PHY 335, Unit 7 Digital electronics, ADC and DAC

Mini Lecture topics:

- **Logic families**
- **Oscillators & Clocks**
- **Combinatoric logic, simple logic gates & logic simplification**
- **Binary Numbers, simple adder**
- **Sequential logic & one bit memory: flip flops**
- **Other building blocks**
- **DAC and ADCs**

For some of the questions on this unit, we will use the electronics module of tinkercad – see `www.tinkercad.com`. You can "copy and tinker" with a basic setup I generated, available here: `https://www.tinkercad.com/things/8eXYvwUzNXC`.

1. Combinatoric logic: XOR using the universal NAND gate. Design and build an Exclusive-OR (XOR) gate using the quad NAND package 7400. Show its truth table. Show the functionality using scope probes or by connecting LEDs. (If you use the LEDs, make sure to limit the current with a resistor in the range 300Ω to 1 kΩ.)

2. Binary Numbers and more combinatoric logic

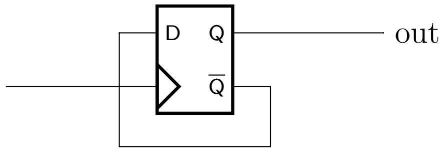
   (a) Convert 137_{10} into binary notation. Convert 1001001_2 into decimal. Convert both numbers into hexadecimal.

   (b) Write the truth table for a one-bit binary full adder (FA). A FA has three one-bit inputs, A, B, and C_{in} and two one-bit outputs, S (sum) and C_{out} (carry). Design and build the circuit using basic gates (AND, OR, NOT, XOR, NAND, NOR)

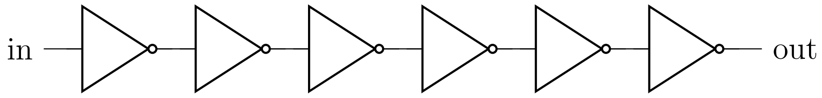
3. Sequential Logic: Flip-Flop (FF, memory element): Use one flip-flop in the 7474 dual package. Use the SG to drive a 0-5 V square wave for the FF clock to a DQ flip-flop. (Be sure that the voltages are correct before you insert the ICs. You may need to use the offset function of the SG.)

   (a) Threshold tests: Use a 20 kOhm pot as a voltage divider to provide a variable 0-5 V signal for the D input of the FF. Use Ch1 and Ch2 of the scope to monitor the clock signal and the Q output. Watch the Q output change when you increase the applied input voltage level from 0 V. How does the output depend on the input voltage level (make a table with numbers)? Now change Ch2 to the $\overline{Q}$ output. Repeat the measurement. Do the measured logic 0 and logic 1 transition voltages agree with the TTL standard?

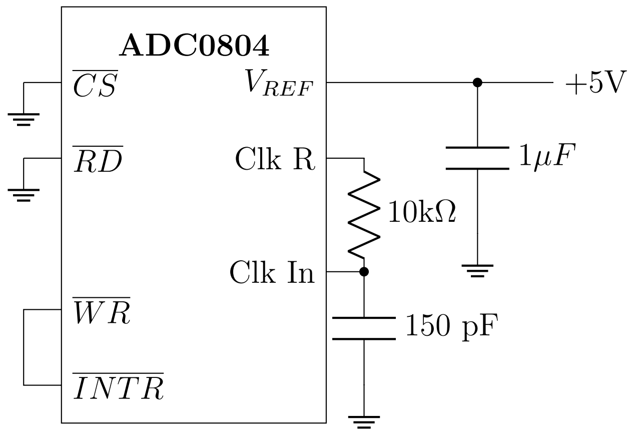
(b) Clock edge and propagation delay: Construct the circuit below. Which edge of the clock (rising or falling) causes the information present at the D input to be transferred to the Q output? (Hint: use the scope in the run-once = single trigger mode.) Measure the propagation delay (the time from the rising edge of the clock at the input to the change of the output state).



(c) Build the circuit below using a single 7404 package. Again measure the propagation delay. In this case, per-gate delay is the measured delay divided by six. Sketch the input and output waveforms in your lab book (and then in your report).



