Spiral 1 / Unit 1

Combinational vs. Sequential Logic

Latency vs. Throughput (Pipelining)

Digital Design Goals

Logic Functions

Spiral Content Mapping

Spiral	Theory	Combinational Design	Sequential Design	System Level Design	Implementation and Tools	Project
1	• Performance metrics (latency vs. throughput) • Boolean Algebra • Canonical Representations	• Decoders and muxes • Synthesis with min/maxterms • Synthesis with Karnaugh Maps	• Edge-triggered flip-flops • Registers (with enables)	• Encoded State machine design	• Structural Verilog HDL • CMOS gate implementation • Fabrication process	
2			• Bistables, latches, and Flip-flops • Counters • Memories			
3						

Outcomes

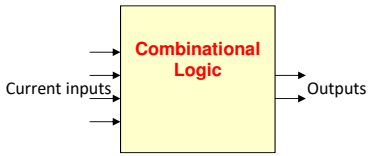
- I know the difference between combinational and sequential logic and can name examples of each.
- I understand latency, throughput, and at least 1 technique to improve throughput
- I can identify when I need state vs. a purely combinational function
  - I can convert a simple word problem to a logic function (TT or canonical form) or state diagram
- I can use Karnaugh maps to synthesize combinational functions with several outputs
- I understand how a register with an enable functions & is built
- I can design a working state machine given a state diagram
- I can implement small logic functions with complex CMOS gates

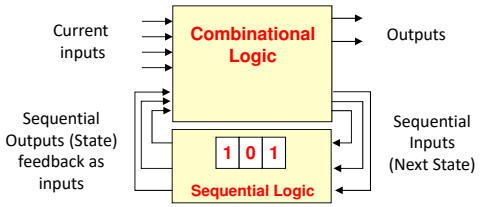
COMBINATIONAL VS. SEQUENTIAL

Combinational vs. Sequential Logic

- All logic is categorized into 2 groups
  - Combinational logic:
    - Outputs = f(_____)
  - Sequential Logic
    - Outputs = f(_____)
    - Sequential logic has the notion of "_____" (remembering inputs or events that happened in the past)

Combinational vs. Sequential



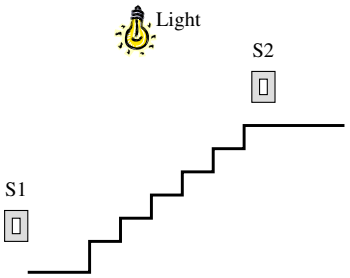


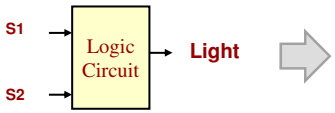
Outputs depend only on current outputs

Outputs depend on current inputs and previous inputs (previous inputs summarized via state)

Combinational Example: Staircase Light Switch

Whether or not the light is on is only dependent on the current position of the switches

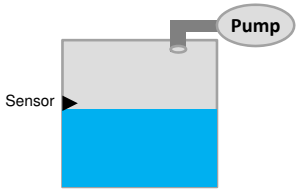


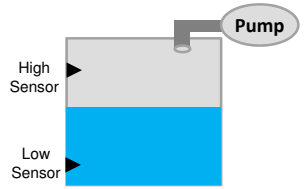


S1	S2	Light
0	0	
0	1	
1	0	
1	1	

Water Tank Problem

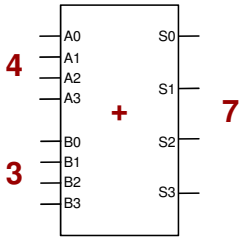
- Build a control system for a pump to keep the tank from going empty





Combinational Logic

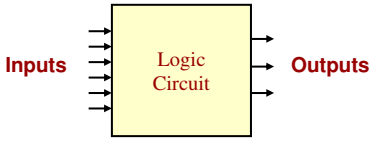
- With *combinational logic* the outputs only depend on what the inputs are _____



It doesn't matter what the inputs were _____

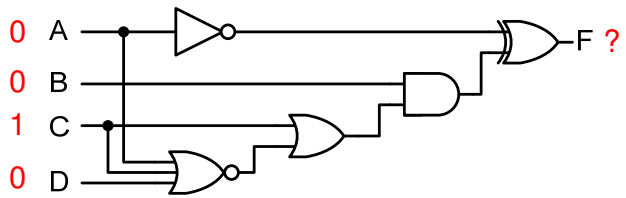
Logic Functions

- Map input combinations of n-bits to desired m-bit output
- Can describe function with a _____ and then find its circuit implementation

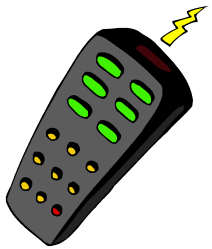


IN0	IN1	IN2	OUT0	OUT1
0	0	0	0	1
0	0	1	1	1
	...			
1	1	1	0	0

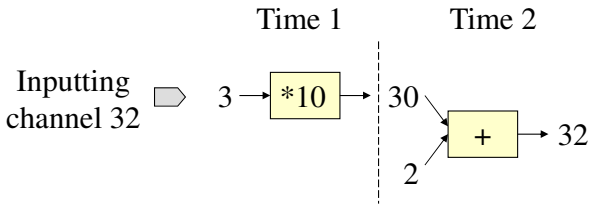
Logic Example



Sequential Example: Remote Control

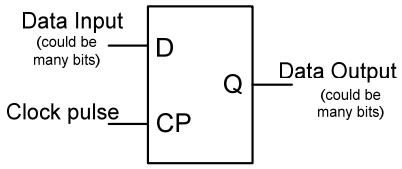


The channel is a _____-dependent function of the first button pressed and the second (we must remember the 3 and then use it with the 2)

