MBED Logic Analyzer

Eric Glunn, Benjamin Newberg, Ryan Dobbs, and Harley Ovell

April 2020

Contents

Chapter 1

Software

1.1 PC to MBED Packet Structure

Each 8-bit packet is analyzed based on its binary bit values.

1.1.1 Analog and Digital

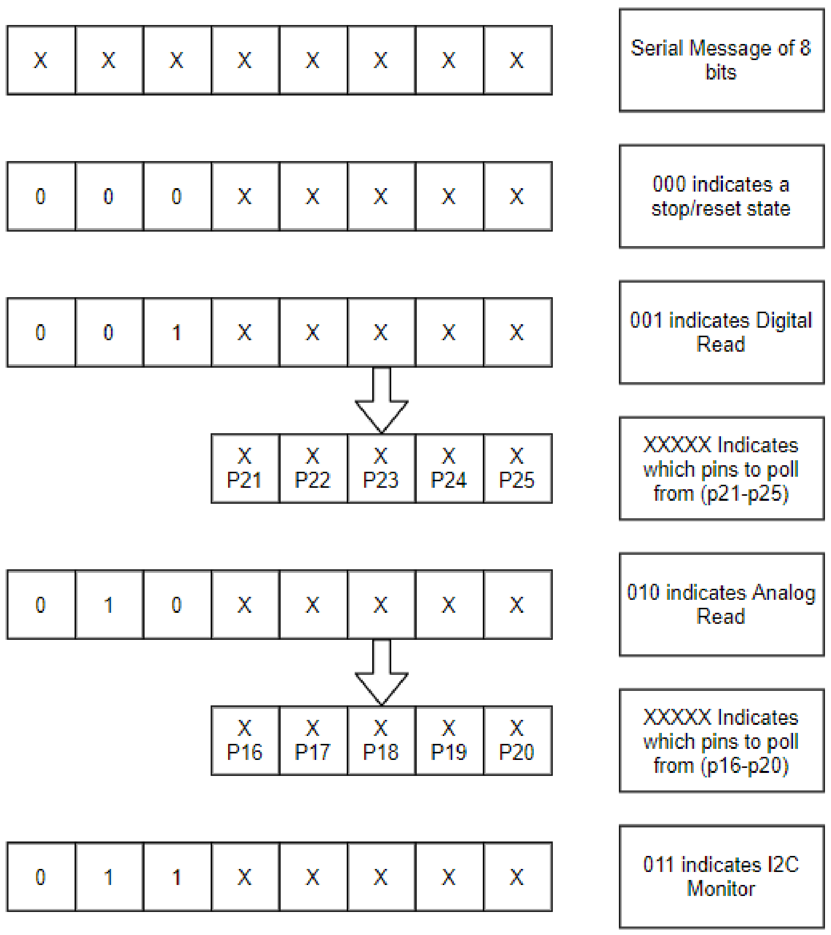
X X X X X X X X
Serial Message of 8 bits
0 0 0 X X X X X
000 indicates a stop/reset state
0 0 1 X X X X X
001 indicates Digital Read
X P21 X P22 X P23 X P24 X P25
XXXXX Indicates which pins to poll from (p21-p25)
0 1 0 X X X X X
010 indicates Analog Read
X P16 X P17 X P18 X P19 X P20
XXXXX Indicates which pins to poll from (p16-p20)
0 1 1 X X X X X
011 indicates I2C Monitor

Figure 1.1: Bit wise explanation for a few analysis options

Examples for Figure 1.1:

```
Digital read on pin 21 : "0b00110000"
Digital read on pins 21 and 23: "0b00110100"
Analog read on pins 18 and 19: "0b01000110"
Reset MBED: "0b000XXXXX"
```

