2019

JYU-447V2 IR Protocol

BLUETOOD CONTROLLED CAR AND IR

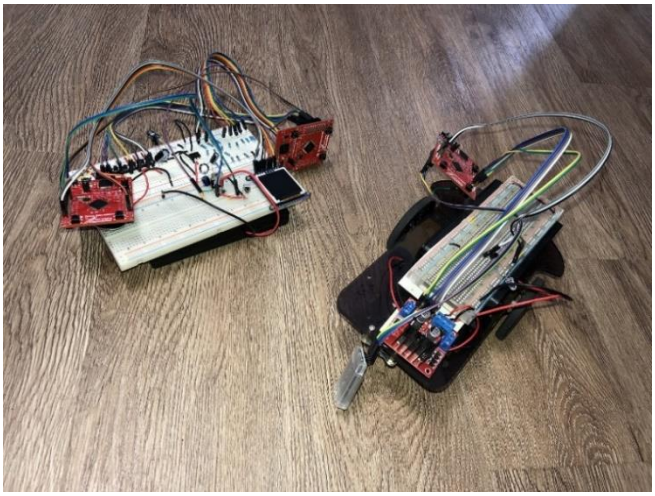
KEVIN LOPEZ

HTTPS://YOUTU.BE/KKXHS1AFPVE

Table of Contents

Table of Figures

Introduction



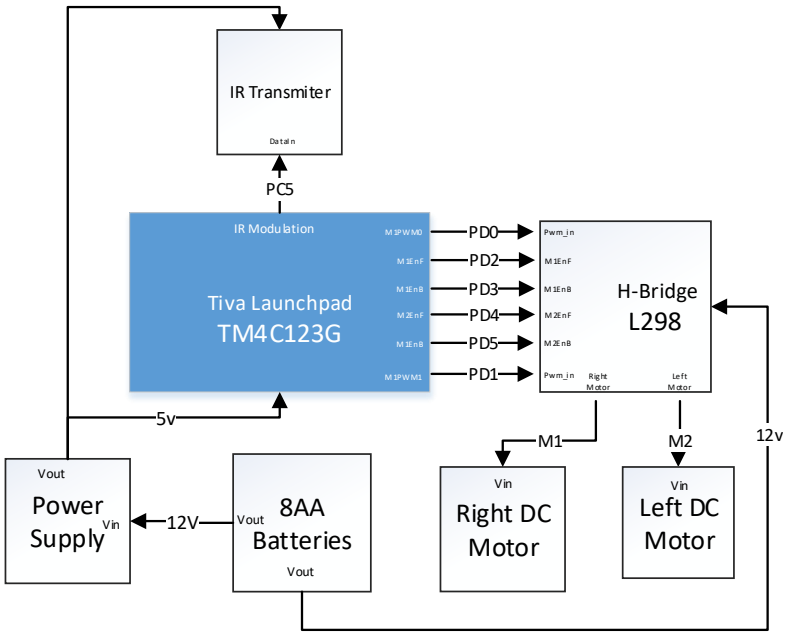
The objective of this project was to create our own IR protocol integrated with a Bluetooth module on a TM4C launchpad. The IR protocols work on a frequency between 38Khz and 40Khz and have a start bit signaling that a new command is being sent. This IR protocol works on a 40Khz frequency and has a 1.5ms start bit, two bits address, and followed by a four-bit Command. This IR protocol is being integrated to a Bluetooth car that transmits to a receiver station. It is composed of an LCD screen, a Digital to analog circuit and an IR receiver module. In the other side, the infrared transmitter is composed of a remote-control car with a Bluetooth module, an IR transmitter, and other components so there could be signals sent and be able to drive such car around.

Operations

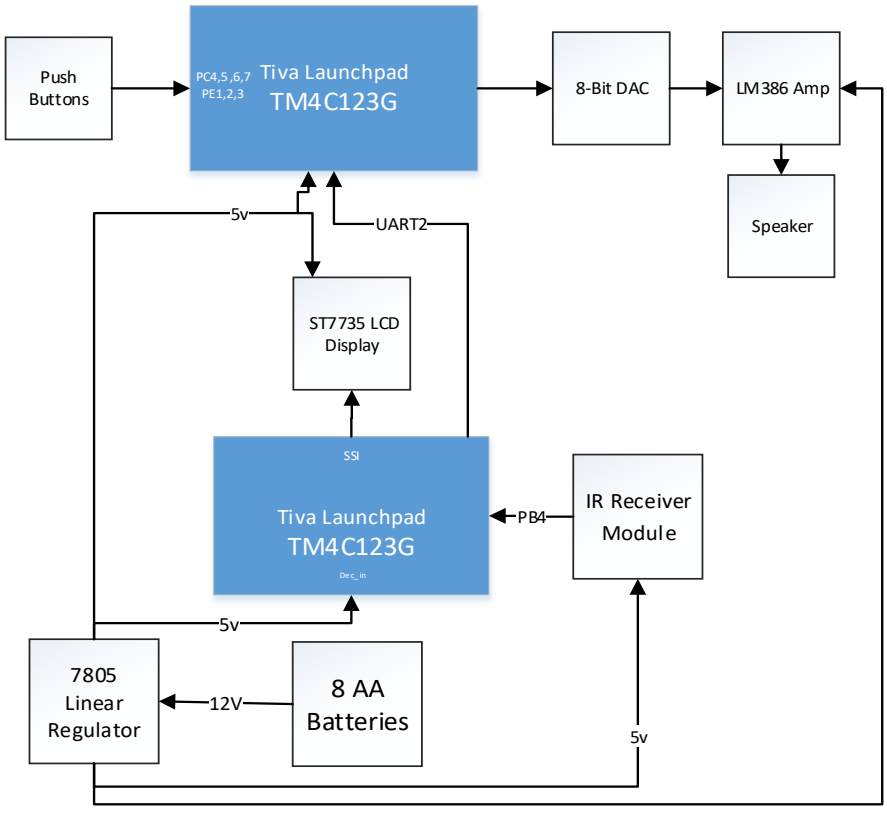
The car has two DC motors to move around and their speed was controlled using Pulse Width Modulation on each individual wheel. It also has the ability to be controlled by Bluetooth then after being positioned to the designed station, we can send IR commands to trigger a specific animation or note coming from our 8-bit DAC. In order to be able to communicate with the IR protocol, we had to use PWM to generate a square wave at 40KHz, then this was controlled by a timer that would change depending if we wanted to transmit a logic 1 or a logic 0. Universal Asynchronous Receiver/Transmitter (UART) was also used in this project heavily to communicate with the HM-10(Bluetooth module) and the DAC created in the past project. Whenever the receiver station received a value that was between 5 and 12, the station would transmit the command to the DAC to play a specific note.

