Oregon State University

ECE 342

Junior Design Project

Junior Design Final Project

USB Oscilloscope

Authors
Brendan Woodrow
Tomas McMonigal

Group
3

February 20, 2020

Contents

1 Project Overview

This Project is a two channel USB Oscilloscope. Voltage is converted into digital values using an ADC, and rotary encoders are used to set the displays time scale, voltage scale and trigger level.The System Uses an FPGA to control data acquisition and to send data over USB to a computer. That data is read by the computer and QT development framework is used to process and display the data on the computer screen. Figure 1 is the systems top level diagram. Interface definitions for each sub block can be found in the following sections.

1.1 Features

- Two Channels
- 1MHz sampling rate
- AC/DC coupling modes
- Adjustable time and voltage scale for each channel
- Adjustable trigger for each channel

2 System Top Level Block Diagram

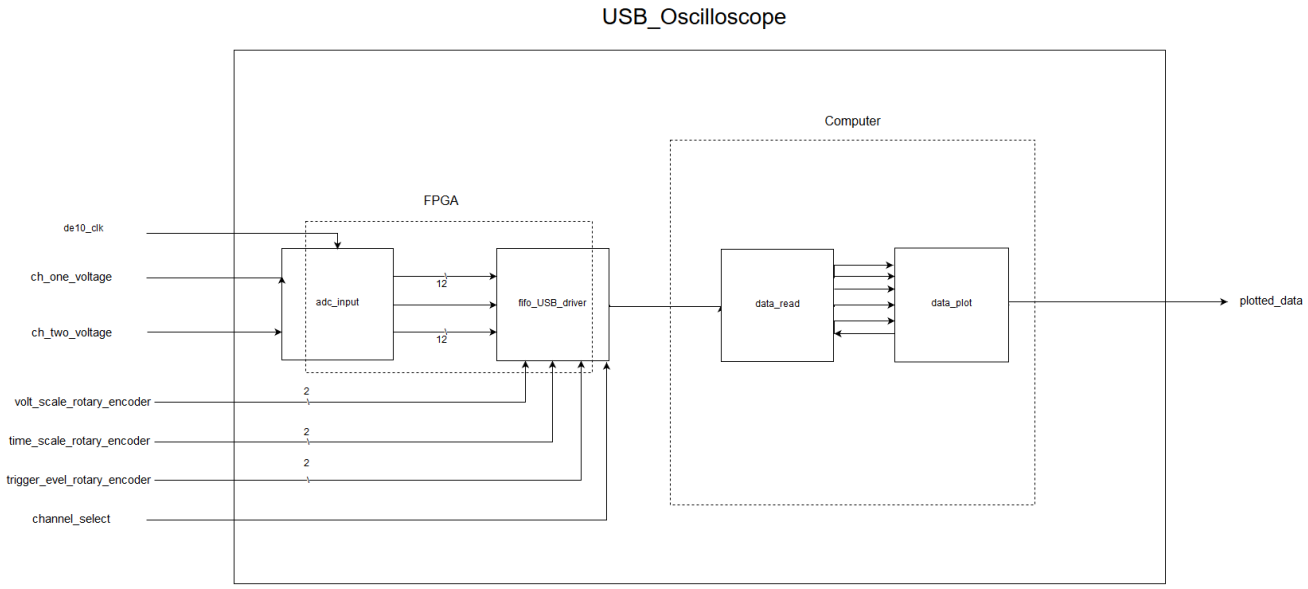

Figure 1: System Top Level Diagram

Figure 1 is the systems top level bock diagram. Voltage and rotary encoder data is read into the FPGA and sent over USB to the computer using the adc_input and fifo_USB_driver blocks. Data is read into the computer using the data_read block and is displayed on the computer screen using the data_plot block.The top level interface are defined in table 1

2.1 Interface Definitions

Interface	Definition
de10_clk	50Mhz clock signal generated on de10 board
ch_one_voltage	Channel Ones probed Voltage Range:-3.3V to 3.3V
ch_two_voltage	Channel Twos probed Voltage Range: -3.3v to 3.3v
volt_scale_rotary_encoder	2 bit value of rotary encoder controlling voltage scale Range: 2'd0 - 2'd3
time_scale_rotary_encoder	2 bit value of rotary encoder controlling time scale Range: 2'd0 - 2'd3
trigger_level_rotary_encoder	2 bit value of rotary encoder controlling trigger level Range: 2'd0 - 2'd3
channel_select	1 bit value of switch controlling which channel rotary encoders are controlling Range: 1'b0 - 1'b1
plotted_data	Voltage data plotted in GUI, displayed on computer screen Ac/DC coupled ch_one_voltage and ch_two_voltage wave forms are plotted In Ac coupling mode all DC components of the input voltages have gone to 0 In DC coupling mode the DC component of the input wave forms is still displayed

Table 1: Top Level Block Diagram Interface Definitions

3 adc_input Block

3.1 adc_input Top Level Diagram

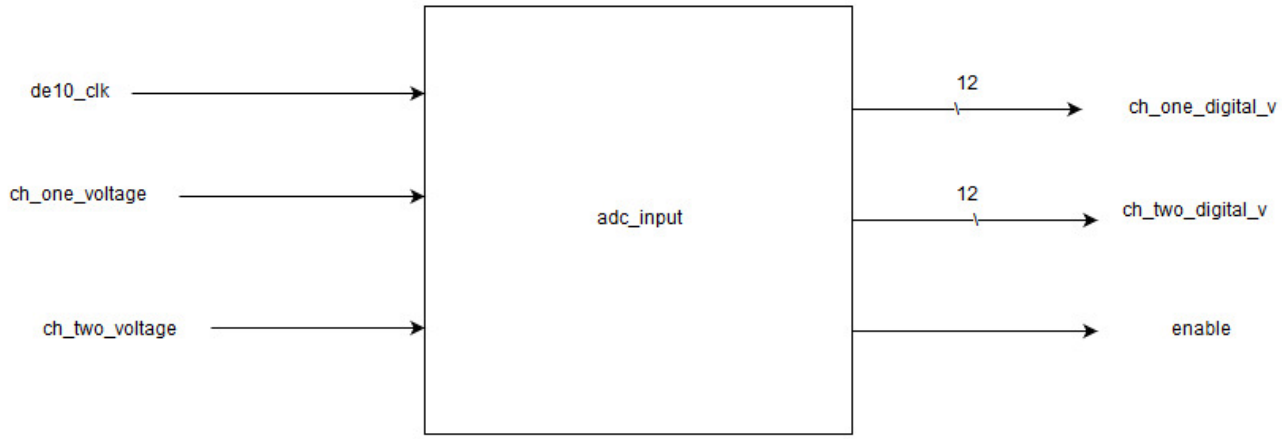

Figure 2: System Top Level Diagram

3.2 adc_input Top Level Interface definitions

Interface	Definition
de10_clk	50Mhz clock signal generated on de10 board signal is used to generate a 25Mhz clock signal which is used to control ADC data acquisition
ch_one_voltage	Probed Voltage, Un-level Shifted, pre ac/dc coupled Range:-3.3V to 3.3V
ch_two_voltage	Probed Voltage, Un-level Shifted, pre ac/dc coupled Range: -3.3v to 3.3v
enable	one bit active low signal to fifo_USB_driver block After all 12 bits of one sample has been read in from ADC this signal goes low for one 25MHz clock cycle A low enable signal tells the fifo_USB_driver a new sample is ready to be sent over USB to the computer
ch_one_digital_v	12 bit digital voltage value from adc -3.3v ch_one_voltage input voltage goes to 12'd4095, 3.3V input goes to 12'd0 Range: 12'd0 - 12'd4095
ch_two_digital_v	12 bit digital voltage value from adc -3.3v ch_one_voltage input voltage goes to 12'd4095, 3.3V input goes to 12'd0 Range: 12'd0 - 12'd4095

Table 2: adc_input top Level Block Diagram Interface Definitions

4 fifo_USB_driver Block

4.1 fifo_USB_driver Top Level Diagram

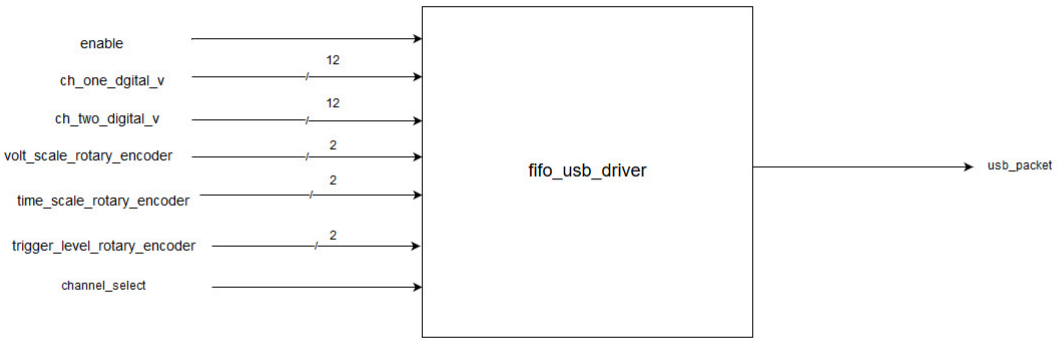

Figure 3: fifo_USB_driver top level block diagram

4.2 fifo_USB_driver Top Level Interface definitions

Interface	Definition
enable	one bit active low signal from adc_input on the falling edge of signal a new USB packet is sent into the fifo on the fifo-Usb and is ready to be read in form the computer
ch_one_digital_v	12 bit digital voltage value from adc -3.3v ch_one_voltage input voltage goes to 12'd4095, 3.3V input goes to 12'd0 Range: 12'd0 - 12'd4095
ch_two_digital_v	12 bit digital voltage value from adc -3.3v ch_one_voltage input voltage goes to 12'd4095, 3.3V input goes to 12'd0 Range: 12'd0 - 12'd4095
volt_scale_rotary_encoder	2 bit value of rotary encoder controlling voltage scale Range: 2'd0 - 2'd3
time_scale_rotary_encoder	2 bit value of rotary encoder controlling time scale Range: 2'd0 - 2'd3
trigger_level_rotary_encoder	2 bit value of rotary encoder controlling trigger level Range: 2'd0 - 2'd3
channel_select	1 bit value of switch controlling which channel rotary encoders are controlling Range: 1'b0 - 1'b1
usb_packet	13 byte usb packet sent to the computer in the form s409500001230 Where s signals the start of the packet followed by the 4 digits of channel 1 then the 4 digits of channel 2, then the 1 digit values of the voltage, time and trigger rotary encoders in that order followed by the value of the switch at the end

Table 3: fifo_usb_driver top Level Block Diagram Interface Definitions

5 data_read Block

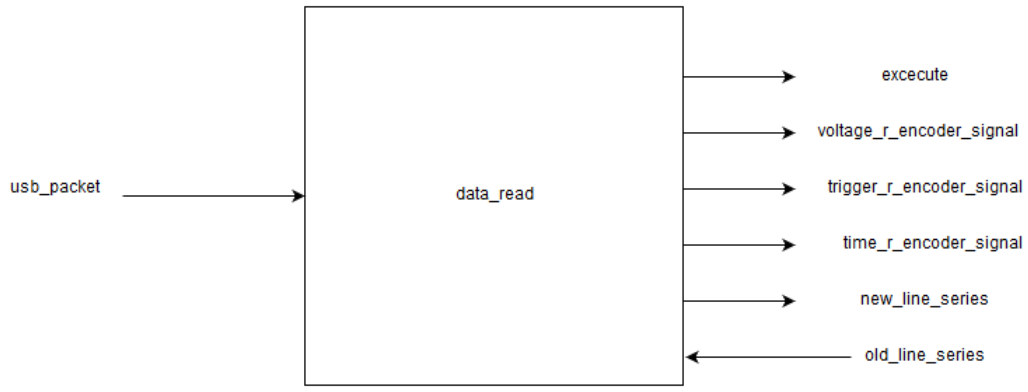

Figure 4: data_read top level block diagram

5.1 data_read Top Level Interface definitions

Interface	Definition
usb_packet	13 byte usb packet sent to the computer in the form s409500001230
execute	signal sent to data_plot at start of program, telling it to initialize its variables
voltage_r_encoder_signal	signal sent to data_plot telling it when the voltage r_encoder is turning clockwise or counter clockwise
trigger_r_encoder_signal	signal sent to data_plot telling it when the trigger r_encoder is turning clockwise or counter clockwise
time_r_encoder_signal	signal sent to data_plot telling it when the time r_encoder is turning clockwise or counter clockwise
old_line_series	old line_series data structure from data_plot the data structure is emptied and the new voltage x and y coordinates are loaded into it for each sample
new_line_series	the new refilled line_series is passed back to data_plot once all samples have been loaded into it

