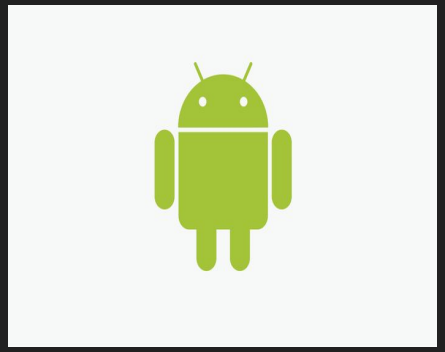
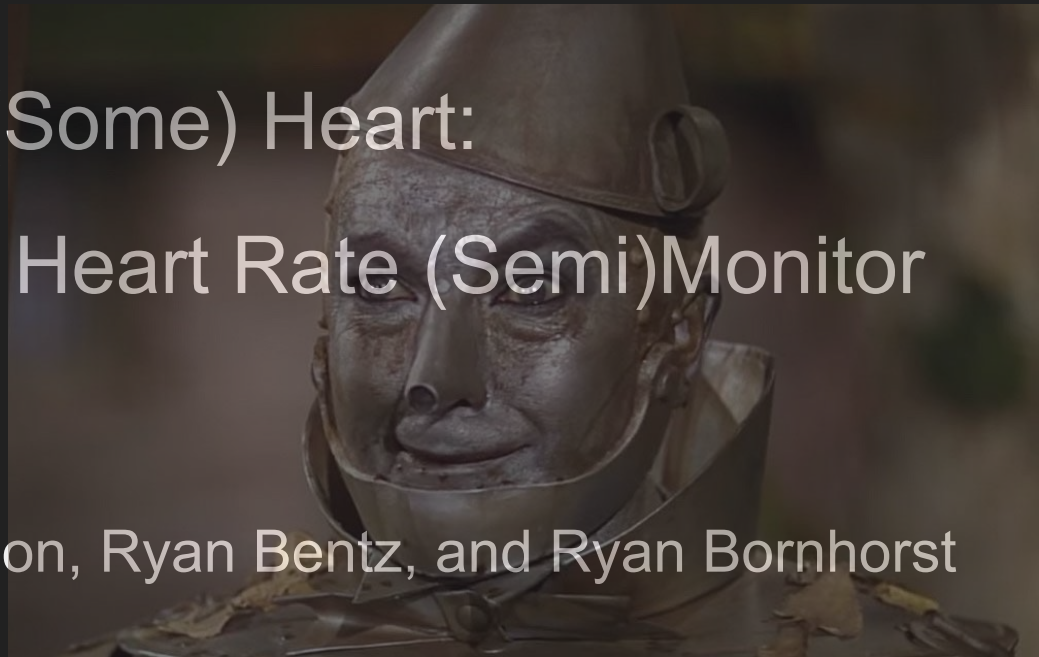


ECE 540
Final Project
Winter 2019

I’ve Got (Some) Heart:

