CMPE 121L: Microprocessor System Design Lab
Spring 2017
Lab Project: Oscilloscope and Logic Analyzer

In this final project, you will use the PSoC-5 microcontroller and the Raspberry Pi 3 to design two separate applications: (i) a dual-channel oscilloscope, and (ii) an 8-channel logic analyzer. You will use the PSoC to sample the signals to be monitored, and the Raspberry Pi to analyze and visualize them. You can use either the UART or SPI interfaces on the boards to send commands from the Raspberry Pi to the PSoC and stream data samples from the PSoC to the Raspberry Pi.

Oscilloscope

The oscilloscope application consists of two parts: The part running on the Raspberry Pi is the master application, which is responsible for configuring the oscilloscope parameters, collecting samples from the PSoC, and displaying the data graphically. On starting up, the program first goes through a configuration phase in which the user can define the operating parameters for the oscilloscope application. The following are the commands the user may enter, to define the operating environment. The program should check for any illegal values entered and provide help to the user to correct the input.

```
set nchannels <num>
```

`<num>` specifies the number of channels desired and can be **1** or **2**.

```
set mode <mode>
```

`<mode>` can be either **`free_run`** or **`trigger.`** In the free_run mode, the scope displays a free-running signal that may not be stable. The trigger mode aligns the sampled signal using a trigger level and trigger slope, so that repeating waveforms will appear stable on the screen.

```
set trigger_level <level>
```

`<level>` specifies the trigger level in millivolts and can range from 0V to 5V in steps of 0.1V.

```
set trigger_slope <slope>
```

`<slope>` can be either positive or negative. Positive slope indicates that the time sweep of the waveform should start on a rising edge when the signal crosses the trigger level. Negative slope indicates that the time sweep of the waveform should start on a falling edge when the signal crosses the trigger level.

```
set trigger_channel <num>
```
`<num>` is the channel (0 or 1) that is the source of the trigger.

```
set yscale <ydiv>
```

`<ydiv>` defines the vertical scale of the waveform display in volts per division. Its value can be 0.5, 1, 1.5 or 2.

```
set xscale <xdiv>
```

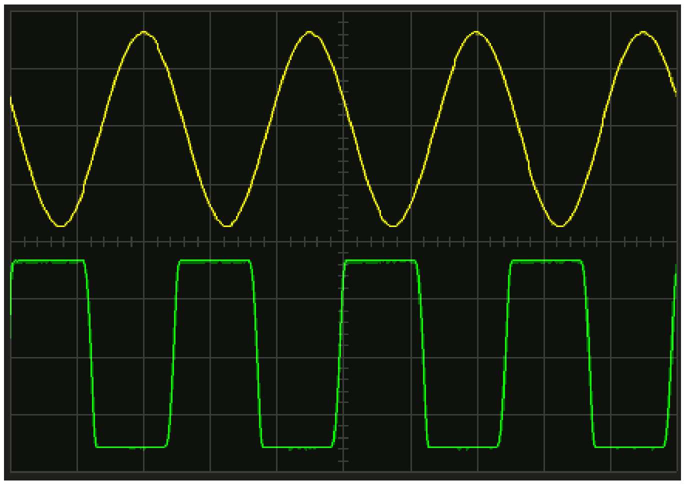
`<xdiv>` defines the horizontal scale of the waveform display in microseconds. Its value can be 1, 10, 100, 500, 1000, 2000, 5000 or 10000.

You may add other parameters specific to your design. The user ends this configuration phase by entering the command

```
start
```

which causes the Raspberry Pi program to send a command to the PSoC to start sampling the input channels, receive the sampled data, and display them continuously on the Raspberry Pi screen.

The waveform display should be similar to a standard oscilloscope screen, showing the data waveforms against a reference grid with the parameter settings (horizontal and vertical scales, trigger parameters, etc.) displayed in text. Here is an example for the dual-channel case (shown without any text).



The text should include labels for the horizontal and vertical axes, the various parameter settings, and the sampling frequency (this is the sampling rate of your analog front-end in the PSoC).

To keep the graphics programming to a minimum, the application does not need to accept any input from the user once it exits the configuration phase. However, when displaying

data from two channels, the user should be able to shift the channel-2 wave up and down using a potentiometer connected to the PSoC (in the above picture, shift the green wave up and down keeping the yellow wave steady).
Here are some of the key specifications you should design to:

1. **Maximum sampling rate**: The sampling rate determines the highest frequency that you can support for your input signals. There is no required minimum sampling rate, but your design should try to achieve the highest sampling rates possible, given the constraints of your hardware. That means you should try to achieve the highest sampling rate possible for your analog front-end. The communication link between the PSoC and the Raspberry Pi should also be able to stream data at this rate.
2. **Sampling resolution**: The sampling resolution should be 8 bits per sample.
3. **Input voltage**: The range of the input voltage is 0 to +5V.

Your PSoC design needs to sample the data from one or two channels and stream it to the Raspberry Pi. While there are many ways to design the PSoC part, you might want to consider a DMA-based approach with ping-pong buffers to allow the sampling to proceed concurrently with data streaming.

Finally, you need use a graphics library to design the waveform display on the Raspberry Pi. I recommend the **OpenVG** library for this purpose. It has a simple API to draw various 2D shapes, which is adequate for our purpose. Information on installing and using this tool will be posted separately.

You can test your oscilloscope with periodic waveforms (sine, square, triangle). Change the frequency of the pattern while the oscilloscope program is running in the trigger mode, and the waveform should change on the display, while remaining stable.