1.1.2 Oscilloscope

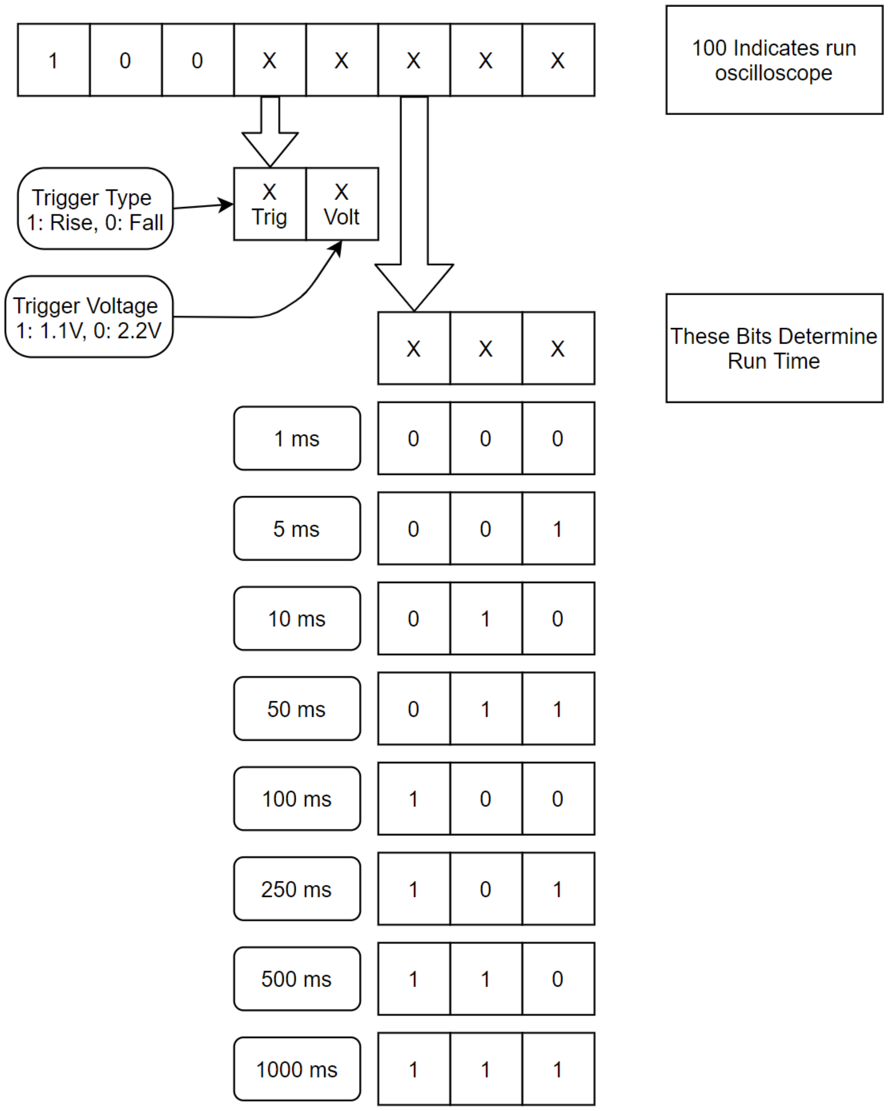

Figure 1.2: Bit wise explanation for oscilloscope

Examples for Figure 1.2

```
Run Oscope, rising trigger @ 1.1V, 5 ms : "0b10010001"
Run Oscope, rising trigger @ 2.2V, 5 ms : "0b10011001"
Run Oscope, falling trigger @ 2.2V, 100ms:"0b10001100"
Run Oscope, falling trigger @ 1.1V, 250ms:"0b10001101"
```

1.2 MBED to PC Packet Structure

Overview

All code for the MBED can be downloaded from the following link
`https://os.mbed.com/users/EricGlunn/code`

1.2.1 Digital Reader

For the Digital Reader, the MBED will send the PC a 1 char packet every 500ms. The first three bits of the packet will specify that it is running the digital reader ("0b001XXXXX"), the remaing 5 bits will specify whether the selected pins are high or low. If a pin is not selected, the packet will always have a 0 in the pins place, if it is selected it will have a 1 for high and 0 for low.

Examples:

```
Reading pins 21 and 25 (both high) : "0b00110001"
Reading pins 21-25 (23 and 24 high): "0b00100110"
```

1.2.2 Analog Reader

For the Analog Reader, the MBED will send the PC 7 char packets every 500ms. The first packet will be the analog mask "0b11111111", the 2nd through 6th will be the analog values (if a pin is not selected the value sent will be zero), finally the last packet will be the initial command sent from the PC.

