Lecture 11

Registers

Registers in the central processor

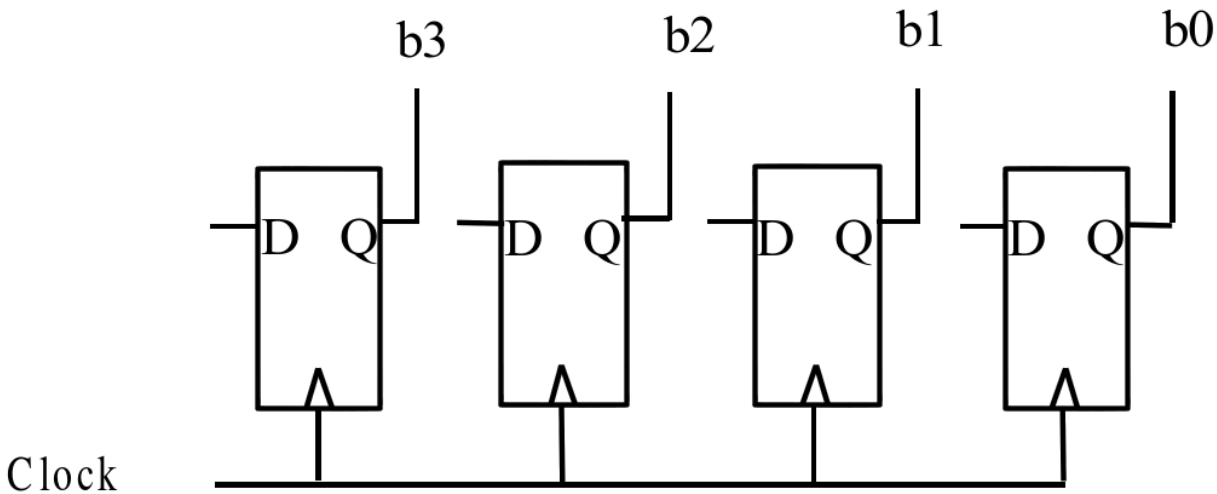
Registers are widely used in computers, and you will have met (or will meet) them in the architecture course:

- Address Register
- Data Registers
- Programme Counter
- Stack Pointer
- etc.

The State Register

We have also used a register in our synchronous design method to store the state of a finite state machine.

Here we used the fact that a set of n flip flops can store an unsigned integer up to 2^n.



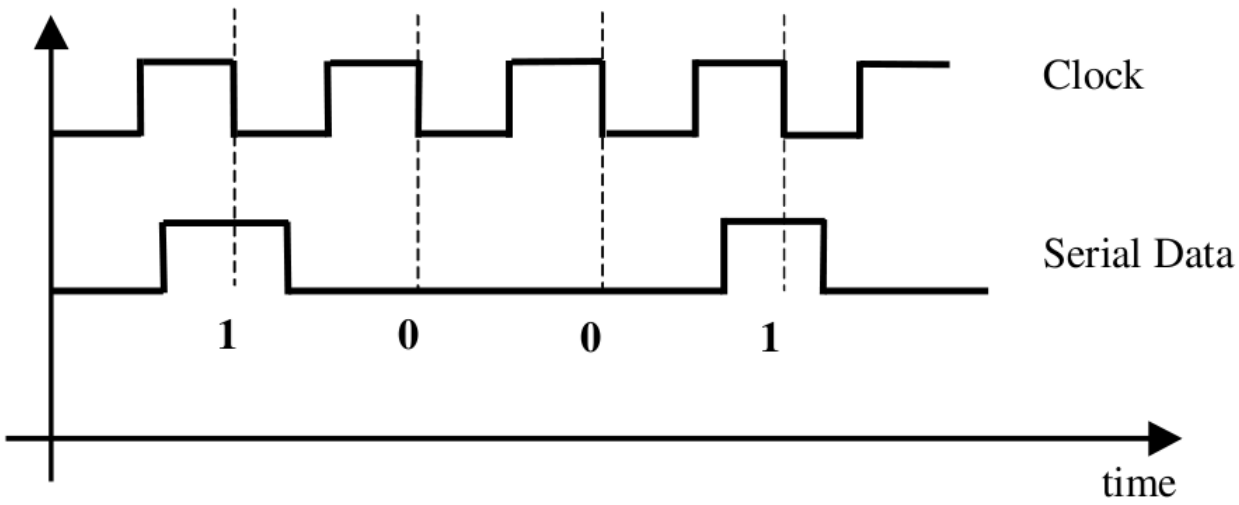
Parallel Data

- Inside a computer data is organised in a parallel form.
- Thus a data register containing say 32 flip flops will have a common clock, and all 32 bits will be set at the same time
- However, for communications, serial data is used in which the bits of a 32 bit word are sent one after the other.

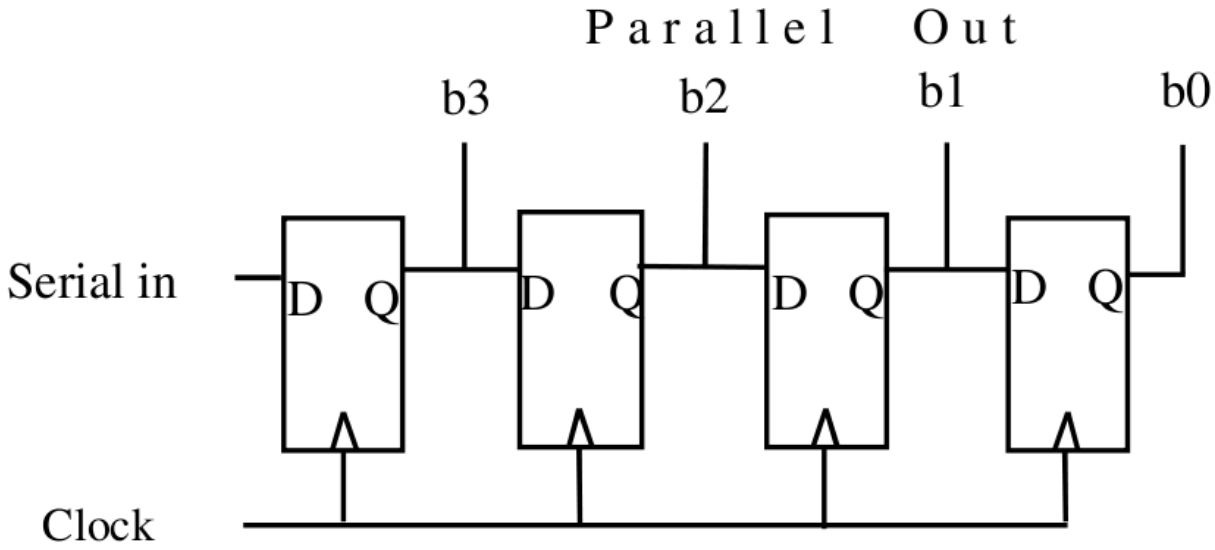
Serial to Parallel Conversion

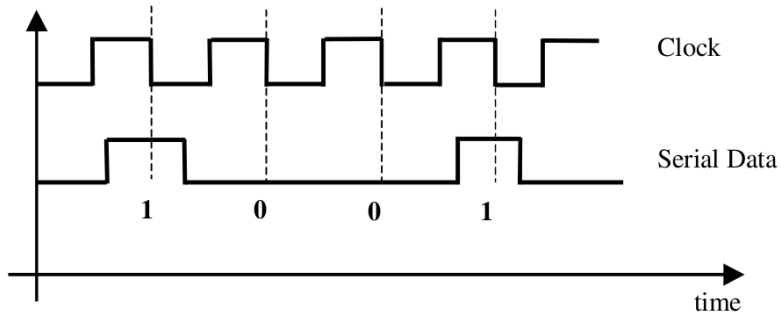
Registers are used to convert data from serial form to parallel form.