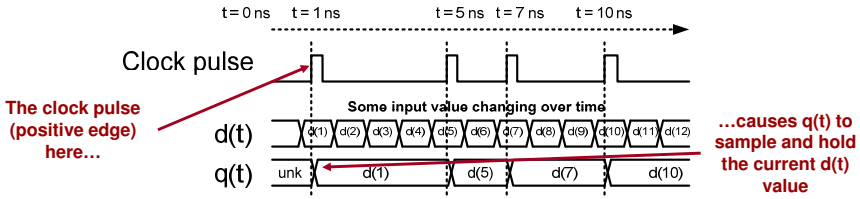


Registers

- Registers are the most common sequential device
- Registers sample the data input (**D**) on the edge of a clock pulse (**CP**) and stores that value at the output (**Q**)
- Analogy: Taking a picture with your digital camera…when you press a button (clock pulse) the camera samples the scene (input) and _____ it as a snapshot (output) until the next trigger

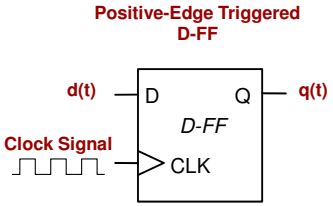


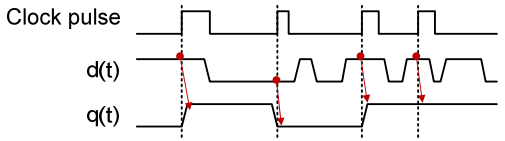
Block Diagram of a Register



Flip-Flops

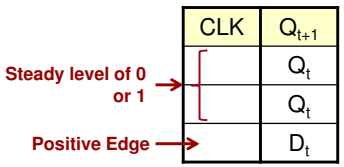
- Flip-flops are the building blocks of registers
  - 1 Flip-flop PER bit of input/output
  - There are many kinds of flip-flops but the most common is the D- (Data) Flip-flop (a.k.a. D-FF)
- D Flip-flop triggers on the clock edge and captures the D-value at that instant and causes Q to remember it until the next edge
  - Positive Edge: instant the clock transition from low to high (0 to 1)

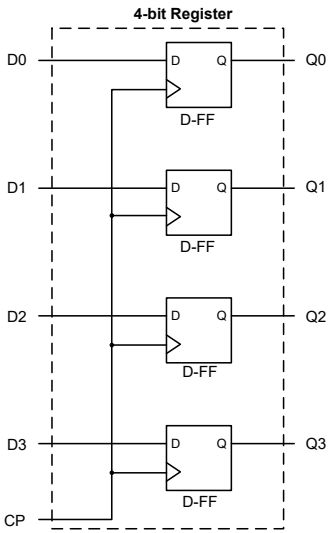




Registers and Flip-flops

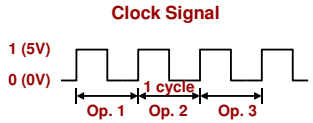
- A register is simply a group of D flip-flops that all trigger on a single clock pulse



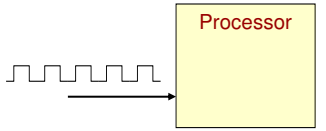


Pulses and Clocks

- Registers need an edge to trigger
- We can generate pulses at specific times (creating an irregular pattern) when we know the data we want has arrived
- Other registers in our hardware should trigger at a regular interval
- For that we use a clock signal…
  - Alternating high/low voltage pulse train
  - Controls the ordering and timing of operations performed in the processor
  - 1 cycle is usually measured from rising/positive edge to rising/positive edge
- Clock frequency (F) = # of cycles per second
- Clock Period (T) = 1 / Freq.



2.8 GHz
= 2.8*10⁹ cycles per second
= 0.357 ns/cycle



Summary

- Combinational logic
  - Perform a specific function (mapping of 2^n input combinations to desired output combinations)
  - No internal state or feedback
    - Given a set of inputs, we will always get the same output after some time (propagation) delay
- Sequential logic ("Storage" devices)
  - Registers made up of flip-flops/latches are the fundamental building blocks
    - Controlled by a "clock" signal
    - Sample data on a "clock" edge and remember that value until the next edge

Combinational vs. Sequential

- Sequential logic (i.e. registers) is used to store values ("storage devices")
  - A register in HW is analogous to a _____ in SW (a variable or register stores a value until needed at a later time)
- Combinational logic is used to process bits (i.e. perform operations on values
  - Combinational logic in HW is analogous to _____ (_____) in SW

THROUGHPUT & LATENCY

Performance Depends on View Point?!

