ECE115C – Digital Electronic Circuits
Class Project

*Due **Wed, March 11th @ 8pm***
(submission instruction will be provided)

Optimal 1.29GHz 6-Bit "Absolute-value Detector"
for use in Neural Spike Sorting

Problem Description

The goal of this project is to design a **6-bit "Absolute-value Detector"** with the **minimum energy and worst-case delay of 775ps**. Here "delay" refers to the worst-case propagation delay and "energy" refers to total energy drawn from V_{DD} given specified input probability distribution. You may use **gate sizing** and **supply voltage scaling** as variables. No registers (i.e. pipelining) are allowed in the design of this project. Work in a **group of 3 students**. If you need help finding a partner, please let the Instructor or TA know.

You will find *appendices* that explain what an "Absolute-value Detector" is and other references you might find helpful at the end of this document. Please read them carefully.

For systematic approach, you can organize your work in three phases. In **phase-1**, use the design expertise you acquired in class to find the optimum architecture that best optimizes the speed-energy goal. Do a quick sketch of several feasible options and figure out the best architecture and circuit style. You may mix circuit styles if that helps. In **phase-2**, first implement the block-level schematic of your Absolute-value Detector and verify the functionality in Spectre. Then, identify critical path and optimize sizing for minimum delay. In the critical path evaluation, you need to determine not only the gates along the path, but also the input operands that cause worst-case delay between input and output bits. In **phase-3**, refine your rough layout sketch from phase-1 and layout your design starting with basic building blocks. Area is defined as the smallest rectangular bounding box a design can fit in. Aspect ratio of the box (long / short side) should be less than 1.5. Below is more detailed explanation of the steps you need to take to ensure the success of your project.

Phase 1: Choosing Topology / Circuit Style **(1 week)**

a) Determine circuit topology that optimizes delay-energy metric.
b) Choose logic style for the implementation. You may mix several logic families.
c) Implement the schematic view of the Absolute-value Detector in Cadence and move on to phase-2.

Phase 2: Critical Path Delay Optimization & V_{DD} Scaling **(1 week)**

a) Check functionality of your design in Spectre.
b) Identify input vectors that will exercise critical path. Size the gates for minimum delay.
c) Verify the critical path delay in Spectre under worst-case input operands.
d) Consider changing V_{DD} (within range 0~1V) to achieve minimum energy while meeting the delay requirement.

Phase 3: Layout and Verification in Spectre **(1 week)**

a) Create layout of the design and make sure it passes DRC and LVS.
b) Extract post-layout netlist and verify critical path in Spectre.
c) Submit pre-layout and post-layout netlists. We will run LVS on your design, and run worse-case test-bench to verify reported critical-path delay.

Prepare Final Presentation (Thursday, March 12th, in-class) **(1/2 week)**

Prepare a **6-slide presentation** (template to be provided soon) representing your effort. Sign up for a presentation slot on wiki. Present your results to the Instructor and TA. Be crisp: highlight your main design decisions, explain why they are the best thing in the world, and prove that they really worked out (or did not). Be brief since you only have **5 minutes** to present. You may write a 1-page report in text to elaborate your idea and attach it to the slides.

Sleep **(March 11th into 12th, 8 hours)**

The purpose of this task is to ensure that you will be able to clearly present your work! The goal is to get through few REM cycles and come to your presentation relaxed.

Constraints (READ CAREFULLY!)

a) Supply voltage: find optimal V_{DD} (0~1V) that meets the delay (775ps) and minimizes energy.

b) Implementation choices:

i. Use only static logic (CMOS, pass-transistor logic).

c) Input operands:

i. Input neural signal is a 6-bit number represented in 2's complement format. Threshold value (5-bit binary) is fixed and will be given to you in the final testbench.

ii. Assume each bit of the input has equal probability of being 0 or 1 and the bits are mutually independent.

d) Loading conditions:

i. The input capacitance of all inputs (A_0-A_5) is less than equal to 2 unit sized inverters (see below for the definition of unit-sized inverter). For simulation purposes, the inputs to your adder are driven by a unit sized buffer (chain of two unit sized inverters). The delay is measured as the delay after the input driver (2 inverters) to before the load (32 times unit sized inverters). **Test-circuit will be provided. (You also need to test your circuit before testbench is provided to ensure full functionality).**

ii. Output bit is loaded with C_L = 32 unit sized inverters. This load will be implemented with inverters (test-bench coming soon).

iii. Unit sized inverter is Wp = 650nm, Wn = 430nm, Lp = Ln = 100nm (drawn L).

e) Layout constraints:

i. Minimum width of V_{DD}/G_{ND} rails is 0.36µm.

ii. You can use up to 5 metal layers.

iii. Aspect ratio of the bounding box (long / short side) should be less than 1.5.

Appendix A – Absolute-value Detector

Spike-sorting algorithms have gained a tremendous interest in the research community with the recent advancements in neural signal acquisition systems. For your ECE115C project this quarter, you are to design one of the commonly used spike-detection algorithms, named absolute-value detection.