Each successive bit is read on a falling clock edge.

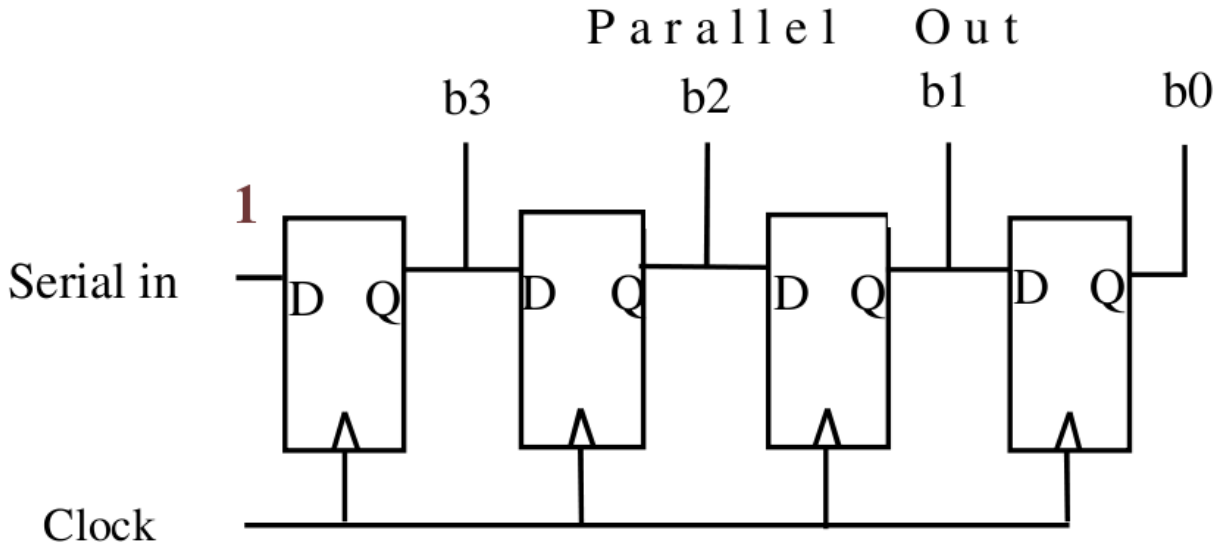


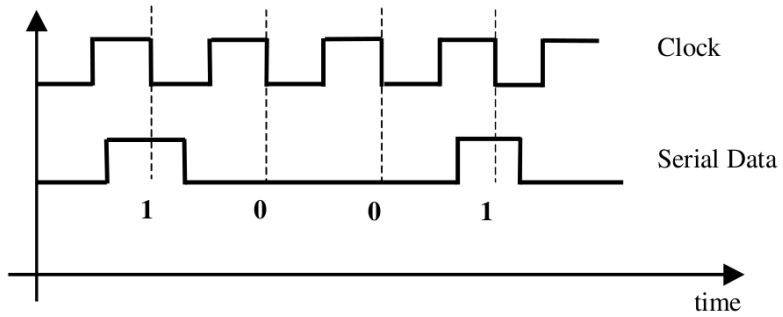
A four bit serial to parallel convertor



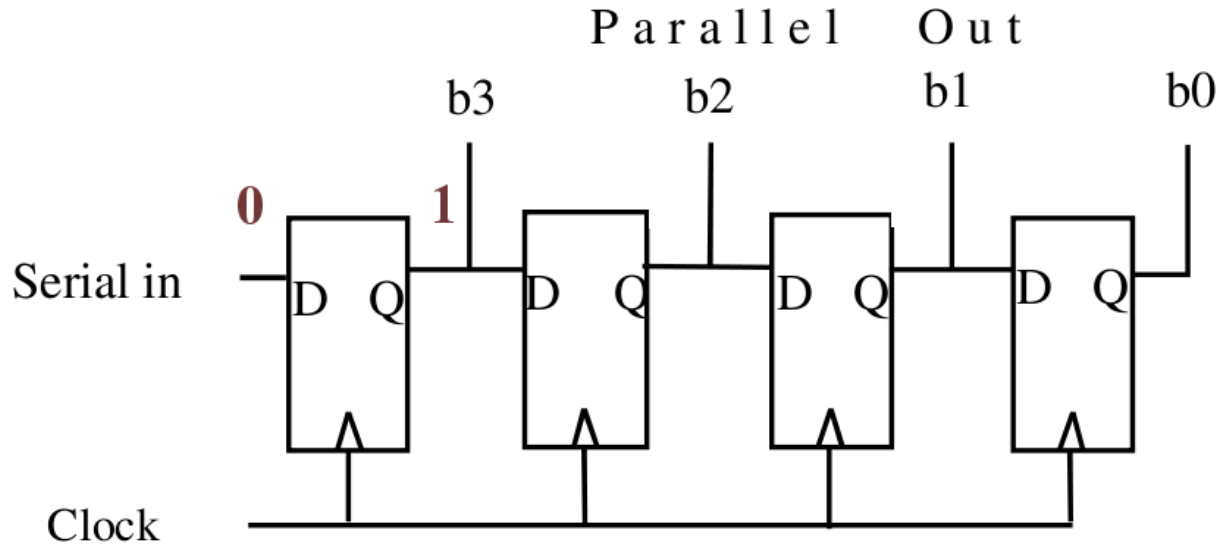


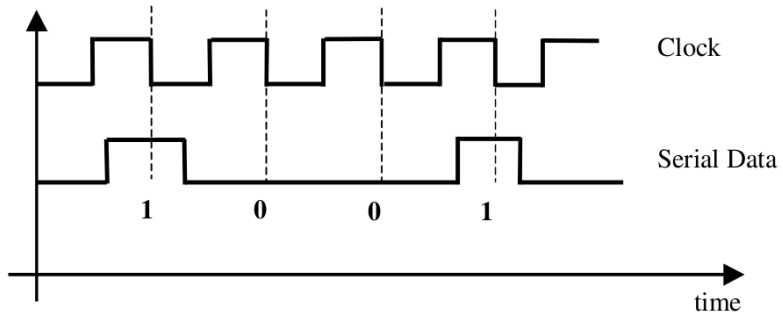
A four bit serial to parallel convertor



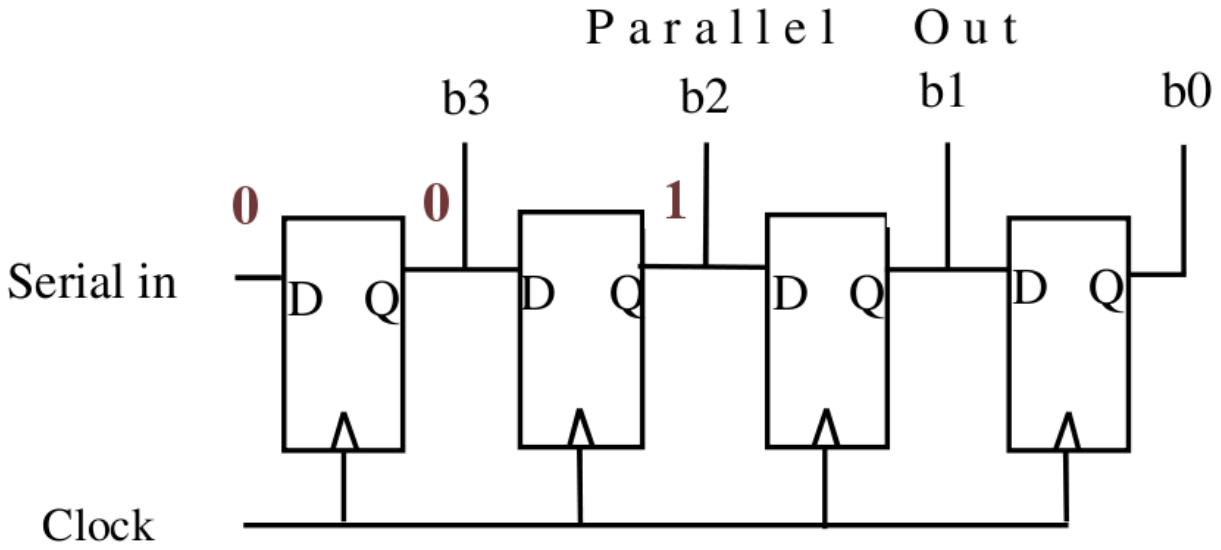


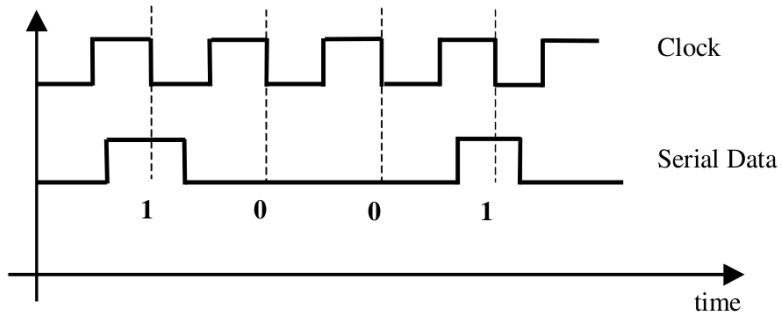
A four bit serial to parallel convertor



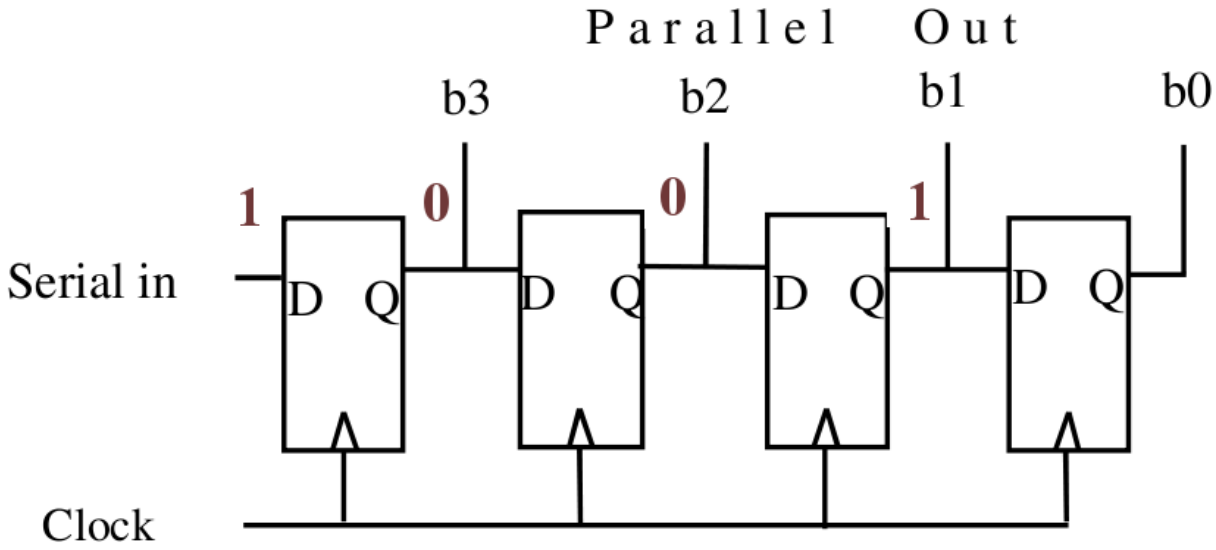


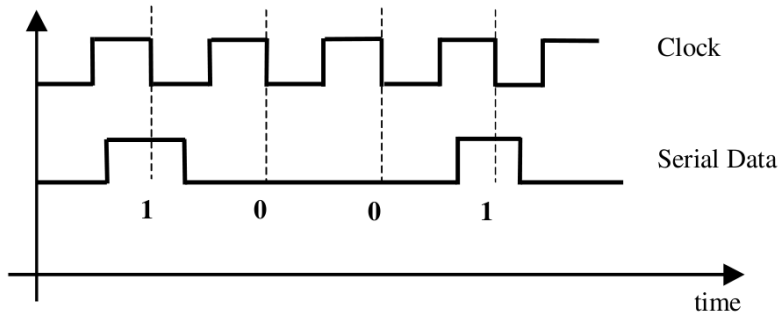
