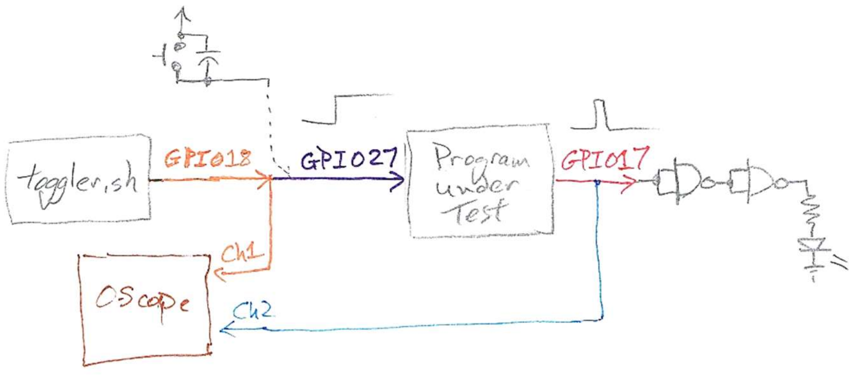
ECE 785: Project 2 (v0.8)
I/O Response Time Analysis

Overview

You will evaluate various programs (including modified versions of some in the **exploringrpi** repository) to measure how quickly the RPi4 can respond to a changing digital input by changing a digital output.

Timing Analysis



You will use the toggler.sh shell script to generate events by toggling the GPIO output pin 18 between 1 and 0, simulating switch presses. You may want to raise the toggling frequency to gather data faster.

Connect GPIO output pin 18 to GPIO input pin 27. You will use several different detector programs to detect rising edges on GPIO input pin 27 and respond by setting the event detector GPIO output pin 17 and then immediately clearing it (creating a short pulse).

Measure the response time between the input event (rising edge on pins 18 and 27) and the output (rising edge on pin 17) using the Analog Discovery 2 (and the program Waveforms) as an oscilloscope, with Channel 1 connected to the input and Channel 2 on the output.

Detector Programs

You will create the different detector programs based on the starter code listed below.

- Compile all code with maximum optimization (-O3) where possible. This may involve modifying a make file or build script.
- Modify the starter programs to generate to a brief output pulse (0 -> 1 -> 0) when the input event is detected. In some cases, you will need to widen the pulse with extra code so the oscilloscope reliably catches and displays it.

Detector Version	**Starter Code**	**Lang.**	**GPIO Peripheral Access**	**Threads**	**Output Control**	**Blocking Behavior**
1	**ERP** Listing 6-2, 6-3	C++	Indirect, via C++ gpio class (uses **sysfs**). Stream is kept open.	1	Main thread	Main blocks with busy-wait
2	6-7	C++		1	Main thread	Main blocks with epoll_wait
3	6-8	C++		2	Callback thread	Callback thread blocks with epoll_wait. Main doesn't block.
4	AES-21/BES/ GPIO-In/ C_devmem/ LED_echo.c	C	Direct, with mmap	1	Main thread	No blocking. Main busy-copies input to output.
5	6-12	C++	Indirect, via WiringPi library (uses **mmap**)	1	ISR	Main doesn't block. ISR runs when triggered.
6	*TBD: gpiomon.c*	C	Indirect, via libgpiod (uses **chardev**)	1	Main thread	Main blocks with ppoll (in gpiod_line_event_wait_bulk)
7	16-3	C	Indirect, via linux kernel gpio module	1	ISR	No blocking. ISR

Installing WiringPi

Refer to ERP Chapter 6 for instructions on how to install WiringPi.

Installing gpiomon and libgpiod (under construction)

To build the gpiomon program…

- *Use Raspberry Pi's Preferences -> Add / Remove Software and search for libgpiod. Install all five packages (gpiod, libgpiod,*
- *Follow the instructions at https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/about/ under BUILDING.*
- *Building uses autoconf.*

