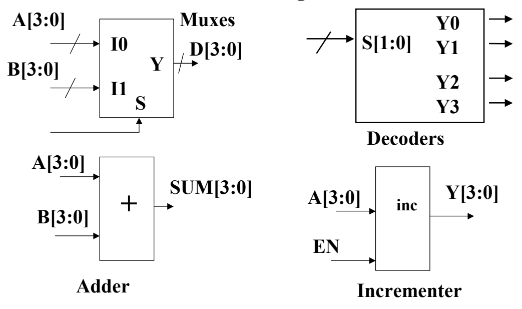
Sequential Building Blocks

Recall the Combinational building blocks we looked at:



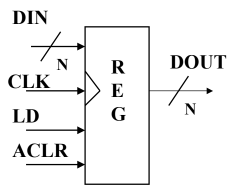
What are some Sequential Building Blocks?

- **Registers : used for holding data.**
- **Counters : used for counting events**
- **Shift Registers : used for serial to parallel, parallel to Serial data conversion**

We will look at implementations of these that combine DFFs with combinational building blocks.

Registers

The most common sequential building block is the register. A register is N bits wide, and has a load line for loading in a new value into the register.

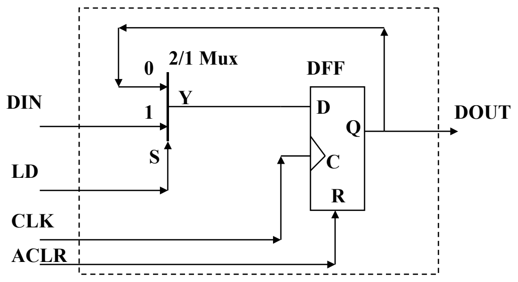


Register contents do not change unless LD = 1 on active edge of clock.

A DFF is NOT a register! DFF contents can change every clock edge.

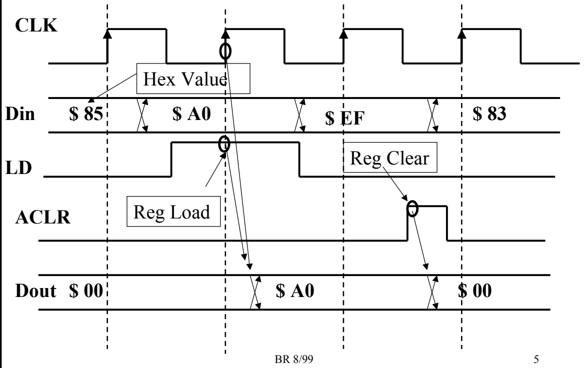
ACLR used to asynchronously clear the register

1 Bit Register using DFF, Mux

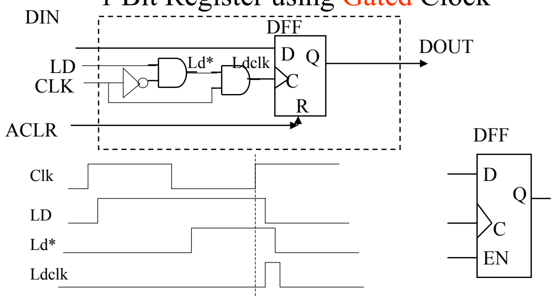


Note that DFF simply loads old value when LD = 0. DFF is loaded every clock cycle.

Register Timing (8 Bit register)



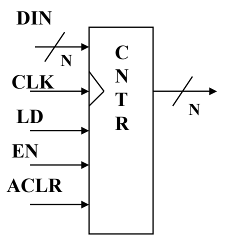
1 Bit Register using Gated Clock



Saves power over previous design since DFF is not clocked every clock cycle. Many FPGAs offer an 'enabled' DFF as an integrated unit. Gating can be optimized at transistor level in 'enabled' DFF.

Counter

Very useful sequential building block. Used to generate memory addresses, or to count events (e.g., count the number of times a datapath operation is performed).

