EE447 TERM PROJECT REPORT

1. General Information

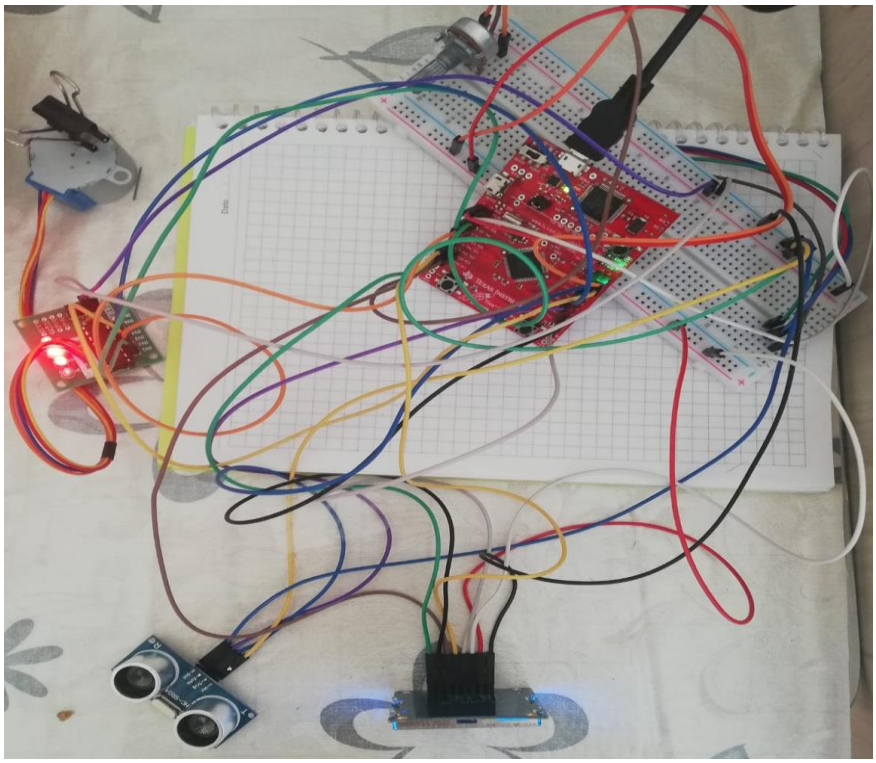

Figure 1: Shows my setup for the project.

I designed the main function as suggested in the project manual. That is:

1. State transition,
2. Print information to Nokia5110,
3. State check and state handling (ADC sequence start, HC-SR04 trigger,…).

