Lab 3: Barrel Shifters

Due: 11:59:59pm, Sunday May 7, 2017

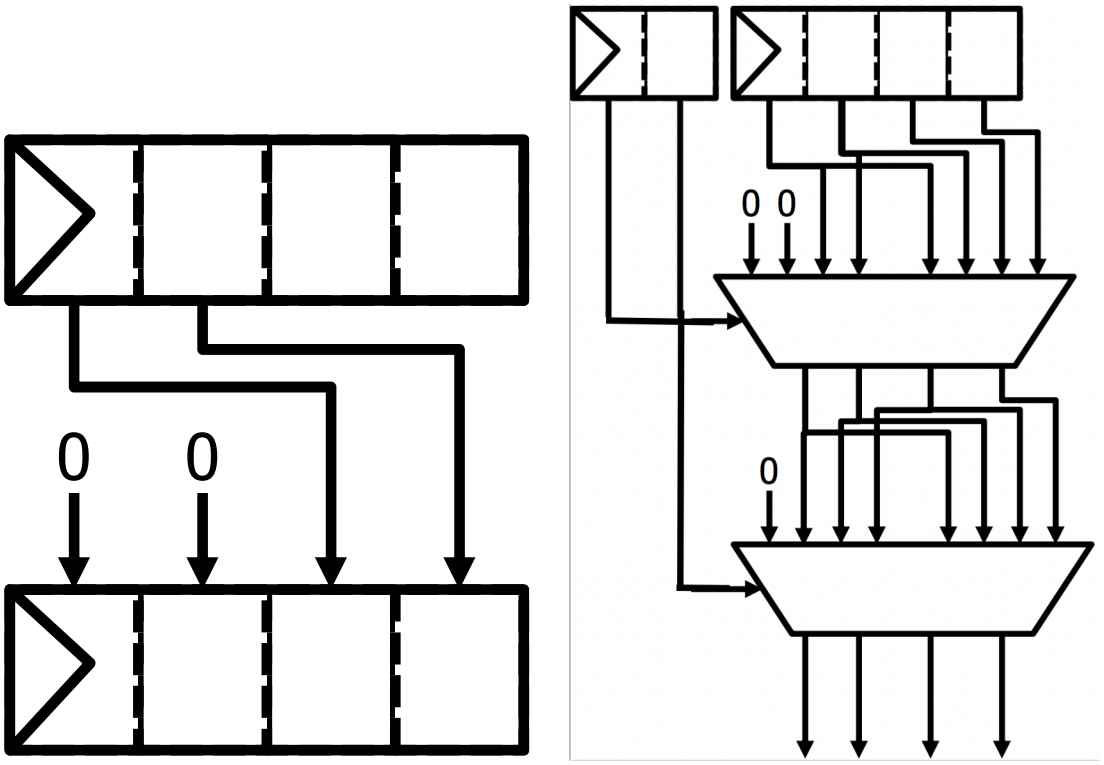
1 Introduction

The left shift (<<) and right shift (>>) operations are employed in computer programs to manipulate bits and to simplify multiplication and division in some special cases. For example, shifting a binary number by 1 bit to the left is equivalent to multiplication by 2. Shifting is considered a simple operation because it has a relatively small and fast implementation in hardware; shifts can typically be implemented in a single processor cycle, while multiplication and division may take multiple cycles.

2 Implementation

Shifting by a constant value can be implemented using only wires, as shown in Figure 1(a) where a 4-bit number is shifted by 2 bits to the right. Variable shifting is more complicated, but still efficient. Figure 1(b) shows the construction of a 4-bit *barrel shifter*. The additional 2-bit register, which can represent $0-3$ holds the amount by which the input data should be shifted. Barrel shifters are logarithmic circuits suitable for variable-length shifting. For example, in the 4-bit barrel shifter in Figure 1(b), shifting is performed in two steps/stages. Based on the shift amount, the muxes in the first and second stages decide whether or not a shift by two and one should be applied, respectively. In this way, the circuit can shift the input data by 0, 1, 2, or 3 bits.

