Introduction to FPGA

In 653 easy steps,

Spending less than a million bucks,

and from a guy with just a few months experience!

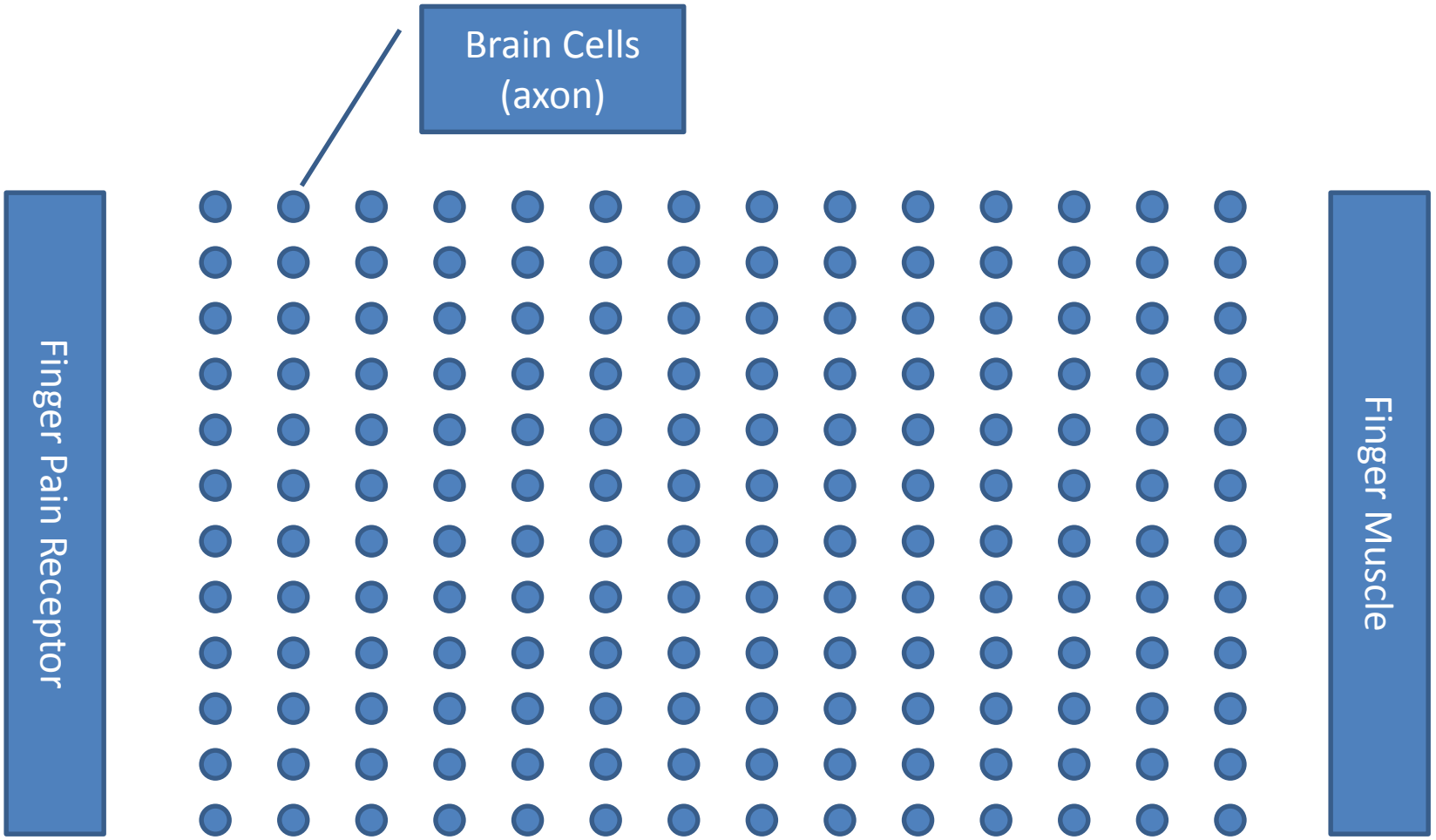
What is an FPGA?

- It is an integrated circuit that can be programmed to implement complex logic.
- Only Boolean logic, nothing analog!
  - Except on special chips

A quick analogy from biology

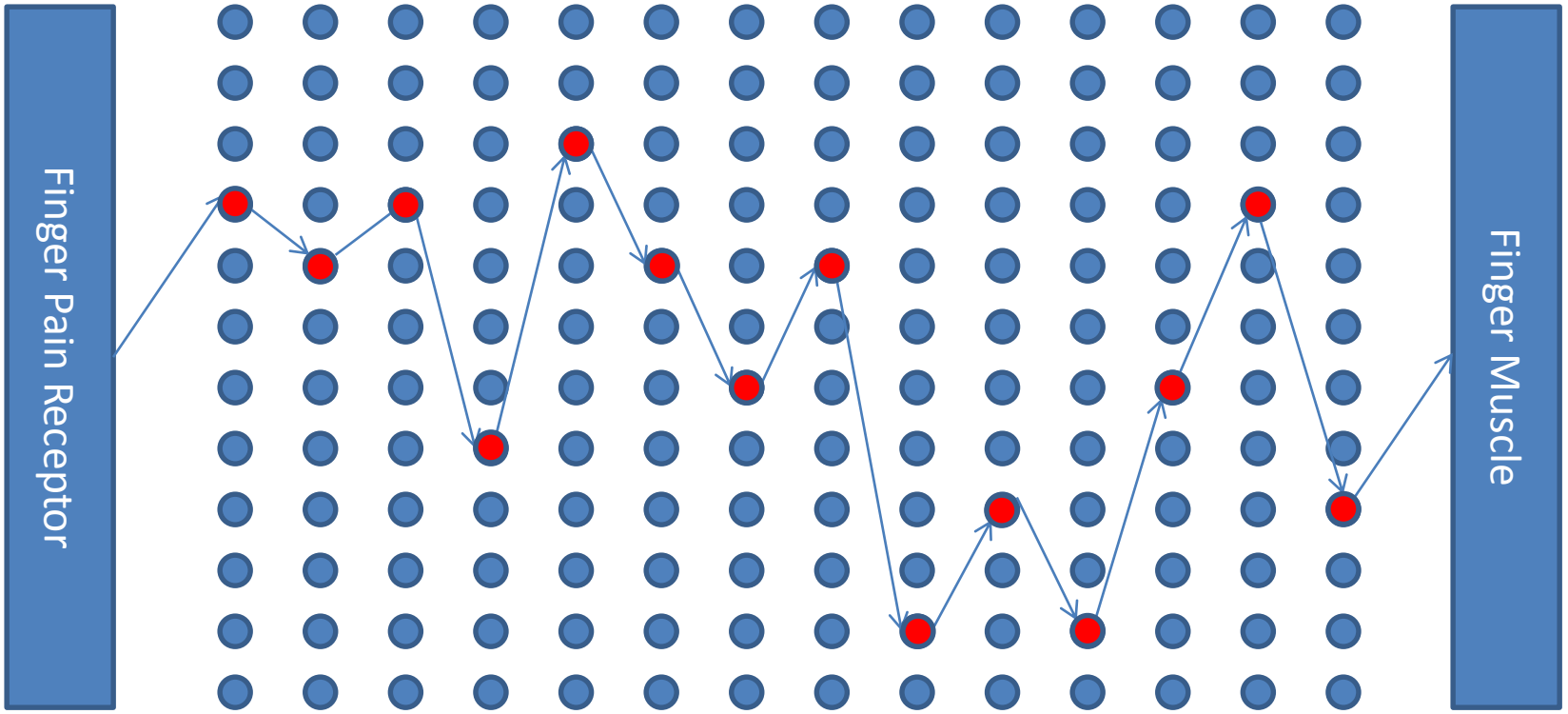
- Supposedly, all humans are born with all the brain cells they will ever have.
- But these cells are not connected up properly, or at all, at birth.
- As you learn (get programmed) connections are made allowing you to do things.