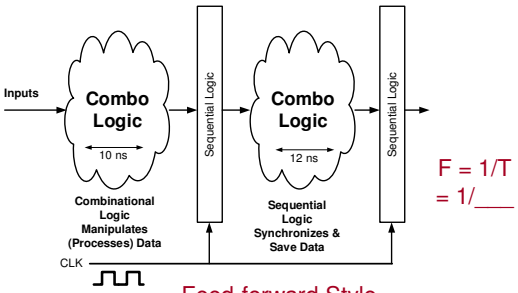
- What's faster:
  - A 747 Jumbo Airliner
  - An F-22 fighter jet
- If you are an individual interested in getting from point A to point B, then the _____
  - This is known as _____ [units of _____]
  - Time from the start of an operation until it completes
- If you are trying to _____ a large number of people, the _____ looks much better
  - This is known as _____ [units of _____]

Throughput vs. Latency

- If Latency is the Time it takes to perform 1 Job to complete and Throughput = Jobs / Time…
- …Is Throughput = 1 / Latency?
- _____!
  - Latency is from the perspective of a _____
  - Throughput is from the perspective of _____
  - Parallelism is the great friend of throughput!
- We will see many times in this course some strategies for improving throughput and sometimes latency

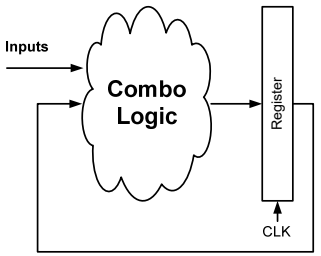
Clocking Methodologies

- Typical designs use both combinational and sequential logic
  - Sequential logic: saves and synchronize data
  - Combinational logic: performs some operation on the data
- Can use feed-forward or feed-back methodology
- Clock cycle must be set for the _____ between registers



F = 1/T
= 1/___

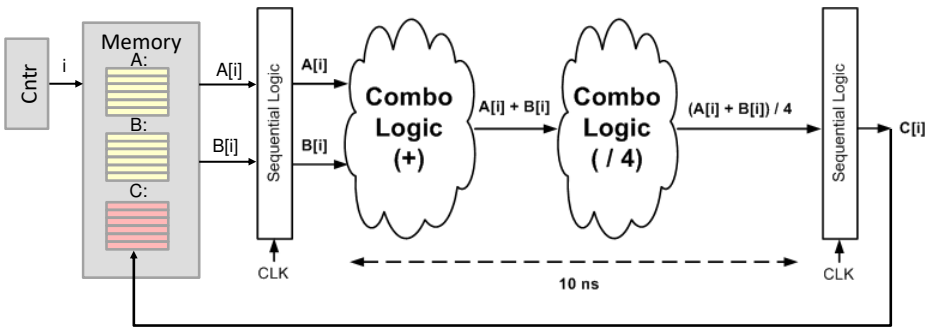
Feed-forward Style



Feed-back Style

Example

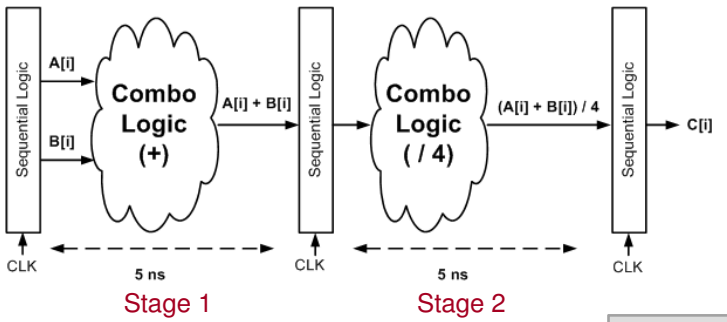
for(i=0; i < 100; i++)
C[i] = (A[i] + B[i]) / 4;



10 ns per input set = _____ ns total

Pipelining Example

for(i=0; i < 100; i++)
C[i] = (A[i] + B[i]) / 4;



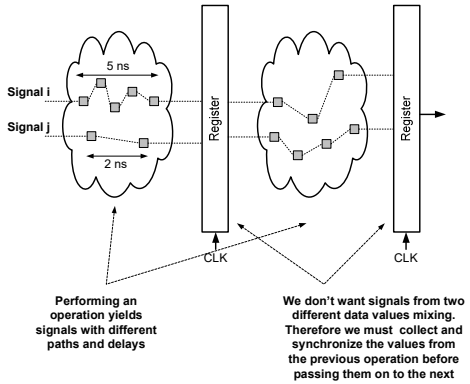
Stage 1 Stage 2

	Stage 1	Stage 2
Clock 0	A[0] + B[0]	
Clock 1		
Clock 2		

Pipelining refers to insertion of registers to ___ combinational logic into smaller stages that can be _____ in time (i.e. create an assembly line)

Need for Registers

- Provides separation between combinational functions
  - Without registers, fast signals could _____ to data values in the next operation stage

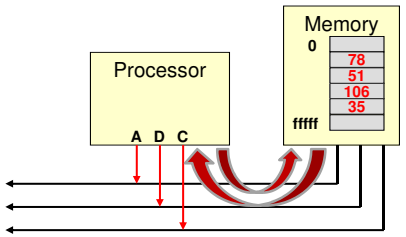


SW vs. HW Sorting (MergeSort)

