Boynton Power Multiplier (BPM) Project v1.0.1*
ECEN 4303: Digital Integrated Circuit Design
Oklahoma State University
Fall 2018 Semester

1 Introduction to Bit Slices

This projects involves an intense project which you will design a 8-bit custom-based datapath slice. Its also named after a great inspiration to our students, Coach Mike Boynton. We admire his enthusiasm, hard work, and what he teaches OSU about life and passion for making the world a better place!

Datapath layout is often performed in custom-level cells for several reasons [1]. The datapath is usually the portion of a circuit that performs the majority of the useful work and contains the circuit's critical path. Therefore, it is worthwhile to make the datpath as fast as possible. Trained mask designers spend hours trying to produce dense and efficient layouts that can minimize the delays along a circuit's critical path or exhibit low power characteristics. The high density is also important to make the layout as compact as possible, since datapath layout is often repeated. Although the custom portion of this design is not super critical to the process, the overall idea of the project will give you a sense of what is needed to design integrated circuits as a whole. It also gives you an idea how custom-level design is critical to target a specific integrated circuit for speed and/or power.

In this project you will putting together several parts within a custom block, such as an adder, multiplexor, and register. For this assignment, utilizing a bit slice technique will help you complete the design more efficiently. Bit-slicing is a popular technique in partitioning designs, so that they are more manageable [1]. This is fairly common in VLSI, since we would like to minimize the amount of time we spend doing custom level layout because it takes a majority of the time in most designs. A bit-slice involves designing datapath elements together for one bit and then repeating the cells in the direction of the word slice. Careful attention to floorplanning and recognizing certain repetitive areas in a cell can lead to good compact designs.

This project will also involve some standard-cell design portion putting both together to form the custom-cell portion and standard-cell portion along with our chip pad to form the final chip. As indicated in class, students can elect to have their design fabricated if they meet specific check marks for verification on the design (e.g., Layout vs. Schematic).

1.1 Planning Layout

The use of hierarchy within a layout is critical. Many people fail to see the correlation between object oriented design and hardware. However, these ideas behind the Object Oriented paradigm are extensively used within the hardware domain. Re-utilization of layout cells can greatly decrease your overall complexity, simplify debugging and limit your design time.

Creating diagrams of you design in terms of its hierarchy is also recommended. For example, a full adder can be made with two xor gates and an or gate. Similarly, an exclusive-or (XOR) gate can be made with NAND, NOR, and Inverter gates.

In this design, try to keep all the ndiff regions near the GND line and the pdiff regions near the VDD line. This is because of the need for body contacts. In general, inputs should be on the left and the top sides of the cells and the outputs should be on the right and the bottom. Make all the cells the same height so that the VDD and GND lines are continuous when abutment is used. This process of keeping everything a certain height is called **pitch matching**.

*Note that this document is updated periodically to provide you with the most reliable information to complete the project. Although previous versions of this document are basically correct, frequent revisions are made in order to help you save time and **not** to hinder your performance. Therefore, make sure you check back periodically for updates. I will add revision numbers to help you identify which version of this document you have.

1.2 Floorplanning

Just like our layout assignments, without a careful plan on how you actually put together your circuits, can lead to hours of pain and agony. This is one of the great things about the stick diagram. One of the ways to extend this to higher level designs, especially designs that utilize hierarchy, is to use floorplanning. Floorplanning involves the simple diagramming of your circuit and how it gets put together.

As we learned previous assignments, a stick diagram may help us to visualize the structure, but it does not help us with the sizing. You can extend this by using pitch. Pitch is the distance, usually between two metals, between each layer. There are various ways to measure the pitch, such as between the center of a metal and its next edge. One way that you can incorporate this into your stick diagrams is to allow a 8 lambda pitch between each contact. Of course, you will have to account for the minimal width of each diffusion region, but it can help you with getting the dimensions of each cell.

Once you have a more or less idea of the size of your cell, you can use this to floorplan your layout. When utilizing hierarchy, certain functions will use each other ostensibly. For example, an AND gate would utilize an NAND and and inverter gate. How you put them together, would involve high-level floorplanning. In other words, putting objects or blocks together and placing dimensions together to get an idea of the size necessary. There may be many times when you have to floorplan your circuit to fit a certain size. Therefore, the better the floorplan, the better the implementation. An example of a floorplan is shown in Figure 1.

