McGill University

Faculty of Electrical, Computer and Software Engineering

MICROPROCESSOR SYSTEMS
ECSE 426

Wireless Mouse

Final Project

Group: 09

Sharhad Bashar :	260519664
Stephan Greto-McGrath :	260535670
Arthur Fabre :	260522364
Anthony Ubah :	260561718
Hassan Ali :	260479064

December 5^{th}, 2016

Contents

1 Abstract

For this final project, the goal was to develop a wireless mouse system based on two STM32F4 Discovery boards. The first board acts as a mouse, sending data to the second board via wireless communication. The second board is connected to the desktop using a USB connection and translates data received from the first board into mouse pointer actions. This board also monitors the connection status of the wireless mouse. Mouse functionality is handled through a combination of pitch and roll of the built in accelerometer as well external push buttons connected to the board. The final result is a wireless mouse system that is functional and can move in any direction in the computer screen's plane, right and left click, and scroll up and down.

2 Problem Statement

In order to create a wireless mouse with the STM34F4 Discovery board, the system must take input from the user which must be translated into a USB mouse input as well as find a way to operate this new mouse design wirelessly. In order to do, two boards must be used. One must be implemented as a sender, and the other as a receiver and they must communicate to each other using the SPI interface and the cc2500 wireless module. The user input must be generated by the pitch and roll of the on board accelerometer and clicks (left, right and middle) using the push buttons connected on GPIO. The mouse must also be able to scroll. Furthermore, the receiver should display the status of the connection with the sender (i.e if it is receiving any data) using the 7 segment display.
As most of the functionality for reading from the accelerometer, displaying on the 7 segment display etc. was already developed in the previous labs, it can be combined with the USB base project creating a codebase that leaves only a few tasks to be completed. These tasks include transforming the accelerometer data into mouse input, creating a method to read input from the pushbuttons, setting up the SPI interface and communicating wirelessly between sender and receiver. An ergonomic and usable physical design must also be considered.

3 Theory and Hypothesis

3.1 Serial Peripheral Interface

The Serial Peripheral Interface (SPI) bus is a synchronous serial communication interface used for short distance communication between peripherals. The SPI communicates with a single master device and with one or many slave devices. At each clock cycle, the slave sends the required data on the MOSI line, while the Master

sends data along the MISO line, as seen in Figure 1 [1]. To continue, the Master is the one that drives the clock. When the Master is receiving data, it needs to send dummy bytes to drive the clock. The slave reads the dummy bytes and sends data.

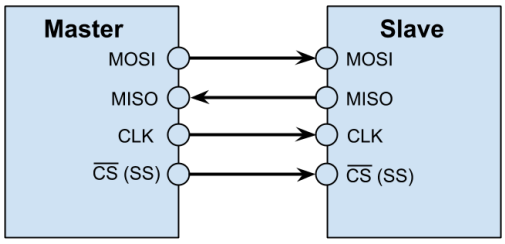

Figure 1: An SPI setup between a Master and a Slave

3.2 Wireless Communication

3.2.1 CC2500 Wireless Transmitter

Once the SPI protocol is configured, i.e., ready to read and write data to the wireless transmitter, then the chipset can be configured. The chip's settings will determine the characteristics with which the pair of wireless transmitters will communicate with each other. Most of the settings can be configured using the default values provided in the CC2500 settings file. In order for the two transmitters to communicate with each other while minimizing possible interference from other transmitters, the channel number must be set to identical values on both the transmitter and receiver chip.

The Wireless Communication system is comprised of a number of subsystems which implement the wireless communication system between the two boards. The system draws from the following concepts:

- Device Drivers
- Protocol Design
- Packet Evolution

The theory involving the wireless communication is very specific to the system and reliant on the TI CC2500 chipset and is discussed along with the implementation of the wireless communication system in Implementation section. According to the project specifications, teams had to set the radio frequency according to the formula:

$$f = 2400 + 8 * (\text{group\#}) \text{ MHz}$$

Team number 9 was used and as such, frequency was found to be 2.472 GHz, which was then converted using the formula from the datasheet:

$$F_{carrier} = \frac{f_{xocs}}{2^{16}}[23:0]$$

This was converted to Hexadecimal to be written in pairs of 2 bits to the FREQ2, FREQ1, FREQ0 Registers as specified in the datasheet.

3.2.2 Strobe Commands

The CC2500 chipset has an internal Finite State Machine (FSM) that controls the state and operations wherein the chipset performs. To be able to send instructions and change the state of the FSM, a strobe command is sent via SPI. Some of the common strobe commands are shown under Table 1 [2]. The simplified state diagram is shown in the Appendix.

Address	Strobe	Description
0x30	SRES	Reset chip.
0x34	SRX	Enable RX. Perform calibration first if coming from IDLE and $MCSM0.FS\_AUTOCAL = 1$.
0x35	STX	In IDLE state: EnableTX. Perform calibration first $MCSM0.FS\_AUTOCAL = 1$. If in RX state and CCA is enabled: Only go to TX if channel is clear.
0x36	SIDLE	Exit RX / TX, turn off frequency synthesizer and exit Wake-On-Radio mode if applicable.
0x3D	SNOP	No operation. May be used to get access to the chip status byte.

