FPGA, ASICs, and CPU: When to Use What, Why, and How

Bryce L. Meyer

brycemeyer@att.net

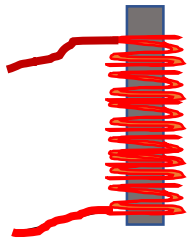
19 March 2020

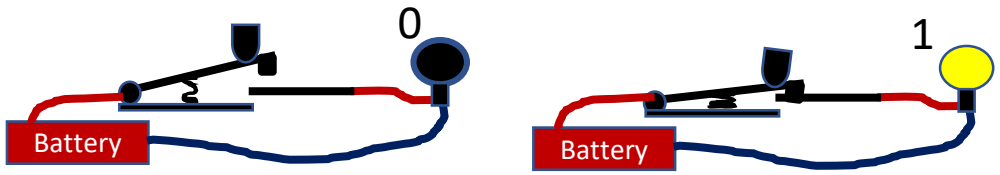
Outline

- History of Logic
  - Logic Gates 101
- Tradeoffs and Usage
  - Matrix of Integrated Circuits
- FPGAs: Who makes them for what?
- HDLs: VHDL and Verilog
- Quartus 101
- Alterra BeMicro10 Test Kit

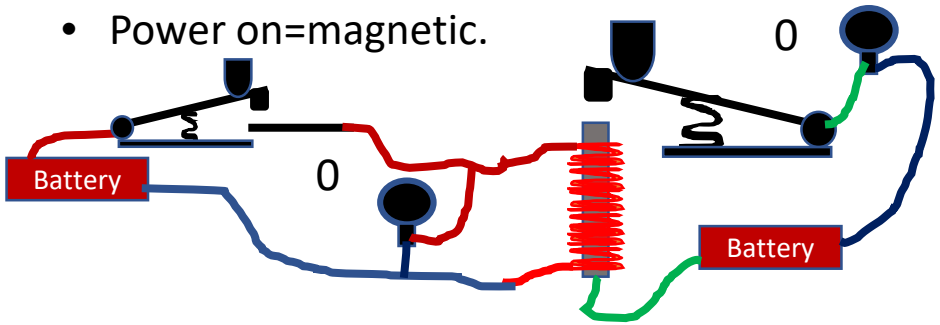
History of Logic:1800's

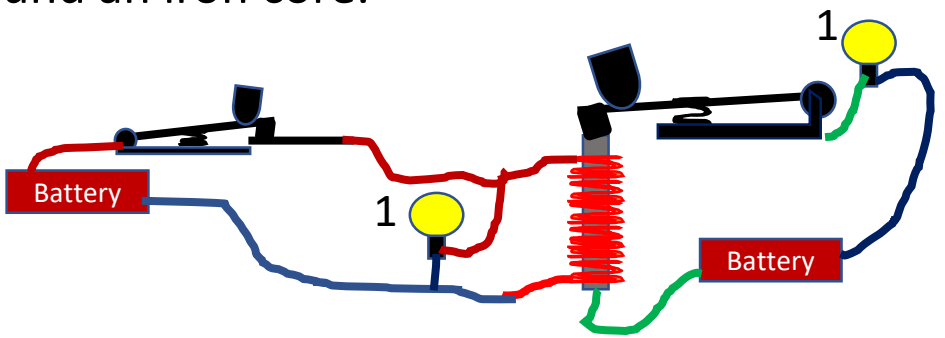
- 19th Century: Electromechanical Switches
  - Switches had to be physically moved to connect
  - 0=off, 1=on





- Add in the Electromagnet Coil: Wire wraps around an iron core.
  - Power on=magnetic.



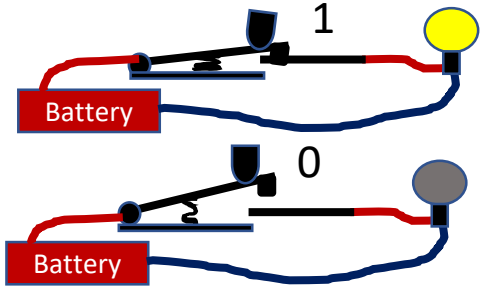


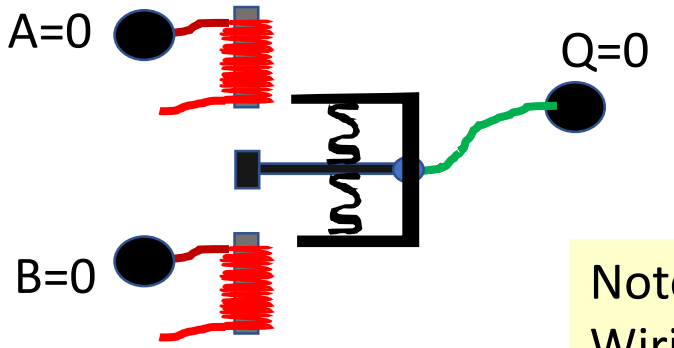
Note: ideally you might design to isolate from core, but not here…

