Computer Structure Assignment 3
Combinational Circuits

Carmi Merimovich

December 30, 2024

When asked to build a circuit you should supply:

1. A block diagram.
2. A circuit diagram.
3. A Verilog module with the **exact** signature as specified and with **identical** file name.

1. Follow the steps used in the lectures to build the module `binaryAdder` to build the module `binarySubtractor`. Thus:

   (a) Design, using **only** gates, the module

   ```
   module fullSubtractor(
                  output bo,
                  output s,
                  input  x,y,bi);
   ```

   (b) Design, using **only** gates and `fullSubtractor`s, the module

   ```
   module binarySubtractor #(N) (
                            output b,
                            output v,
                            output [N-1:0] z,
                            input  [N-1:0] x, y,
                            input  bIn)
   ```

   returning in `z` the value `x - y - bIn` with `b` and `v` being the usual overflow flags.

2. Build a module with the signature

   ```
   module negate #(N) (
                   output V,
   ```

```
                output [N-1:0] out,
                input  [N-1:0] in)
```

accepting a signed integer and returning its negation. `V` should be true if and only if there is an overflow.

The `binaryAdder` module can be used.

3. Build the module

```
module binarySubtractor1 #(N) (
                output b,
                output v,
                output [N-1:0] z,
                input  [N-1:0] x, y,
                input  bIn)
```

accepting two integers `x`, `y` and returning `x - y - bIn` in `z`. The `b` and `v` flags are as in the `binaryAdder` module.

Use the `negate` module from the previous question.

The `binaryAdder` can be used.

4. Build an unsigned comparator module:

```
module #(N) comparator(
                    output lt, eq, gt,
                    input [N-1:0] x,
                    input [N-1:0] y);
```

The behavior is as follows:

$$\begin{aligned} \text{gt} = 1 &\iff \text{x>y} \\ \text{eq} = 1 &\iff \text{x=y} \\ \text{lt} = 1 &\iff \text{x<y} \end{aligned}$$

5. Build a signed comparator module:

```
module #(N) comparatorSigned(
                    output lt, eq, gt,
                    input [N-1:0] x,
                    input [N-1:0] y);
```

The behavior is the same as in the previous questions, just the numbers here are signed.

6. Build (from gates only) a 4×2 priority encoder

```
module encoderPri4(
                output v,
                output [1:0] z,
                input [3:0] x);
```

7. Build a priority encoder giving the index of the minimal set input.

```
module encoderMin #(SEL) (
                          output v,
                          output [SEL-1:0] z,
                          input  [2**SEL-1:0] x);
```

8. Build a reasonably efficient left and right shifters with the following signatures:

```
module shiftLeft #(SEL) (output [2**SEL-1:0] out,
                          input [2**SEL-1:0] in,
                          input [SEL-1:0] cnt);

module shiftRight #(SEL) (output [2**SEL-1:0] out, //Logic
                          input [2**SEL-1:0] in,
                          input [SEL-1:0] cnt);

module shiftARight #(SEL) (output [2**SEL-1:0] out, // Arithmetic
                          input [2**SEL-1:0] in,
                          input [SEL-1:0] cnt);
```

Note. In `shiftRight` the MSB gets a zero. In `shiftARight` the MSB is replicated. E.g.,

shift right	$0111 \mapsto 0011$
shift right	$1111 \mapsto 0111$
shift right arithmetic	$0111 \mapsto 0011$
shift right arithmetic	$1111 \mapsto 1111$

9. Build a binary Arithmetic Logic Unit with signature:

```
module alu #(N) (
        output [N-1:0] out,
        input  [N-1:0] in0,in1,
        input  [2:0] op
        )
```

where:

op	Operation	
0	$\text{out} = \text{in0} + \text{in1}$	addition
1	$\text{out} = \text{in0} - \text{in1}$	subtraction
2	$\text{out} = \text{in0} \,\&\, \text{in1}$	bitwise and
3	$\text{out} = \text{in0} \mid \text{in1}$	bitwise or
4	$\text{out} = \sim \text{in0}$	bitwise not
5	$\text{out} = \text{in0} << \text{in1}$	
6	$\text{out} = \text{in0} >>_{\text{u}} \text{in1}$	unsignd shift
7	$\text{out} = \text{in0} >>_{\text{s}} \text{in1}$	arithmetic shift