Examples:

```
Analog reading on pins 17 and 18: "[0b11111111] [0b00000000] [0bXXXXXXXX]
[0bXXXXXXXX] [0b00000000] [0b00000000] [0b01001100]"

Analog reading on pins 19 and 20: "[0b11111111] [0b00000000] [0b00000000]
[0b00000000] [0bXXXXXXXX] [0bXXXXXXXX] [0b01000011]"

Analog reading on pins 16 and 20: "[0b11111111] [0bXXXXXXXX] [0b00000000]
[0b00000000] [0b00000000] [0bXXXXXXXX] [0b01010001]"
```

1.2.3 Oscilloscope

When the oscilloscope command is sent, the MBED will sample analog pin 16 for the designated time, then send one long array of chars to the PC. The array always begins with "0b11111111", and has no end char. Each sample is spaced 50 μs apart. So a formula to calculate array length is available in equation 1.1. Converting to voltage can be done using equation 1.2.

$$Array_{length} = 1 + 20 \cdot T \tag{1.1}$$

$$Voltage = Char \cdot \frac{3.3}{254} \tag{1.2}$$

Examples:

```
Oscope read for 10ms: "[0b11111111] [char 0 (0 us)] [char 1 (50 us)] [char 2 (100 us)]
... [char 200 (10000 us)]"
```

1.3 Digital Reader

1.3.1 Explanation

To run the MBED digital reader, the PC will specify it wants to perform a digital read and which pins to read from by sending the char "0b001XXXXX". When the MBED receives this char, the sampling function ("sample_func") will be attached to the ticker "tickerboi". Every 500ms the ticker will call the function. This function reads from each of the specified pins, Oring it together and will return an unsigned char to the computer every time the function runs. When the PC sends the MBED "0b000XXXXX", the MBED will treat this as a return to the off state and detach the digital sampling function from the ticker.

1.3.2 Digital Code

```
#include "mbed.h"

#define reset       0b00000000          //Define masks
#define digital     0b00100000
#define analog      0b01000000
#define stateMask   0b11100000
#define pinMask     0b00011111
#define pin1        0b00010000
#define pin2        0b00001000
#define pin3        0b00000100
#define pin4        0b00000010
#define pin5        0b00000001


DigitalOut myled(LED1);
DigitalOut DigitalStatus(LED2);

DigitalIn  d1(p21);                     //Always set these pins for digital in
DigitalIn  d2(p22);
DigitalIn  d3(p23);
DigitalIn  d4(p24);
DigitalIn  d5(p25);

Serial PC(USBTX,USBRX);
Ticker tickerboi;                                                   //Initialize the ticker for Sampling

volatile uint8_t DigitalSample;
volatile unsigned char command;
volatile unsigned char pins;
volatile bool DigitalRunning;
volatile unsigned char state;
volatile bool newdigital=false;




void sample_func(void)                          //sampling function, samples each pin specified and Ors it
{
    DigitalSample = digital;

    if( (command & pin1) == pin1) {
        DigitalSample=DigitalSample +  d1*pin1;
    }
    if( (command & pin2) == pin2) {
        DigitalSample=DigitalSample + d2*pin2;
    }
    if( (command & pin3) == pin3) {
        DigitalSample=DigitalSample + d3*pin3;
    }
    if( (command & pin4) == pin4) {
        DigitalSample=DigitalSample + d4*pin4;
    }
    if( (command & pin5) == pin5) {
        DigitalSample=DigitalSample + d5*pin5;
    }
    newdigital=true;             //tell main loop new data available to send
}

void SerialInterrupt(void)
{
    command=PC.getc();
    state= command & stateMask;
    switch(state) {
        case reset:
            tickerboi.detach();           //detach Ticker
            DigitalRunning=false;             //turn off status LED
            break;

        case digital:
            tickerboi.attach(&sample_func, .5);          //Ticker will call sample func every 100 ms
            DigitalRunning=true;                        // Bool to control status LED
            break;
    }
}

int main()
{
    PC.attach(&SerialInterrupt, Serial::RxIrq);

    while(1) {

        DigitalStatus=DigitalRunning;        //Status LED = Digital Running bool
        if(newdigital) {                     //if the sample_func was run, this will send the new data
            PC.putc(DigitalSample);
            newdigital=false;               //new data sent, no need to send anything else
        }
        myled = 1;                                //indicate the Main loop is running
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

1.4 Analog Reader

1.4.1 Explanation

To run the MBED Analog reader, the PC will specify it wants to perform an analog read and which pins to read from by sending the char "0b010XXXXX". When the MBED receives this char, the sampling function "analog_sample_func" will be attached to the ticker "tickerboi". Every 500ms the ticker will call the function. This function reads from each of the specified pins and places the analog value in the unsigned char array "AnalogSample". When the PC sends the MBED "0b000XXXXX", the MBED will treat this as a return to the off state and detach the analog sampling function from the ticker.

1.4.2 Analog Code

