Combinational Logic Design II— A Simple Calculator

You will learn how to use hierarchy and busses to realize a modular design of a simple datapath.

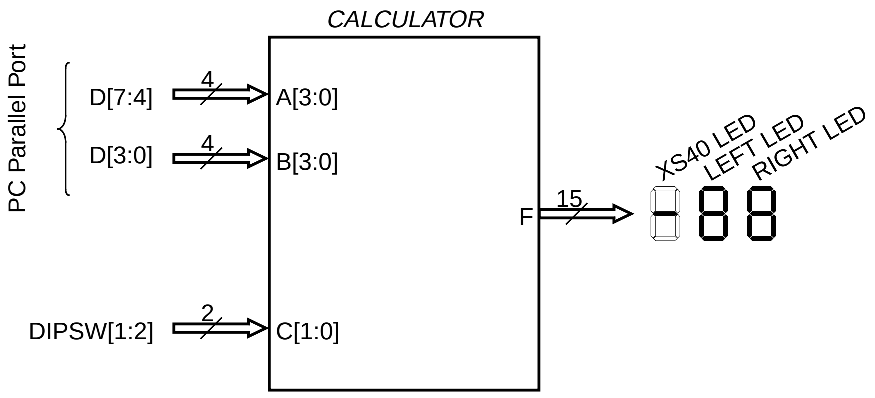
1.0 Overview

In this experiment you will learn about modular design of combinational circuits. The type of circuit you'll be designing lends itself very naturally to this style of design: it is a datapath whose structure, or architecture, is typically determined by the types of operations it is required to perform. In addition, datapath circuit signals come in two distinct varieties: data and control. Control signals determine where to send, or route, the data signals and what operations to perform on them. Data signals, on the other hand, are the primary carriers of information in the circuit, and are typically bundled as multi-bit words. Datapath circuits tend to be quite regular, allowing the use of a structured design approach that simplifies the design process and leads to easily testable implementations.

2.0 Preparation

Now is a good time to review two's complement arithmetic from Chapter 2 of Wakerly. Also, review the design of adders in Sec. 5.10. You should consult (i.e. carefully read!) the on-line documentation of the *Xilinx Foundation Project Manager* and the Xilinx tutorial to understand how to create macros, how to use busses, etc.

FIGURE 5. Top-level schematic of *CALCULATOR*



3.0 Design Specification

The top-level schematic of the *CALCULATOR* circuit is shown in Figure 5. It has a 2-bit *opcode C* and two 4-bit operands *A* and *B* that represent two's complement integers. The output *F* must be generated according to the following *function table*:

$C1$	$C0$	F	Description
0	0	$A + B$	Add *B* to *A*
0	1	$A - B$	Subtract *B* from *A*
1	0	$\lvert A \rvert$	Absolute value of *A*
1	1	$\lvert B \rvert$	Absolute value of *B*

The *A* and *B* operands should be connected, respectively, to the most and least significant nibbles[1] of the PC parallel port. The opcode bits should be controlled by the two left most DIP Switches. The *CALCULATOR* output *F* should be displayed in decimal signed-magnitude notation using the three 7-segment LEDs: the XS40 LED to display a negative sign (segment S3) for negative results, and the LEFT and RIGHT LEDs to display the 2-digit decimal magnitude.

4.0 Design Notes and Hints

This problem is best approached by analyzing the common features among the various data operations (addition, subtraction, and absolute value) and designing an efficient multi-function datapath unit that can perform each of those operations in response to the applied opcodes. The core of the datapath must be an adder/subtractor (which you should be able to use to compute absolute value) with appropriately-controlled multi-

1. A nibble is 4 bits.

plexers on its inputs. You will also need to design a code converter to display the two's complement result on the 7-segment LEDs.

This circuit can be designed "flat" by either drawing a single gate-level schematic or writing a single ABEL module. You'll be a lot more productive, though, if you use hierarchy to make your design modular; hierarchy can help you speed up both design entry and design verification. Components that you create are referred to as *macros* and are kept in a separate user library bearing your project's name. You may find some high-level components in the system library that already perform a function that is identical or very similar to ones you're considering for your circuit. You may use such components without having to create your own versions from scratch. Note, though, that you may have to augment a library component to make it conform to your specific design needs. **What you're not allowed to do is to design this circuit as an ABEL module that uses arithmetic operators; at this stage in your training as a logic designer, we want you to understand what's behind that innocent-looking plus sign.**

Another time-saving device is the use of busses to combine related single-bit signals. Working with busses requires some skill, but the effort you put into it now will more than pay for itself later when you design even more complex circuits.

The above hints should probably be sufficient for you to go ahead and start working on your design. However, to save you even more time, without giving away the store, consider the following additional clues:

1. A binary adder can be easily adapted to perform both addition and subtraction of two's complement numbers by noting that $A - B = A + B' + 1$ where A and B are n-bit words, and -, +, and $'$ refer, respectively, to subtraction, addition, and bit-wise complementation.
2. An XOR gate $Y = S \oplus A = S'A + SA'$ acts as a multiplexer that generates either A or A' depending on whether the select input S is, respectively, 0 or 1. It is handy for realizing conditional complementation of a signal.
3. Even though the input operands for our circuit are 4 bits wide, the range of the results cannot be represented in 4 bits. That is in fact why we need two decimal digits (and a sign) to display them. The implication of this is that the internal datapath must be wider than 4 bits; you need to determine how much wider and how to extend the 4-bit input operands to match this width.
4. Truth tables are ideal for such things as code converters. Consider writing ABEL macros, rather than gate netlists, for translating the two's complement result of your circuit to LED segment activation signals. This will be easier to specify and debug than a seemingly random interconnection of gates (whose design is not very inspiring either).

5.0 Deliverables

5.1 Pre-Lab (25 Points)

1. Hardcopy of *CALCULATOR*'s schematic, ABEL, and UCF files.

2. Hardcopy of simulation results for the scenarios shown in the following table:

	Scenario A	Scenario B	Scenario C
Signal	**Stimulator**	**Stimulator**	**Stimulator**
C1..C0	00 (A + B)	01 (A - B)	10 (|A|)
A3..A0	Bc3..Bc0	Bc3..Bc0	Bc3..Bc0
B3..B0	0011	0110	0101

Display simulation traces for each of these scenarios showing the resulting waveforms on the “sum” outputs of your adder; you may optionally show the waveforms on the outputs of your two’s complement-to-signed- magnitude decoder for easier inspection of the results.

5.2 In-Lab (25 Points)

Generate the .bit file for *CALCULATOR*, download it to the XESS board, and verify that your implementation is working properly. When you are satisfied that you have a correctly-functioning circuit, demonstrate its operation to your lab GSI and have him sign your experiment’s cover sheet.

5.3 Post-Lab (Corrected pre-lab: 15 Points; Lab report: 35 Points)

Prepare your lab report as described in the *EECS270 Laboratory Overview* handout. Make sure you complete and include all parts of the report including the *Cover Sheet*, the *Design Narrative* section, and the *Design Documentation* section. Include as part of your design documentation all corrected pre-lab requirements. In your design narrative, provide enough detail about the choices you made to realize your particular implementation of this circuit. Specifically, document how you decomposed the design and why you chose that particular decomposition. Also, note any difficulties you encountered and how you resolved them.