Table 4: data_read top Level Block Diagram Interface Definitions

6 data_plot Block

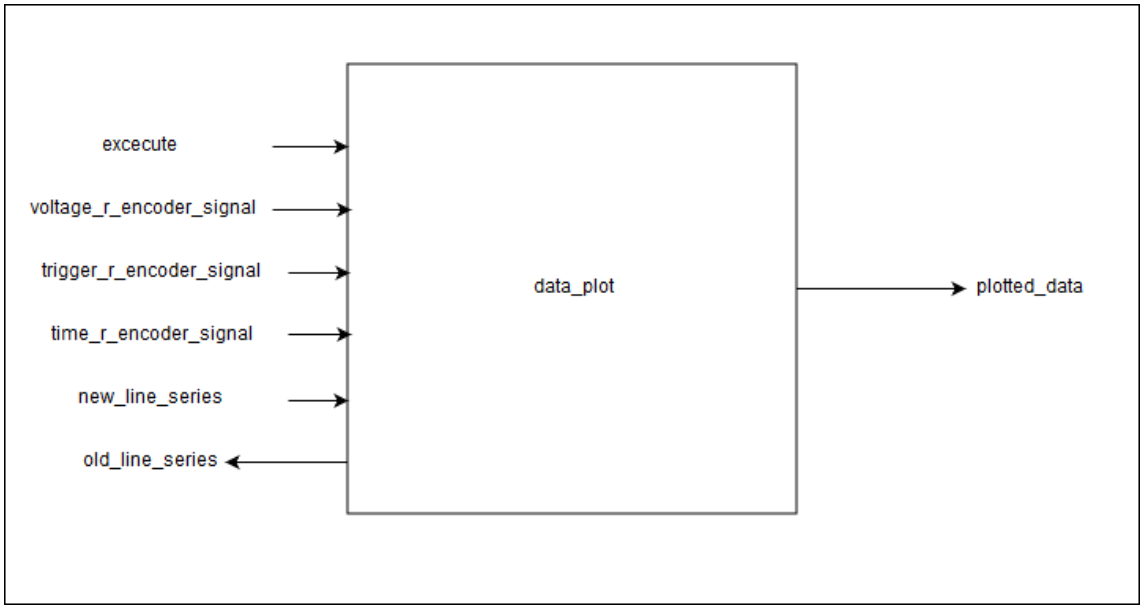

Figure 5: data_plot top level block diagram

6.1 data_plot Top Level Interface definitions

Interface	Definition
execute	signal sent to data_plot from data_read at start of program, telling it to initialize its
voltage_r_encoder_signal	signal from data_read telling it when the voltage r_encoder is turning clockwise or counter clockwise
trigger_r_encoder_signal	signal from data_read telling it when the trigger r_encoder is turning clockwise or counter clockwise
time_r_encoder_signal	signal from data_read telling it when the time r_encoder is turning clockwise or counter clockwise
old_line_series	old line_series data structure is passed to data_read to have new x and y coordinates loaded into it for each sample
new_line_series	the new refilled line_series is passed back to data_plot
plotted_data	Voltage data plotted in GUI, displayed on computer screen Ac/DC coupled ch_one_voltage and ch_two_voltage wave forms are plotted In Ac coupling mode all DC components of the input voltages have gone to 0 In DC coupling mode the DC component of the input wave forms is still displayed

Table 5: data_plot Level Block Diagram Interface Definitions

7 PCB Layers

7.1 Front Copper Layer

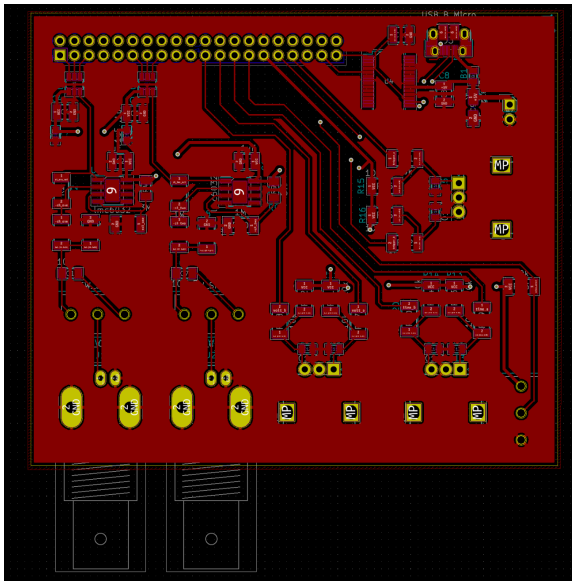

Figure 6: PCB Front Copper Layer

7.2 Back Copper Layer

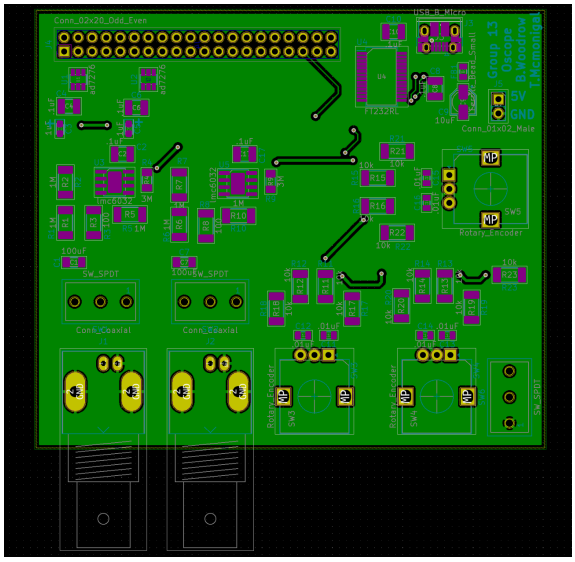

Figure 7: PCB Back Copper Layer

8 Project Schematic

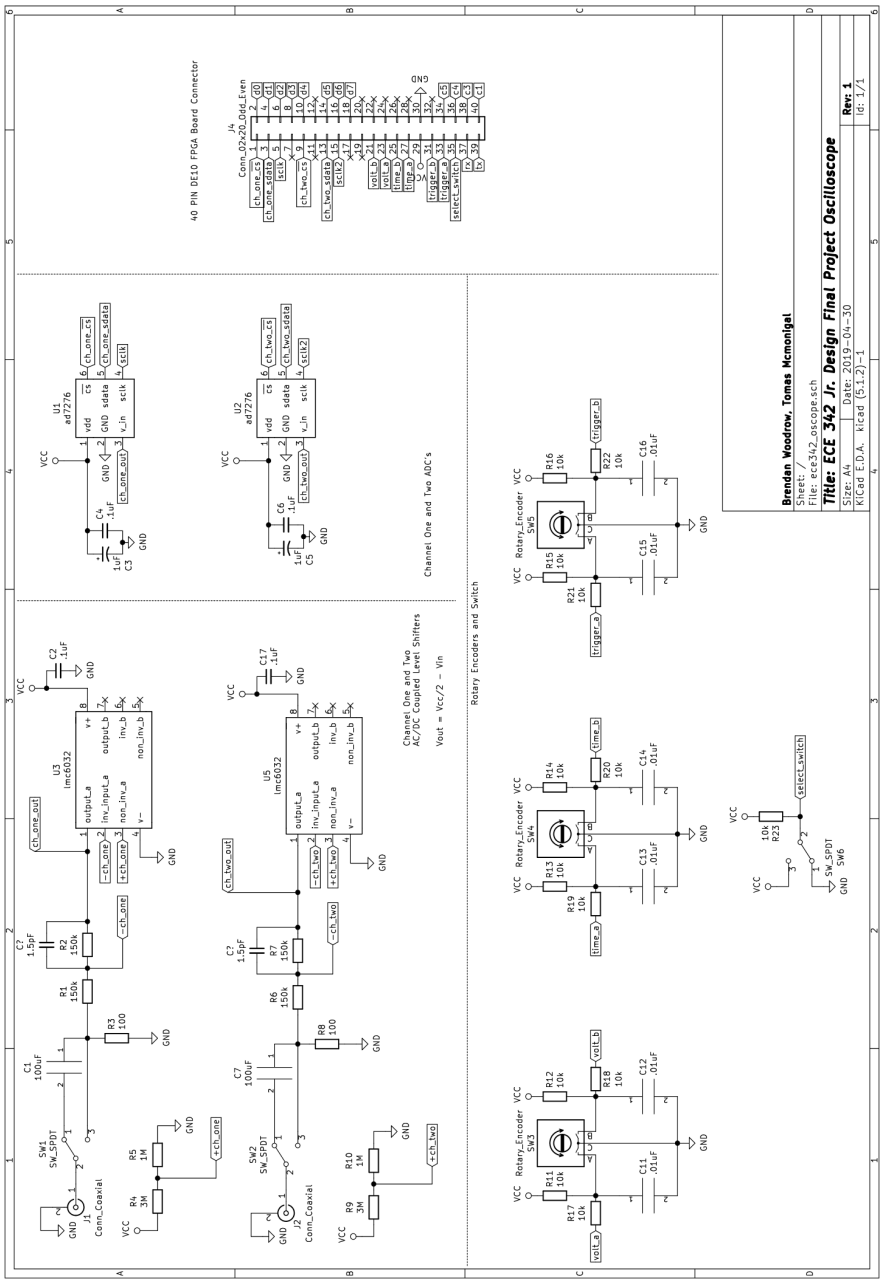

Figure 8: system schematic

9 Bill of Materials

Google sheets link to spread sheet:
https://drive.google.com/open?id=1NuWa4Z1dMhkNWoJHjqEXBRm4rsR5HQQH0FSQQlBqDKQ

A SystemVerilog Files

A.1 adc_input block code

A.1.1 adc_driver.sv (block's top module)

```
module adc_driver(

        input logic clk,                //50Mhz clock
        input logic sdata_ch2,  //serial data from adc
        input logic sdata_ch1,
        input logic reset_n, //reset

        output logic sclk,      //25Mhz clock tro adc

        output logic cs,                //controll signal to adc

        output logic enable, //enable signal to fifo to start new burt of data


        output logic [11:0] data_ch1, //12 bit digital data read in from adc
        output logic [11:0] data_ch2,


 /*
        // seven seg output logic if seven seg driver module is connected

        output logic [6:0] h0,

        output logic [6:0] h1,

        output logic [6:0] h2,

        output logic [6:0] h3

*/

        );

        logic [1:0]state;               //state of fsm
        logic [4:0] count;      //count of counter
        logic reset_count;      //reset the counter


        clock_counter_adc clk_count(
                .clk_in(clk),
                .reset_n(reset_n),
                .clk_slow(sclk)


        );


        counter counter(
                .sclk(sclk),
                .reset_n(reset_count),
                .count(count)

        );

        fsm fsm(                        //fsm drives sipo for both channels

                .clk(sclk),
                .reset_n(reset_n),
                .count(count),
                .cs(cs),
                .reset_cnt_n(reset_count),
                .state(state),
                .enable(enable)
        );

        sipo sipo_ch_one(       //sipo for channel one

        .sclk(sclk),

        .sdata(sdata_ch1),

        .state(state),

        .data(data_ch1)


        );

        sipo sipo_ch_two(               //sipo for channel two

        .sclk(sclk),

        .sdata(sdata_ch2),

        .state(state),

        .data(data_ch2),


        );

        /*

        LED_FSM_top LED_FSM_top(  //Can Output one of the channels digital value to 7 seb to debug

        .clk(sclk),

        .num(data_ch2),

        .h0(h0),

        .h1(h1),

        .h2(h2),

```

```
107           .h3(h3)
108
109
110           );
111
112
113
114   */
115
116
117   endmodule
```

A.1.2 clock_counter.sv

```
1   //This module is used to slow the de10's 50Mhz clk down to 25Mhz for ADC data transfer
2
3   module clock_counter_adc(
4
5           input logic clk_in,             //50 Mhz clock in
6
7           input logic reset_n,            //reset
8
9           output logic clk_slow    //25Mhz clock for sckl
10
11
12
13          );
14
15          logic [13:0] count;             //clk_in count
16
17
18
19
20          always_ff @ (posedge clk_in, negedge reset_n)
21                  begin
22
23
24                          if(!reset_n)            //if reset clk goes low and count is 0
25                                  begin
26                                          clk_slow <= 1'b0;
27                                          count <= 1'd0;
28                                  end
29
30                          else if(count >= 14'd0)         //count this many cyles of input clk
31                                  begin
32                                          clk_slow <= ~clk_slow;
33                                          count <= 1'd0;
34                                  end
35                          else
36                                  begin
37                                          count <= count + 1'd1;  //if its not been more than 2 posedges
                                                increment count
38                                  end
39
40
41                  end
42
43
44
45  endmodule
```

A.1.3 counter.sv

```
1   module counter( //counts the  number of clock cycles since the last resent_n
2
3           input logic sclk,               //25Mhz clock
4
5           input reset_n,                  //active low reset count to 0
6
7           output logic [4:0] count
8
9           );
10
11          always_ff @(negedge sclk, negedge reset_n)      //count the number of negative falling edge
12                  begin
13                          if(!reset_n)                    //if reset set count to 0
14                                  count <= 5'd0;
15                          else
16                                  count <= count + 5'd1;          //else increment count
17
18                  end
19
20
21  endmodule
```

A.1.4 adc_fsm.sv

```
1   module fsm(
2
3           input logic clk,                //25Mhz clock
4
5           input logic [4:0] count,                //count how many clock cycles have passed
6
7           input logic reset_n,                    //active low signal to reset module
8
9           output logic cs,                        //controll signal
10
11          output logic reset_cnt_n,       //reset counter
12
13          output logic [1:0] state,
14
15          output logic enable  //enable sent to fifo_usb_driver block during s1
16
17          );
18
19
20          logic [1:0] state_next;
21
22          parameter s0 = 2'd0;                    //Idle state, wait for  enough clock cycles for  t_quiet
                to pass
```

```
23          parameter s1 = 2'd1;                    //reset counter
24          parameter s2 = 2'd2;                    //read in data, wait for 14 clock cycles
25          parameter s3 = 2'd3;                    //reset counter
26
27          always_ff @ (posedge clk, negedge reset_n)
28                  begin
29                          if(!reset_n)
30                                  begin
31                                          state <= s0;
32                                  end
33
34                          else
35                                  begin
36                                          state <= state_next;            //state always goea to next
                                                state
37                                  end
38                  end
39
40          always_ff @ (*)
41                  begin
42
43                          case(state)
44                                  s0:begin                                                        //state s0
45
46
47                                          cs <= 1'b1;                                             //cs and count
                                                reset are both high
48                                          reset_cnt_n <= 1'b1;
49
50                                          enable <= 1'b1;
51
52                                          if(count < 5'd2)                        //if enough clk cycles
                                                havent passed stay in s0
53                                                  state_next <= s0;
54
55                                          else                                                    //else
                                                go to next state
56                                                  state_next <= s1;
57
58
59                                  end
60                                  s1:begin                                                //state s1
61
62
63                                          cs <= 1'b1;                                             //reset_cnt_n
                                                goes low to reset count
64                                          reset_cnt_n <= 1'b0;
65                                          state_next <= s2;                       //allways go to next
                                                state
66
67                                          enable = 1'b0;
68
69                                  end
70
71                          s2:begin                                                                //state
                                s2
72
73
74                                          cs <= 1'b0;                                             //cs goes low to
                                                start serial data transfer
75                                          reset_cnt_n <= 1'b1;
76
77                                          enable <= 1'b1;
78
79                                          if(count < 5'd14)               //if 14 cycles havent passed
                                                stay in s2
80                                                  state_next <= s2;
81                                          else
82                                                  state_next <= s3;               //after 14 clk cycles go
                                                        to next state
83
84                                  end
85
86                                  s3:begin                        // state s3
87
88
89                                          cs <= 1'b1;                                             //cs goes high
                                                to end serial transfer
90                                          reset_cnt_n <= 1'b0;            //reset the clock to get ready
                                                for s0
91                                          state_next <= s0;                       //always go to s0 next
92
93                                          enable <= 1'b1;
94                                  end
95
96                                  default:begin                       //default
97                                          cs <= 1'b1;
98                                          reset_cnt_n <= 1'b1;
99                                          state <= s0;
100                                         state_next <= s0;
101                                         enable <= 1'b1;
102                                 end
103                         endcase
104
105
106                 end
107
108 endmodule
```