```
#include "mbed.h"
#define reset       0b00000000          //Define masks
#define analog      0b01000000
#define stateMask   0b11100000
#define pinMask     0b00011111
#define pin1        0b00010000
#define pin2        0b00001000
#define pin3        0b00000100
#define pin4        0b00000010
#define pin5        0b00000001
#define FULL        0b11111111


DigitalOut myled(LED1);
DigitalOut AnalogStatus(LED3);          //LED for when Analog read is running

AnalogIn   a1(p16);                     //Always set these pins for analog in
AnalogIn   a2(p17);
AnalogIn   a3(p18);
AnalogIn   a4(p19);
AnalogIn   a5(p20);

Serial PC(USBTX,USBRX);
Ticker tickerboi;                     //Initialize the ticker for Sampling

volatile unsigned char AnalogSample[5];
volatile bool AnalogRunning;
volatile unsigned char command;
volatile unsigned char state;
volatile bool newAnalog=false;

void analog_sample_func(void)
{
    for(int i=0; i<5; i++) {
        AnalogSample[i]=0;
    }
    if( (command & pin1) == pin1) {
        AnalogSample[0]=a1*254;
    }
    if( (command & pin2) == pin2) {
        AnalogSample[1]=a2 * 254;
    }
    if( (command & pin3) == pin3) {
        AnalogSample[2]=a3*254;
    }
    if( (command & pin4) == pin4) {
        AnalogSample[3]=a4*254;
    }
    if( (command & pin5) == pin5) {
        AnalogSample[4]=a5*254;
    }
    newAnalog=true;

}


void SerialInterrupt(void)
{
    command=PC.getc();
    state= command & stateMask;
    switch(state) {
        case reset:
            tickerboi.detach();                 //detach Ticker
            AnalogRunning=false;
            break;

        case analog:
            tickerboi.attach(&analog_sample_func, 1);        //Ticker will call analog read every 10ms
            AnalogRunning=true;
    }
}

int main()
{
    PC.attach(&SerialInterrupt, Serial::RxIrq);

    while(1) {

        AnalogStatus=AnalogRunning;                 //Indicate analogRead is running
        if(newDigital) {                           //if digital sampling was run
            PC.putc(DigitalSample);
            newDigital=false;
        }
        if(newAnalog) {                            //if analog sampling was run
            PC.putc(FULL);                        //First bit is 0b11111111
            for(int i=0; i<5; i++) {               //Send all of the analog bits
                PC.putc(AnalogSample[i]);
            }
            PC.putc(command);                         //Last bit is command from PC
            newAnalog=false;
        }

        myled = 1;                                 //indicate main loop is running
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

1.5 Oscilloscope

1.5.1 Explanation

When the PC sends the start command ”0b100XXXXX” to the MBED, the MBED will turn on the oscilloscope status LED and set the oscilloscope poll bool to true. This bool calls the ”oScope_poll_trig()” function in the main loop. This function does three main things. 1. The beginning of the function will call a while loop that holds the processor there until the oscilloscope trigger has been met. Once the trigger condition is satisfied, 2. The function then uses a switch statement to attach a ticker (oScopeCut) to call the oscilloscope kill function based on the run time specified by the PC command. Finally, 3, a ticker (oScopeTicker) is attached to the oscilloscope sampling function (oScope_sample_func). From here the code returns to the main loop and will remain there just blinking the myled LED. While in the main loop, every 50 μs the sampling function is called by the ticker (OscopeTicker). The function will read the analog voltage on pin 16 and record this in the array ”oScopeArray”. When the run time is up, the ticker ”oScopeCut” will run and this will call the oscilloscope termination fucntion ”oScope_kill_func”. This function detaches both the termination and sampling tickers, declares there to be a new message ready to send, and finally turns off the oscilloscope status LED. Lastly, when the processor returns back to the main loop, the if statement will be satisfied and the messages will be sent from the array to the PC.

