... everything doesn’t happen at once.

Timing, the 10,000 ft view. Locally synchronous, globally asynchronous.

On the same page. Minimal set of timing concepts you need for project.

Break

RTL Examples. Better timing through micro-architecture.

Electrical details. Just so you know ...
Moore’s Law

2.6 Billion

1 Million

2 Thousand

Synchronous logic on a single clock domain is not practical for a 2.6 billion transistor design.
GALS: Globally Asynchronous, Locally Synchronous

Another approach to interfacing locally synchronous blocks is using specially designed asynchronous FIFO buffers. Such a system can tolerate very large interconnect delays and is also robust with regard to metastability. Designers can use this method to interconnect asynchronous and synchronous systems and also to construct synchronous-asynchronous interfaces. Figure 2 diagrams a typical FIFO interface, which achieves an acceptable data throughput.

In addition to the data cells, the FIFO structure includes an empty/full detector and a special deadlock detector. The advantage of FIFO synchronizers is that they don't affect the locally synchronous module’s operation. However, with very wide interconnect data buses, FIFO structures can be costly in silicon area. Also, they require specialized complex cells to generate the empty/full flags used for flow control. The introduced latency might be significant and unacceptable for high-speed applications.

As an alternative, Beigne and Vivet designed a synchronous-asynchronous FIFO based on the bisynchronous classical FIFO design using gray code, for the specific case of an asynchronous network-on-chip (NoC) interface. Their aim was to maintain compatibility with existing design solutions and to use standard CAD tools. Thus, even with some performance degradation or suboptimal architecture, designers can achieve the main goal of designing GALS systems in the standard design environment.

Boundary synchronization

A third solution is to perform data synchronization at the borders of the locally synchronous island, without affecting the inner operation of locally synchronous blocks and without relying on FIFO buffers. For this purpose, designers can use standard two-flop, one-flop, predictive, or adaptive synchronizers for mesochronous systems, or locally delayed latching. This method can achieve very reliable data transfer between locally synchronous blocks. On the other hand, such solutions generally increase latency and reduce data throughput, resulting in limited applicability for high-speed systems. Table 1 summarizes the properties of GALS systems’ synchronization methods.

Advantages and limitations of GALS solutions

The scientific community has shown great interest in GALS solutions and architectures in the past two decades. However, this interest hasn't culminated in many commercial applications, despite all reported advantages. There are several reasons why standard design practice has not adopted GALS techniques.

1. Many proposed solutions require programmable ring oscillators. This is an inexpensive solution that allows full control of the local clock. However, it has significant drawbacks. Ring oscillators are impractical for industrial use. They need careful calibration because they are very sensitive to process, voltage, and temperature variations. Moreover, embedded ring oscillators consume additional power through continuous switching of the chained inverters.

On the other hand, careful design of the delay line can reduce its power consumption to a level below that of a corresponding clock tree. In addition, 432

Synchronous modules typically 50K-1M gates, so that the synchronous logic approach works well without requiring heroics. Examples ...

Thursday, September 5, 13
Stars denote FIFOs that create separate synchronous domains. An example of how architecture and circuits work together.
Rocket uses GALS for accelerator interface

Your project interfaces with the RISC-V pipeline and the memory system using FIFOs.

Your timing closure is independent of the CPU logic domain.
Today: Timing insights for your project

What we’re not doing. If this class was EE 241 and your project was an SRAM:

You could see through down to the layout.

Timing? Use SPICE on this hand-drawn schematic.
Technology X: The CS 250 timing challenge.

What we are doing --->

If your accelerator is too slow ... two options:


Bottom-up: Take control away from logic synthesis. Use HDL as textual schematic. Also, use command-line tool flags. Sometimes necessary. Ben is the expert, ask in discussion section.
A Logic Circuit Primer

“Models should be as simple as possible, but no simpler ...”  Albert Einstein.
Inverters: A simple transistor model

Correctly predicts logic output for simple static CMOS circuits.

Extensions to model subtler circuit families, or to predict timing, have not worked well ...

pFET. A switch. "On" if gate is grounded.

nFET. A switch. "On" if gate is at Vdd.

Combinational Logic: Like function evaluation
Wires: Carry signals from one point to another

PMOS Transistor
NMOS Transistor

CMOS:
Elements of the design zoo

Basic Technology: CMOS

Apply a LOW (GND) to its gate
Apply a HIGH (Vdd) to its gate

PMOS (P-Type Metal Oxide Semiconductor) transistors
NMOS (N-Type Metal Oxide Semiconductor) transistors

Circuit

"1"
"0"
"1"
"0"

Vdd
PMOS
NMOS

"1"
"0"

In
Out

In
Out

0 1
1 0
Transistors as water valves. *(Cartoon physics)*

If electrons are water molecules, transistor strengths *(W/L)* are pipe diameters, and capacitors are buckets ...