A four bit serial to parallel convertor



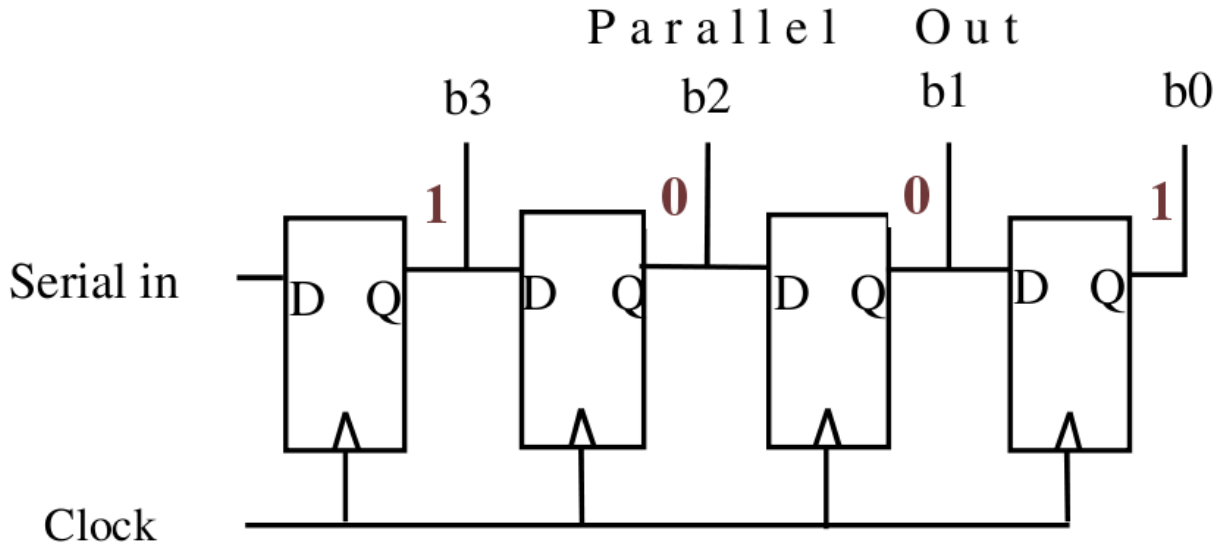


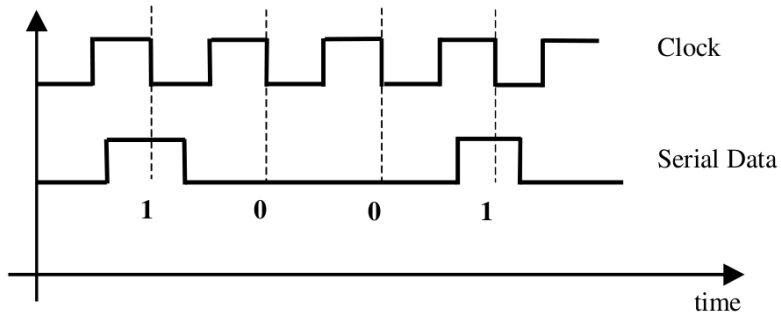
A four bit serial to parallel convertor





A four bit serial to parallel convertor





Timing serial Input

- Note that the length of time taken to load serial data will depend on the length of the shift register.
- In practice serial data is loaded at a much slower rate than the processor clock, and a separate clock is used for the purpose.
