CPEN 211 Lab Proficiency Test #3

Start time: **15:40**. End time: **17:30**. No late submissions. Sign out before leaving.

TL;DR

In this LPT will design the **control logic** to control an existing datapath of a two-function calculator.

The datapath is in `calc_dp.sv`, a template for remaining the RTL you need to write is in `calc.sv`, and a very rudimentary testbench to get you started is in `tb_calc.sv`.

You only submit the modified `calc.sv`, via Canvas like in LPT 1 and LPT 2. You **must sign out** with a TA before you leave the exam room.

SPECIFICATION

The calculator has an output display, as well as 14 buttons: `RESET`, `+`, `-`, `=`, and buttons for digits `0` through `9`. The user can only input **single digits** (e.g., input can't be `42`), but the result can be multiple digits (e.g., `9+8` → `17`). There is no way to enter negative numbers.

Interface

The interface ports are defined in the provided template `calc.sv`. Pressing the various calculator buttons result in the following inputs on the ports:

- The `RESET` button asserts `rst_n` for one or more clock cycles. Reset is **active-low**.
- Each press of any of the remaining input buttons is encoded as an *instruction*, presented for exactly **one cycle per button press** on the input `instr`.
- When no button is being pressed, the `instr` input is zero (no instruction).
- The instruction encoding is as follows:

button pressed	instruction encoding
none	`00000`
`+`	`00010`
`-`	`00011`
`=`	`00001`
`0` — `9`	`1nnnn`, where *nnnn* is the unsigned binary encoding of *d* (e.g., `9` → `11001`)

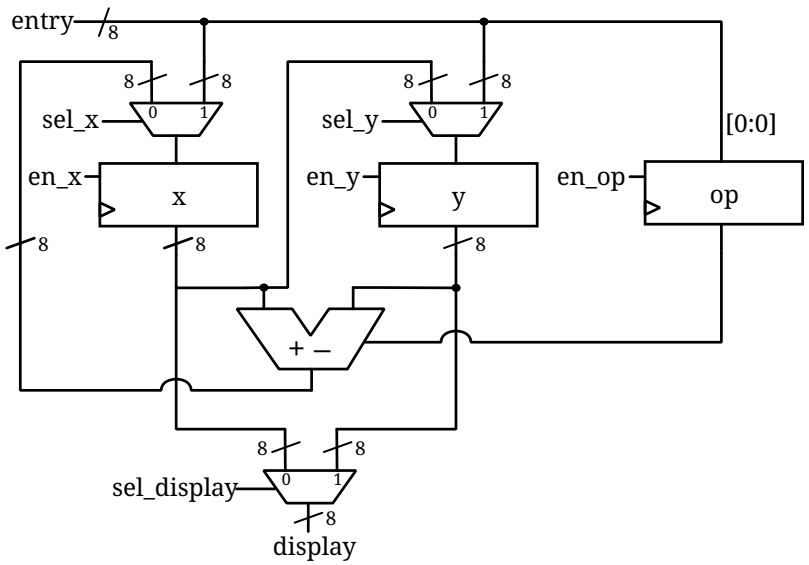
- Behaviour on any input that does not correspond to the encodings above while reset is not asserted is **undefined**, and will not be tested.

Datapath description

The datapath has already been implemented in `calc_dp.sv`, and is shown on the next page.

Registers `x` and `y` store the operands entered, respectively, before and after the arithmetic function (`+` or `-`) has been entered; register `x` also stores the operation result. Register `op` identifies the desired arithmetic function. All registers have a corresponding enable signal. The `display` output can be selected to show either `x` or `y`.

The adder/subtractor operates in two's complement; its function is determined by the value in `op` (0 → `x+y`, 1 → `x-y`). Arithmetic wraps around on overflow and underflow: for example, `0-1` → `255`.



Functionality

The calculator has two input modes:

- **first-operand mode**, when inputs go into x and x is displayed, and
- **second-operand mode**, when inputs go into y and y is displayed.

The calculator starts in first-operand mode, enters second-operand mode on `+` or `-`, and goes back to first-operand mode when `=` is pressed. For example, for input `1` `+` `2` `=`, this causes 1 to go into x, 2 to go into y, and the result to be put into x.