4. Analog to Digital conversion: Use the output voltage from part 4 as the input to an ADC0804 in free running mode (See below and/or an ADC0804 data sheet for the control connections to set the ADC in free-run mode). To start the ADC, you have to pull $\overline{INTR}$ to ground momentarily. Use a potentiometer to generate a voltage between 0 and 5V as the $V+$ input for the ADC. Connect $V-$ to ground. Measure several input voltages and compare with the ADC output. You can either measure the output voltages with a DVM, or use LEDs to display them. Note that the outputs can only sink current, not source it. This means that you have to connect your LEDs to 5V, with a series resistor of 1.3-1.5k. The LEDs will then light up when the output is LOW. Plot the values. How linear is the ADC?



5. (Tinkercad) Microcontroller

   (a) “Copy and tinker” the basic setup from above. Add an Arduino Uno R3 to it. An Arduino is a small microcontroller development board, based on Atmel (now Microchip) AVR microcontroller. The microcontroller is programmable, and Tinkercad supports either simple graphical programming via blocks, or proper programming via C. For this part, the programming via blocks is enough.

   (b) Connect one of the “GND” pins of the Arduino with the ground on your breadboard.

   (c) Connect a potentiometer, so that it provides a voltage between 0 and 5V to the A0 pin of the Arduino.

   (d) Connect a piezo speaker to digital pin 3 and ground.

   (e) Click on code, and remove all existing blocks on the right (right-click, delete block). The blocks you place in the right part of the window will be executed from top to bottom. The moment the simulation reaches the bottom, it will jump back to the beginning and start over.

   (f) We want to read in the analog value from the potentiometer. The uC has an integrated 10 bit ADC.

   (g) We first generate a new variable to save the value we got from the ADC. Click the Variables section under Blocks, and then “Create variable”. Name the new variable poti. This will crate new blocks to add to the code.

   (h) As the first block, we’ll add “set poti to 0”.

   (i) Now, select the input blocks, an select “read analog pin A0”. Place it on top of the 0 from the block before. It should “snap in”, and the block should read “set poti to read analog pin a0”

   (j) We want to output this as a tone on pin 3. However, we first have to map into to the right range: Our analog value is between 0 and 1024, but the tone must be between 35 and 127. Fortunately, there is a map function for that. Make a new variable, named “note”, and add a block “set note to”. Drop in a map from the Math block menu on the 0 of the set tone, drop in poti to the first 0 of the map, and set the range to 35 to 127. It should now read “set note to map poti to range 35 to 127”.

   (k) Now we have to output it. From the Output blocks, add a block “play speaker on pin...”. Set the pin to 3, drop in “note” on the 60 behind tone.

   (l) Press “Start Simulation”. You should hear a tone, proportional to the poti settings.

   (m) In the code view, select “Blocks+Text” instead of “Blocks”. Behind the scenes, Tinkercad convertes the blocks into C code. Try to match the lines in the code to the blocks.

(n) Now it's up to you to add more funtionality. Connect 4 pins of the arduino (for example 4 to 7) to the LEDs. Can you make them light up, proportional to the position of the poti? Hint: The easiest (but not most elegant) solution is to use 4 "if then else" blocks from the control blocks. These blocks check a condition, and if the conditions is fulfilled, do the first part, else the second. The conditions you can assemble with the blocks in Math (the second block!), and to set the pin, you use set pin to high (or low to switch the LED off). To say it in words, for example for the lowest LED, you want to build something like this: If poti>150, then set pin 4 to high, else set pin 4 to low. Did it work?

6. Bonus: (Tinkercad) C Programming. The block programming is quickly at it's limit. Let's do some real programming.

   (a) Start a fresh tinkercad page and place one Arduino.

   (b) Connect three potis between GND, and 5V, with the middle connections to A0, A1 and A2 (one each)

   (c) Connect a NeoPixel Ring 24 to the Arduino: PWR to +5V, G to GND and IN to pin 0.

   (d) The NeoPixel are digitially addressable RGB LEDs. Let's write a program so that each pot controls one color component. We want that all LEDs light up the same color.

   (e) First we have to add the NeoPixel library to the code. Click on the file cabinet icon, and select NeoPixel. The icon on the right leads you to the documentation for it. You can copy the initializing code from the documentation, and place it before the void setup() line. Change the two defines to the right values.

   (f) In the setup function, add strip.begin().

   (g) in the loop function, you now have to read in the three potentiometers via the ADC, map it into the range of 0 to 255 (that is, divide by 4), and set all pixels to the three colors. At the end, call strip.show().

   (h) Have a look at the startup and loop functions from the unit before. You can see how to read the ADC. You have to set the pinMode for A0-A2 to INPUT, and for pin 0 to OUTPUT.

   (i) Show a screenshot of the circuit and the code you wrote in the report.