Digital 3 - Counting and Shifting

Creating more complete digital circuits.

Preparations:

This is again a lab that **you would do well to prepare for**. This lab would be considered a bit more challenging, but with some persistence and focus, you can make all of this work. Your homework assignment should also help with this lab.

The lab has significant stretch goals. It is OK if you do not get to do everything, but do focus so you can get as far as possible.

In Lab Preparation:

Make sure you have the most recent updates in the course libraries on your Raspberry Pi. Log in and go to the Phys605 directory. Once there, execute the command "git pull". This will update the libraries. Specifically, it will get the drivers in the Python/DevLib library. Check that the SN74HC165.py driver is there. If not, get help.

READ:

- Lecture Notes.
- Data sheets: SN74HC193, SN74HC165, SN74HC4040
  - These data sheets are *not* trivial! Specifically, read the description, which tells you what the device does, and study the timing diagram, which tells you how it does this.
- The DevLib code on GitHub for the shifter and display: https://github.com/mholtrop/Phys605/tree/master/Python/DevLib
- **Practical Electronics:**
  - 12.7 Counters (especially p783): read about the 74HC193 ripple counter. Selective reading in this chapter will help in your understanding of how this chip works.
  - 12.8: Shift registers (especially p797): read about the 74HC165 parallel to serial shift register.
  - (12.10: Displays) This is optional background reading on how displays work.
- **Exploring Raspberry Pi:**
  - Chapter 6: up to C++ control of GPIO. You should have read this already.
  - Chapter 8: Read about the I2C and SPI bus, up to UART
  - The discussion of the 74HC595 is relevant for us (and we do have that chip.) The 74HC595 is a SIPO (serial in parallel out), while you will be using a PISO (parallel in serial out), the 74HC165.

Goals:

- Learn to build larger, more complete logic circuits.
  - Learn to build a circuit step by step, and test at each step.
  - Learn to debug a logic circuit.
  - Learn about logic analyzers.
  - Learn to continue to build on an existing circuit.
  - Learn to plan the layout of a circuit.
- Learn about serial interfaces to the Raspberry Pi.

    - ♦ Read in data from a serial counter you built.
- Learn about timing in circuits.
- Stretch: Learn about the execution time uncertainty for code running on a Raspberry Pi.

Expectations:

- Keep good and accurate notes in your lab book.
- Carefully follow these instructions so you end up with the right measurements.
- Compare measurements with expected values.

Important

This is a multi-part lab where you will be putting together a serious bit of electronics. You are more likely to be successful if you plan ahead, and lay your circuit out carefully on your proto-board. **Do not make assumptions**, check the datasheet! Try to keep the circuitry compact in one area of the workspace, so that you have other parts free to make other circuits. You will need to save the circuits you are building for future experiments.

Stick with 3.3V:

The CMOS chips you are using work excellently well at 3.3 V. Since at some point you will be connecting the circuit to the Raspberry Pi, please stay at 3.3 V throughout. Before connecting any circuit to the RPi, please double check your power supply. Only use the power supply from the Analog Discovery or the Raspberry Pi.

Introduction:

In this lab we will finally make our first measurement using our digital knowledge and the Raspberry Pi. The simplest measurement to make with a computer is to count electrical pulses. If the pulses come in slowly enough, you can use a single GPIO port on the RPi, and in a loop, check how often the input goes from low to high. Each time it does so, you increment a counter variable. This works really well for counts that come in slowly, such as a button press. If the counts come in very rapidly, however, the RPi will not be able to keep up. This happens in part because the RPi has a lot of other things to do while it runs your program: it checks the network connectivity, it checks if you typed something, it checks and handles quite a few things. There are probably close to 100 small programs running at any one time. In this way, the RPi is different from a micro-controller (µC), which only runs *one* program, yours. A µC would be able to count up to a specific frequency of input pulses with 100% accuracy, but even with a µC you would need to add discreet electronics if you want to get to very high frequencies. The one programmable device that could keep up with fairly high frequencies would be an FPGA (Field Programmable Gate Array), but these sophisticated devices are both expensive and difficult to program. You Analog Discovery is based on an FPGA.

All clocks, except perhaps sundials, are counters that count the oscillations of some known oscillator. If you want to do accurate *timing* you can count the pulses from a known high frequency clock source for the given time interval. The elapsed time is then proportional to the count. Such high frequency counters and accurate timers are used for many different types of physics research. In particle physics, Time to Digital Converters (TDC) are used to measure the time it takes for a particle to travel from one detector to another, which allows one to compute the velocity of the particle.

