Shift and Rotate Operations

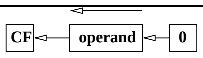
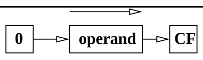
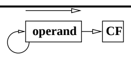
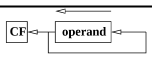
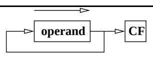
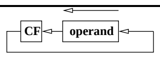
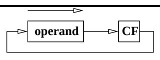
Dinesh Sharma
EE Department, IIT Bombay

May 17, 2019

We often need to shift and rotate operands in order to perform various operations such as serialization/de-serialization of data, multiplication etc.

1 Shift and Rotate Operations

The following operations are often required for processing of data:

SHL	op, count	
SAL	op, count	Same as SHL
SHR	op, count	
SAR	op, count	
ROL	op, count	
ROR	op, count	
RCL	op, count	
RCR	op, count	

These operations can be implemented by bi-directional shift registers with some control logic which provides circuits to choose the value and point of entry of new bits. However, this implementation can be very slow for a large number of shifts.

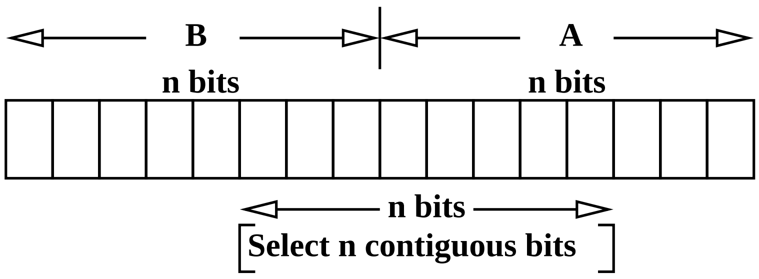
2 Barrel Shifters

Shift and rotate operations involve no computation. So why should we spend a large number of clock cycles for performing these operations? Ideally, we would like a shifter which produces the result in a single clock cycle.

2.1 Shift/Rotate as Select Operations

We can view Shift/Rotate operations as selection operations. In that case, the output can be produced directly by selecting the correct bits to appear in the desired position at the output.

Imagine two words A and B positioned as shown in the figure below.



We just have to choose B and A appropriately and select the correct range of n contiguous bits to implement various shift or rotate operations.

For all rotate operations, we choose B=A=data. For Shift Left, we choose B=data and A=0. For Logical Shift Right we choose B=0 and A=data. For Arithmetic Shift Right, we make B = replicated MSB of data, and set A = data.

Of course we do not actually copy data bits to A/B. Each output bit is produced by a mux which picks out the correct input data bit.

3 Barrel Shifters

Shifters which produce outputs as select operations are called barrel shifters. The name comes from viewing the inputs as well as outputs as a circular arrangement of bits. The shifter then connects the input circle to the output circle like the sections of a barrel.

A brute force implementation of such a shifter will require n multiplexers of n bits each, where the control inputs for each multiplexer are generated from the amount and type of shift/rotate that is desired. This requires quite a large number of complex n way multiplexers and puts a heavy load on data bits. Therefore rather than trying to complete the entire operation in one go, we use logarithmic barrel shifters, which are not so complex and take of the order of $\log_2 n$ operations to produce the result

3.1 Logarithmic Barrel Shifters

The control logic of a one step barrel shifter is complex because the amount of shift is variable. The loading on data lines and control logic complexity can be reduced if we break up the shift/rotate process into parts. We can carry out shifts in different stages, each stage corresponding to a single bit of the binary representation of the **shift amount**. Now the i'th stage needs to produce either no shift (if the corresponding control bit is '0') or a shift by a constant amount which is 2^i. Thus each stage requires only a two way mux for each bit. For example, a shift by 6 (binary: 110) will be carried out by first doing a 4 bit shift and then a 2 bit shift.

Since we need n bits to represent a maximum shift amount of $2^n - 1$ places, the number of bits to express the shift amount (and hence the number of shift stages required) is logarithmic in the maximum shift desired.