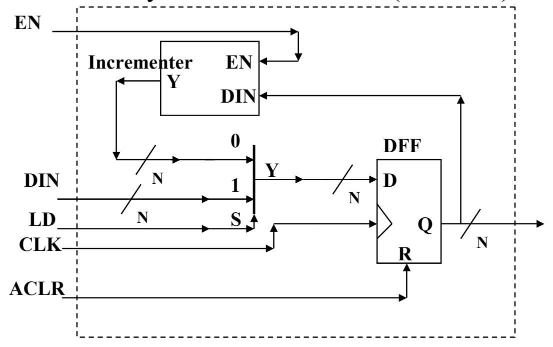


LD asserted loads counter with DIN value.

EN asserted will increment counter on next active clock edge.

ACLR will asynchronously clear the counter.

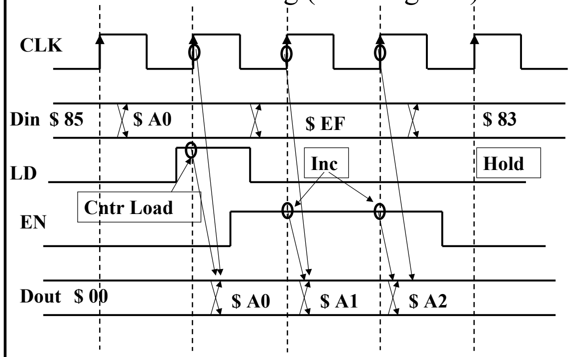
One way to build a Counter (Cntr 'A')



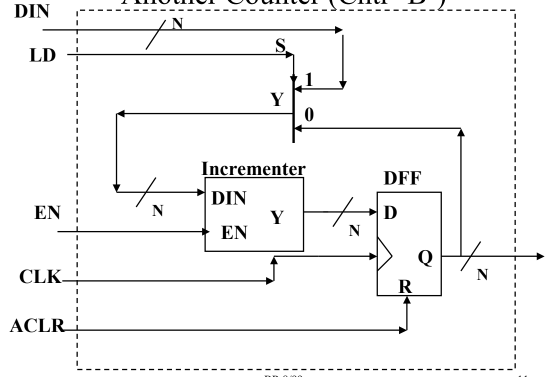
Counter Operation

Counter A						
Aclr	Clk	En	LD	Q	Q+	Op
H	X	X	X	X	0	Async Clr
L	↑	X	H	X	Din	Load
L	↑	H	L	Q	Q+1	Increment
L	X	L	L	Q	Q	Hold

Counter Timing (8 Bit register)



Another Counter (Cntr 'B')



Counter Operation

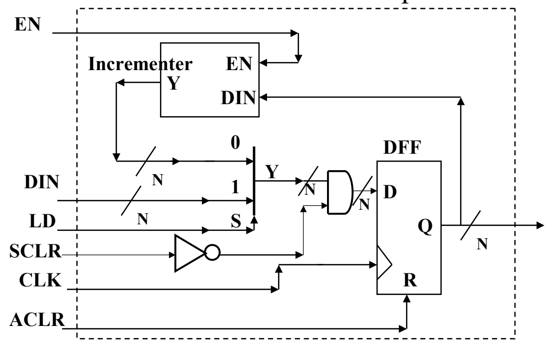
Counter B						
Aclr	Clk	En	LD	Q	Q+	Op
H	X	X	X	X	0	Async Clr
L	↑	L	H	X	Din	Load
L	↑	H	L	Q	Q+1	Increment
L	X	L	L	Q	Q	Hold
L	↑	H	H	Q	Din+1	Load Inc

EN=H, LD=H will load an incremented version of Din

Synchronous vs Asynchronous Clear

- The ACLR line is tied to the asynchronous reset of the DFF
  - Asynchronous clear is independent of clock, will occur anytime clear is asserted
  - Usually tied to Power-On-Reset (POR) circuit
  - Not very useful for normal operation since any glitch on ACLR will clear the counter
- Would like a Synchronous Clear input (SCLR) in which the clear operation takes place on the next active clock edge.