Hardware

Car Hardware Block Diagram

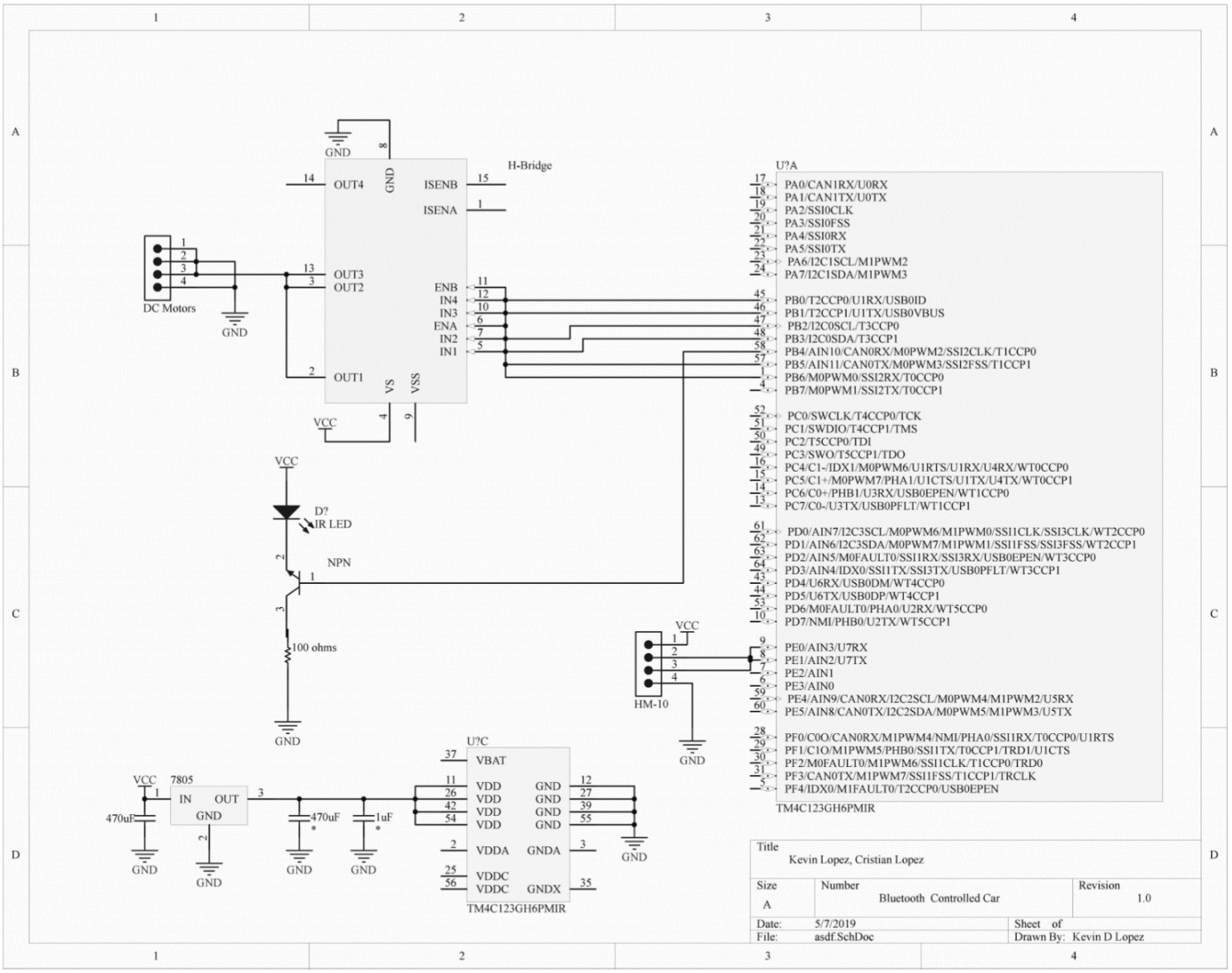


Receiver Station Hardware Block Diagram

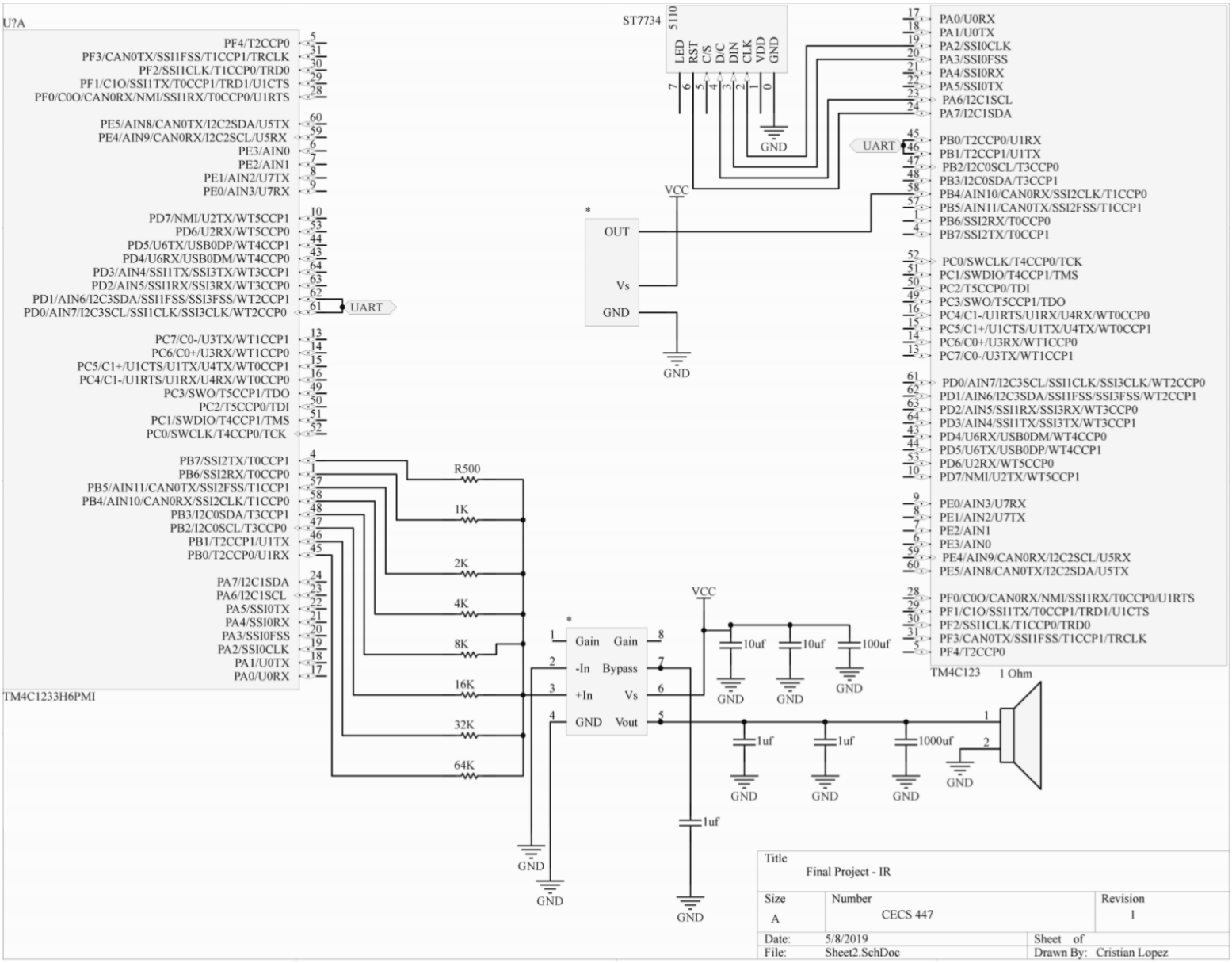


Schematic

2.1 Bluetooth Controlled Car



2.2 Receiver Station & 8-bit DAC



List of Components

- 3 x TMC4123 Launchpad
- HM-10
- EK8443 infrared Emitter led
- IR Receiver module (40Khz)
- Linear regulator (7805).
- ST778 LCD Display.
- L298 H-Bridge.
- 2 x DC motors.
- 12 x AA batteries.
- Transistors.
- LM386 Operational amplifier.
- 1 Ohm Speaker.
- 500, 1k, 2k and up to 64k 1% tolerance resistors.
- Buttons, wires, and capacitors.
- Power supply, oscilloscope and Keil Uvision 4 for debugging.

Hardware Explanation

The whole project circulated around the IR protocol since was the new subject to learn for communication. For such protocol, an EK8443 infrared Emitter led is used hence the low cost. The car was controlled with an HM-10 module that was communicating via UART to a Tiva Launchpad Microcontroller. This module was used since it has Bluetooth 4.0 and works with all Apple Devices. The MCU in the car was programmed so that different characters received from our phone would transmit IR signals to the receiver station, we can see these specific characters in the table below. In the other side, we had a receiving station composed of an IR receiver module and two TM4C123 launchpads communicating with each other via UART. One

of the launchpads demodulates the IR signal and does animations, the other microcontroller is used to control the Binary Weighted DAC and plays the notes.

Bluetooth	Car Action's	IR signal Sent	Receiver Station Action	DAC Actions
w	Moves Car Forward	There is no signal Sent	No Action	No Action
s	Moves car to the Right	There is no signal Sent	No Action	No Action
d	Moves car Backwards	There is no signal Sent	No Action	No Action
a	Moves car to the Left	There is no signal Sent	No Action	No Action
q	Stops car	There is no signal Sent	No Action	No Action
0	No Action	00_0000	Set Animation 1	No Action
1	No Action	00_0001	Set Animation 2	No Action
2	No Action	00_0010	Set Animation 3	No Action
3	No Action	00_0011	Set Animation 4	No Action
4	No Action	00_0100	Set Animation 5	No Action
5	No Action	00_0101	Send Data to DAC 'C'	Play Note C
6	No Action	00_0110	Send Data to DAC 'D'	Play Note D
7	No Action	00_0111	Send Data to DAC 'E'	Play Note E
8	No Action	00_100	Send Data to DAC 'F'	Play Note F
9	No Action	00_1001	Send Data to DAC 'G'	Play Note G
A	No Action	00_1010	Send Data to DAC 'A'	Play Note A
B	No Action	00_1011	Send Data to DAC 'B'	Play Note B

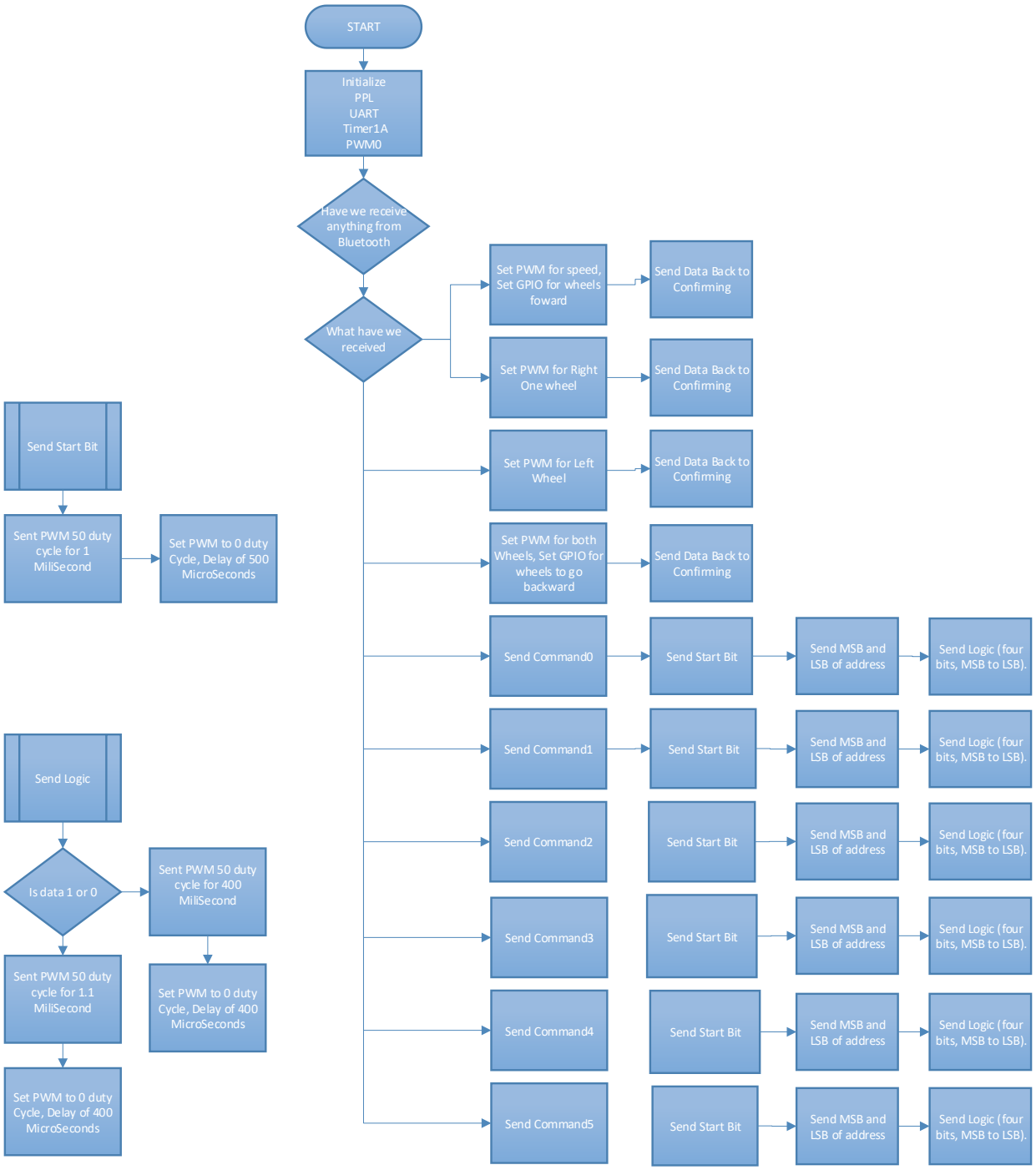
Software

Approach

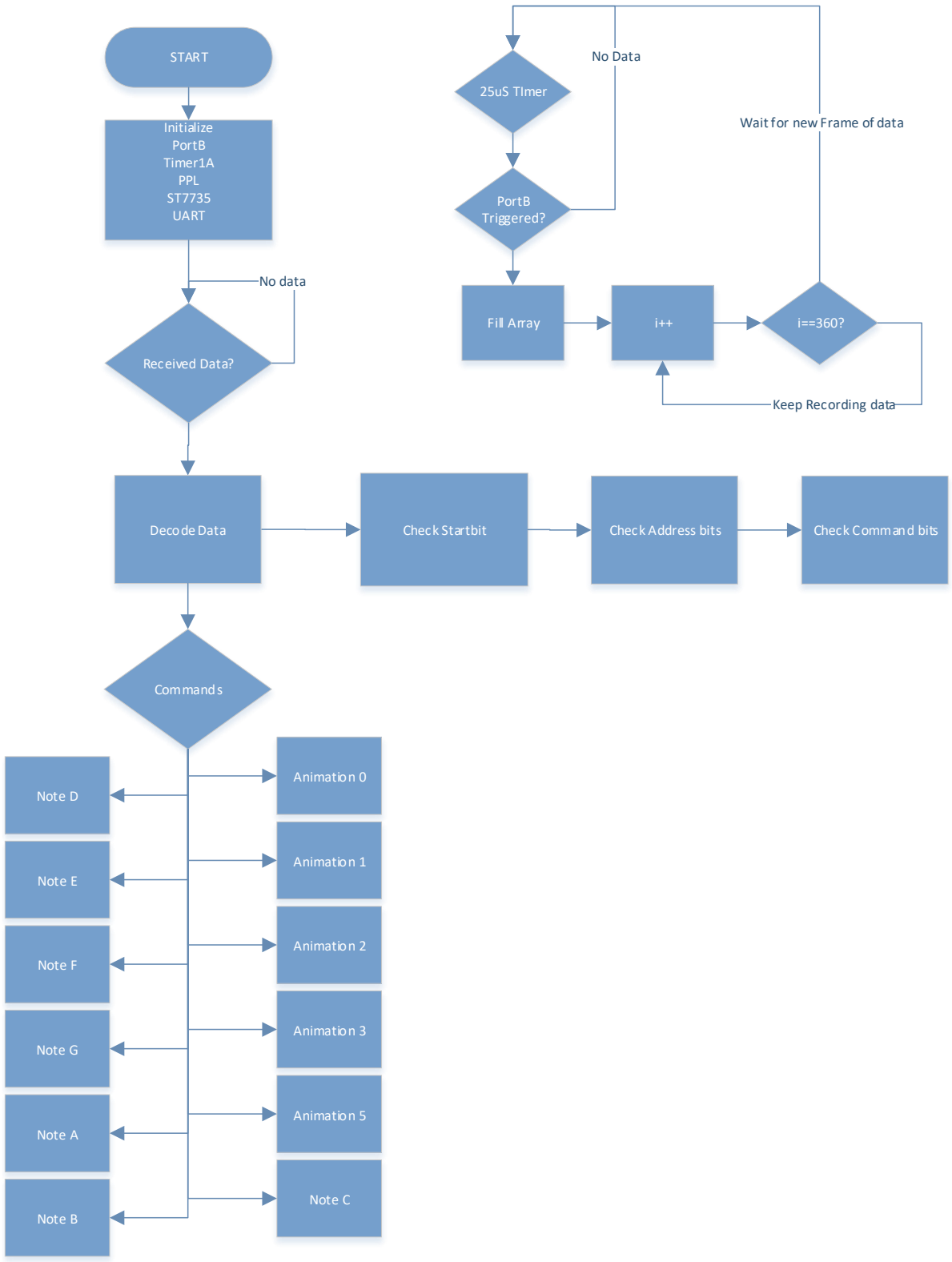
To achieve that goal, we used a HM10 Bluetooth module communicating to our TM4C by UART at a baud rate of 57600, 8 Data bits, 1 stop, and odd parity. Then to achieve the encoding of the IR protocol we used concepts of Pulse Width Modulation and timers. The PWM frequency was set to 40Khz and a timer was used to keep count for how long were sending an actual signal. The decoding also used the concepts of timers but in this case, we took advantage of the difference from a logic 1 and logic 0 which created a low to high level difference of 600 msec. We sampled around this area of time to see the bit we received. After we decoded the data received, we triggered the specific action if the command sent was in the list.