Demo video: https://youtu.be/ccnS-USlvdM

2. Flow Chart

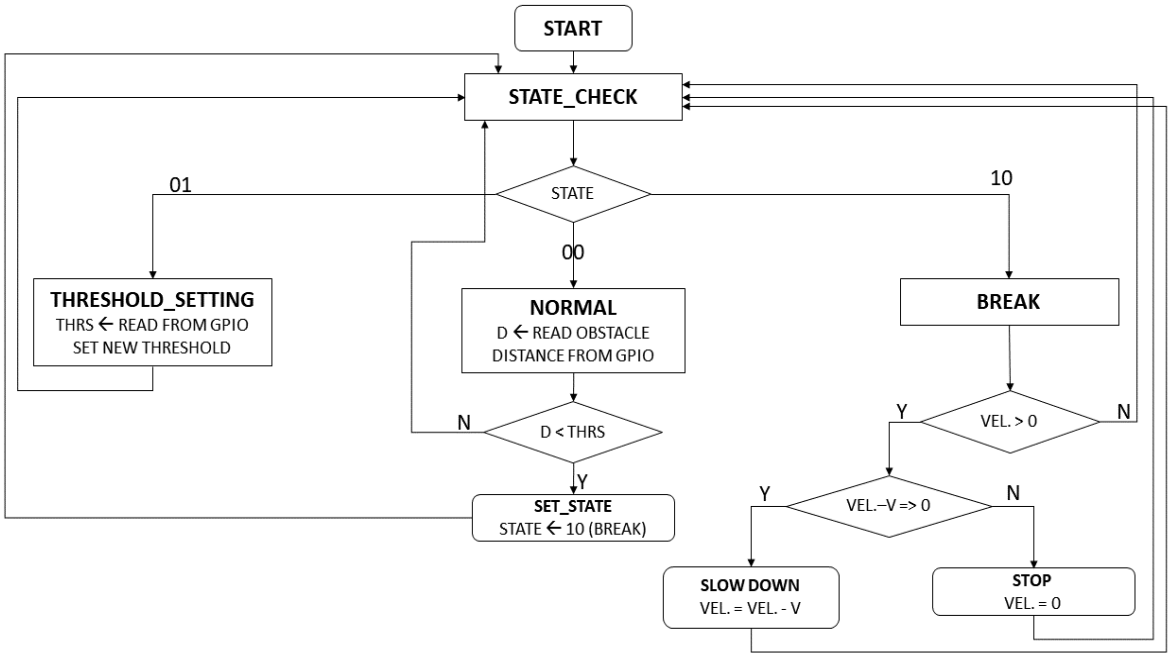

Figure 2: Shows the flow chart of overall design.

3. Routines

3.1. ST_CTRL.s

State transition subroutines are in the ***ST_CTRL.s*** file, which are SWITCH_INIT and ST_CTRL:

3.1.1. SWITCH_INIT Routine

As the name indicates, SWITCH_INIT subroutine initializes the GPIO pins for SW1 & SW2, GPIO interrupts for relevant pins (PF0 & PF4). I set the registers for both edge interrupts. I did not unmask the interrupts, instead, my program pools RIS register to check interrupts.

GPIO_PORTF_IS &= ~0x11 ; edge sensitive

GPIO_PORTF_IBE |= 0x11 ; both edge interrupt

GPIO_PORTF_ICR |= 0x11 ; clear interrupts

3.1.2. ST_CTRL Routine

State control routine. Handles sensing which button is pressed and related state transition. The state is coded in R6 register. The routine does the following:

- Pool RIS
  - If there is an interrupt, take the data from GPIODATA register with pooling and release control, as we did in the labs.
  - If not, exit the routine (no change in the state)
- State transition
  - Check the current state and button input
  - Change the state (R6 register) accordingly.

By using pooling, we do not spend 200ms for debouncing controls if there is no button input.

```
LDR         R0, =GPIO_PORTF_RIS
LDR         R1, [R0]
ANDS        R1, #0x11
BEQ         exit ; no interrupt, no state change
LDR         R0, =GPIO_PORTF_DATA
; debouncing and release controls as we did in the labs
```

3.2. HCSR04_ROUTINES.s

Trigger and detect subroutines for HC-SR04 sensor are in the ***Pulse.s*** file, which are TRIG_GPIO_INIT, DETECT_INIT, TRIG, and My_Timer0A_Handler:

3.2.1. TRIG_GPIO_INIT

Just makes the GPIO initialization as PF2 is output pin for trigger pulse.

3.2.2. DETECT_INIT

Initializes the PC7 pin and corresponding timer (WTIMER1A) for counting the high pulse time of the ECHO pin. The timed is in capture, edge time mode. Timer configuration:

```
                    ......
LDR R1, =WTIMER1_CFG
MOV R2, #0x04 ; set bits 2:0 to 0x04 for 32bit timer
STR R2, [R1]
; set for edge time and capture mode
LDR R1, =WTIMER1_TAMR
MOV R2, #0x07 ; set bit2 to 0x01 for Edge Time Mode,
STR R2, [R1] ; set bits 1:0 to 0x03 for Capture Mode
; set edge detection to both
LDR R1, =WTIMER1_CTL
LDR R2, [R1]
ORR R2, R2, #0x0C ; set bits 3:2 to 0x03
STR R2, [R1]
; set start value
LDR R1, =WTIMER1_TAILR ; counter counts down,
MOV R0, #0xFFFFFFFF ; so start counter at max value
STR R0, [R1]
                    ......
```