Cntr 'A' with SCLR Input

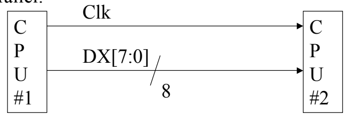


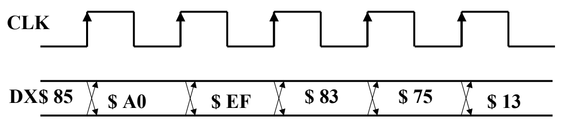
Counter Operation

Counter A with SCLR							
Aclr	Sclr	Clk	En	LD	Q	Q+	Op
H	X	X	X	X	X	0	Async Clr
L	H	↑	X	X	X	0	Sync Clr
L	L	↑	X	H	X	Din	Load
L	L	↑	H	L	Q	Q+1	Increment
L	L	X	L	L	Q	Q	Hold

Parallel Data Transfer

To transfer data between two computers, we can do it in parallel:

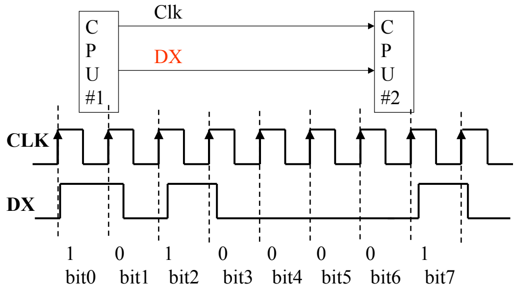




Parallel Data transfer requires a lot of lines to be run between computers; cabling be expensive, and bulky. Not practical for long distances.

Serial Data Transfer

We can transfer data in serial fashion, e.g., one bit at a time.



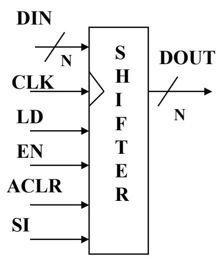
$ 85 = 10000101, data transmitted LSB to MSB

More on Serial Data Transfer?

- Serial data transfer is more common than data parallel communication because less wires than parallel data transfer, can be run longer distances
- Data can be transferred either LSB (least significant bit) to MSB (most significant bit) or vice-versa
  - Most common is LSB to MSB
- To implement serial data transfer we need a sequential building block that is called a SHIFT register.

Shift Register

Very useful sequential building block. Used to perform either parallel to serial data conversion or serial to parallel data conversion.



LD asserted loads register with DIN value.

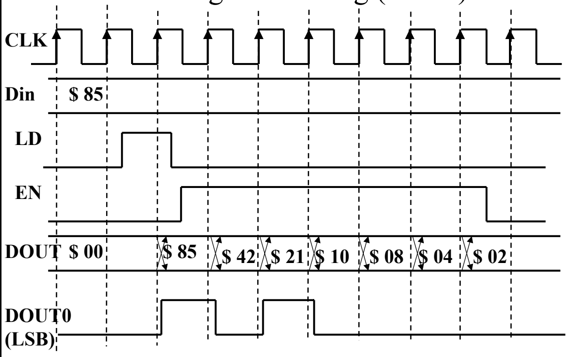
EN asserted will shift data on next active clock edge.

ACLR is async clear.

SI is serial data in.

Look at LSB of DOUT for serial data out.

Shift Register Timing (SI = 0)



Understanding the shift operation

MSB LSB

$85 = 1 0 0 0 0 1 0 1
SI = 0

$42 = 0 1 0 0 0 0 1 0 **1st right shift**
SI = 0

$21 = 0 0 1 0 0 0 0 1 **2nd right shift**
SI = 0

$10 = 0 0 0 1 0 0 0 0 **3rd right shift**

Etc….