Figure 1 shows the basic diagram for an Absolute-value detector that needs to be designed for your project. The inputs (shown in blue) are given to you (See the constraints for more detail). The Absolute-value detector (shown in black) is to be designed and implemented by you.

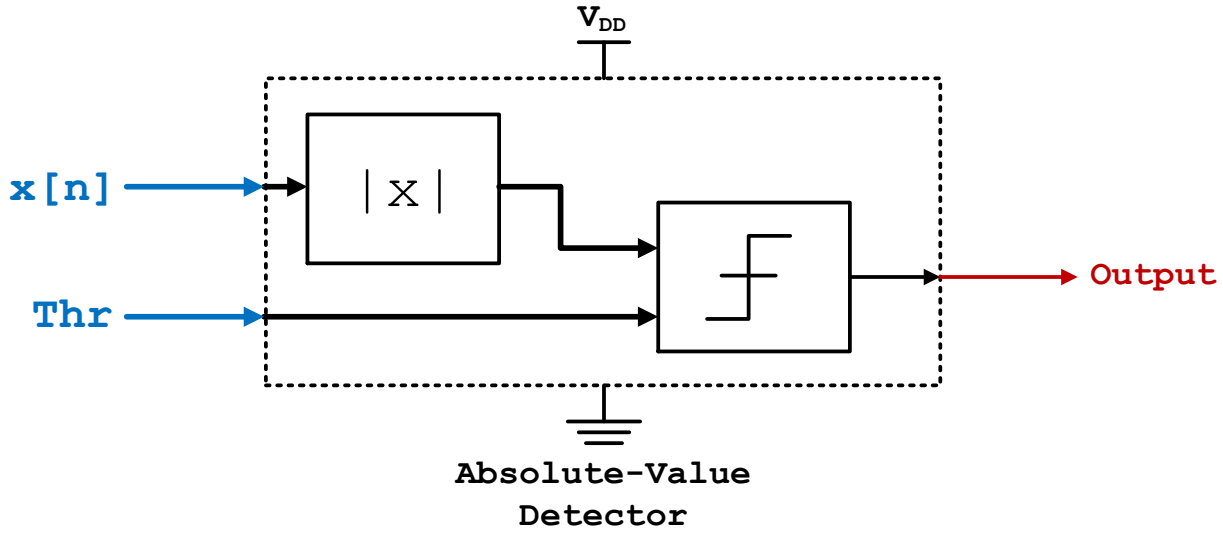

Figure 1

As shown above there are two main components to the absolute-value detection.

(i) finding the magnitude (absolute value) of your neural signal (`x[n]`) and
(ii) comparing the magnitude to the given threshold value (`Thr`).

If the magnitude of your signal is greater than the threshold output should display a "`1`" (high logic value), otherwise the output should be a "`0`" (low logic value).

Appendices B and C provide some background and references for the design of absolute-value and comparator blocks respectively.

Appendix B – Magnitude of a 2's Complement Number

Your input signal, `x[n]`, is given in a 2's complement representation. You should be familiar with 2's complement representation of binary numbers. If you are not, it is strongly suggested you review your EEM16 course material, see the TA, or refer to any of the following references:

1. "Digital Design" by M. Morris Mano.
2. "Digital Design: A Systems Approach" by William J. Dally and R. Curtis Harting.

Your input values can range from `-31` to `+31` and will be given to you in 2's complement format.

- Although using 6-bits -32 can also be represented in 2's complement, you can assume this will never be given as an input. This way your magnitude will always be 5-bits which is compared to the given 5-bit threshold value in your comparator.

Few reminders about 2's complement representation of numbers:

- If the most significant bit is a "`0`", the number is positive → Magnitude is the same as the number given to you.
  - Example 1:
    `+28` represented in 6-bit 2's complement format is `011100`.
    It's 5-bit magnitude is `11100`.
  - Example 2:
    `+5` represented in 6-bit 2's complement format is `000101`.
    It's 5-bit magnitude is `00101`.

- If the most significant bit is a "`1`", the number is negative → Magnitude is found by flipping (inverting) all bits and adding a `1`.
  - Example 3:
    `-28` represented in 6-bit 2's complement format is `100100`.
    To fine its magnitude we do the following:

```
  100100     (-28)

  011011     (bits are flipped)
+ 000001     (add 1)
  ------
  011100     (Magnitude – You can ignore the 6th bit)
```

    It's 5-bit magnitude is `11100`.
  - Example 4:
    `-5` represented in 6-bit 2's complement format is `111011`.
    To fine its magnitude we do the following:

```
  111011     (-5)

  000100     (bits are flipped)
+ 000001     (add 1)
  ------
  000101     (Magnitude – You can ignore the 6th bit)
```

    It's 5-bit magnitude is `00101`.