3.2.3. TRIG

Trigger the sensor. PF2 is connected to TRIG pin.

```
            ...
            LDR R1, =GPIO_PORTF_DATA ; set direction of PF2
            LDR R0, [R1]
            ORR R0, #0x04
            STR R0, [R1]

            MOV R0, #160 ; 16MHz general clock -> 160: 10us
            MOV R2, #0
wait        CMP R2, R0
            ADDNE R2, #1
            BNE wait

            LDR R0, [R1]
            BIC R0, #0x04
            STR R0, [R1]
            ...
```

3.3. ADC_ROUTINES.s

ADC routines are in the ***ADC_ROUTINES.s*** file, which are PWM_INIT, AIN0_INIT, and DTOA.

3.3.1. PWM_INIT Routine

Initializes the GPIO for PF3 and corresponding timer, TIMER1B, to set the brightness of the green LED depending on the read ADC level.

3.3.2. AIN0_INIT

Initializes GPIO PE2 and AIN0 to sample the threshold value from the potentiometer. SS3 and 125kps is selected.

```
        ...
; Disable sequencer 3
LDR R1, =ADC_ACTSS
LDR R0, [R1]
BIC R0, #0x08 ; clear 3th bit
STR R0, [R1]
; Select sampling
LDR R1, =ADC_EMUX
LDR R0, [R1]
BIC R0, #0xF000 ; clear bits 15:12 to select for ss3
STR R0, [R1]
; Select ATD channel 0 for sample sequencer 3
LDR R1, =ADC_SSMUX3
LDR R0, [R1]
BIC R0, #0x0F ; clear bits 3:0 to select AIN0
STR R0, [R1]
; Interrupt configuration
LDR R1, =ADC_SSCTL3
LDR R0, [R1]
BIC R0, #0x0F
ORR R0, #0x06 ; set IE0 and END0 bits
STR R0, [R1]
; Select sampling rate
LDR R1, =ADC_PC
LDR R0, [R1]
BIC R0, #0x0F ; clear 3:0 bits
ORR R0, #0x01 ; 125 ksps
STR R0, [R1]
        ....
```

3.3.3. DTOA Routines

Maps the read ADC value form the range [0,4095] to [0, 329]. The threshold range a chose is [21, 350] which mean an offset +21 is applied to the converted reading of ADC.

```
            ....
; first, multiply the number by 330
MOV R3, R4, LSL #8 ; R5 = R4*256
ADD R3, R4, LSL #6 ; R5 += R4*64
ADD R3, R4, LSL #3 ; R5 += R4*8
ADD R3, R4, LSL #1 ; R5 += R4*2
; R3 = R4*(256+64+8+2) = R4*330
MOV R4, R3, LSR #12        ; R4 = R3/4096
            ....
```

3.4. MOTOR_ROUTINES.s

Motor routines are in the ***MOTOR_ROUTINES.s*** file, which are INIT_MOTOR_GPIO,

3.4.1. INIT_TIMER0A

Initializes the WideTimer0A to create periodic interrupts for updating the motor speed and exciting transistor in full step mode.

3.4.2. INIT_MOTOR_GPIO Routine

Just initializes PB[0:3] for output to control motor transistors.

3.4.3. My_WTimer0A_Handler

ISR to update motor speed and excite the transistors.

3.5. SPI_ROUTINES.s

SPI routines to control Noki5110 are in the ***SPI_ROUTINES.s*** file, which are SPI_INIT, CMD5110, TX, INIT_SCREEN, CLR_SCR, PRINT_INFO, Str2Disp

3.5.1. SPI_INIT Routine

Initializes PA ports and SSI0 module.