In this lab we will build a dedicated counter circuit to count pulses. The SN74HC193N chip and the SN74HC4040N chip are dedicated counter circuits. The SN74HC193N has only 4-bits to count with, so it can count from 0 up to 15 (2^4-1). This chip can count down as well as up, and has the capability to load a number, so you can count up or down from a pre-loaded point, which makes it quite versatile. You can chain any number of these chips together to get counters that can count to larger numbers. The SN74HC4040N has 12-bits to count with, so it can count up to 2^{12} -1= 4095 with a single counter, and two counters can count to just over 16 million. This counter work up to 50MHz without a problem, giving you 20 ns timing resolution. If you want to count even higher frequencies, you can buy specialized fast counters, but those usually come in tiny surface mount packages that are a bit too advanced for our setup, and are also quite expensive.

Reading out such a large number from a counter directly would take a lot of GPIO pins, one for each bit, and additional pins for the controls (load, start, stop etc.). This is not practical, so instead of the RPi reading the counter out directly, we copy the contents of the counters into a shift register and then read the bits out using only two GPIO pins, one for the serial data and one for the clock. This is very close to what an SPI serial bus does. We will use the SN74HC165N chip, which is an 8-bit parallel in, serial out (PISO) and serial in serial out (SISO) shift register chip. The SISO property of this chip allows us to chain many of them together so that we can read any number of bits. There are also chips on the market that combine the counting and shifting capabilities. The SN74LV8154 has two 16-bit counters and a PISO shift register all in one chip, allowing for a single chip 32-bit counter.

The final component you would need to do *timing,* and not only *counting*, is a reliable clock source. You can then count the pulses from the clock source to do timing. You can also use the clock source for the gate, opening it for a period of Δt, in which case your counter measures a frequency as F=counts/Δt. In this lab we can use the Digilent Analog Discovery to create a solid and stable clock source for us[1]. You can add a dedicated clock system at some later time is you would like to. Ask me about the options for clock systems.

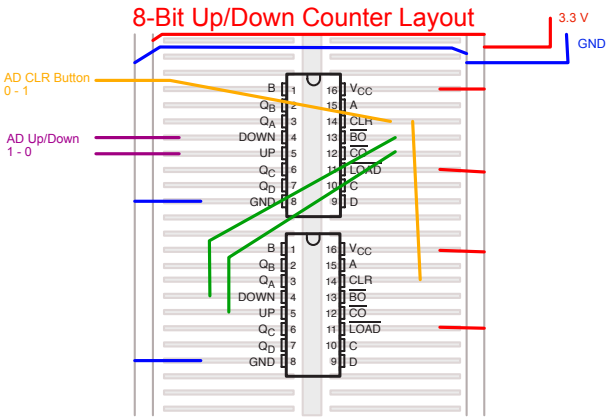
The counting circuits

The circuits in this lab are a bit more involved. You can successfully wire them up if you are careful and patient, and if you make sure each step works by testing your circuit at regular intervals. If you have a question, get help!

The 8-bit counter circuit.

Start by exploring one of the 4-bit counters. Wire up one of the SN74HC193N counters and connect the outputs (QA – QD) to the logic inputs 0-3 on the Analog Discovery (AD). Note from the datasheet that the "load" pin has to be tied to +3.3V for the counter to work. If you pulse it negative, it will load the values from the A-D input pins. There is a "clear" (CLR) pin which needs to be tied to ground, unless you want to clear the counter (set the value to zero). There are *two* clock inputs, one for counting up and one for counting down. If you want to count up, you *must* connect the down clock input to +3.3V, and vice versa. This counter advances (i.e. counts) on the low to high transition of one of the clock inputs. If you connect the up and down clock inputs to buttons of the AD, set the buttons for 1→ 0 (released 1, pressed 0). You can connect the CLR to another button set for 0→ 1. *(See the TASKS summary below.)

Extending the Counter to 8-bits.



Now we will extend the counter to an 8-bit counting device. I suggest the layout shown in the figure on the right. Create this circuit at the very top of the left most strip of your large proto-board. That was you have space later for everything else you will need to build. Note the convention to use the long up/down strips of your bread-board for powering all the chips. This is, in general, a good choice for layout. Try to keep your circuit neat!