As you noticed you need an adder for this block. Adder designs will be covered in class and more information can be found in chapter 11.3 of the textbook. **It is important**, however, to note that you always add a 1 to your numbers. This fact can allow you to **significantly reduce the complexity of your design** and result in a much **more optimized logic**.

Appendix C – Comparator

One you have found the 5-bit magnitude of your signal you will need to compare its value with the given threshold. If the magnitude of your signal is greater than the threshold output should display a "1" (high logic value), otherwise the output should be a "0" (low logic value).

The strategy here is compare the most significant bits of the two number. If one is greater than the other, we know that number is greater. If they are equal we will move to the second most significant bit, and so on.

For your reference a 4-bit Magnitude Comparator is shown in Figure 2.

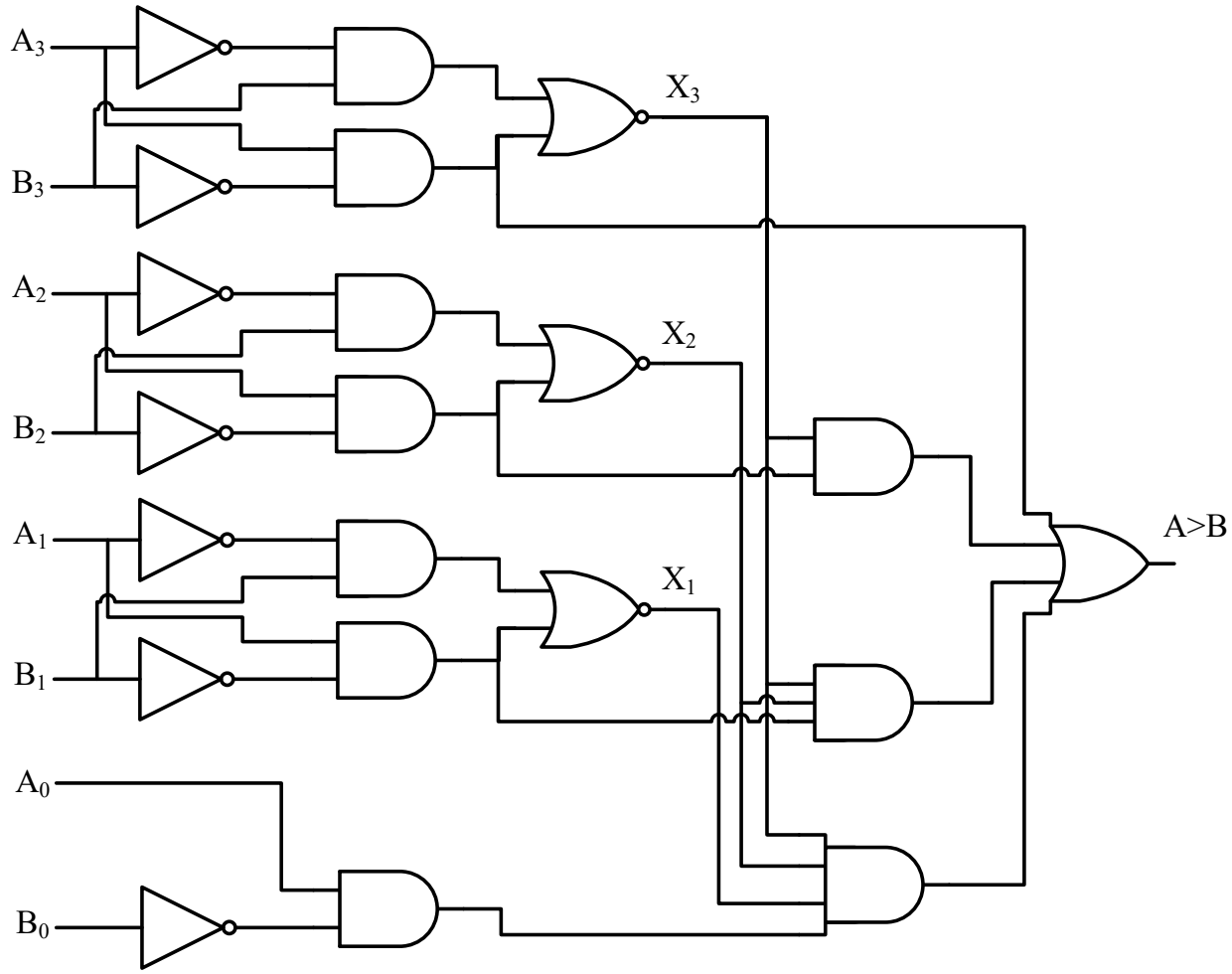

Figure 2

Here: $(A > B) = A_3\overline{B_3} + X_3A_2\overline{B_2} + X_3X_2A_1\overline{B_1} + X_3X_2X_1A_0\overline{B_0}$, where $X_i = A_iB_i + \overline{A_i}\,\overline{B_i}$

HAVE FUN!