REAL-WORLD EXAMPLE

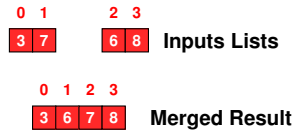
Sorting: Software Implementation

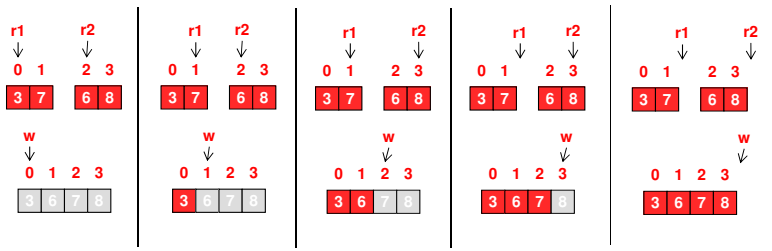
- Let's select a "good" sorting algorithm: mergesort
  - To sort n elements takes time O(_____)
  - Big-O (e.g. O(f(n))) just means exec. time is roughly proportional to f(n)
- Let's then compare the performance of a SW implementation vs. a hardware-accelerated process



Merge Two Sorted Lists

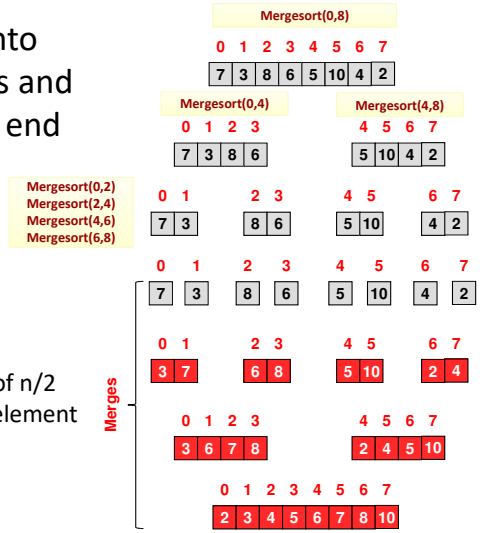
- Consider the problem of merging two sorted lists into a new combined sorted list
- Keep a "read" pointer (r1 and r2) for each sorted array and a "write" (w) pointer to the destination
- Key concept: One comparison yields correct placement of 1 number in the output
  - Implies runtime of merge is O(n)





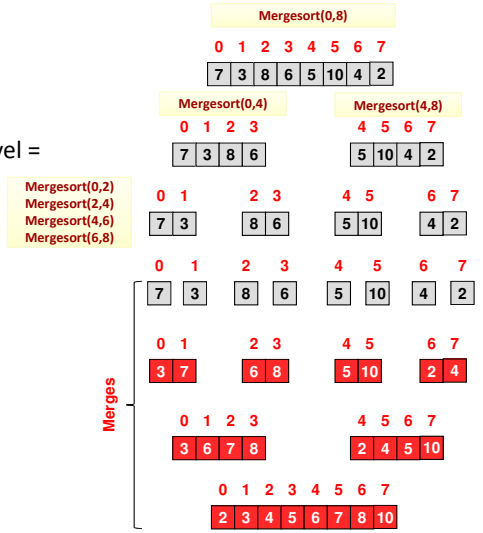
Recursive Sort (MergeSort)

- Break sorting problem into smaller sorting problems and merge the results at the end
- Mergesort(0..n)
  - If list is size 1, return
  - Else
    - Mergesort(0..n/2 - 1)
    - Mergesort(n/2 .. n)
    - Combine each sorted list of n/2 elements into a sorted n-element list



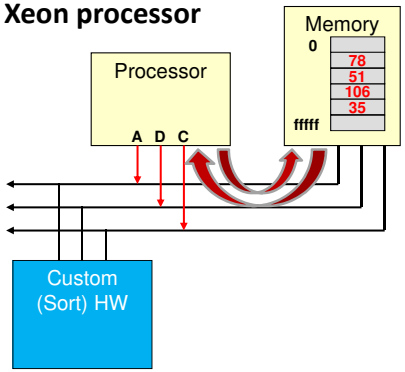
Recursive Sort (MergeSort)

- Run-time analysis
  - # of recursion levels =
    - _____
  - Total operations to merge each level =
    - ___ operations total to merge two lists over all recursive calls at a particular level
- Mergesort = $O(n * \log_2(n))$



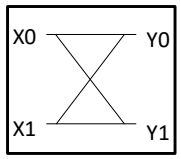
Sorting: Software Implementation

- To perform the algorithm in software means the processor fetches instructions, executes them, which causes the processor to then read and write the data in memory into it's sorted positions
- **Sorting 64 element on a 2.8 GHz Xeon processor**
  - _____ **microseconds**
- **Can we do better w/ more HW?**



HW Sort Network

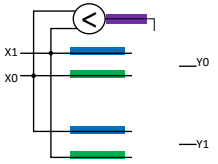
- Start with a small building block in HW: **compare_and_swap (CAS)**
  - _____ input passed to Y0 and _____ to Y1



compare_and_swap HW block diagram

```
if( X0 < X1 ) {
   Y0 = X0;  Y1 = X1;
} else {
   Y0 = X1;  Y1 = X0;
}
```