Table 1: Wireless transmission commonly used strobe commands

3.3 Threads

In order to minimize the active time of the CPU, threads can be used. If set up on hardware timer interrupts, they can be reliable triggered at set intervals, setting up a frequency of operation. If configured correctly with minimal interrupt service routines (ISRs), they can leave the CPU idle for the majority of the time the device is in use (this is because the code for this activity can execute in a very small amount of time). For more information, the reader can refer back to a more in depth documentation provided in the report for Lab 4.

3.3.1 Thread Communication

In order to ensure that threads can communicate, set and get methods should be used in order to change and retrieve global variables in specific c files that must be used in another file's functions. In order to make sure that these variables are not the source of conflict when multiple threads attempt to access the same space in memory, semaphores can be used. A detailed description of thread protection using semaphores can be found in the report for Lab 4.

3.4 Mouse Human Interface Device Driver

The Human Interface Device (HID) includes 2 entities, namely, the host and the device. The host receives input from the device and displays the result using a mapping. Devices define their data in the form of packets and then present a HID descriptor to the host. The descriptor input includes how many packets the device supports, the size of the packets, and the purpose of each byte and bit in the packet. The host then reads the HID descriptor, retrieves the data packets, identifies what kind of action the device meant to do and subsequently performs it. [3]

3.5 Push button

The Push buttons are breadboard friendly and have 4 pins. They are single-pull-single-throw (SPST) as opposed to single-pull-dual-throw (SPDT). The difference between the two can be seen in Figure 2. In other words, they select between an open circuit and a short circuit through the switch.

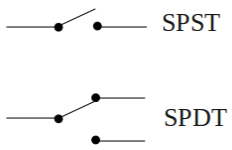

Figure 2: SPST vs.SPDT

All switches fall in either a maintained or momentary categories. The Push buttons dealt with this project are momentary switches and are rated up to $50mA$.As shown in Figure 3, the top two pins are connected, as well as the bottom two. One of these sets of pins are then connected to a voltage source (3V) and the other set to the board. Pushing the button closes the circuit, and allows current to flow from the voltage source to the board. This is picked up as a signal indicating the switch is pressed on a GPIO set as a pulldown input.

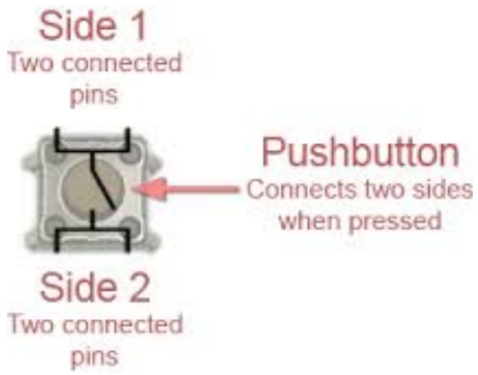

Figure 3: Push button diagram showing the pin connections

3.5.1 Maintained Switches

Maintained switches function like light switches. Once activated, they remain in that state until acted upon again. These switches are also called ON/OFF switches [4].

3.5.2 Momentary Switches

Momentary switches remain active as long as it is acted upon. Once no longer actuated, the switch cuts the connection [4].

3.6 Interrupts

Interrupts on the Discovery board used in this lab are defined the the startup file. These can the be detected once added to `stm32f4xx_it.c` and can trigger a callback which can be implemented where the developer desires. The advantage of this approach is that it greatly reduces development efforts and increases portability between hardware. [5]

3.6.1 Hardware Timers

Hardware timers can be configured to fire after a certain count of internal clock ticks. Hardware timers TIM2, TIM3 and TIM4 are connected to $APB1$ and configured to run at frequencies of $100Hz$, $5Hz$ and $500Hz$ respectively. The frequency of TIM interrupts can be set using the following formula [6]:

$$DesiredTimerFrequency = \frac{TimerInputFrequency}{Prescaler * Period} \tag{1}$$

3.6.2 EXTI Interrupts

EXTI interrupts are edge triggered interrupts on a GPIO pin. [5]

3.7 Accelerometer

For all information on the theory and calibration of the accelerometer, please refer to the documentation in Lab 4

3.8 Hypothesis

It is hypothesized that using the codebase already developped, and implementing SPI communication and the CC2500 wireless module, the team will be able to translate raw date from the accelerometer and GPIO pushbuttons from one sender board to one receiver board and input this information in the for of an HID connected to a computer input allowing the user to use the STM32F4 Discovery board as a wireless mouse with its connection status shown on the 7 segment display.

4 Implementation

The design process began by identifying the needs of the end user (mouse clicks, movement, etc.). Once these needs were identified, functional requirements were developed to allow the system to perform according to the user's specifications (this can be referenced in the preliminary design document). Subsequently, a high level design was developed which took into consideration these functional requirements.

4.1 High Level Design

Figure 4 is a high level block design of the wireless mouse-like computer interfacing device. In each block is a brief explanation of the moduleâĂŹs purpose. The flow of data is represented by arrows.