I used 965. Page of the manual and the C library for configuring and controlling Nokia5110 I found on the internet [1]. The library is written in C. I try to understand what it does and convert it to the Assembly language.

3.6. InitSysTick.s

Creates interrupts to update the screen, with InitSysTick routine and My_ST_ISR interrupt service routine.

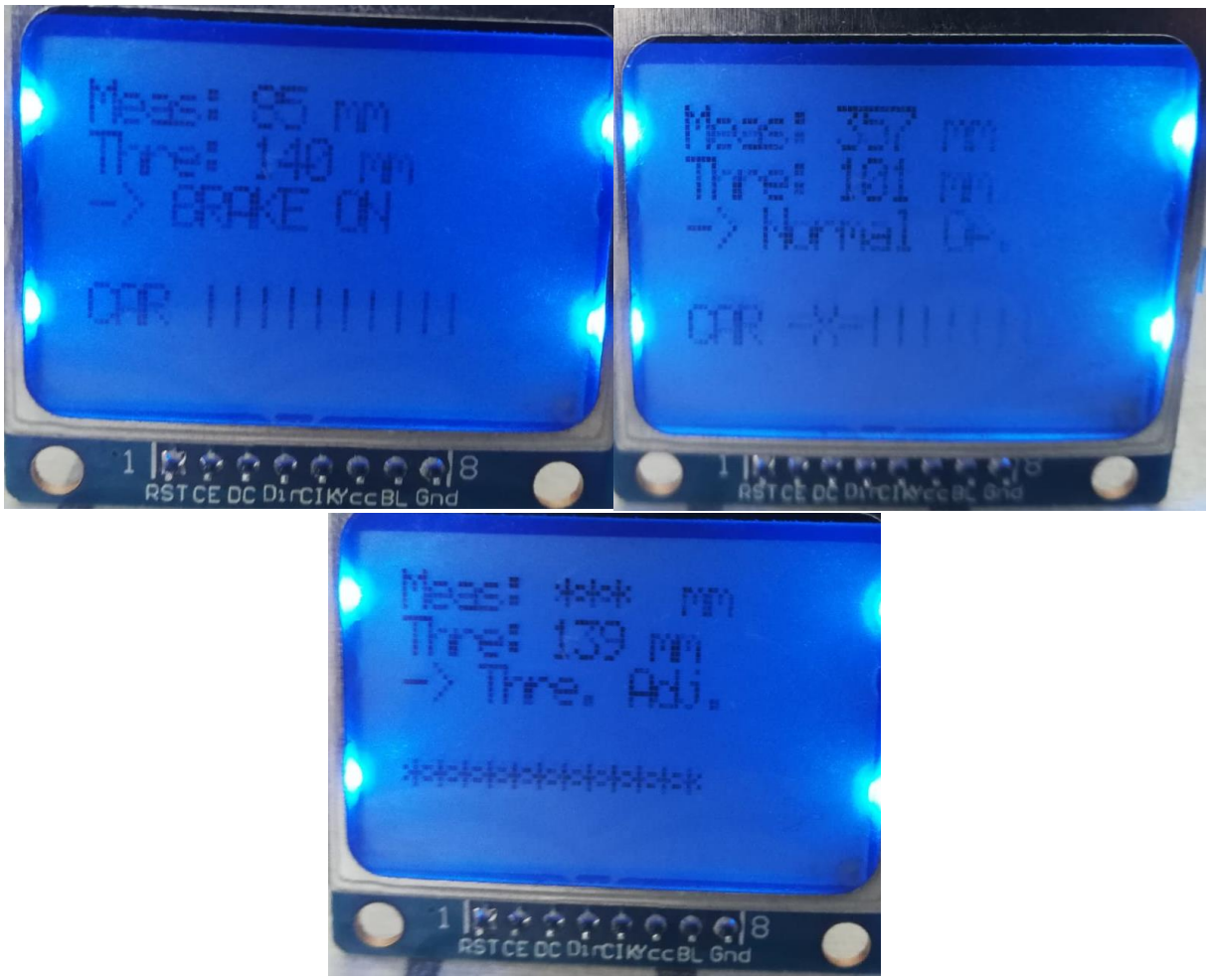

Figure 3: Shows some examples on the LCD screen.