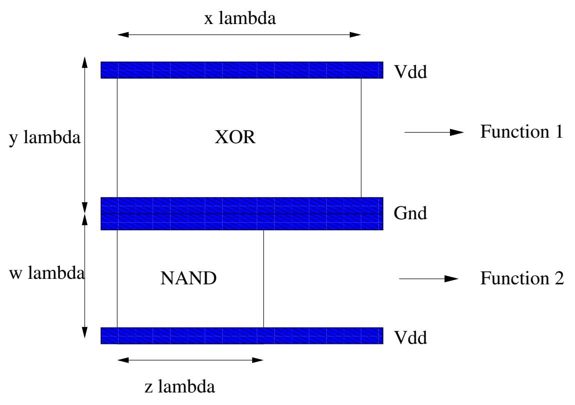

Figure 1: Sample Floorplan.

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: XOR Truth table.

2 Complex Gates

Normally, an XOR gate can be formed by using other functions such as from AND and OR gates. However, we can use a little trick to help us make the XOR and subsequently, the full adder more robust.

If we look at the basic equations for an xor gate, it is:

$$f_{xor} = A \oplus B$$

With some simple applications of DeMorgan's Theorem, the xor gate can be broken into the following:

$$\begin{aligned} f_{xor} &= \overline{(A \cdot B)} \cdot \overline{(\overline{A} \cdot \overline{B})} \\ &= (\overline{A} + \overline{B}) \cdot (A + B) \\ &= \overline{A} \cdot B + A \cdot \overline{B} \end{aligned}$$

This can be easily be built in CMOS using the techniques we utilized during the first part of the semester.

2.1 Mirror Circuits

The XOR circuit is one of the most beneficial circuits to design engineers. Not only is it vital to adder circuits, but it plays a key role in cryptographic circuits, such as parity generation. Therefore, anything that can be done to speed up this circuits is monumental.

One method of making the XOR circuit more compact and quicker is to use the *mirror circuit*. The mirror circuit is basically where the NMOS and the PMOS gates have exactly the same structure. However, they do not use series-parallel, but use many of the same characteristics (i.e. the mirror does not have any parallel structures except at the folding point).

The origin of the mirror circuit comes from the truth table of the circuit to be designed. For the XOR gate, it is easy to see that it appears the truth table is flipped as shown in Table 1. In order to construct the mirror XOR gate, it is necessary to remove the parallel structure for the PUN group of $A \cdot B$. This is achieved by removing the parallel structure normally found in the XOR gate (i.e. $(\overline{A} \cdot \overline{B}) + (A \cdot B)$) and rewriting the equation. The circuit is shown in Figure 2.

It is important to note that although the basic element of an XOR gate is built-using mirror circuits, that it still requires inverters. This is a fundamental problem with most circuit styles today and it appeals strongly to static CMOS design. Other circuit styles such as dynamic CMOS can not easily make an inverted signal, since dynamic CMOS is non-inverting. Therefore, although dynamic circuits tend to be much faster than static CMOS circuits, they still need inverters. Most dynamic logic styles in CMOS still utilize the static CMOS inverter due to is ability to create easy inversions and PMOS' ability to be the "perfect" load.

2.2 Full Adder

The full adder is the basic cell for most adder implementations. The basic equations for addition using the full adder leaf cell are with the following Boolean equations:

$$\begin{aligned} Sum &= A \oplus B \oplus Cin \\ Cout &= A \cdot B + Cin \cdot (A + B) \end{aligned}$$

Because the full adder is basically counting whether or not one value is above a certain level, the full adder is typically called a $(3, 2)$ counter. This is because there are 3 inputs and 2 outputs.

Similar to the XOR gate, the full adder can utilize a mirror structure as seen by the full adder's truth table shown in Table 2. Although *Cin* does not appear to have any mirrored structure, it is clearly evident that *Sum* does.

The Full Adder is made from the 28 transistors. You will continue to use hierarchy in your designs, therefore, using the cells used in previous assignments is useful. You will also add specific logic to perform either a logic addition or subtraction. Subtraction is

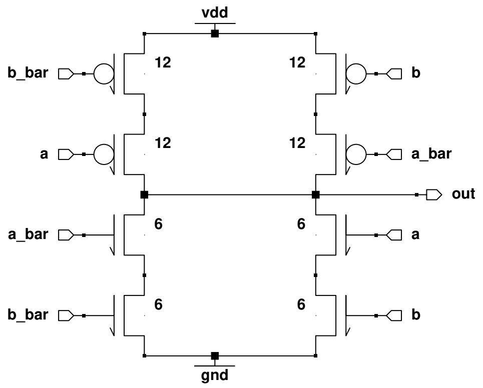

Figure 2: XOR Function Using Mirror Circuits.

