YORK
UNIVERSITÉ
UNIVERSITY

EECS 3201: Digital Logic Design Lecture 16

Ihab Amer, PhD, SMIEEE, P.Eng.

Forget EECS 3201 – Lets try to solve a "real problem" ☺

INPUT:
dirty laundry

Device: Washer

Function: Fill, Agitate, Spin

$Washer_{PD}$ = 30 mins

OUTPUT:
6 more weeks

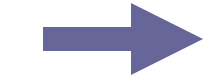
Device: Dryer

Function: Heat, Spin

$Dryer_{PD}$ = 60 mins

One Load at a Time

Non-Engineers would do that



Step 1:

Step 2:

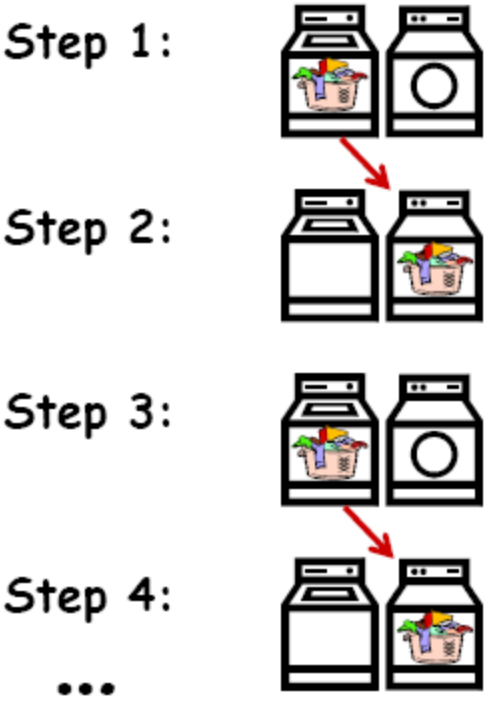
$$\text{Total} = \text{Washer}_{PD} + \text{Dryer}_{PD}$$

$$= \underline{\quad 90 \quad} \text{ mins}$$

Not a smart idea!

Doing N Loads of Laundry

Still with the non-engineers "combinational" way

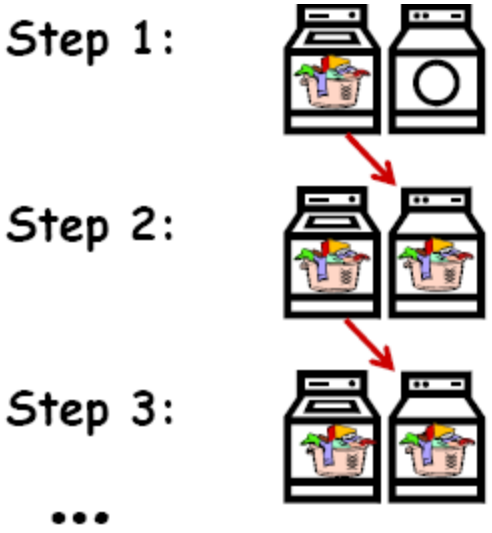


$$\text{Total} = N*(\text{Washer}_{PD} + \text{Dryer}_{PD})$$

$$= \underline{\quad N*90 \quad} \text{ mins}$$

This is how engineers would do it

So, engineering sense, sometimes, makes sense!



Actually, it's more like N*60 + 30 if we account for the startup transient correctly. When doing pipeline analysis, we're mostly interested in the "steady state" where we assume we have an infinite supply of inputs.

$$\text{Total} = N * \text{Max}(\text{Washer}_{PD}, \text{Dryer}_{PD})$$

$$= \underline{\quad N*60 \quad} \text{ mins}$$

Some Definitions (1/2)

Latency

The *delay* from when an input is established until the output associated with that input becomes valid