3.6.1. CMD5110 Routine

Again, I take the [1] code as a helper to me and converted it to Assembly language. Command to 5110 routine. R4 = 0: R5 is a command, R4 != 0: R5 is a data.

```
; R5: ASCII char R4=0: command, R4!=0: data

;void N5110_Cmnd(char DATA) /* Function for sending command to Nokia5110
display */
;{
;  digitalWrite(DC, LOW); /* DC = 0, display in command mode */
;  digitalWrite(chipSelectPin, LOW); /* Make chip select pin low to enable
SPI commuincation */
;  SPI.transfer(DATA); /* Send data(command) */
;  digitalWrite(chipSelectPin, HIGH); /* Make chip select pin high to
disable SPI commuincation */
;  digitalWrite(DC, HIGH); /* DC = 1, display in data mode */
;}
            PUSH{R0-R5, LR}

            LDR R1, =GPIO_PORTA_DATA
            LDR R0, [R1]

            ;  digitalWrite(DC, LOW),   digitalWrite(chipSelectPin, LOW)

            BIC R3, R0, #0x04 ; pa3 (Fss) low. store original value at R0
            CMP R4, #0
```

```
            BICEQ R3, #0x40 ; command. pa4 (DC) low,
            ORRNE R3, #0x40 ; data. pa4 (DC) high,

            STR R3, [R1]

            ;  SPI.transfer(DATA)
            BL TX
            ;  digitalWrite(DC, HIGH),  digitalWrite(chipSelectPin, HIGH)
            STR R0, [R1]

            POP{R0-R5, LR}
            BX LR
            ENDP
```

3.6.2. TX Routine

Transmits the data in R5 to the 5110. I took the code from lecture notes and modified a bit.

```
            ; Preload R4 with UART data address
            LDR R4, =SSI0_DR
            ; check for incoming character
check       LDR R1, =SSI0_SR ; load status register address
            LDR R0, [R1] ;
            ANDS R0,R0,#0x10 ; check if previou tx is complete(BSY is 0)
            BNE check ; if not, check again, else
            STR R5, [R4] ; store char
```

3.6.3. INIT_SCREEN Routine

Sends the initialization commands as stated in the lab manual and Nokia5110 datasheet provided in OdtuClass.

```
INIT_SCREEN PROC
        ; Set H=1 for Extended Command Mode, V=0 for Horizontal Addressing
        ; Set VOP . You may need to sweep values between 0x[B0-C0] for
correct operation.
        ; Set temperature control value. You may need to sweep values
between 0x[04-07] for correct operation.
        ; Set voltage bias value as 0x13.
        ; Set H=0 for Basic Command Mode
        ; Configure for Normal Display Mode
        ; Set Cursor to detemine the start address

        ; reset the screen. Rst pin: PA7
        PUSH{R0-R5, LR}
            BL INIT_MEM
            LDR R1, =GPIO_PORTA_DATA
            LDR R0, [R1]
            BIC R0, #0x80 ; pa7 low->reset
            STR R0, [R1]

            PUSH{LR}
            BL DELAY100
            POP{LR}

            ORR R0,#0x80
            STR R0, [R1]

            MOV R4, #0
            MOV R5, #0x21
            BL CMD5110
```

```
            MOV R5, #0xBA
            BL CMD5110

            MOV R5, #0x07
            BL CMD5110


            MOV R5, #0x13
            BL CMD5110

            MOV R5, #0x20
            BL CMD5110

            MOV R5, #0x0C
            BL CMD5110

            BL DELAY100

            POP{R0-R5, LR}
            BX LR
            ENDP
```

3.6.4. CLR_SCR Routine

Iterates over all pixels and clears the screen.

```
            MOV R0, #503
            MOV R1, #0
            MOV R5, #0
            MOV R4, #0x01
loop1       MOV R5, #0
            BL CMD5110
            CMP R1, R0
            ADDNE R1, #1
            BNE loop1
```

3.6.5. PRINT_INFO Routine

Prints the information in a suitable format to the screen, depending on the state. I try to copy the interface given in the lab manual (see: Figure 3).

