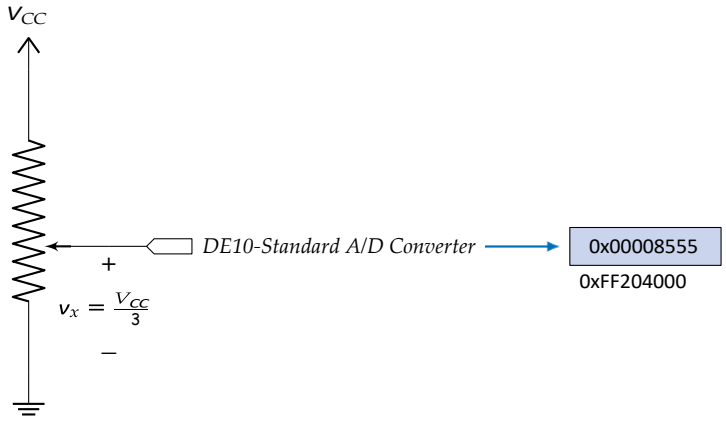
ECE3375B Lab 3: GPIO and A/D Conversion

Objective

Analog-to-digital (A/D) conversion requires reading the voltages of analog devices. In this lab, we have some extra hardware that plugs into the DE10-Standard board. This hardware connects some potentiometers (dials) to the DE10-Standard's A/D input. The potentiometers act as a simple voltage divider, so turning the dials adjusts the analog voltage seen by the A/D converter. You will program the DE10-Standard so that you can turn the dials and read the voltage. We will display the voltage on green LEDs connected to the GPIO port (not the red LEDs we've used before). We will use a slide switch to control which of the two potentiometer dials are displayed on the LEDs.



A/D Converter reading in voltage v_x.

Version History: This lab manual was originally written by Prof. Ken McIsaac and used in ECE3375 from 2019 to 2022. It was rewritten by Prof. John McLeod in 2023.

Remember that *most* the features of hardware and the debugging interface are available in the online simulator: https://cpulator.01xz.net/?sys=arm-de1soc. The online simulator has the A/D converter, but obviously it can't connect to hardware. The A/D converter in the simulator simply returns an arbitrary value each time you read from it. You can still prepare for Lab 3 with the A/D controller on the simulator: as long as your program successfully reads from the A/D converter it will work, but the value read from the A/D converter doesn't have any physical meaning on the simulator.

You may write your code in Assembly or C. If you use C, the names of the hardware in address_map_arm.h are given here for convenience. Of course you do not need to use this header file in your code, and you do not need to use the same names for the hardware addresses if you prefer something different.

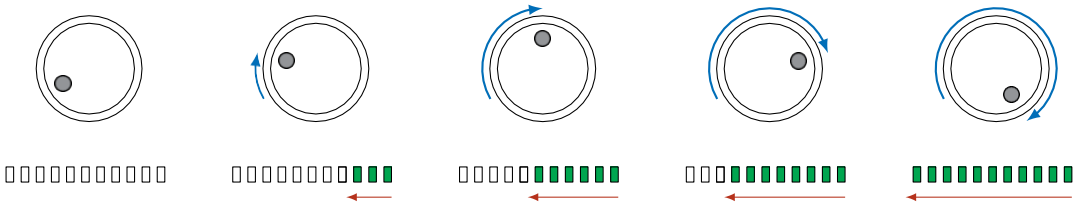
Deliverables

For this lab, all you need to do is demonstrate the working hardware and answer some questions from the TAs. The TAs may wish to play around with your potentiometer dial.

You do not need to submit any report or show your code, however if you are **unable** to get the timer working in hardware then you may show the TAs your code to possibly obtain partial credit for this lab.

Hardware Operation

The extra hardware we will plug into the DE10-Standard board includes two potentiometer dials and ten green LEDs. The two potentiometers provide a voltage from V_{CC} (either 3.3 V or 5 V, but it doesn't matter what the exact value is) down to zero depending on how the dial is turned. We want to read this analog voltage with the ARM Cortex A9, then represent this voltage as a sliding scale on the green LEDs. All LEDs should be off if the voltage is zero, then more LEDs progressively light up from right-to-left (or left-to-right if you want, it doesn't really matter) until all LEDs are lit up at the maximum potentiometer voltage.