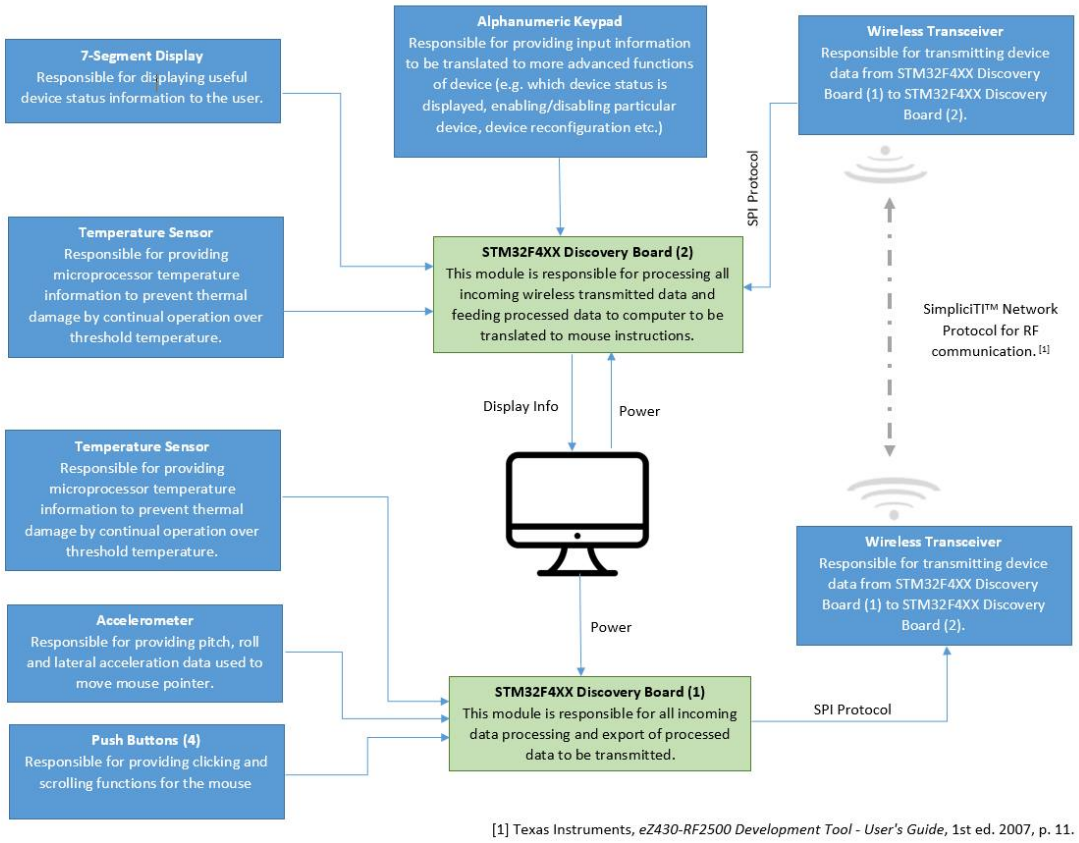

Figure 4: High Level Device Design

4.2 Hardware Overview

4.2.1 Sender

The mouse sender was implemented using a breadboard connected to the STMF32 board. Push buttons were added to the breadboard in order to implement the left, right and middle click buttons and the cc2500 wireless transmitter was also attached using double-sided tape. All of these were interfaced with the board using the GPIO pins. Wires were used instead of jumpers so that they could hug the breadboard and not rise above it, pressing into the users hand during operation. This ergonomic design allows for the mouse to be more easily manipulated making possible finer control. This can be seen in Figure 5. A pinout diagram for the connections can be seen in 6.

