Computer Systems and -architecture

Project 4: Memory

1 Ba INF 2022-2023

Brent van Bladel
brent.vanbladel@uantwerpen.be

Don't hesitate to contact the teaching assistant of this course. You can reach him in room M.G.305 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 2, 2022, 22u00**
- Evaluation and feedback: **December 6, 2022**

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 (from Logisim) that are updated on the *falling edge* (beware: in Logisim D, flip-flops are by default set to update on rising edge). Inputs are:

   - 16-bit "D", which denotes the input data
   - 1-bit "reset", that sets the contents of the register to `0000000000000` 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, which will contain 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 8 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.

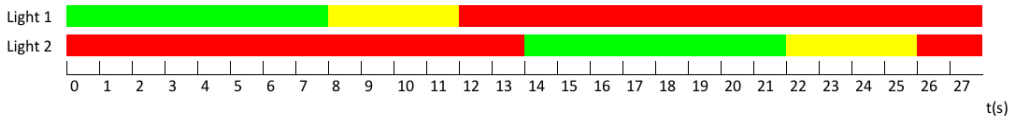
Register 0 is a special case: it always contains zero, and writing to it does not modify its contents. The register file has the following in- and outputs:

name	in/out	width	meaning
rs	I	3 bits	register `rs` index number
rt	I	3 bits	register `rt` index number
rd	I	3 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 3-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 (a 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 **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 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:



5. (*Bonus*) Build a 16-bit **stack** using the logisim RAM element. Use your own 16-bit register to store the stack pointer. The stack pointer should always point to the next free address after the top of the stack. By default, the *peek* operation is performed, which simply outputs the value of the top of the stack, or zero if the stack is empty. Two 1-bit inputs will be used to indicate a *push* or a *pop* operation. The *push* operation will place the data from the `Data` input on top of the stack, and increase the stack pointer by one. The *pop* operation will replace the data on top of the stack with zero, and decrease the stack pointer by one.

name	in/out	width	meaning
Push	I	1 bit	perform the push operation?
Pop	I	1 bit	perform the pop operation?
Data	I	16 bits	used as input for the push operation
C	I	1 bit	clock input
reset	I	1 bit	reset memory and stack pointer?
Top of Stack	O	16 bits	value on top of the stack
Error	O	1 bit	outputs 1 in case pop is performed on an empty stack

Note: The 16-bit *Top of Stack* output should be zero in case of a *push* or a *pop* operation. *Note 2:* When both the *push* and the *pop* operation are requested simultaneously, the behaviour can be considered undefined but the error output should indicate this.

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