A SoC (Not) Wireless Heart Rate (Semi)Monitor



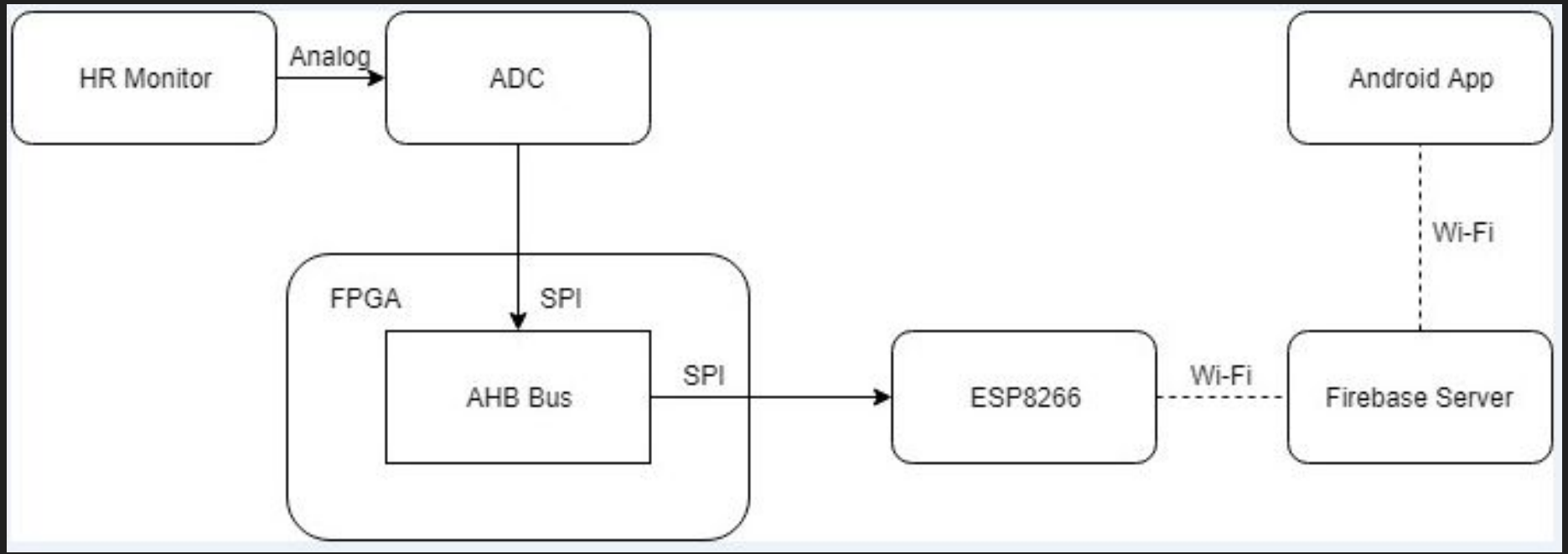
By: Andrew Capatina, Alex Olson, Ryan Bentz, and Ryan Bornhorst

Team Member Roles

- Andrew Capatina
  - mipsFPGA software program
  - Hardware timer
- Alex Olson
  - Heart rate sensor serial communication to FPGA
- Ryan Bentz
  - WiFi communication to Firebase
  - Android app
  - FPGA communication with ESP8266
- Ryan Bornhorst
  - FPGA communication with ESP8266
  - WiFi communication to Firebase

Hardware Block Diagram

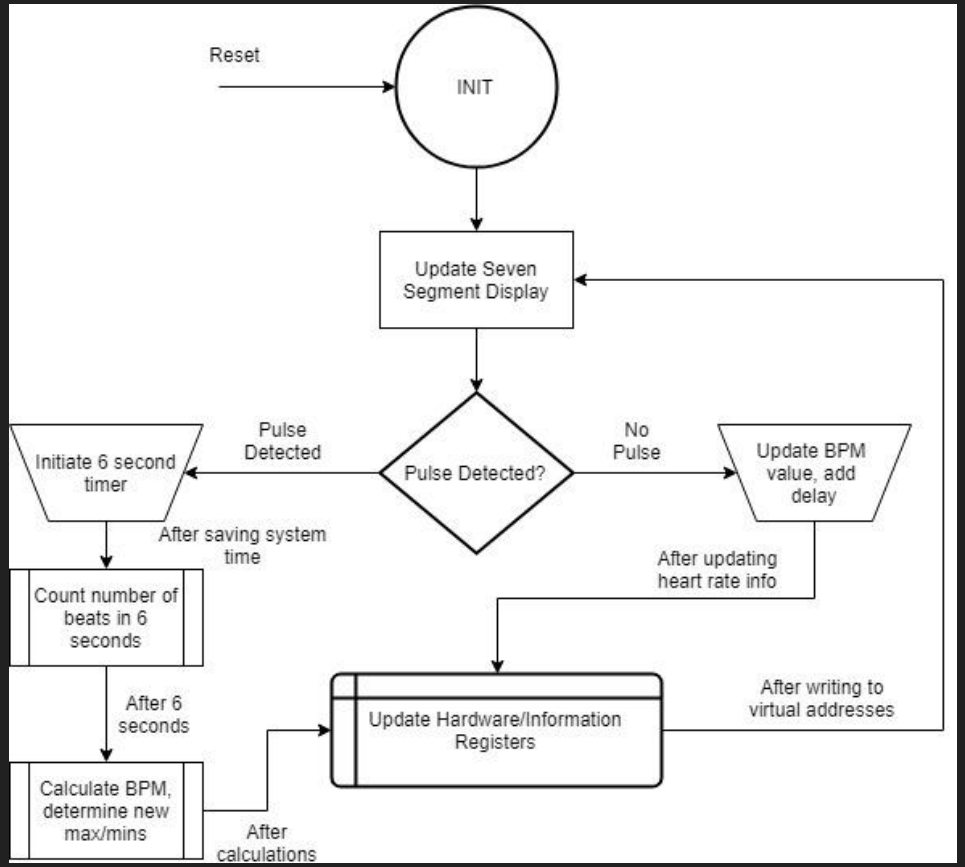
- Inputs: analog signal from HR monitor
- Outputs: HR data in beats per minute
- Displayed on 7-segment display and Android application through Firebase server