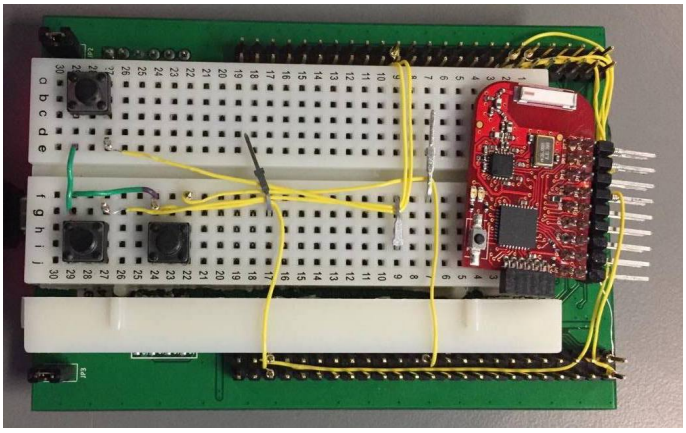

Figure 5: Sender

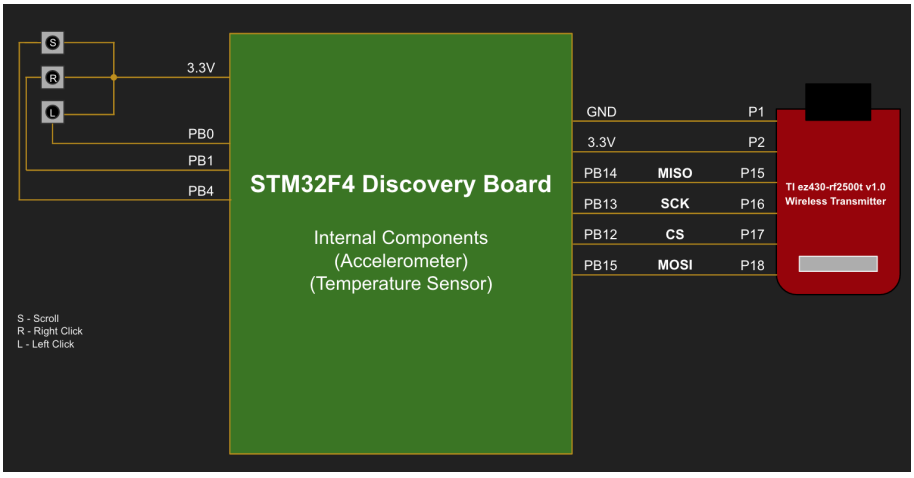

Figure 6: Sender Board Pinout

4.2.2 Receiver

The receiver consists of an STM32F4 board connected to the 7 segment display and keypad all of which can be seen in Figure 7. The CC2500 wireless module was also attached (not shown). The keypad, while completely functional, was not used in the final design as the decision was made to keep the display limited to the device status and not to toggle between the display of the values received. All of these components were connected via GPIO. These connections can be see in Figure 8.

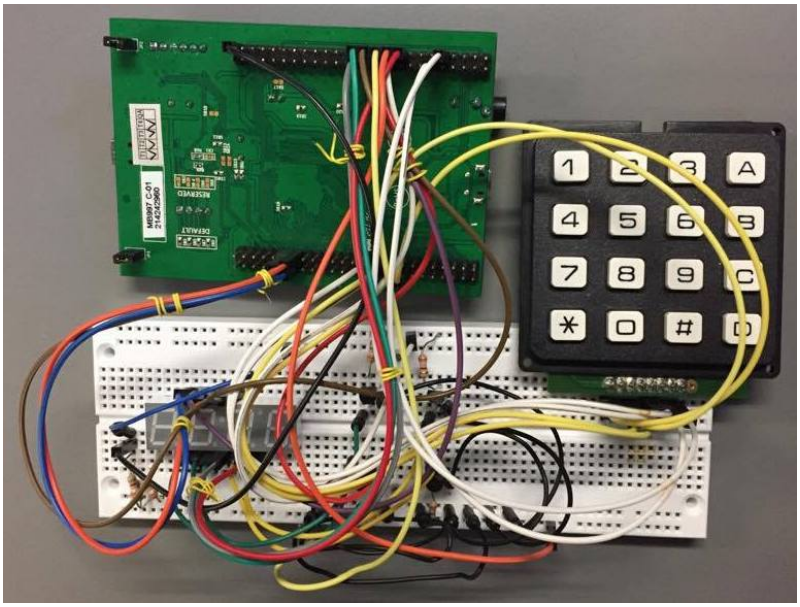

Figure 7: Receiver Board

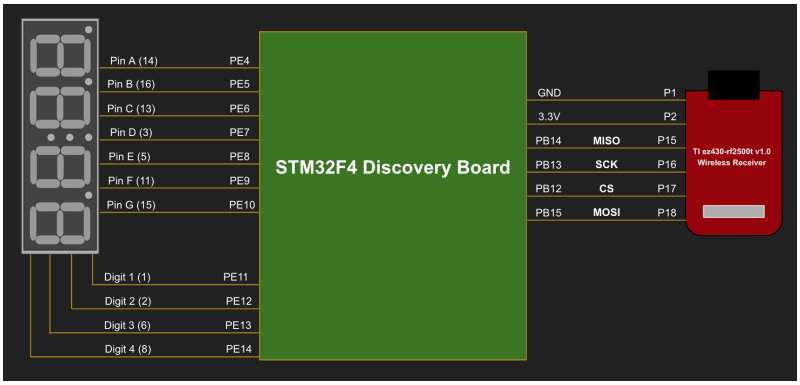

Figure 8: Receiver Board Pinout

4.3 CC2500 Wireless

The CC2500 drivers provide hardware abstraction so that a user can easily exchange data packets between devices wirelessly using the chipsets. In order to use the TI CC2500 chipsets HAL driver were implemented for which the details are elaborated below in this section. The code can be found in cc2500.h and cc2500.c. The following tables (Tables 2-4) provide descriptions of the functionality.

