ECE4530 Fall 2014: Homework 2

Homework 2 is a series of small programming exercises in the GEZEL RTL Language. The objective of this homework is to learn the basic concepts of register-transfer modeling; the next homework will employ these skills in a more complex assignment.

You need to develop your solution in the GEZEL hardware description language. On the class website, refer to the document GEZEL Codesign Kit Installation for ECE4530" for instructions on how to install the cosimulation tools that you will need in this class.

The Homework is a coding excercise: your response will consist of a zip file with 10 GEZEL programs. Work through each of the following assignments, and code a response for each. You will start from the testbench files that are provided with the assignment.

Preliminaries: Running GEZEL

This handout is not a tutorial; please refer to the GEZEL homepage or the textbook for an in-depth discussion on GEZEL coding and simulation. The homepage is at `http://rijndael.ece.vt.edu/gezel2`.

The GEZEL tools are installed in the `/opt/gezel` directory. To run the GEZEL simulator, you would use something like

```
/opt/gezel/bin/fdlsim mydesign.fdl 10
```

which simulates 30 clock cycles from a design file `mydesign.fdl`. Here is an example design file. Open an editor and copy this code into `mydesign.fdl`.

```
dp showme {
  reg a : ns(8);
  always {
    a = a + 1;
    $display($cycle, " shows me ", a);
  }
}

system S {
  showme;
}
```

If you simulate this file with the command as shown above, you will see the following output.

```
0 shows me 0/1
1 shows me 1/2
```

```
2 shows me 2/3
3 shows me 3/4
4 shows me 4/5
5 shows me 5/6
6 shows me 6/7
7 shows me 7/8
8 shows me 8/9
9 shows me 9/a
```

The notation `x/y` indicates the value at the output and input, respectively, of register `a`. Verify that you can run this simulation correctly before starting with the assignments.

In each of the following assignments, you need to complete a file such to meet that you meet the specifications. The correct reference output is shown, as well.

Design 1: A simple four-bit counter

Design a circuit with a four-bit output that counts from 0 to 15, and then wraps around to 0.

- Design file: `c1.fdl`
- Reference output: The first 20 clock cycles of the output looks as follows. The first column is the clock cycle, the second column is the output count.

```
0  0
1  1
2  2
3  3
4  4
5  5
6  6
7  7
8  8
9  9
10  10
11  11
12  12
13  13
14  14
15  15
16  0
17  1
18  2
19  3
```

Design 2: An up-down counter

Design a circuit that counts from 0 to 5, and then back to 0.

- Design file: `c2.fdl`
- Reference output: The first 20 clock cycles of the output looks as follows. The first column is the clock cycle, the second column is the output count.

```
0 0
1 1
2 2
3 3
4 4
5 5
6 4
7 3
8 2
9 1
10 0
11 1
12 2
13 3
14 4
15 5
16 4
17 3
18 2
19 1
```

Design 3: A ones counter

Design a circuit that counts the number of 1-bits in a string. The circuit has a 1-bit input, and accepts one bit every clock cycle. The circuit has an 8-bit output, reflecting the count of 1-bits. The counter needs to have 8-bit resolution (ie. counting a sequence of up to 255 bits). The test bench will provide a pseudorandom stream.

- Design file: `c3.fdl`
- Reference output: The first 20 clock cycles of the output looks as follows. The first column is the clock cycle, the second column is the bit at the input, the third column is the output count. The output count comes with a delay of one clock cycle, to account for the latency of the accumulation.

```
0  0  0
1  0  0
2  0  0
3  0  0
4  0  0
5  1  0
6  0  1
7  1  1
8  0  2
9  1  2
10  1  3
11  1  4
12  0  5
13  1  5
14  1  6
15  0  7
16  0  7
17  0  7
18  1  7
19  1  8
```

Design 4: A delay circuit

Design a circuit with a one-bit input and a one-bit output. The circuit generates a done- pulse five clock cycles after a start-pulse appeared at its input. The test bench generates a start pulse and then waits for a done pulse to appear before generating the next start pulse.