A.1.5 sipo.sv

```
1   module sipo(
2
3           input logic sclk,                               //25Mhz clock to contoll data transfer from adc
4
5           input logic sdata,                      //serial data in
6
7           input logic [1:0] state,                //state from fsm
8
9           output logic [11:0] data                //12 bits data out
10
11
12
13          );
```

```
14
15          logic [11:0] q;          //shift register
16
17
18          always_ff @ (negedge sclk)              //always at neg edge clk
19                  begin
20                          case(state)
21                                  2'b00:data <= data;              //if were in s0,s1 data doesnt get
                                        updated
22                                  2'b01:data <= data;
23                                  2'b10:begin                                          //if were in s2 clock in
                                        the data into the shift register
24                                            q <= {q[10:0],sdata};
25                                            data <= data;                            //dont update data yet
26                                  end
27                                  2'b11:data <= q;                           //in s3 we can update data to be
                                        equal to the values in shift register
28                                  default:data <= data;
29                          endcase;
30                  end
31
32
33
34
35  endmodule
```

A.2 fifo_USB_driver code

A.2.1 fifo_to_usb_top.sv

```
1   module fifo_to_usb(  //top level module
2
3           output logic [7:0] data_out,     //8 bit data bus out to ft232h fifo-usb bridge
4
5           input logic txe,                 //active low transmit enable, when low data can be written to
                ft232h fifo
6
7           input logic clk_in,      //60Mhz clock signal from ft232h board
8
9           input logic reset_n,     //active low to reset the entire module
10
11          input logic enable,              //from adc_driver, send new packet into ft232h fifo on falling
                edge
12
13
14          input logic [11:0] ch_one_volt, //digital voltage values from adc_driver
15
16          input logic [11:0] ch_two_volt,
17
18          input logic a_v,//2 bit rotary encoder values
19          input logic b_v,//voltage
20          input logic a_t,//time
21          input logic b_t,
22          input logic a_tr,//trigger
23          input logic b_tr,
24          input logic switch,//channel select switch
25
26
27
28          output logic wr, // active low writte enable to the ft232h board
29                                                          //when signal is low ft232h reads in whatever
                                                                data is on the bus each 60mhz clock cycle
30
31          output logic siwu        //always high, sent out to ft232h board
32
33          );
34
35          logic [1:0] state;
36
37          logic [3:0] select;
38
39          logic [7:0] data_in;
40
41          ////
42
43          logic [7:0] ch_one_th;
44
45          logic [7:0] ch_one_h;
46
47          logic [7:0]      ch_one_t;
48
49          logic [7:0]      ch_one_o;
50
51          logic [7:0]      ch_two_th;
52
53          logic [7:0]      ch_two_h;
54
55          logic [7:0]      ch_two_t;
56
57          logic [7:0]      ch_two_o;
58
59          logic [7:0]      v_scale;
60
61          logic [7:0]      t_scale;
62
63          logic [7:0]      trigger_level;
64
65          logic [7:0]      switch_value;
66
67
68
69
70          fifo_driver fifo_driver(
71
72                  .state(state),
73
74                  .data_out(data_out),
75
76                  .data_in(data_in),
77
78                  .wr(wr),
79
```

```
80                      .siwu(siwu)
81
82
83
84                      );
85
86              fifo_fsm  fifo_fsm(
87
88                      .clk(clk_in),
89
90                      .txe(txe),
91
92                      .reset_n(reset_n),
93
94                      .enable(enable),
95
96                      .state(state),
97
98                      .select(select)
99
100                     );
101
102
103
104             selector  selector(
105
106                     .select(select),
107
108                     .ch_one_th(ch_one_th),
109
110                     .ch_one_h(ch_one_h),
111
112                     .ch_one_t(ch_one_t),
113
114                     .ch_one_o(ch_one_o),
115
116                     .ch_two_th(ch_two_th),
117
118                     .ch_two_h(ch_two_h),
119
120                     .ch_two_t(ch_two_t),
121
122                     .ch_two_o(ch_two_o),
123
124                     .v_scale(v_scale),
125
126                     .t_scale(t_scale),
127
128                     .trigger_level(trigger_level),
129
130                     .switch_value(switch_value),
131
132                     .data(data_in)
133
134             );
135
136             ascii_decoder ch_one(
137                     .num(ch_one_volt),
138
139                     .thousands(ch_one_th),
140
141                     .hundreds(ch_one_h),
142
143                     .tens(ch_one_t),
144
145                     .ones(ch_one_o)
146
147             );
148
149             ascii_decoder ch_two(
150
151                     .num(ch_two_volt),
152
153                     .thousands(ch_two_th),
154
155                     .hundreds(ch_two_h),
156
157                     .tens(ch_two_t),
158
159                     .ones(ch_two_o)
160
161             );
162
163             rotary_encoder_decoder volt_knob(
164
165             .a(a_v),
166
167             .b(b_v),
168
169             .rotary_value(v_scale)
170
171             );
172
173             rotary_encoder_decoder time_knob(
174
175             .a(a_t),
176
177             .b(b_t),
178
179             .rotary_value(t_scale)
180
181             );
182
183             rotary_encoder_decoder trigger_knob(
184
185             .a(a_tr),
186
187             .b(b_tr),
188
189             .rotary_value(trigger_level)
190
191             );
192
193              switch_ascii_decoder witch_ascii_decoder(
194
195                     .switch(switch),
```

```
196
197                 .switch_value(switch_value)
198
199         );
200
201 endmodule
```

A.2.2 ascii_decoder.sv

```
1   module ascii_decoder(
2           input logic [11:0] num,
3
4           output logic [7:0] thousands,
5
6           output logic [7:0] hundreds,
7
8           output logic [7:0] tens,
9
10          output logic [7:0] ones
11
12  );
13
14          always_comb
15                  begin           //splits 12bit number into it's 4 digits and adds 48 to get its ascii
                        value
16
17
18                          ones = (num % 10'd10) + 8'd48;
19                          tens = ( (num / 10'd10) % 10'd10 ) + 8'd48;
20                          hundreds = ( (num / 10'd100) % 10'd10 ) + 8'd48;
21                          thousands = ((num / 10'd1000) % 10'd10) + 8'd48;
22
23
24                  end
25
26  endmodule
```

A.2.3 fifo_fsm.sv

```
1   module fifo_fsm(
2           input logic clk,
3
4           input logic txe,
5
6           input logic reset_n,
7
8           input logic enable,
9
10          output logic [1:0] state,
11
12          output logic [3:0] select
13
14          );
15
16          parameter s0 = 2'd0;            //reset value at data_out bus
17          parameter s1 = 2'd1;            //send value at data_out bus to fifo on ft232h
18          parameter s2 = 2'd2;            //wait for falling edge of enable signal to send next packet
19
20          logic [1:0] next_state;
21          logic [3:0] next_select;
22
23          always_ff@(posedge clk, negedge reset_n)
24                  begin
25                          if(!reset_n)
26                                  begin   //during reset reset back to s0 and select = 0
27
28                                  state <= s0;
29                                  select <= 4'd0;
30
31                                  end
32                          else
33                                  begin                   //next_state and next_select always go to there
                                        next values
34
35                                  state <= next_state;
36                                  select <= next_select;
37
38                                  end
39                  end
40
41          always_ff@(*)
42                  begin
43
44                          case(state)
45
46                                  s0:begin
47
48
49                                          if(txe == 1'b0 )                //if txe is low we can move to
                                                next state and send
50                                                                                              //value
                                                                                                    at
                                                                                                    data_out

                                                                                                    bus
                                                                                                     to

                                                                                                    ft232h
51                                                  begin
52                                                          next_state <= s1;
53
54                                                  end
55
56                                          else
57                                                  begin
58                                                          next_state <= s0;       //if txe - 1 stay in
                                                                this state
59
60                                                  end
61
```

```
62                                              next_select <= select;  // select stays the same
63
64                                      end
65                                      s1:begin
66
67
68                                              if(select <= 4'd11)             //if select <= 11 then
                                                    incriment select and go back to s1
69                                                  begin
70                                                          next_select <= select + 4'd1;    //
                                                                increment select to get the next
                                                                charecter in the packet
71                                                          next_state <= s0;
72                                                  end
73                                              else
74                                                  begin
75                                                          next_select <= 4'd0;     //once all
                                                                charecters have been sent reset
                                                                select back to 0
76                                                          next_state <= s2;                //go to
                                                                s2 and wait for new sample to send
                                                                over
77                                                  end
78
79
80
81                                      end
82
83                                      s2:begin
84
85
86                                              next_select= 4'd0;              //select stays at 0
87
88                                              if(enable == 1'b0)                      //on falling edge of
                                                    enable go to s0 and send next packet
89                                                  begin
90                                                          next_state <= s0;
91
92                                                  end
93
94                                              else
95                                                  begin
96                                                          next_state <= s2;       //else stay in this
                                                                state
97
98                                                  end
99
100
101                                     end
102
103                                     default:begin
104                                             next_state <= s2;
105                                             next_select <= 4'd0;
106
107                                     end
108
109                             endcase
110
111             end
112
113 endmodule
```

A.2.4 fifo_driver.sv

```
1   module fifo_driver(
2
3           input logic [1:0] state,
4
5           input logic [7:0] data_in,
6
7           output logic [7:0] data_out,
8
9           output logic wr,
10
11          output logic siwu
12
13
14          );
15
16
17
18          parameter s0 = 2'd0;     //
19          parameter s1 = 2'd1;
20          parameter s2 = 2'd2;
21
22          always_ff@(*)
23                  begin
24
25                          siwu <= 1'd1;
26
27                          case(state)
28                                  s0:begin
29
30                                          wr <= 1'd1;      //fifo no longer reads data in
31                                          data_out <= data_in;     //update data_out with new data_in
32
33                                  end
34                                  s1:begin
35
36                                          wr <= 1'd0;                     //fifo can read in data from
                                                data_out bus
37                                          data_out <= data_out;           //data_out doesnt get updated
38
39
40                                  end
41
42                                  s2:begin
43
44                                          wr <= 1'd1;             //fifo cant read data at data_out bus
45                                          data_out <= data_in;
46
47
48                                  end
49                                  default:begin
```

```
50
51                                          wr <= 1'd1;                         //fifo cant read data at
                                                 data_out bus
52                                          data_out <= data_in;
53
54
55                                  end
56                          endcase;
57                  end
58
59  endmodule
```

A.2.5 selector.sv

```
1   module selector(
2
3           input logic [3:0] select,
4
5
6           input logic [7:0] ch_one_th,
7
8           input logic [7:0] ch_one_h,
9
10          input logic [7:0]       ch_one_t,
11
12          input logic [7:0]       ch_one_o,
13
14          input logic [7:0]       ch_two_th,
15
16          input logic [7:0]       ch_two_h,
17
18          input logic [7:0]       ch_two_t,
19
20          input logic [7:0]       ch_two_o,
21
22          input logic [7:0]       v_scale,
23
24          input logic [7:0]       t_scale,
25
26          input logic [7:0]       trigger_level,
27
28          input logic [7:0]       switch_value,
29
30          output logic [7:0] data
31
32          );
33
34          always_comb
35                  begin
36
37                          case(select)            //select which charecter in the packet to send out over
                                uart
38                                  4'd0:data = 8'd115;             //s
39                                  4'd1:data = ch_one_th;  //channel 1 thousands
40                                  4'd2:data = ch_one_h;
41                                  4'd3:data = ch_one_t;
42                                  4'd4:data = ch_one_o;   //channel 1 ones place
43                                  4'd5:data = ch_two_th;  //channel 2
44                                  4'd6:data = ch_two_h;
45                                  4'd7:data = ch_two_t;
46                                  4'd8:data = ch_two_o;
47                                  4'd9:data = v_scale;            //1 digit value of voltage rotary
                                        encoder
48                                  4'd10:data = t_scale;           //time rotary encoder
49                                  4'd11:data = trigger_level;     //trigger rotary encoder
50                                  4'd12:data = switch_value;              //value of switch
51                                  default:data = 8'd115;
52                          endcase;
53
54                  end
55
56
57
58
59  endmodule
```

A.2.6 rotary_encoder_decoder

```
1    module rotary_encoder_decoder(
2
3           input logic a,
4
5           input logic b,
6
7           //output logic [7:0] rotary_value               //ascii value of r encoder
8
9           output logic [11:0] rotary_value
10
11          );
12
13          logic [1:0] num;
14
15          always_comb             //combine the 2 bits of the rotary encoder into the first 2 bits of a 12
                bit number
16                                          //and convert it into its ascii vallue
17                  begin
18                          num = {a,b};
19
20                          rotary_value = num + 8'd48;             //add 48 to get its ascii value
21
22                  end
23
24  endmodule
```

A.2.7 switch_ascii_decoder.sv

```
1    module switch_ascii_decoder(
2
3           input logic switch,
```

```
4
5           output logic [7:0] switch_value
6
7           );
8
9
10          always_comb
11                  begin
12                          switch_value = switch + 8'd48;          //add 48 to get the ascii value of the
                                switches current state
13
14                  end
15
16  endmodule
```

