CMPE 121L: Microprocessor System Design Lab

Spring 2019

Lab Project: Dual-Channel Oscilloscope

In this final project, you will use the PSoC-5 microcontroller and the Raspberry Pi to design a dual-channel oscilloscope. You will use the PSoC to sample the signals to be monitored, and the Raspberry Pi to analyze and visualize them. You will primarily use the USB OTG 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, but will also use the I2C bus to implement additional control channels.

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. The slave application running on the PSoC samples analog data from the two channels, converts them to digital, and transmits the data to the Raspberry Pi. The slave application also implements the user interface with two potentiometers, which can be used to scroll the waveforms up and down.

The program on the Raspberry Pi is called with the following command-line arguments. Any argument could be omitted, in which case its default value will be used. The arguments could be in any order.

```
myscope -m <mode> -t <trigger-level> -s <trigger-slope> -r <sample-rate>
-c <trigger-channel> -x <xscale> -y <yscale>
```

The program should check for any illegal values entered and exit when the input is invalid, with a message to help the user correct the input. The various arguments are defined below.

`-m <mode>`

`<mode>` can be either **free** 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. By default, the mode is set to **trigger**.

`-t <trigger-level>`

`<trigger-level>` specifies the trigger level in millivolts and can range from 0 to 5000 in steps of 100. By default, the trigger level is set to 2500 (that is, 2.5V).

`-s <trigger-slope>`

<trigger-slope> can be either **pos** or **neg**. Positive slope (**pos**) indicates that the time sweep of the waveform should start on a rising edge when the signal crosses the trigger level. Negative slope (**neg**) indicates that the time sweep of the waveform should start on a falling edge when the signal crosses the trigger level. By default, the trigger-level is set to **pos**.

`-r <sample-rate>`

This parameter defines the sampling rate for the ADCs of the two channels, in kilosamples per second. The sampling rate can be 1, 10, 20, 50 and 100 ksamples per second.

`-c <trigger-channel>`

<trigger-channel> is the channel (1 or 2) that is the source of the trigger. By default, channel 1 is selected as the trigger channel.

`-x <xscale>`

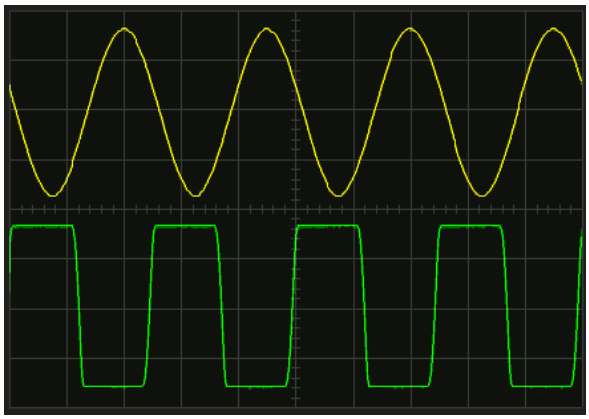
<xscale> defines the horizontal scale of the waveform display in microseconds. Its value can be 100, 500, 1000, 2000, 5000, 10000, 50000 or 100000. Its default value is 1000 (1 millisecond per division).

`-y <yscale>`

<yscale> defines the vertical scale of the waveform display in millivolts per division. Its value can be 100, 500, 1000, 2000 or 2500. Its default value is 1000 (1V per division).

On starting up, the Raspberry Pi program sends commands to the PSoC to start sampling the input channels, receives the sampled data, and displays 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).

The Raspberry Pi application does not need to accept any keyboard input from the user while it is running. However, you should connect two potentiometers to the PSoC for scrolling the waveforms. The user should be able to shift the waveform for each channel up or down by turning the potentiometers.

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. The SAR ADCs in the PSoC have a maximum sampling rate of 1 Megasamples/second. Your design should support sampling rates up to 100 ksamples/second, but you should try to estimate the highest sampling rates possible, given the constraints of your hardware. A key consideration is that the communication link between the PSoC and the Raspberry Pi should be able to stream data at the rate generated by the ADCs.
2. **Sampling resolution**: The sampling resolution is always 8 bits per sample.
3. **Input voltage**: The range of the input voltage for each channel is 0 to +5V.