A “on” p-FET fills up the capacitor with charge.

A “on” n-FET empties the bucket.

This model is often good enough ...
What is the bucket? A gate’s “fan-out”.

“Inverter:

- Driving wires slows a gate down.
- Driving it’s own parasitics slows a gate down.

“Fan-out”: The number of gate inputs driven by a gate’s output.
A closer look at fan-out ...

Driving more gates adds delay.

Linear model works for reasonable fan-out

Delay time of an inverter driving 4 inverters.

F04: Fanout of four delay.
Propagation delay graphs ...

- Cascaded gates:

\[ V_{in} \rightarrow V_{out} \]

\[ 1 \rightarrow 0 \]
\[ 0 \rightarrow 1 \]

INV1.

\[ V_{dd}/2 \]

\[ t \]

INV2.

\[ V_{dd}/2 \]

propagation delay for INV2

INV3.

\[ V_{dd}/2 \]

propagation delay for INV2 & INV3 in series

1 - \( V_{dd}/2 \)

0 - \( V_{dd}/2 \)

INVERTER
TRANSFER FUNCTION

Vout:

Vin:

Thursday, September 5, 13
Worst-case delay through combinational logic

T2 might be the worst-case delay path (critical path)

\[ x = g(a, b, c, d, e, f) \]

If \( d \) going 0-to-1 switches \( x \) 0-to-1, delay is \( T_1 \).
If \( a \) going 0-to-1 switches \( x \) 0-to-1, delay is \( T_2 \).
It would be surprising if \( T_1 > T_2 \).
Why “might”? Wires have delay too ...

- Wires possess distributed resistance and capacitance

- Time constant associated with distributed RC is proportional to the square of the length

- Signals are typically “rebuffered” to reduce delay:
Clocked Logic Circuits
Timing Analysis

What is the smallest $T$ that produces correct operation?

<table>
<thead>
<tr>
<th>$f$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 MHz</td>
<td>1 μs</td>
</tr>
<tr>
<td>10 MHz</td>
<td>100 ns</td>
</tr>
<tr>
<td>100 MHz</td>
<td>10 ns</td>
</tr>
<tr>
<td>1 GHz</td>
<td>1 ns</td>
</tr>
</tbody>
</table>
Timing Analysis and Logic Delay

If our clock period $T >$ worst-case delay through CL, does this ensure correct operation?
Flip Flops have internal delays ...

Value of $D$ is sampled on positive clock edge.

$Q$ outputs sampled value for rest of cycle.
Flip-Flop delays eat into “time budget”

\[ T \geq \tau_{clk\rightarrow Q} + \tau_{CL} + \tau_{setup} \]
Clock skew also eats into “time budget”

\[ T \geq T_{CL} + T_{setup} + T_{clk\rightarrow Q} + \text{worst case skew}. \]

As \( T \rightarrow 0 \), which circuit fails first?

- Control clock skew
  - Careful clock distribution. Equalize path delay from clock source to all clock loads by controlling wires delay and buffer delay.
  - Don’t “gate” clocks.

Most modern large high-performance chips (microprocessors) control end to end clock skew to a few tenths of a nanosecond.

Clock skew, delay in distribution
Schematic diagram of global clock generation and distribution. Figure 6 shows the relationship between delay and tuned sector trees. The grid and sector buffers are visible at this scale. The clock tree delays are tuned to ensure optimal performance of the IBM "Power" CPU.
Clock Tree Delays, IBM Power
SomeFlipFlopshave "hold" time ...

What is the intended function of this circuit?

Does flip-flop hold time affect operation of this circuit? Under what conditions?

\[ t_{\text{clk-to-Q}} + t_{\text{inv}} > t_{\text{hold}} \]

For correct operation.
Searching for processor critical path

Timing Analysis

What is the smallest T that produces correct operation?

Must consider all connected register pairs.

Why might I suspect this one?
Combinational paths for IBM Power 4 CPU

The critical path

Most wires have hundreds of picoseconds to spare.

How to retime logic

Critical path is 5. We want to improve it without changing circuit semantics.

Add a register, move one circle. Performance improves by 20%.

Circles are combinational logic, labelled with delays.

Figure 1: A small graph before retiming. The nodes represent logic delays, with the inputs and outputs passing through mandatory, fixed registers. The critical path is 5.

Figure 2: The example in Figure 2 after retiming. The critical path is reduced from 5 to 4.