- Synchronisation with the main processor is achieved using other control lines.

Parallel to Serial Conversion

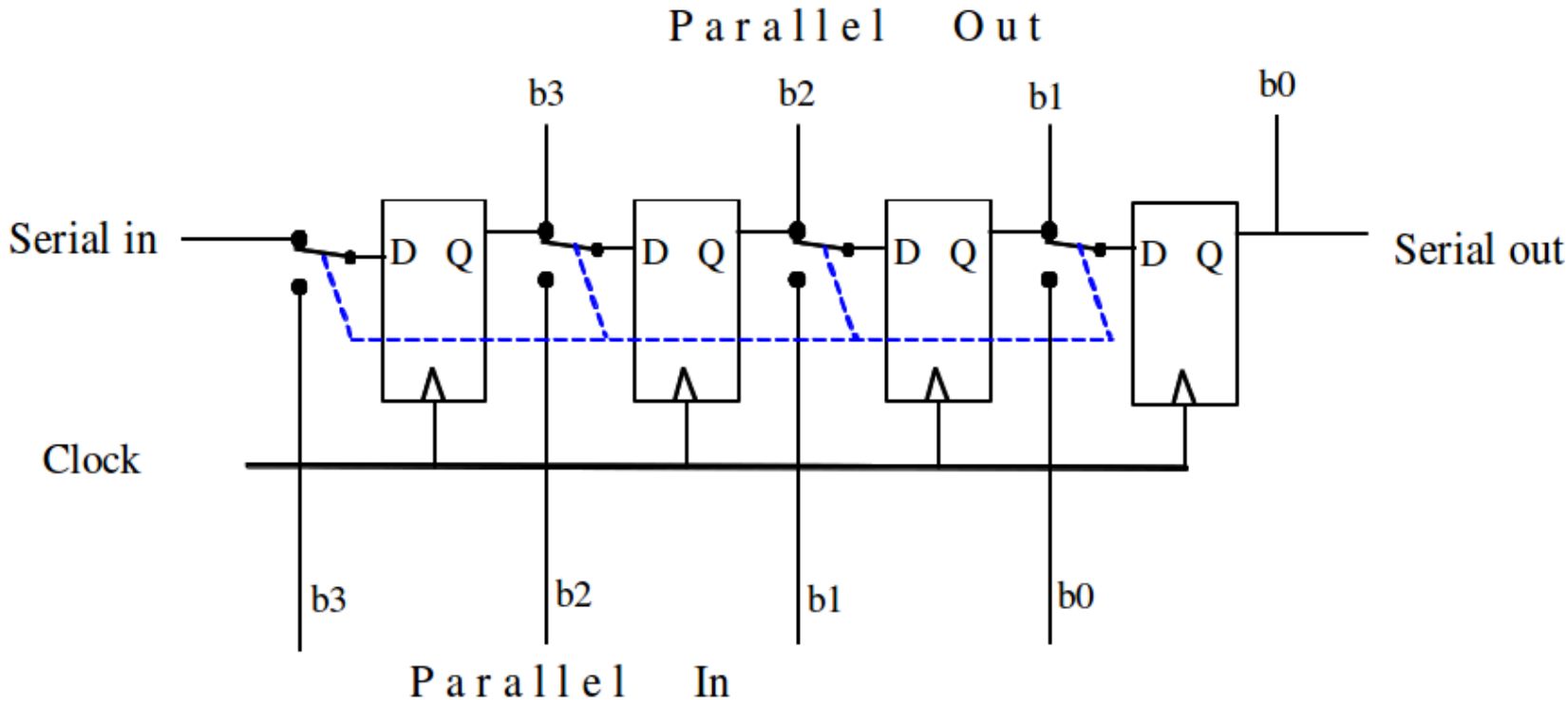
This is carried out in two stages:

1. Load parallel data on to the D-type flip flops
2. Shift the data out in serial form

This means we must somehow switch the inputs of the D-Q flip flops

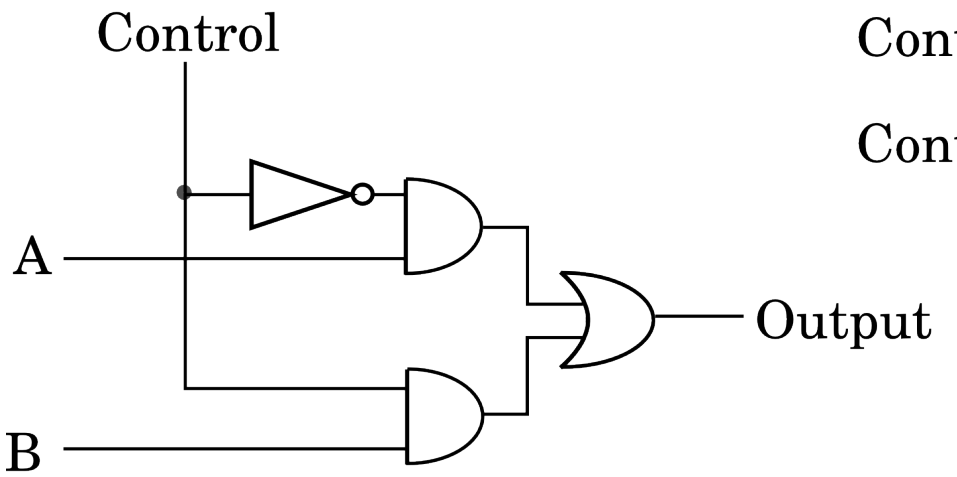
Parallel to Serial Conversion

The inputs to each flip flop are changed to change the function from parallel load to serial input/output.