Software Flowchart

IR Transmitter Station

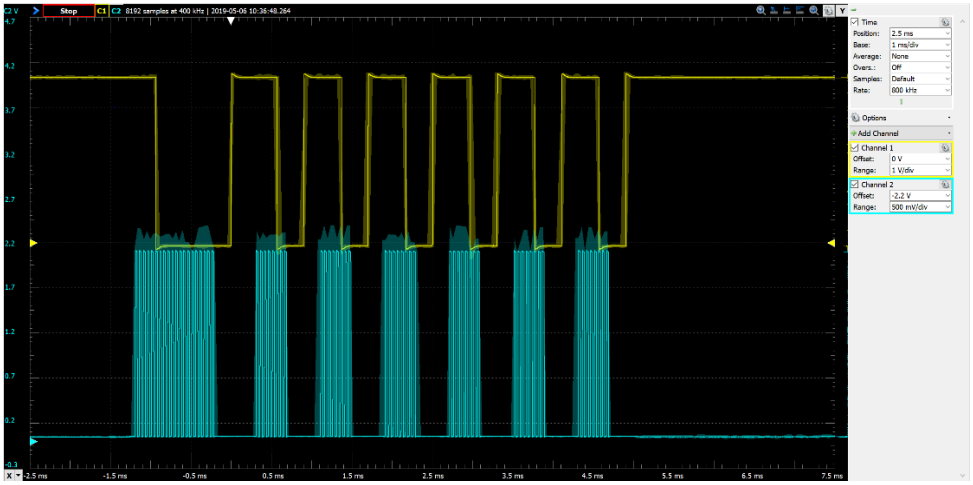


IR Receiver Station

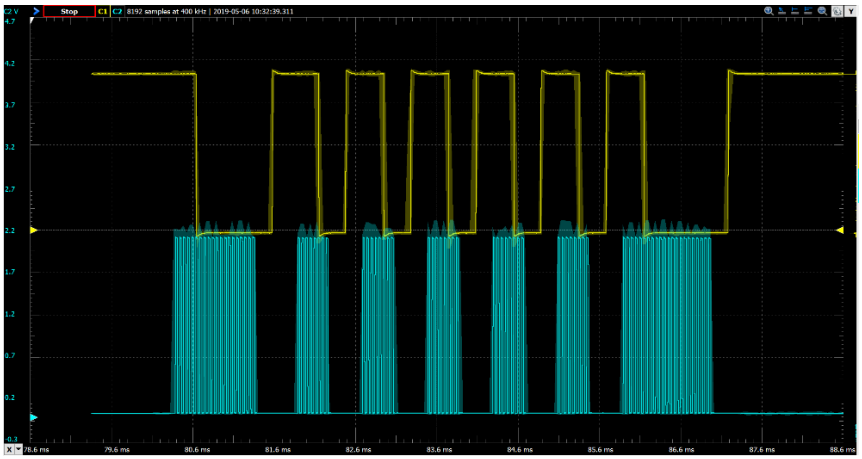


Waveforms

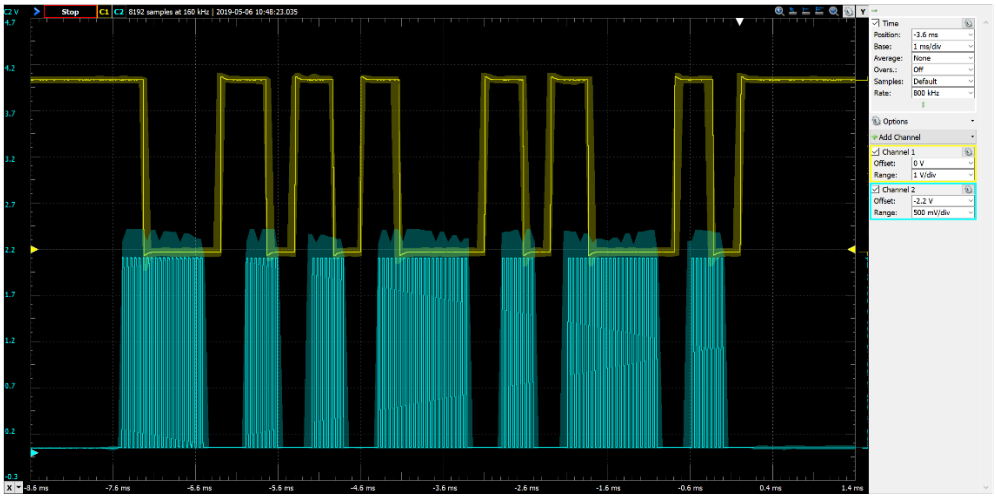
Address 0 Command 0



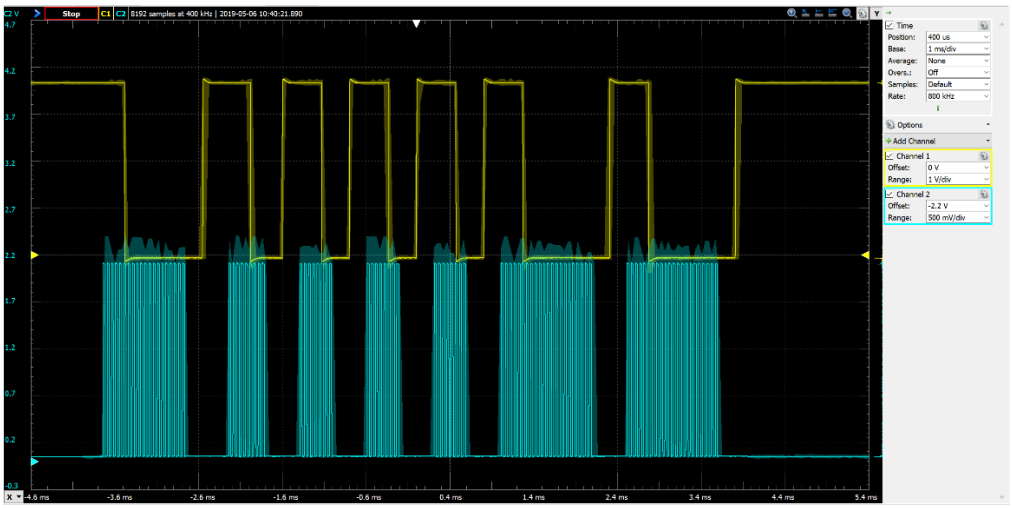
Address 0 Command 1



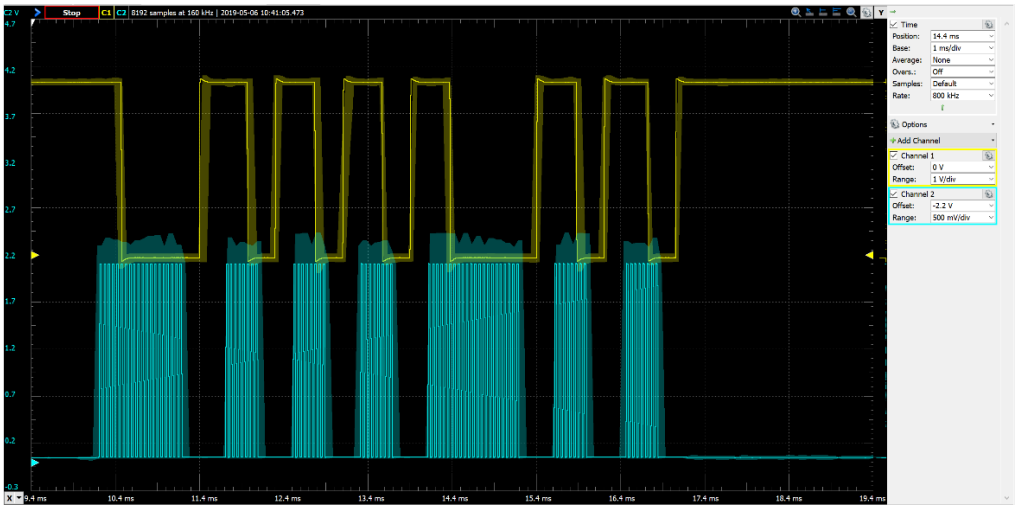
Address 0 Command 2



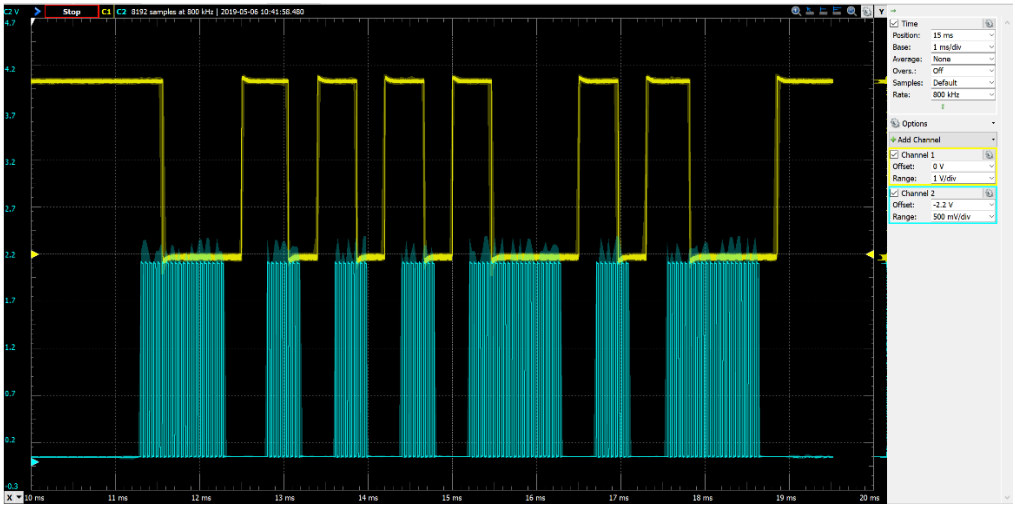
Address 0 Command 3



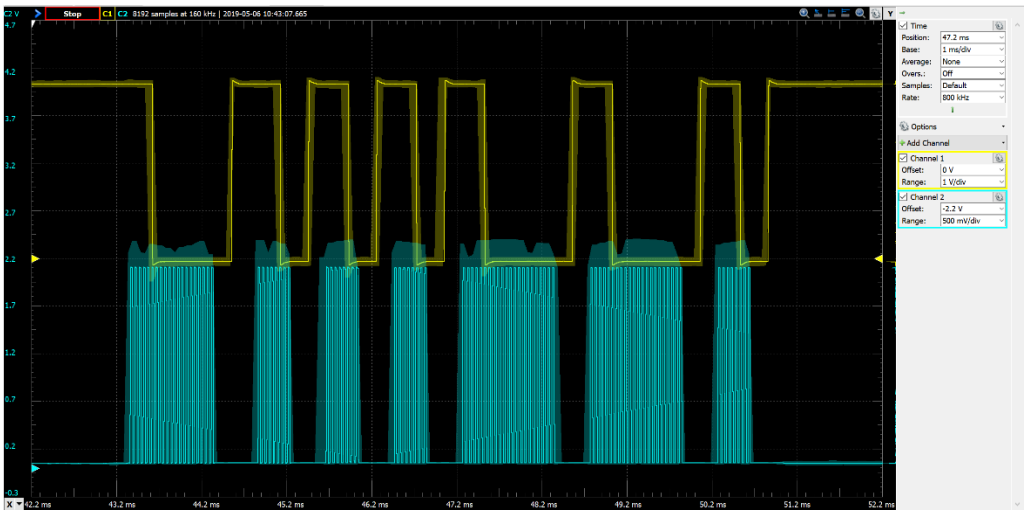
Address 0 Command 4



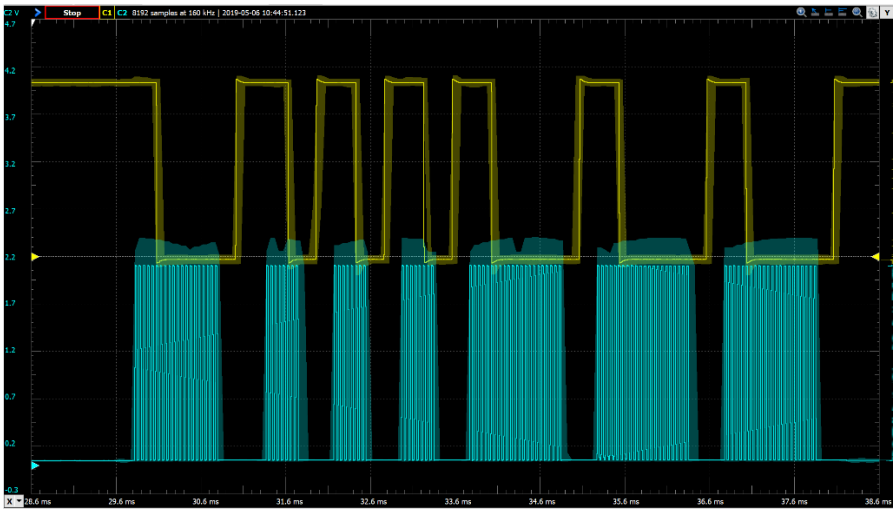
Address 0 Command 5



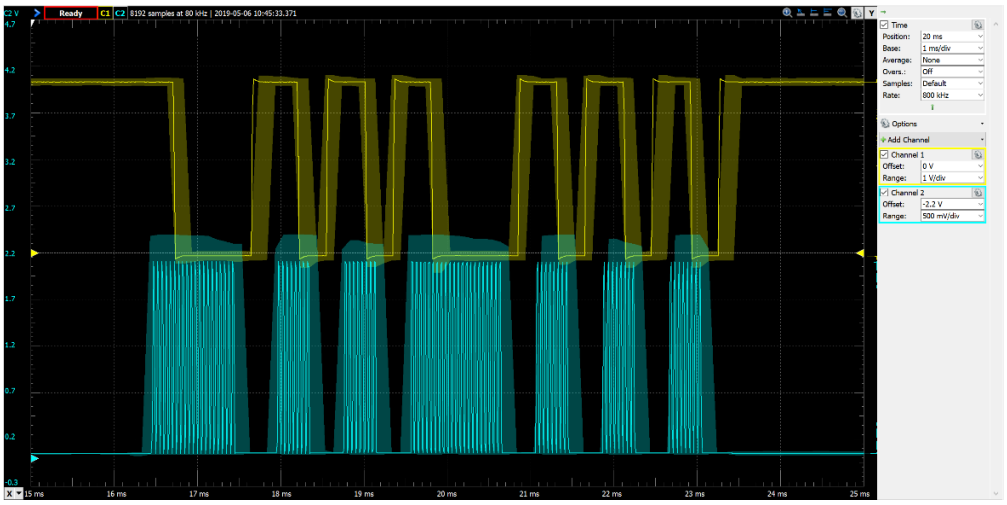
Address 0 Command 6



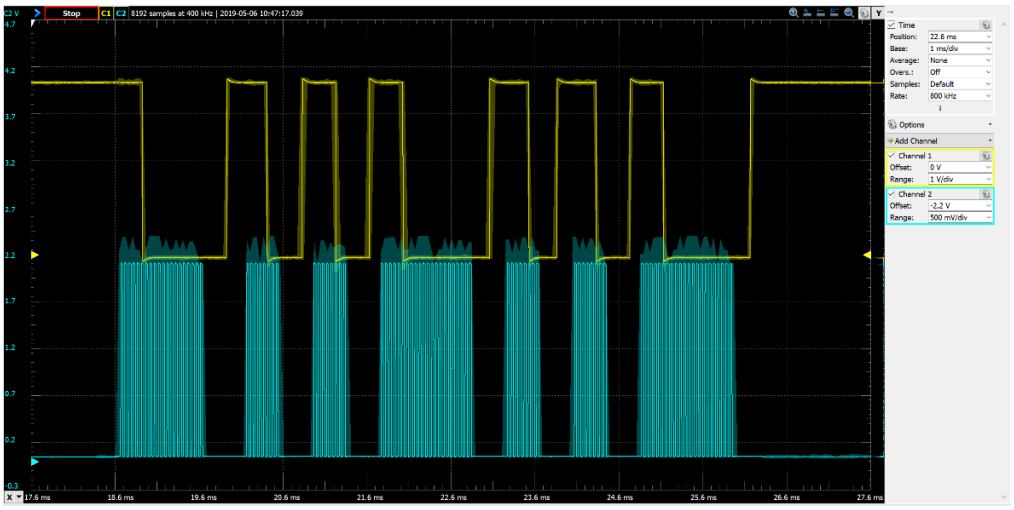
Address 0 Command 7



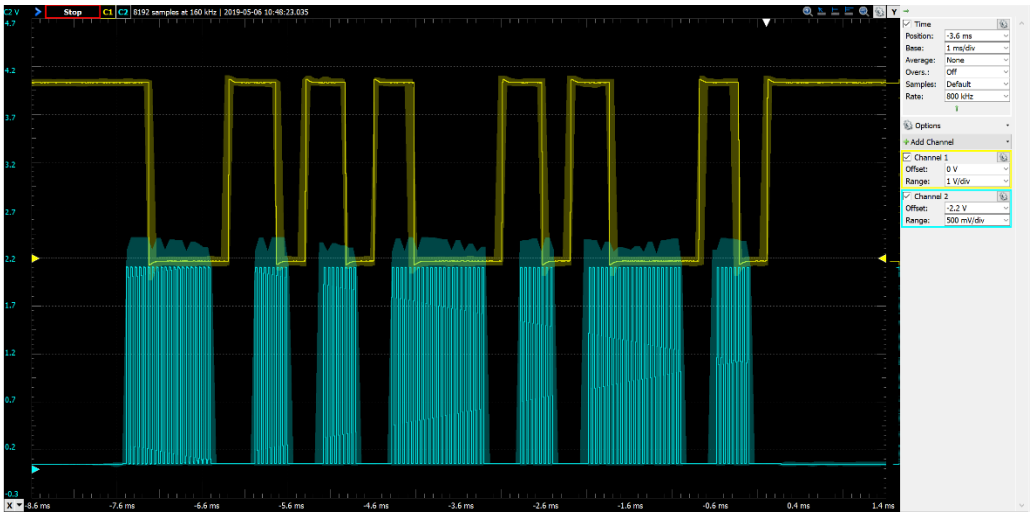
Address 0 Command 8



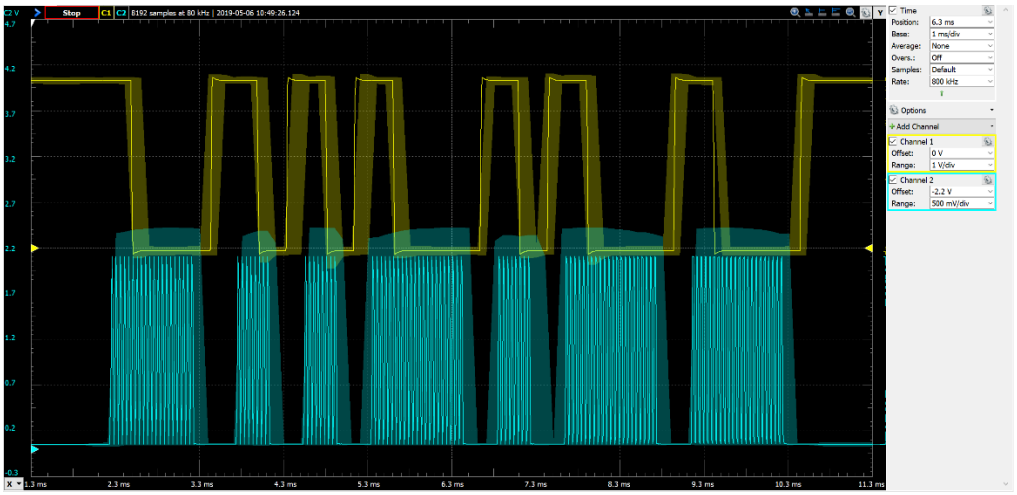
Address 0 Command 9



Address 0 Command A

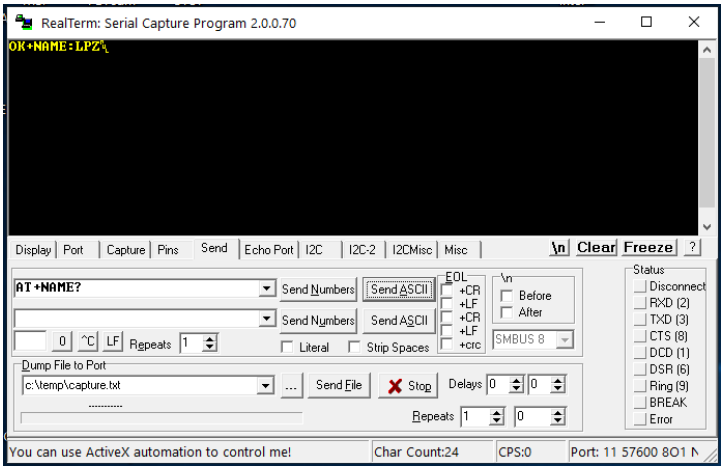


Address 0 Command B

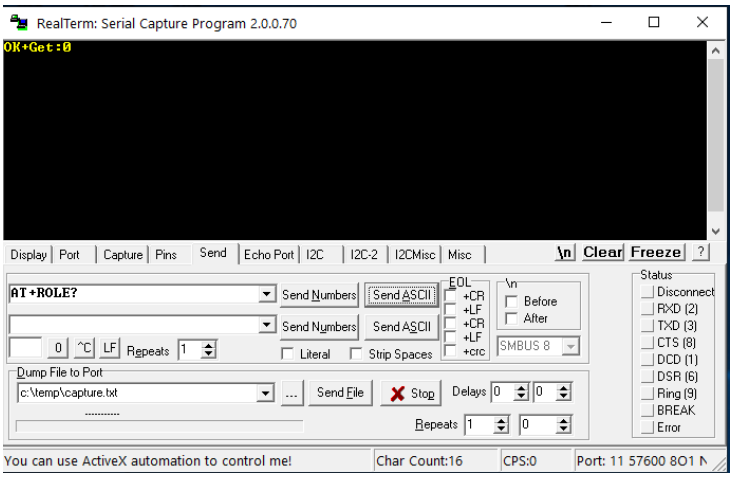


Bluetooth Configuration mode

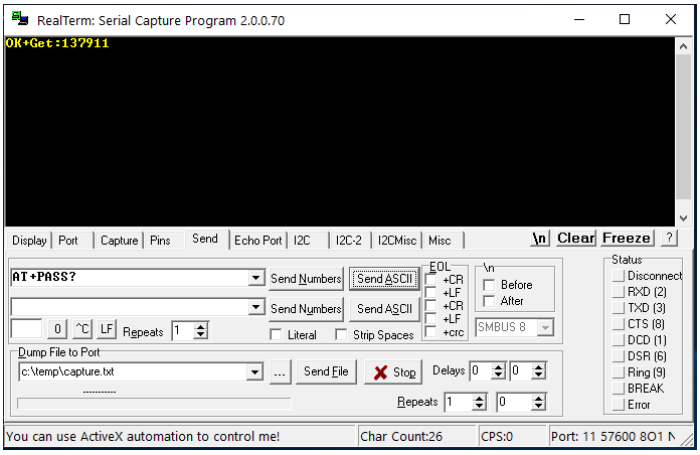
Name Set up



