Computer Systems and -architecture

Project 4: Memory

1 Ba INF 2016-2017

Bart Meyers
bart.meyers@uantwerpen.be

Don't hesitate to contact the teaching assistant of this course. You can reach him in room M.G.3.17 or by e-mail.

Time Schedule

Projects are solved in pairs of two students. Projects build on each other, to converge into a unified whole at the end of the semester. During the semester, you will be evaluated three times. At these evaluation moments, you will present your solution of the past projects by giving a demo and answering some questions. You will immediately receive feedback, which you can use to improve your solution for the following evaluations.

For every project, you submit a small report of the project you made by filling in `verslag.html` completely. A report typically consists of 500 words and a number of drawings/screenshots. Put all your files in one tgz archive, as explained on the course's website, and submit your report to the exercises on Blackboard.

- Report deadline: **December, 7 2016, 23u55**
- Evaluation and feedback: **December, 9 2016**

Project

Read sections B.7, B.8 and B.10 of Appendix B. You can use all Logisim libraries for this assignment.

1. Build a **16-bit register** using 16 D flip-flops that are updated on the *falling edge* (beware: in Logisim D flip-flops are by default on the rising edge). Inputs are:

   - 16-bit "D", which denotes the input data
   - 1-bit "reset", that sets the contents of the register to `0000000000000000` if its value is 1
   - 1-bit "write", that enables writing the value of D to the register if its value is 1
   - 1-bit C, the clock signal

   The only output is a 16-bit Q that contains the contents of the register.

2. Build a **register file** made of 16 of your own 16-bit registers. The register file must be able to read from and write to specified registers. In this case, the register file reads from two registers, and can possibly write to a register at the same time. Register 0 is a special

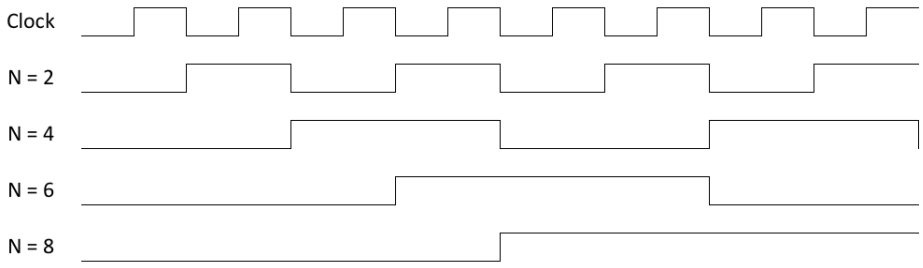
case: it always contains zero, and writing to it does not modify its contents. The register file has the following in- and outputs:

name	in/out	width	meaning
rs	I	4 bits	register `rs` index number
rt	I	4 bits	register `rt` index number
rd	I	4 bits	register `rd` index number
Data	I	16 bits	used as input for the write operation, i.e., the new $rd value
write	I	1 bit	write to `rd` enabled?
C	I	1 bit	clock input
reset	I	1 bit	reset all registers?
S	O	16 bits	`$rs`; register `rs` content
T	O	16 bits	`$rt`; register `rt` content

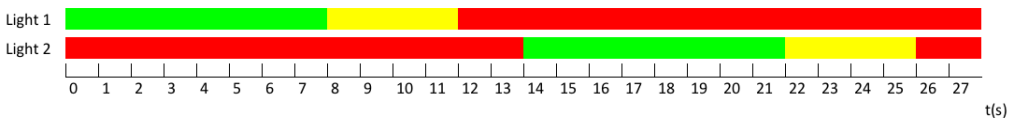
We refer to a 4-bit register *name* (i.e., index number) as e.g., rs or r1, and to its 16-bit *value* (i.e., data content) as respectively $rs or $r1.

3. Build a **counter** using your own 16-bit carry lookahead adder and 16-bit register. Inputs are C (the clock) and D (an 16-bit number up to which the counter counts), the output is the current 16-bit value of the register. At every clock tick, the counter adds 1 to the number in the register. When the register value is equal to or greater than D, the value is reset to zero. A counter with its D-input equal to 3 counts from 0 to 2. You can use the Logisim built-in *Comparator*.

4. Build a **clock divider**. A clock divider is used to create a slow "daughter" clock from a faster "parent" clock. Inputs are C (the clock) and N (16-bit number). You can assume that N is even. The clock divider generates an output clock signal with a frequency that is N times lower than the input clock signal. Use components of your own as much as possible.
Optional: can you make the divider work with odd numbers as well?

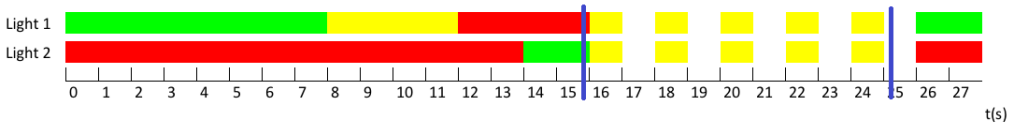


5. Build a **finite-state machine** that implements a traffic light system on a cross section. Finite-state machines use memory and a clock. Since finite-state machines are *synchronous*, a new state is computed every clock cycle. A 2 Hz clock has a full clock cycle of 1 second. Use your counter and clock divider to advance through the states and make sure your state transitions happen at the right time. The two traffic lights behave like the following figure:



Additionally, it is possible for a police officer who wants to regulate traffic himself to override the traffic light. This switch is represented by a "Button" from the Input/Output library. When pressed, the traffic lights start blinking the yellow light. When pressed

again, the traffic light restarts its regular behaviour by starting a new cycle. The following is an example of a police officer pressing the button twice (represented by a blue vertical line):



Note how pressing a button can happen in the middle of a clock cycle, and needs to take effect at the start of the next clock cycle.

6. To prepare for the next lab session, read sections 4.1, 4.2, 4.3 and 4.4 of Chapter 4.