- Design file: `c4.fdl`
- Reference output: The first 20 clock cycles of the output looks as follows. The first column is the clock cycle, the second column is the bit at the input, the third column is the output bit. The first start pulse is in clock cycle 0. In clock cycle 5, the done output is asserted. The done pulse remains high until the next start pulse is provided in clock cycle 8.

```
0  1  0
1  0  0
2  0  0
3  0  0
4  0  0
5  0  1
6  0  1
7  0  1
8  1  1
9  0  0
10  0  0
11  1  0
12  0  0
13  0  0
14  0  0
15  0  0
16  0  1
17  0  1
18  0  1
19  1  1
```

Design 5: A parity circuit

Design a circuit that computes a parity bit over the last 8 input bits. A parity bit makes the number of '1' bits in a bit-string even, when the parity bit is appended to that bitstring. For example, in the bitstring '00111001', the parity is 0, because the bitstring '001110010' contains an even number of bits.

- Design file: `c5.fdl`
- Reference output: The first 20 clock cycles of the output looks as follows. The first column is the clock cycle, the second column is the bit at the input, the third column is the output bit. Note that the parity is computed with one clock cycle of delay. For example, the output bit in cycle 10 is 1. This is because the input bitstring consists of the 8 input bits provided in clock cycle 2 to 9, ie. the bitstring '00010101'

```
0  0  0
1  0  0
2  0  0
3  0  0
4  0  0
5  1  0
6  0  1
7  1  1
8  0  0
9  1  0
10  1  1
11  1  0
12  0  1
13  1  1
14  1  1
15  0  0
16  0  1
17  0  1
18  1  0
19  1  0
```

Design 6: A pattern matching circuit with no overlap

Design a circuit that accepts one bit every clock cycle. The circuit will output a '1' when it scans the bit pattern '1101' in the input stream. The output is generated with one clock cycle delay. You don't have to take overlapping patterns into account. For example, the bitstring '1101101' contains two overlapping copies of '1101'. However, you only need to match the first one. The corresponding output string, in this case, would be '0000100'.

- Design file: `c6.fdl`
- Reference output: The first 32 clock cycles of the output looks as follows. The first column is the clock cycle, the second column is the bit at the input, the third column is the output bit. The output is computed with one clock cycle of delay.

```
0 0 0
1 0 0
2 0 0
3 0 0
4 0 0
5 1 0
6 0 0
7 1 0
8 0 0
9 1 0
10 1 0
11 1 0
12 0 0
13 1 0
14 1 1
15 0 0
16 0 0
17 0 0
18 1 0
19 1 0
20 1 0
21 1 0
22 1 0
23 0 0
24 0 0
25 1 0
26 1 0
27 0 0
28 1 0
29 0 1
30 0 0
31 1 0
```

Design 7: A pattern matching circuit with overlap

Design a circuit that accepts one bit every clock cycle. The circuit will output a '1' when it scans the bit pattern '0101' in the input stream. The output is generated with one clock cycle delay. You need to handle overlap. For example, the bitstring '0101010' contains two overlapping copies of '0101'. The corresponding output string, in this case, would be '0000101'.

- Design file: `c7.fdl`
- Reference output: The first 32 clock cycles of the output looks as follows. The first column is the clock cycle, the second column is the bit at the input, the third column is the output bit. The output is computed with one clock cycle of delay.

```
0  0  0
1  0  0
2  0  0
3  0  0
4  0  0
5  1  0
6  0  0
7  1  0
8  0  1
9  1  0
10  1  1
11  1  0
12  0  0
13  1  0
14  1  0
15  0  0
16  0  0
17  0  0
18  1  0
19  1  0
20  1  0
21  1  0
22  1  0
23  0  0
24  0  0
25  1  0
26  1  0
27  0  0
28  1  0
29  0  0
30  0  0
31  1  0
```

Design 8: Computing a rolling maximum

Design a circuit that computes the rolling maximum over three numbers. The circuit has an 8-bit input, and reads one data byte every clock cycle. The circuit has an 8-bit output, which holds the maximum over that last three numbers seen. The maximum is computed over unsigned numbers (0 to 255), and is computed with a delay of one clock cycle. The testbench provides a pseudorandom sequence of bytes.