B Qt Makefile

```
1   #############################################################
2   # Project: JD_oscilloscope
3   # File Name: datasource.h
4   # Modified  and Adapted by Tomas McMonigal
5   # Original files provided by The Qt Company Ltd. under the terms
6   # of the GNU Free Documentation License Version 1.3 published by the
7   # Free Software Foundation.
8   # Date Modified: 4/14/19
9   # Description: Qt Makefile
10  #############################################################
11
12  QT += charts qml quick serialport core
13
14  HEADERS += \
15       datasource.h \
16       readfifo.h
17
18  SOURCES += \
19       main.cpp \
20       datasource.cpp \
21       readfifo.cpp
22
23  RESOURCES += \
24       resources.qrc
25
26  DISTFILES += \
27       qml/qmloscilloscope/*
28
29  target.path = $$[QT_INSTALL_EXAMPLES]/charts/qmloscilloscope
30  INSTALLS += target
31
32  unix|win32: LIBS += -L/usr/include/local/lib -lftd2xx
```

C C++ files

C.0.1 datasource.h

```
1   /*****************************************
2    * Project: JD_oscilloscope
3    * File Name: datasource.h
4    * Modified and Adapted by Tomas McMonigal
5    * Original files provided by The Qt Company Ltd. under the terms
6    * of the GNU Free Documentation License Version 1.3 published by the
7    * Free Software Foundation.
8    * Date Modified: 5/31/19
9    * Description: Class definition file for DataSource
10   *****************************************/
11
12  #ifndef DATASOURCE_H
13  #define DATASOURCE_H
14
15  #include <QtCore/QObject>
16  #include <QtCharts/QAbstractSeries>
17  #include <QtSerialPort>
18  #include <QTextStream>
19  #include <QByteArray>
20  #include <QTime>
21  #include <QtCore/QIODevice>
22  #include <QtCore/QPointF>
23  #include <QtCore/QVector>
24  #include <QtCharts/QChartGlobal>
25  #include <ftdi/ftd2xx.h>
26  #include <QElapsedTimer>
27  #include <QQuickItem>
28  #include <QObject>
29  #include <iostream>
30  #include <QQmlEngine>
31  #include <QMutex>
32  #include <QMutexLocker>
33
34
35  QT_BEGIN_NAMESPACE
36  class QQuickView;
37  QT_END_NAMESPACE
38
39  QT_CHARTS_USE_NAMESPACE
40
41  class DataSource : public QObject
42  {
43      Q_OBJECT
44      QThread workerThread;
45  public:
46      explicit DataSource(QQuickView *appViewer, QObject *parent = 0);
47
48  Q_SIGNALS:
49
50  public slots:
51      void update(QAbstractSeries *series, int series_num);
52      void changeTimeScale(int time_scale);
53      void testData();
54      int readData_fifo();
```

```
55      int initialize_fifo();
56      void modifyVoltageScale(int value);
57      void modifyTimeScale(int value);
58      void changeTrigger(int value);
59      void updateTrigger(int index);
60      void set_channel_switch(int sw);
61
62  signals:
63      void signal_modifyTimeScale(QVariant value);
64      void signalTimeScale1_10000();
65      void signalTimeScale1_1000();
66      void signalTimeScale1_100();
67      void signalTimeScale1_10();
68      void signalTimeScale2_10000();
69      void signalTimeScale2_1000();
70      void signalTimeScale2_100();
71      void signalTimeScale2_10();
72      void signalVoltageScale1_1();
73      void signalVoltageScale1_2();
74      void signalVoltageScale1_3();
75      void signalVoltageScale1_4();
76      void signalVoltageScale1_5();
77      void signalVoltageScale2_1();
78      void signalVoltageScale2_2();
79      void signalVoltageScale2_3();
80      void signalVoltageScale2_4();
81      void signalVoltageScale2_5();
82      void signalIncreaseTrigger1();
83      void signalDecreaseTrigger1();
84      void signalChannelChanged1();
85      void signalChannelChanged2();
86  private:
87      int m_status_rotary3;
88      FT_HANDLE fthandle1;
89  //      QObject *object;
90
91      qint64 time1 = 0;
92
93      QQuickView *m_appViewer;
94      QList<QVector<QPointF> > m_data1;
95      QList<QVector<QPointF> > m_data2;
96      int m_index;
97      double sample_period;
98
99      // data structure members
100     QVector<QPointF> s1;
101     QVector<QPointF> s2;
102     qreal timeScale1;
103     qreal timeScale2;
104     int period_average = 1000;
105     qreal num_samples = 1;
106     QElapsedTimer elapsedTimer;
107     QVector<int> voltageRotary;
108     QVector<int> timeRotary;
109     int channel_switch;
110     int voltageScale1;
111     int voltageScale2;
112     qreal trigger1;
113     qreal trigger2;
114     int trigger1Set;
115     int trigger2Set;
116     qreal lastYValue1;
117     qreal lastYValue2;
118
119
120
121     // readfifo.h variables that are not repeated
122     qreal timeScale;
123     QVector<int> rotary1;
124     QVector<int> rotary2;
125     QVector<int> rotary3;
126
127 };
128
129 #endif // DATASOURCE_H
```