SW-Equiv. Operation

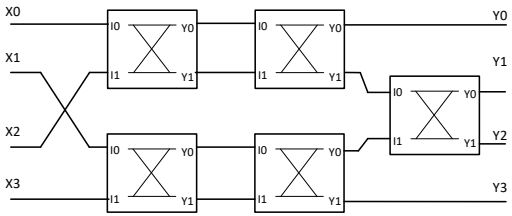


HW Schematic

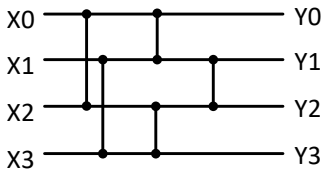
https://www.mn.uio.no/ifi/english/research/projects/cosrecos/publications/paper/fpga11koch.pdf

HW Sort Network

- Now we can use multiple CAS blocks to sort multiple values



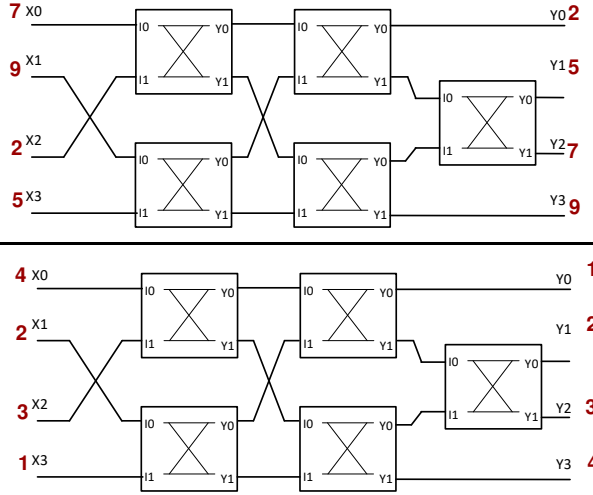
4-Input/Output Sorting Network



Simplified Diagram (Each vertical line is a CAS between the attached elements)

http://dbis.cs.tu-dortmund.de/cms/en/publications/2012/sorting-networks/sorting-networks.pdf

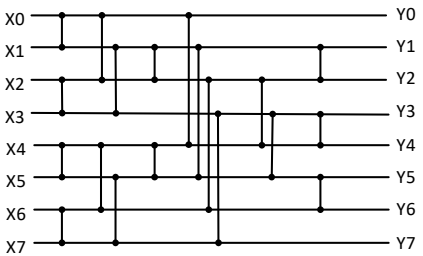
HW Sort Network Example



http://dbis.cs.tu-dortmund.de/cms/en/publications/2012/sorting-networks/sorting-networks.pdf

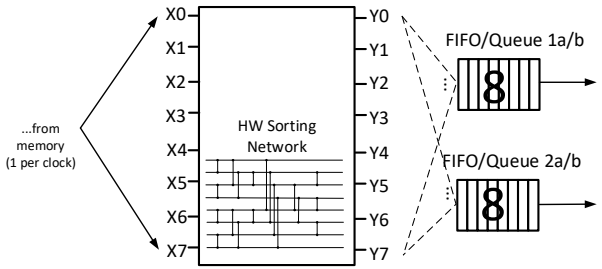
HW Implementation

- A full 64-input/output sorting network in HW may not be feasible due to number of input/output signals
- Let us use an 8-input/output sorting network
  - Use it _____ to produce 8 groups of 8 sorted numbers
  - Then _____ the 8 groups of 8 into a single group of 64



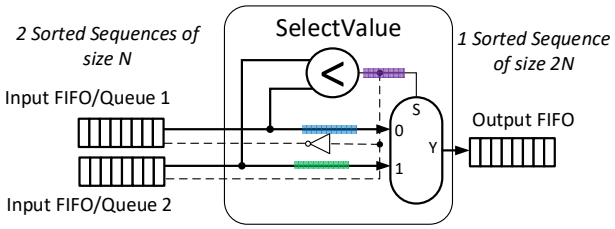
First Stage Sorting

- We will read 8 numbers in 8 clocks from memory
- Sorting can be performed in a _____ and the outputs saved
- We will read in 8 new numbers while we place the previous group of 8 sorted numbers into a Queue/FIFO (_____)
- The next sorted group will go into a 2nd FIFO to be merged with the first



Select-Value Unit

- Now that we have 2 sorted sequences of size N we need to merge them into a single sorted sequence of size _____
- We can design a "Select-Value" unit shown below

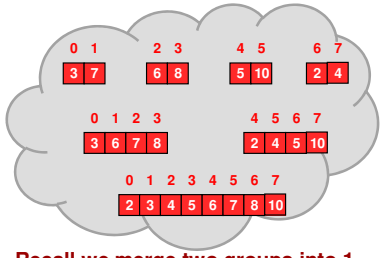



```
if( X0 < X1 ) {
   Y0 = X0;
} else {
   Y0 = X1;
}
```