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2: Half Adder Truth table.

the same as addition except we add the negative version of a given operand. That is, instead of adding a + b, the operation is able to subtract b from a by complementing b (i.e. $a + (-b)$). Since this project will utilize two's complement addition/subtraction, your project must be able to add the magic one or unit in the last position (ulp), because it is a radix-complement of the binary system. Therefore, the adder will calculate $a + \overline{b} + ulp$ [2]. An ulp is an acronym that signifies the unit in the last place [3]. This terminology is the correct method of describing adding one to the number, since we do not know if the number is an integer or a fraction. For the remainder of this project definition, the phrase ulp will be utilized.

The layout for the full adder can be made rather easily using XOR gates. In order to accomplish this you want to bit slice the adder design for the input size you require. Careful attention to the full adder cell will allow you to slice the cell while still allowing a certain abutment and minimizing space. Your finished adder/subtractor should have its inputs and outputs routed so that it will be pitch matched with successive Full Adders in creation of a Ripple Carry Adder or Carry Propagate Adder (CPA). The operands, A and B, should enter at the top of the cell and Sum should exit at the bottom. The carry in or Cin should enter from the side and carry out or Cout should exit to the other side. Cin and Cout should cross the borders of the cell at the same position and be of the same material, preferably metal. This will allow multiple cells to be stacked next to each other, with Cout feeding Cin of the next cell.

2.3 Registers

Building registers in one of most crucial elements that exists in any digital system design. In fact, successful products need good use of reading and writing digital quantities. This is because they are crucial to storing data and re-using it for future use. Therefore, having efficient, low-latency, minimal area design is important for any design. In this section, you will implement a register that will follow your adder.

To make things easier, you will be implementing a complete D flip-flop, similar to the Full Adder design. Although hierarchical designs can sometimes be better for more structured gates, some gates, such as the D-flip-flop, are better when they are structured as an individual gate. Ultimately, circuit-level simulation is best suited to adequately determine the objective in which to optimize a gate.

3 Main Project

This project's task is straigh forward and two-folded. The main task is for you to understand the importance of bit-slicing and the second is to work on integrating many of the ideas you had throughout the semester into one package or learning experience. The bad part about this is that it will involve you to task your time throughout the semester or you will not have enough time to complete this project. Therefore, please pay attention to time milestones when working on this project.

This project will be a 4-bit/8-bit sequential multiplier. Binary multipliers take in two n-bit operands and produce a $2 \cdot n$-bit product. Since multiplication is an important operation for general-purpose computing, it is typically included as a hardware element within most computer architectures [4]. The trick is making the hardware fast as it often time consuming to compute. For this project, you will be implementing a sequential multiplier that uses a clock and produces the answer n cycles later. For example, a 4-bit sequential binary multiplier takes 4 cycles to complete. In order to get your hardware going, 1 additional cycle is needed to preload the mulitplicand into a register. Consequently, your n-bit sequentail multiplier will take 1 additional cycle or $n + 1$ cycles to complete. At the end of computation, your $2 \cdot n$ product will be produced from the final output register.

Two forms of the project are available. Individuals wishing to work alone can work on the 4-bit version. However, groups of 2 teams on the 8-bit version. To simplify the process, all schematics will be provided to help groups get started quickly.

The bitslicing custom-cell part of the project will involve a portion of a 4 or 8-bit datapath design, specifically the sequential multiplier. The block diagram for the design along with the bitslice is shown in Figure 3. Please note that it important to understand that this design can take days or even months if floorplanning is not envisioned correctly. Therefore, proper floorplanning and design structuring of a 1-bit version of certain blocks and then bitslicing it for 4 or 8 bits will make this portion of the project easy and simple. Ignoring my warning about floorplaning, could result in endless night/day sessions of grunt work in layout without any success. Most importantly, the art and science of bit slicing is such a valuable skill that it is one of my major hopes that you understand the importance of bitslicing in this class. It is also how companies like Intel and AMD make large VLSI designs and still make their deadlines for release.