Table 2: Initialization Functions

Functions	**Description**
void cc2500_Init ()	- Initialize CC2500 and SPI - Provides high level device initialization - Enables clock to the required peripherals, including GPIOB and SPI - Sets select line to active or low according to devices requirements - SPI is configured to support and meet the peripheral requirements
void cc2500_Chipset_Init ()	- Write configurations to CC2500 Chip
void cc2500_SPI_Init ()	- Initialize SPI Settings for SPI2 - Enables HAL SPI2 clock
void cc2500_GPIO_Init ()	- Initialize GPIO pins for SPI setup - Pin mapping done in the header file - Sets SPI pin configuration for: – SCK – MOSI – MISO – CS

Table 3: Functions for Receiving or Transmitting

Functions	**Description**
void cc2500_ReadPacket (uint8_t* packet)	- Receive a packet over wireless channel - Pointer to beginning address of packet - Use the Read to strobe from 0x35 - If RX is full, flush it - If buffer has a value use Read to get the value at 0x34 - Use the Read to strobe from 0x7B
void cc2500_SendPacket(uint8_t num)	- Send a packet over a wireless channel - Pointer to the begenning address of a packet - Use the read to strobe from 0x35 - Use Write to put data on the TX resister - Flush TX if it is in underflow state

The two functions above were derived based on the diagram provided in the datasheet for CC2500, as shown in Figure 12:

Table 4: Functions

Functions	**Description**
void cc2500_SPI_SendData (SPI_HandleTypeDef *hspi, uint16_t Data)	- Write in the DR register of the chip - This data is to be sent
void cc2500_SPI_SendByte (SPI_HandleTypeDef *hspi, uint16_t Data)	- Returns the most recent received data by the SPIx/I2Sx peripheral
uint8_t cc2500_SendByte (uint8_t byte)	- Sends a byte from MCU to CC2500 - Loop while DR register in not empty - Send a Byte through the SPI peripheral - Wait to receive a Byte - Return the Byte read from the SPI bus
void cc2500_Read (uint8_t* pBuffer, uint8_t ReadAddr, uint16_t NumByteToRead)	- Reads a block of data from the CC2500 - Can be used to read device registers and FIFO - Uses burst mode to send multiple data bytes in a single stream. Takes 3 parameters indicating the address in MCU memory to the write the data the device to read the data from, and the number of bytes to read - Set chip select Low at the start of the transmission - Send the Address of the indexed register - Receive the data that will be read from the device (MSB First) - Send dummy byte (0x00) to generate the SPI clock to CC2500 (Slave device) - Set chip select High at the end of the transmission
void cc2500_Write (uint8_t* pBuffer, uint8_t WriteAddr, uint16_t NumByteToWrite)	- Writes one byte to the CC2500 - Configure the MS bit: – When 0, the address will remain unchanged in multiple read/write commands – When 1, the address will be auto incremented in multiple read/write - Takes 3 parameters indicating the data to be written, the address where the data needs to be written on the device, and the number of bytes to write - Address to the device registers and FIFO are defined in cc2500.h - Set chip select Low at the start of the transmission - Send the Address of the indexed register - Send the data that will be written into the device (MSB First) - Set chip select High at the end of the transmission

4.4 Threading

Threading was configured in a similar way to lab 4. Threads were set up to run on the same hardware timers as in the previous lab. Using set and get methods, the threads could communicate with each other. Within these set and get methods, the globals which could be accessed by multiple threads were protected by a semaphore.

4.5 Mouse (Send and Receive)

The mouse thread contains all the functionality of the sender and the receiver and can be configured to perform either of the two tasks based on a integer input (1 for sender and 2 for receiver). This thread, like the others runs on a hardware timer to free the CPU when it is not needed. These send and receive functions begin on start-up and need not be re-initiated at any point in time.

4.5.1 Sender Mode

In sender mode, the thread polls pushbutton inputs as well as communicates with the accelerometer, retrieving the pitch and roll data to be input through the HID interface. This is done via the the $set\_click(get\_click())$, $set\_pitch(get\_pitch()$ and $set\_roll(get\_roll())$. These functions utilize the get functions that access globals in the accelerometer threads as well as a new function ($get\_click()$), which polls the GPIOs connected the pushbuttons to see which buttons are pressed. Once this data is retrieved, it is transformed into mouse inputs (pitch for left and right movement and roll for up down movement) and then is broadcast wirelessly to be picked up by the receiver.

4.5.2 Receiver Mode

In receiver mode, the thread continually searches for wireless information. It communicates with the display thread via the $set\_display()$ method. If mouse data is being received it displays *data* whereas if no data is being received the display will read *null*. When data is identified and received it is passed into $USBD\_HID\_GetReportTrigger()$, which passes the mouse inputs via USB to the computer allowing the device to perform the mouse functions on screen.

4.5.3 Communication Protocol

At the time of the demo, burst transmission was not yet configured. As such, single byte packets were used to transmit the data wirelessly from sender to receiver. In order to ensure that pitch, roll and clicks are properly identified, their byte value is

padded before and after with a code to identify them. The receiver seeing a sequence of ID_CODE-$INFO$-ID_CODE will then take the $INFO$ byte as a valid input for that ID_CODE which corresponds to a data type (pitch, roll, click).

4.5.4 Mouse Control

As the mouse is suspended in mid air when held by the user, slight pitch and roll are unavoidable. These small movements can be triggered when clicking a button or simply when holding the mouse with an unsteady hand. To correct this and allow for better control of the mouse, a threshold for pitch and roll was implemented. Below this threshold (15°), the mouse would not move. Once above the threshold, the values of pitch and roll would be scaled down to allow for the fine control available when the mouse was moved only a few degrees.