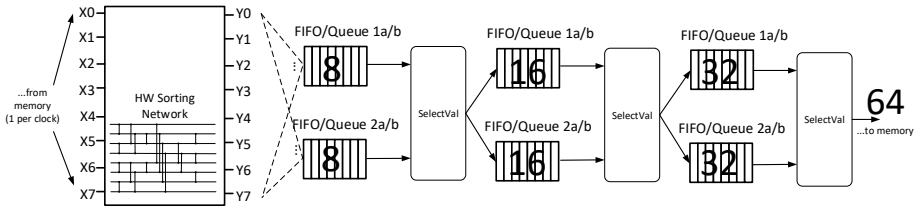
Operation

Merge Stages

- If we have a total of 64 numbers to sort we can arrange our merging in stages
  - We can continue to merge until we get one sequence of 64 (the desired size)

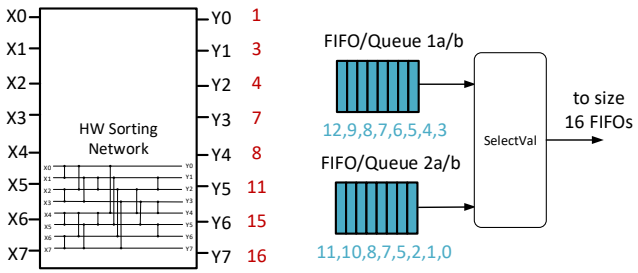


Recall we merge two groups into 1



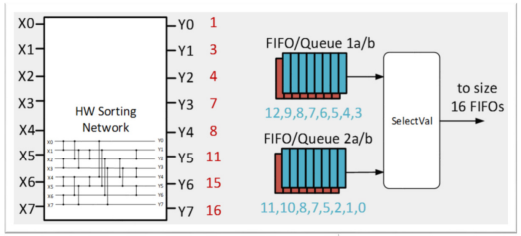
Merge Stages

- We can overlap each stage
  - Merge 2 groups of 8 while we merge 2 groups of 16, etc.
  - Without care, data that is output from one stage may overwrite data in the next stage that has yet to be merged

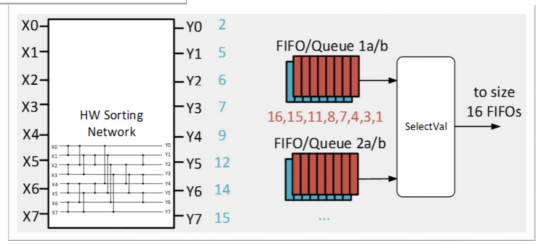


Double (Ping-Pong) Buffers

- Need two sets of FIFOs at each stage (ping-pong buffers) where 1 set is used to fill while we process the other

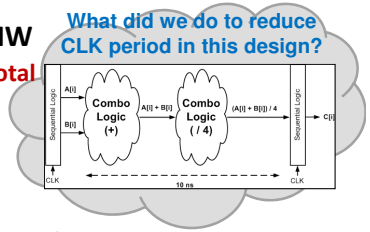


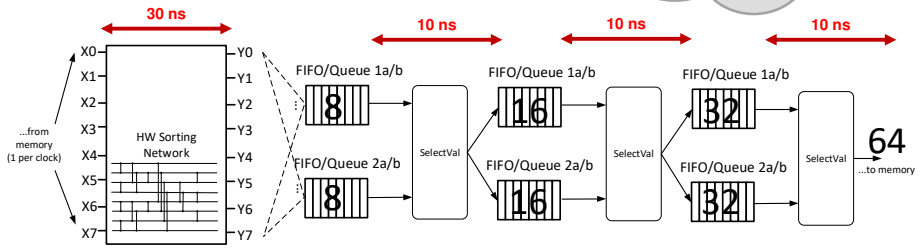
Flip which pair of FIFOs we use for each group of 8. While one group fills with new data we merge the data in the other pair



Sorting: Hardware Implementation

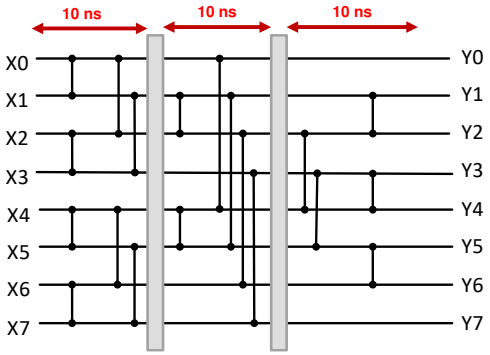
- **Sorting 64 element on a 2.8 GHz Xeon processor [SW only]**
  - **16 microseconds**
- **Sorting 64 numbers in [old] custom HW**
  - **CLK period = 30 ns => __ microseconds total**
  - **30 ns is due to the 8 number HW sorter**
  - **Merging (Select-Val) stages are < 10 ns**
  - **Can we improve?**





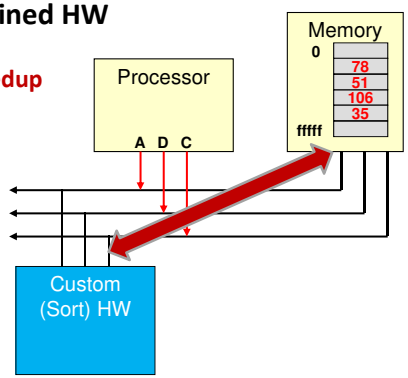
Pipelined Sorter

- Cut sorting network into 3 stages
- In any stage a signal encounters ___ compare-and-swap elements