Non-Engineers' Laundry = *90* mins

Engineers' Laundry = *120* mins

At steady state

Throughput

The *rate* of which inputs or outputs are processed

Non-Engineers' Laundry = *1/90* outputs/mins

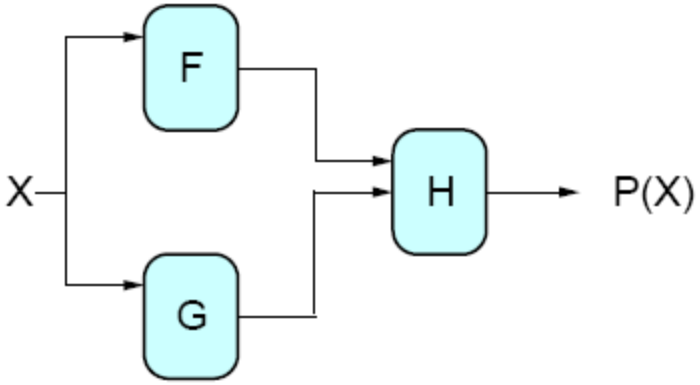
Engineers' Laundry = *1/60* outputs/mins

Some Definitions (2/2)

Pipelining

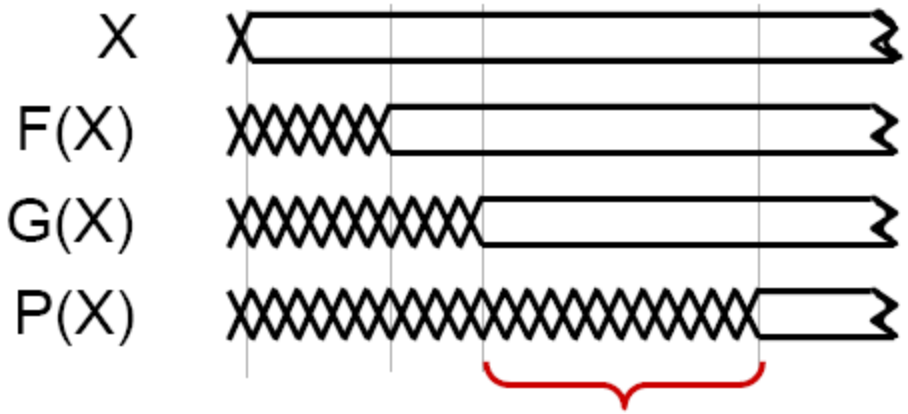
Break task into stages, each stage outputs data for next stage, all stages operate concurrently (if they have data)

Ok… Back to Digital Logic ☹



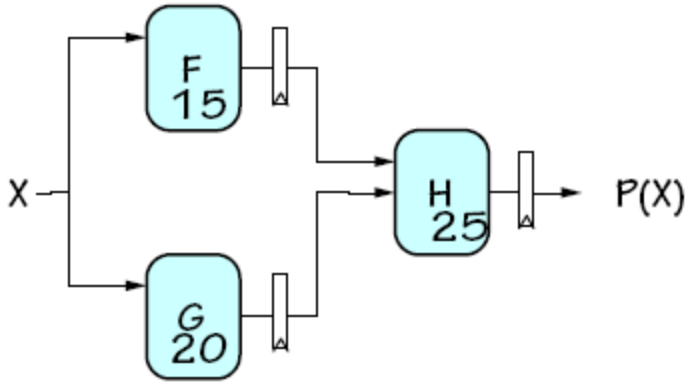
For combinational logic:
latency = t_{PD},
throughput = $1/t_{PD}$.

We can't get the answer faster, but are we making effective use of our hardware at all times?



F & G are "idle", just holding their outputs stable while H performs its computation

Pipelined Circuits – Use registers to hold H's input stable

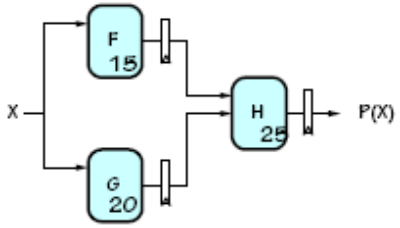


Now F & G can be working on input X_{i+1} while H is performing its computation on X_i. We've created a 2-stage *pipeline*: if we have a valid input X during clock cycle j, P(X) is valid during clock j+2.