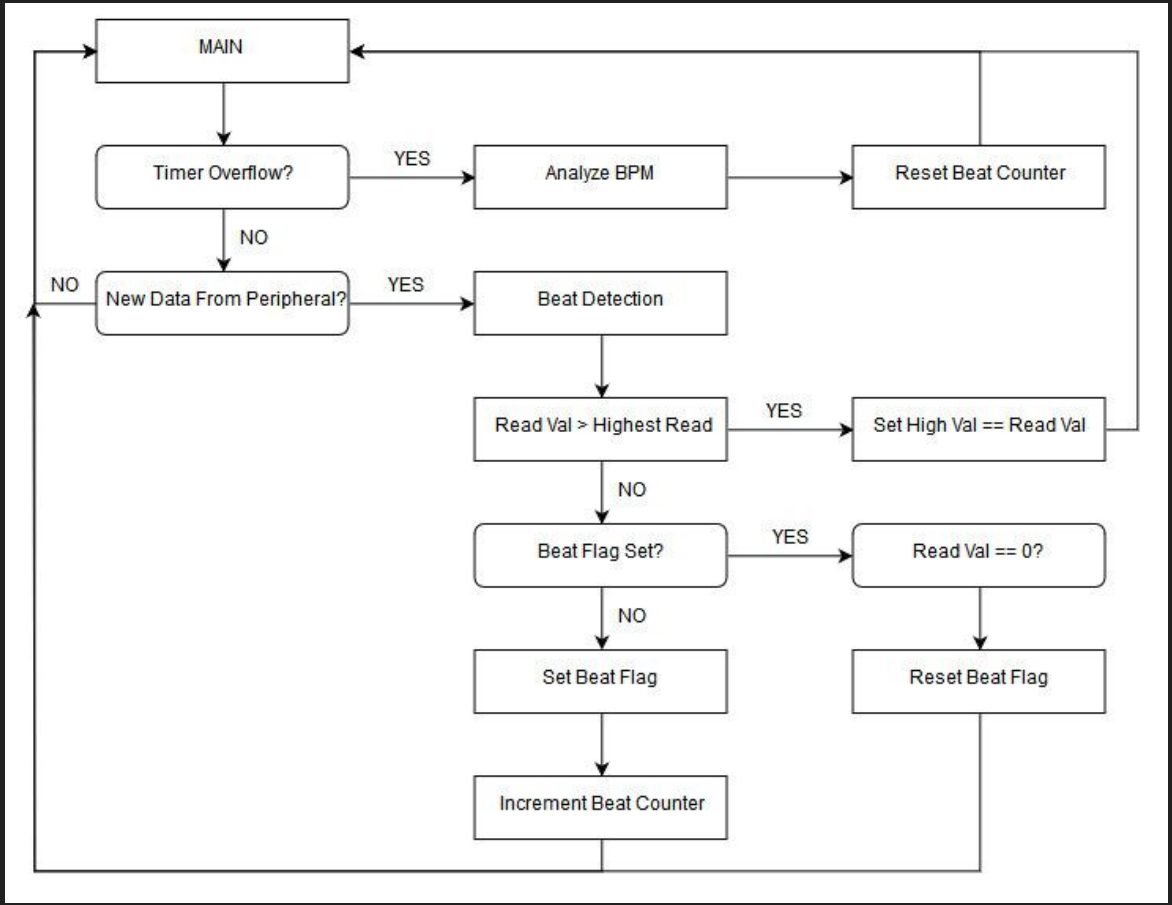
Software: General Information

- FSM
  - Primarily based on heart rate input.
  - Pushbuttons determine what kind of data is displayed.
    - Average BPM
    - Minimum BPM
    - Maximum BPM
- Software/Hardware:
    - Wifi peripheral
    - Seven segment display
    - Heart rate sensor/ADC
    - Pushbuttons
    - Timer

Software: Flow Chart



Beat Detection Algorithm

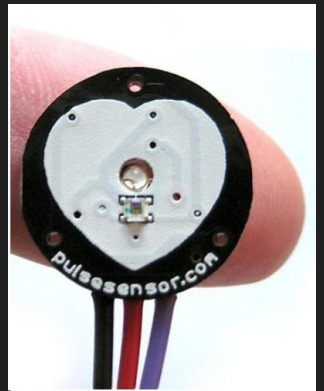


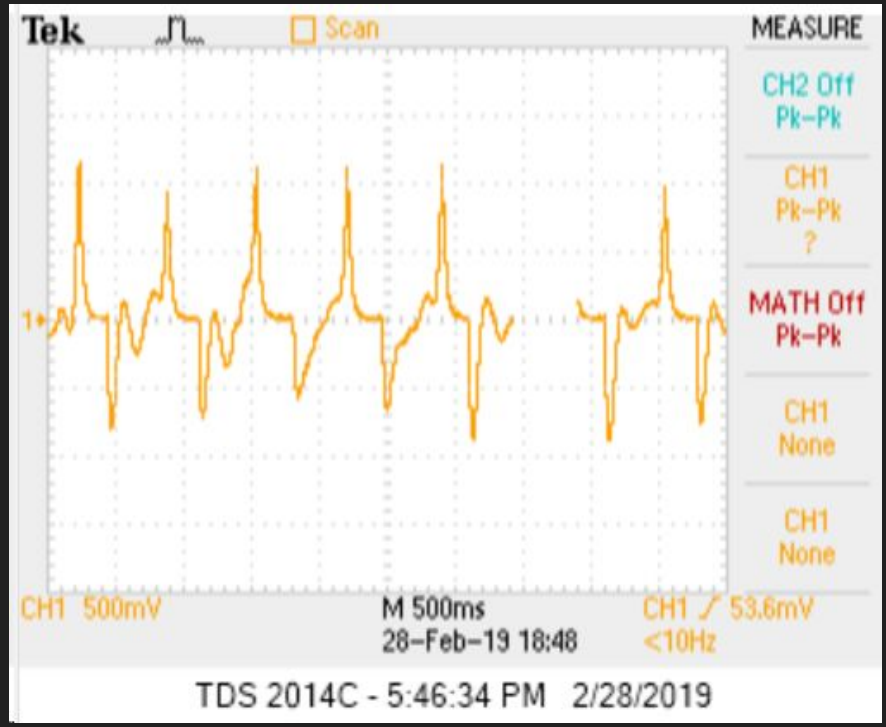
Software: Final Status Report

- What has been completed:
  - High level algorithm/flow chart.
  - Framework for writing simulated heart rate values to the seven segment display.
  - Can cycle through the 3 heart rate statistics that need to be displayed.
  - Timing interval for calculating heart rate.
  - Communication with all hardware peripherals on bus.
  - BPM Calculation
- Challenges
  - Tool chain
    - Debugger not working.
    - Computer crashed, lost some progress.
  - Timing
    - Using a delay counter
  - SPI Communication
    - Understanding what data from ADC implies
  - Balancing time between classes