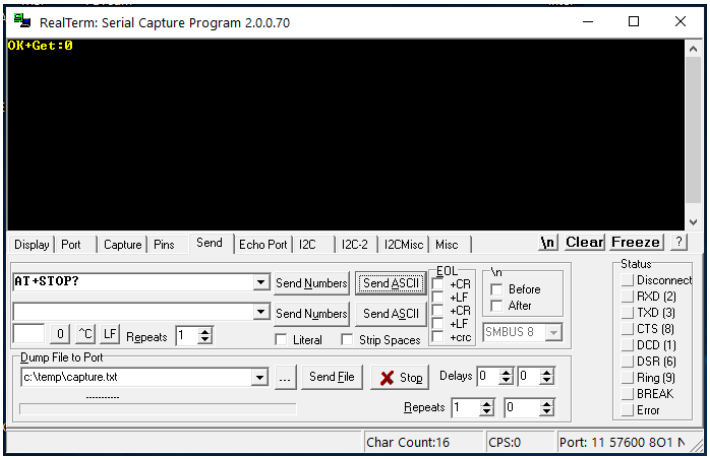
Slave mode set up



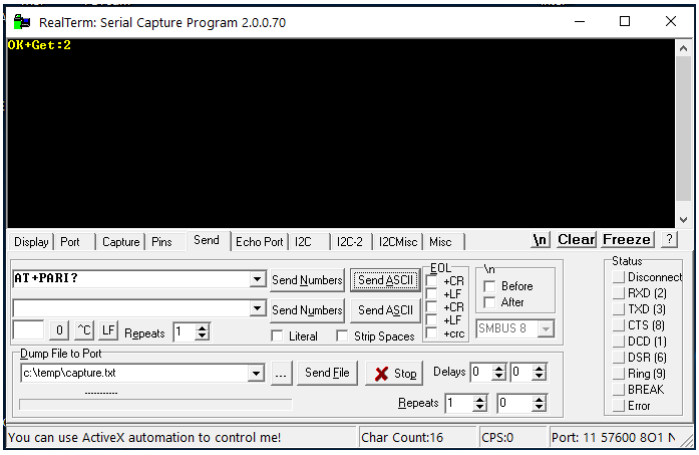
Password set up



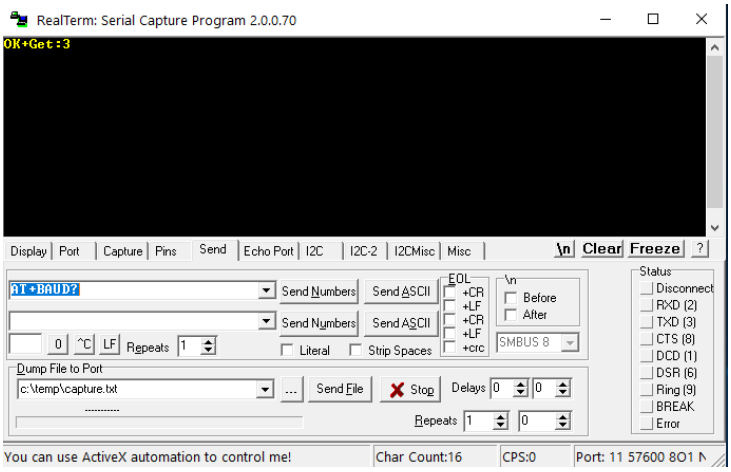
Stop-bit set up



Odd Parity set up



Baud rate set up



Conclusion

Creating a new IR Protocol is a unique and challenging task. Working on this project gave us more inside of the benefits from using Oscilloscopes and working in a team since we divided the receiving and transmitting, but both sides had to work efficiently together to achieve the end goal, which was to create a new Infrared protocol. For this project, we learned the usefulness of PWM and timers to modulate signals, as the waveforms in the top prove it. UART was also very involved to communicate with the Bluetooth module and the receiver station with the 8-bit Binary Weighted DAC. IR is an old form of communication that seems to be taken over by Bluetooth and other wireless communications, however, the reasons it still around shows the important use in cheaper designs.