Suppose F, G, H have propagation delays of 15, 20, 25 ns and we are using ideal zero-delay registers:

	latency	throughput
unpipelined	45	1/45
2-stage pipeline	50	1/25
	worse	better

Pipeline Diagrams



Clock cycle →

Pipeline stages ↓

	i	i+1	i+2	i+3	
Input	X_i	X_{i+1}	X_{i+2}	X_{i+3}	…
F Reg		$F(X_i)$	$F(X_{i+1})$	$F(X_{i+2})$	…
G Reg		$G(X_i)$	$G(X_{i+1})$	$G(X_{i+2})$	
H Reg			$H(X_i)$	$H(X_{i+1})$	$H(X_{i+2})$

The results associated with a particular set of input data moves *diagonally* through the diagram, progressing through one pipeline stage each clock cycle.

Pipeline Conventions

DEFINITION:

a *K-Stage Pipeline* ("K-pipeline") is an acyclic circuit having exactly K registers on *every* path from an input to an output.

a COMBINATIONAL CIRCUIT is thus an *O*-stage pipeline.

CONVENTION:

Every pipeline stage, hence every K-Stage pipeline, has a register on its *OUTPUT* (not on its input).

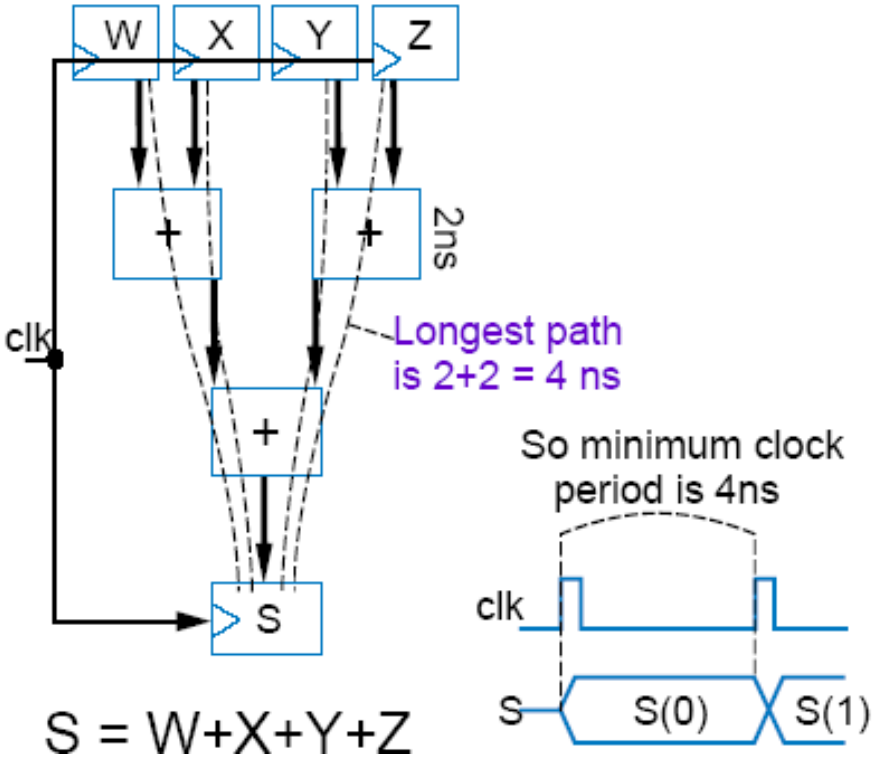
ALWAYS:

The CLOCK common to all registers must have a period sufficient to cover propagation over combinational paths PLUS (input) register t_{PD} PLUS (output) register t_{SETUP}.