Heart Rate Sensor

- Analog data built from green LED light reflected off skin and onto a light sensor
- Analog output from pulse sensor
- Converted by 10-bit ADC MCP
- SPI output from ADC into FPGA





ADC Timing Requirements

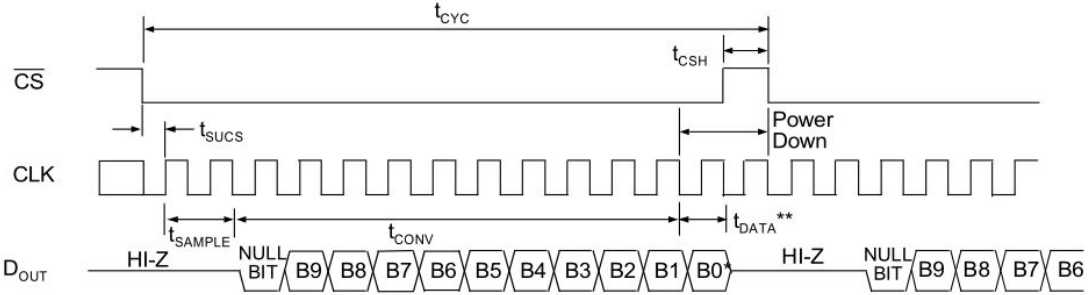
First 2 clock cycles sample the analog data

The next cycle is a null bit

10 Cycles of data

MSB is sent first

1.05MHz @ 3.3V



* After completing the data transfer, if further clocks are applied with $\overline{CS}$ low, the ADC will output LSB first data, followed by zeros indefinitely. See Figure below.

** t_{DATA}: during this time, the bias current and the comparator powers down and the reference input becomes a high impedance node.

ADC Input Voltage

From ADC datasheet $$LSB\ Size = \frac{V_{REF}}{1024}$$

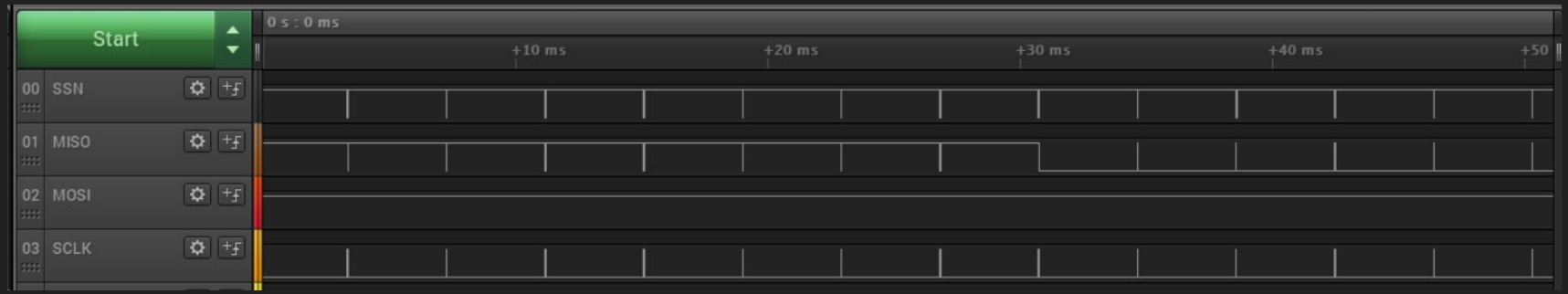
Measured heartbeat sensor voltage: 20mV - 30mV