C.0.2 datasource.cpp

```
1   /*****************************************
2    * Project: JD_oscilloscope
3    * File Name: datasource.h
4    * Modified and Adapted by Tomas McMonigal
5    * Original files provided by The Qt Company Ltd. under the terms
6    * of the GNU Free Documentation License Version 1.3 published by the
7    * Free Software Foundation.
8    * Date Modified: 5/31/19
9    * Description: Reads data from FT232H Chip in FIFO mode, processes the data and stores it.
10   * This data is then passed on to ScopeView.qml to be plotted.
11   *****************************************/
12
13  #include "datasource.h"
14  #include <QtCharts/QXYSeries>
15  #include <QtCharts/QAreaSeries>
16  #include <QtQuick/QQuickView>
17  #include <QtQuick/QQuickItem>
18  #include <QtCore/QDebug>
19  #include <QtCore/QRandomGenerator>
20  #include <QtCore/QtMath>
21  #include <QFile>
22  #include <QString>
23  #include <QTextStream>
24  #include <QDebug>
25  #include <iostream>
26  #include <QIODevice>
27  #include <stdio.h>
28  #include <string.h>
29  #include <ftdi/ftd2xx.h>
30  #include <unistd.h>
31  #include <QElapsedTimer>
32  #include <iomanip>
33  #include <string.h>
34  #include <QThread>
35  #include <QChartView>
36  #include <QGraphicsView>
```

```
37  #include <QChart>
38  #include <QQmlEngine>
39  #include <QString>
40  #include <QQmlApplicationEngine>
41
42  #define MAX_DATA 10000
43  #define VCC 3.345
44  #define BUFFER_SIZE 130000
45  #define BUFF_SIZE 130000
46  #define MAX_PACKETS 10000
47
48  QT_CHARTS_USE_NAMESPACE
49
50  Q_DECLARE_METATYPE(QAbstractSeries *)
51  Q_DECLARE_METATYPE(QAbstractAxis *)
52
53
54  DataSource::DataSource(QQuickView *appViewer, QObject *parent) :
55      QObject(parent),
56      m_appViewer(appViewer),
57      m_index(-1),
58      m_status_rotary3(0)
59  {
60      qRegisterMetaType<QAbstractSeries*>();
61      qRegisterMetaType<QAbstractAxis*>();
62
63      //itializes time scale to 10000 us
64      timeScale1 = 10000;
65      timeScale2 = 10000;
66
67      //sets voltage scales to initial value in qml
68      voltageScale1 = 5;
69      voltageScale2 = 5;
70
71      //initializes triggers
72      trigger1 = 4;
73      trigger2 = 4;
74
75      channel_switch = 0;
76
77      trigger1Set = 1;
78      trigger2Set = 1;
79
80      lastYValue1 = 0;
81      lastYValue2 = 0;
82
83      // reserves memory for data structures
84      s1.reserve(MAX_DATA);
85      s2.reserve(MAX_DATA);
86
87
88      // ReadFifo contructor
89      elapsedTimer.start();
90      rotary1.reserve(4);
91      rotary2.reserve(4);
92      rotary3.reserve(4);
93      initialize_fifo();
94
95  }
96
97
98  void DataSource::updateTrigger(int index){
99      if (channel_switch == 0){
100         if (lastYValue1 < trigger1 && s1.value(index).y() > trigger1 && s1.value(index).y() < trigger1 +
                0.2){
101             trigger1Set = 0;
102         }
103         else{
104             trigger1Set = 1;
105         }
106     }
107     else if (channel_switch == 1){
108         if (lastYValue2 < trigger2 && s2.value(index).y() > trigger2 && s2.value(index).y() < trigger2 +
                0.2){
109             trigger2Set = 0;
110         }
111         else{
112             trigger2Set = 1;
113         }
114     }
115 }
116
117
118
119
120 void DataSource::modifyTimeScale(int value){
121 //        std::cout << "DataSource:: entering modifyTimeScale" << std::endl;
122     if (channel_switch == 0){
123         if (value == 1){
124             if (timeScale1 == 1000){
125                 emit signalTimeScale1_10000();
126                 timeScale1 = 10000;
127             }
128             else if (timeScale1 == 100){
129                 emit signalTimeScale1_1000();
130                 timeScale1 = 1000;
131             }
132             else if (timeScale1 == 10){
133                 emit signalTimeScale1_100();
134                 timeScale1 = 100;
135             }
136         }
137         else if (value == 0){
138             if (timeScale1 == 10000){
139                 emit signalTimeScale1_1000();
140                 timeScale1 = 1000;
141             }
142             else if (timeScale1 == 1000){
143                 emit signalTimeScale1_100();
144                 timeScale1 = 100;
145             }
146             else if (timeScale1 == 100){
147                 emit signalTimeScale1_10();
148                 timeScale1 = 10;
149             }
150         }
```

```
151         }
152         else if (channel_switch == 1){
153             if (value == 1){
154                 if (timeScale2 == 1000){
155                     emit signalTimeScale2_10000();
156                     timeScale2 = 10000;
157                 }
158                 else if (timeScale2 == 100){
159                     emit signalTimeScale2_1000();
160                     timeScale2 = 1000;
161                 }
162                 else if (timeScale2 == 10){
163                     emit signalTimeScale2_100();
164                     timeScale2 = 100;
165                 }
166             }
167             else if (value == 0){
168                 if (timeScale2 == 10000){
169                     emit signalTimeScale2_1000();
170                     timeScale2 = 1000;
171                 }
172                 else if (timeScale2 == 1000){
173                     emit signalTimeScale2_100();
174                     timeScale2 = 100;
175                 }
176                 else if (timeScale2 == 100){
177                     emit signalTimeScale2_10();
178                     timeScale2 = 10;
179                 }
180             }
181         }
182     }
183
184     void DataSource::changeTrigger(int value){
185         if (channel_switch == 0){
186             if (value == 1){
187                 trigger1 += 0.05;
188             }
189             else if (value == 0){
190                 trigger1 -= 0.05;
191             }
192             std::cout << "trigger channel 1 value = " << trigger1 << std::endl;
193         }
194         else if (channel_switch == 1){
195             if (value == 1){
196                 trigger2 += 0.05;
197             }
198             else if (value == 0){
199                 trigger2 -= 0.05;
200             }
201             std::cout << "trigger channel 2 value = " << trigger2 << std::endl;
202         }
203     }
204
205
206     void DataSource::modifyVoltageScale(int value){
207         std::cout << "DataSource: modifyVoltageScale" << std::endl;
208         // case when the switch is on channel 1
209         if (channel_switch == 0){
210             // case for increasing voltage scale
211             if (value == 1){
212                 if (voltageScale1 == 1){
213                     voltageScale1 = 2;
214                     emit signalVoltageScale1_2();
215                 }
216                 else if (voltageScale1 == 2){
217                     voltageScale1 = 3;
218                     emit signalVoltageScale1_3();
219                 }
220                 else if (voltageScale1 == 3){
221                     voltageScale1 = 4;
222                     emit signalVoltageScale1_4();
223                 }
224                 else if (voltageScale1 == 4){
225                     voltageScale1 = 5;
226                     emit signalVoltageScale1_5();
227                 }
228             }
229             // case for decreasing voltage scale
230             else if (value == 0){
231                 if (voltageScale1 == 5){
232                     voltageScale1 = 4;
233                     emit signalVoltageScale1_4();
234                 }
235                 else if (voltageScale1 == 4){
236                     voltageScale1 = 3;
237                     emit signalVoltageScale1_3();
238                 }
239                 else if (voltageScale1 == 3){
240                     voltageScale1 = 2;
241                     emit signalVoltageScale1_2();
242                 }
243                 else if (voltageScale1 == 2){
244                     voltageScale1 = 1;
245                     emit signalVoltageScale1_1();
246                 }
247             }
248         }
249         // case when the switch is on channel 2
250         else if (channel_switch == 1){
251             // case for increasing voltage scale
252             if (value == 1){
253                 if (voltageScale2 == 1){
254                     voltageScale2 = 2;
255                     emit signalVoltageScale2_2();
256                 }
257                 else if (voltageScale2 == 2){
258                     voltageScale2 = 3;
259                     emit signalVoltageScale2_3();
260                 }
261                 else if (voltageScale2 == 3){
262                     voltageScale2 = 4;
263                     emit signalVoltageScale2_4();
264                 }
265                 else if (voltageScale2 == 4){
266                     voltageScale2 = 5;
```

```
267                     emit signalVoltageScale2_5();
268                 }
269             }
270             // case for decreasing voltage scale
271             else if (value == 0){
272                 if (voltageScale2 == 5){
273                     voltageScale2 = 4;
274                     emit signalVoltageScale2_4();
275                 }
276                 else if (voltageScale2 == 4){
277                     voltageScale2 = 3;
278                     emit signalVoltageScale2_3();
279                 }
280                 else if (voltageScale2 == 3){
281                     voltageScale2 = 2;
282                     emit signalVoltageScale2_2();
283                 }
284                 else if (voltageScale2 == 2){
285                     voltageScale2 = 1;
286                     emit signalVoltageScale2_1();
287                 }
288             }
289         }
290     }
291
292
293     void DataSource::set_channel_switch(int sw){
294         if (sw == 0){
295             channel_switch = 0;
296             emit signalChannelChanged1();
297         }
298         else if (sw == 1){
299             channel_switch = 1;
300             emit signalChannelChanged2();
301         }
302     }
303
304
305     void DataSource::testData(){
306         for (int i = 0; i < MAX_DATA; i++){
307             QPointF series1_data(i, 3);
308             QPointF series2_data(i, 4);
309             s1.append(series1_data);
310             s2.append(series2_data);
311         }
312     }
313
314     void DataSource::changeTimeScale(int time_scale){
315         if (channel_switch == 0){
316             timeScale1 = time_scale;
317         }
318         else if (channel_switch == 1){
319             timeScale2 = time_scale;
320         }
321     }
322
323     void DataSource::update(QAbstractSeries *series, int series_num)
324     {
325         readData_fifo();
326         if (series_num == 1){
327             if (series) {
328                 int time_len = timeScale1;
329                 // next two lines define boundaries of window (x-axis)
330                 int minX = -time_len/2;
331                 int maxX = time_len/2;
332                 // points is local variable used to store elements in the window give by time_len
333                 QVector<QPointF> points;
334                 QXYSeries *xySeries = static_cast<QXYSeries *>(series);
335                 // finds the first value to fit window size (x-axis) of length time_len
336                 QPointF last_point = s1.value(s1.size() - 1);
337                 qreal last_x_value = last_point.x();
338                 qreal window_start = last_x_value - time_len;
339                 int first_element_index;
340                 for (int i = 0; i < s1.size(); i++){
341                     if (s1.value(i).x() >= window_start){
342                         first_element_index = i;
343                         break;
344                     }
345                 }
346                 QPointF first_element = s1.value(first_element_index);
347
348                 updateTrigger(first_element_index);
349
350                 // appends to points elements within window given with the new x-axis values for window to
                        start from minX
351                 qreal maxValue = 0;
352                 for (int i = first_element_index; i < s1.size(); i++){
353                     qreal x_axis = s1.value(i).x() - first_element.x() + minX;
354                     qreal y_axis = s1.value(i).y();
355                     if (y_axis > maxValue){
356                         maxValue = y_axis;
357                     }
358                     QPointF element(x_axis, y_axis);
359                     points.append(element);
360                 }
361                 if (!trigger1Set){
362                     xySeries->replace(points);
363                 }
364                 lastYValue1 = first_element.y();
365             }
366
367         }
368         else if (series_num == 2){
369
370             if (series) {
371                 int time_len = timeScale2;
372                 // next two lines define boundaries of window (x-axis)
373                 int minX = -time_len/2;
374                 int maxX = time_len/2;
375                 // points is local variable used to store elements in the window give by time_len
376                 QVector<QPointF> points;
377                 QXYSeries *xySeries = static_cast<QXYSeries *>(series);
378                 // finds the first value to fit window size (x-axis) of length time_len
379                 QPointF last_point = s2.value(s2.size() - 1);
380                 qreal last_x_value = last_point.x();
381                 qreal window_start = last_x_value - time_len;
```

```
382                 int first_element_index;
383                 for (int i = 0; i < s2.size(); i++){
384                     if (s2.value(i).x() >= window_start){
385                         first_element_index = i;
386                         break;
387                     }
388                 }
389                 QPointF first_element = s2.value(first_element_index);
390                 updateTrigger(first_element_index);
391
392                 // appends to points elements within window given with the new x-axis values for window to
                        start from minX
393                 for (int i = first_element_index; i < s2.size(); i++){
394                     qreal x_axis = s2.value(i).x() - s2.value(first_element_index).x() + minX;
395                     QPointF element(x_axis, s2.value(i).y());
396                     points.append(element);
397                 }
398                 // plots window
399                 if (!trigger2Set){
400                     xySeries->replace(points);
401                 }
402                 lastYValue2 = first_element.y();
403             }
404         }
405
406         else if (series_num == 4){
407
408             if (series){
409                 int min = -5000;
410                 int max = 5000;
411                 QVector<QPointF> points;
412                 QXYSeries *xySeries = static_cast<QXYSeries *>(series);
413                 if (channel_switch == 0){
414                     for (int i = min; i <= max; i++){
415                         qreal x_axis = i;
416                         QPointF element(x_axis, trigger1);
417                         points.append(element);
418                     }
419                     xySeries->replace(points);
420                 }
421                 else if (channel_switch == 1){
422                     for (int i = min; i <= max; i++){
423                         qreal x_axis = i;
424                         QPointF element(x_axis, trigger2);
425                         points.append(element);
426                     }
427                     xySeries->replace(points);
428                 }
429             }
430         }
431     }
432
433
434     int DataSource::initialize_fifo(){
435         std::cout << "initializing fifo" << std::endl;
436         FT_STATUS status;    //device status
437
438         //0 is index of device since we're only opening 1 device
439
440         status = FT_Open(0, &fthandle1);
441
442         if (status != FT_OK) {       //if device is not opened succesfully
443     //         qDebug() << "open device status is not ok " << status;
444             printf("initialize_fifo: open device status is not ok %d\n", status);
445             return 0;
446         }
447
448
449         //set read and write timeouts as 500ms
450         status = FT_SetTimeouts(fthandle1, 500, 500);
451
452         if (status != FT_OK) {       //if timeout are not setup succesfully
453     //         printf("timeout device status not Ok %d\n", status);
454             qDebug() << "timeout device status not ok " << status;
455         }
456
457         UCHAR MaskA = 0x00; // set data bus to inputs
458         UCHAR modeA = 0x40; //configure ft232h into synch fifo mode
459         //fifo mode must already have been programmed in eeprom
460
461         //set the chip mode
462         status = FT_SetBitMode(fthandle1, MaskA, modeA);
463
464         if (status != FT_OK)//if mode seelect was not succesfull
465             printf("mode A status not ok %d\n", status);
466
467         usleep(500); // sleep for 500 microseconds
468
469         elapsedTimer.restart();
470
471         // initializes rotary enconders
472         for (int i = 0; i < 4; i++){
473             rotary1.append(i);
474             rotary2.append(i);
475             rotary3.append(i);
476         }
477
478         return 0;
479
480     }
481
482
483
484     int DataSource::readData_fifo(){
485
486         // ***************************** INITIALIZATION OF VARIABLES
                ************************************
487         //             std::cout << "entering read data fifo" << std::endl;
488         DWORD data_read; //num bytes read from device
489         DWORD total_bytes_read = 0;
490         char buffer[BUFF_SIZE];
491         memset(buffer, '\0', sizeof(buffer));
492         FT_STATUS status;
493
494         // ****************** READING DATA ***********************
495         elapsedTimer.restart();
```

```
496         status = FT_Read(fthandle1, buffer, BUFFER_SIZE, &data_read);
497         sample_period = (elapsedTimer.nsecsElapsed())/(BUFFER_SIZE/13);
498         //        std::cout << "bytes read: " << RxBytes << std::endl;
499         // weighted average - same way TCP throughput is calculated
500         period_average = (period_average * num_samples * 0.875 + sample_period * 0.125 * num_samples)/(
                num_samples + 1);
501         num_samples++;
502         sample_period = period_average;
503         total_bytes_read = BUFFER_SIZE;
504                 std::cout << "sample period = " << sample_period << std::endl;
505
506         // ********** ERROR CHECKING ****************************
507         if (status != FT_OK) {
508             printf("status not OK %d\n", status);
509             return 0;
510         }
511
512         // ****************** FINDING PACKET START POINT *********************************
513         else {
514             qreal raw_channel1, raw_channel2, voltage_channel1, voltage_channel2;
515             int ch1_0, ch1_1, ch1_2, ch1_3;
516             int ch2_0, ch2_1, ch2_2, ch2_3;
517             int r1, r2, r3, sw;
518             int marker = 0;
519             unsigned int indexS;
520             for (int i = 0; i < 1000; i++){
521                 if (buffer[i] == 's'){
522                     indexS = i;
523                     marker = 1;
524                     break;
525                 }
526             }
527
528             // ************************* ERROR CHECKING *************************
529             if (marker == 0){
530                 std::cout << "ERROR: DATA CORRUPTED" << std::endl;
531                 return 0;
532             }
533
534             // ********************** PROCESSING DATA *************************
535             int points_total_each_channel = 0;
536             unsigned int i = indexS;
537             qreal counter = 0;
538             s1.clear();
539             s2.clear();
540             while (i < total_bytes_read - 13){ // - 13 because we must have at least one packet of data left
                    at the end
541                 if (buffer[i] == 's'){
542                     i++;
543                     ch1_0 = buffer[i++] - '0';
544                     ch1_1 = buffer[i++] - '0';
545                     ch1_2 = buffer[i++] - '0';
546                     ch1_3 = buffer[i++] - '0';
547                     ch2_0 = buffer[i++] - '0';
548                     ch2_1 = buffer[i++] - '0';
549                     ch2_2 = buffer[i++] - '0';
550                     ch2_3 = buffer[i++] - '0';
551                     r1 = buffer[i++] - '0';
552                     r2 = buffer[i++] - '0';
553                     r3 = buffer[i++] - '0';
554                     sw = buffer[i++] - '0';
555
556                     if (channel_switch != sw){
557                         channel_switch = sw;
558                         std::cout << "channel_switch: " << channel_switch << std::endl;
559                         set_channel_switch(sw);
560                     }
561
562                     if (r1 != rotary1.value(0)){
563                         //                          std::cout << "rotary1: " << r1 << std::endl;
564                         rotary1.pop_back();
565                         rotary1.push_front(r1);
566                         int last = rotary1.value(0) * 1000 + rotary1.value(1) * 100 + rotary1.value(2) * 10
                                + rotary1.value(3);
567                         if (last == 3201 || last == 2013 || last == 0132 || last == 1320){
568                             modifyTimeScale(1);
569                         }
570                         else if (last == 2310 || last == 3102 || last == 1023 || last == 0231){
571                             modifyTimeScale(0);
572                         }
573
574                     }
575                     if (r2 != rotary2.value(0)){
576                         //                          std::cout << "rotary2: " << r2 << std::endl;
577                         rotary2.pop_back();
578                         rotary2.push_front(r2);
579                         int last = rotary2.value(0) * 1000 + rotary2.value(1) * 100 + rotary2.value(2) * 10
                                + rotary2.value(3);
580                         if (last == 3201 || last == 2013 || last == 0132 || last == 1320){
581                             modifyVoltageScale(1);
582                         }
583                         else if (last == 2310 || last == 3102 || last == 1023 || last == 0231){
584                             modifyVoltageScale(0);
585                         }
586                     }
587                     if (r3 != rotary3.value(0)){
588                         //                          std::cout << "rotary 3: " << r3 << std::endl;
589                         rotary3.pop_back();
590                         rotary3.push_front(r3);
591                         int last = rotary3.value(0) * 1000 + rotary3.value(1) * 100 + rotary3.value(2) * 10
                                + rotary3.value(3);
592                         if (last == 3201 || last == 2013 || last == 0132 || last == 1320){
593                             changeTrigger(1);
594                         }
595                         else if (last == 2310 || last == 3102 || last == 1023 || last == 0231){
596                             changeTrigger(0);
597                         }
598
599                     }
600
601
602                     raw_channel1 = ch1_0 * 1000 + ch1_1 * 100 + ch1_2 * 10 + ch1_3;
603                     raw_channel2 = ch2_0 * 1000 + ch2_1 * 100 + ch2_2 * 10 + ch2_3;
604
605                     // converts raw adc value to voltage
606                     voltage_channel1 = 2 * (VCC/2 - (raw_channel1/4095)*VCC) + 0.025;
```

```
607                     voltage_channel2 = 2 * (VCC/2 - (raw_channel2/4095)*VCC) + 0.025;
608                     counter++;
609                     points_total_each_channel++;
610
611                     QPointF point_s1((counter - 1) * sample_period / 1000, voltage_channel1);
612                     QPointF point_s2((counter - 1) * sample_period / 1000, voltage_channel2);
613                     if (s1.size() < MAX_DATA){
614                         s1.append(point_s1);
615                         //                      std::cout << "channel 1: " << point_s1.y() << std::endl;
616                     }
617                     if (s2.size() < MAX_DATA){
618                         s2.append(point_s2);
619                         //                      std::cout << "channel 2: " << point_s2.y() << std::endl;
620
621                     }
622                 }
623                 else {
624                     std::cout << "data corrupted" << std::endl;
625                     return 0;
626                     i++;
627                 }
628             }
629             //close device
630             //         status = FT_Close(fthandle1);
631
632         }
633         return 0;
634     }
```