The QA1-QD1 and QA2-QD2 output go to the AD digital I/O (LEDS). Test the circuit as before, making sure it counts correctly past 15, all the way up to 255.

Clearly, as the bits increase, it gets more challenging to test whether the counter follows the expected sequence. Pressing a button 255 times might still be OK, but it won't be for a 12 (or 24) bit counter. A better way to test these circuits is to use the logic analyzer and the pattern generator of the AD. The logic analyzer is like an oscilloscope for digital signals, so it shows a trace but only has it high or low. The advantage is that you can have many such traces, up to 16. The pattern generator can create any digital pattern on any of the digital I/O lines of the AD. The combination of these two features allow you to investigate most digital circuits is a

[1] My measurements show that this source is accurate to about 10 parts per million (ppm).

lot of detail.

You can leave the AD hooked up as before, but now look at the signals with the logic analyzer. Add the inputs in order, from bit0 to bit7 on your counter. Instead of the buttons, use the pattern generator. You would want to set your clock output to "PP" (push-pull), type to "Clock", and Parameter 1 to a decent frequency (1 kHz, so one pulse every millisecond). At the top, set trigger to manual, wait to "none", Run to 127 ms, and repeat to 1. Now if you press run, the AD will pulse the output (which you send into your clock) exactly 127 times. This is a very accurate system. You can set it to 50 MHz for one second and it will output exactly 50 million pulses.

TASKS Summary:

- Wire up one SN74HC193N counter, and test it.
  - Copy down your *exact* circuit into your lab book.
  - Do the LEDS count a binary sequence as you expect?
  - Does it work for both UP and DOWN counting?
  - Can you clear the counter as expected?
- Chain a second 4-bit counter to the first one to make an 8-bit counter. Use the AD inputs 4-7 for the new outputs.
  - The CLR (clear) input of counter 2 should be connected to the CLR on counter 1 so both chips clear at the same time.
  - Copy the way you chained together the counters into your lab book.
  - Does the output still make sense for both counting directions, all the way up to 255?
- Stop the StaticIO (red stop sign in tab) and open a Logic screen.
  - On the logic screen, add a "signal" input for lines 0-7
  - Set the logic analyzer to "normal" trigger, and the trigger for a rising edge on DIO-0.
- Open an "Patterns" window in AD.
  - Create 2 outputs, one for UP and one for DOWN.
  - You will want PP (=push-pull) output.
  - You can now set a pattern of UP or DOWN (or both). The simplest pattern is "Clock", at a specific frequency, for a fixed duration. The output will be the *exact* number of clock pulses for that duration. You can also set a custom sequence.
  - Setup the sequence to count UP 255 pulses.
  - Clear your counter before starting the sequence.
  - You can start the sequence with the "run" button.
- On the Logic Analyzer screen, make sure your output makes sense.
  - Save the output for your explanation of the circuit in your progress report.
- Optional extra test: Connect the Quadrature Decoder from last lab to your counter.
  - Does the counter accurately count the clicks when you turn the dial?

Readout with the Raspberry Pi.

The next step is to read the counter you just made with the RPi. To do so, connect one of the SN74HC165N 8-bit shifters to the outputs of your 8-bit counter. The SN74HC165 will shift the highest bit (H) out first. The driver in DevLib expects the most significant bit (MSB) first, which in this case is bit7, representing the number 2^7. So you want QA from the first counter to go to the corresponding A input of the shifter, and so on. The second counter's QA goes to the E input of the shifter, etc. This way you will indeed have the "most significant bit" (MSB) to be shifted out first. Tie the SER input to ground so the shifter reads in a zero on each clock, and also tie the CLK INH (clock inhibit) to ground.

Connect the QH output to an unused input of the AD, and connect the CLK input and SH/LDbar to two other DIO lines which you make into outputs. Test this circuit using the AD StaticIO (easiest) and more challenging the logic analyzer and pattern generator.

Once you are satisfied it works, connect the circuit to the RPi GPIO lines and test the system using the SN74HC165 driver which you find in the DevLib library. From your homework, you should know how this works and be able to write a small program that makes use of this library. (Hint: the main() at the bottom of the driver tests the driver, and

yes, that is a good start for your own program. Do not modify the driver! Write a program that imports the driver.)

