RTL Design

Introduction

In this lab you will design a special-purpose computer.

Requirements

Your design will compute the sum, the maximum or the minimum of the six rightmost digits of your student number.

The processing to be done by your design will depend on the fourth digit of your 8-digit student number:

rightmost digit	calculation required
4, 5, 6	sum
1, 2, 3	maximum
7, 8, 9	minimum

For example, if your BCIT was A00987654 the fourth digit is 8 so you would find the minimum of the six rightmost digits (4).

The binary values of the six rightmost digits of your student number should be stored in a ROM modelled as an array of 4-bit `unsigned` values.

Your design should use `key(0)` as a clock input and `key(1)` as an active-low synchronous reset input. Intermediate results should be displayed on `led(6 downto 0)`. `led(7)` should be turned on when the computation is complete. When the computation is complete further clock edges should have no effect until the next reset.

Design

Datapath

Your datapath will consist of: an address register which selects the value to be read from the memory, a result register which holds the intermediate result at each stage of the computation and a comparator or adder (depending on the computation).

In each clock cycle the datapath can carry out one of the following operations:

- Initialize the result and memory address registers.
- Update the intermediate result register based on that register's current value and the value being read from memory. At at the same time, the address register should be updated to point to the next memory location. As with all synchronous designs, the updates to the register values happen on the next clock edge.
- Do nothing (`led(7)` should be on and the final result should be displayed on `led(6 downto 0)`).

Controller

The controller will execute the following algorithm after being reset (described in C-like pseudo-code):

```
// initialization
result = <initial value> ;
address = 0 ;

// processing
while ( address != 6 ) {
  result = <value based on result and memory[address]> ;
  address = address + 1 ;
}

// done
turn on led(7) and wait for next reset ;
```

where `<initial value>` and `<value based on...>` depend on which computation you are doing.

Pre-Lab Report

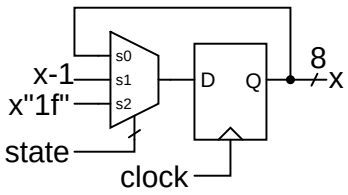
Submit the following to the appropriate dropbox on the course web site:

- the sequence of values you expect to see displayed on the LEDs (note that this depends on

your student number as well as the algorithm you must use). For example, if your student number is A00324156 you would expect to see the sequence 3, 3, 4, 4, 5, 6 since the fourth digit is 2 which means you have to compute the maximum of the six digits.

- Pseudo-code for the algorithm (as above, but with the expressions in angle brackets filled in for your specific design).
- A state transition diagram for the controller. Label the state transition conditions.
- A block diagram of the datapath. This will consist of one register for each variable in your algorithm. The input of each register is connected to a multiplexer. Each input of the multiplexer is one of the values that can be loaded into the register.

  For example, if you had an 8-bit register `x` that, in different states, could be loaded with `x"1f"`, `x-1` or `x` you might draw it as follows[1]:



- A table showing the multiplexer input that is selected for each state for each register. For example[2]:

state	value loaded into:	
	address register	result register
s0	0	0
s1	address+1	result−memory(address)
...	...	...

- VHDL code that implements the required algorithm for *your* student number (note that each design will be different). You may place multiple simple assignments in one process statement.

[1] It is better practice to use an enumerated type to select the operation rather than labelling the multiplexer inputs with the states.

[2] This is definitely *not* an example applicable to this lab.

Procedure

Follow the instructions given in Lab 1.

Lab Report

Submit a PDF document containing the following to the appropriate dropbox:

- your VHDL code with all errors corrected
- a screen capture of the Flow Summary section of the Compilation report
- the block diagram produced by running Tools > Netlist Viewers > RTL Viewer.

FAQ and Hints

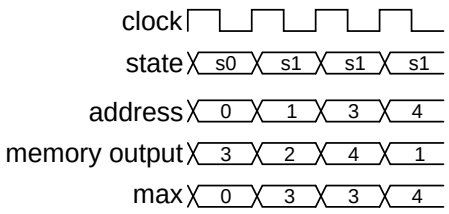
What are the steps in completing an RTL design? Briefly:

- Make sure you understand the algorithm. If necessary, go through some examples and keep track of the values stored in each variable.
- Decide on the data structures (variables) required to implement the algorithm (already done for you, see above). Each variable in your algorithm will become a register in your hardware design.
- Break down the algorithm into a sequence of steps. Each step is a controller state. You can combine steps that don't depend on the results of a previous step.
- Determine the different values (constants, register outputs and functions of these) that need to be loaded into each register. Each of these will be an input to the multiplexer at is at the input to that register (see schematic above).
- Decide on what value is to be loaded into each register in each state.
- Write the VHDL that implements a state machine for each register (including the state register if necessary).

Can I use the memory address as a state variable? If this works for you, yes.

I'm confused about what happens when. All register outputs change simultaneously at the rising edge of the clock. In-between rising clock edges the register outputs propagate through the combinational logic (including multiplexers) to the register inputs.

It can help to draw a timing diagram showing the value of each register in each clock period. For the example above the register values would (might) be:

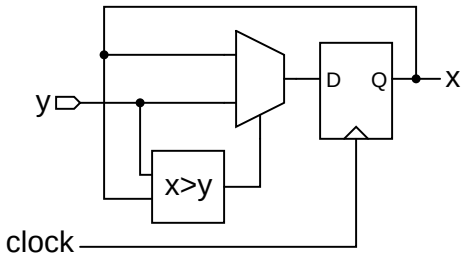


How do I declare and initialize a ROM memory? See the lecture notes for how to declare and use memories in VHDL. You can initialize an array used as a ROM as follows (this example is for BCIT ID A00123456):

```
subtype nybble is unsigned (3 downto 0) ;
type rom6x4 is array (0 to 5) of nybble;
signal mynum : rom6x4 := ( x"1", x"2", x"3", x"4", x"5", x"6" ) ;
```

What should I use as the initial result value? Would 0, 0 and 9 (respectively) work for the three algorithms? Or you could use the first digit.

What does the datapath for min/max look like? Perhaps something like[3]:



which in VHDL might be written as:

```
next_x <= x if x > y else y ;
```

[3] With the appropriate signal names, of course.

How is this different than using a microcontroller?

- The hardware can be faster. Note that the datapath updates all registers at the same time. This parallelism allows a special-purpose computer to operate faster than a general-purpose one.
- The special-purpose computer may require less [silicon die] area and consume less power since only the registers and logic required for this specific algorithm need to be implemented.
- Designing hardware will usually take longer than writing software to do the same job (in fact, the hardware design usually starts with a software model).
- If the task is simple and slow enough for a microcontroller, then that is likely to be less expensive than using a PLD or FPGA. This is because the microcontroller will likely have a smaller die area and higher sales volume.

What are some practical applications of RTL design? Look around you. The digital ICs (and any FPGAs) in almost any modern electronic device were designed this way.