3.6.6. Str2Disp Routine

I took the idea from OutStr.s file given in the OdtuClass, which we used in almost all labs. The routine takes a pointer to a string of ASCII characters to be printed to the screen, sends necessary commands until 0x04 char is read.

```
Str2Disp    PROC
;*********************
; R3: Pointer to the beginning of the string, 0x04: end of string char
;*********************
            PUSH{R0-R5, LR}
loop3       LDRB    R0, [R3], #1                ; load ansii code of the
character, post inc. address
            CMP     R0, #0x04                   ; has end character been
reached?
            BEQ     done                        ; if so, end
            ; find the memory index (M[i]), starting at ((ANSII_code -
20)*5), and 5 bytes
            ; see: INIT_MEM.s
            SUBS    R0, #0x20
            LSLNE   R2, R0, #2
```

```
            ADDNE   R0, R2, R0
            ; print five bytes for each char
            MOV     R1, #0
            MOV     R2, #5
            LDR     R5, =PNT
            ADD     R0, R5 ; r0: the pointer
chr         LDRB    R5, [R0]
            ; print one byte. initial index is in R5
            MOV     R4, #0x01; display is in the data mode
            BL      CMD5110                     ; write the char to the
display
            ADD     R0, #1 ; increment pointer
            ADD     R1, #1
            CMP     R1, R2
            BLT     chr
            B       loop3
done        POP{R0-R5, LR}
            BX LR
            ENDP
```

2.3.8. INIT_MEM.s

Also, I have INIT_MEM routine under the file INIT_MEM.s file, which holds an array of commands, which has a length of 5 to print each ASCII char on the screen. Again, I take the idea form [1]. The array is not in C file but in other supplementary materials (see: Appendix-1: ASCII Char to Nokia5510 Data Commands). A wrote this array in Assembly.

We can reach the memory offset for starting address of a command for each ASCII char by the formula: (ASCII_Code – 0x20)*5