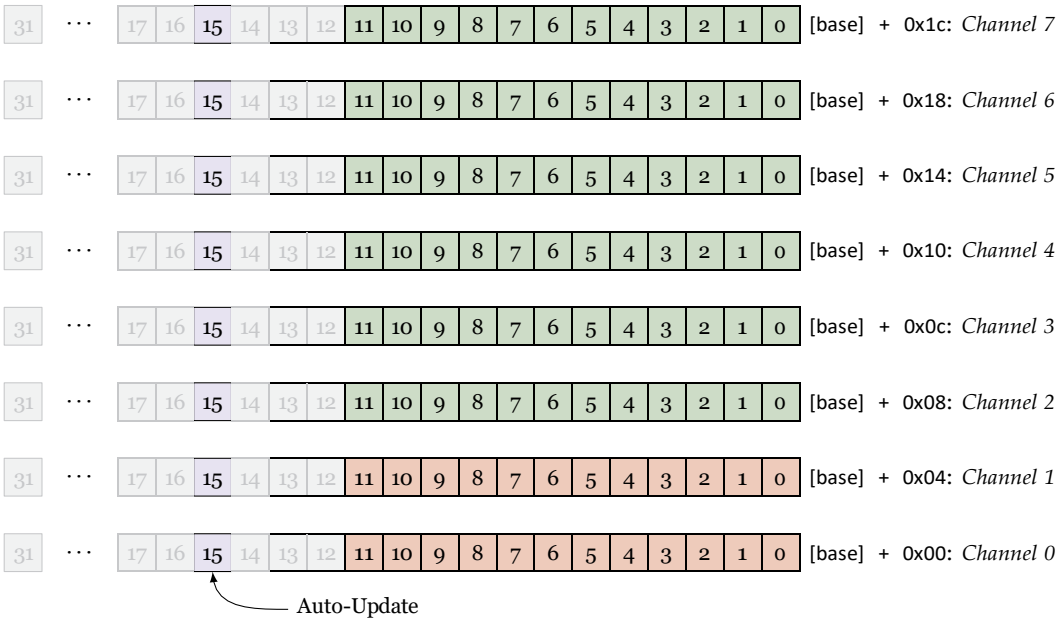


Because there are two potentiometer dials, use sliding switch SW0 on the DE10-Standard board to toggle between which one is displayed.

If you are testing the code on the simulator, there aren't any actual LEDs connected to the GPIO port, but you can observe whether the pins are input/output and whether they have a high or low voltage based on the colour and presence of a check mark. Therefore it is easy to test this part in the simulator and observe whether it would work in hardware.

A/D Converter

This lab obviously requires using an A/D converter. The A/D converter on the DE10-Standard is an 8-channel, 12-bit ADC which uses the successive-approximation method. The extra hardware with the potentiometers plugs into channels 0 and 1 of the A/D converter. The base address of the A/D converter is 0xFF204000. This address is called ADC_BASE in address_map_arm.h. The A/D converter has the structure shown below.



The A/D converter functions as follows:

- To **update all channels** write any value to the *Channel 0* register.
- To **have all channels auto-update** write any value to the *Channel 1* register.
- To **read from a channel**, read from the corresponding *Channel n* register. The lowest 12 bits are the converted data.

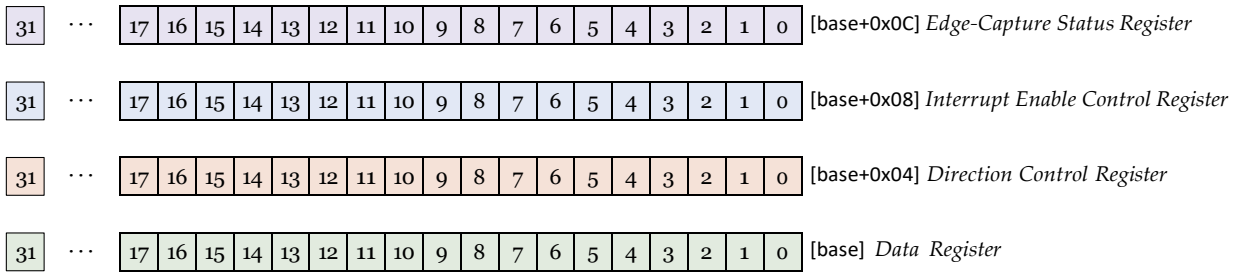
- To **check if conversion is complete**, check bit 15 *after reading from the channel*. If that bit is set to one, the conversion was complete. If it is zero, the conversion was incomplete.
- Bit 15 is **automatically cleared** after the channel is read.

Because bit 15 is automatically cleared, you cannot check the status then read the data in two steps: you must read from the channel and check bit 15 without clearing the rest of the data. If the channels are set to **auto-update** and you read from the channel to check bit 15, then read from the channel *again* to get the data, the data you acquire in the second read will be the partially-converted data from the next update!

GPIO Ports

Just to spice things up, we will display the A/D converter reading on the green LEDs provided on the same hardware plugin that hosts the potentiometer dials. These will be plugged into one of the two 32-pin GPIO expansion ports. Functionally these are the same as the red LEDs on the board: the red LEDs, in fact, also use a GPIO port but one in which the direction is fixed for output. Here, because the DE10-Standard doesn't know what is plugged into the GPIO port, we will need to configure the port for output.

The DE10-Standard has two 32-pin GPIO expansion ports, but only one has the correct physical pin hardware. This one is at memory address `0xFF200060` and is called `JP1_BASE` in `address map_arm.h`. The GPIO expansion port has the structure shown below.



The GPIO expansion port functions as follows:

- To **configure a pin as output**, set the corresponding bit in the *Direction Control Register* to 1.
- To **configure a pin as input**, set the corresponding bit in the *Direction Control Register* to 0.
- To **write to a pin**, write the appropriate data to the corresponding bit in the *Data Register*. This has no effect if that pin was set to **input**.
- To **read from a pin**, read from the *Data Register* and using masking or bitwise logic to check the value of the corresponding bit. This does not produce useful data if that pin was set to **output**.

We won't be implemented interrupt-enabled hardware, so the *Interrupt Enable Control Register* and teh *Edge-Capture Status Register* can be ignored.

In this lab, the additional hardware that connects to the GPIO expansion port has LEDs connected to pins 0 to 9. LEDs are output-only hardware (hopefully that is obvious), so bits 0 to 9 in the *Direction Control Register* must be set to output.

Evaluation

You will be evaluated based on demonstrating working hardware and answering questions from the TA.

Category	Grade
Demonstrating working A/D Converter and GPIO LED Display	40
Answering Questions	10

If you are unable to demonstrate working hardware, then try to get as much of the hardware working as possible and show the TA your code to obtain partial credit.