1.5.2 Oscilloscope Function Diagram

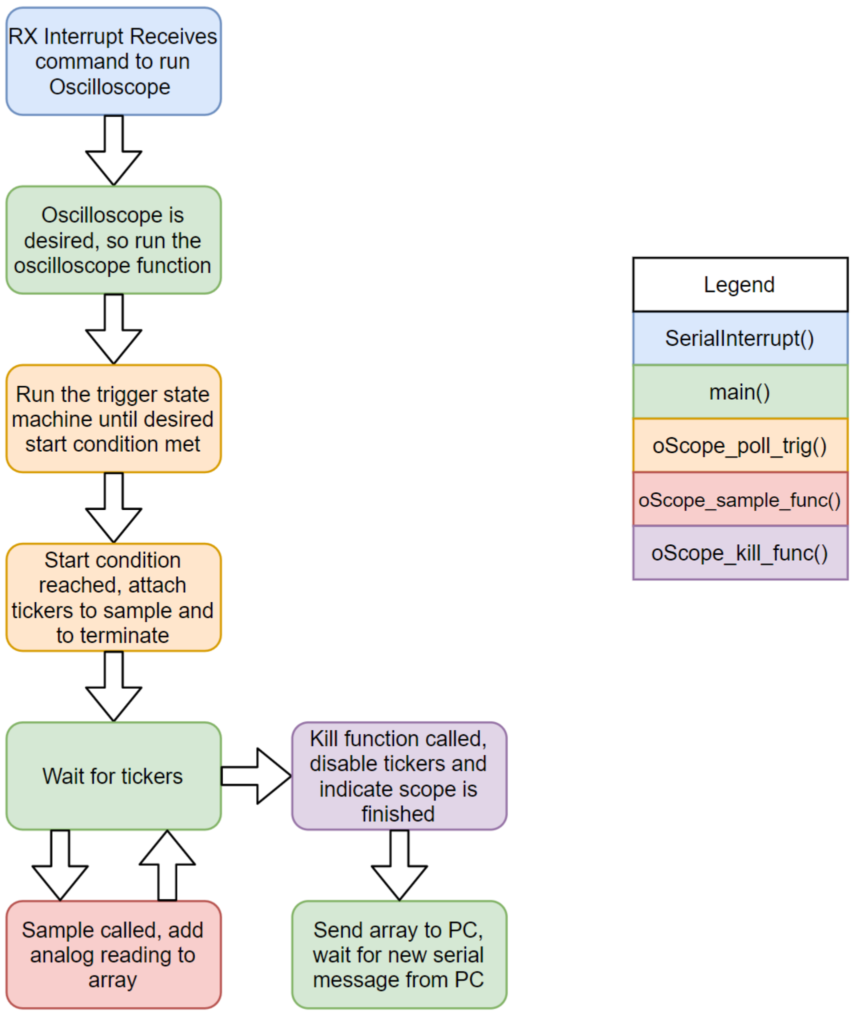
RX Interrupt Receives command to run Oscilloscope
Oscilloscope is desired, so run the oscilloscope function
Run the trigger state machine until desired start condition met
Start condition reached, attach tickers to sample and to terminate
Wait for tickers
Kill function called, disable tickers and indicate scope is finished
Sample called, add analog reading to array
Send array to PC, wait for new serial message from PC
Legend
SerialInterrupt()
main()
oScope_poll_trig()
oScope_sample_func()
oScope_kill_func()