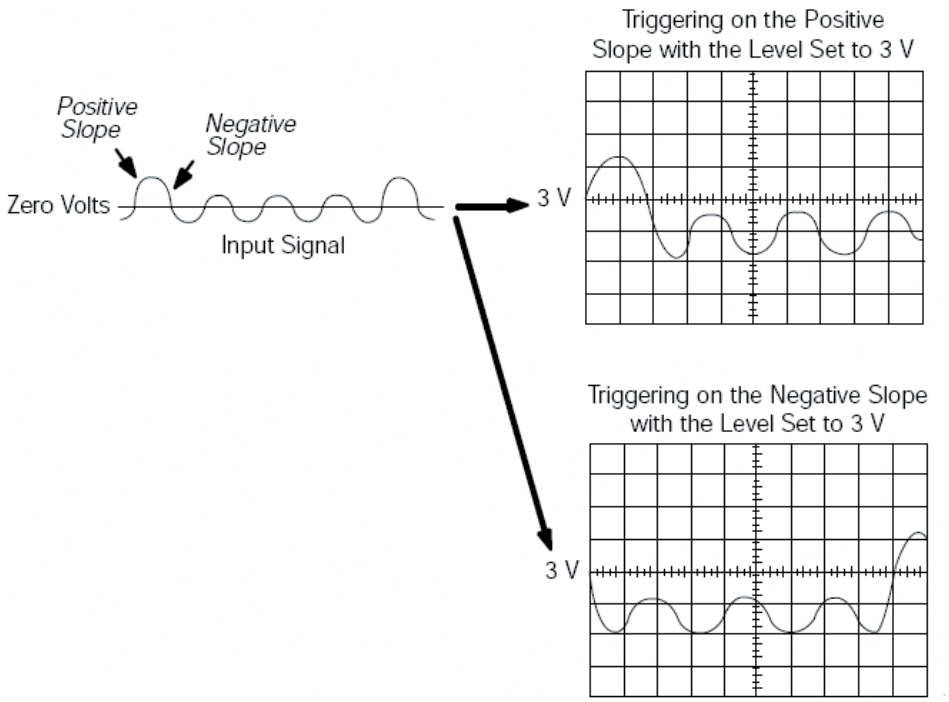


Illustration of how the trigger level and slope affect the waveform display (source: Tektronix).

While many of the internal details of your design are left for you to work on, here are some guidelines that will get you started:

1. **Analog front-end:** You should use separate SAR ADCs (which have the highest sampling rates) to sample the analog data from the two channels. Set the resolution to 8 bits, input range to $V_{ssa} - V_{dda}$ (single-ended), and sample mode to free-running. You can connect the DMA request input to a free-running clock and vary its frequency to control the sampling rate.
2. **Moving data from the ADCs:** You should use a separate DMA channel to move data from each ADC into ping-pong buffers, similar to the design in Lab 4/5.
3. **Streaming data to the Raspberry Pi:** You may use USB bulk transfers to stream the sampled data to the Raspberry Pi. You can configure the bulk transfers by defining two separate Endpoints in the USBFS configuration (in this case both transfers are IN transfers).
4. **Interfacing the potentiometers**: You need to use a third ADC to sample the settings of the two potentiometers. The Sigma Delta ADC (which is slower) is adequate for this. You can share the same ADC across the two potentiometers using an Analog Mux component (You should use the *Analog Mux* component, not the *Analog Hardware Mux*. The former is software-controlled, while the latter is hardware-controlled). You should use an I2C link between the PSoC and the Raspberry Pi to send the potentiometer data to the oscilloscope application. This can be done by including an I2C slave component in your PSoC design and using the I2C interface of the Raspberry Pi read the potentiometer settings through the I2C bus periodically

(approximately every 100 milliseconds). Example code and documentation to configure and use the I2C link are posted on Canvas.

5. **Coordination between the PSoC and the Raspberry Pi:** An additional communication channel is needed for the oscilloscope application running on the Raspberry Pi to send commands to the PSoC. At a minimum, this command channel should allow the Raspberry Pi application to start and stop the data sampling, set the sampling rate, etc.. This can be done through the same I2C link used to read the potentiometer settings.
6. **Detection of Trigger Condition:** Let a_n denote the amplitude of the nth sample of the signal. For rising edge trigger, the trigger conditions consist of
   (i) The previous sample a_{n-1} < trigger level.
   (ii) The current sample $a_n \geq$ trigger level.
   (iii) The current slope of the signal is positive.

   For falling edge trigger, the trigger conditions consist of
   (iv) The previous sample a_{n-1} > trigger level.
   (v) The current sample $a_n \leq$ trigger level.
   (vi) The current slope of the signal is negative.

   The samples may need some smoothing to detect positive and negative slopes of the signal reliably. False reading of slope from high-frequency glitches can be prevented by passing the signal through a low-pass filter. This can be done by maintaining a running average of the signal samples. For example, if a_n denotes the amplitude of the nth sample, we can compute a moving average

   $$b_n = (a_{n-4} + 2a_{n-3} + 3a_{n-2} + 2a_{n-1} + a_n)/9.$$

   The slope can then be determined by comparing b_n with b_{n-1} (note that the division by 9 is not required as we are only interested in the sign of the result).

   The smoothing part is not required for the project, but it is recommended.

Finally, you need use a graphics library to design the waveform display on the Raspberry Pi. You will use 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, with example code, has been posted separately.

You can use the AD-2 pattern generator or another PSoC board to generate various periodic waveforms and test your oscilloscope with them (sine, square, ramp). Example tester code for the PSoC will be posted. If the frequency of the test pattern is changed while the oscilloscope program is running in the trigger mode, the waveform should change on the display, while remaining stable.

Code Structure

You should break your design functionally into multiple C files. For example, you may want to organize your Raspberry Pi code for the oscilloscope as follows:

- Header file: scope.h
- main control loop: main.c
- command argument parser: cmdargs.c
- USB communication tasks: usbcomm.c
- graphical display: graphics.c
- data processing and trigger detection: data.c

You should then set up a makefile to compile the files and generate the executable (an example makefile has been posted).

Maintaining Files on GitLab Server

You must use the SOE GitLab server to regularly back up your Raspberry Pi and PSoC source files. During checkoff, you will be asked to pull the latest version of the code from the server and compile it. We will also use the version of the code in your GitLab repository at checkoff time for grading.

Checkoff Schedule

The schedule for checkoff and submission of reports is posted on Canvas. We will provide more details on the checkoff tests and report organization later.