Baby

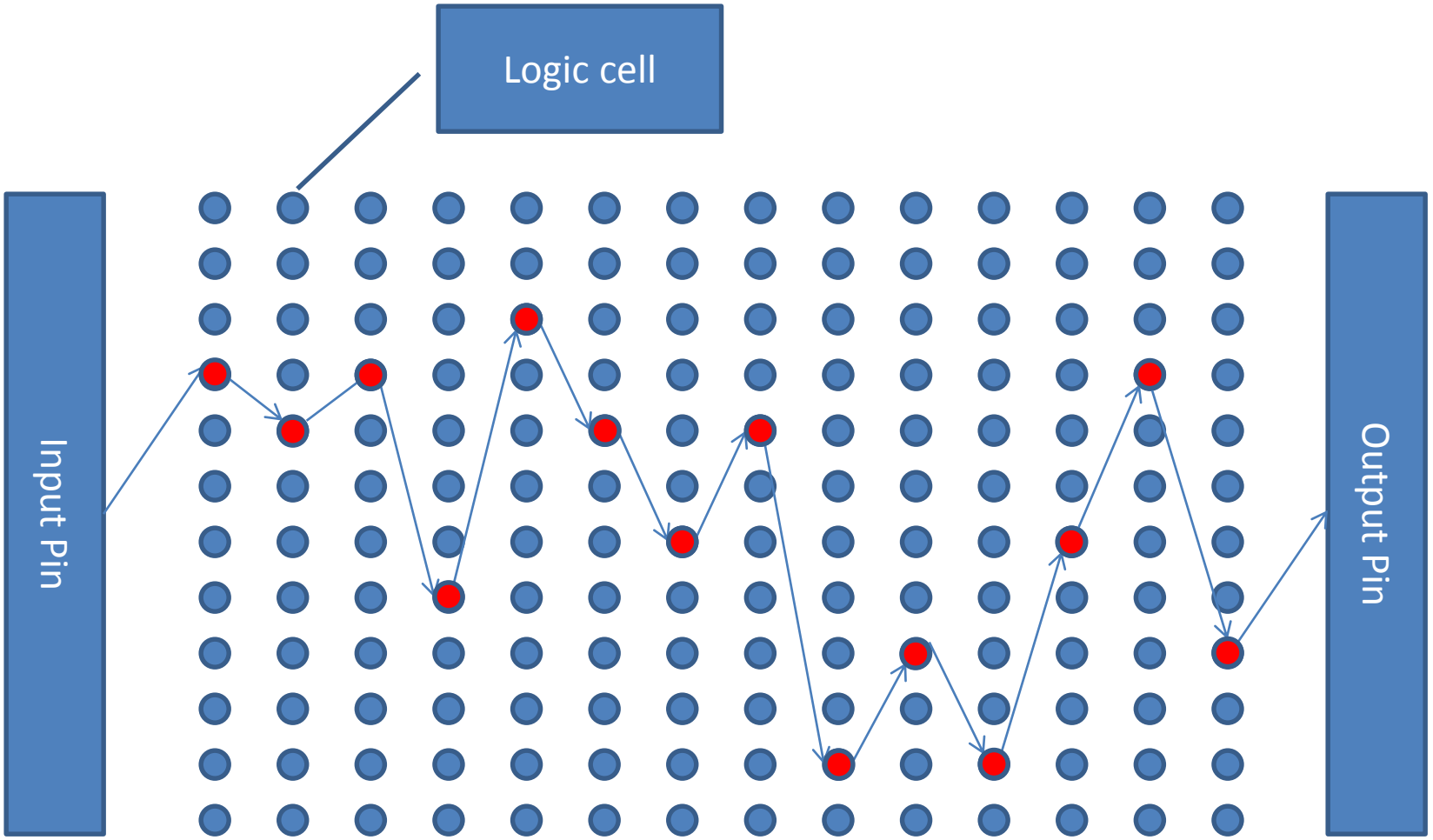


Learning Event(s)

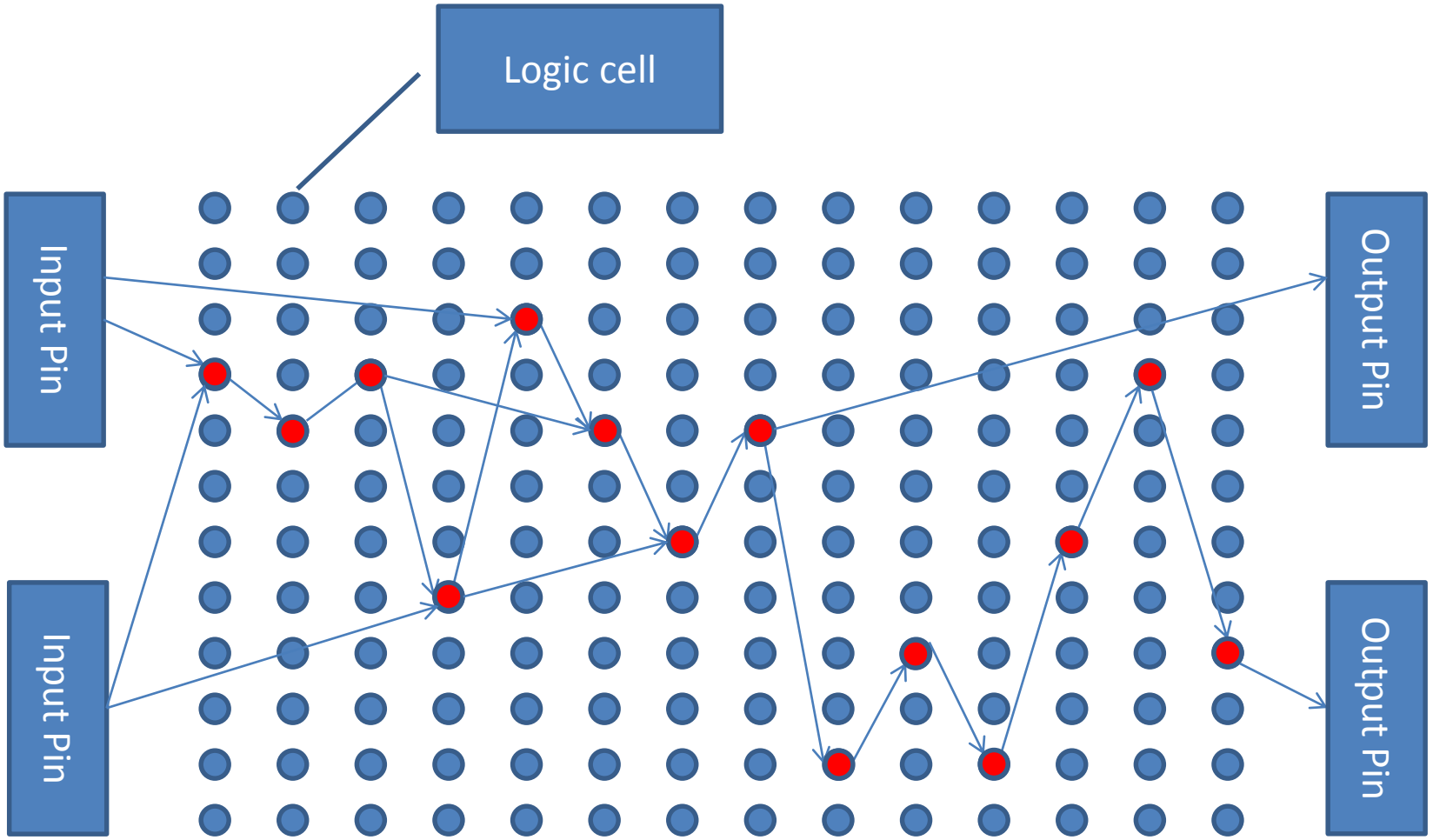
Toddler



FPGA



FPGA



A Brief History

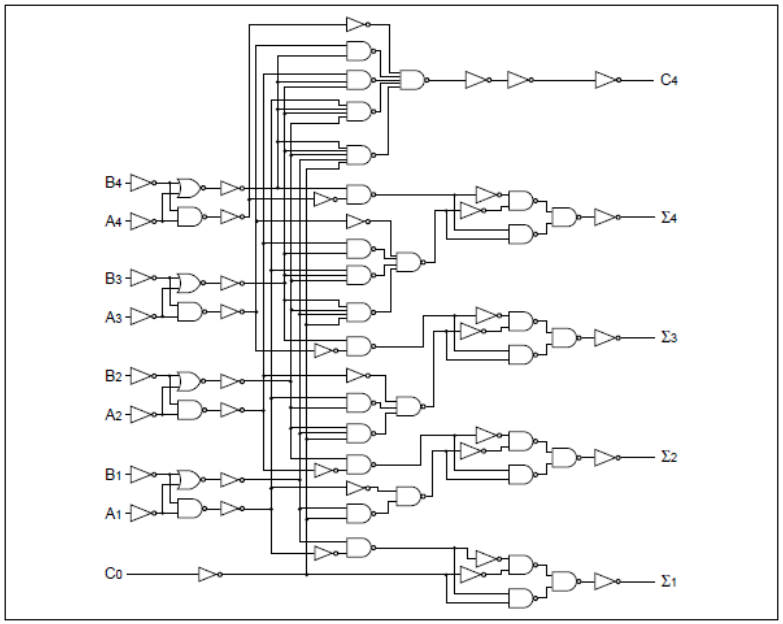
- To appreciate the FPGA, we need to go back in time…
- So close your eyes and go back to the 1960s.
  - (Sorry Noah)