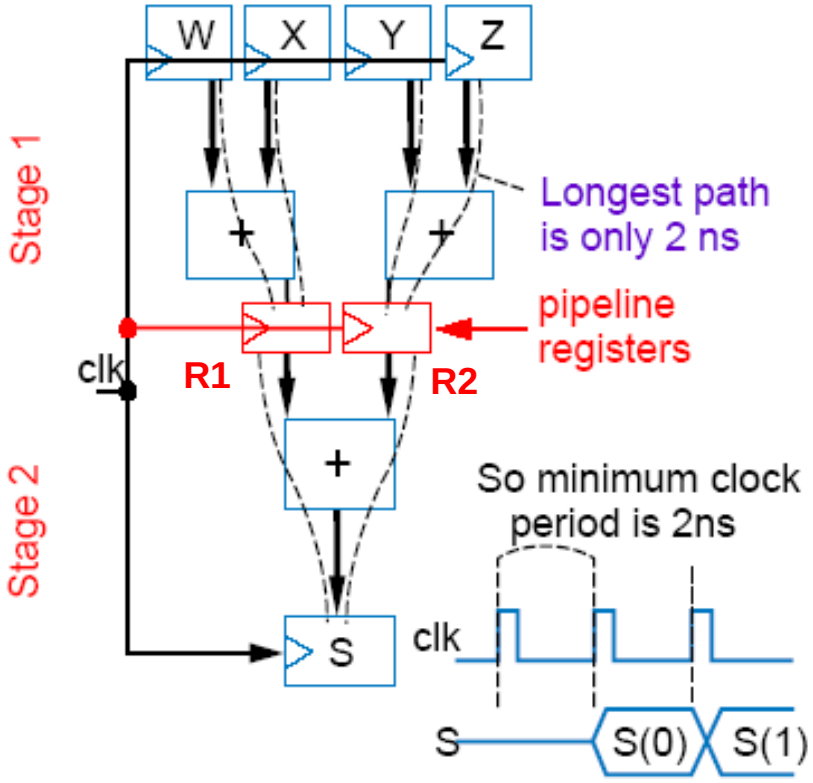
The LATENCY of a K-pipeline is K times the period of the clock common to all registers.

The THROUGHPUT of a K-pipeline is the frequency of the clock.

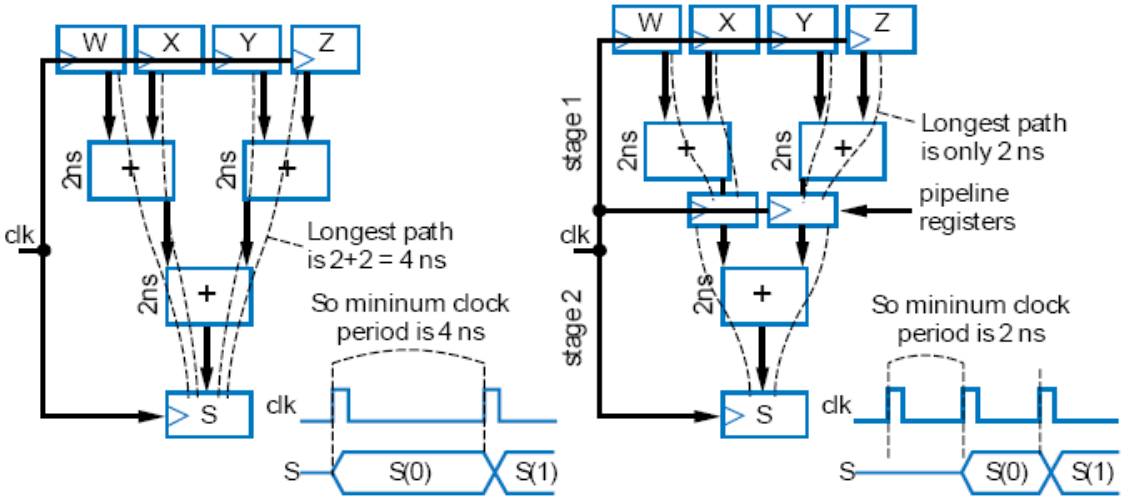
Pipelining Example (1/5)