4.6 Status Display

The 7-segment display with 14 pins is used to display the connection status. If packets are being received, then the display will show *data* to say there is a connection. In other hand, if the display shows *null*, then there is no connection. The pins were connected to port D. Multiplexing was used by connecting a transistor to each of these pins, and the transistors were controlled by port E, pins 0 to 3 on the board.

In order to make it appear to the human eye that all digits were being displayed at the same time, each digit was displayed every $2ms$ in the main thread. The display was updated with every loop iteration, thus giving appearance of a flicker-free display. Every digit that was being displayed was being fed into a switch statement which determined which segments of the 7 segment display to turn on. More details on the 7 segment display can be found in reports for previous labs.

5 Testing and Observations

5.1 Serial Peripheral Interface

The SPI protocol was tested by connecting the wireless chipset and continuously reading and writing values to the registers of the transmitter. A loop was used to automate the testing process and, at first,only one data value was written, read back and printed to the screen for each iteration of the loop. If the read value matched the written one, the interface was configured correctly.

5.2 Strobe Commands

Strobe commands were confirmed to be working properly after the commands $SIDLE$, SRX, and STX were sent to their respective registers. After each strobe command was sent, enough time was given for the results to settle, the $SNOP$ strobe allowed the current state of the wireless chipset's FSM to be returned and printed to the screen, which are listed under Table 5.

Address	**Srobe Name**	**State Returned from SNOP**
0x36	SIDLE	IDLE (000)
0x34	SRX	RX (001)
0x35	STX	TX (010)

Table 5: After Strobe Command Test

5.3 Wireless Transmission and Reception

5.3.1 Receiver Mode Testing

Testing of the wireless chip began with receiving data from the beacon which transmitted single digit numeric code. The transmitter was sending data at 2.433 Ghz. Figure 9 illustrates the output that was obtained from the receive function. The transmitter was sending the number 0 to 9. As it is shown in the picture, the mouse was also receiving these values. The following link to a YouTube video shows the receiver in action : link
There was a slight delay in receiving the number, but as time progressed, this delay became unnoticeable.

Additional test for distance:

distance [feet]	**d > 15**	**10 < d < 15**	**d < 10**
transmission [bytes]	0 out of 4	1 out of 4	4

Table 6: Reception Test with varying distance

Conclusion: The system can successfully receive 1 byte at a time.

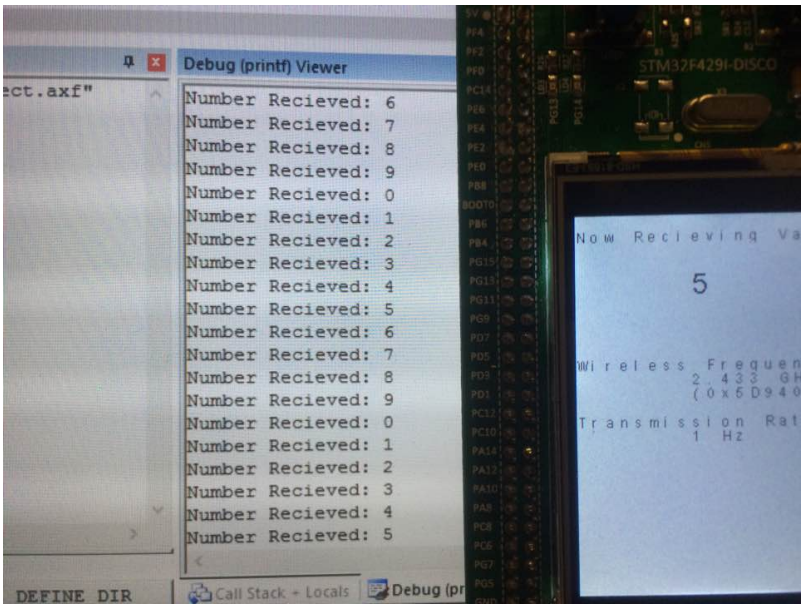

Figure 9: Testing the wireless in receiving mode

5.3.2 Transmission Mode Testing

Following similar steps, the transmitter was also created. It was transmitting at a frequency of 2.433 GHz. The following number sequence was being sent: 3-1-4-2, with 1 second delays in between. Figure 10 displays the numbers that were being transmitting on the screen. The LCD on the board shows that the values are indeed being correctly transmitted. The following YouTube link displays the working transmitter : link