Discreet Logic

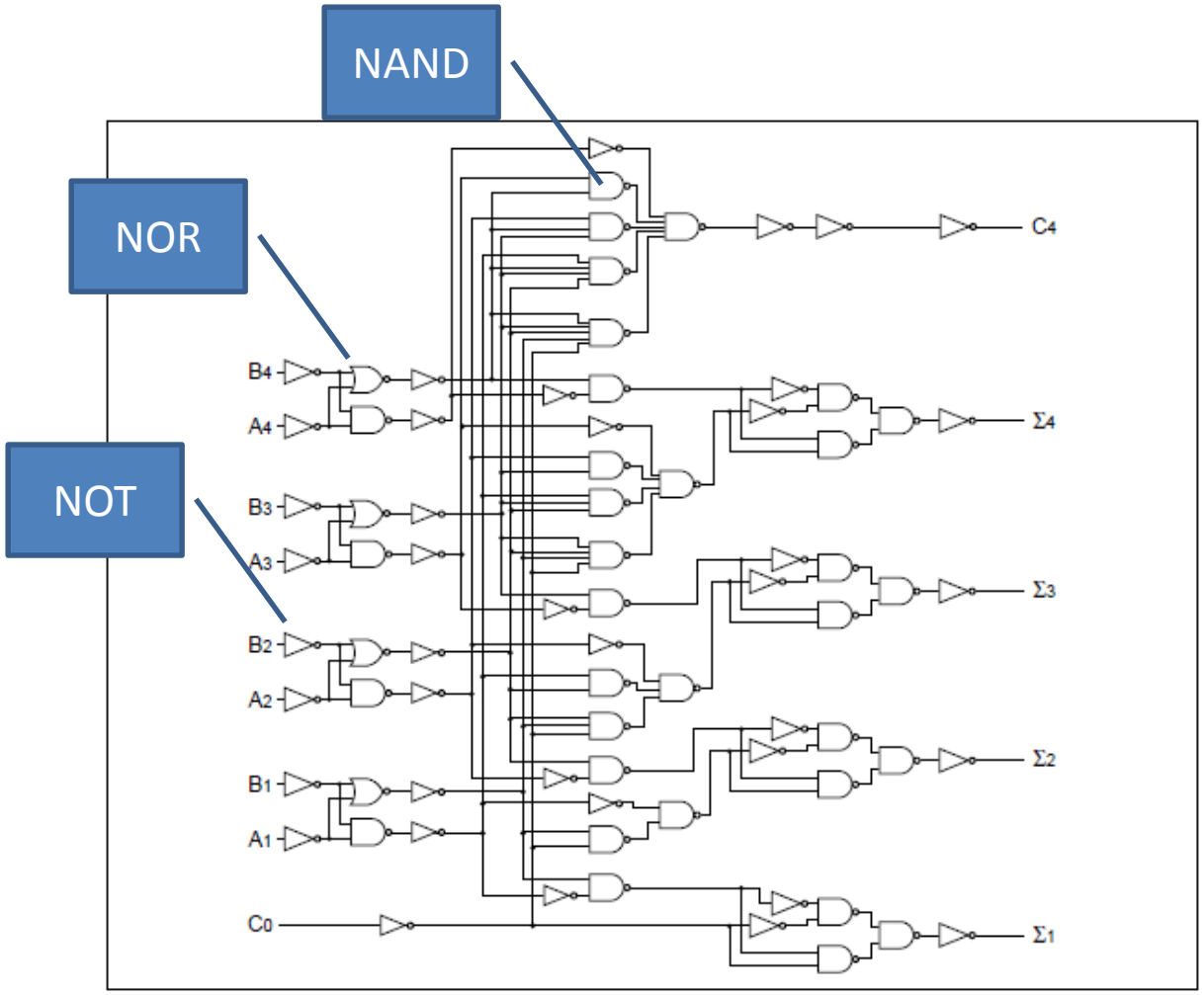
- Microcontrollers are non existent.
- Computers are expensive.
- So most devices used discreet/individual logic chips to make things work.
  - Or they still used gears, cams, and other mechanical marvels to perform logic.
- The most famous of these chips was the 74xx series.
- They provided And, Or, Nor, Not, Xor, etc.
- With these building blocks you could design more complicated things.

4 Bit Adder

- Suppose your product needed to add two 4 bit (0-15) numbers. Without a micro what would you do? Well, you would spend a few days and come up with this circuit:



More Detail



Complexity Grows Quickly

- The 4-bit adder needs 79 gates (NOR, NAND, ..,) and over 200 places to goof up the wiring.
- An 8-bit adder is not just twice the complexity of the 4-bit adder. It is more like 3 times.
- And this only allows you to add numbers. It does not allow for storage, multiplication, or anything else.
- This is why a simple microcontroller has 100K+ logic gates!

No Demo ☹

- I discarded all my 74 logic chips a few years ago.

So what to do?

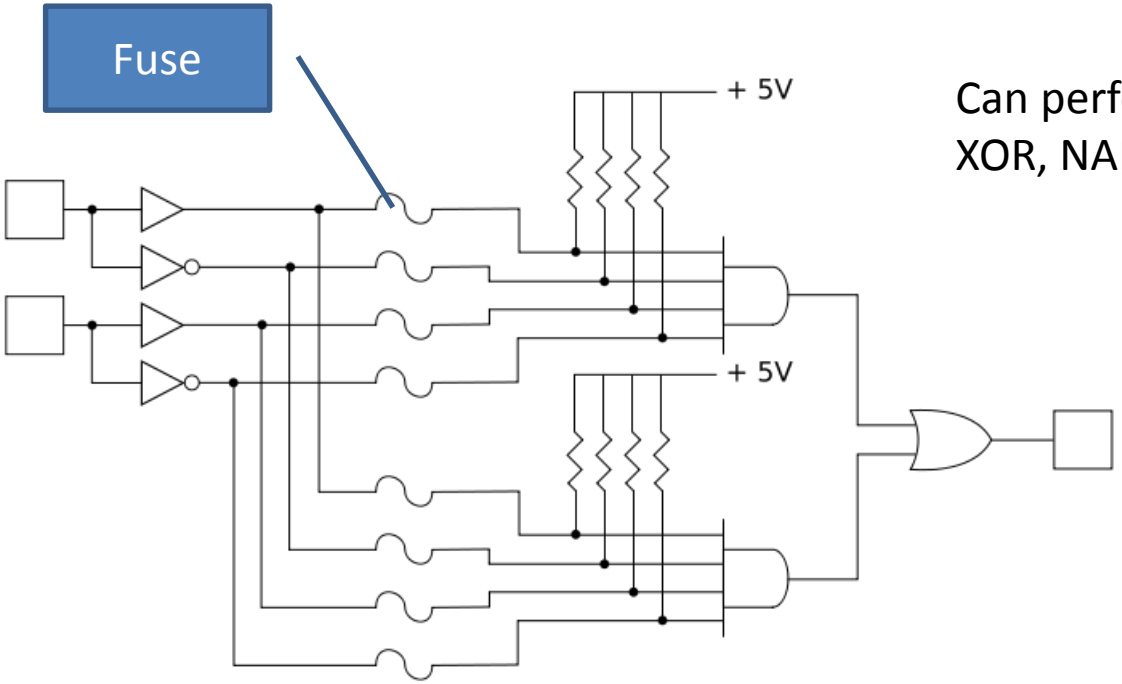
- If your project grew, so did your part count and so did the possible error sources.
- At some point you may have been forced to get a custom chip made.
- The 4-bit adder is such an example. Enough people needed one that a chip was made just to perform that function (7483).
- But custom chips were EXPENSIVE to have made for you!
  - So maybe you went back to gears and cams ☹

Demo

Along came the PAL

- Someone got the brilliant idea of making a ‘generic’ logic chip. A Programmable Array Logic.
- The idea was to put a bunch of logic gates into a chip, but to allow the user to make the internal connections during programming.

A Simple PAL



Simplified programmable logic device

Can perform NOT, OR, AND, XOR, NAND, NOR

So to program this PAL, you had to burn out the unneeded fuse.
It was a one-time programmable device.