S = W+X+Y+Z



Pipelining Example (2/5)

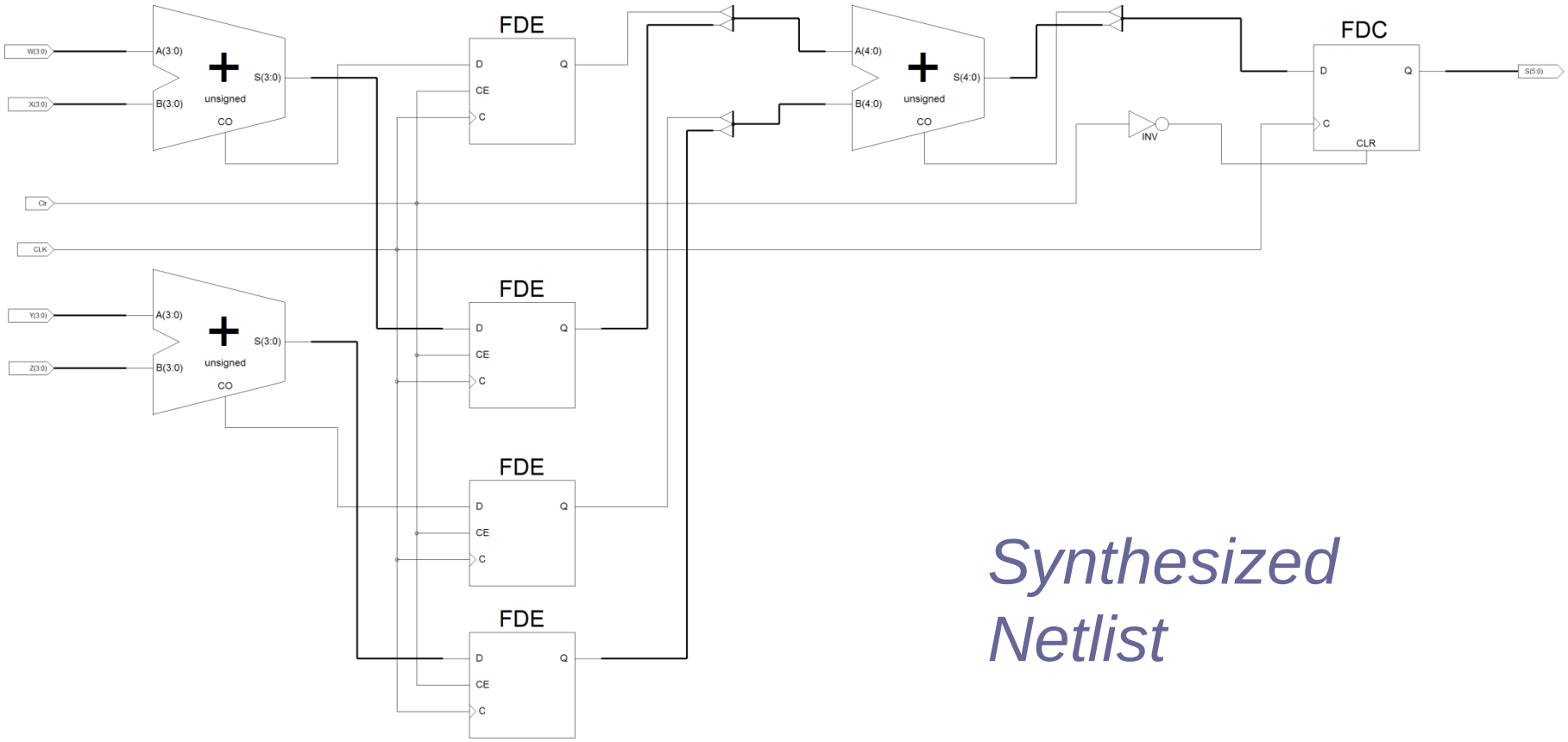


- Datapath on left has *critical path* of 4 *ns*, so shortest clock period is 4 *ns*
  - Can read new data, add, and write data to *S*, every 4 *ns*
- Datapath on right has *critical path* of only 2 *ns*
  - So can read new data every 2 *ns* – *sort of doubled performance*
- So Pipelining the above system
  - Doubled the *throughput*, from 1 item / 4 *ns*, to 1 item / 2 *ns*
  - Latency stayed the same: 4 ns (sometimes it may increase)