References

[1] Nokia5110.c file at: http://users.ece.utexas.edu/~valvano/arm/

Appendices

Appendix-1: ASCII Char to Nokia5510 Data Commands

```
static const uint8_t ASCII[][5] = {
  {0x00, 0x00, 0x00, 0x00, 0x00} // 20
  ,{0x00, 0x00, 0x5f, 0x00, 0x00} // 21 !
  ,{0x00, 0x07, 0x00, 0x07, 0x00} // 22 "
  ,{0x14, 0x7f, 0x14, 0x7f, 0x14} // 23 #
  ,{0x24, 0x2a, 0x7f, 0x2a, 0x12} // 24 $
  ,{0x23, 0x13, 0x08, 0x64, 0x62} // 25 %
  ,{0x36, 0x49, 0x55, 0x22, 0x50} // 26 &
  ,{0x00, 0x05, 0x03, 0x00, 0x00} // 27 '
  ,{0x00, 0x1c, 0x22, 0x41, 0x00} // 28 (
  ,{0x00, 0x41, 0x22, 0x1c, 0x00} // 29 )
  ,{0x14, 0x08, 0x3e, 0x08, 0x14} // 2a *
  ,{0x08, 0x08, 0x3e, 0x08, 0x08} // 2b +
  ,{0x00, 0x50, 0x30, 0x00, 0x00} // 2c ,
  ,{0x08, 0x08, 0x08, 0x08, 0x08} // 2d -
  ,{0x00, 0x60, 0x60, 0x00, 0x00} // 2e .
  ,{0x20, 0x10, 0x08, 0x04, 0x02} // 2f /
  ,{0x3e, 0x51, 0x49, 0x45, 0x3e} // 30 0
  ,{0x00, 0x42, 0x7f, 0x40, 0x00} // 31 1
  ,{0x42, 0x61, 0x51, 0x49, 0x46} // 32 2
```

```
,{0x21, 0x41, 0x45, 0x4b, 0x31} // 33 3
,{0x18, 0x14, 0x12, 0x7f, 0x10} // 34 4
,{0x27, 0x45, 0x45, 0x45, 0x39} // 35 5
,{0x3c, 0x4a, 0x49, 0x49, 0x30} // 36 6
,{0x01, 0x71, 0x09, 0x05, 0x03} // 37 7
,{0x36, 0x49, 0x49, 0x49, 0x36} // 38 8
,{0x06, 0x49, 0x49, 0x29, 0x1e} // 39 9
,{0x00, 0x36, 0x36, 0x00, 0x00} // 3a :
,{0x00, 0x56, 0x36, 0x00, 0x00} // 3b ;
,{0x08, 0x14, 0x22, 0x41, 0x00} // 3c <
,{0x14, 0x14, 0x14, 0x14, 0x14} // 3d =
,{0x00, 0x41, 0x22, 0x14, 0x08} // 3e >
,{0x02, 0x01, 0x51, 0x09, 0x06} // 3f ?
,{0x32, 0x49, 0x79, 0x41, 0x3e} // 40 @
,{0x7e, 0x11, 0x11, 0x11, 0x7e} // 41 A
,{0x7f, 0x49, 0x49, 0x49, 0x36} // 42 B
,{0x3e, 0x41, 0x41, 0x41, 0x22} // 43 C
,{0x7f, 0x41, 0x41, 0x22, 0x1c} // 44 D
,{0x7f, 0x49, 0x49, 0x49, 0x41} // 45 E
,{0x7f, 0x09, 0x09, 0x09, 0x01} // 46 F
,{0x3e, 0x41, 0x49, 0x49, 0x7a} // 47 G
,{0x7f, 0x08, 0x08, 0x08, 0x7f} // 48 H
,{0x00, 0x41, 0x7f, 0x41, 0x00} // 49 I
,{0x20, 0x40, 0x41, 0x3f, 0x01} // 4a J
,{0x7f, 0x08, 0x14, 0x22, 0x41} // 4b K
,{0x7f, 0x40, 0x40, 0x40, 0x40} // 4c L
,{0x7f, 0x02, 0x0c, 0x02, 0x7f} // 4d M
,{0x7f, 0x04, 0x08, 0x10, 0x7f} // 4e N
,{0x3e, 0x41, 0x41, 0x41, 0x3e} // 4f O
,{0x7f, 0x09, 0x09, 0x09, 0x06} // 50 P
,{0x3e, 0x41, 0x51, 0x21, 0x5e} // 51 Q
,{0x7f, 0x09, 0x19, 0x29, 0x46} // 52 R
,{0x46, 0x49, 0x49, 0x49, 0x31} // 53 S