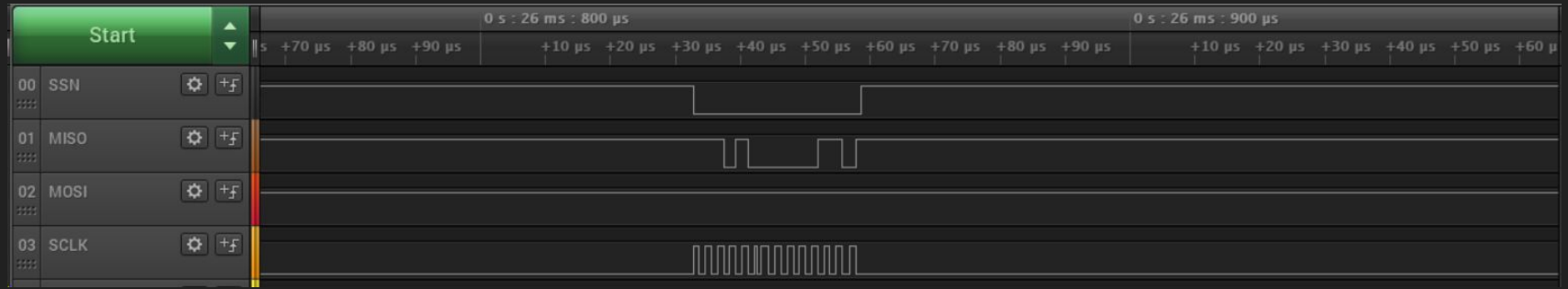
An LSB size of 1mV sounds like a reasonable resolution → V_{REF} = 1.024V

Measured FPGA electrical data: 3.338V driven from PMOD ports with 1.365mA

R = ΔV / I = (3.3V - 1.024V) / 0.001365A = 1667Ω → **1.5kΩ**

ADC Output

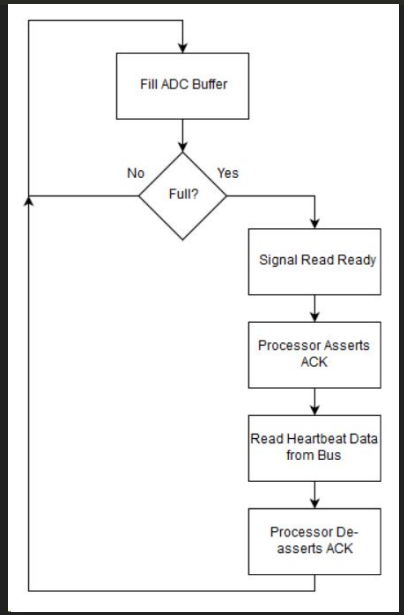

Start
0 s : 0 ms
+10 ms
+20 ms
+30 ms
+40 ms
+50
00 SSN
01 MISO
02 MOSI
03 SCLK


Start
0 s : 26 ms : 800 µs
0 s : 26 ms : 900 µs
+70 µs
+80 µs
+90 µs
+10 µs
+20 µs
+30 µs
+40 µs
+50 µs
+60 µs
+70 µs
+80 µs
+90 µs
+10 µs
+20 µs
+30 µs
+40 µs
+50 µs
00 SSN
01 MISO
02 MOSI
03 SCLK