3.1 Timing

Custom-level parts are optimized for area, energy and, more importantly, for delay. Today, things have changed where some delay optimization is exchanged for optimizing layout to conserve power dissipation. Although power is an essential element to most VLSI designs today, the aspects of optimizing for power is somewhat similar to that of delay. Therefore, for this project we will concentrate on just delay-based optimizations. To accomplish our timing measurements, Synopsys `hspice` will be our primary analysis tool. As we saw from our earlier use of `hspice`, sometimes it is extremely difficult to measure values on plots within Waveform Viewer. To make things easier, we will take advantage of the power that engineers wield within systems. That is, engineers are highly adept at taking advantage of making things easier and one way VLSI engineers do this is using scripts.

Scripts are just efficient way to get answers with software by doing tasks that take long times with the `hspice` software. `hspice` uses these scripts by employing something called `.measure` statements. The `.measure` statements have several different format to allow different measurements, but the basic elements are basically the same. However, the central item is to use ingenuity to get answers more efficiently. Since there are several variety of **.measure** statements within hspice, please follow the tutorial on website `http://vlsiarch.ecen.okstate.edu/wiki` for measuring the critical path. Although these statements will aid you in finding the propagation delay of your circuit, you will still have to think about what vectors are necessary to have your circuit behave in the worst-possible manner. As with the homework, do not trust the `.measure` statements without checking them on your SPICE waveform output first.

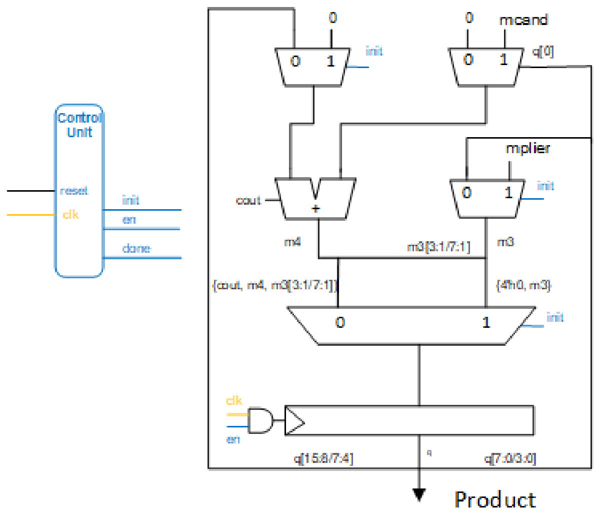

Figure 3: Block Diagram of Unit.

4 Verification

Many designs today require a substantial effort with verifying a design is correct. In fact, I have heard estimates from some companies that 60% of effort in producing a design is due to verification. For this project, verification will be just as important. Therefore, it is important to take the time to make sure your output is correct.

Many verification programs use something called a *golden file*. A golden file is usually a plain or ASCII file that contains true answers to your results. This golden file is usually created with some program (e.g., C or MATLAB) and written into a file. The golden file is then compared to the output produced by simulating your design to verify whether its operation is correct.

Although SPICE decks can produce good output, they typically take long times to run. Therefore, engineers want easy ways to verify a design quickly and accurately. Switch-level simulation is one method engineers utilize to make sure simulations complete quickly. Switch-level simulation basically reproduces transistors as simple switching mechanisms. Then, programs use event-driven simulation to produce output to verify the golden file.

For this project, a sample IRSIM file will be provided that produces a golden file. Although the IRSIM example is a good example, its only 1 example and to make sure your design is working properly, you should probably run a couple of vectors. Output should be checked for any errors and problems can debugged for validating whether your datapath is correct. The `assert` keyword in IRSIM can be utilized to compare the observed output versus the golden file.

4.1 System on Chip Design

Most digital systems today are composed of a variety of custom and standard-cell designs. These are typically System-on-Chip (SoC) designs as they are comprised of a variety of systems. In order to give you an idea about this process, your design will have both a custom and standard-cell design portions. Most standard-cell designs are done through the use of some Hardware Descriptive Lanaguage (HDL). For our design, we will utilize a top-level design that will look like the following in Figure 4. For the HDL in Figure 4, notice the clock is utilized to synchronize both systems. The custom-cell portion of your design, which is where you will spend most of your time, is the `mult` item in Figure 4.

The standard-cell design portion will be basically a Finite State Machine (FSM). The FSM for this design is quite simple and just makes sure thing work once the multiplication starts. The state diagram for the 8-bit FSM is shown in Figure 5. The 4-bit version is basically the same except that you can remove states `S5` through `S8`.