Figure 1.3: Summary of function flow

1.5.3 Oscilloscope Trigger State Machine

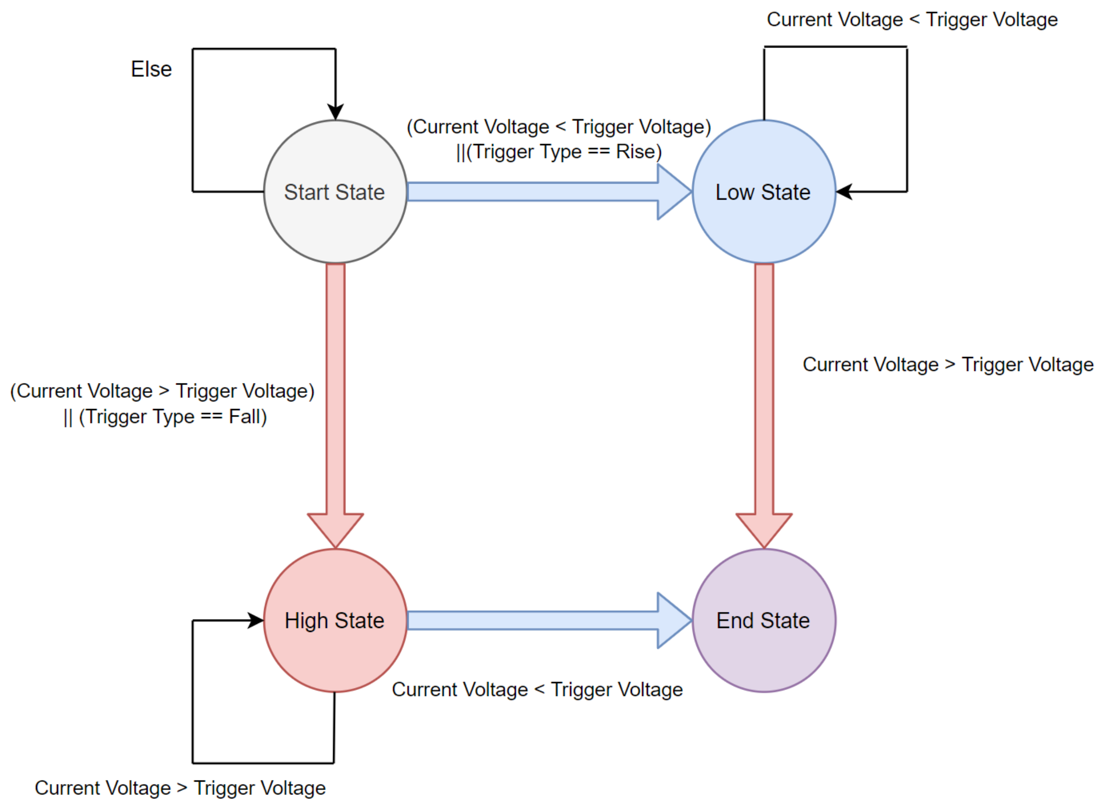
Current Voltage < Trigger Voltage
Else
(Current Voltage < Trigger Voltage)
||(Trigger Type == Rise)
Start State
Low State
Current Voltage > Trigger Voltage
(Current Voltage > Trigger Voltage)
|| (Trigger Type == Fall)
High State
End State
Current Voltage < Trigger Voltage
Current Voltage > Trigger Voltage