In this assignment, you will build three different barrel shifters: a logical right shifter, an arithmetic right shifter and an logical shifter that can shift left or right based on a control bit.



(a) Constant Shift (b) Barrel Shift

Figure 1: Constant and variable-length 4-bit shifters. The barrel shifter is more general, but requires more hardware. The 2-bit register on the left of the barrel shifter determines the number of positions to shift.

3 Lab Assignment

First, make a copy of the source code using the following command

```
$ cp -r  /home/linux/ieng6/cs140g/public/lab3 .
```

3.1 Guidelines

Please follow these instructions strictly. You will get zero marks if the given instructions are not followed for an exercise.

1. You are NOT allowed to use BSV built-in left shift (<<) and right shift (>>) operators.
2. You are NOT allowed to use conditional statements including `if` statements, `case` statements, and the ternary operator (`?:`). Instead, you should use 32-bit multiplexers by calling the function `multiplexer32` provided in `BarrelShifter.bsv`.
3. You are NOT allowed to use any other BSV built-in function.

3.2 Building a Logical Right Shifter

Since we are creating combinational circuits, we will use `functions` to describe them in BSV. The inputs of the shifter are the 32-bit number you want to shift, `operand`, and the number of bits you want to shift it by, represented as a 5-bit unsigned integer, `shamt`. In the case of logical right shift, you should fill the higher order bits with zeros as in Figure 1(b).

Exercise 1: 20 points

In `BarrelShifter.bsv`, complete the function `logicalBarrelShifter` using `for` loop(s) to implement a logical barrel right shifter. Verify your implementation using:

```
$ make logical
$ ./logical
```

The testbench will print `PASSED LOGICAL` if your implementation is correct.

3.3 Building an Arithmetic Right Shifter

Arithmetic right shift can be used for dividing *signed* numbers by powers of two. The input operand is therefore in two's complement format and contrary to the logical right shifter which fills the left bits with zeros as it shifts the input, an arithmetic right shifter replaces the left bits with the sign bit (which is the leftmost bit) of the original number to preserve its sign.

Exercise 2: 40 points

In `BarrelShifter.bsv`, complete the function `arithmeticBarrelShifter` to implement an arithmetic barrel right shifter using `for` loop(s). Verify your implementation using:

```
$ make arithmetic
$ ./arithmetic
```

The testbench will print `PASSED ARITHMETIC` if your implementation is correct.

3.4 Building a Logical Left/Right Shifter

At first, it might seem that building a bi-directional barrel shifter would require two barrel shifters. However, we will use only one right shifter and get around the introduction of a second barrel shifter by adding multiplexers to reverse the operand twice. Using this method, a logical left shift is implemented using a right shifter in three steps: first, the operand is revered (so that the new LSB is the original MSB and vice versa). After a logical right shift is applied, the result is revered again. This requires the introduction of two more multiplexers which is somewhat cheaper than introducing a second, full-sized barrel shifter.

Exercise 3: 40 points

In `BarrelShifter.bsv`, complete the function `logicalLeftRightBarrelShifter` to implement a logical barrel shifter that can shift left or right. Note the additional 1-bit control input `shiftLeft` in the function signature. You should implement the algorithm described above and use the function `logicalBarrelShifter` that you implemented in Exercise 1 exactly once. Verify your implementation using:

```
$ make leftright
$ ./leftright
```

The testbench will print `PASSED LEFT AND RIGHT` if your implementation is correct.

4 Submission

Write down your name, PID and e-mail address in `Lab3.txt` located in `lab3/` directory. If you worked in a group, write down your partner's information as well. While your solutions can be identical, each group member must make their own submission. You will be scored based on your own submission. To submit your assignment, simply run the following command from `lab3/` directory:

```
$ bundleP3 <your-email>
```