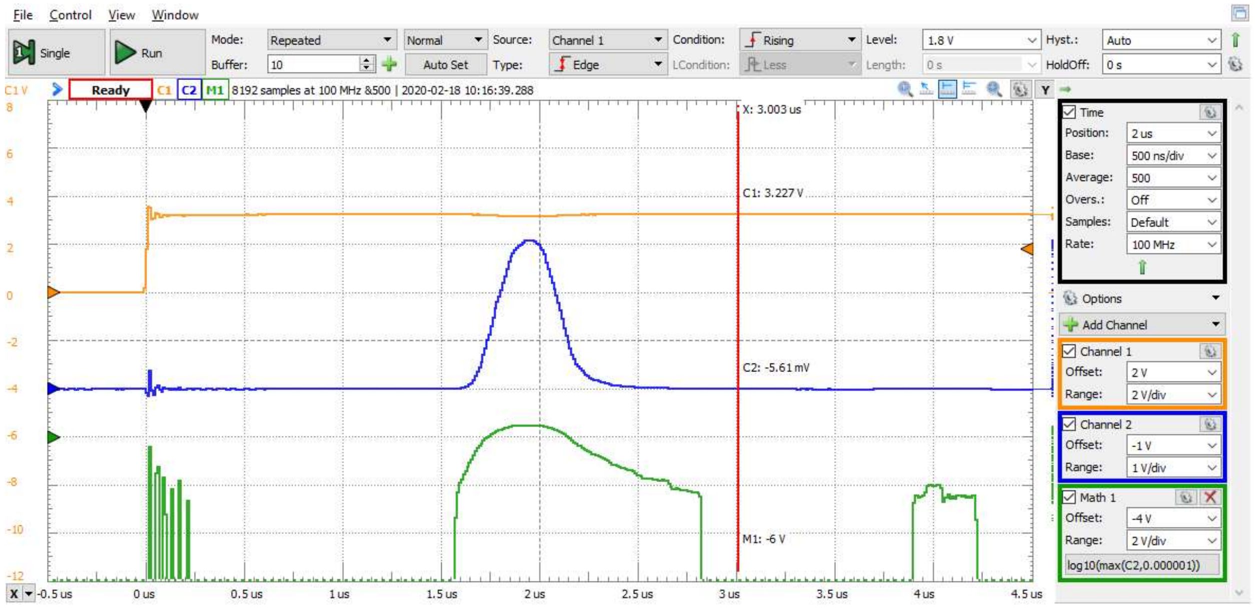
… *Further information to be determined and posted.*

Loadable Kernel Module

To build the C Loadable Kernel Module ISR (IRQ Handler), refer to the instructions in **ERP** Chapter 16.

- Read pp. 647-665.
- Be sure to edit the ISR in the kernel module so it does not call gpio_set_debounce(), which is not needed and would slow down the response.
- Build the kernel module by following the instructions in the book. The following steps worked for me:
    - p. 655: (upgrade took several minutes)
        - sudo apt update
        - sudo apt upgrade
        - sudo reboot
        - uname -a
        - sudo apt-get install linux-headers
        - apt-cache search linux-headers
    - p. 657:
        - make
        - ls -l
        - everything else
    - pp. 658-9: everything
    - pp. 664-665: everything

Testing and Analysis



Mode: Repeated | Normal | Source: Channel 1 | Condition: Rising | Level: 1.8 V | Hyst.: Auto
Buffer: 10 | Type: Edge | LCondition: Less | Length: 0 s | HoldOff: 0 s

Ready | 8192 samples at 100 MHz &500 | 2020-02-18 10:16:39.288

X: 3.003 us
C1: 3.227 V
C2: -5.61 mV
M1: -6 V

Time — Position: 2 us; Base: 500 ns/div; Average: 500; Overs.: Off; Samples: Default; Rate: 100 MHz

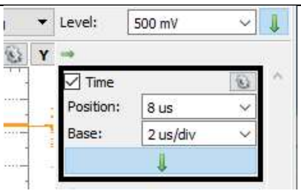
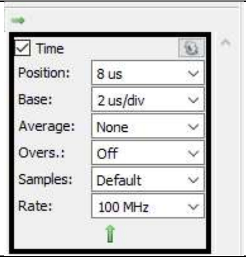
Channel 1 — Offset: 2 V; Range: 2 V/div

Channel 2 — Offset: -1 V; Range: 1 V/div

Math 1 — Offset: -4 V; Range: 2 V/div; log10(max(C2,0.000001))

You will run two batches of tests: one with the RPi4 running at 600 MHz, and another at 1.5 GHz. For each detector version, use your Analog Discovery 2 and Waveforms to measure the delays for ten seconds of events. Gather the following information for each detector:

- Distribution of response times for a program run of ten seconds. (upper right), set the number of samples to average large enough that it takes at least ten seconds to capture all the samples (and therefore update the display. This number of samples will vary based on how fast your code generates events (i.e. toggles the output signal). Configure Waveforms to create a response time distribution as follows:

In the Time panel on the right side of the window, press the green down arrow to reveal the Time options.	Change Average from None to 50 (for example). A larger average count will result in slower updates but better resolution.
 Level: 500 mV; Time — Position: 8 us; Base: 2 us/div	 Time — Position: 8 us; Base: 2 us/div; Average: None; Overs.: Off; Samples: Default; Rate: 100 MHz

- Determine the minimum and maximum observed **response times**. Here are two possible ways:
  - Add a math channel which displays the logarithm of the averaged channel 2 (e.g. log10(max(C2,0.0000001)), as shown above. This will make rare events easier to see.
  - Disable the averaging mode and enable the infinite persistence mode. Run the tests again (for at least ten seconds).

Report Contents

Response Time Plots

For each configuration, include a screenshot of response times (as shown in **Error! Reference source not found.**) for 600 MHz and another for 1.5 GHz.

Response Time Data

Complete a data table with the following response time statistics.

	Response Time					
	600 MHz			1.5 GHz		
Detector version	**Minimum**	**Distribution Peak (Mode)**	**Maximum**	**Minimum**	**Distribution Peak (Mode)**	**Maximum**
1						
2						
3						
4						
5						
6						
7						

Submit an archive of your source code and a PDF of the report on Moodle.