The multiplexer

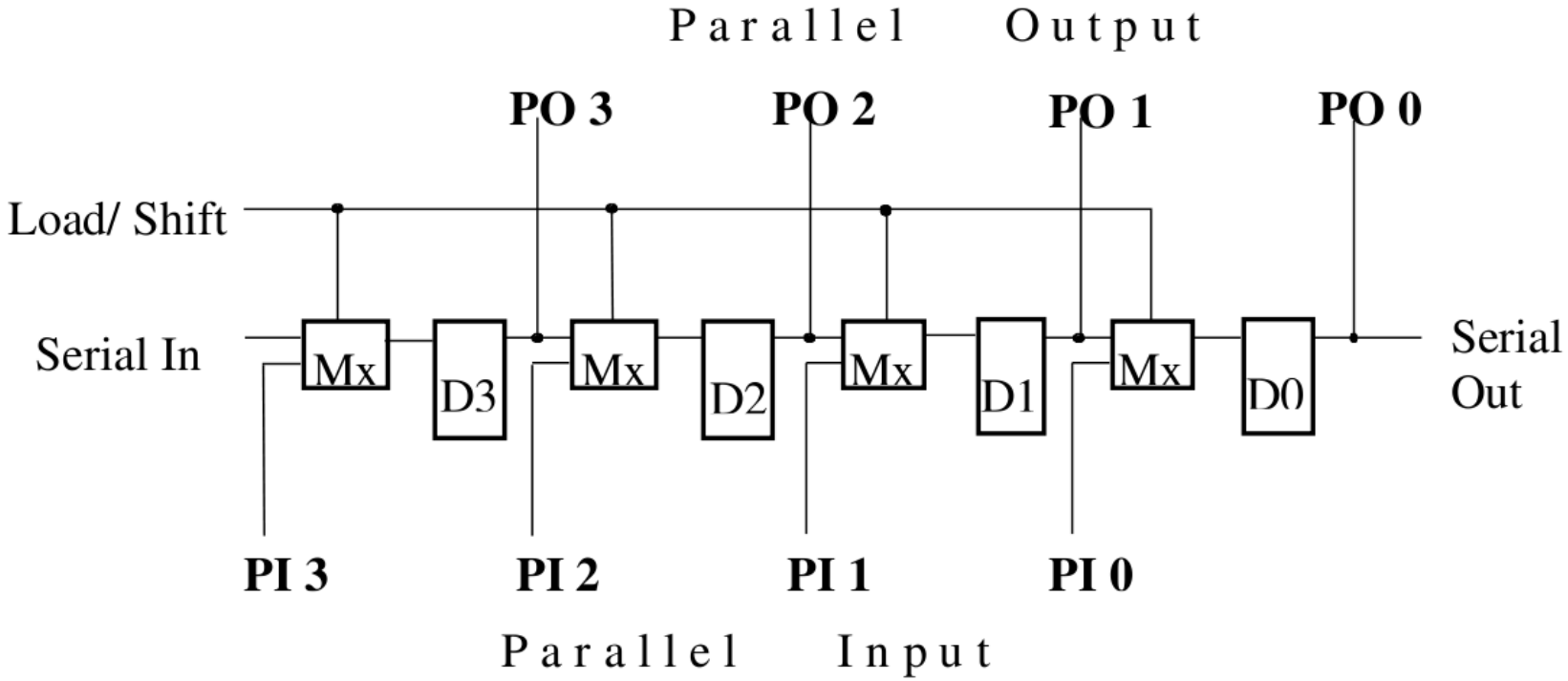
Obviously we will not use a mechanical switch, instead we will use our old friend the multiplexer.



Control=0: Output follows A

Control=1: Output follows B

Four bit shift register



Multiplying and Dividing by 2

- Multiplying by 2 in binary arithmetic is equivalent to shifting the bits of a number one place to the left and filling the bottom bit with a 0.
- This can be done with a shift register.
- Similarly, divide by 2 can be done by shifting right one space and discarding the bottom bit.

Problem Break

- What possible errors can you get using shift left to multiply a number by 2?
- When you use shift right to divide a number by 2, what value do you put in the top bit?

Multi Function registers

We saw how to make a register perform two functions:

1. Parallel Load
2. Serial Input or Output

We can extend this concept to registers that perform the arithmetic shifts

Four function shift register

Our next example will be a shift register with the following four functions:

- 00 Hold
- 01 Shift Right
- 10 Shift Left
- 11 Parallel Load

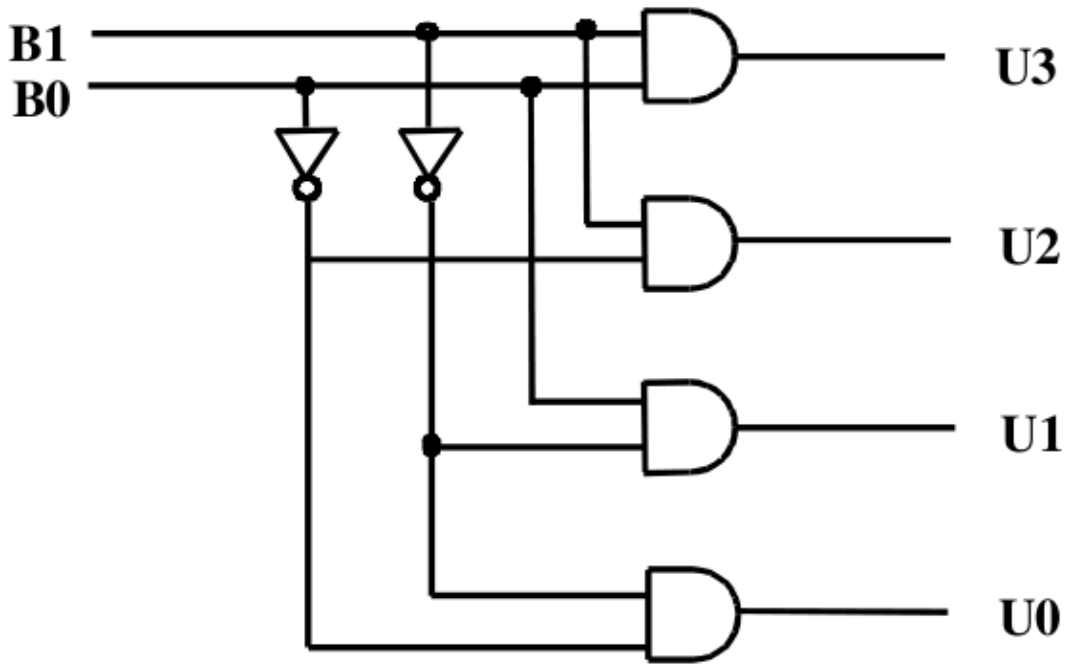
It will be controlled by a two bit binary number

Four Way Multiplexer

- We can give a shift register four functions by using a four way multiplexer (switch) to select the D input to the flip-flops.
- Designing a four way multiplexer is similar to the two way multiplexer, but we will do it in two stages to generalise the design process.