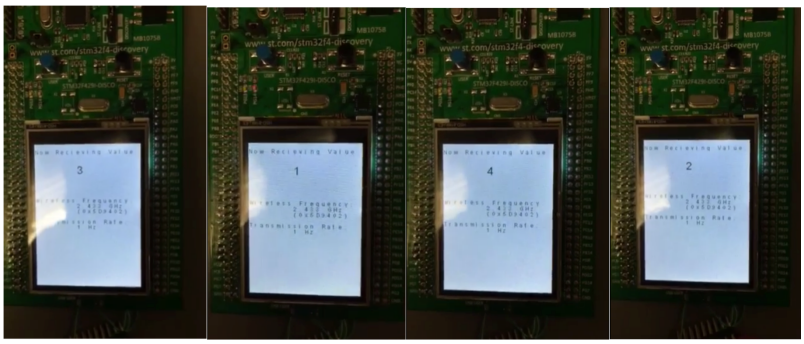

Figure 10: Testing the wireless in transmitting mode

There was a slight delay in the initial setup before the transmissions were being read, but afterwards, it was working as smoothly as the initial beacon that was provided. **Conclusion:** The system are capable of sending 1 byte of data at a time and this for four consecutive times and hence it can be assumed the transmitting system

Number sent	Number displayed by LED
3	3
1	1
4	4
2	2

Table 7: Reception Test with varying distance

works properly for one byte of data.

5.4 Mouse Functionality

5.4.1 Movements

Testing of the mouse movement functionality was conducted in two stages. The first test was to test the motion functionality of the mouse and the second was to test the threshold system which filters out minor movements.

The Discovery board acting as the mouse was held and rotated on its x and y axes. The motion of the mouse pointer was observed and recorded. The mouse moved in sync with the user's motions. During this test, it was the discovered that the mouse pointer responded to fluctuations due to instability of the users grip as well as the slight movement caused by a button being pressed. This led to the development of a threshold system that filters out such fluctuations.

The second test of the mouse movement functionality was to test the threshold system. For this test, the mouse was moved slightly (below the specified threshold) and the mouse pointer on the screen was observed. No movement was observed below the threshold value. This was tested with multiple thresholds in order to determine an ideal point.

The following is a YouTube link to the video of the test conducted: link

Conclusion: The mouse movement is smooth and not prone to fluctuations caused by user grip instability or buttons being pressed.

5.4.2 Clicks

Testing of the clicking functionality of the mouse was straight forward and the same for all 3 of the push buttons. The button was depressed when the mouse pointer was over an appropriate target. The resulting action was observed and compared against what was expected. Also, the button was depressed and held in this state.

The result was a single click being registered.

Conclusion: Clicking functionality of the mouse works as expected.

5.4.3 Scrolling

Testing of the scrolling functionality of the mouse was straight forward and similar to the click tests. The scrolling was tested by depressing the appropriate button and rolling the board forward and backward. The resulting action was observed and compared against the expected action. The initial test was performed in the Keil uVision IDE. Scrolling, the way it was implemented, did not work here because the application does not support this type of scrolling (click and move mouse). However, when tested in other applications such as Google Chrome and Microsoft Edge, scrolling worked extremely well.

Conclusion: Scrolling works well in applications that support it.

6 Conclusion

The final result of the project was a system which met most of the functional requirements outlined in the preliminary design document. The only functionality missing was the full wireless transmission of mouse action data from the mouse board to the receiving board. This was mainly due to the downtown experienced in the McGill servers that handled the distribution of the Keil uVision software. However, the team was still able to achieve wireless transmission of bytes between the two Discovery boards and the implementation of deducing and displaying connection status between the two boards. It was also still possible to achieve a system akin to a mouse when the mouse board was connected to the computer via USB. The system includes all basic capabilities of a regular mouse (i.e., moving the mouse around the screen, scrolling, right-click and left-click). If this project were to be repeated it would be wise to note that the wireless communication between the boards is imperfect at higher frequencies and large volumes of data. This is to say that packets begin to drop when the transmission rate becomes too high or when the mouse board tries to send too much information at a time. Implementing a more robust two-way protocol between the boards would have solved this issue. Despite these issues, the system performs beautifully. With a few upgrades to the form factor (i.e. a more aesthetically pleasing external casing), the mouse can be used as an everyday device.

Bibliography

[1] Spi structure. http://dlnware.com/theory/SPI-Structure. Accessed: 2016-12-02.

[2] Cc2500 low-cost low-power2.4 ghz rf transceiver.

[3] Tutorial about usb hid report descriptors. http://eleccelerator.com/tutorial-about-usb-hid-report-descriptors/. Accessed: 2016-12-02.

[4] Switch basics. https://learn.sparkfun.com/tutorials/switch-basics. Accessed: 2016-12-02.

[5] Ashraf Suyyagh. Stm32f4 cube hal. Tutorial slides, McGill University, 2016.

[6] STMicroelectronics. *STM32F405xx STM32F407XX ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM, USBOTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces camera*, 2016.