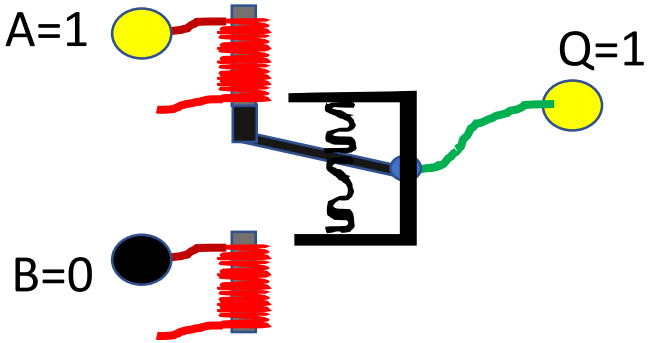
- Combined with springs and contacts: One switch can throw another, even bigger switch
  - This is how the telegraph and relays worked….eventually telephones

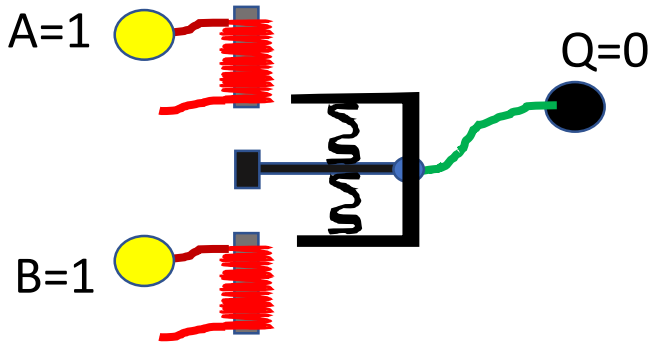
History of Logic:19th Century: Electromechanical Circuits

- Complex structures allow a variety of switch-logic structures
- You can describe the function of each in language and in tables

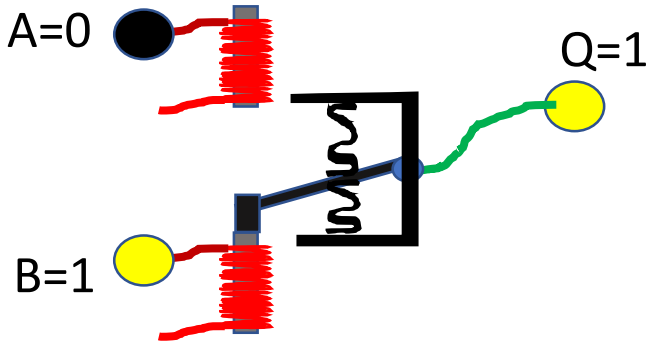








Note:
Wiring below simplified to show gate logic….

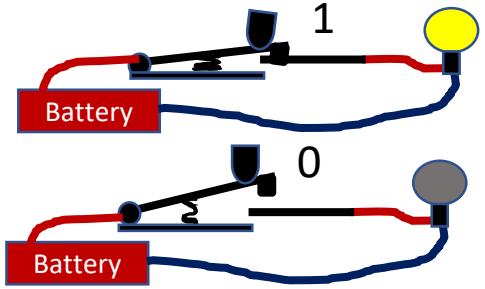


INPUT		OUTPUT
A	B	Q
0	0	0
1	0	1
0	1	1
1	1	0

*If **exclusively** A **or** B is on, Q is on, but if A and B are on, or A and B are off, then Q is off*

Gate Type:
XOR (Exclusive Or)

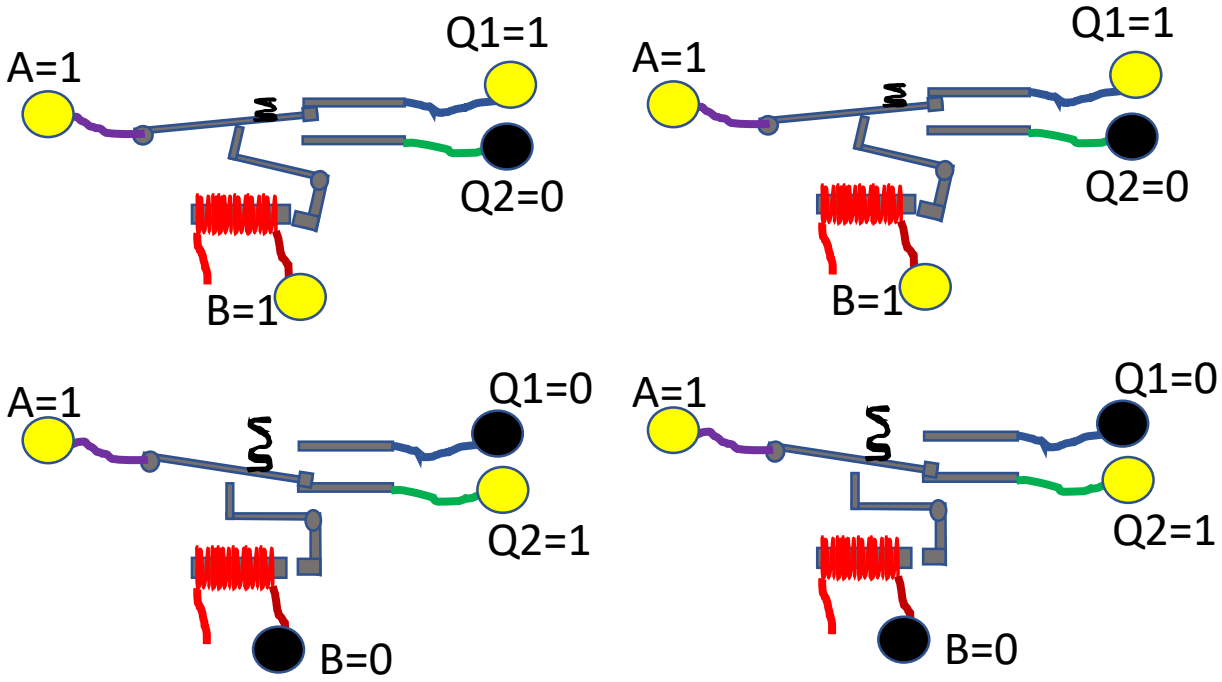
History of Logic:19th Century: Electromechanical Circuits