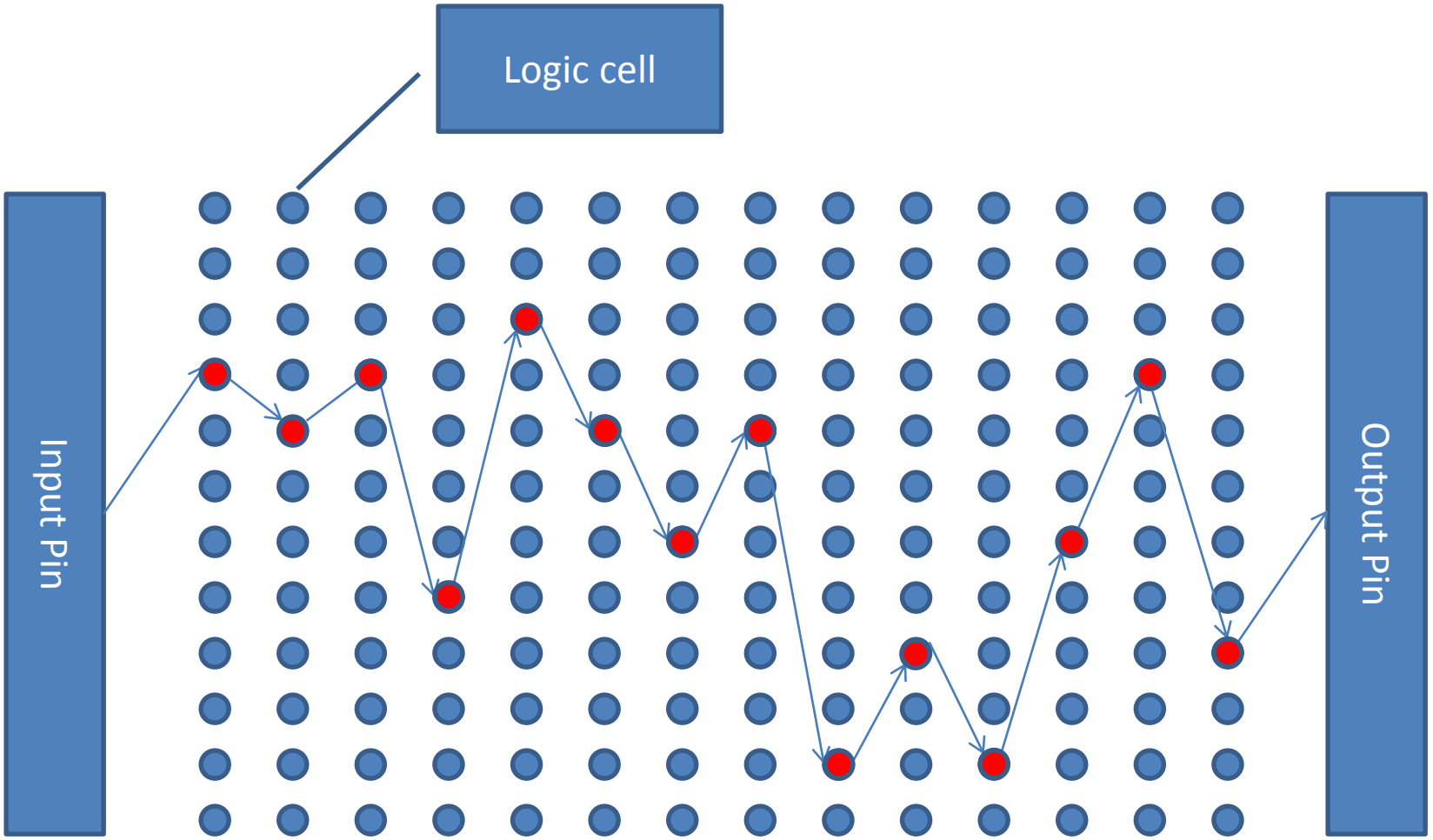
Fast forwarding

- PAL's got bigger and bigger
- Some PAL's used memory for the fuses and could be reprogrammed! (Happy days!)
- PAL's hit the wall and CPLDs started to arrive.
  - CPLD is nothing more than a big fat juicy PAL.
- CPLDs got bigger and bigger.
- But then they hit the wall.
  - But they are still used today for glue logic apps.

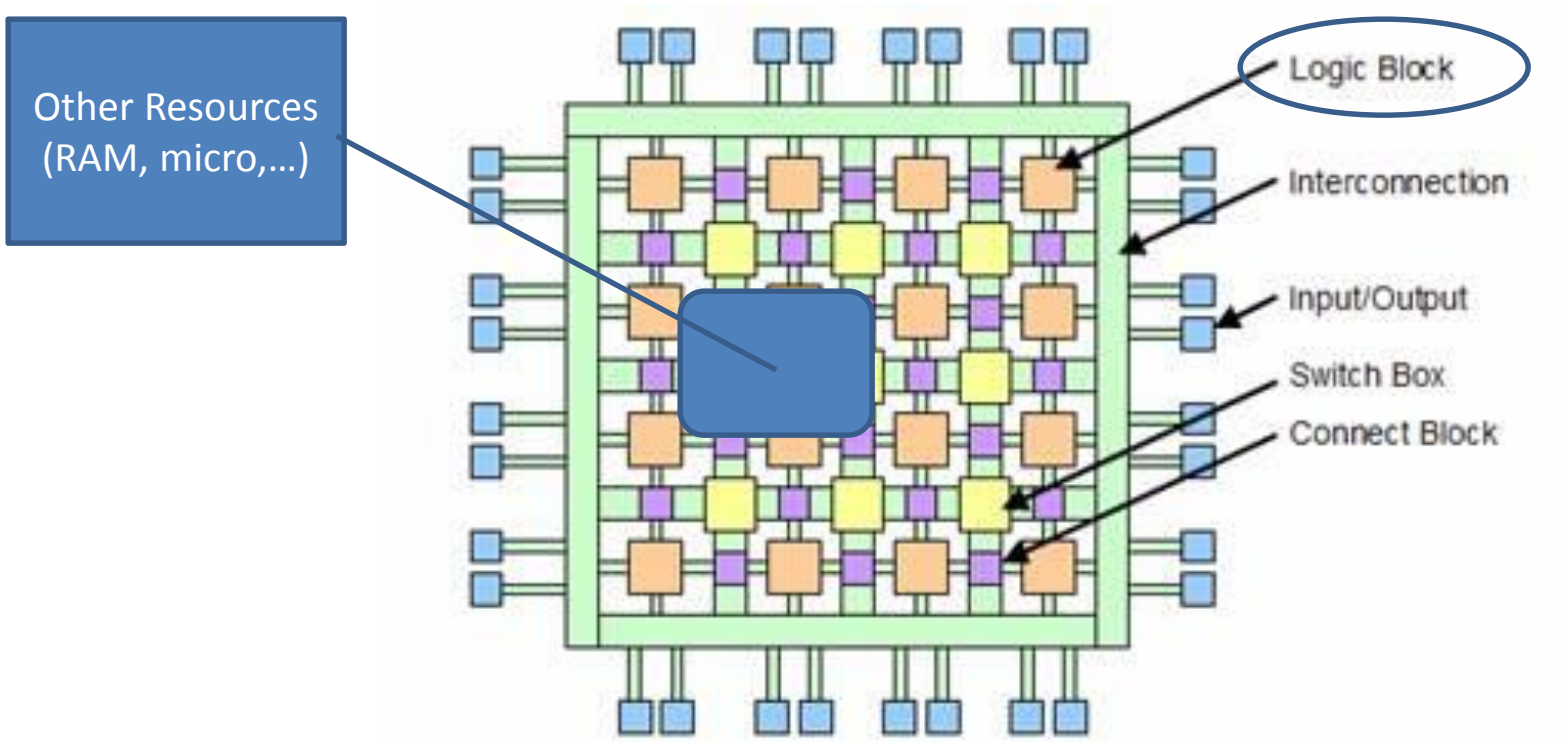
Birth of the FPGA

- Altera introduced the first FPGAs in 1984.
- These had logic, memory, and clock resources all in one reprogrammable chip.
- Programmable logic was now advanced enough for them to become sentient and to become our overlords.
  - See SkyNet

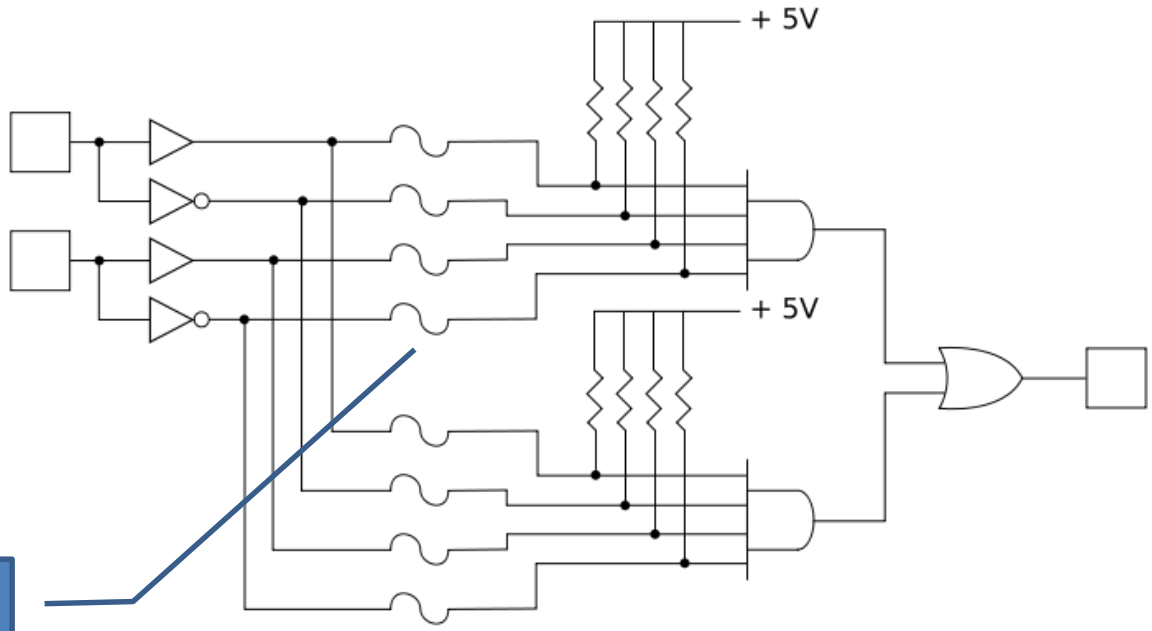
FPGA (reminder)