That is why such shifters are called Logarithmic Barrel Shifters. We can optionally buffer the outputs after each stage.

Bit i of the **shift amount** represents no shift (if it is 0) and a constant shift by 2^i places (if it is 1). If the amount of shift is fixed, the required bit can just be wired from the input bits.

The 2 way mux is controlled by bit i of the **shift amount**, Using this, we can choose either the unshifted operand bit or the operand bit 2^i places away from it.

This can be done for all bits of the operand in parallel. This constitutes one stage of the logarithmic shifter. The output can then be shifted again in the next stage, controlled by the next significant bit.

3.2 Right Rotate for an 8 bit Operand

To perform a right rotate operation, we use the scheme shown in Figure 1. For an 8 bit operand, the amount of rotation will be between 0 to 7 places, which can be represented by 3 bits. So we shall need a 3 stage implementation.

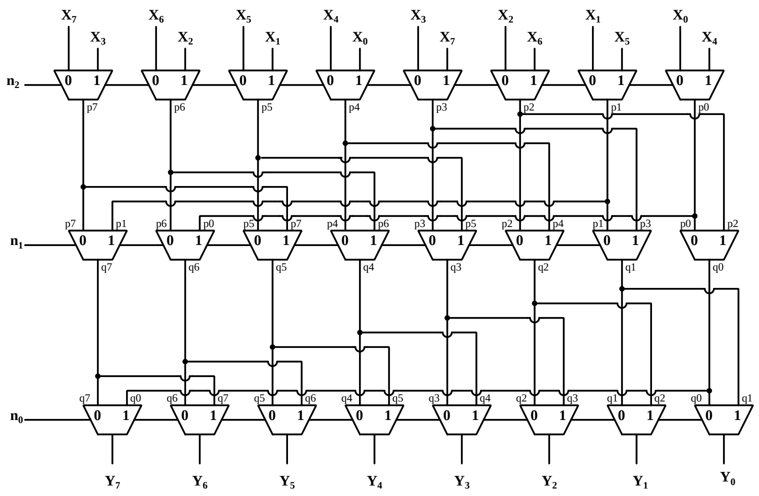


Figure 1: Right rotate for an 8 bit operand

- Each input bit drives just two muxes, each with just 2 inputs.
- At each stage, the muxes select either the unshifted bit or a bit 2^n places from it.
- 3 stages are required for 0 to 7 bits of shift.

3.3 8 bit Logical Shift Right

If we need a shift instead of a rotate, we feed a 0 instead of the corresponding data bit. We

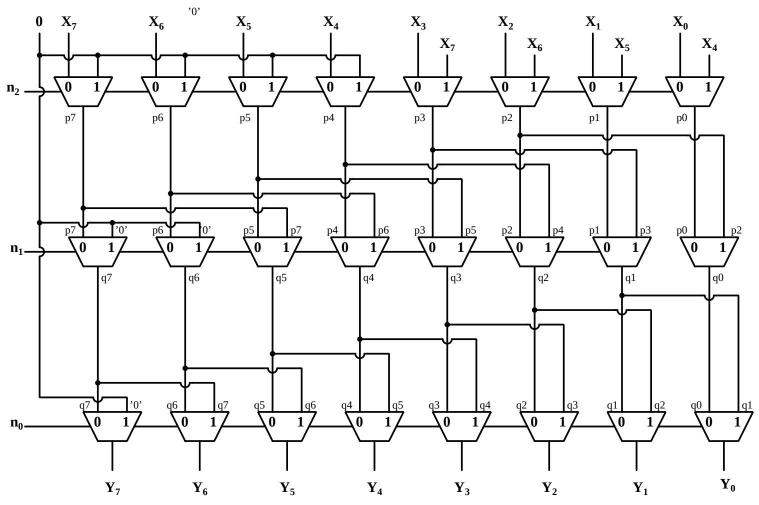


Figure 2: Logical Right Shift for an 8 bit operand

need to input 0's instead of data bits to the first 4 muxes in the first stage, to the first 2 muxes in the second stage and to 1 mux in the last stage.