Source Code

Transmitter Station

```
1
2
3   /*
4       Kevin Lopez, Cristian Lopez
5       Robot car with bluetooth module and IR
6        Project 2
7
8       UART7_Tx()         to send data
9       UART7_Rx()         receive data
10      UART7_Transmit_String() send a string
11
12    Using IR protocol to use protocol
13   */
14
15  #include <stdint.h>
16  #include <math.h>
17  #include "PLL.h"
18  #include "PWM.h"
19  #include "tm4c123gh6pm.h"
20  #include "INIT.c"  //UART PORTc PWM TIMER initialization
21
22
23
24  /*
25       USED PINS
26
27        PC 6  PC.7       wheel
28        PC 5  PC 4         wheel
29        PB.6  PB.7       wheels speed
30
31        PB4          IR signal
32        PE.0  PE.1       UART (RX, TX)
33
34
35  */
36
37
38
39  int main(void){
40      uint8_t bt_data;
41      PLL_Init();                     //  bus clock at 80 MHz
42      PWM0A_Init(40000, 20000);       //  initialize PWM0, 1000 Hz, 37% duty       RIGHT WHEEL
43      PWM0B_Init(40000, 20000);       //  initialize PWM0, 1000 Hz, 37% duty       LEFT WHEEL
44      UART7_Init();                   //   57600, 8 Data bits, 1 stop, odd parity
45      Switch_Init();
46      PWM1A_Init  ();                 // 40Khz 50% duty cycle                    PB4
47      Timer1A_Int(1000);              //MicroSeconds
48
49
50
51
52
53    uint8_t addr;   //default address is device0
54    uint8_t ZERO[] =  "Trasmited 0 \n\r";
55    uint8_t ONE[] =   "Trasmited 1 \n\r";
56    uint8_t TWO[] =   "Trasmited 2 \n\r";
57    uint8_t THREE[] = "Trasmited 3  \n\r";
58    uint8_t FOUR[] =  "Trasmited 4  \n\r";
59    uint8_t FIVE[] =  "Trasmited 5  \n\r";
60    uint8_t SIX[] =   "Trasmited 6  \n\r";
61    uint8_t SEVEN[] = "Trasmited 7  \n\r";
62    uint8_t EIGHT[] = "Trasmited 8  \n\r";
63    uint8_t NINE[] =  "Trasmited 9  \n\r";
64    uint8_t A_HEX[] = "Trasmited A  \n\r";
65    uint8_t B_HEX[] = "Trasmited B  \n\r";
66    uint16_t A_INIT_SPEED  = 17000, B_INIT_SPEED  = 20000;
67    addr = 1;
68    addr = 0;
69
70      while(1){
71
```

```
72          bt_data = UART7_Rx();
73
74          switch(bt_data){
75            case 'w'  :  GPIO_PORTC_DATA_R |= 0xC0; GPIO_PORTC_DATA_R &= ~0x30;//FOWARDS
76                         PWM0_0_CMPA_R = A_INIT_SPEED; PWM0_0_CMPB_R = B_INIT_SPEED;
77                         UART7_Transmit_String (FOWARD);  break;
78
79            case 'a'  : GPIO_PORTC_DATA_R |= 0xC0;  GPIO_PORTC_DATA_R &= ~0x30; // LEFT TURN
80                         PWM0_0_CMPA_R =  14000; PWM0_0_CMPB_R = 8000;
81                         UART7_Transmit_String (LEFT);  break;
82
83            case 'd'  : GPIO_PORTC_DATA_R |= 0xC0;  GPIO_PORTC_DATA_R &= ~0x30; //RIGHT TURN
84                         PWM0_0_CMPA_R = 7000; PWM0_0_CMPB_R = 15000;
85                         UART7_Transmit_String (RIGHT); break;
86
87            case 's'  :   GPIO_PORTC_DATA_R &= ~0xC0;  GPIO_PORTC_DATA_R |= 0x30; // BACKWARDS
88                         PWM0_0_CMPA_R =  A_INIT_SPEED;  PWM0_0_CMPB_R = B_INIT_SPEED;
89                         UART7_Transmit_String (BACKWARDS); break;
90
91            case 'q'  : PWM0_0_CMPA_R =  0; PWM0_0_CMPB_R = 0; // stop the car from movement
92                         GPIO_PORTC_DATA_R |= 0xC0;  GPIO_PORTC_DATA_R &= ~0x30;//FOWARDS
93                         UART7_Transmit_String (STOP);  break;
94
95            case  'k' : A_INIT_SPEED -= 4000; B_INIT_SPEED -= 4000; break; //DECREASE SPEED
96
97            case  'i' : A_INIT_SPEED += 4000; B_INIT_SPEED += 4000; break; //INCREASE SPEED
98
99            case 'c'  : UART7_Transmit_String (ADD); addr = UART7_Rx(); break;   // change address
100
101           case '0'  : startbit(); address(addr); logic(0); logic(0); logic(0); logic(0);
102                        UART7_Transmit_String (ZERO); break; //command0
103
104           case '1'  : startbit(); address(addr); logic(0); logic(0); logic(0); logic(1);
105                        UART7_Transmit_String (ONE); break;  //command1
106
107           case '2'  : startbit(); address(addr); logic(0); logic(0); logic(1); logic(0);
108                        UART7_Transmit_String (TWO); break;  //command2
109
110           case '3'  : startbit(); address(addr); logic(0); logic(0); logic(1); logic(1);
111                        UART7_Transmit_String (THREE); break;  //command3
112
113           case '4'  : startbit(); address(addr); logic(0); logic(1); logic(0); logic(0);
114                        UART7_Transmit_String (FOUR); break; //command4
115
116           case '5'  : startbit(); address(addr); logic(0); logic(1); logic(0); logic(1);
117                        UART7_Transmit_String (FIVE); break; //command5
118
119           case '6'  : startbit(); address(addr); logic(0); logic(1); logic(1); logic(0);
120                        UART7_Transmit_String (SIX); break;  //command6
121
122           case '7'  : startbit(); address(addr); logic(0); logic(1); logic(1); logic(1);
123                        UART7_Transmit_String (SEVEN); break;  //command7
124
125           case '8'  : startbit(); address(addr); logic(1); logic(0); logic(0); logic(0);
126                        UART7_Transmit_String (EIGHT); break;  //command8
127
128           case '9'  : startbit(); address(addr); logic(1); logic(0); logic(0); logic(1);
129                        UART7_Transmit_String (NINE); break;   //command9
130
131           case 'A'  : startbit(); address(addr); logic(1); logic(0); logic(1); logic(0);
132                        UART7_Transmit_String (A_HEX); break;  //commandA
133
134           case 'B'  : startbit(); address(addr); logic(1); logic(0); logic(1); logic(1);
135                        UART7_Transmit_String (B_HEX); break;  //commandB
136
137           default : break;
138         }
139           Timer1A_Int( 1000 ); // SO WE DONT SEND TO MUCH DATA BACK TO BACK TO IR
143       }// while
144   }// main
145
146
147
```

```
#include "tm4c123gh6pm.h"
#include <stdint.h>

    uint8_t FOWARD []    = "MOVING FOWARD\n\r" ;
    uint8_t LEFT []        = "TURNING LEFT\n\r" ;
    uint8_t RIGHT []     = "TURNING RIGHT\n\r" ;
    uint8_t BACKWARDS []   = "MOVING BACKWARDS\n\r" ;
    uint8_t STOP []      = "STOP\n\r" ;
    uint8_t CHANGEaDDRE [] = "CHANGE DEVICE ADDRESS\n\r"  ;
    uint8_t COMMAND0 [] = "COMMAND0" ;
    uint8_t COMMAND1 [] = "COMMAND1" ;
    uint8_t COMMAND2 [] = "COMMAND2" ;
    uint8_t COMMAND3 [] = "COMMAND3" ;
    uint8_t COMMAND4 [] = "COMMAND4" ;



////////////////////////////////////////////////////////
////////////////////// SYSTICK /////////////////////////
////////////////////////////////////////////////////////
void Systick(void){ // Systick interrupt for ADC
  NVIC_ST_CTRL_R  = 0;                // Disables SysTick timer in order to change reload value
  NVIC_ST_RELOAD_R  = 80000-1;    // reload reg with about .5 seconds
  NVIC_ST_CTRL_R  = 7;                // enable SysTick timer, and interrupt, system clock source
}


////////////////////////////////////////////////////////
////////////////////// PORTF ///////////////////////////
////////////////////////////////////////////////////////
void Switch_Init(void){  unsigned long volatile delay;
  SYSCTL_RCGC2_R  |= 0x00000024 ; // (a) activate clock for port F,A
  delay = SYSCTL_RCGC2_R ;
  GPIO_PORTF_LOCK_R  = 0x4C4F434B ; // unlock GPIO Port F
  GPIO_PORTF_CR_R  = 0x11;          // allow changes to PF4,0
  GPIO_PORTF_DIR_R  |= 0x0E;     // (c) ENABLE LIGHTS
  GPIO_PORTF_DIR_R  &= ~0x11;     // (c) make PF4,0 in (built-in button)
  GPIO_PORTF_AFSEL_R  &= ~0x11;  //     disable alt funct on PF4,0
  GPIO_PORTF_DEN_R  |= 0x1F;      //     enable digital I/O on PF4,0
  GPIO_PORTF_PCTL_R  &= ~0x000F000F ; //  configure PF4,0 as GPIO
  GPIO_PORTF_AMSEL_R  &= ~0x11;  //     disable analog functionality on PF4,0
  GPIO_PORTF_PUR_R  |= 0x11;      //     enable weak pull-up on PF4,0
  GPIO_PORTF_IS_R  &= ~0x11;      // (d) PF4,PF0 is edge-sensitive
  GPIO_PORTF_IBE_R  &= ~0x11;     //     PF4,PF0 is not both edges
  GPIO_PORTF_IEV_R  &= ~0x11;     //     PF4,PF0 falling edge event
  GPIO_PORTF_ICR_R  = 0x11;       // (e) clear flags 4,0
  GPIO_PORTF_IM_R  |= 0x11;       // (f) arm interrupt on PF4,PF0
  NVIC_PRI7_R  = (NVIC_PRI7_R &0xFF00FFFF )|0x00400000 ; // (g) priority 2
  NVIC_EN0_R  = 0x40000000 ;      // (h) enable interrupt 30 in NVIC

  GPIO_PORTC_DEN_R  |= 0xC0;      //     enable digital I/O on PC4,5
  GPIO_PORTC_DIR_R  |= 0xC0;      // digital output for PA3,4
  GPIO_PORTC_DEN_R  |= 0x30;      //     enable digital I/O on PC6,7
  GPIO_PORTC_DIR_R  |= 0x30;      // digital output for PA6,7
}


////////////////////////////////////////////////////////
////////////////////// UART7 ///////////////////////////
////////////////////////////////////////////////////////
      // 57600, 8 Data bits, 1 stop, odd pari
void UART7_Init(void){ //PE0,1  rx,tx
  SYSCTL_RCGCUART_R  |=  SYSCTL_RCGCUART_R7 ;
  SYSCTL_RCGCGPIO_R  |=  0x10; // CLOCK FOR PORT E
  UART7_CTL_R  = 0;
  UART7_IBRD_R  = 86;   //17;
  UART7_FBRD_R  = 51; //731
  UART7_CC_R  = 0;
  UART7_LCRH_R  |= 0X62;  // 8 bits,  Odd Paraty enable
  UART7_CTL_R  |= 0x300;  // ENABLE RX TX
  UART7_CTL_R  |= 1;      // ENABLE UART
```

```
73    GPIO_PORTE_DEN_R   |= 3;
74    GPIO_PORTE_AFSEL_R   |= 3; // ALTERNATE FUNTION
75    GPIO_PORTE_PCTL_R   |= 0x00000011 ;
76  }
77
78  void UART7_Tx (uint8_t  ByteToSend ){
79    while ((UART7_FR_R  & UART_FR_TXFF ) != 0);
80      UART7_DR_R  = ByteToSend ;
81  }
82
83  uint8_t  UART7_Rx (void){
84    while ((UART7_FR_R  & UART_FR_RXFE ) != 0);
85    return  UART7_DR_R ;
86  }
87
88  void UART7_Transmit_String  (const uint8_t  *MessageString ){
89    while (*MessageString ){ // while there is a character
90      UART7_Tx (*MessageString ); //send character
91      MessageString ++;
92    }
93  }
94
95
96
97
98  ////////////////////////////////////////////////////////////////////////////////
99  ////////////////////////////////////////      PWM ///////////////////////////////////
100 ////////////////////////////////////////////////////////////////////////////////
101
102 void  PWM1A_Init ( ){ ////////// PB4        40Khz
103 //                                            pmw2  Gen1A
104   SYSCTL_RCGCPWM_R   |= 0x01;              // 1) activate PWM0
105   SYSCTL_RCGCGPIO_R   |= 0x02;             // 2) activate port B
106
107   while ((SYSCTL_PRGPIO_R &0x02) == 0){}; //delay
108
109   ////////////////////////port B INIT
110   GPIO_PORTB_AFSEL_R   |= 0x10;            // enable alt funct on PB4
111   GPIO_PORTB_PCTL_R   &= ~0x000F0000 ;     // configure PB4 as PWM0
112   GPIO_PORTB_PCTL_R   |= 0x00040000 ;
113   GPIO_PORTB_AMSEL_R   &= ~0x10;           // disable analog functionality on PB6
114   GPIO_PORTB_DEN_R   |= 0x10;              // enable digital I/O on PB6
115   GPIO_PORTB_DIR_R   |= 0x10;              // set  PB4 as OUTPUT
116   ////////////////////// PWM CONFIG
117
118   PWM0_1_CTL_R  = 0;                       // 4) re-loading down-counting mode
119   PWM0_1_GENA_R  = 0xC8;                   // low on LOAD, high on CMPA down
120   // PB6 goes low on LOAD
121   // PB6 goes high on CMPA down
122   PWM0_1_LOAD_R  = 2000;                   // 5) 80Mhz/40k = 2000
123   PWM0_1_CMPA_R  = 0;                   // 6) count value when output rises
124   PWM0_1_CTL_R  |= 0x00000001 ;            // 7) start PWM0
125   PWM0_ENABLE_R  |= 0x00000004 ;           // enable PB6/M0PWM0
126
127 }
128
129 ////////////////////////////////////////////////////////
130 ////////////////////// TIMER //////////////////////////
131 ////////////////////////////////////////////////////////
132
133
134 void  Timer1A_Int (uint32_t  delay){//micro seconds
135   uint32_t  i = 0;
136   // Timer1A configuration
137   SYSCTL_RCGCTIMER_R   |= 0x02; // enable clock to timer Block 1
138   TIMER1_CTL_R  = 0;            // disable Timer1 during configuration
139   TIMER1_CFG_R  = 0x04;         // 16-bit timer
140   TIMER1_TAMR_R  = 0x02;        // periodic countdown mode
141   //TIMER1_TAPR_R = 10;         // 16MHZ / 10 = 160 KHZ
142   TIMER1_TAILR_R  = 80 - 1;       //  80E6/80 = 1E6    period = 1MicroSecond
143   TIMER1_ICR_R  |= 0x1;           // clear Timer1A timeout flag
144   //TIMER1_IMR_R |= 0x01;       // enable Timer1A timeout interrupt
```

```
145     TIMER1_CTL_R  |= 0x01;        // enable Timer1A
146     NVIC_EN0_R  |= 0x00200000 ;    // enable IRQ21
147
148
149     while (i < delay){
150       while ((TIMER1_RIS_R  & 0x01) == 0); // wait for timeout
151       TIMER1_ICR_R  = 0x01;     // clear timerA timeout flag
152       i++;
153     }
154
155
156   }
157
158
159
160
161
162   ////////////////////////////////////////////////////////
163   ////////////////////////  IR FUNTIONS //////////////////////
164   ////////////////////////////////////////////////////////
165   uint8_t  ADD[] = "Enter address  bit\n\r"   ;
166   uint8_t  ran[] = "press another key\n\r"  ;
167
168
169   void  startbit (){
170   //  • Start pulse: 1ms high, 500us low
171     PWM0_1_CMPA_R  = 1000;//set bit high
172     Timer1A_Int ( 1000   );  /////////// coount for 1ms
173     PWM0_1_CMPA_R  = 0;//set bit low
174      Timer1A_Int (  500 );// count for .5ms
175   }
176
177
178
179    void  logic (uint8_t  bitToSend ){
180   /*  • Logical 1: 1.1ms high, 400us low
181       • Logical 0: 400us high, 400us low    */
182
183       if (bitToSend  == 1){
184           PWM0_1_CMPA_R  = 1000;
185           Timer1A_Int (1100); //1100 microSeconds
186           PWM0_1_CMPA_R  = 0;//set bit low
187           Timer1A_Int (  400 );// count for .4ms
188       }else if (bitToSend  == 0){
189           PWM0_1_CMPA_R  = 1000;
190           Timer1A_Int (400); //1100 microSeconds
191           PWM0_1_CMPA_R  = 0;//set bit low
192           Timer1A_Int ( 400 );// count for .4ms
193       }
194    }
195
196
197
198
199   void  address (uint8_t  add){
200
201         uint8_t  msb_bin , lsb_bin ;
202
203         switch (add){
204           case  '0' : msb_bin  = 0; lsb_bin  = 0; break ;
205           case  '1' : msb_bin  = 0; lsb_bin  = 1; break ;
206           case  '2' : msb_bin  = 1; lsb_bin  = 0; break ;
207           case  '3' : msb_bin  = 1; lsb_bin  = 1; break ;
208         }
209
210         logic ( msb_bin  );
211         logic ( lsb_bin  );
212   }
213
214
215
```