Clock Domain Synchronization

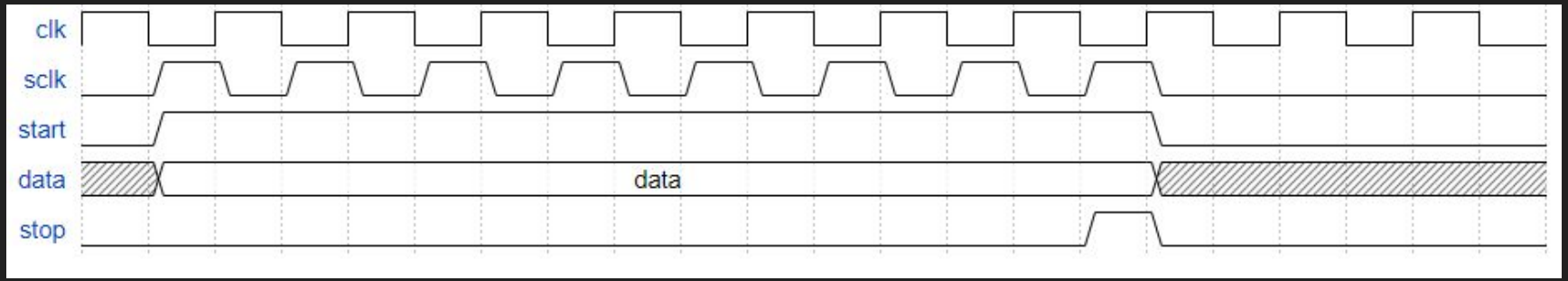
```
// Heart sensor ADC SPI interface
spi_if #(TRANSFER_SIZE) heart_sensor_master (
    .refCLK(refCLK[3]),     // 1.05MHz
    .resetN(resetN),
    .enableN(enableN),
    .din(din),
    .dout(dout),
    .SCLK(SCLK),
    .SSN(SSN),
    .MISO(MISO),
    .MOSI(MOSI)
);

// Drive the reference clock
always_ff @(posedge sysCLK) begin
    if (refCLK == TOPCOUNT+1)
        refCLK <= '0;
    else
        refCLK <= refCLK + 1;
end

// Logic tied to the 8.4 MHz clock domain
always_ff @(posedge sysCLK) begin
    if (refCLK == TOPCOUNT) begin               // Turn off enable after 13 cycles
        enableN <= 1;
        wr_loc <= wr_loc + 1;                   // Increment write pointer after a transfer is over
    end
    else if (refCLK[15]) begin                  // Sample at 128 samples per second
        enableN <= 0;
        wr_loc <= wr_loc;
    end
    else if (wr_loc == BUFSIZE) begin           // Reset buffer pointer when th buffer is full
        enableN <= 1;
        wr_loc <= '0;
    end
    else begin                                  // Else maintain state
        enableN <= 1;
        wr_loc <= wr_loc;
    end
end
```

```
// Data ready logic
always_comb begin
    if (IO_READ_ACK) begin                      // If the cpu acknowledged the signal
        IO_READ_RDY = 0;                        // Reset data ready signal to 0 for next
    end
    else if (wr_loc == BUFSIZE-1) begin         // Else if the buffer is full
        IO_READ_RDY = 1;                        // Signal to the cpu that data is ready to be read
    end
    else begin
        IO_READ_RDY = IO_READ_RDY;              // Else maintain the state
    end
end
```



NodeMCU Peripheral

- AHB-Lite Peripheral
  - Provides registers for application program to write data to
  - Reads HR data off the AHB bus
  - Serial interface writes data to the NodeMCU
  - Sends out data with a synchronized clock signal to the NodeMCU



NodeMCU Peripheral

- Challenges faced:
  - Had a lot of trouble getting valid data on the I/O pins
  - Not able to find a good SPI interface on the Arduino/NodeMCU side
  - Tried to create our own interface for sending data

- Proposed Solution:
  - Created our own interface to communicate between the peripheral and NodeMCU
  - NodeMCU uses interrupt to detect synchronized clock signal
  - Serial data gets sent out with the clock for 2 byte transactions
  - Still not working as intended

WiFi Interface

- NodeMCU:
  - ESP8266 based development board
  - Open-sourced Lua based Firmware which runs on the ESP8266
  - Implemented in C and layered on the Espressif NON-OS SDK
  - Dev board exposes wealth of peripheral capability:
    - GPIO, I2C, SPI, Timers, UART, USB-to-TTY

- ESP8266 Core for the Arduino IDE
  - Arduino library that handles the ESP8266 TCP/IP connections to the network

- Firebase Arduino
  - Arduino library that handles Firebase Connectivity and I/O

WiFi Interface Implementation

- Interrupt-based serial interface
- On the rising edge of the clock, sample the data line to get a bit
- Shift bits into 16-bit word and split into two bytes
- First Byte: Tag identifies what kind of data it is
- Second Byte: Data for the identified tag

Android App

- Single Activity application
- Listens for changes to values in Firebase and updates display
- Real-time beat flag triggers heart beat animation on app

Challenges

- Understanding and using the NexysA7 PMOD pins
- Vivado usage and debugging
- Git and file Management
- Projecting workload and degree of difficulty
- Time management

DEMO

Thank you!