3.4 Combining Rotate and Shift Operations

We can combine the circuits for rotate and shift functions by putting muxes where different inputs need to be presented for the two functions. Further, we can include the Arithmetic Shift function by choosing between 0 or X7 as the bit to be inserted from the right.

3.5 Rotate and Shift by Masking

We can also combine the rotate and shift functions by masking. We use the rotate function which does not lose any information, as the primary circuit. Now we can mask n bits at the left to 0 if a right shift operation was desired instead. In case of an arithmetic shift, n bits on the left have to be set to the same value as X7.

Shift/Rotate Left case is similar, except that the Logical and Arithmetic shifts are not different operations.

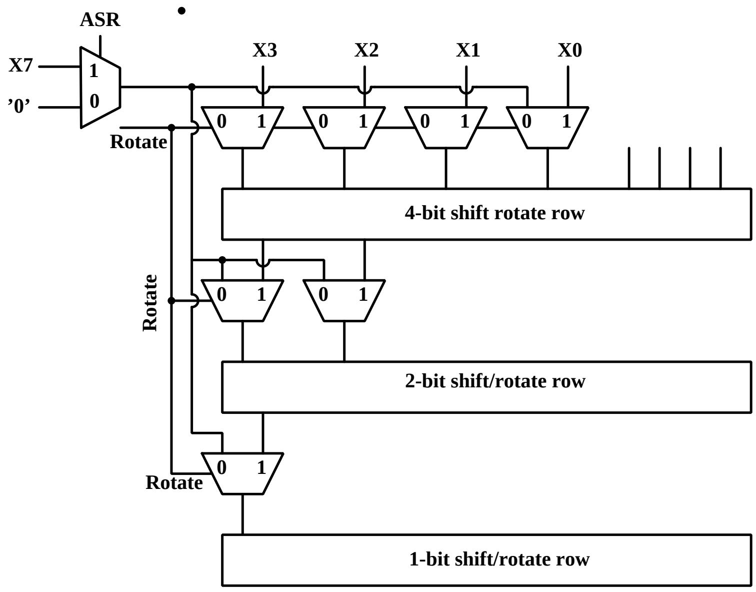


Figure 3: Combined Right Rotate/Shift for an 8 bit operand

3.6 Bidirectional Shift and Rotate Operations

It is possible to use the same hardware for left and right shift/rotate operations. This can be

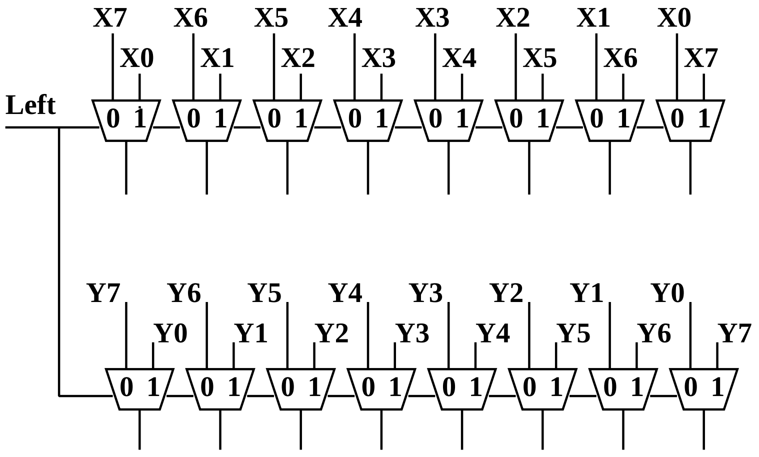


Figure 4: Bit reversal for bidirectional Shift and Rotate Operations

done by adding rows of muxes at the input and output which reverse the order of bits.

We can also make use of the fact that a left rotate by n places is the same as a right rotate by $2^n - n$ places. Now $2^n - n$ is just the 2's complement of n. By presenting the 2's complement of n at the mux controls, we can convert a right rotate to a left rotate. This can be followed by a mask operation, if a shift operation was required, rather than a rotate.