Tasks Summary:

- Build the circuit and record the schematic in your notebook.
- To test the circuit:
  - Generate a known number on the counter.
  - Pulse the SH/LDbar input *low*, to load the shifter.
  - Pulse the CLK input low→high. At each pulse a different bit is shown on QH.
  - Check that these bits correspond to the number you loaded! If not there is a wiring issue.
- When you are satisfied the circuit works:
  - Check that the voltage on the circuit is no more than 3.3V.
  - Disconnect the 3 shifter signals from the AD.
  - Connect the 3 shifter signals (QH, CLK and SH/LDbar) to GPIO pins on the RPi.
    - If you don't want to modify the driver use:
      - QH – GPIO 18
      - CLK – GPIO 19
      - SH/LD – GPIO 20
- Readout data on the Raspberry Pi:
  - On the RPi use the SN74HC165 driver in the DevLib library. It comes from GitHub in Python/DevLib:
    - Login on the RPi.
    - If the "Phys605" directory is not there, checkout the course code:
      - git clone https://github.com/mholtrop/Phys605.git
      - You now have a directory called "Phys605" and in it a directory called "Python"
      - In the "Python" directory, you find the directory "DevLib", in there are some useful Python drivers for this lab. The first one you want is "SN74HC165.py".
      - You should have read and understood the code already from your preparations.
  - You don't want to modify the driver, instead write a program that *uses* it.
  - Clear the counter and give it a known number of pulses. Then run your script to read out this number. Do the numbers match?
    - Repeat for several numbers, so you are sure it works.
    - If the number does not match, check with the AD if you are indeed loading the number that you thought was there.
      - Do you have a wiring issue? Go back to the datasheet and check connections.

Part II: Extending the bits

This part of this lab is highly recommended, but if it is really too complicated for you or you are running out of time, you can skip it and go to the "Timing" section and complete the lab with your 8 bit counter.

The 8-bit UP/Down counter you created is nice, but it counts only to 255, which is pretty limited in use. Leave your 8-bit counter and shifter untouched, but now create a new counter with two SN74HC4040 chips. This will be a second counter with 24-bits, that only counts up. It will go to up to 2^{24}-1=16777215. You will need 3 shift registers to read it all out. You chain together the shift registers by connecting all the clock (CLK) inputs together, and all the load (SH/LDbar) inputs together, and connecting SER (serial in) of chip 2 to the QH of chip 1, and the SER of chip 3 to the QH of chip 2. Clearly, with some effort, you could extend this circuit to any number of bits. Note that the "4040" counters do not have a very logical assignment of the pin numbers, so be careful when you build this. You may want to make a table before you start wiring everything, since the outputs of the counters (from A to L) do match the 3 shifter inputs very well. Make sure you end up with a circuit that shifts the MSB out first.

If you need help, ASK!

Tasks:

- Build the circuit, and document in some detail what you did.

- Hook the circuit up to the RPi, same as before. Note that you do not need any more GPIO pins, though you now have a lot higher count.
- Test the circuit.
  - Clear the counter, send a known number of pulses, and read the number with the RPi.
  - Note that you need to modify the ReadSerial.py script to increase the number of bits.
  - The AD is very accurate. You can send a clock of 50MHz for 0.1 seconds and get exactly 5 million pulses. This is a great way to make sure you didn't cross any wires.

Making a Timer

The SN74HC4040 does not have a clock inhibit, but it is easy to add one. Get a NAND (SN74HC00N) and create a gated clock input for your counter. One input from one NAND gate goes to one of the RPi outputs (e.g. GPIO 16), and the other input goes to the AD clock output. The output goes to the input CLK of the first stage of your counter.

Connect the CLEAR of the counter to another port on the RPi (e.g. GPIO 17). Note that the clear happens when this is pulsed to zero, so in the software we need to make sure we pull it high when we want the counter to run.

Add a Display

So far we have used the desktop computer to connect to the RPi and get information back. It is more fun to give the RPi a display of its own, so that it can become a more complete standalone system. The MAX7219 chip makes it relatively easy to interface a display of 8 7-segment numbers. It uses the SPI interface to communicate with the RPi. There is a driver in DevLib, which can use either GPIO pins or the SPI interface. If you do not want to modify the code, it is setup to use: GPIO4 for data in (DIN), GPIO 5 for the clock (CLK), and GPIO 6 for chip select (CS). Hookup your display and use the driver's test function to see if it works. You can also test it from the Python command line interface.