Sorting: Final Comparison

- **Sorting 64 element on a 2.8 GHz Xeon processor [SW only]**
  - **16 microseconds total time**
- **Sorting 64 numbers in [old] custom HW**
  - **CLK period = 30 ns => 6 microseconds total = ~2.5x speedup**
- **Sorting 64 numbers in [old] pipelined HW**
  - **CLK period = 10 ns =>**
    __ microseconds total = ~__x speedup
  - **Processor is freed to do other work**



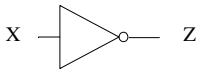
Basic Gates

DIGITAL LOGIC

Digital Logic

- Digital Logic is built on…
  - Binary variables can be only one of two possible values (e.g. 0 or 1)
  - Three operations on binary variables
    - AND (all inputs true => output is true)
    - OR (any inputs true => output is true)
    - NOT (output is opposite of input)

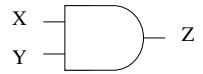
AND, OR, NOT Gates



NOT (Inverter)

$Z = X'$ or $\overline{X}$ or $\sim X$

X	Z
0	1
1	0

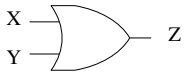


AND

$Z = X \cdot Y$

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

AND = 'ALL'
(true when ALL inputs are true)



OR

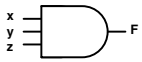
$Z = X + Y$

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

OR = 'ANY'
(true when ANY input is true)

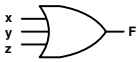
Gates

- Gates can have more than 2 inputs but the functions stay the same
  - AND = output = 1 if _____ inputs are 1
    - Outputs 1 for only 1 input combination
  - OR = output = 1 if _____ input is 1
    - Outputs 0 for only 1 input combination



X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

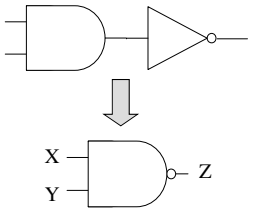
3-input AND



X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

3-input OR

NAND and NOR Gates



NAND

$Z = \overline{X \cdot Y}$

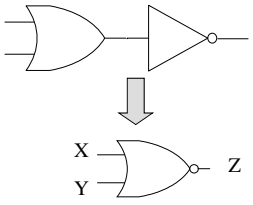
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

AND

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

NAND

True if NOT ALL inputs are true



NOR

$Z = \overline{X + Y}$

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

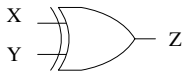
OR

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

NOR

True if NOT ANY input is true

XOR and XNOR Gates

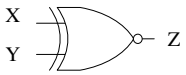


XOR

$Z = X \oplus Y$

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

True if an odd # of inputs are true
2 input case: True if inputs are different



XNOR

$Z = \overline{X \oplus Y}$

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

True if an even # of inputs are true
2 input case: True if inputs are same

Speed, area, and power

DIGITAL DESIGN GOALS

Digital Design Goals

- When designing a circuit, we want to optimize for the following three things:
  - _____ (minimize)
  - Speed (maximize) / Delay (minimize)
  - _____ (minimize)
- Can usually only optimize 2 of the 3
  - There is a huge trade space! This is what engineering is all about!

Minimizing Circuit Area

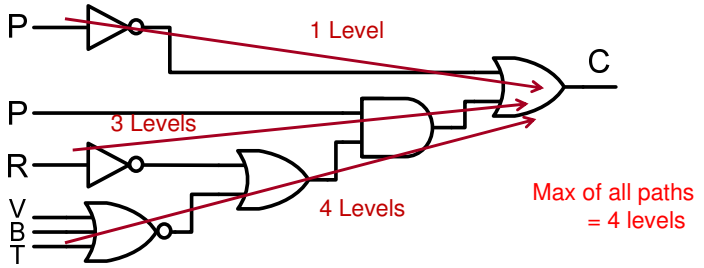
- Approaches:
  - Reduce the _____ used to implement a circuit
  - Reduce the number of _____ to each gate
    - In general a gate with n inputs requires ____ transistors to implement
- Simplify logic expressions (usually by factoring and then canceling terms) to reduce the number of gates

Maximizing Speed

- Speed is affected by:
  - Levels of logic (path length)
  - _____
  - Number of _____to the gate
  - Number of outputs a gate connects to (fan-out)
  - Feature size and implementation technology

Levels of Logic

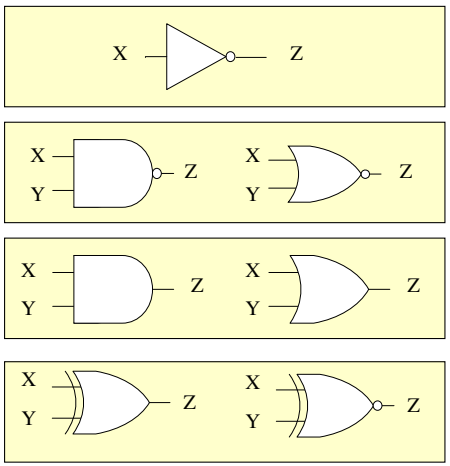
- Definition: _____ number of gates [not including inverters] on _____ path from an input to the output



$$C = \overline{P} + P(\overline{(V+B+T)+R})$$

Gate Delays

- Order the gate types (1,2,3,4) in terms of fastest to slowest?
- Typical gate delay for a 2-input NAND or NOR is under a _____ ps.