Logic Analyzer

A logic analyzer is used to debug digital systems by monitoring signals during normal operation of the system and capturing them around points of interest. For example, you can look for a state machine reaching a specific state and record its input and output signals starting from several cycles before the event, continuing for many cycles after the event, and displaying the resulting waveforms. Although their displays may look similar to that of an oscilloscope, they have a key difference in that they usually deal with only digital signals. They also have many more channels than a typical oscilloscope, allowing a large number of signals (including wide buses) to be captured simultaneously (Watch this YouTube video for a tutorial on logic analyzers).

The design of the logic analyzer application is similar to the oscilloscope, with the following key differences.

1. There are 8 channels to be sampled simultaneously, but these are digital channels. Their binary states can be captured by a register (with a clock frequency as high as your design allows). This gives you one byte per clock cycle, which can then be transferred to memory using DMA. Thus, the design requires no analog components.

2. The trigger condition can be specified using a logical expression of the states of the 8 channels (for example, when the eight inputs are 1000 0101).
3. The trigger event in the logic analyzer is a one-time event. Typically, the user sets up the trigger condition and starts the application. The application then monitors the signals, waiting for the trigger event to occur, while also capturing the signals. When the trigger event is detected, it will continue to capture the signals for a designated time. It will then stop monitoring the signals and display the waveforms of the sampled signals around the trigger event.

Just like the oscilloscope application, the logic analyzer application consists of two parts: The part running on the Raspberry Pi is the master application, which is responsible for configuring the logic analyzer parameters, collecting samples from the PSoC, and displaying the data graphically. On starting up, the program first goes through a configuration phase in which the user can define the operating parameters for the application. The following are the commands the user may enter, to define the operating environment. The program should check for any illegal values entered and provide help to the user to correct the input.

```
set nchannels <num>
```

`<num>` specifies the number of channels desired and can be any number from 1 through 8. The channels will be designated as ch1 through ch8.

```
set trigger_cond <logic expression>
```

This defined the enabling condition for the trigger event. `<logic expression>` can be any two-level logic expression involving AND(&), OR (|) and NOT(~) operations on the enabled channels. Some examples for logic expressions are

ch0

~ch0 & ch1

(ch0 & ch1) | (~ch2 & ch3) | (ch4 & ~ch5 & ch6 & ~ch7)

```
set trigger_dir <direction>
```

`<direction>` can be either **positive** or **negative**. Positive direction specifies that the trigger should become active when the trigger condition changes from false to true, and negative direction specifies that the trigger should become active when the condition changes from true to false.

```
set mem_depth <sample_count>
```

This command defines the size of the sample window. For example, if mem_depth is 1000, the analyzer will maintain a window of the last 1000 samples in memory. When triggered, it will continue to collect samples for the next 500 cycles, and will then display a waveform containing 500 samples before the trigger point and 500 cycles after the trigger point. Your design must support a memory depth of at least 5000 samples.

```
set sample_freq <frequency>
```

This command sets the sampling frequency in kHz (frequency of the clock used by the PSoC to sample the input channels. Its value can range from 1 to the maximum clock frequency that can be supported by your design.

```
set xscale <xdiv>
```

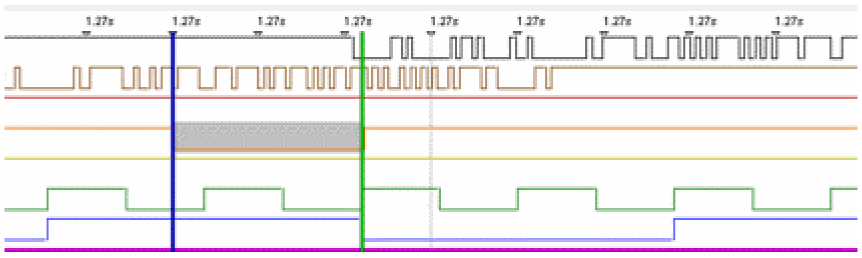
`<xdiv>` defines the horizontal scale of the waveform display in microseconds. Its value can be 1, 10, 100, 500, 1000, 2000, 5000 or 10000.

You may add commands to set other parameters specific to your design. The user ends this configuration phase by entering the command

```
run
```

which causes the Raspberry Pi program to send a command to the PSoC to start sampling the input channels, looking for the trigger condition. The check for the trigger condition may be performed either in the PSoC or in the Raspberry Pi. In the former case, the PSoC needs to send the captured data samples over to the Raspberry Pi only after the trigger event has occurred, whereas in the latter, it needs to stream the data to the Raspberry Pi as soon as it starts sampling.

When the trigger event has occurred, the Raspberry Pi must display all the channel waveforms on the screen. An example format is shown below.



Only one of the vertical bars is needed, which identifies the trigger point. Use negative time values to identify the part of the waveforms to the left of the trigger point and positive time values for the part to the right of the trigger point. Use text labels to identify the channels and display them in different colors.

The waveform display should be similar to a standard oscilloscope screen, showing the data waveforms against a reference grid with the parameter settings (horizontal and vertical scales, trigger parameters, etc.) displayed in text. Here is an example for the dual-channel case (shown without any text).

When displaying data from the channels, the user should be able to slide the waveform window to the left or right by turning the potentiometer (that is, it will display the leftmost part of the sampling window when the potentiometer is all the way to the left, and the rightmost part of the window when it is all the way to the right).