C.0.3 main.cpp

```
1   /*****************************************
2    * Project: JD_oscilloscope
3    * File Name: datasource.h
4    * Modified  and Adapted by Tomas McMonigal
5    * Original files provided by The Qt Company Ltd. under the terms
6    * of the GNU Free Documentation License Version 1.3 published by the
7    * Free Software Foundation.
8    * Date Modified: 5/31/19
9    * Description: Main function which creates objects and connections between c++,
10   * and qml files and executes the application.
11   *****************************************/
12
13
14  #include <QQmlContext>
15  #include <QQmlApplicationEngine>
16  #include <QQuickWindow>
17  #include <QtWidgets/QApplication>
18  #include <QtQml/QQmlContext>
19  #include <QtQuick/QQuickView>
20  #include <QtQml/QQmlEngine>
21  #include <QtCore/QDir>
22  #include <QFile>
23  #include <QString>
24  #include <QTextStream>
25  #include <QDebug>
26  #include <QSerialPort>
27  #include <QSerialPortInfo>
28  #include <QVector>
29  #include <QPointF>
30  #include <iostream>
31  #include <unistd.h>
32  #include "datasource.h"
33
34  int main(int argc, char *argv[])
35  {
36      // The QApplication class manages the GUI application flow control and main settings
37      // Qt Charts uses Qt Graphics View Framework for drawing, therefore, QApplication must be used
38      QApplication app(argc, argv);
39
40      // The QQuickView class provides a window for displaying a Qt Quick user interface
41      QQuickView viewer;
42
43      // The following are needed to make examples run without having to install the module
44      // in desktop environments.
45  #ifdef Q_OS_WIN
46      QString extraImportPath(QStringLiteral("%1/../../../../%2"));
47  #else
48      QString extraImportPath(QStringLiteral("%1/../../../%2"));
49  #endif
50
51      // QQmlEngine class provides an environment for instantiating QML components
52      // Prior to creating any QML components, an application must have created a QQmlEngine to gain
            access to a QML context
53      // This next two line adds a path directory where the engine searches for installed modules in a URL
            -based directory structure
54      viewer.engine()->addImportPath(extraImportPath.arg(QGuiApplication::applicationDirPath(),
55                                     QString::fromLatin1("qml")));
56
57      // The fuction Qobject::connect(const QObject* sender, const char* signal, const QObject* receiver,
            const char* method)
58      // creates a connection of the given type from the signal in the sender object to the method in the
            receiver object.
59      // Basically, it allows the program to end when the window is closed.
60      QObject::connect(viewer.engine(), &QQmlEngine::quit, &viewer, &QWindow::close);
61
62
63      // Sets the tittle for the graph
64      viewer.setTitle(QStringLiteral("JD Oscilloscope"));
65
66
67      // Creates an object of class DataSource which is created and defined in datasource.h and datasource
            .cpp
68      // the object viewer of class QQuickView is passed as an argument to the constructor of DataSource
            which
69      // links it to the QQuickView private member of the DataSource class for later use
70      DataSource* dataSource = new DataSource(&viewer);
71  //     QThread* thread = new QThread;
72  //     dataSource->moveToThread(thread);
73  //     QObject::connect(thread, SIGNAL (started()), dataSource, SLOT (readData_fifo()));
74
75       // allows type QVector<QPointF> to be passed on to slots in a signal
76       qRegisterMetaType<QVector<QPointF> >();
77       // allows type qreal to be passed on to slots in a signal
```

```
78          qRegisterMetaType<qreal>();
79
80
81
82      // The following line accesses the QQmlContext class member, contexts allow data to be exposed to
            the QML components instantiated by the QML engine.
83      // Each QQmlContext contains a set of properties, distinct from its QObject properties, that allow
            data to be explicitly bound to a context by name.
84      // The context properties are defined and updated by calling QQmlContext::setContextProperty().
85      viewer.rootContext()->setContextProperty("dataSource", dataSource);
86
87      // Sets the source to the url, loads the QML component and instantiates it
88      viewer.setSource(QUrl("qrc:/qml/qmloscilloscope/main.qml"));
89
90      //get root object from view
91  //      QObject *object = viewer.rootObject();
92
93      // Sets the root item to automatically resize the view to the size of the view.
94      viewer.setResizeMode(QQuickView::SizeRootObjectToView);
95
96      // sets the color of the window
97      viewer.setColor(QColor("#404040"));
98
99      // Allows the window to be displayed
100     viewer.show();
101
102
103     // Executes the app
104     return app.exec();
105 }
```

C.0.4 datasource.cpp

```
1   /*********************************************
2    * Project: JD_oscilloscope
3    * File Name: datasource.h
4    * Modified and Adapted by Tomas McMonigal
5    * Original files provided by The Qt Company Ltd. under the terms
6    * of the GNU Free Documentation License Version 1.3 published by the
7    * Free Software Foundation.
8    * Date Modified: 5/31/19
9    * Description: Reads data from FT232H Chip in FIFO mode, processes the data and stores it.
10   * This data is then passed on to ScopeView.qml to be plotted.
11   *********************************************/
12
13  #include "datasource.h"
14  #include <QtCharts/QXYSeries>
15  #include <QtCharts/QAreaSeries>
16  #include <QtQuick/QQuickView>
17  #include <QtQuick/QQuickItem>
18  #include <QtCore/QDebug>
19  #include <QtCore/QRandomGenerator>
20  #include <QtCore/QtMath>
21  #include <QFile>
22  #include <QString>
23  #include <QTextStream>
24  #include <QDebug>
25  #include <iostream>
26  #include <QIODevice>
27  #include <stdio.h>
28  #include <string.h>
29  #include <ftdi/ftd2xx.h>
30  #include <unistd.h>
31  #include <QElapsedTimer>
32  #include <iomanip>
33  #include <string.h>
34  #include <QThread>
35  #include <QChartView>
36  #include <QGraphicsView>
37  #include <QChart>
38  #include <QQmlEngine>
39  #include <QString>
40  #include <QQmlApplicationEngine>
41
42  #define MAX_DATA 10000
43  #define VCC 3.345
44  #define BUFFER_SIZE 130000
45  #define BUFF_SIZE 130000
46  #define MAX_PACKETS 10000
47
48  QT_CHARTS_USE_NAMESPACE
49
50  Q_DECLARE_METATYPE(QAbstractSeries *)
51  Q_DECLARE_METATYPE(QAbstractAxis *)
52
53
54  DataSource::DataSource(QQuickView *appViewer, QObject *parent) :
55      QObject(parent),
56      m_appViewer(appViewer),
57      m_index(-1),
58      m_status_rotary3(0)
59  {
60      qRegisterMetaType<QAbstractSeries*>();
61      qRegisterMetaType<QAbstractAxis*>();
62
63      //itializes time scale to 10000 us
64      timeScale1 = 10000;
65      timeScale2 = 10000;
66
67      //sets voltage scales to initial value in qml
68      voltageScale1 = 5;
69      voltageScale2 = 5;
70
71      //initializes triggers
72      trigger1 = 4;
73      trigger2 = 4;
74
75      channel_switch = 0;
76
77      trigger1Set = 1;
78      trigger2Set = 1;
79
80      lastYValue1 = 0;
81      lastYValue2 = 0;
```

```
82
83      // reserves memory for data structures
84      s1.reserve(MAX_DATA);
85      s2.reserve(MAX_DATA);
86
87
88      // ReadFifo contructor
89      elapsedTimer.start();
90      rotary1.reserve(4);
91      rotary2.reserve(4);
92      rotary3.reserve(4);
93      initialize_fifo();
94
95  }
96
97
98  void DataSource::updateTrigger(int index){
99      if (channel_switch == 0){
100         if (lastYValue1 < trigger1 && s1.value(index).y() > trigger1 && s1.value(index).y() < trigger1 +
              0.2){
101             trigger1Set = 0;
102         }
103         else{
104             trigger1Set = 1;
105         }
106     }
107     else if (channel_switch == 1){
108         if (lastYValue2 < trigger2 && s2.value(index).y() > trigger2 && s2.value(index).y() < trigger2 +
              0.2){
109             trigger2Set = 0;
110         }
111         else{
112             trigger2Set = 1;
113         }
114     }
115 }
116
117
118
119
120 void DataSource::modifyTimeScale(int value){
121 //         std::cout << "DataSource:: entering modifyTimeScale" << std::endl;
122     if (channel_switch == 0){
123         if (value == 1){
124             if (timeScale1 == 1000){
125                 emit signalTimeScale1_10000();
126                 timeScale1 = 10000;
127             }
128             else if (timeScale1 == 100){
129                 emit signalTimeScale1_1000();
130                 timeScale1 = 1000;
131             }
132             else if (timeScale1 == 10){
133                 emit signalTimeScale1_100();
134                 timeScale1 = 100;
135             }
136         }
137         else if (value == 0){
138             if (timeScale1 == 10000){
139                 emit signalTimeScale1_1000();
140                 timeScale1 = 1000;
141             }
142             else if (timeScale1 == 1000){
143                 emit signalTimeScale1_100();
144                 timeScale1 = 100;
145             }
146             else if (timeScale1 == 100){
147                 emit signalTimeScale1_10();
148                 timeScale1 = 10;
149             }
150         }
151     }
152     else if (channel_switch == 1){
153         if (value == 1){
154             if (timeScale2 == 1000){
155                 emit signalTimeScale2_10000();
156                 timeScale2 = 10000;
157             }
158             else if (timeScale2 == 100){
159                 emit signalTimeScale2_1000();
160                 timeScale2 = 1000;
161             }
162             else if (timeScale2 == 10){
163                 emit signalTimeScale2_100();
164                 timeScale2 = 100;
165             }
166         }
167         else if (value == 0){
168             if (timeScale2 == 10000){
169                 emit signalTimeScale2_1000();
170                 timeScale2 = 1000;
171             }
172             else if (timeScale2 == 1000){
173                 emit signalTimeScale2_100();
174                 timeScale2 = 100;
175             }
176             else if (timeScale2 == 100){
177                 emit signalTimeScale2_10();
178                 timeScale2 = 10;
179             }
180         }
181     }
182 }
183
184 void DataSource::changeTrigger(int value){
185     if (channel_switch == 0){
186         if (value == 1){
187             trigger1 += 0.05;
188         }
189         else if (value == 0){
190             trigger1 -= 0.05;
191         }
192         std::cout << "trigger channel 1 value = " << trigger1 << std::endl;
193     }
194     else if (channel_switch == 1){
195         if (value == 1){
```

```
196                 trigger2 += 0.05;
197             }
198             else if (value == 0){
199                 trigger2 -= 0.05;
200             }
201             std::cout << "trigger channel 2 value = " << trigger2 << std::endl;
202         }
203     }
204
205
206     void DataSource::modifyVoltageScale(int value){
207         std::cout << "DataSource: modifyVoltageScale" << std::endl;
208         // case when the switch is on channel 1
209         if (channel_switch == 0){
210             // case for increasing voltage scale
211             if (value == 1){
212                 if (voltageScale1 == 1){
213                     voltageScale1 = 2;
214                     emit signalVoltageScale1_2();
215                 }
216                 else if (voltageScale1 == 2){
217                     voltageScale1 = 3;
218                     emit signalVoltageScale1_3();
219                 }
220                 else if (voltageScale1 == 3){
221                     voltageScale1 = 4;
222                     emit signalVoltageScale1_4();
223                 }
224                 else if (voltageScale1 == 4){
225                     voltageScale1 = 5;
226                     emit signalVoltageScale1_5();
227                 }
228             }
229             // case for decreasing voltage scale
230             else if (value == 0){
231                 if (voltageScale1 == 5){
232                     voltageScale1 = 4;
233                     emit signalVoltageScale1_4();
234                 }
235                 else if (voltageScale1 == 4){
236                     voltageScale1 = 3;
237                     emit signalVoltageScale1_3();
238                 }
239                 else if (voltageScale1 == 3){
240                     voltageScale1 = 2;
241                     emit signalVoltageScale1_2();
242                 }
243                 else if (voltageScale1 == 2){
244                     voltageScale1 = 1;
245                     emit signalVoltageScale1_1();
246                 }
247             }
248         }
249         // case when the switch is on channel 2
250         else if (channel_switch == 1){
251             // case for increasing voltage scale
252             if (value == 1){
253                 if (voltageScale2 == 1){
254                     voltageScale2 = 2;
255                     emit signalVoltageScale2_2();
256                 }
257                 else if (voltageScale2 == 2){
258                     voltageScale2 = 3;
259                     emit signalVoltageScale2_3();
260                 }
261                 else if (voltageScale2 == 3){
262                     voltageScale2 = 4;
263                     emit signalVoltageScale2_4();
264                 }
265                 else if (voltageScale2 == 4){
266                     voltageScale2 = 5;
267                     emit signalVoltageScale2_5();
268                 }
269             }
270             // case for decreasing voltage scale
271             else if (value == 0){
272                 if (voltageScale2 == 5){
273                     voltageScale2 = 4;
274                     emit signalVoltageScale2_4();
275                 }
276                 else if (voltageScale2 == 4){
277                     voltageScale2 = 3;
278                     emit signalVoltageScale2_3();
279                 }
280                 else if (voltageScale2 == 3){
281                     voltageScale2 = 2;
282                     emit signalVoltageScale2_2();
283                 }
284                 else if (voltageScale2 == 2){
285                     voltageScale2 = 1;
286                     emit signalVoltageScale2_1();
287                 }
288             }
289         }
290     }
291
292
293     void DataSource::set_channel_switch(int sw){
294         if (sw == 0){
295             channel_switch = 0;
296             emit signalChannelChanged1();
297         }
298         else if (sw == 1){
299             channel_switch = 1;
300             emit signalChannelChanged2();
301         }
302     }
303
304
305     void DataSource::testData(){
306         for (int i = 0; i < MAX_DATA; i++){
307             QPointF series1_data(i, 3);
308             QPointF series2_data(i, 4);
309             s1.append(series1_data);
310             s2.append(series2_data);
311         }
```

```
312 }
313
314 void DataSource::changeTimeScale(int time_scale){
315     if (channel_switch == 0){
316         timeScale1 = time_scale;
317     }
318     else if (channel_switch == 1){
319         timeScale2 = time_scale;
320     }
321 }
322
323 void DataSource::update(QAbstractSeries *series, int series_num)
324 {
325     readData_fifo();
326     if (series_num == 1){
327         if (series) {
328             int time_len = timeScale1;
329             // next two lines define boundaries of window (x-axis)
330             int minX = -time_len/2;
331             int maxX = time_len/2;
332             // points is local variable used to store elements in the window give by time_len
333             QVector<QPointF> points;
334             QXYSeries *xySeries = static_cast<QXYSeries *>(series);
335             // finds the first value to fit window size (x-axis) of length time_len
336             QPointF last_point = s1.value(s1.size() - 1);
337             qreal last_x_value = last_point.x();
338             qreal window_start = last_x_value - time_len;
339             int first_element_index;
340             for (int i = 0; i < s1.size(); i++){
341                 if (s1.value(i).x() >= window_start){
342                     first_element_index = i;
343                     break;
344                 }
345             }
346             QPointF first_element = s1.value(first_element_index);
347
348             updateTrigger(first_element_index);
349
350             // appends to points elements within window given with the new x-axis values for window to
                    start from minX
351             qreal maxValue = 0;
352             for (int i = first_element_index; i < s1.size(); i++){
353                 qreal x_axis = s1.value(i).x() - first_element.x() + minX;
354                 qreal y_axis = s1.value(i).y();
355                 if (y_axis > maxValue){
356                     maxValue = y_axis;
357                 }
358                 QPointF element(x_axis, y_axis);
359                 points.append(element);
360             }
361             if (!trigger1Set){
362                 xySeries->replace(points);
363             }
364             lastYValue1 = first_element.y();
365         }
366
367     }
368     else if (series_num == 2){
369
370         if (series) {
371             int time_len = timeScale2;
372             // next two lines define boundaries of window (x-axis)
373             int minX = -time_len/2;
374             int maxX = time_len/2;
375             // points is local variable used to store elements in the window give by time_len
376             QVector<QPointF> points;
377             QXYSeries *xySeries = static_cast<QXYSeries *>(series);
378             // finds the first value to fit window size (x-axis) of length time_len
379             QPointF last_point = s2.value(s2.size() - 1);
380             qreal last_x_value = last_point.x();
381             qreal window_start = last_x_value - time_len;
382             int first_element_index;
383             for (int i = 0; i < s2.size(); i++){
384                 if (s2.value(i).x() >= window_start){
385                     first_element_index = i;
386                     break;
387                 }
388             }
389             QPointF first_element = s2.value(first_element_index);
390             updateTrigger(first_element_index);
391
392             // appends to points elements within window given with the new x-axis values for window to
                    start from minX
393             for (int i = first_element_index; i < s2.size(); i++){
394                 qreal x_axis = s2.value(i).x() - s2.value(first_element_index).x() + minX;
395                 QPointF element(x_axis, s2.value(i).y());
396                 points.append(element);
397             }
398             // plots window
399             if (!trigger2Set){
400                 xySeries->replace(points);
401             }
402             lastYValue2 = first_element.y();
403         }
404     }
405
406     else if (series_num == 4){
407
408         if (series){
409             int min = -5000;
410             int max = 5000;
411             QVector<QPointF> points;
412             QXYSeries *xySeries = static_cast<QXYSeries *>(series);
413             if (channel_switch == 0){
414                 for (int i = min; i <= max; i++){
415                     qreal x_axis = i;
416                     QPointF element(x_axis, trigger1);
417                     points.append(element);
418                 }
419                 xySeries->replace(points);
420             }
421             else if (channel_switch == 1){
422                 for (int i = min; i <= max; i++){
423                     qreal x_axis = i;
424                     QPointF element(x_axis, trigger2);
425                     points.append(element);
```

```
426                     }
427                     xySeries->replace(points);
428                 }
429             }
430         }
431     }
432
433
434     int DataSource::initialize_fifo(){
435         std::cout << "initializing fifo" << std::endl;
436         FT_STATUS status;    //device status
437
438         //0 is index of device since we're only opening 1 device
439
440         status = FT_Open(0, &fthandle1);
441
442         if (status != FT_OK) {       //if device is not opened succesfully
443     //         qDebug() << "open device status is not ok " << status;
444             printf("initialize_fifo: open device status is not ok %d\n", status);
445             return 0;
446         }
447
448
449         //set read and write timeouts as 500ms
450         status = FT_SetTimeouts(fthandle1, 500, 500);
451
452         if (status != FT_OK) {       //if timeout are not setup succesfully
453     //         printf("timeout device status not Ok %d\n", status);
454             qDebug() << "timeout device status not ok " << status;
455         }
456
457         UCHAR MaskA = 0x00; // set data bus to inputs
458         UCHAR modeA = 0x40; //configure ft232h into synch fifo mode
459         //fifo mode must already have been programmed in eeprom
460
461         //set the chip mode
462         status = FT_SetBitMode(fthandle1, MaskA, modeA);
463
464         if (status != FT_OK)//if mode seelect was not succesfull
465             printf("mode A status not ok %d\n", status);
466
467         usleep(500); // sleep for 500 microseconds
468
469         elapsedTimer.restart();
470
471         // initializes rotary enconders
472         for (int i = 0; i < 4; i++){
473             rotary1.append(i);
474             rotary2.append(i);
475             rotary3.append(i);
476         }
477
478         return 0;
479
480     }
481
482
483
484     int DataSource::readData_fifo(){
485
486         // ****************************** INITIALIZATION OF VARIABLES
                 ***************************************
487         //             std::cout << "entering read data fifo" << std::endl;
488         DWORD data_read; //num bytes read from device
489         DWORD total_bytes_read = 0;
490         char buffer[BUFF_SIZE];
491         memset(buffer, '\0', sizeof(buffer));
492         FT_STATUS status;
493
494         // ******************* READING DATA ***********************
495         elapsedTimer.restart();
496         status = FT_Read(fthandle1, buffer, BUFFER_SIZE, &data_read);
497         sample_period = (elapsedTimer.nsecsElapsed())/(BUFFER_SIZE/13);
498         //         std::cout << "bytes read: " << RxBytes << std::endl;
499         // weighted average - same way TCP throughput is calculated
500         period_average = (period_average * num_samples * 0.875 + sample_period * 0.125 * num_samples)/(
                num_samples + 1);
501         num_samples++;
502         sample_period = period_average;
503         total_bytes_read = BUFFER_SIZE;
504                 std::cout << "sample period = " << sample_period << std::endl;
505
506         // ********** ERROR CHECKING ****************************
507         if (status != FT_OK) {
508             printf("status not OK %d\n", status);
509             return 0;
510         }
511
512         // ****************** FINDING PACKET START POINT *********************************
513         else {
514             qreal raw_channel1, raw_channel2, voltage_channel1, voltage_channel2;
515             int ch1_0, ch1_1, ch1_2, ch1_3;
516             int ch2_0, ch2_1, ch2_2, ch2_3;
517             int r1, r2, r3, sw;
518             int marker = 0;
519             unsigned int indexS;
520             for (int i = 0; i < 1000; i++){
521                 if (buffer[i] == 's'){
522                     indexS = i;
523                     marker = 1;
524                     break;
525                 }
526             }
527
528             // ************************* ERROR CHECKING *************************
529             if (marker == 0){
530                 std::cout << "ERROR: DATA CORRUPTED" << std::endl;
531                 return 0;
532             }
533
534             // ********************** PROCESSING DATA *************************
535             int points_total_each_channel = 0;
536             unsigned int i = indexS;
537             qreal counter = 0;
538             s1.clear();
539             s2.clear();
```

```
540             while (i < total_bytes_read - 13){ // - 13 because we must have at least one packet of data left
                    at the end
541                 if (buffer[i] == 's'){
542                     i++;
543                     ch1_0 = buffer[i++] - '0';
544                     ch1_1 = buffer[i++] - '0';
545                     ch1_2 = buffer[i++] - '0';
546                     ch1_3 = buffer[i++] - '0';
547                     ch2_0 = buffer[i++] - '0';
548                     ch2_1 = buffer[i++] - '0';
549                     ch2_2 = buffer[i++] - '0';
550                     ch2_3 = buffer[i++] - '0';
551                     r1 = buffer[i++] - '0';
552                     r2 = buffer[i++] - '0';
553                     r3 = buffer[i++] - '0';
554                     sw = buffer[i++] - '0';
555
556                     if (channel_switch != sw){
557                         channel_switch = sw;
558                         std::cout << "channel_switch: " << channel_switch << std::endl;
559                         set_channel_switch(sw);
560                     }
561
562                     if (r1 != rotary1.value(0)){
563                         //                          std::cout << "rotary1: " << r1 << std::endl;
564                         rotary1.pop_back();
565                         rotary1.push_front(r1);
566                         int last = rotary1.value(0) * 1000 + rotary1.value(1) * 100 + rotary1.value(2) * 10
                                + rotary1.value(3);
567                         if (last == 3201 || last == 2013 || last == 0132 || last == 1320){
568                             modifyTimeScale(1);
569                         }
570                         else if (last == 2310 || last == 3102 || last == 1023 || last == 0231){
571                             modifyTimeScale(0);
572                         }
573
574                     }
575                     if (r2 != rotary2.value(0)){
576                         //                          std::cout << "rotary2: " << r2 << std::endl;
577                         rotary2.pop_back();
578                         rotary2.push_front(r2);
579                         int last = rotary2.value(0) * 1000 + rotary2.value(1) * 100 + rotary2.value(2) * 10
                                + rotary2.value(3);
580                         if (last == 3201 || last == 2013 || last == 0132 || last == 1320){
581                             modifyVoltageScale(1);
582                         }
583                         else if (last == 2310 || last == 3102 || last == 1023 || last == 0231){
584                             modifyVoltageScale(0);
585                         }
586                     }
587                     if (r3 != rotary3.value(0)){
588                         //                          std::cout << "rotary 3: " << r3 << std::endl;
589                         rotary3.pop_back();
590                         rotary3.push_front(r3);
591                         int last = rotary3.value(0) * 1000 + rotary3.value(1) * 100 + rotary3.value(2) * 10
                                + rotary3.value(3);
592                         if (last == 3201 || last == 2013 || last == 0132 || last == 1320){
593                             changeTrigger(1);
594                         }
595                         else if (last == 2310 || last == 3102 || last == 1023 || last == 0231){
596                             changeTrigger(0);
597                         }
598
599                     }
600
601
602                     raw_channel1 = ch1_0 * 1000 + ch1_1 * 100 + ch1_2 * 10 + ch1_3;
603                     raw_channel2 = ch2_0 * 1000 + ch2_1 * 100 + ch2_2 * 10 + ch2_3;
604
605                     // converts raw adc value to voltage
606                     voltage_channel1 = 2 * (VCC/2 - (raw_channel1/4095)*VCC) + 0.025;
607                     voltage_channel2 = 2 * (VCC/2 - (raw_channel2/4095)*VCC) + 0.025;
608                     counter++;
609                     points_total_each_channel++;
610
611                     QPointF point_s1((counter - 1) * sample_period / 1000, voltage_channel1);
612                     QPointF point_s2((counter - 1) * sample_period / 1000, voltage_channel2);
613                     if (s1.size() < MAX_DATA){
614                         s1.append(point_s1);
615                         //                          std::cout << "channel 1: " << point_s1.y() << std::endl;
616                     }
617                     if (s2.size() < MAX_DATA){
618                         s2.append(point_s2);
619                         //                          std::cout << "channel 2: " << point_s2.y() << std::endl;
620
621                     }
622                 }
623                 else {
624                     std::cout << "data corrupted" << std::endl;
625                     return 0;
626                     i++;
627                 }
628             }
629             //close device
630             //        status = FT_Close(fthandle1);
631
632         }
633         return 0;
634     }
```