Figure 1.4: State machine of the trigger function

1.5.4 Oscilloscope Code

```
#include "mbed.h"

#define reset       0b00000000      //Define function masks
#define oScope      0b10000000

#define stateMask   0b11100000      //general masks
#define pinMask     0b00011111
#define FULL        0b11111111

#define ms1         0b00000000      //Oscope masks
#define ms5         0b00000001
#define ms10        0b00000010
#define ms50        0b00000011
#define ms100       0b00000100
#define ms250       0b00000101
#define ms500       0b00000110
#define ms1000      0b00000111
#define rise        0b00010000
#define fall        0b00000000
#define timeMask    0b00000111
#define triggerType 0b00010000
```

```
#define triggerVolt 0b00001000
#define volt11      0b00000000
#define volt22      0b00001000

DigitalOut myled(LED1);
DigitalOut oScopeStatus(LED4);          //LED for Oscilloscope



AnalogIn   a1(p16);                     //Always set these pins for analog in

Serial PC(USBTX,USBRX);

Ticker oScopeTicker;        //ticker that will sample
Ticker oScopeCut;           //ticker that will detach

volatile unsigned char command;
volatile unsigned char state;

volatile unsigned char oScopeTime;      //char to hold the data on how long to run the scope
volatile bool oScope_poll;
volatile bool newoScope=false;
volatile unsigned char oScopeArray[20000];  //huge array to hold samples
volatile int oScopeCount=0;
volatile unsigned char oScopeTriggerType;   //char that says whether the trigger is rise or fall
volatile unsigned char oScopeTriggerVoltageBit;     //intermediate trigger voltage storage
volatile unsigned char oScopeTriggerVoltageVal;     //final trigger voltage

volatile bool oScopeRunning;        //is the oscope function running?

void oScope_kill_func(void)         //kill the function
{
    oScopeTicker.detach();
    oScopeCut.detach();
    newoScope=true;                 //hey send the data to the PC
    oScopeRunning=0;
}

void oScope_sample_func(void){
    oScopeArray[oScopeCount++]=a1*254;          //sample and add to storage array
}

void oScope_poll_trig(){
    bool nTriggered = true;             //state machine while loop run
    myled=1;                            //indicate trigger state machine running
    unsigned char trig;
    int state =0;                       //state machine in restart state
    if(oScopeTriggerVoltageBit == volt11){oScopeTriggerVoltageVal=85;}   //configure trigger voltage
    else{oScopeTriggerVoltageVal=169;}

    while(nTriggered){              //run oscope state machine

        trig=a1*254;
        switch(state){

            case 0:
                if((trig < oScopeTriggerVoltageVal) & (oScopeTriggerType == rise)  ){state = 1;}
                else if((trig > oScopeTriggerVoltageVal) & (oScopeTriggerType == fall) ){state = 2;}
                else { state = 0;}
                break;
            case 1:
                if(trig < oScopeTriggerVoltageVal){state=1;}
                if(trig > oScopeTriggerVoltageVal){state=3;}
                break;
            case 2:
                if(trig > oScopeTriggerVoltageVal){state = 2;}
                if(trig < oScopeTriggerVoltageVal){state = 3;}
                break;
            case 3:
                nTriggered=false;
                break;
        }
        wait(.01);
    }
    myled=0;                    //no longer in trigger FSM
    switch(oScopeTime) {            //How long to run before stop

    case ms1:
        oScopeCut.attach(oScope_kill_func, .001);
        break;

    case ms5:
        oScopeCut.attach(oScope_kill_func, .005);
        break;

    case ms10:
        oScopeCut.attach(oScope_kill_func, .01);
        break;

    case ms50:
        oScopeCut.attach(oScope_kill_func, .05);
        break;

    case ms100:
        oScopeCut.attach(oScope_kill_func, .1);
        break;

    case ms250:
        oScopeCut.attach(oScope_kill_func, .25);
        break;

    case ms500:
        oScopeCut.attach(oScope_kill_func, .5);
        break;

    case ms1000:
        oScopeCut.attach(oScope_kill_func, 1);
        break;
    default:
        oScopeCut.attach(oScope_kill_func, .01);
        break;
    }

    oScopeTicker.attach_us(oScope_sample_func, 50);
```

```
}


void SerialInterrupt(void)
{
    command=PC.getc();
    state= command & stateMask;

    switch(state) {
        case reset:
            if(oScopeStatus){ oScope_kill_func(); }
            break;

        case oScope:
            oScopeTime=command & timeMask;
            oScope_poll=true;
            oScopeRunning=1;
            oScopeTriggerType= command & triggerType;
            oScopeTriggerVoltageBit = command & triggerVolt;
            break;
}

int main()
{
    PC.attach(&SerialInterrupt, Serial::RxIrq);


    while(1) {

        if(oScope_poll){
            oScope_poll=false;
            oScope_poll_trig(); }

        if(newoScope){
            PC.putc(FULL);
            for(int j=0; j<oScopeCount; j++){
                PC.putc(oScopeArray[j]);
            }
            oScopeCount=0;
            newoScope=false;
        }


        myled = 1;                              //indicate main loop is running
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

1.6 I2C

1.6.1 Explanation

A potential way to capture an I2C packet is to have an I2C state machine running in software, where the states change based on rise events from a pin tied to SCL. While the state machine is running and keeping track of the address and data bits, a separate sampling function could be called using a ticker which samples the analog line in between state events. If the ticker is fast enough it could sample the voltage of the line many times in between states, recording this data and delivering it to the PC. In the end the PC will have both the exact bit values of the address and data as well as an array of all the analog activity on the line while this took place.

A separate method could be to just sample the data and clock lines as fast as possible and then export this data to the PC. Then the PC would simply look at the data and match the rise on the clock line with the data line and extract the address and data bits this way.

Shell code for the first method is provided below, however the second method may be more useful and significantly simpler to design.

1.6.2 A Proposed State Machine

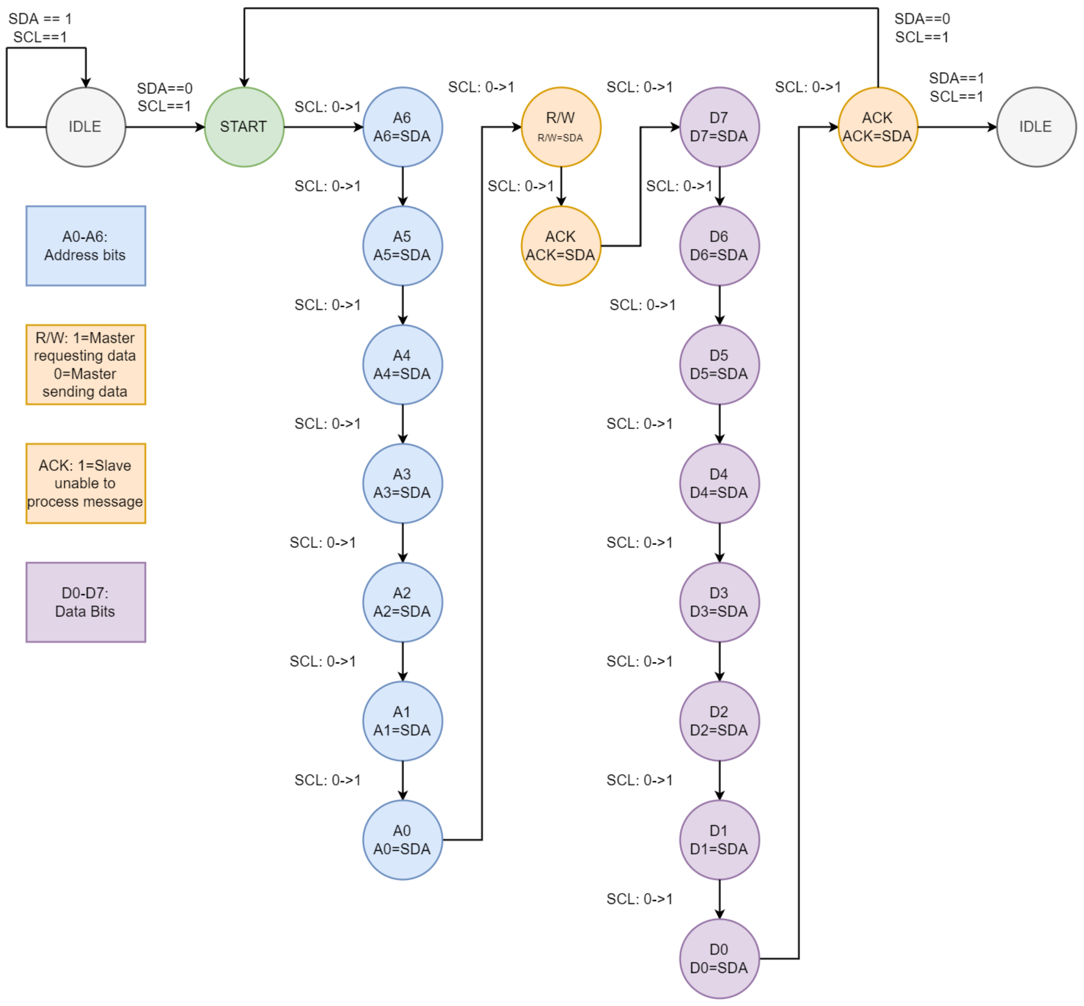

Figure 1.5: A simplified I2C state machine

1.6.3 I2C Shell Code