Pipelining Example (3/5)

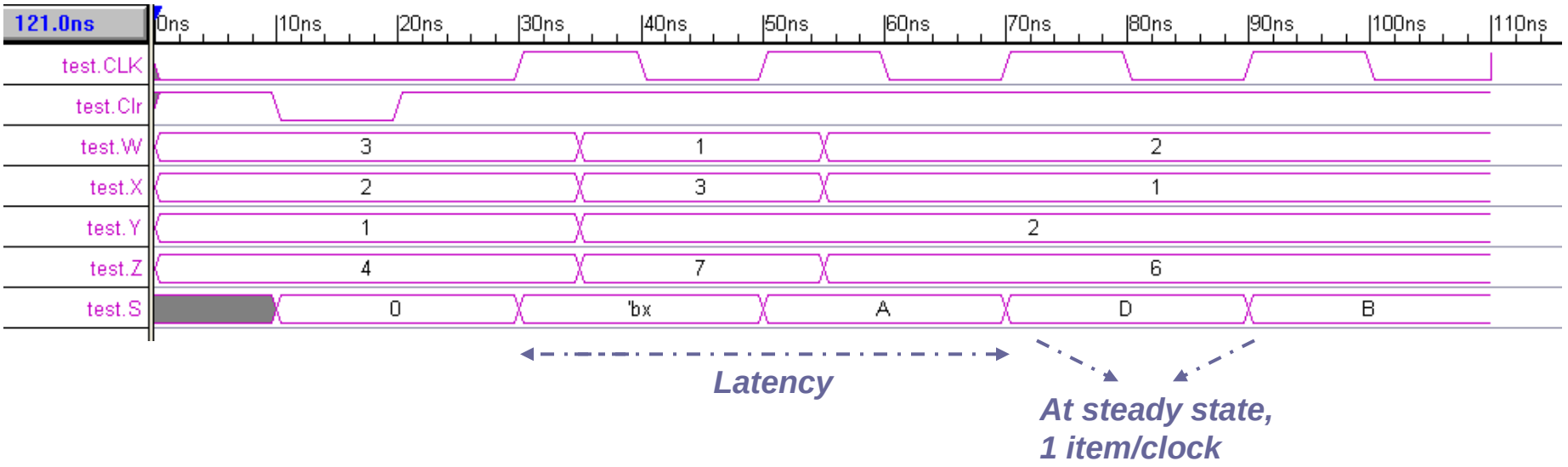
```
module pipeline(W, X, Y, Z, CLK, Clr, S);
  input CLK, Clr;
  input [3:0] W; input [3:0] X; input [3:0] Y; input [3:0] Z;
  output [5:0] S;
  reg [5:0] S;
  reg [4:0] R1;
  reg [4:0] R2;
  always @ (posedge CLK or negedge Clr)
    if (~Clr) S <= 6'b000000;
    else
        begin
            R1 <= W + X;
            R2 <= Y + Z;
            S <= R1 + R2;
        end
endmodule
```

Pipelining Example (4/5)



Synthesized Netlist

Pipelining Example (5/5)



Simulation

Pipelining Summary

- Advantages:
  - Allows us to increase throughput, by breaking up long combinational paths and (hence) increasing clock frequency
- Disadvantages:
  - May increase latency
  - Only as good as the weakest link: slowest step constraints system throughput

References

- MIT Lecture Notes on: http://www.ece.concordia.ca/~asim
- http://www.ics.uci.edu/~harris/ics151/