,{0x01, 0x01, 0x7f, 0x01, 0x01} // 54 T
,{0x3f, 0x40, 0x40, 0x40, 0x3f} // 55 U
,{0x1f, 0x20, 0x40, 0x20, 0x1f} // 56 V
,{0x3f, 0x40, 0x38, 0x40, 0x3f} // 57 W
,{0x63, 0x14, 0x08, 0x14, 0x63} // 58 X
,{0x07, 0x08, 0x70, 0x08, 0x07} // 59 Y
,{0x61, 0x51, 0x49, 0x45, 0x43} // 5a Z
,{0x00, 0x7f, 0x41, 0x41, 0x00} // 5b [
,{0x02, 0x04, 0x08, 0x10, 0x20} // 5c '\'
,{0x00, 0x41, 0x41, 0x7f, 0x00} // 5d ]
,{0x04, 0x02, 0x01, 0x02, 0x04} // 5e ^
,{0x40, 0x40, 0x40, 0x40, 0x40} // 5f _
,{0x00, 0x01, 0x02, 0x04, 0x00} // 60 `
,{0x20, 0x54, 0x54, 0x54, 0x78} // 61 a
,{0x7f, 0x48, 0x44, 0x44, 0x38} // 62 b
,{0x38, 0x44, 0x44, 0x44, 0x20} // 63 c
,{0x38, 0x44, 0x44, 0x48, 0x7f} // 64 d
,{0x38, 0x54, 0x54, 0x54, 0x18} // 65 e
,{0x08, 0x7e, 0x09, 0x01, 0x02} // 66 f
,{0x0c, 0x52, 0x52, 0x52, 0x3e} // 67 g
,{0x7f, 0x08, 0x04, 0x04, 0x78} // 68 h
,{0x00, 0x44, 0x7d, 0x40, 0x00} // 69 i
,{0x20, 0x40, 0x44, 0x3d, 0x00} // 6a j
,{0x7f, 0x10, 0x28, 0x44, 0x00} // 6b k
,{0x00, 0x41, 0x7f, 0x40, 0x00} // 6c l
,{0x7c, 0x04, 0x18, 0x04, 0x78} // 6d m
,{0x7c, 0x08, 0x04, 0x04, 0x78} // 6e n
,{0x38, 0x44, 0x44, 0x44, 0x38} // 6f o
```

```
  ,{0x7c, 0x14, 0x14, 0x14, 0x08} // 70 p
  ,{0x08, 0x14, 0x14, 0x18, 0x7c} // 71 q
  ,{0x7c, 0x08, 0x04, 0x04, 0x08} // 72 r
  ,{0x48, 0x54, 0x54, 0x54, 0x20} // 73 s
  ,{0x04, 0x3f, 0x44, 0x40, 0x20} // 74 t
  ,{0x3c, 0x40, 0x40, 0x20, 0x7c} // 75 u
  ,{0x1c, 0x20, 0x40, 0x20, 0x1c} // 76 v
  ,{0x3c, 0x40, 0x30, 0x40, 0x3c} // 77 w
  ,{0x44, 0x28, 0x10, 0x28, 0x44} // 78 x
  ,{0x0c, 0x50, 0x50, 0x50, 0x3c} // 79 y
  ,{0x44, 0x64, 0x54, 0x4c, 0x44} // 7a z
  ,{0x00, 0x08, 0x36, 0x41, 0x00} // 7b {
  ,{0x00, 0x00, 0x7f, 0x00, 0x00} // 7c |
  ,{0x00, 0x41, 0x36, 0x08, 0x00} // 7d }
  ,{0x10, 0x08, 0x08, 0x10, 0x08} // 7e ~
//  ,{0x78, 0x46, 0x41, 0x46, 0x78} // 7f DEL
  ,{0x1f, 0x24, 0x7c, 0x24, 0x1f} // 7f UT sign
};
```

Appendix-2: Pin Configurations

Motor Control Pins

ULN2003A	Tiva Board
IN1	PB0
IN2	PB1
IN3	PB2
IN3	PB3

ADC Pins

Potentiometer: 3.3V, PE3, GND

Nokia5510 Pins

Signal	Nokia 5110	Tiva Board
Reset	RST (pin1)	PA7
SSI0Fss	CE (pin 2)	PA3
Data/Command	DC (pin 3)	PA6
SSI0Tx	Din (pin 4)	PA5
SSI0Clk	Clk (pin 5)	PA2
3.3V	Vcc (pin 6)	3.3V
back light	BL (pin 7)	3.3V
Ground	Gnd (pin8)	ground

HC-SR04 Pins

ECHO: PC6

TRIG: PF2

3.3V, GND