Receiver Station

```
/*
Kevin Lopez
Cristian Lopez

ASUMING INVERTED LOGIC

Incoming data from PB4

UART PB0-1
*/


#include <stdint.h>
#include "JYU_447V2.c"
#include "ST7735.h"


int main(void){
    PORTB_INIT();   //modulator comes from portB4 and interrupt is enable
    Timer1A_Init();   // 25 microSeconds to check for data
    PLL_Init();     // 80Mhz
    UART1_poll_Init();
    ST7735_InitR(INITR_REDTAB);
    ST7735_FillScreen(0xFFFF);            // set screen to white
    uint8_t command;
    while(1){

        if ( (AR_Count >= 56) && (AR_Count <= 60) ){
          ////////////////////////---------Check Start pulse
          for(int i = 10; i < 32; i ++){ //ASUMING INVERTED LOGIC
              if (START[i] != 0) { //error
                //reset count
                AR_Count = 0;
              }
          }

          for (int i = 45; i < 55; i++){
            if (START[i] == 0) {
              //reset count
              AR_Count = 0;

            }
          }
          //addrCommCount = 0;
        }// IF 60


        if    ( addrCommCount >= 360) {
          checkAdd_1st_bit(); //msb
          checkAdd_2nd_bit();
          check_1st_bit(); //msb
          check_2nd_bit();
          check_3rth_bit();
          check_4rth_bit(); //lsb

          if ( (addrBitONe ==0) && (addrBitTwo ==0) ){ // if our device is being selected.


            command =  convertBinaryToDecimal( dataBit1, dataBit2, dataBit3, dataBit4 ) ;

            switch(command){
              case 0:
                ST7735_FillScreen(0xFFFF);            // set screen to white
                ST7735_DrawString( 4, 6," Try again...lol", ST7735_WHITE, 1 );
                UART1_poll_Tx('N');
                break;
              case 1:
                ST7735_FillScreen(0xFFFF);            // set screen to white
                ST7735_DrawBitmap(20, 110, Athur, 90, 90);
                DelayWait10ms(7);
                ST7735_DrawBitmap(22, 108, Athur, 90, 90);
                DelayWait10ms(7);
```

```
73                  ST7735_DrawBitmap (24, 110, Athur, 90, 90);
74                  DelayWait10ms (7);
75                  ST7735_DrawBitmap (22, 108, Athur, 90, 90);
76                  DelayWait10ms (7);
77                  UART1_poll_Tx ('N');
78                  break;
79                case 2:
80                  ST7735_FillScreen (0xFFFF);            // set screen to white
81                  ST7735_DrawString (1, 0, "Hiii" , ST7735_RED , 5);
82                  DelayWait10ms (7);
83                  ST7735_DrawString (1, 1, "Hiii" , ST7735_RED , 5);
84                  DelayWait10ms (7);
85                  ST7735_DrawString (1, 2, "Hiii" , ST7735_RED , 5);
86                  DelayWait10ms (7);
87                  ST7735_DrawString (1, 3, "Hiii" , ST7735_RED , 5);
88                  DelayWait10ms (7);
89                  ST7735_DrawString (1, 4, "Hiii" , ST7735_RED , 5);
90                  DelayWait10ms (7);
91                  ST7735_FillScreen (0xFFFF);  // set screen to white
92                  ST7735_DrawString (1, 5, "Hiii" , ST7735_RED , 5);
93                  UART1_poll_Tx ('N');
94                  break; //display something.
95                case 3:
96                    ST7735_FillScreen (0xFFFF);             // set screen to white
97                    for(int i = 0; i < 120; i++){
98                    ST7735_FillCircle (65, 65, i, ST7735_BLUE );
99                    DelayWait10ms (5);
100                   }
101                   break;
102               case 4:
103                 ST7735_FillScreen (0xFFFF);            // set screen to white
104                 ST7735_DrawBitmap (20, 110, Athur, 90, 90);
105                 DelayWait10ms (90);
106                 ST7735_SetRotation (2);
107                 ST7735_DrawBitmap (20, 110, Athur, 90, 90);
108                 DelayWait10ms (90);
109                 ST7735_SetRotation (3);
110                 ST7735_DrawBitmap (20, 110, Athur, 90, 90);
111                 DelayWait10ms (90);
112                 ST7735_SetRotation (1);
113                 UART1_poll_Tx ('N');
114                 break;
115               case 5:
116                 UART1_poll_Tx ('C');
117                 ST7735_FillScreen (0xFFFF);            // set screen to white
118                 ST7735_DrawString ( 4, 6," Playing node C" , ST7735_WHITE , 1 );
119                 break; //send UART command to make sound
120               case 6:
121                 UART1_poll_Tx ('D');
122                 ST7735_FillScreen (0xFFFF);            // set screen to white
123                 ST7735_DrawString ( 4, 6," Playing node D" , ST7735_WHITE , 1 );
124                 break; //send UART command to make sound
125               case 7:
126                 UART1_poll_Tx ('E');
127                 ST7735_FillScreen (0xFFFF);            // set screen to white
128                 ST7735_DrawString ( 4, 6," Playing node E" , ST7735_WHITE , 1 );
129                 break; //send UART command to make sound
130               case 8:
131                 UART1_poll_Tx ('F');
132                 ST7735_FillScreen (0xFFFF);            // set screen to white
133                 ST7735_DrawString ( 4, 6," Playing node F" , ST7735_WHITE , 1 );
134                 break; //send UART command to make sound
135               case 9:
136                 UART1_poll_Tx ('G');
137                 ST7735_FillScreen (0xFFFF);            // set screen to white
138                 ST7735_DrawString ( 4, 6," Playing node G" , ST7735_WHITE , 1 );
139                 break; //send UART command to make sound
140               case 10:
141                 UART1_poll_Tx ('A');
142                 ST7735_FillScreen (0xFFFF);            // set screen to white
143                 ST7735_DrawString ( 4, 6," Playing node A" , ST7735_WHITE , 1 );
144                 break; //send UART command to make sound
```

```
145                 case 11:
146                   UART1_poll_Tx('B');
147                   ST7735_FillScreen(0xFFFF);            // set screen to white
148                   ST7735_DrawString( 4, 6," Playing node B", ST7735_WHITE, 1 );
149                   break; //send UART command to make sound
150                 default:
151                   ST7735_FillScreen(0xFFFF);            // set screen to white
152                   ST7735_DrawString( 1, 6," Undefined Input", ST7735_WHITE, 1 );
153                   UART1_poll_Tx('N');
154                   break;
155               }
156             }
157
158         }// if >360
159           //check bits and do animation related to it.
160       } //while 1
161
162   }//main
163
164   void GPIOPortB_Handler(void){ //THIS HAPPENDS EVERYTIME B IS LOW
165     volatile int readback;
166
167       GPIO_PORTB_IM_R &= ~0x10;    // DISABLE INTERUPT
168       STARTLOW =  true;
169
170       GPIO_PORTB_ICR_R &= ~0x10;      // DISABLE INTERUPT
171
172   //  readback = GPIO_PORTB_ICR_R;   // force flag to clear
173   }
174
175
176   void Timer1A_Handler(void){
177       volatile uint32_t readback;
178       if(TIMER1_MIS_R & 0x01){      // Timer1A timeout flag is set
179
180         data = GPIO_PORTB_DATA_R & 0x10; //CHECK DATA TAHT COMES FORM PORT PB4
181
182         if (AR_Count < 60 && STARTLOW == true){
183           START[AR_Count] = data;
184           AR_Count++;
185         }
186
187         if    ( (AR_Count == 60)  && (STARTLOW == true) && (addrCommCount <= 360)  ){
188           ADDRESS_COMMAND[addrCommCount] = data;
189           addrCommCount++;
190         }
191
192         TIMER1_ICR_R = 0x01;         // clear TimerA timeout flag
193         readback = TIMER1_ICR_R;     // force interrupt flag clear
194       }
195       else{
196         TIMER1_ICR_R = TIMER1_MIS_R;  // clear all flags
197         readback = TIMER1_ICR_R;      // force interrupt flags clear
198       }
199   }
200
201
```