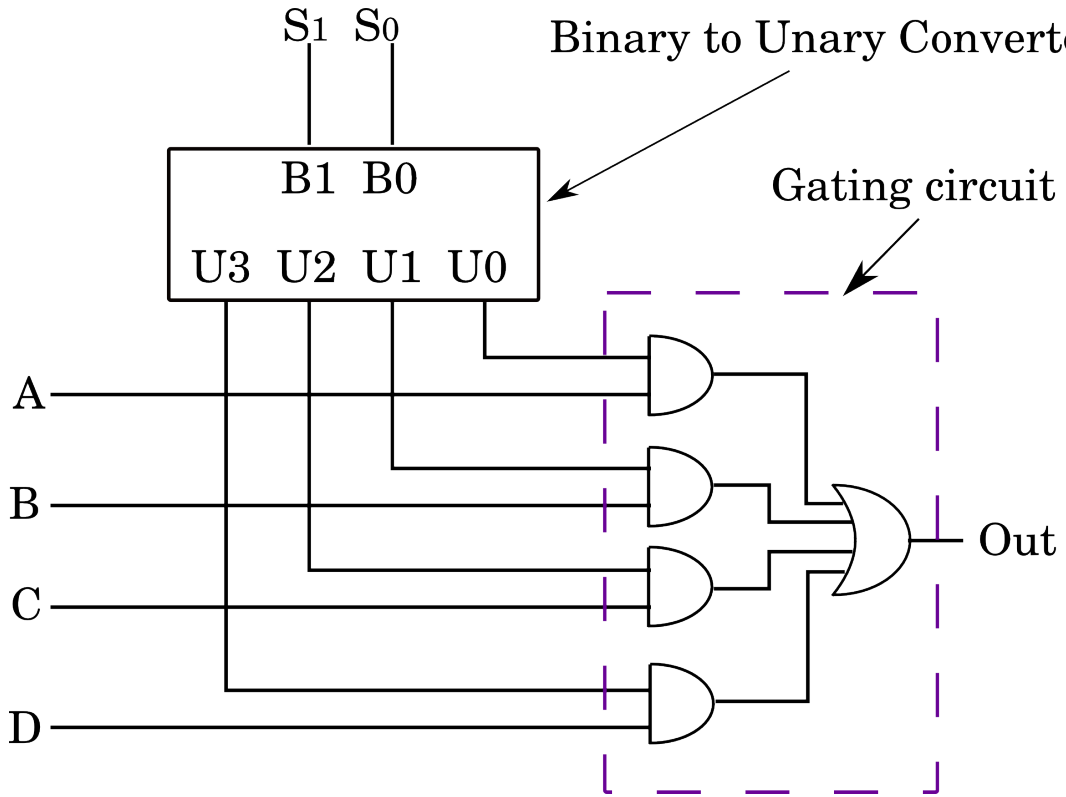
Binary to Unary Converter or Decoder

The input is a two bit binary number. The output is its unary representation. For a given input, only one output is non zero.



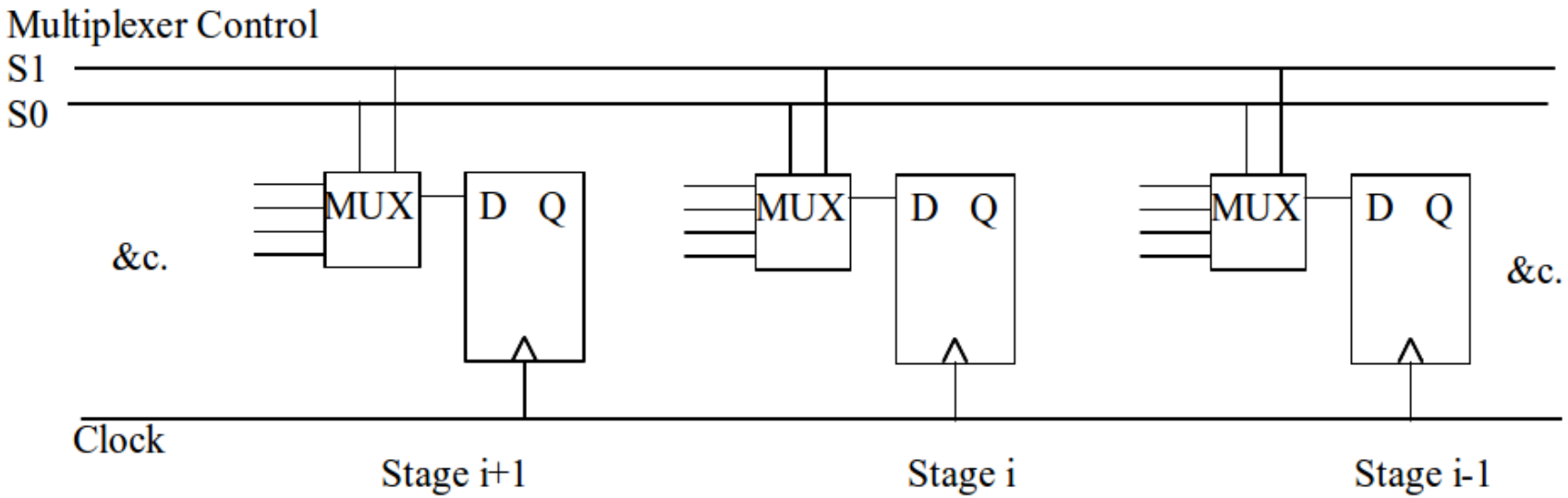
Four way multiplexer

Given a binary to unary converter, the multiplexer can be built trivially by providing a gating circuit.