- Design file: `c8.fdl`
- Reference output: The first 32 clock cycles of the output looks as follows. The first column is the clock cycle, the second column is the byte at the input, the third column is the output byte. For example, the output in clock cycle 11 is 231, since 231 is the maximum of the inputs from clock cycle 10, 9, and 8: 231, 168, and 45.

```
0  0  0
1  95  0
2  50  95
3  233  95
4  52  233
5  3  233
6  134  233
7  45  134
8  168  134
9  231  168
10  26  231
11  177  231
12  92  231
13  11  177
14  238  177
15  117  238
16  80  238
17  111  238
18  2  117
19  121  111
20  132  121
21  19  132
22  86  132
23  189  132
24  248  189
25  247  248
26  234  248
27  65  248
28  172  247
29  27  234
30  190  172
31  5  190
```

Design 9: Computing the median over three numbers

Design a circuit that computes the median over a block of three numbers. The median is the middle value of a set. For example, the median of 8, 28, and 3 is 8. The circuit has an 8-bit input, and reads one data byte every clock cycle. The circuit has an 8-bit output, which holds the median over a block of three numbers. This is not a rolling median: the input stream is treated as blocks of three numbers, with each block taking three clock cycles to load. The median is computed over unsigned numbers (0 to 255), and is computed with a delay of one clock cycle. The testbench provides a pseudorandom sequence of bytes.

- Design file: `c9.fdl`
- Reference output: The first 32 clock cycles of the output looks as follows. The first column is the clock cycle, the second column is the byte at the input, the third column is the output byte. For example, the output in clock cycle 11 is 168, since 168 is the median of the inputs from clock cycle 10, 9, and 8: 26, 231, and 168.

```
0 0 0
1 95 0
2 50 0
3 233 50
4 52 50
5 3 50
6 134 52
7 45 52
8 168 52
9 231 134
10 26 134
11 177 134
12 92 177
13 11 177
14 238 177
15 117 92
16 80 92
17 111 92
18 2 111
19 121 111
20 132 111
21 19 121
22 86 121
23 189 121
24 248 86
25 247 86
26 234 86
27 65 247
28 172 247
29 27 247
30 190 65
31 5 65
```

Design 10: Computing a runlength encoding

A runlength code is a condensed representation of a bitstring. For example if the bitstring is '00011110', then (3, 4, 1) is a valid runlength encoding assuming that we know that the first bit is a '0'. Design a circuit to compute a runlength code. The circuit has a one-bit input, and accepts a bit every clock cycle. The circuit has an 8-bit output that will hold the runlength code. The circuit also as a one-bit output that indicates during what clock cycles the runlength code is valid. The runlength code is computed with one clock cycle of delay. The testbench provides a pseudorandom bitstream.

- Design file: `c10.fdl`
- Reference output: The first 31 clock cycles of the output looks as follows. The first column is the clock cycle, the second column is the input bit, the third column is the runlength code, and the last column is the valid bit. For example, the output in cycle 23 is (5,1). This means that cycle 23 ends a string of 5 bits of the same value. Indeed, cycle 18 through 22 hold a '1' at the input.

```
0  0  0  0
1  0  1  0
2  0  2  0
3  0  3  0
4  0  4  0
5  1  5  1
6  0  1  1
7  1  1  1
8  0  1  1
9  1  1  1
10  1  1  0
11  1  2  0
12  0  3  1
13  1  1  1
14  1  1  0
15  0  2  1
16  0  1  0
17  0  2  0
18  1  3  1
19  1  1  0
20  1  2  0
21  1  3  0
22  1  4  0
23  0  5  1
24  0  1  0
25  1  2  1
26  1  1  0
27  0  2  1
28  1  1  1
29  0  1  1
30  0  1  0
```

What to turn in

- A ZIP file with all completed designs, c1.fdl through c10.fdl. Please follow guidelines inside of the files as to where to enter your edits. Return the ZIP file as attachment to your answer in Scholar.