“Technology X” can often do this.
Power 4: Timing Estimation, Closure

Timing Estimation

Predicting a processor’s clock rate early in the project

Power 4: Timing Estimation, Closure

Timing Closure

Meeting (or exceeding!) the timing estimate

Floorplaning: essential to meet timing.

(Intel XScale 80200)
Break
Simple exercises for gaining intuition about timing for your process + EDA tools.

Know Your Numbers

Thanks to Bhupesh Dasila, Open-Silicon Bangalore
Synthesize gate chains using hand-specified library cells

Exercises

Cell library and place and route tools.

Let's you know how many levels of logic you can use in the best case.

Delay of a chain of 3 inverters with strongest strength. “Guaranteed not to exceed” speed.

40 nm process

29 ps/gate av.

Synthesis constrained to 2ns clock.

Helps you “see through” … “Technology X”.

Bhupesh Dasila
Force P&L to drive a long wire with a known buffer cell.

Vary driver strength, wire length, metal layer.

Shows the maximum distance two gates can be placed and still meet your clock period.

Distributed RC is the square of the length is clearly seen!

<table>
<thead>
<tr>
<th>buffer type</th>
<th>delay/length (ps/mm)</th>
<th>area (um^2)</th>
<th>leakagePower (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer8X</td>
<td>195.8</td>
<td>2.8</td>
<td>1565.6</td>
</tr>
<tr>
<td>Buffer12X</td>
<td>175.7</td>
<td>3.9</td>
<td>2199.0</td>
</tr>
<tr>
<td>Buffer16X</td>
<td>165.2</td>
<td>5.1</td>
<td>3037.7</td>
</tr>
<tr>
<td>Buffer20X</td>
<td>162.0</td>
<td>6.2</td>
<td>3693.8</td>
</tr>
</tbody>
</table>
Driving Large Loads

- Large fanout nets: clocks, resets, memory bit lines, off-chip
- Relatively small driver results in long rise time (and thus large gate delay)

Strategy:

- Optimal trade-off between delay per stage and total number of stages $\Rightarrow$ fanout of $\sim 4-6$ per stage
Register file: Synthesize, or use SRAM?

R0 - The constant 0

D En R1

D En R2

D En R31

“two read ports”

Speed will depend on how large it lays out ...
Synthesized, custom, and SRAM-based register files, 40nm

For small register files, logic synthesis is competitive.

Not clear if the SRAM data points include area for register control, etc.

Figure 3: Using the raw area data, the physical implementation team can get a more accurate area estimation early in the RTL development stage for floorplanning purposes. This shows an example of this graph for a 1-port, 32-bit-wide SRAM.
Techniques
Pipelining
Starting point: A single-cycle processor

Challenge: Speed up clock while keeping CPI == 1

Seconds
Program
\[ \frac{\text{Instructions}}{\text{Program}} = \frac{\text{Cycles}}{\text{Instruction}} \]  \frac{\text{Seconds}}{\text{Cycle}}

CPI == 1
This is good.

Slow.
This is bad.

PC
\[ D \quad Q \]

Instr Mem
\[ \text{Addr} \quad \text{Data} \]

RegFile
\[ rs1 \quad rs2 \quad \text{rd1} \quad \text{rd2} \quad \text{ws} \quad \text{wd} \quad \text{WE} \]

Instr
\[ \text{Mem} \]

Data Memory
\[ \text{Addr} \quad \text{Din} \quad \text{Dout} \quad \text{WE} \]

ALU
\[ \text{Op} \quad 32 \quad 32 \]

Ext

MemToReg

0x4
Reminder: How data flows after posedge

0x4

RegFile
rs1  rs2  rd1  rd2
ws    wd  WE

Logic

Instr
Mem
Addr  Data

PC
D  Q

ALU

op

Addr

Data

Instr

Mem

Logic

RegFile
rs1  rs2  rd1  rd2
ws    wd  WE

0x4

ALU
Next posedge: Update state and repeat
Observation: Logic idle most of cycle

For most of cycle, ALU is either “waiting” for its inputs, or “holding” its output

Ideal: a CPU architecture where each part is always “working”.

For most of cycle, ALU is either “waiting” for its inputs, or “holding” its output
Inspiration: Automobile assembly line

Assembly line moves on a steady clock. Each station does the same task on each car.
Inspiration: Automobile assembly line

Simpler station tasks $\rightarrow$ more cars per hour. Simple tasks take less time, clock is faster.
Inspiration: Automobile assembly line

Line speed limited by slowest task. Most efficient if all tasks take same time to do.
Inspiration: Automobile assembly line

Simpler tasks, complex car → long line!