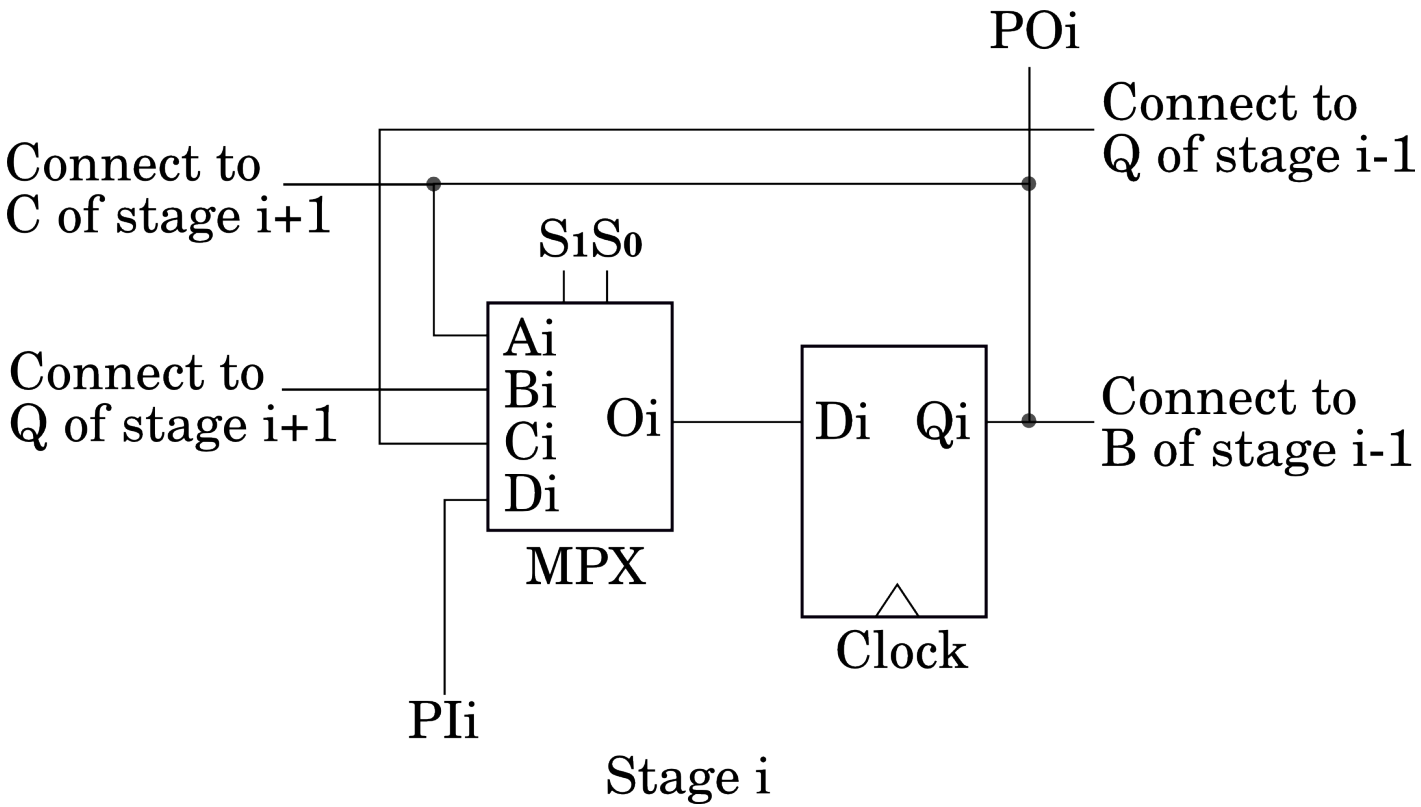


Connecting up the shift register

We can make the shift register as long as we like. The clock and the selection bits are common to every stage.

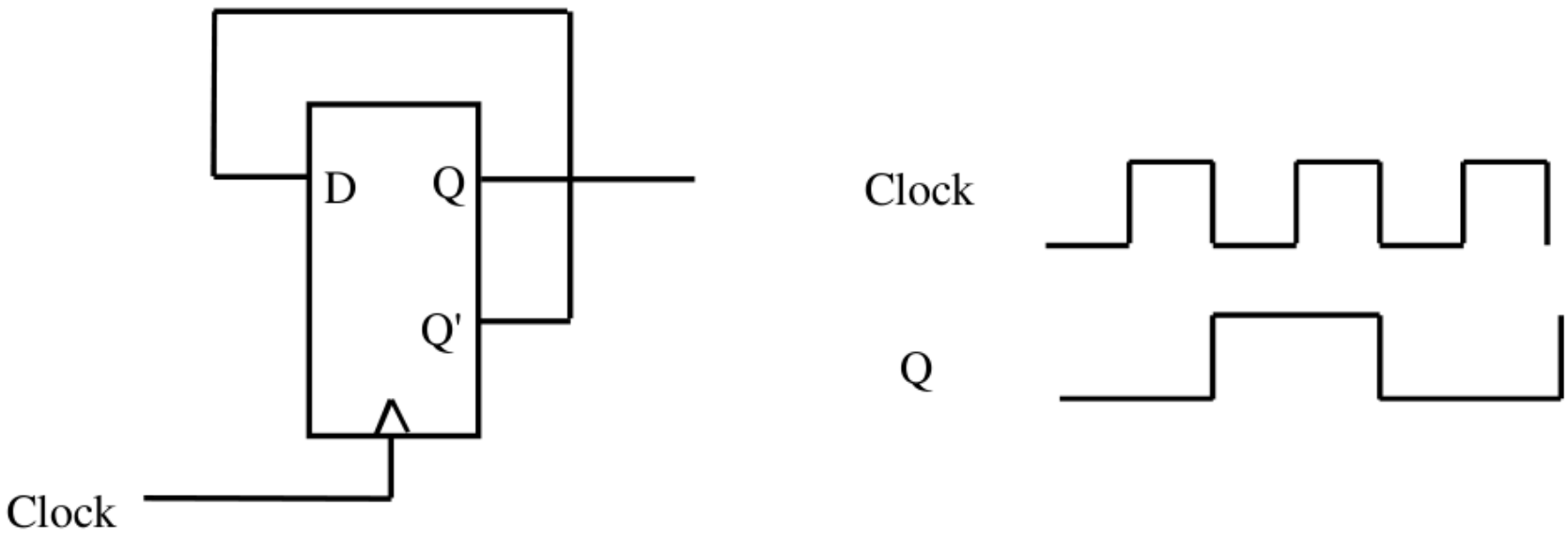


We connect individual stages as follows



Clock Dividers

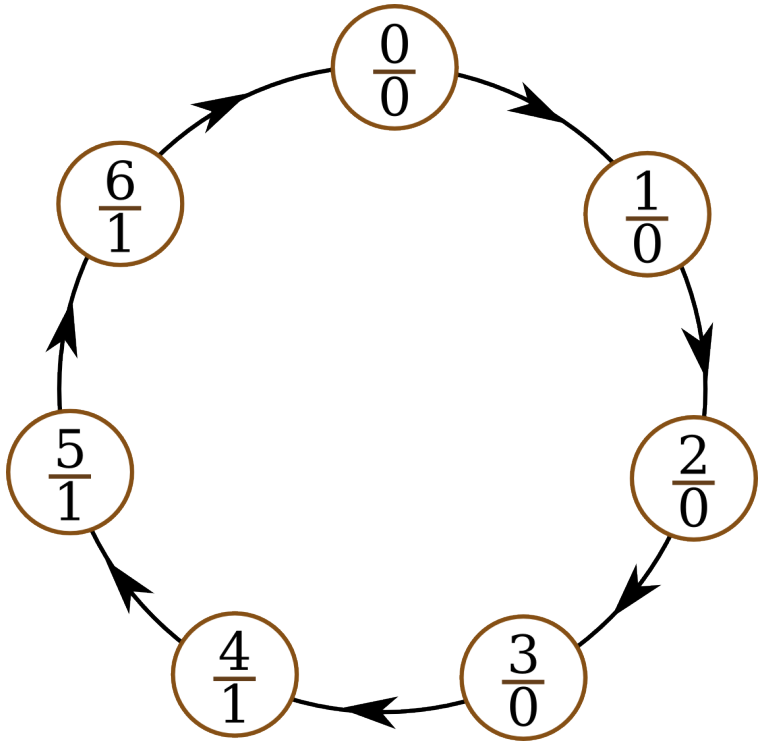
Clock dividers form an interesting use of registers. Synchronous divide by 2 is done as follows:



Divide by an integer

A synchronous divide by any integer can be easily designed using our methodology.