IR Protocol

```
#include "tm4c123gh6pm.h"
#include <stdbool.h>
#include "Init.c"
#include <math.h>



#define INDEX4ONE          60  //arrasy index to trasmit a one
#define INDEX4ZERO         32  // arrasy index to trasmit a zero
#define FIRST_BIT_CHECK    23 -1
#define FIRST_BIT_CHECK2   25 -1


/*
      TIMER IS 25 MicroSeconds pulse

      START PULSE IS 15000 MicroSeconds

          • Logical 1: 1.1ms low, 400us high
          • Logical 0: 400us low, 400us high

          We check between 575Ms and 625 Ms
                  23rd   and   25th bit

*/

uint8_t START[60];// UP TO           1.5mS
uint8_t ADDRESS_COMMAND [360];//UP TO 2 BITS   9mS
uint16_t AR_Count, addrCommCount;
bool START_FLAG = false;
bool SIGNAL_DONE = false;
bool STARTLOW = false;
uint8_t addrBitONe, addrBitTwo;
uint8_t dataBit1, dataBit2, dataBit3, dataBit4;

uint8_t data; //stores data form PB4




uint16_t secondBitTime, secondBitTime2;
uint16_t thirdBitTime, thirdBitTime2;


uint16_t firstDaBitTime, firstDaBitTime2;
uint16_t secondDaBitTime, secondDaBitTime2;
uint16_t thirdDaBitTime, thirdDaBitTime2;
uint16_t fourthDaBitTime, fourthDaBitTime2;

void check_start_bit (){




}


void checkAdd_1st_bit ( ){ //ADDRESS MSB

          addrBitONe = 0; //assuming its a ZERO
          secondBitTime = FIRST_BIT_CHECK + INDEX4ZERO;
          secondBitTime2 = secondBitTime + 2;

          for(int i = FIRST_BIT_CHECK; i < FIRST_BIT_CHECK2; i++){
              if(ADDRESS_COMMAND [i] == 0){ // ITS A ONE
                  addrBitONe = 1;
                  secondBitTime = FIRST_BIT_CHECK + INDEX4ONE;
                  secondBitTime2 = secondBitTime + 2;
              }//if
          }//for
```

```
73
74
75  }
76
77  void checkAdd_2nd_bit (){ //ADDRESS LSB
78
79
80        addrBitTwo  = 0; //assuming its a high
81        firstDaBitTime  =  secondBitTime  + INDEX4ZERO;
82        firstDaBitTime2  = firstDaBitTime  + 2;
83
84        for(int i = secondBitTime ; i < secondBitTime2 ; i++){
85            if(ADDRESS_COMMAND [i] == 0){ // ITS A ONE
86                addrBitTwo  = 1;
87                firstDaBitTime  =  secondBitTime  + INDEX4ONE;
88                firstDaBitTime2  = firstDaBitTime  + 2;
89            }//if
90        }//for
91
92
93  }
94
95  void check_1st_bit (){  //DATA MSB
96
97
98        dataBit1  = 0; //assuming its a high
99        secondDaBitTime  =  firstDaBitTime  + INDEX4ZERO;
100       secondDaBitTime2  = secondDaBitTime  + 2;
101
102       for(int i = firstDaBitTime ; i < firstDaBitTime2 ; i++){
103           if(ADDRESS_COMMAND [i] == 0){ // ITS A ONE
104               dataBit1  = 1;
105               secondDaBitTime  =  firstDaBitTime  + INDEX4ONE;
106               secondDaBitTime2  = secondDaBitTime  + 2;
107           }//if
108       }//for
109 }
110
111
112
113 void check_2nd_bit (){
114
115
116       dataBit2  = 0; //assuming its a high
117       thirdDaBitTime  =  secondDaBitTime  + INDEX4ZERO;
118       thirdDaBitTime2  = thirdDaBitTime  + 2;
119
120       for(int i = secondDaBitTime ; i < secondDaBitTime2 ; i++){
121           if(ADDRESS_COMMAND [i] == 0){ // ITS A ONE
122               dataBit2  = 1;
123           thirdDaBitTime  =  secondDaBitTime  + INDEX4ONE;
124             thirdDaBitTime2  = thirdDaBitTime  + 2;
125           }//if
126       }//for
127 }
128
129
130
131
132 void check_3rth_bit (){
133
134
135       dataBit3  = 0; //assuming its a high
136       fourthDaBitTime  =  thirdDaBitTime  + INDEX4ZERO;
137       fourthDaBitTime2  = fourthDaBitTime  + 2;
138
139       for(int i = thirdDaBitTime ; i < thirdDaBitTime2 ; i++){
140           if(ADDRESS_COMMAND [i] == 0){ // ITS A ONE
141             dataBit3  = 1;
142           fourthDaBitTime  =  thirdDaBitTime  + INDEX4ONE;
143           fourthDaBitTime2  = fourthDaBitTime  + 2;
144           }//if
```

```
145         }//for
146     }
147
148
149     void check_4rth_bit (){   //DATA LSB
150      ////////////////////reset everything after done
151
152
153         dataBit4  = 0; //assuming its a high
154
155         for(int i = fourthDaBitTime ; i < fourthDaBitTime2 ; i++){
156             if(ADDRESS_COMMAND [i] == 0){ // ITS A ONE
157               dataBit4  = 1;
158               }//if
159
160           }//for
161           ////////   reset   ///////////
162       SIGNAL_DONE  = true;
163       STARTLOW  = false;
164
165        PORTB_INIT (); //enable interrupt again
166        addrCommCount  = 0;
167        AR_Count  = 0;
168
169
170
171     }
172
173
174
175     uint8_t convertBinaryToDecimal (uint8_t msb, uint8_t thirdBinaryBit , uint8_t secondBinaryBit , uint8_t LSB
         )
176     {
177
178         uint16_t decimalNumber  = 0, i = 0, remainder , binaryNum  = 0;
179
180         binaryNum =  (msb*1000)   +   (thirdBinaryBit *100)  + (secondBinaryBit *10)  +  LSB;
181         while (binaryNum !=0)
182         {
183             remainder  = binaryNum %10;
184             binaryNum  /= 10;
185             decimalNumber  += remainder *pow(2,i);
186             ++i;
187         }
188         return decimalNumber ;
189     }
190
191
192
193
```

Binary Weighted DAC Code

```
// Kevin Lopez, Cristian Lopez
// Feb, 27
// Lab2 piano

#include "Init.c"
uint8_t data;
int main(void){
  PortB_Init();
  PortF_Init();
  Timer1A_Int();
  PortCnE();
  UART2_Init(); // UART 2    8-bits, NO parity, 1 STOP


  mode = 5;
  note = 10; // OFF STATE
  while(1){


data = UART2_Rx();
switch(data){ //UART GETS THE NOTES WE WANT TO PLAY
        case 'C': note = 0; mode = 5; Tapr =  6106*1.8; break;
        case 'D': note = 1; mode = 5; Tapr =  5442*1.8; break;
        case 'E': note = 2; mode = 5; Tapr =  4848*1.8; break;
        case 'F': note = 3; mode = 5; Tapr =  4584*1.8; break;
        case 'G': note = 4; mode = 5; Tapr =  4081*1.8;  break;
        case 'A': note = 5; mode = 5; Tapr =  3636*1.8; break;
        case 'B': note = 6; mode = 5; Tapr =  3252*1.8; break;
        default : note = 10; // ANYOTHER CHARACTER TURN OFF PIANO
      }
      TIMER1_TAILR_R  = Tapr/256;


  }//while
}//main


  void GPIOPortC_Handler (void){
    volatile int readback;
    if (mode == 5){

        if (on ==1){
          on = 0;
          portCNum = GPIO_PORTC_MIS_R  & 0xF0;
          switch(portCNum ){
            case 0x10: note = 0; break;
            case 0x20: note = 1;break;
            case 0x40: note = 2;break;
            case 0x80: note = 3;break;
            default: break;
          }//switch
        }else if(on ==0){
          on = 1;
          note = 10; // this will turn off PORT C
        }//else if


           switch(note) { //it will be set by interupts in port C
             case 0: Tapr =  6106 *1.8;  break;//c
             case 1: Tapr =  5442 *1.8;  break;//D
             case 2: Tapr =  4848 *1.8;  break;//E
             case 3: Tapr =  4584 *1.8;  break;//F
             case 4: Tapr =  4081 *1.8;  break;//G
             case 5: Tapr =  3636 *1.8;  break;//A
             case 6: Tapr =  3252 *1.8;  break;//B
             default: break;

           }//note switch
```

```
73              TIMER1_TAILR_R  = Tapr/256;
74
75        }//mode = 5
76
77
78
79        GPIO_PORTC_ICR_R  |= 0xF0;        // clear the interrupt flag
80        readback = GPIO_PORTC_ICR_R ;     // read to force interrupt flag clear
81      }
82
83
84
85      void GPIOPortE_Handler (void){
86        volatile int readback;
87        if (mode == 5){
88
89          if (on ==1){
90            on = 0;
91
92               portCNum = GPIO_PORTE_MIS_R  & 0x0E;
93              switch(portCNum ){
94                case 0x02: note = 4; break;
95                case 0x04: note = 5;break;
96                case 0x08: note = 6;break;
97                default: break;
98            }//switch
99          }else if(on ==0){
100           on = 1;
101           note = 10; // this will turn off PORT C
102
103         }//else if
104
105               switch(note) { //it will be set by interupts in port C
106                 case 0: Tapr =  6106*1.8;;  break;//c
107                 case 1: Tapr =  5442*1.8;;  break;//D
108                 case 2: Tapr =  4848*1.8;;  break;//E
109                 case 3: Tapr =  4584*1.8;;  break;//F
110                 case 4: Tapr =  4081*1.8;;  break;//G
111                 case 5: Tapr =  3636*1.8;;  break;//A
112                 case 6: Tapr =  3252*1.8;;  break;//B
113                 default: break;
114
115               }//note switch
116               TIMER1_TAILR_R  = Tapr/256;
117
118       }//mode = 5
119
120
121
122       GPIO_PORTE_ICR_R  |= 0x0F;        // clear the interrupt flag
123       readback = GPIO_PORTC_ICR_R ;     // read to force interrupt flag clear
124     }
125
126
127     void GPIOPortF_Handler (void){
128     volatile int readback;
129
130     ////////////////// CHANGE MODES ///////
131     if(mode >= 5 ){
132       mode = 1;
133     }else {mode++;}
134
135     /////////// CHANGE FRECUENCY MODE /////////////////
136       switch(mode){ // frecuency
137         case 1: Tapr =  waveFrecuency/1.15;   break;//sawtooth  good
138         case 2: Tapr =  waveFrecuency/1.15;   break; //triangle good
139         case 3: Tapr =  waveFrecuency*1.8;    break;//sine
140         case 4: Tapr =  waveFrecuency/1.15;   break;// square wave
141         case 5: note = 10;  //piano mode
142
143           break;
144       default:  break;
```

```
145     }//switch
146
147     TIMER1_TAILR_R = Tapr/256;
148
149     GPIO_PORTF_ICR_R |= 0x10;       // clear the interrupt flag
150     readback = GPIO_PORTF_ICR_R;    // read to force interrupt flag clear
151   }
152
153
154
155
156   void Timer1A_Handler(void){
157     volatile uint32_t readback;
158       if(TIMER1_MIS_R & 0x01){      // Timer1A timeout flag is set
159
160
161
162
163
164
165     /////////// CHANGE INCREASING MODE /////////////////
166       switch(mode){ // frecuency
167         case 1: PortB += 0x01; if( PortB >= 0xFF){ PortB = 0x00;}   break;//sawtooth
168         case 2: if(upDown == 1){ PortB += 0x2; if(PortB >= 0xFE) {upDown = 0;}  } //up
169             else if (upDown == 0){ PortB -= 0x02; if ( PortB <= 0x01) { upDown = 1;} }    break; //triangle
170         case 3: sinC += 1; if (sinC >= 127){ sinC = 0;} PortB = sinF[sinC]; break;//sine 0-127
171         case 4: if (sqUD <= 127){PortB = 0xFF; sqUD +=1;} else {PortB = 0x00; sqUD +=1; if(sqUD >= 255){
    sqUD = 0;} }break;// square wave
172         case 5: //piano mode
173           switch(note) { //it will be set by interupts in port C & E
174             case 0:   PortB = sinF[sinC]; sinC += 1; if (sinC >= 127){ sinC = 0;} break;//c
175             case 1:   PortB = sinF[sinC]; sinC += 1; if (sinC >= 127){ sinC = 0;} break;//D
176             case 2:   PortB = sinF[sinC]; sinC += 1; if (sinC >= 127){ sinC = 0;} break;//E
177             case 3:   PortB = sinF[sinC]; sinC += 1; if (sinC >= 127){ sinC = 0;} break;//F
178             case 4:   PortB = sinF[sinC]; sinC += 1; if (sinC >= 127){ sinC = 0;} break;//G
179             case 5:   PortB = sinF[sinC]; sinC += 1; if (sinC >= 127){ sinC = 0;} break;//A
180             case 6:   PortB = sinF[sinC]; sinC += 1; if (sinC >= 127){ sinC = 0;} break;//B
181             default:    PortB = 0; break;
182           }//note switch
183           break;
184       default:  PortB = 0; break;
185     }//switch
186
187     GPIO_PORTB_DATA_R =  PortB;
188
189
190       TIMER1_ICR_R = 0x01;        // clear TimerA timeout flag
191       readback = TIMER1_ICR_R;    // force interrupt flag clear
192     }
193     else{
194       TIMER1_ICR_R = TIMER1_MIS_R;  // clear all flags
195       readback = TIMER1_ICR_R;      // force interrupt flags clear
196     }
197   }
198
199
200
201
202
203
204
```