These lines go 24 x 7, and rarely shut down.
Lessons from car assembly lines

- Faster line movement yields more cars per hour off the line.

- Faster line movement requires more stages, each doing simpler tasks.

- To maximize efficiency, all stages should take same amount of time (if not, workers in fast stages are idle)

- “Filling”, “flushing”, and “stalling” assembly line are all bad news.
Key Analogy: The instruction is the car

Pipeline Stage #1: Instruction Fetch

Stage #2: Controls hardware in stage 2

Stage #3: Controls hardware in stage 3

Stage #4: Controls hardware in stage 4

Stage #5: Controls hardware in stage 5

"Data-stationary control"
Example: Decode & Register Fetch Stage

Pipeline Stage #1
Instr Fetch

Stage #2
Decode & Reg Fetch

Stage #3

**A sample program**

ADD R4, R3, R2
OR R7, R6, R5
SUB R10, R9, R8

*R’s chosen so that instructions are independent - like cars on the line.*
Hazards: An instruction is not a car ...

---

Stage #1  
Instr Fetch

Stage #2  
Decode & Reg Fetch

Stage #3

---

OR R5, R4, R2  
... wrong value of R4 fetched from RegFile, contract with programmer broken! Oops!

ADD R4, R3, R2  
R4 not written yet ...

New sample program

ADD R4, R3, R2  
OR R5, R4, R2

An example of a “hazard” -- we must (1) detect and (2) resolve all hazards to make a CPU that matches ISA
Some ways to cope with hazards makes CPI > 1 “stalling pipeline”

Added logic to detect and resolve hazards increases clock period

“Software slows the machine down”
Seymour Cray
Superpipelining
Superpipelining: Add more stages

Goal: Reduce critical path by adding more pipeline stages.

Example: 8-stage ARM XScale: extra IF, ID, data cache stages.

Difficulties: Added penalties for load delays and branch misses.

Ultimate Limiter: As logic delay goes to 0, FF clk-to-Q and setup.

Also, power!
Note: Some stages now overlap, some instructions take extra stages.

5 Stage

- IF
- ID+RF
- EX
- MEM
- WB

8 Stage

- F1
- F2
- ID
- TH/ID
- RF
- RF
- X1
- X2
- WB

IF now takes 2 stages (pipelined I-cache)
ID and RF each get a stage.
ALU split over 3 stages
MEM takes 2 stages (pipelined D-cache)
Superpipelining techniques ...

- Split **ALU** and **decode** logic over several pipeline stages.

- **Pipeline memory**: Use more banks of smaller arrays, add pipeline stages between decoders, muxes.

- **Remove** “rarely-used” **forwarding networks** that are on critical path.
  - Creates stalls, affects CPI.

- **Pipeine the wires of frequently used forwarding networks**.

  *Also: Clocking tricks (example: use posedge and negedge registers)*
Hardware limits to superpipelining?

FO4: How many fanout-of-4 inverter delays in the clock period.

Historical limit: about 12 FO4s

CPU Clock Periods 1985-2005

MIPS 2000
5 stages

Pentium Pro
10 stages

Pentium 4
20 stages

Power wall: Intel Core Duo has 14 stages

Thanks to Francois Labonte, Stanford
F04 Delays Per Cycle for Processor Designs

F04 delay per cycle is roughly proportional to the amount of computation completed per cycle.
Multithreading
Multithreading of Static Pipelines

Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe

T1: LW r1, 0(r2)
T2: ADD r7, r1, r4
T3: XORI r5, r4, #12
T4: SW 0(r7), r5
T1: LW r5, 12(r1)

Last instruction in a thread always completes writeback before next instruction in same thread reads regfile

4 CPUs, each run at 1/4 clock

Many variants...
At the logic level ...

Synchronous logic we want to “multithread”. Critical path is 5.

2X multi-threading: double each register.

Modern synthesis will retime this as shown: critical path is now 2.

Post-Placement C-slow Retiming for the Xilinx Virtex FPGA

Figure 1: A small graph before retiming. The nodes represent logic delays, with the inputs and outputs passing through mandatory, fixed registers. The critical path is 5.

Figure 3: The example in Figure 2 2-slowed. This design now operates on 2 independent data streams.

Figure 4: The example in Figure 3 after retiming. The combination of C-slowing and retiming reduced the critical path from 5 to 2.
Good fit for GALS

Two input queues (red and green). The mux control logic implements turn-taking.

Outputs placed into two output queues.
Crossbar Networks
When register files get big, they get slow.

Even worse: adding ports slows down as $O(N^2)$ ...