In order for you to make sure your complete design works, you should simulate the Verilog HDL. HDL is just another way to make a schematic albeit in a textual form. Its also convenient for designers to make sure their design is correctly designed as schematics are harder to implement. In your `verilog` directory

```
module top (input [3:0] mcand, mplier,
            input logic clk, reset,
            output logic [7:0] q);

   logic                     init, en;

   fsm control (done, reset, init, en, ~clk);
   mult datapath (mcand, mplier, clk, init, en, q);

endmodule // top
```

Figure 4: Top Verilog SoC Design

of your class project directory is a Verilog model. Everything is present in that directory except your FSM. You should implement the FSM per Figure 5 and the design size you are implementing (e.g., 4 bits). Then, to simulate, just run the following using Mentor Graphics ModelSim which is a HDL simulator. The command to run is the following using the 4-bit design as an example:

```
vsim -do smult4.do
```

Even if you have never used Verilog extensively, the simulation window should open and present the final answer for the `Q` output.

5 Extra Credit

For that may be looking for some extra credit, the datpath we are designing can be significantly modified for extra functionality. You can add this for some extra credit along with other potential enhancements. Some easy elements you can add that are relatively easy to add are:

1. Signed Two's Complement Multiplier
2. An extra Verilog standard-cell item (you will need to add control to handle both units – i.e., the multiplier and your additonal items with some additional control)
3. Power/Energy analysis of your system
4. IRSIM simulation through your chip pad
5. Signing up for fabrication

6 Design Considerations and Constraints

There are many facets to this project, but there are also plenty of opportunities for you to inject your own personal element to this project. Consequently, you can earn extra points by adding extra features, more custom-level logic, or just enhancements that you think might be useful. The following requirements must be met for your design:

- Start NOW! Do not wait - you will not be able to complete the project if you wait, **guaranteed**.
- There will be a prize for the best project! Let's see what you can do in this course and what you have learned.
- Your chip must fit within a MOSIS "tiny-chip" form factor, which means that overall chip including pads will be 2.25 mm^2 despite having approximately only 0.9 $mm \times 0.9$ mm (3000 $\lambda \times$ 3000 λ) of allowable space to use. This means that if your layout is too big, you could possibly get deductions for not making your layout compact enough. In other words, your total layout size should fit within this area.
- You must use a single cell for the full-adder (FA) and the flip-flop. I would use the schematics that were given to you for these parts. You must **not** use multiple gates to create the full adder (e.g., XOR and OR gates). It **must** be a single cell, such as the mirror FA circuit. Points will be heavily deducted if you decide to not use the gates we provided for these parts. You are welcome to use different circuits, but they must be approved in writing (e.g., E-mail) by the instructor.
- To minimize the gate resistance for each transistor and as a recommendation from MOSIS and ON semiconductor (i.e., our fabrication facility that will fabricate successful projects), you should use a *minimum* gate width of 3.0 $\mu m = 10 \cdot \lambda$ for any nMOS/pMOS transistor.