Right Shift vs Left Shift

A right shift is MSB to LSB

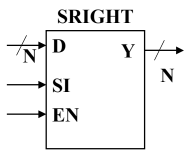
In: D7 D6 D5 D4 D3 D2 D1 D0
SIN
Out: SIN D7 D6 D5 D4 D3 D2 D1

A left shift is LSB to MSB

In: D7 D6 D5 D4 D3 D2 D1 D0
SI
Out: D6 D5 D4 D3 D2 D1 D0 SI

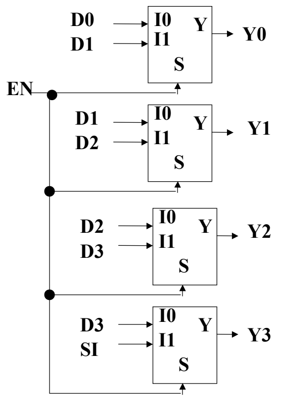
Combinational Right Shifter

We need a combinational block that can either shift right or pass data unchanged



When EN = 1, Y = D shifted right by 1 position.

When EN=0, Y = D



4-bit Combinational RIGHT Shifter Implementation

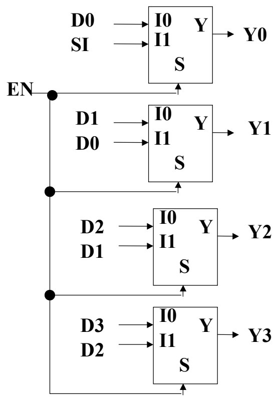
When EN = 0, then:

Y = D3 D2 D1 D0

When EN = 1, then:

Y = SI D3 D2 D1

(right shifted by one position)



4-bit Combinational LEFT Shifter Implementation

When EN = 0, then:

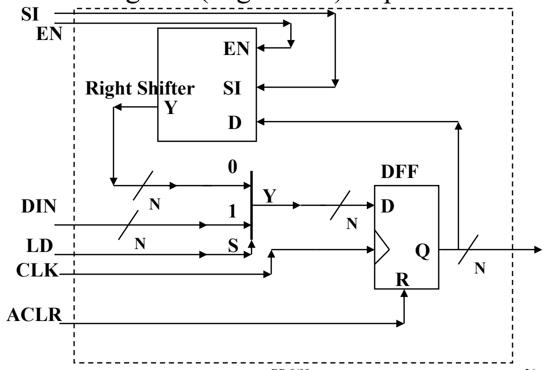
Y = D3 D2 D1 D0

When EN = 1, then:

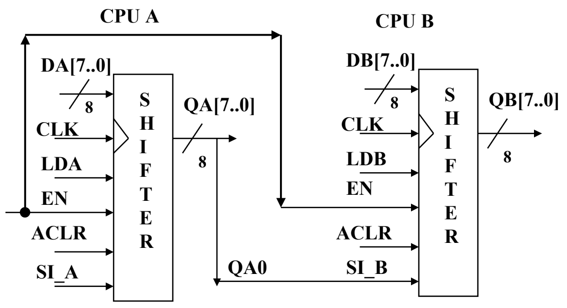
Y = D2 D1 D0 SI

(left shifted by one position)

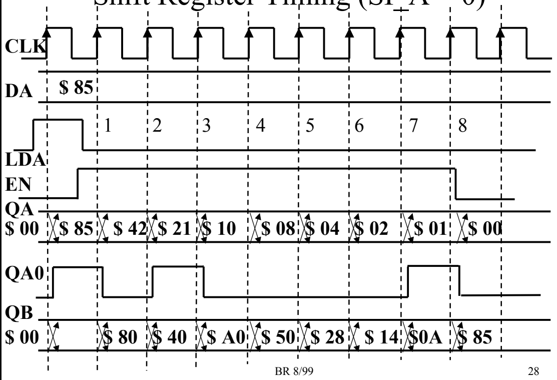
Shift Register (Right shift) Implementation



Serial Communication



Shift Register Timing (SI_A = 0)



Comments on Shift operation

- Took 8 clock cycles to serially send the 8 bits in CPU A to CPU B.
- Shift Register at CPU A ended up at $00; Shift Register at CPU B ended up with CPU A value ($85)
- Initial contents of CPU B shift register does not matter
- Data shifted out LSB to MSB from CPUA to CPUB. Note that data enters the MSB at CPUB and progresses toward the LSB.