Summary

- When designing a circuit, we want to optimize for the following three things:
  - Area (minimize)
    - Use fewer number of gates
    - Use gates w/ fewer inputs
  - Speed (maximize) / Delay (minimize)
    - Fewer levels of logic
      - Levels of logic = max. # of gates on a path from ANY input to output
    - Relative speed of gates: INV, NAND/NOR, AND/OR, XOR/XNOR
  - Power (minimize)
    - How much energy the circuit consumes when switching between 0 and 1
- Can usually only optimize 2 of the 3

LOGIC FUNCTIONS INTRO

Arithmetic vs. Logic Functions

Arithmetic => $f(x_1,x_2,\ldots,x_n)$

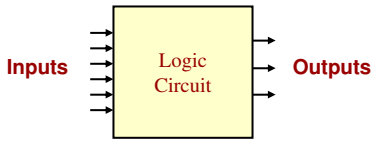
- Domain => $\{Real\}^n$
- Range => Real

Logic => $f(x_1,x_2,\ldots,x_n)$

- Domain => $\{0, 1\}^n$
  - Vector of n zeros or ones
  - 2^n such vectors are possible
- Range => {0, 1}

Logic Functions

- Map input combinations of n-bits to desired m-bit output
  - When we design logic circuits we must describe the output for _____ possible input combination
  - Can describe function with a truth table and then find its circuit implementation



IN0	IN1	IN2	OUT0	OUT1
0	0	0	0	1
0	0	1	1	1
	...			
1	1	1	0	0

Logic Function Domain

- Should specify ALL input combinations
- Most common representation is a truth table
  - For those with SW experience, think of this as a large if..else if or switch structure to categorize the input

X	Y	Z
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Truth Table

```
if(x,y,z == 000) then
  …
else if (x,y,z == 001) then
  …
else if (x,y,z == 010) then
  …
```

If or Case statement

3-bit Prime Number Function

- Should specify ALL input combinations
- Most common representation is a truth table
  - For those with SW experience, think of this as a large if..else if or switch structure to categorize the input

Primes between 0-7

X	Y	Z	P
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Truth Table

X	Y	Z	P
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

OFF-set

ON-set

ON-Set (____terms) : Combinations where output=1
OFF-Set (____terms) : Combinations where output=0

```
if(x,y,z == 000) then
  P = 0
else if (x,y,z == 001) then
  P = 0
else if (x,y,z == 010) then
  P = 1
```

If or Case statement

Multi-output Functions

- N-inputs, m-outputs
  - Rather than simply T/F output, may want to produce a set of signals (i.e. a multi-bit number, etc.)
- Write out all combos, interpret combos, then write in answer

I3	I2	I1	C1	C0
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

1's Count of Inputs

I3	I2	I1	M1	M0
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

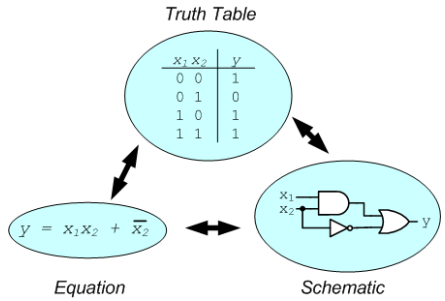
Encode the highest input ID (ie. 3, 2, or 1) that is ON (=1)

Logic Function Examples

- Billy likes pizza but can only afford one-topping: **S**ausage, **P**epperoni, and **M**ushrooms. But today only there is a sale on a mushroom and sausage pizza.
- What pizza's can Billy afford? Describe this function with a truth table.

Logic Functions

- 3 possible representations of a function
  - Equation
  - Schematic
  - Truth Table
- Can convert between representations
- Truth table is only _____ representation*
- We need a way to "_____" (convert from TT to equation/schematic) a function



* Canonical Sums/Products (minterm/maxterm) representation provides a standard equation/schematic form that is unique per function

Example: Automobile Buzzer

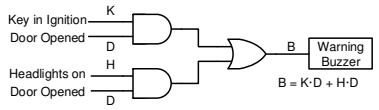
- Consider an automobile warning **B**uzzer that sounds if you leave the **K**ey in the ignition and the **D**oor is open OR the **H**eadlights are on and the **D**oor is open.
- We can easily derive an equation and implementation: B = _____

Example: Automobile Buzzer

- But we see that we can alter this equation…
  - From B = KD + HD
  - To B = _____
    - Buzzer sounds if the Door is open and ***either*** the Key is in the Ignition or the Headlights are on
- Which is better?
- Notice that equations/circuit are not unique
  - The truth table would be the same for both (i.e. unique)

D	K	H	B
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Truth Table is Unique



Non-unique circuit/equation

Non-unique circuit/equation