A Modern FPGA



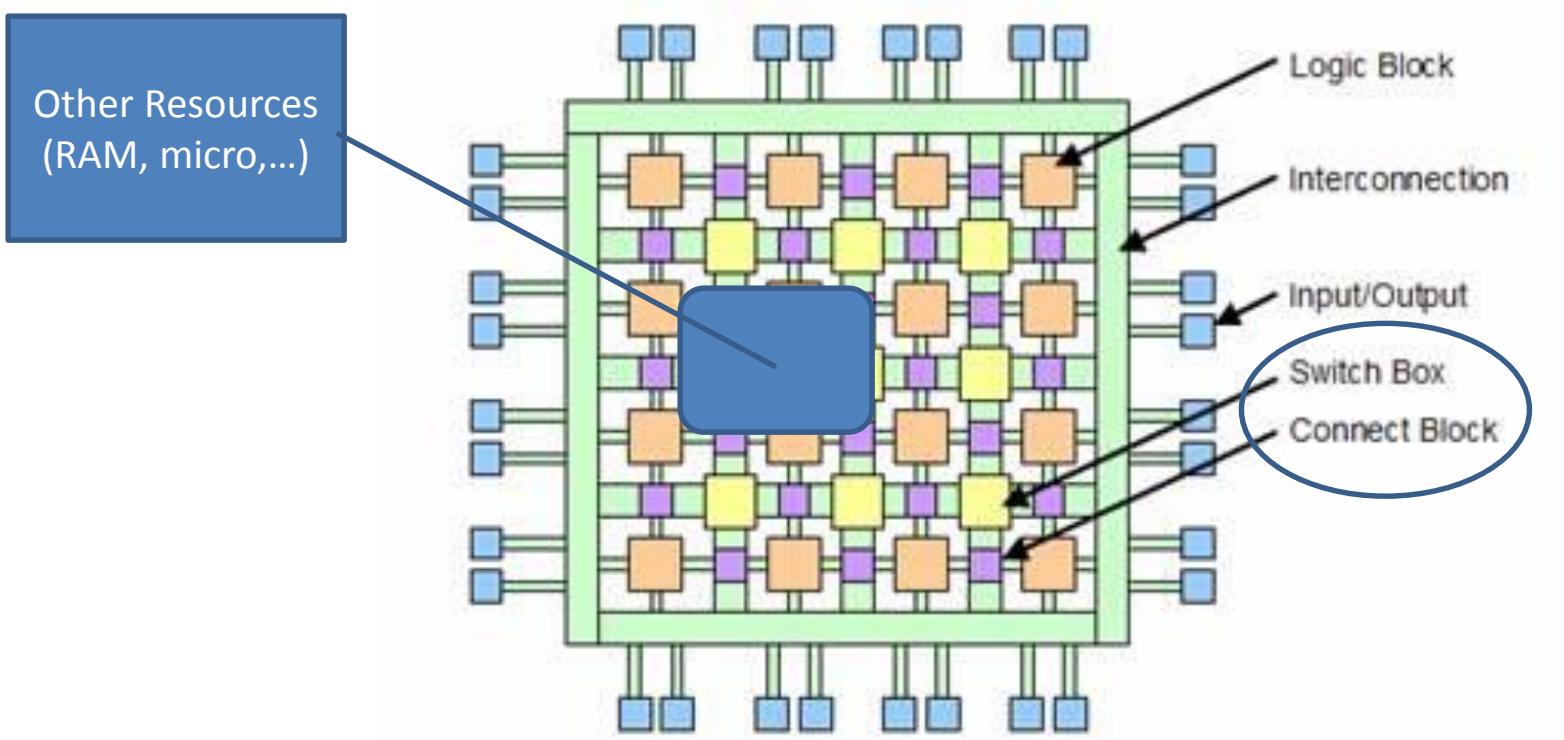
Logic Cell



Now Implemented as a look up table

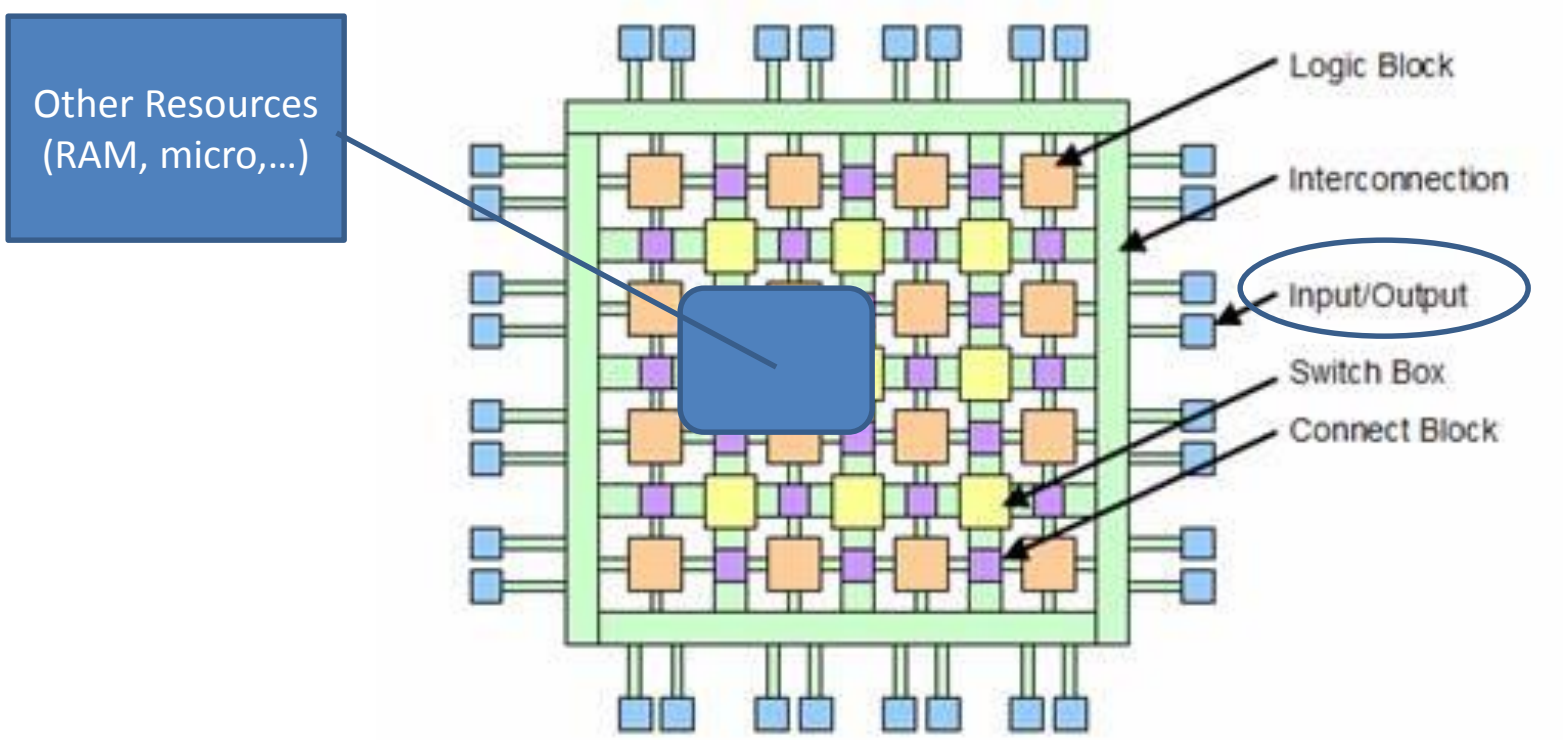
Our Friend the PAL/CPLD logic block

A Modern FPGA

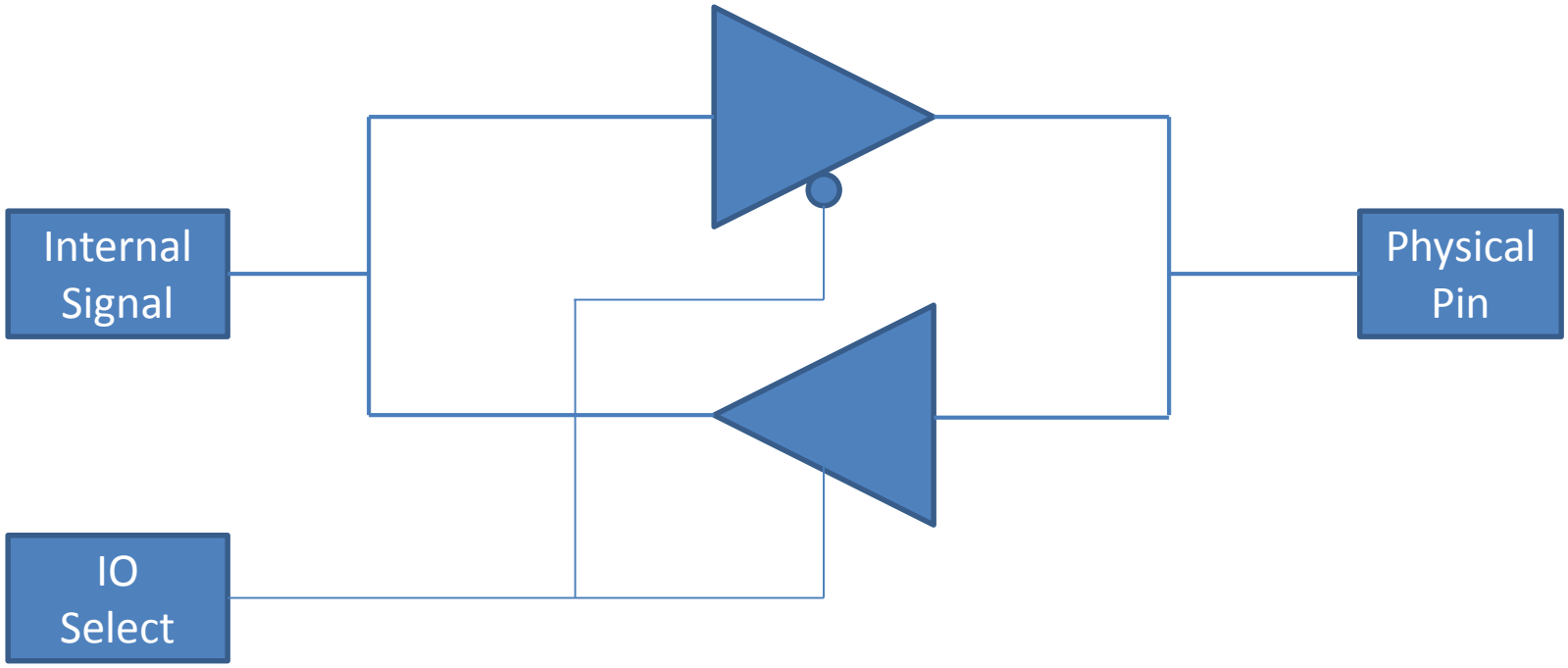


A Switch/Connect Blocks

A Modern FPGA



IO Pins



Often includes debouncing, differential drive, various voltage levels, etc.