Initialization File

```
// Kevin Lopez, Cristian Lopez
// Feb, 27
// Lab2 piano

#include <stdint.h>
#include "inc/tm4c123gh6pm.h"

#define SW1       0x10                      // on the left side of the Launchpad board
#define SW2       0x01                      // on the right side of the Launchpad board
#define RED    0x02
#define BLUE   0X04
#define GREEN    0x08
#define OFF    0x00
uint32_t waveFrecuency   = 3636;
uint32_t PortB;

////// ARRAY FOR FRECUENCY  PIANO  //////////////
///////////          262   294     330  349     392   440   492
uint32_t tapr[] = { 6106, 5442,  4848, 4584,   4081, 3636,  3252 };
uint32_t Tapr = 3636;
uint8_t mode = 5; // sawtooth, triangle, sine, square and piano
uint8_t note = 0; //
uint8_t upDown = 1; //1 = up      0 = down
uint8_t sinC = 0;
uint8_t sqUD = 0, on = 1;
int portCNum;


uint32_t sinF[] = {0x74,0x7B,0x83,0x8B,0x93,0x9B,0x9E,0xA5,0xAB,0xB3,0xBB,0xBE,0xC4,0xCB,0xD3,0xD4,0xDB,
0xDD,0xE3,0xE6,0xEB,0xED,0xF3,0xF3,0xF6,0xFB,0xFB,0xFC,0xFD,0xFE,0xFE,0xFF,0xFE,0xFE,0xFD,0xFC,0xFB,0xFB,
0xF6,0xF3,0xF3,0xED,0xEB,0xE6,0xE3,0xDD,0xDB,0xD4,0xD3,0xCB,0xC4,0xBE,0xBB,0xB3,0xAB,0xA5,0x9E,0x9B,0x93,
0x8B,0x83,0x7B,0x74,0x6D,0x66,0x63,0x5B,0x53,0x4B,0x43,0x3C,0x35,0x33,0x2B,0x23,0x1C,0x16,0x13,0xB,0x6,
0x3,0x3,0x3,0x3,0x3,0x3,0x3,0x3,0x3,0x3,0x3,0x2,0x1,0x0,0x0,0x0,0x0,0x0,0x1,0x2,0x3,0x3,0x3,0x3,0x3,0x3,
0x3,0x3,0x3,0x3,0x3,0xB,0x13,0x16,0x1C,0x23,0x2B,0x33,0x35,0x3C,0x43,0x53,0x5B,0x63,0x66,0x6D
};



////////////////////////////////////////////////////////
////////////////////// UART2 ////////////////////////////    PED6,7  rx,tx
////////////////////////////////////////////////////////
      // 9600, 8 Data bits, 1 stop, NO pari
void UART2_Init(void){ //PD6 7  rx,tx
  SYSCTL_RCGCUART_R  |=  0x04;
  SYSCTL_RCGCGPIO_R  |=  0x08; // CLOCK FOR PORT D
  UART2_CTL_R  = 0;
  UART2_IBRD_R  = 104; // 16E6 / 16*9600
  UART2_FBRD_R  = 10; //0.1666*0.64 + 0.05
  UART2_CC_R  = 0;
  UART2_LCRH_R  |= 0X60;  // 8 bits,  NO Paraty enable
  UART2_CTL_R  |= 0x300;  // ENABLE RX TX
  UART2_CTL_R  |= 1;      // ENABLE UART

  GPIO_PORTD_DEN_R  |= 0xC0;
  GPIO_PORTD_AFSEL_R  |= 0xC0; // ALTERNATE FUNTION
  GPIO_PORTD_PCTL_R  |= 0x11000000;

}

void UART2_Tx(uint8_t ByteToSend){
  while((UART2_FR_R & UART_FR_TXFF) != 0);
    UART2_DR_R = ByteToSend;
}

uint8_t UART2_Rx(void){
  while((UART2_FR_R & UART_FR_RXFE) != 0);
  return UART2_DR_R;
}

////////////////////////////////////////////////////////
////////////////////// PORTCnE  ////////////////////////////
```

```
73  ////////////////////////////////////////////////////////
74  ////////////// BOTH edge trigger interrupt 2 PORTC.4-7  n PORTE.0-3
75  void PortCnE(){
76    volatile uint32_t delay;
77    SYSCTL_RCGCGPIO_R  |= 0x00000014;  // 1) activate clock for Port E and C
78    delay = SYSCTL_RCGCGPIO_R ;        // allow time for clock to start
79    GPIO_PORTC_DEN_R  |= 0xF0;          //Enable PC.4-7  input
80    GPIO_PORTC_DIR_R  |= 0x00;          // PC.4-7  input
81
82    GPIO_PORTE_DEN_R  |= 0x0E;          //Enable PE0-3 input
83    GPIO_PORTE_DIR_R  |= 0x00;          // PC.4-7  input
84
85
86      // configure PORTC4-7 for BOTH edge trigger interrupt
87    GPIO_PORTC_IS_R  &= ~0xF0;        // make PC.4-7 edge sensitive
88    GPIO_PORTC_IBE_R  |= 0xF0;    // trigger on both edges
89    GPIO_PORTC_ICR_R  |= 0xF0;        // clear any prior interrupt
90    GPIO_PORTC_IM_R  |= 0xF0;         // unmask interrupt
91    NVIC_PRI2_R  = 3 << 5;          // set interrupt priority to 3
92    NVIC_EN0_R  |= 0x00000004;          // enable IRQ30 bit 30 of EN0
93
94        // configure PE1-3 for BOTH edge trigger interrupt
95    GPIO_PORTE_IS_R  &= ~0x0E;        // make PE0-3 edge sensitive
96    GPIO_PORTE_IBE_R  |= 0x0E;      // trigger on both edges
97    GPIO_PORTE_ICR_R  |= 0x0E;        // clear any prior interrupt
98    GPIO_PORTE_IM_R  |= 0x0E;         // unmask interrupt
99
100   NVIC_EN0_R  |= 0x00000010;          // enable IRQ4 bit 4 of EN0----- 4rth bit
101
102 }
103
104
105 ////////////////////////////////////////////////////////
106 ////////////////////// PORTB  /////////////////////////
107 ////////////////////////////////////////////////////////
108 /////// PORTB.0 - 7 OUTPUT
109 void PortB_Init(){ //PORT B INITIALIZION ( INPUT FOR DAC )
110   volatile uint32_t delay;
111   SYSCTL_RCGCGPIO_R  |= 0x02;       //Enalbe port B clock
112   delay = SYSCTL_RCGCGPIO_R ;         // allow time for clock to start
113   GPIO_PORTB_DEN_R  = 0xFF;           // enable digital I/O on PB0-7
114   GPIO_PORTB_DIR_R  = 0xFF;           // PB0-7 out
115
116 }
117
118
119
120 ////////////////////////////////////////////////////////
121 ////////////////////// PORTF  /////////////////////////
122 ////////////////////////////////////////////////////////
123 /////// INTERUPT FOR 1ST SWITCH
124 void PortF_Init(void){
125  volatile uint32_t delay;
126   SYSCTL_RCGCGPIO_R  |= 0x00000020;  // 1) activate clock for Port F
127   delay = SYSCTL_RCGCGPIO_R ;         // allow time for clock to start
128   GPIO_PORTF_LOCK_R  = 0x4C4F434B;    // 2) unlock GPIO Port F
129   GPIO_PORTF_CR_R  = 0x1F;            // allow changes to PF4-0
130   // only PF0 needs to be unlocked, other bits can't be locked
131   GPIO_PORTF_AMSEL_R  = 0x00;         // 3) disable analog on PF
132   GPIO_PORTF_PCTL_R  = 0x00000000;    // 4) PCTL GPIO on PF4-0
133   GPIO_PORTF_DIR_R  = 0x0E;           //    1, 2) PF4 in, PF3-1 out
134   GPIO_PORTF_AFSEL_R  = 0x00;         // 6) disable alt funct on PF7-0
135   GPIO_PORTF_PUR_R  = 0x11;           // enable pull-up on PF0 and PF4
136   GPIO_PORTF_DEN_R  = 0x1F;           // 7) enable digital I/O on PF4-0
137
138
139     // configure PORTF4 for falling edge trigger interrupt
140   GPIO_PORTF_IS_R  &= ~0x10;        // make PF4 edge sensitive
141   GPIO_PORTF_IBE_R  &= ~0x10;       // trigger is controlled by IEV
142   GPIO_PORTF_IEV_R  &= ~0x10;       // falling edge trigger
143   GPIO_PORTF_ICR_R  |= 0x10;        // clear any prior interrupt
144   GPIO_PORTF_IM_R  |= 0x10;         // unmask interrupt
```

```
145
146     // enable interrupt in NVIC and set priority to 3
147     NVIC_PRI30_R  = 3 << 5;           // set interrupt priority to 3
148     NVIC_EN0_R  |= 0x40000000 ;       // enable IRQ30 bit 30 of EN0
149   }
150
151   ////////////////////////////////////////////////////////
152   ////////////////////// TIMER1A  ////////////////////////
153   ////////////////////////////////////////////////////////
154   // INTERUPT ENABLE
155   void Timer1A_Int (){
156
157       // Timer1A configuration
158     SYSCTL_RCGCTIMER_R  |= 0x02; // enable clock to timer Block 1
159     TIMER1_CTL_R  = 0;           // disable Timer1 during configuration
160     TIMER1_CFG_R  = 0x04;        // 16-bit timer
161     TIMER1_TAMR_R  = 0x02;       // periodic countdown mode
162
163     TIMER1_TAPR_R  = 10;         // 16MHZ / 10 = 160 KHZ
164     TIMER1_TAILR_R  = Tapr/256;       // 160 KHZ / Tapr
165
166     TIMER1_ICR_R  |= 0x1;          // clear Timer1A timeout flag
167     TIMER1_IMR_R  |= 0x01;       // enable Timer1A timeout interrupt
168     TIMER1_CTL_R  |= 0x01;       // enable Timer1A
169     NVIC_EN0_R  |= 0x00200000 ;  // enable IRQ21
170
171   }
172
173
174
```