The behaviour is specified as follows:

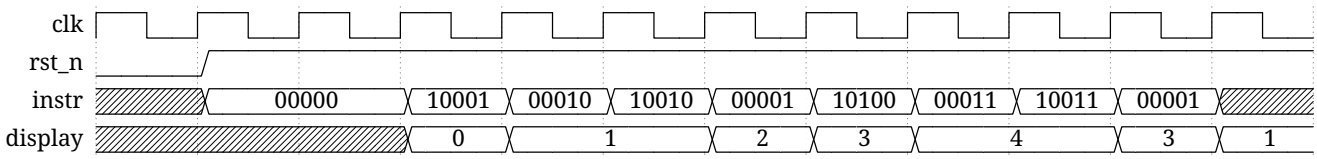
- At most two clock cycles after 0, the display shows 0, and the calculator is in first-operand mode.
- The display output shows:
  - the value of x in first-operand mode;
  - the value of y in second-operand mode.
- When a digit is entered, it is:
  - written to x in first-operand mode;
  - written to y in second-operand mode.
- When a digit is entered immediately after another digit, it overwrites the previous digit.
- When either `+` or `-` is pressed:
  - the desired function is stored in op (0 for add, 1 for subtract);
  - the value in x is copied into y;
  - the calculator enters second-operand mode.
- When `=` is pressed:
  - the arithmetic operation result (x+y or x-y) in two's complement is written into x, with the result truncated to 8 bits in case of over/underflow.
  - the calculator enters first-operand mode.

Timing

For full credit, you must meet the following timing criteria:

- The first instruction may be entered **two clock cycles** after reset is deasserted, or at any later time.
- Subsequent instructions may be entered in **adjacent clock cycles**, or at any later time.
- Multiple instruction sequences may be entered back-to-back without intervening no-instruction cycles.

The timing diagram below illustrates these requirements:



If you cannot meet this timing requirement, you may implement a variant that requires an additional clock cycle with no button pressed both after reset and between any pair of instructions; you will receive **partial credit** for this variant.

Examples

The examples below show the state *after* the following buttons have been pressed:

button sequence entered	mode	x	y	display
RESET	first	0	0	0
RESET 1	first	1	0	1
RESET 1 +	second	1	1	1
RESET 1 + 2	second	1	2	2
RESET 1 + 2 =	first	3	2	3
RESET 4 - 1 =	first	3	1	3
RESET 1 - 4 =	first	253	4	253
RESET 1 + 2 3 =	first	4	3	4
RESET 1 + =	first	2	1	2
RESET 1 + = =	first	3	1	3
RESET 1 + = + =	first	4	2	4
RESET 1 + + =	first	2	1	2
RESET 1 + - =	first	0	1	0

Note that these examples **do not cover all cases** and passing all of them does not guarantee that you will receive full credit. See the specification above for the full requirements.

DIRECTIONS

- Download `calc.sv`, `calc_dp.sv`, and `tb_calc.sv`.
- Design your state machine in `calc.sv`.
- Submit `calc.sv` **only** on Canvas.

MARKING

We will only mark your **last** submission; marking is by autograder only.

You will receive **zero** if:

- You did not submit anything before the Canvas deadline.
- You did not sign out with the TA before leaving the exam room.
- The file you submit is not called `calc.sv`.
  - If you submit multiple times, Canvas may rename this as `calc-1.sv` etc.; this is fine.
- Port declarations have been modified.
- The `calc_dp` module has been modified.
- The name of the `calc_dp` instance in `calc` has been modified.
- The `calc_dp` module *declaration* has been included in the `calc` file.
- Sequential logic is triggered on anything other that the rising clock edge.
- Reset is not active-low.
- Your solution does not synthesize with Quartus.
- Gate-level simulation yields different results from RTL-level simulation.
- Your design does not pass all tests in the provided `tb_calc.sv`.

You will receive **partial credit only** if:

- Your design fails *any* of our test cases (we have *many* more than provided in `tb_calc.sv` and the examples table).
- Your RTL-level and gate-level simulations are identical but you have latches in your circuit.
- You implemented the relaxed-timing variant.

ADVICE

- Read the specification carefully and think through the provided examples. Make sure you understand how the calculator operates **before writing any RTL**.
- Get the simple test in `tb_calc.sv` to pass first; submit that, and then worry about meeting the rest of the spec.
- Add tests as you implement new features, and **always run all tests** so you don't submit an "improved" solution that is broken. We only test your last submission.
- Use the examples in the table to start your tests.
- Submit whenever you have implemented **and tested** more features.
- Don't overcomplicate. The reference solution is 25 lines.