So Why Would I Want One?

- In all likelihood you do not need one.
- The task:
  - Needs to be done fast
  - Has few complicated decisions
  - Can be done mostly on information you have now
    - Not what you had or will have.
  - Needs to meet critical timing (like video)
  - Cool factor

How much?

- FPGA's vary in cost from a $2 to $10K each.
  - Low end FPGAs and CPLDs are mostly equivalent.
- It all depends on:
  - The number of pins
  - The complexity of the part
  - The amount of memory
  - The speed of the part
  - Embedded processor or other special features

Start Simple

- Find a cheap FPGA development board and get your feet wet.
- The Mojo by EmbeddedMicro.com is a good choice. ($75)
  - Great tutorials, forum, add on boards,
- Other choices range from $30 to $500

Sizing a Part

- Trying to size an FPGA is complex.
- Your best bet is to:
  - Pick a vendor (Xlinx / Altera)
  - Install their tools
  - Code your application
  - Compile
  - Then place it on a part.
  - If it does not fit, it will tell you.

Consider Special Resources

- Based on your design, you may want to consider:
  - Embedded Micro (Like an ARM, PowerPC, …)
  - Amounts of memory and logic
  - Hardware support for
    - SDRAM,
    - PCI Express,
    - ADC, DAC
    - USB
  - Number of pins
  - Package type (BGA, QFP, …)

Steps to ~~Program~~ Configure

- Write some ‘code’ (Verilog, VHDL, …)
- ‘Compile’ it

Classic Approach

- ‘Download’ to FPGA
- Run and test it
- Rinse and repeat

Modern Approach

- Simulate the code
- Rinse and repeat

Write Some Code

- Each major vendor has their own tool suite.
  - Entry level tools are often free
  - Enterprise versions can cost thousands
- Two basic languages are Verilog and VHDL
  - Each has good and bad points
  - Each has a loyal following
  - May be application and region specific

What Does Verilog Look Like?

```
module servo (
    input clk,
    input rst,
    input [7:0] position,
    output servo
  );

  reg pwm_d, pwm_q;
  reg [19:0] ctr_d, ctr_q;

  assign servo = pwm_q;

  always @(*) begin
    ctr_d = ctr_q + 1'b1;

    if (position + 9'd165 > ctr_q[19:8])
      pwm_d = 1'b1;
    else
      pwm_d = 1'b0;
  end

  always @(posedge clk) begin
    if (rst) begin
      ctr_q <= 1'b0;
    end else begin
      ctr_q <= ctr_d;
    end

    pwm_q <= pwm_d;
  end

endmodule
```

Defines a chunk of your design

Looks **deceptively** like C!

Only gets evaluated if the variables inside change

Only gets evaluated on the rising edge of ‘clk’

Discrete 4 Bit Adder

b4a = ~ b4
a4a = ~ a4
b3a = ~ b3
a3a = ~ a3
b2a = ~ b2
a2a = ~ a2
b1a = ~ b1
a1a = ~ a1

t1 = ~(b1b & ~(a1b))
t2 = ~(t1)
t3 = ~(~(cin))
t4 = ~(t2 & t3)
t5 = ~(t1 & ~(cin))
s1 = ~(~(t4 & t5))

b4b = ~ (~ (b4a ^ a4b))
a4b = ~ (~ (b4a & a4b))
b3b = ~ (~ (b3a ^ a3b))
a3b = ~ (~ (b3a & a3b))
b2b = ~ (~ (b2a ^ a2b))
a2b = ~ (~ (b2a & a2b))
b1b = ~ (~ (b1a ^ a1b))
a1b = ~ (~ (b1a & a1b))

l1 = ~ a4b
l2 = ~ (a3b & b4b)
l3 = ~ (a2b & b4b & b3b)
l4 = ~ (a1b & b4b & b3b & b2b)
l5 = ~ (b4b & b3b & b2b & b1b & Cin)
m1 = ~ (l1 & l2 & l3 & l4 & l5)
c4 = ~(~(~(m1)))

….

n1 = ~(b4b & ~(a4b))
n2 = ~(a3b)
n3 = ~(a2b & b3b)
n4 = ~(a1b & b3b & b2b)
n5 = ~(a1b & b3b & b2b)

p1 = ~(n2 & n3 & n4 & n5)
p2 = ~(n1)
p3 = ~(p1)
p4 = ~(p2 & p3)
p5 = ~(n1 & p1)
s4 = ~(~(p4 & p5))

4 Bit Adder

- Wire a[3..0]
- Wire b[3..0]
- Wire result [4..0]

- Result <= a + b;

Resources for Coding

- Most vendors provide building blocks for things like UARTS, FIFOs, SPI, …
- EmbeddedMicro.com (Mojo) has good tutorials.
- OpenCores.com has lots of free building blocks as well.
- YouTube has tutorials.
- Google is your friend

~~Compile~~ Synthesis It

- Just like most development environments, you can run ‘Build Project’.
- Building is a multistage operation.
  - Syntax checking
  - Synthesis
  - Routing
- Depending on the size of the project, this could take a few minutes or a few hours!
- Output is a binary file where each bit turns on or off a connection inside the FPGA.
  - These tend to be pretty big.

Download

- Most modern FPGAs do not store their own configuration file.
- They need another source to send over the configuration data.
- This can be a memory chip or it could be another processor that feeds it the data
- Most FPGAs have multiple ways to configure it based on the state of a few bootstrap pins.

Download

- So ‘Downloading’ may involve:
  - Programming an external memory chip
  - Putting the configuration file someplace another processor can read from to write it to the FPGA.
    - This might be FLASH, and SD card, or other memory
  - How this is done varies by the development board you buy.
    - The Mojo has an Atmel part to allow you to program either a FLASH part or the FPGA directly via USB.

Run and test it

- On power on, the FPGA looks at its mode pins to decide what to do.
  - If it is told to read from a memory device it does that.
  - If it is told it will get it pushed to it, it waits for the data to be clocked over.
- When it gets the last byte, it will start to run.
- Now you can see if it does what you want it to.
- If not, go back and correct the code.

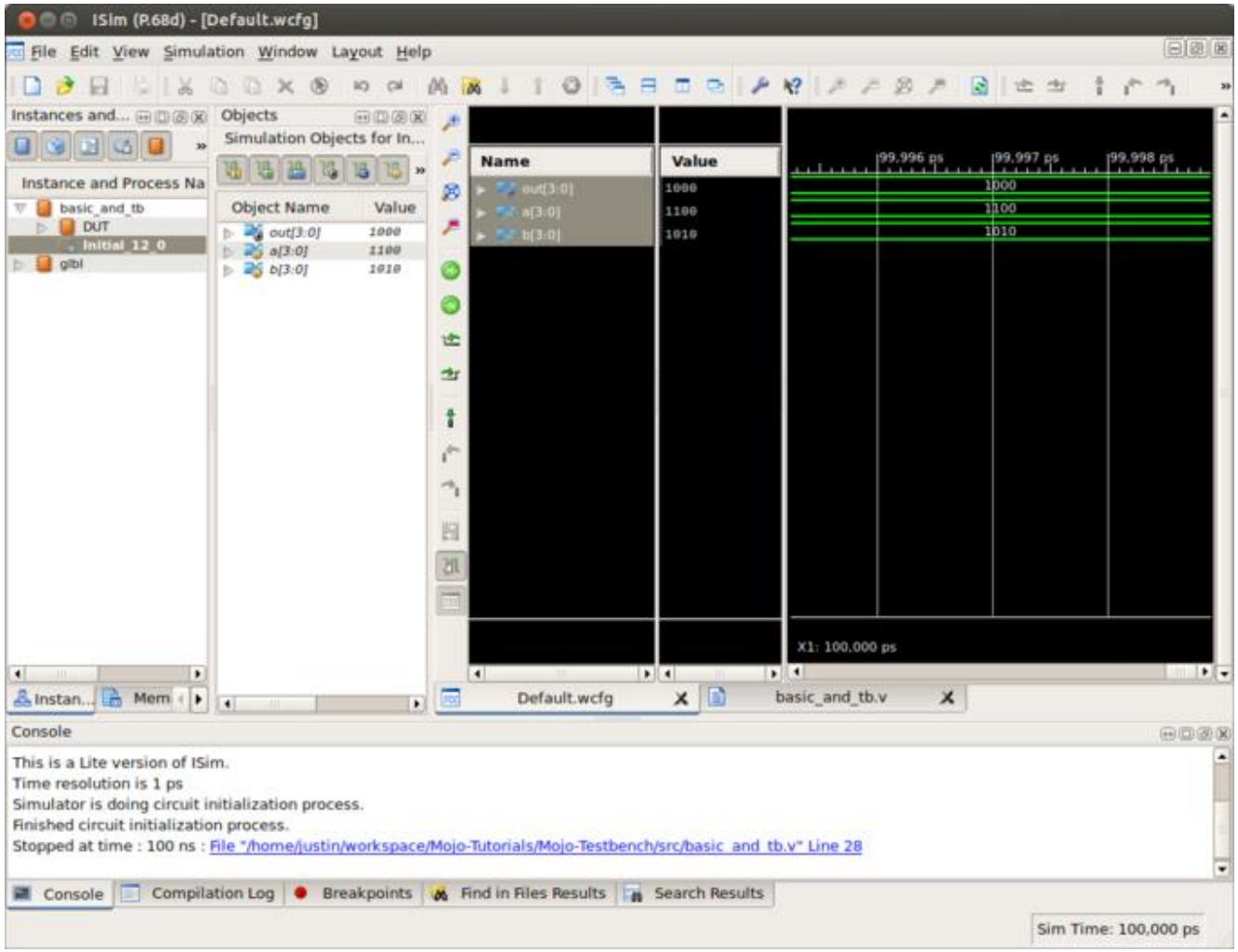
Debugging (traditional)

- Since an FPGA runs all of its code all the time, a classic single step debugger is meaningless.
- The correct way to debug is to bring intermediate states to debug pins where you can see them on a scope.
- Vendors may also provide logic blocks you can link with your design to store various signals into a block of internal memory.
  - You can then review it later.
  - Like having a built in logic analyzer

Debugging (modern)

- Vendors now supply simulation tools that allow you to test your design without real hardware.
- You supply a file with how each pin should be driven at each moment in time.
- The tool then shows you with a strip chart what your outputs are doing.
- Simulation has come a long way in recent years. Often its good enough to account for all but the weirdest problems.