For example, divide by 7 can be done with the finite state machine opposite.



Clocks and Watches

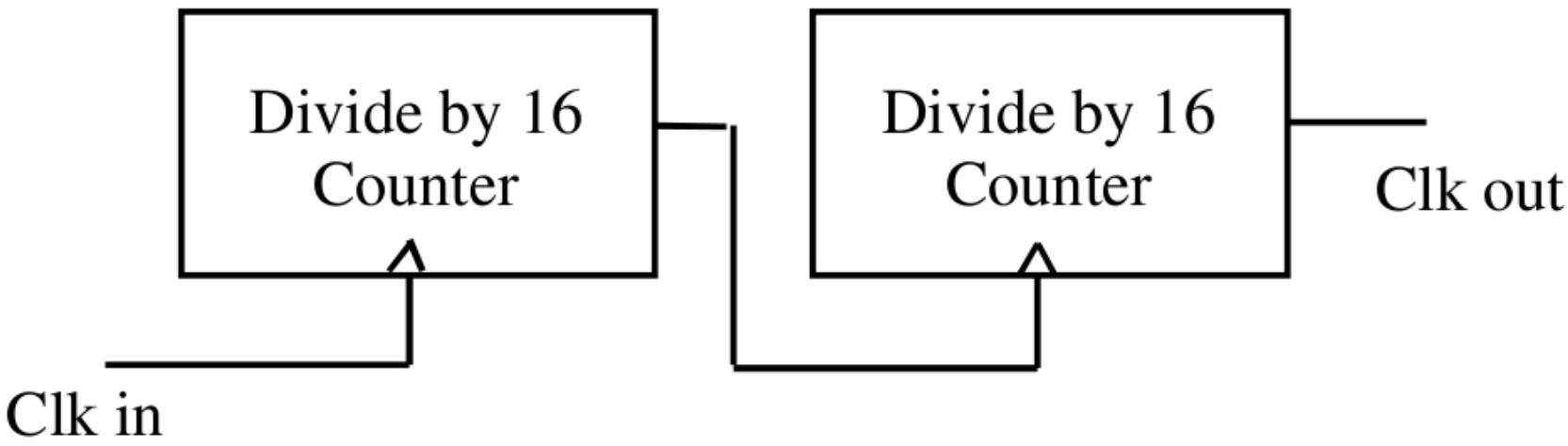
- In practice we may need to divide by much larger numbers. Consider a wrist watch:
- A regulating crystal produces a steady waveform of 1MHz. (That is 10^6 falling edges per second)
- A watch stepper motor requires 1 pulse per second to drive it.
- Hence we need to divide by 1,000,000

Clocks and Watches

- We could use a synchronous circuit to divide by 10^6, but it would require a synchronous counter with 10^6 states, and therefore 20 D-Q flip-flops!
- This is difficult to design, and so clock dividing is often done in stages

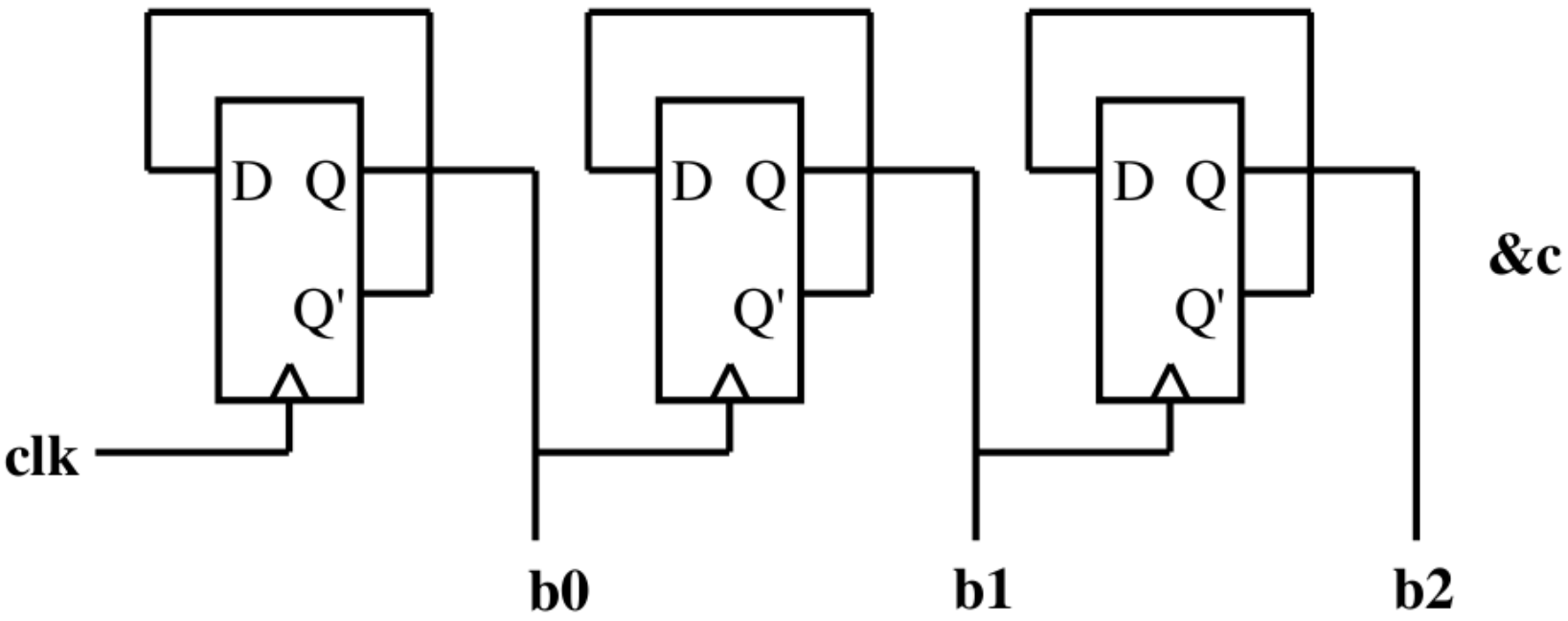
Divide by 256

Dividing by 16 can be easily designed synchronously.
Cascading two dividers gives us a divide by 256.



The ripple through counter

This delightfully simple counter can be arbitrarily large, but it is not synchronous.



Using a counter to divide by any number

- Suppose that we wish to divide a clock by a number which is not a power of two.
- The first step is to design a counter to the next higher power of 2.
- In a simple example, if we wish to divide by 5, we first design a counter to count to 8

Using a counter to divide by any number

- Recall that when we designed the flip flops we included a clear input which set the output $Q = 0$.
- For example we can let a circuit count 0,1,2,3,4, but as soon as we detect 5 on the output we reset all the Q values to 0.

Asynchronous Divide by 5