D qml files

D.0.1 main.qml

```
1   /****************************************
2    * Project: JD_oscilloscope
3    * File Name: datasource.h
4    * Modified  and Adapted by Tomas McMonigal
5    * Original files provided by The Qt Company Ltd. under the terms
6    * of the GNU Free Documentation License Version 1.3 published by the
7    * Free Software Foundation.
```

```
 8   * Date Modified: 5/31/19
 9   * Description: Main qml file - creates the connections between signals
10   * from other qml files and to the C++ files.
11   ****************************************/
12
13  import QtQuick 2.0
14
15  //![1]
16  Item {
17  //    signal callTimeScaleChange(int value)
18
19      Connections {
20          target: dataSource
21
22          onSignalTimeScale1_10000: {
23              scopeView.axisX().max = 5000;
24              scopeView.axisX().min = -5000;
25          }
26          onSignalTimeScale1_1000 : {
27              scopeView.axisX().max = 500;
28              scopeView.axisX().min = -500;
29          }
30          onSignalTimeScale1_100 : {
31              scopeView.axisX().max = 50;
32              scopeView.axisX().min = -50;
33          }
34          onSignalTimeScale1_10 : {
35              scopeView.axisX().max = 5;
36              scopeView.axisX().min = -5;
37          }
38
39          onSignalTimeScale2_10000: {
40              scopeView.changeAxisX2(5000);
41  //              scopeView.axisX2().min = - 5000;
42          }
43          onSignalTimeScale2_1000: {
44              scopeView.changeAxisX2(500);
45  //              scopeView.axisX2().max = 500;
46  //              scopeView.axisX2().min = - 500;
47          }
48          onSignalTimeScale2_100: {
49              scopeView.changeAxisX2(50);
50  //              scopeView.axisX2().max = 50;
51  //              scopeView.axisX2().min = - 50;
52          }
53          onSignalTimeScale2_10: {
54              scopeView.changeAxisX2(5);
55  //              scopeView.axisX2().max = 5;
56  //              scopeView.axisX2().min = - 5;
57          }
58
59          onSignalVoltageScale1_1: scopeView.changeVoltageScale1(1);
60          onSignalVoltageScale1_2: scopeView.changeVoltageScale1(2);
61          onSignalVoltageScale1_3: scopeView.changeVoltageScale1(3);
62          onSignalVoltageScale1_4: scopeView.changeVoltageScale1(4);
63          onSignalVoltageScale1_5: scopeView.changeVoltageScale1(5);
64
65          onSignalVoltageScale2_1: scopeView.changeVoltageScale2(1);
66          onSignalVoltageScale2_2: scopeView.changeVoltageScale2(2);
67          onSignalVoltageScale2_3: scopeView.changeVoltageScale2(3);
68          onSignalVoltageScale2_4: scopeView.changeVoltageScale2(4);
69          onSignalVoltageScale2_5: scopeView.changeVoltageScale2(5);
70      }
71
72
73      function modifyTimeScale(value){
74          console.log("main.qml: modifyTimeScale")
75          scopeView.axisX.max = 50;
76          scopeView.axisX.min = -50;
77          dataSource.changeTimeScale(value);
78  //          callTimeScaleChange(value);
79          return "main.qml: changed time scale"
80      }
81      function myQmlFunction(msg) {
82             console.log("Got message:", msg)
83             return "some return value"
84        }
85
86      id: main
87      width: 600
88      height: 400
89
90      ControlPanel {
91          id: controlPanel
92          anchors.top: parent.top
93          anchors.topMargin: 10
94          anchors.bottom: parent.bottom
95          anchors.left: parent.left
96          anchors.leftMargin: 10
97  //![1]
98
99          function changeTimeScale(newValue){
100             dataSource.changeTimeScale(newValue);
101             scopeView.axisX().max = newValue/2;
102             scopeView.axisX().min = -newValue/2;
103         }
104
105         onSignalTimeScaleChanged: {
106             console.log("onSignalTimeScale");
107             console.log(scopeView.axisX().max);
108             dataSource.changeTimeScale(sampleCount);
109             scopeView.axisX().max = sampleCount/2;
110             scopeView.axisX().min = -sampleCount/2;
111         }
112
113
114         onSeriesTypeChanged: scopeView.changeSeriesType(type);
115         onRefreshRateChanged: {
116             scopeView.changeRefreshRate(rate);
117 //              scopeView.changeVoltageScale(rate);
118          }
119         onVoltageScaleChanged1: {
120             scopeView.changeVoltageScale1(newBoundary);
121             dataSource.setVoltageScale1(newBoundary);
122         }
123
```

```
124             onVoltageScaleChanged2: {
125                 scopeView.changeVoltageScale2(newBoundary);
126                 dataSource.setVoltageScale2(newBoundary);
127             }
128
129             onTriggerButtonChanged: scopeView.triggerVisible(enabled)
130             onTriggerButtonChanged2: scopeView.triggerVisible2(enabled)
131
132             // onAntialiasingEnabled name needs to be changed to onSingal1Enabled
133             onAntialiasingEnabled: scopeView.signal1Visible(enabled);
134                 //scopeView.antialiasing = enabled;
135             // onOpenGlChanged needs to be changed to onSignal2Enabled
136             onOpenGlChanged: {
137                 scopeView.signal2Visible(enabled);
138                 //scopeView.openGL = enabled;
139             }
140             onTimeAxisChanged:
141 //                console.log("changing time scale");
142                 scopeView.axisX().max = sampleCount;
143
144 //                scopeView.xAxisChanged(xAxisRange);
145         }
146
147 //![2]
148         ScopeView {
149             id: scopeView
150             anchors.top: parent.top
151             anchors.bottom: parent.bottom
152             anchors.right: parent.right
153             anchors.left: controlPanel.right
154             height: main.height
155             property string property0: "none.none"
156
157
158             onOpenGLSupportedChanged: {
159                 if (!openGLSupported) {
160                     controlPanel.openGLButton.enabled = false
161                     controlPanel.openGLButton.currentSelection = 0
162                 }
163             }
164         }
165 //![2]
166
167 }
```