Xlinx Sample Simulation

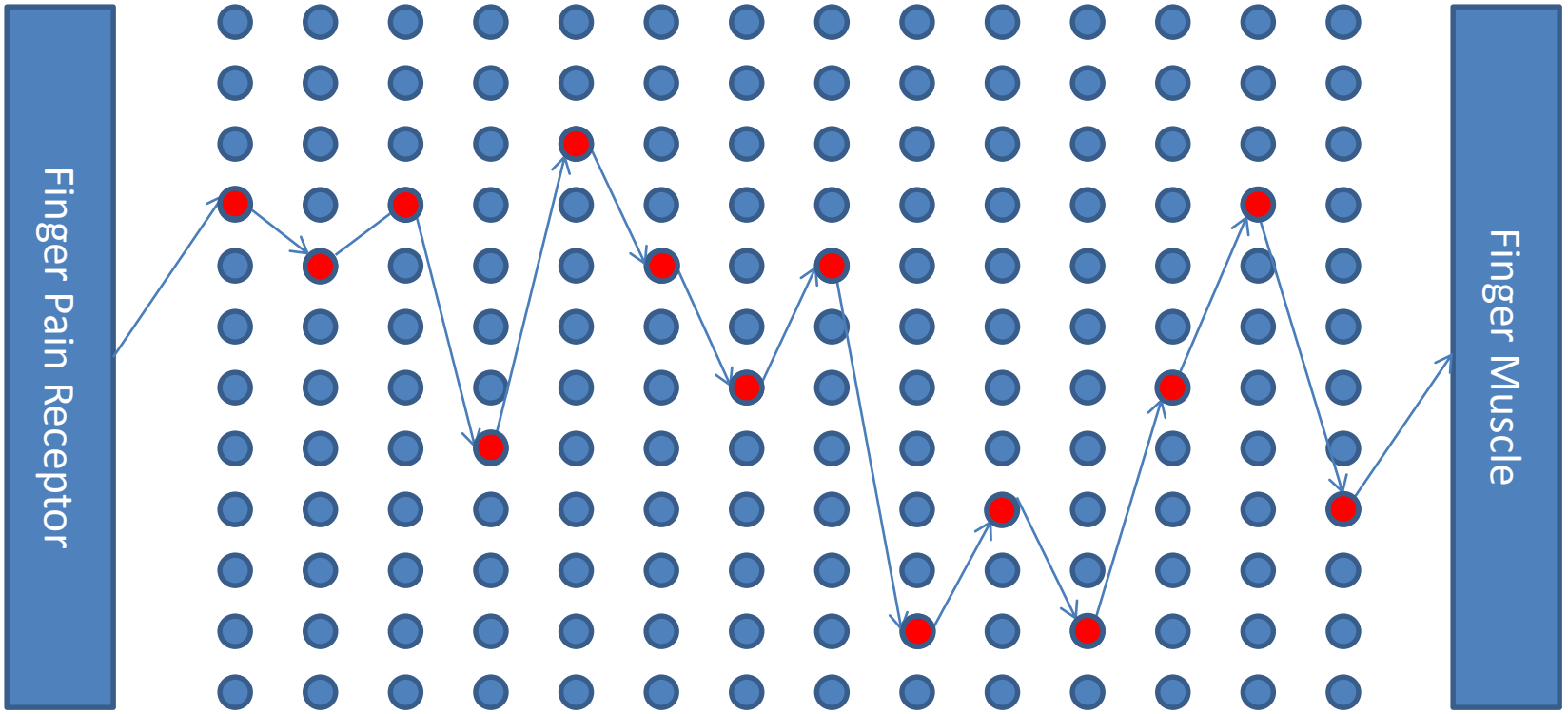


All at the Same Time

Since an FPGA runs all of its code all the time, a classic single step debugger is meaningless.

But its running code, how could this be?