```
#include "mbed.h"

#define reset       0b00000000     //Define function masks
#define digital     0b00100000
#define analog      0b01000000
#define I2C         0b01100000
#define oScope      0b10000000

#define stateMask   0b11100000     //general masks
#define pinMask     0b00011111
#define FULL        0b11111111


DigitalOut myled(LED1);


AnalogIn   a1(p16);                    //SDA
InterruptIn i1(p15);                   //SCL

volatile bool I2CRunning=false;
volatile bool newI2CData=false;
Ticker I2C_ticker;


volatile unsigned char command;
volatile unsigned char state;
Serial PC(USBTX,USBRX);


void I2C_state_machine(void){
        //Implement the state machine here to record address, data, R/W, ACK
}

void I2C_sample_func(void){
        //This function will sample the SDA line every 10uS
}

void SerialInterrupt(void)
{
    command=PC.getc();
    state= command & stateMask;

    switch(state) {
        case reset:

            I2CRunning=false;
            I2C_ticker.detach();
            newI2CData=true;


            break;

        case I2C:
            I2CRunning=true;
            I2C_ticker.attach_us(I2C_sample_func, 10);
            break;

    }


}

int main()
{
    PC.attach(&SerialInterrupt, Serial::RxIrq);
    i1.rise(&I2C_state_machine);                //here is the interrupt that will call the state machine

    while(1) {

        while(I2CRunning){       //Hang here and do nothing


        }
        if(newI2CData){
            //I2C done running send the data to PC
            newI2CData=false;
        }



        myled = 1;                              //indicate main loop is running
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```