D.0.2 ScopeView.qml

```
1   /*****************************************
2    * Project: JD_oscilloscope
3    * File Name: datasource.h
4    * Modified  and Adapted by Tomas McMonigal
5    * Original files provided by The Qt Company Ltd. under the terms
6    * of the GNU Free Documentation License Version 1.3 published by the
7    * Free Software Foundation.
8    * Date Modified: 5/31/19
9    * Description: qml file that controls the functionality of the elements of the graph.
10   *****************************************/
11
12  import QtQuick 2.0
13  import QtCharts 2.1
14
15  //![1]
16  ChartView {
17      signal signalTimeScaleChanged2(int sampleCount)
18
19      id: chartView
20      animationOptions: ChartView.NoAnimation
21      theme: ChartView.ChartThemeDark
22      property bool openGL: true
23      property bool openGLSupported: true
24      property int xAxisScaleValue: 1000
25
26      onOpenGLChanged: {
27          if (openGLSupported) {
28              series("Channel 1").useOpenGL = openGL;
29              series("Channel 2").useOpenGL = openGL;
30          }
31      }
32      Component.onCompleted: {
33          if (!series("Channel 1").useOpenGL) {
34              openGLSupported = false
35              openGL = false
36          }
37 //           if (!series("Channel 2").useOpenGl){
38 //               openGLSupported = false
39 //               openGL = false
40 //           }
41      }
42
43      ValueAxis {
44          id: axisY1
45          min: -5
46          max: 5
47          tickCount: 11
48          titleText: "Voltage Channel 1"
49      }
50
51      ValueAxis {
52          id: axisY2
53          min: -5
54          max: 5
55          tickCount: 11
56          titleText: "Voltage Channel 2"
57      }
58
59      ValueAxis {
60          id: axisX
61          min: -5000
62          max: 5000
63          tickCount: 11
64          titleText: "Time Channel 1 (us)"
65 //           labelsVisible: false
66      }
67      ValueAxis {
```

```
68            id: axisX2
69            min: -5000
70            max: 5000
71            tickCount: 11
72            titleText: "Time Channel 2 (us)"
73        }
74
75        LineSeries {
76            id: lineSeries1
77            name: "Channel 1"
78            axisX: axisX
79            axisY: axisY1
80            useOpenGL: chartView.openGL
81        }
82        LineSeries {
83            id: lineSeries2
84            name: "Channel 2"
85            axisXTop: axisX2
86            axisYRight: axisY2
87            useOpenGL: chartView.openGL
88        }
89        LineSeries {
90            id: trigger
91            name: "Trigger"
92            axisX: axisX
93            axisY: axisY1
94            useOpenGL: chartView.openGL
95        }
96  //      LineSeries {
97  //          id: trigger2
98  //          name: "Trigger Channel 2"
99  //          axisXTop: axisX2
100 //          axisYRight: axisY2
101 //          useOpenGL: chartView.openGL
102 //      }
103
104 //![1]
105
106     //![2]
107     Timer {
108         id: refreshTimer
109         interval: 1// / 60 * 1000 // 60 Hz
110         running: true
111         repeat: true
112         onTriggered: {
113 //              console.log("timer triggered");
114 //              dataSource.readData_fifo();
115 //              dataSource.read_data();
116             dataSource.update(chartView.series(0), 1);
117             dataSource.update(chartView.series(1), 2);
118             dataSource.update(chartView.series(2), 4);
119 //              dataSource.update(chartView.series(3), 4);
120 //              updateTimeAxis();
121         }
122     }
123
124     //![2]
125
126     //![3]
127     function changeSeriesType(type) {
128         chartView.removeAllSeries();
129
130         // Create two new series of the correct type. Axis x is the same for both of the series,
131         // but the series have their own y-axes to make it possible to control the y-offset
132         // of the "signal sources".
133         if (type === "line") {
134             var series1 = chartView.createSeries(ChartView.SeriesTypeLine, "Channel 1",
135                                                  axisX, axisY1);
136             series1.useOpenGL = chartView.openGL
137
138             var series2 = chartView.createSeries(ChartView.SeriesTypeLine, "Channel 2",
139                                                  axisX2, axisY2);
140             series2.useOpenGL = chartView.openGL
141
142             var series3 = chartView.createSeries(chartView.SeriesTypeLine, "Trigger",
143                                                  axisX, axisY1);
144             series3.useOpenGL = charView.openGl
145
146
147         } else {
148             var series1 = chartView.createSeries(ChartView.SeriesTypeScatter, "Channel 1",
149                                                  axisX, axisY1);
150             series1.markerSize = 2;
151             series1.borderColor = "transparent";
152             series1.useOpenGL = chartView.openGL
153
154             var series2 = chartView.createSeries(ChartView.SeriesTypeScatter, "Channel 2",
155                                                  axisX2, axisY2);
156             series2.markerSize = 2;
157             series2.borderColor = "transparent";
158             series2.useOpenGL = chartView.openGL
159
160             var series3 = chartView.createSeries(chartView.SeriesTypeLine, "Trigger",
161                                                  axisX, axisY1);
162             series3.useOpenGL = charView.openGl
163         }
164     }
165
166     function createAxis(min, max) {
167         // The following creates a ValueAxis object that can be then set as a x or y axis for a series
168         return Qt.createQmlObject("import QtQuick 2.0; import QtCharts 2.0; ValueAxis { min: "
169                                   + min + "; max: " + max + " }", chartView);
170     }
171     //![3]
172
173     function setAnimations(enabled) {
174         if (enabled)
175             chartView.animationOptions = ChartView.SeriesAnimations;
176         else
177             chartView.animationOptions = ChartView.NoAnimation;
178     }
179
180     function changeRefreshRate(rate) {
181         refreshTimer.interval = 1 / Number(rate) * 1000;
182     }
183     function changeVoltageScale1(newBoundary) {
```

```
//          console.log("changing voltage scale 1");
            axisY1.min = -newBoundary;
            axisY1.max = newBoundary;
        }
        function changeVoltageScale2(newBoundary){
            axisY2.min = -newBoundary;
            axisY2.max = newBoundary;
        }
        function changeAxisX2(value){
            axisX2.max = value;
            axisX2.min = -value;
        }

        function signal1Visible(enabled){
            if (enabled)
                series("Channel 1").visible = false;
            else
                series("Channel 1").visible = true;
        }

        function triggerVisible(enabled){
            if (enabled)
                series("Trigger").visible = false;
            else
                series("Trigger").visible = true;
        }

        function triggerVisible2(enabled){
            if (enabled)
                series("Trigger Channel 2").visible = false;
            else
                series("Trigger Channel 2").visible = true;
        }

        function signal2Visible(enabled){
            if (enabled)
                series("Channel 2").visible = false;
            else
                series("Channel 2").visible = true;
       }

}
```

D.0.3 ControlPanel.qml

```
/*****************************************
 * Project: JD_oscilloscope
 * File Name: datasource.h
 * Modified  and Adapted by Tomas McMonigal
 * Original files provided by The Qt Company Ltd. under the terms
 * of the GNU Free Documentation License Version 1.3 published by the
 * Free Software Foundation.
 * Date Modified: 5/31/19
 * Description: Control Pannel for all the Buttons on the GUI
 * and the timer.
 *****************************************/

import QtQuick 2.1
import QtQuick.Layouts 1.0

ColumnLayout {
    property alias signal2Button: signal2Button
    property alias signal1Button: signal1Button
    spacing: 8
    Layout.fillHeight: true
    signal animationsEnabled(bool enabled)
    signal seriesTypeChanged(string type)
    signal refreshRateChanged(variant rate)
    signal signalTimeScaleChanged(int sampleCount)
    signal voltageScaleChanged1(int newBoundary)
    signal voltageScaleChanged2(int newBoundary)
    signal antialiasingEnabled(bool enabled)
    signal openGlChanged(bool enabled)
    signal timeAxisChanged(int xAxisRange);
    signal triggerButtonChanged(bool enabled)
    signal triggerButtonChanged2(bool enabled)

    Text {
        text: "Controls"
        font.pointSize: 18
        color: "white"
    }

    MultiButton {
        id: signal1Button
        text: "Channel 1: "
        items: ["on", "off"]
        enabled: true
        currentSelection: 0
        // antialiasingEnabled needs to be changed to signal1Enabled
        onSelectionChanged: antialiasingEnabled(currentSelection == 1);
    }



    MultiButton {
        id: signal2Button
        text: "Channel 2: "
        items: ["on", "off"]
        currentSelection: 0
        // openGlChanged needs to be changed to signal2Enabled
        onSelectionChanged: openGlChanged(currentSelection == 1);
    }

    MultiButton {
        id: triggerButton
        text: "Trigger: "
        items: ["on", "off"]
        currentSelection: 0
        onSelectionChanged: triggerButtonChanged(currentSelection == 1)
    }

//    MultiButton {
//        id: triggerButton2
```

```
//          text: "Trigger: "
//          items: ["on", "off"]
//          currentSelection: 0
//          onSelectionChanged: triggerButtonChanged2(currentSelection == 1)
//      }


        MultiButton {
            text: "Graph: "
            items: ["line", "scatter"]
            currentSelection: 0
            onSelectionChanged: seriesTypeChanged(items[currentSelection]);
        }

        MultiButton {
            id: sampleCountButton
            text: "Time Elapsed (ms): "
            items: ["10000", "1000", "100", "10"]
            currentSelection: 0
            // onSelectionChanged: timeAxisChanged(items[currentSelection]);

            onSelectionChanged: signalTimeScaleChanged(selection);
        }

        MultiButton {
            text: "Refresh rate(Hz): "
            items: ["1", "24", "60"]
            currentSelection: 2
            onSelectionChanged: refreshRateChanged(items[currentSelection]);
        }

        MultiButton {
            id: voltageScaleButton1
            text: "Voltage Signal 1: "
            items: ["5", "4", "3", "2", "1"]
            currentSelection: 0
            onSelectionChanged: voltageScaleChanged1(
                                    voltageScaleButton1.items[voltageScaleButton1.currentSelection],
                                    5,
                                    selection);
        }

        MultiButton {
            id: voltageScaleButton2
            text: "Voltage Signal 2: "
            items: ["5", "4", "3", "2", "1"]
            currentSelection: 0
            onSelectionChanged: voltageScaleChanged2(
                                    voltageScaleButton2.items[voltageScaleButton2.currentSelection],
                                    5,
                                    selection);
        }


}
```

D.0.4 MultiButton.qml

```
/*****************************************
 * Project: JD_oscilloscope
 * File Name: datasource.h
 * Modified  and Adapted by Tomas McMonigal
 * Original files provided by The Qt Company Ltd. under the terms
 * of the GNU Free Documentation License Version 1.3 published by the
 * Free Software Foundation.
 * Date Modified: 5/31/19
 * Description: qml file that describes the functionality of each botton on the GUI
 *****************************************/

import QtQuick 2.0
import QtQuick.Controls 1.0
import QtQuick.Controls.Styles 1.0

Item {
    id: button

    property string text: "Option: "
    property variant items: ["first"]
    property int currentSelection: 0
    signal selectionChanged(variant selection)

    signal clicked

    implicitWidth: buttonText.implicitWidth + 5
    implicitHeight: buttonText.implicitHeight + 10

    Button {
        id: buttonText
        width: parent.width
        height: parent.height

        style: ButtonStyle {
            label: Component {
                Text {
                    text: button.text + button.items[currentSelection]
                    clip: true
                    wrapMode: Text.WordWrap
                    verticalAlignment: Text.AlignVCenter
                    horizontalAlignment: Text.AlignHCenter
                    anchors.fill: parent
                }
            }
        }
        onClicked: {
            currentSelection = (currentSelection + 1) % items.length;
            selectionChanged(button.items[currentSelection]);
        }
    }
}
```