Appendix

List of Abbreviations

STM STMicroelectronics
TI Texas Instruments
GHz Gigahertz
LCD Liquid Crystal Display
OS Operating System
SPI Serial Peripheral Interface
MHz Megahertz
PC Personal Computer
API Application Program Interface
MCU Microcontroller Unit
GPIO General Purpose Input and Output
Rx Reception
Tx Transmission
FIFO First In First Out
MOSI Multiple Output Single Input
MISO Multiple Input Single Output
SCK Sending the clock signal
Clk Clock
HAL Hardware Abstraction Layer

Gantt Chart

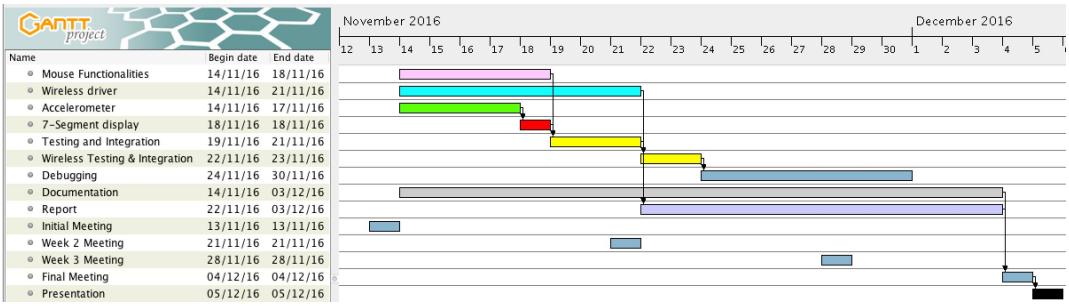

Figure 11: Gantt Chart

List of Figures

List of Tables

CC2500 Finite State Machine

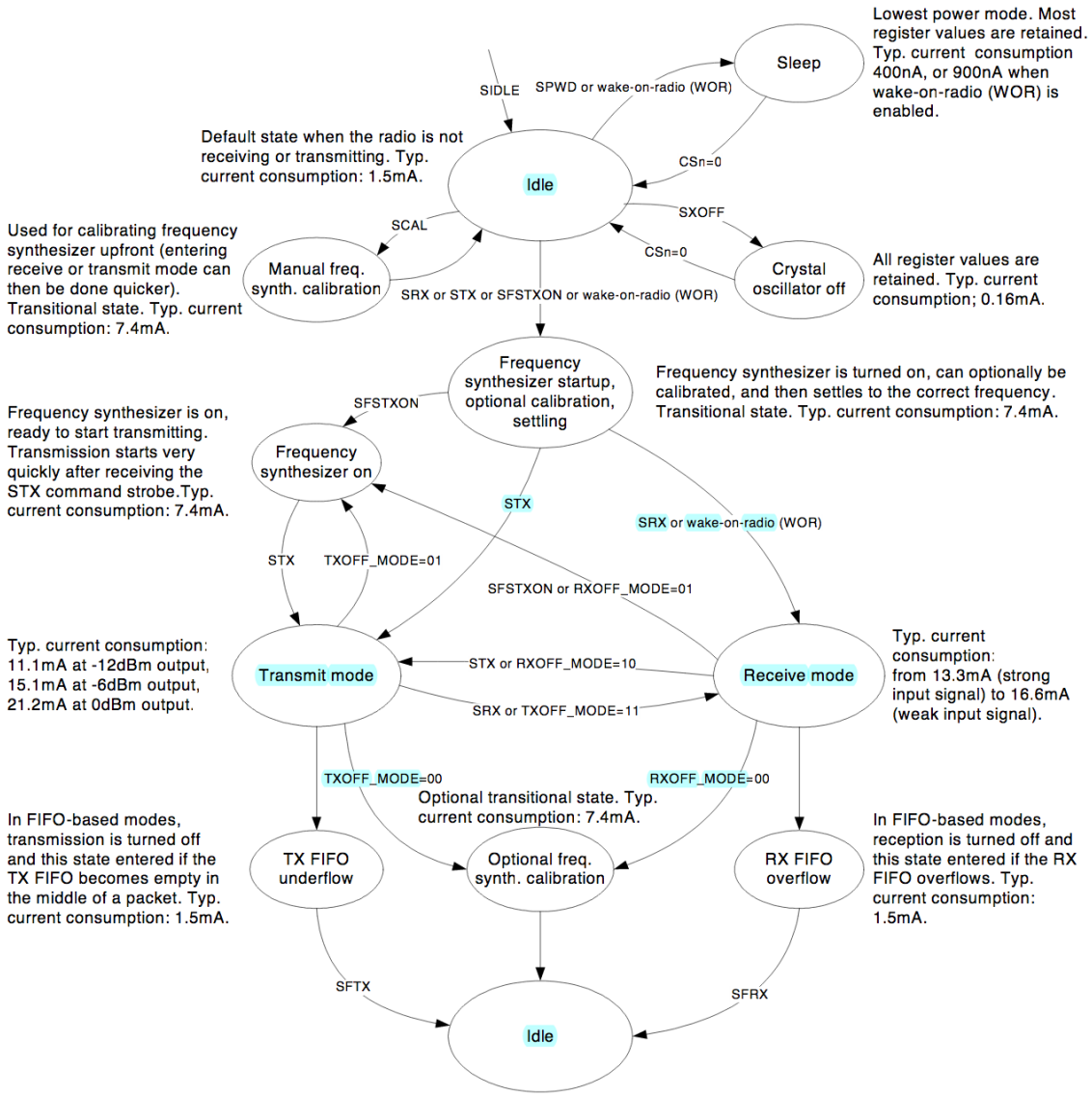

Figure 12: Simple State Diagram for the Wireless Chip