A different type of display would be the 4x32 character blue LCD display. There are a few of these available, but you will need to find or create your own driver, which may be a bit much for this lab.

Testing the Counter

On your RPi, in the directory Phys605/Python/Counter you find two example codes, "GatedCounter.py" and "Gated_Clock_calibrate.py". These codes should function as it, but you may need to modify these codes slightly for your own purpose (i.e. change the pin definitions). This can be a starting point for your own code.

Tasks:

- Build the gated counter circuit and record the schematic in your notebook.
- Hookup the display.
  - Test to make sure your display is working as expected.
- To test the circuit:
  - Set the AD to clock out a high frequency continuously (i.e. start with 1MHz, and increase to 50MHz).
  - The gate will now control the counter.
  - Run the program GatedCounter.py, which will clear the counter, open the gate, close the gate, and read the number of counts in a loop.
  - Note that this does not always give the same number. Why?
  - Modify the GatedCounter.py code to increase the time between the gate open and gate close. Do the counter numbers increase proportionally?

Discovery/Extra

There are lots of things you can time. Some of the following ideas could be a sub-component of a final project, or be expanded into a full final project.

- How about measuring your own reaction time?

    - You could rewire the circuit so that the gate close is controlled by a button and have the RPi light up an LED when the gate is opened.
    - Hint: use a Flip-Flop to control your gate. The "set" comes from the RPi, the "reset" from your button.
    - You then press the button and the gate is closed. The counter contains the time it took between the LED going on and the button being pressed. Can you calibrate the counter so the display shows seconds with an accuracy of micro-seconds? What is the required clock frequency?
    - It would be a better test of reaction time if the time to light the LED is random. Can you make this random?
    - Can you turn this into a game? You could keep a score of reaction times and display the points you gained...
  - You could test the accuracy of the AD clock against the clock in the RPi.
    - You could do even better by comparing against one of the internet clocks, or a GPS clock. If you run the RPi service "ntpd", then occasionally the RPi re-synchronizes its internal clock against the internet time, which is then quite accurate, though it may not be quite so accurate from second to second.
    - Add your own time source, using your own crystal oscillator which you calibrate carefully.
    - Compare against the DS3231 RTC chip.
    - For reference, NIST does this a lot better, see: https://www.nist.gov/news-events/news/2014/04/nist-launches-new-us-time-standard-nist-f2-atomic-clock , but they too are basically just counting. These very accurate references are quite important, and create interesting science opportunities.
  - If you are running a program for which the timing is critical, this external timer can be a very accurate way to measure the time delays in your code, and the fluctuations of those time delays.
  - You could make a “frequency counter”, where you use a known Delta-T during which you open your gate. The frequency on the counter is then N/Delta-T. You can use the AD for a 1Hz reference, or you can use an RTC (Real Time Clock). Such frequency counters are very common in laboratories, and are also used for (amateur) radio setups. There should be a frequency counter available in the lab to compare against.
  - The 8-bit up/down counter could be used to read out a rotary encoder. The encoder would need some extra circuitry, see: https://www.systemvision.com/design/rotary-encoder-decoder-ii
  - With some additional circuitry, or with additional sensors and/or other sources of clock signals, you can do all sorts of experiments with this counter:
    - Time the echo of an audio pulse, thus creating a sonar device.
    - Time the charge rate and/or discharge rate of a capacitor, creating an accurate capacitance meter.

For Progress Report:

Required Parts in the report:

- Describe the circuit for your 8-bit Up/Down counter with the shifter for readout.
- Include a schematic.
  - There are no good, free, online drawing programs that I could find for digital circuits other than gates. You can try http://www.digikey.com/schemeit, which does gates and square boxes for the chips. There are a number of free programs you can download. You can also use a CAD program or other program. If none of these work for you, use a pencil and a ruler (it still needs to look professional) and a decent scanner.
- Describe how you obtained the pattern of the 8 outputs from the logic analyzer, and include the graph that you acquired in lab.
- Describe how you created a known number of pulses for the counter, and how you read out your system with the RPi.
- Describe the circuit for your 24-bit counter with shifter for readout. If you did not get to build this, explain why you were not able to.
- Describe the gated timer.
- Describe how you tested this system so that you know it is working.
  - What were the results of the GatedCounter tests, and *exactly* how were they obtained?
- Describe any extra tests you did.
- What are possible things you might want to use these circuits for?
- What you learned and who helped section.