Why? Number of loads on each $Q$ goes as $O(N)$, and the wire length to port mux goes as $O(N)$.
Crossbar networks: general case of this problem

Sun Niagara II: 8 cores, 4MB L2, 4 DRAM channels

Each DRAM channel: 50 GB/s Read, 25 GB/s Write BW.

Crossbar BW: 270 GB/s total (Read + Write).

(Also shared by an I/O port, not shown)
Sun Niagara II
8 x 9 Crossbar

100-200 wires/port (each way).

4 cycle latency (715ps/cycle).

Cycles 1-3 are for arbitration.

Transmit data on cycle 4.

Pipelined.

8 ports on CPU side (one per core)

8 ports for L2 banks, plus one for I/O
A complete switch transfer (4 epochs)

Epoch 1: All input ports (that are ready to send data) request an output port.

Epoch 2: Allocation algorithm decides which inputs get to write.

Epoch 3: Allocation system informs the winning inputs and outputs.

Epoch 4: Actual data transfer takes place.

Allocation is pipelined: a data transfer happens on every cycle, as does the three allocation stages, for different sets of requests.
Every cross of blue and purple is a pass gate with a unique control signal.

72 control signals (if distributed unencoded).
Sun Niagara II
Crossbar Notes

- Low latency: 4 cycles (less than 3 ns).
- Uniform latency between all port pairs.
- Crossbar defines floorplan: all port devices should be equidistant to the crossbar.

Did not scale up for 16-core Rainbow Falls. Rainbow Falls keeps the 8 x 9 crossbar, and shares each CPU-side port with two cores.

Design alternatives to crossbar?
CLOS networks: from telecom world ...

Build a high-port switch by tiling fixed-sized shuffle units. Pipeline registers naturally fit between tiles. Trades scalability for latency.
CLOS networks: an example route

Numbers on left and right are port numbers. Colors show routing paths for an exchange. Arbitration still needed to prevent blocking.
Electrical Details
Flip Flops Revisited
Recall: Static RAM cell (6 Transistors)

Cross-coupled inverters
Recall: Positive edge-triggered flip-flop

A flip-flop “samples” right before the edge, and then “holds” value.

Sampling circuit

Holds value

16 Transistors: Makes an SRAM look compact!

What do we get for the 10 extra transistors?

Clocked logic semantics.
Delay in Flip-flops

- **Setup time**: results from delay through the first latch.
- **Clock to Q delay**: results from delay through the second latch.

A flip-flop “samples” right before the edge, and then “holds” value.

---

**Sampling circuit**

- When **clk = 0** and **clk’ = 1**
  - **D** is sampled on the posedge.
  - **Q** outputs the last value captured.

- When **clk = 1** and **clk’ = 0**
  - **D** is sampled on the negedge.
  - **Q** holds the value captured on the previous edge.

---

**Holds value**
Capture: When clock goes high

A flip-flop “samples” right before the edge, and then “holds” value.

Sampling circuit

Holds value

\[ \text{clk} = 1 \]
\[ \text{clk'} = 0 \]

Remembers value just captured.

Outputs value just captured.
Flip Flop delays:  \( \text{clk-to-Q?} \quad \text{setup?} \quad \text{hold?} \)

**CLK == 0**

Sense D, but Q outputs old value.

**CLK 0->1**

Capture D, pass value to Q
More Detailed Gate Models
Inverters: Circuits and Layout
Inverter: Die Cross Section

V_{in} \rightarrow V_{out}

V_{out} \rightarrow \text{Inverter}

n^+ \quad p^- \quad n^+ \quad p^+ \quad n^+

n-well

oxide
Inverters with $Vin = Gnd$, $Vout = Vdd$

$$Isd = k \frac{W}{L} [Vsg - Vt] [Vsd]$$

Is $Vsd > Vsg - Vt$ once $Vout$ is $Vdd$?

Is $Vsg > Vt$?

This goes as close to 0 as it can while still supplying the leakage current.

$Ids \approx 0$, but really a small leakage current
Inverters with \( V_{in} = V_{dd}, V_{out} = Gnd \)

\[ I_{sd} \approx 0, \text{ but really a small leakage current} \]

\[ I_{ds} = k(W/L)[V_{gs} - V_{t}] [V_{ds}] \]

This goes as close to 0 as it can while still supplying the leakage current.

\[ \text{Is } V_{ds} > V_{gs} - V_{t} \text{ once } V_{out} \text{ is Gnd?} \]

\[ \text{Is } V_{gs} > V_{t} ? \]
On Tuesday ... Power and Energy

Heat Source

Heat Sink

Lecture 4: Physical Realities, Part II: Energy and power.