10. **In this assigment we are interested only in the `bcdAdder` and `bcdSubtractor` modules. The detailed module list below is due to popular demand!**

    Binary Coded Decimal (BCD) is, well, a binary code for decimal numbers, e.g.;

    $$(192)_{10} = (0001\,1001\,0010)_{\text{BCD}}.$$

    BCD is a signed/magnitude method, where A, C, E, F code a non-negative number and B,D code negative number, e.g.,

    $$(+192)_{10} = (1010\,0001\,1001\,0010)_{\text{BCD}},$$
    $$(-192)_{10} = (1011\,0001\,1001\,0010)_{\text{BCD}}.$$

    4 binary digits which are coding a digit or a sign are called **a nibble**.

    (a) Build a full bcd adder:

    ```
    module bcdFullAdder(
            output cOut,
            output [3:0] z,
            input  [3:0] x,
            input  [3:0] y,
            input  cIn)
    ```

    Your friends are `binaryAdder`, `binarySubtractor`, and `comparator`.

    (b) Build a full bcd subtractor:

    ```
    module bcdFullSubtractor(
            output bOut,
            output [3:0] z,
            input  [3:0] x,
            input  [3:0] y,
            input bIn)
    ```

    Your friends are `binaryAdder`, `binarySubtractor`, and `comparator`.

    (c) Build the unsigned bcd adder. There are no sign nibbles here!

    ```
    module bcdUAdder #(parameter N=1) (
            output cOut,
            output [3:0] z[N],
            input  [3:0] x[N],
            input  [3:0] y[N],
            input  cIn)
    ```

    (d) Build te unsigned bcd subtractor. There are no sign nibbles here!

```
module bcdUSubtractor #(parameter N=1) (
        output bOut,
        output [3:0] z[N],
        input  [3:0] x[N],
        input  [3:0] y[N],
        input  bIn)
```

(e) Build a BCD adder. Here the highest numbered nibbles are sign nibbles!

```
module #(N) bcdAdder(
                output c,
                output [3:0] z[N],
                input  [3:0] x[N],
                input  [3:0] y[N]);
```

Your friends are `bcdUAdder`, `bcdUSubtractor` and `comparator`. Knowledge of how signed decimal numbers addition is done by humans is assumed.

(f) Build a BCD subtractor. Here the highest number nibbles are sign nibbles!

```
module #(N) bcdSubtractor(
                output bOut,
                output [3:0] z[N],
                input  [3:0] x[N],
                input  [3:0] y[N]);
```

Your friend is `bcdAdder`. (This is **really** simple) Knowledge of how signed decimal numbers subtraction is done by humans is assumed.

11. Build (from gates only) a negative decoder with a negative enable. I.e., the decoder is active when the enable line is clear. The output lines are in general set except for the selectd line.

```
module decoderNeg #(parameter SEL=1) (
  output [2**SEL-1:0] m,
  input [SEL-1:0] sel
);
```

The following is **optional**

---

12. (Not easy) Build an unsigned binary multiplier.

```
module #(N) multiply(
                output [N+N-1:0]z,
                input  [N-1:0] x,y)
```

This is really easier than it looks (if optimizations are to be ignored, which they are). Use `mux`es to get either zero or the multiplicand for each bit in the multipler. Use (wide) adders to add the shifted number we got out of the `mux`es.

13. (Even less easier) Build a unsigned binary divider.

    Simple long division will do here.

    Use the comparator module and the `binarySubtractor`. Again, `mux`es will decide if subtraction is done with the divider or with zero.