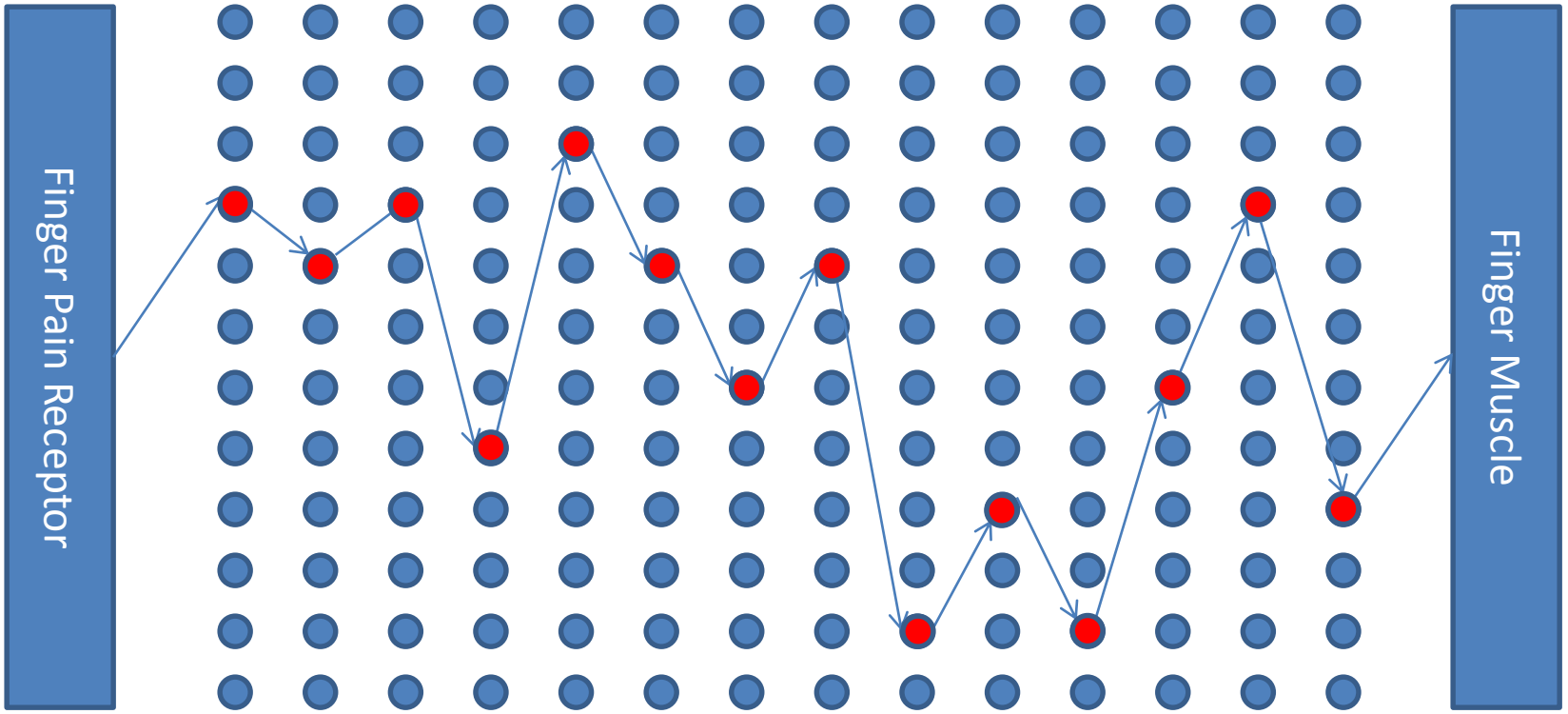
Back to biology



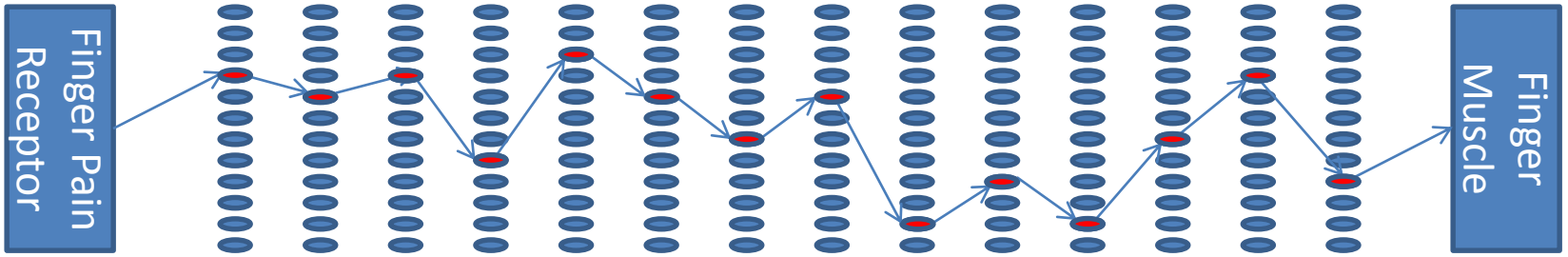
Back to biology

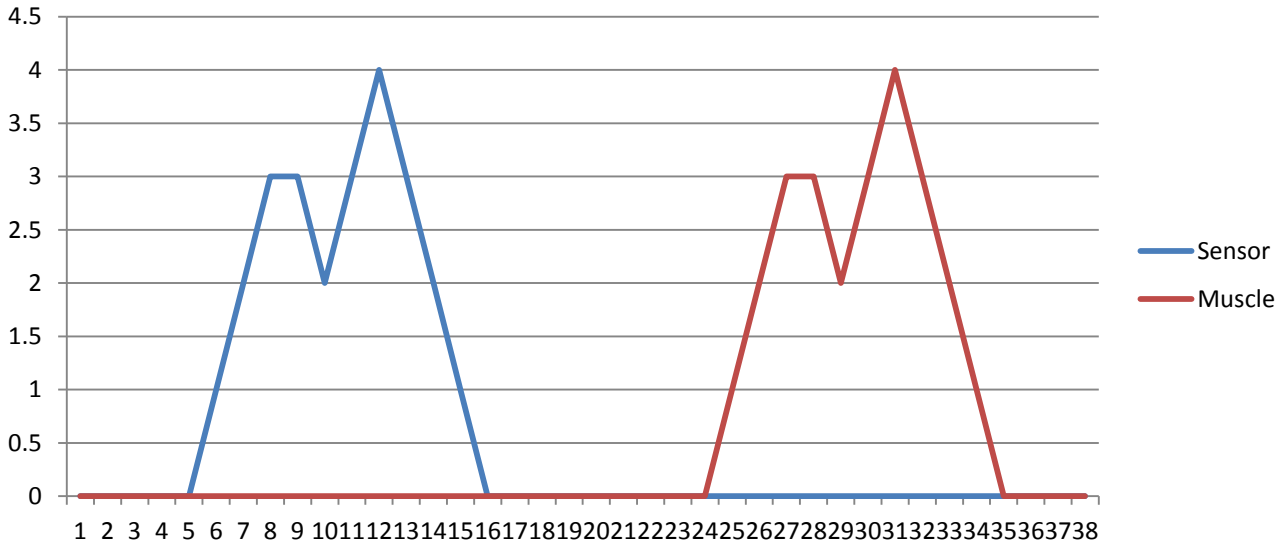
- The cells in your brain do not run in sequence
- Each cell takes in information from its connected neighbors, processes that data and then sends out a response.
- They do this pretty fast. As fast as the chemicals can be regenerated in the axon.
- All cells are doing “their thing” all at the same time.
- Although strictly incorrect, you might consider each logic cell to be its own extremely fast parallel processor.

Misleading Diagram



Misleading Diagram



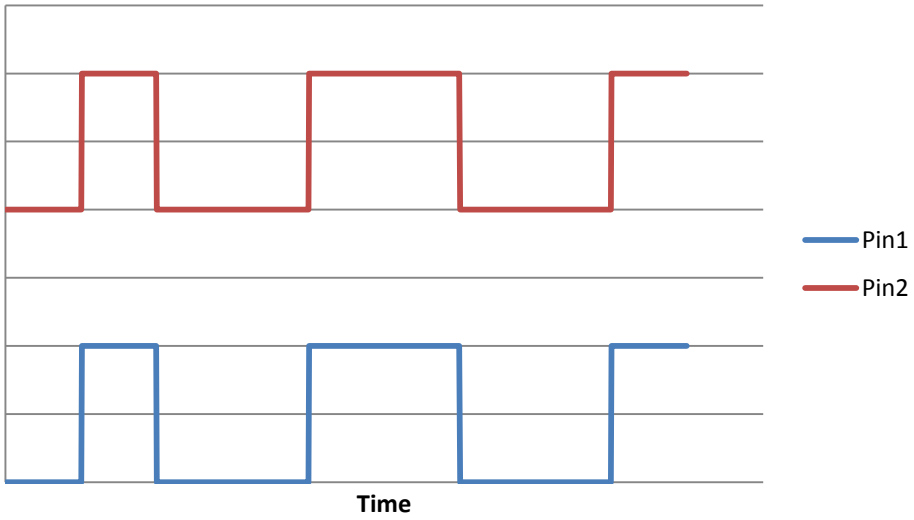


The Crucial Leap Of Knowledge

- **All** parts of your design are evaluated at the same time.
- If you don't like this, its up to **you** to impose discipline.
- Discipline can supplied by a clock (or other signal) to force your design to be synchronous.

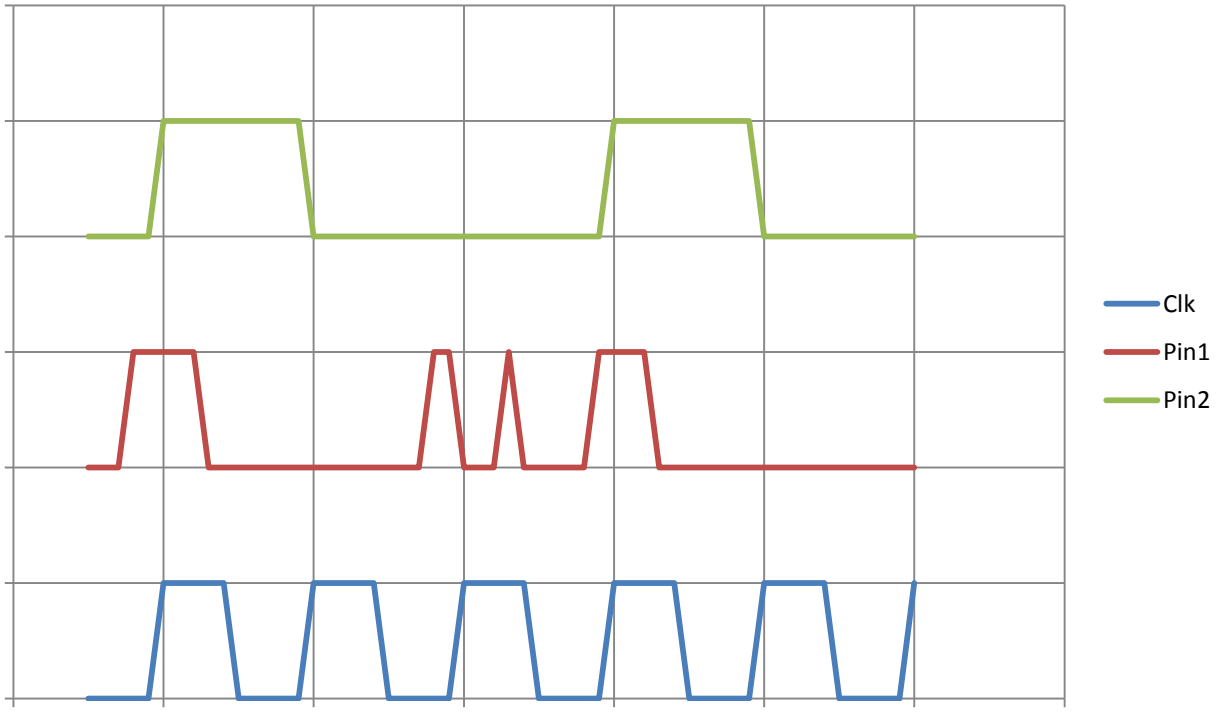
Asynchronous design

- Pin1 becomes the value of Pin2 **always**
- Need not happen on clock edges.
- In fact you don't even need a clock!
- Pin 2 follows Pin 1 by a small propagation delay.



Synchronous design

- Pin1 becomes the value of Pin2 at each clock rising edge.



Robot Applications

- Vision (blob detection)
  - CMU Cam is an FPGA based system
- Inertial Measurement Unit (IMU)
  - Integrates all the rates at high speed
- Laser range finder
  - Time to travel 10 feet is only 10 ns! Need a FAST way to measure time.
- Gate controller for hexapod robot
- Acoustic or IR triangulation
- Looks good on your resume!

How do I get started?

- Buy a low cost development board
  - A good choice is the Mojo at $75.
- Download the tools from the vendor
- Start watching/reading tutorials
- Write simple things to get started!
  - How about a 4 bit adder? ☺

My Plans

- Integrate an FPGA to my ARM processor
- Have FPGA perform high speed integration of my inertial sensors
- Have FPGA use forward looking camera for orange barrel avoidance.
- Have FPGA use upward looking camera to find sun (celestial compass).

My Status

- Purchased Mojo and daughter boards
- Have run many of the tutorials
- Have started to lay out robot board to include FPGA.
- Purchased fisheye lens to find sun.
- Having a good time, but it is a steep learning curve. The “all at once” is tough to comprehend!

Questions?