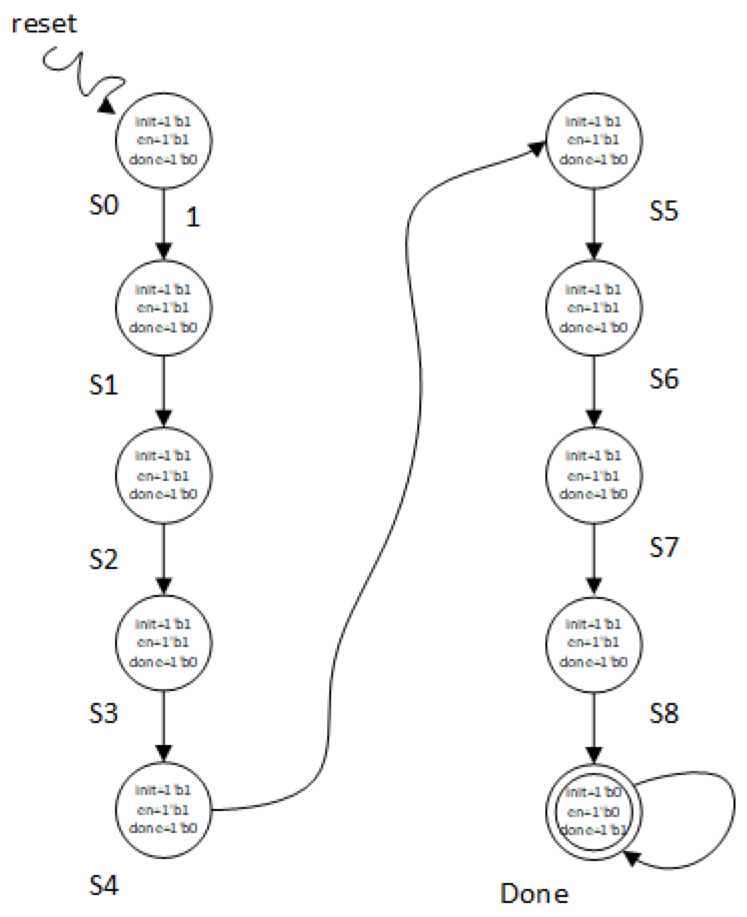

Figure 5: Finite State Machine (FSM) Diagram.

- The custom-cell datapath project should be completed first and is composed of multiplexors, a carry-propagate adder (CPA), and a register with a gated clock. You can add more elements to the custom-cell.
- Make sure you spend some time stick diagramming your bitslice and, hopefully, thinking about the overall floorplan of your design. Unfortunately, you are not going to be perfect with your layout, so you will get some space, but you should spend time making sure you do your best to floorplan this structure well (i.e., with limited space between polygons).
- Although any CPA will work, it is far easier to incorporate a ripple-carry adder as your adder.
- Many of the schematics and some of the layouts completed in previous assignments (as well as some gifts) are available at:

  ```
  /classes/ecen4303F18/project_F18
  ```

  Feel free to use these elements for your project. Or, if you wish to use the layouts you previously designed, this is highly encouraged.
- You can work in teams of two, however **NO** sharing of files are allowed. I can determine if you use another person's file, so be careful not to overstep your bounds on academic integrity. If you work in teams of two, a page **must** be added to your report indicating what you worked on and the time spent. **A** 10% **deduction will occur for your project if you forget this item!**
- Your chip should be designed for operation at 75 MHz. In order to determine the proper speed, you should make sure you correctly identify the critical path and use SPICE to determine its propagation delays. Moreover, to receive full

credit for the project, your design must be fully simulated in IRSIM to show correct operation.

- You must accurately measure the delay of your device for its critical path. Pretend you are passing this block off to another group that intends to fit your design into their larger design, therefore, having good numbers for area and delay are important. If you wish to compute the power dissipation, you can perform this for extra credit, as well. A table in your final report should give these values as a summary of your final design. A final figure of your layout through *Pplot* file output can easily be inserted in Word or TeX files.
- You should have a complete design, including all connections. Your entire design should be thoroughly simulated with IRSIM to check for correctness and Synopsys' `hspice` for timing. You should hand in all design work created in previous steps, the IRSIM input vectors you used to test your circuit, and a report describing what you did, what problems you encountered, what you learned.
- You should provide some sample vectors, so you can use this to test whether your project actually works correctly. This is similar to how companies work in that they provide a "golden file" of correct outputs and you must match that output through your IRSIM output. Use the provided test template to help you generate good test vectors.
- For more information about the final report, see the final report guidelines published on D2L.
- Do **NOT** attempt extra credit without getting the baseline project done first! This is a really bad way of wasting time and not completing the assignment on time.
- Feel free to contact me if you have questions or just want to check to see what you are attempting is possible or if you want to see if you are on track to complete the project safely before the due date. We are here to help you!
- Start NOW! The project is due December 7, 2018 at 11:59:59 PM **NO EXCEPTIONS!!!!!!. Projects must be handed in to ES202 or uploaded to D2L**. Layouts will also be required and a script will be provided that will copy your layouts for submission. It is advisable to keep a good file structure to make your project easier. If you get stuck and things just do not work, document what and what does not work and remember to hand something in.
- Most importantly, have fun as this is a great experience that not many have. Enjoy!

References

[1] N. H. E. Weste and D. M. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th Edition, Addison-Wesley, 2011.

[2] I. Koren, Computer Arithmetic and Algorithms, A. K. Peters/CRC Press, 2002.

[3] M. D. Ercegovac and T. Lang, Digital Arithmetic, Morgan Kaufmann Publishers, 2003.

[4] David A. Patterson and John L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2007.