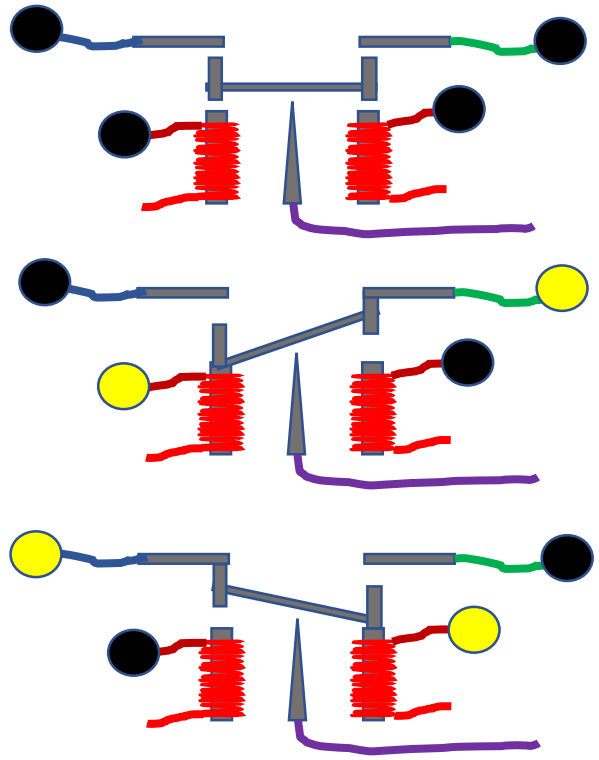
- Lots of other switch types emerged fyi…

Note:
Wiring below simplified to show gate logic….

Relay

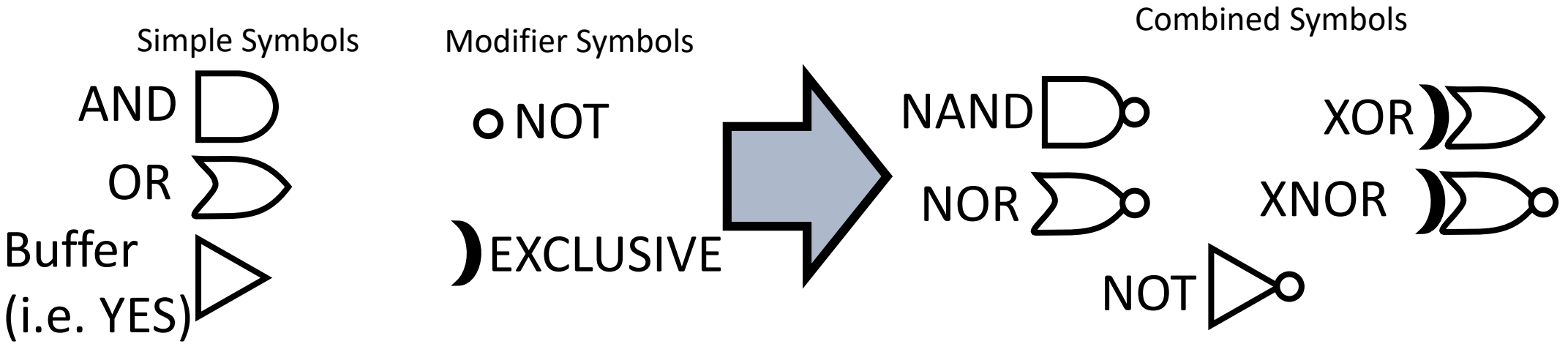


Flip Flop…



Standardized Logic

- Logic theory (discrete math) discovered by many cultures in antiquity refined into equations in the 18th century gained a physical reality in the switches (see prior slides)
- As a result in the 19-20th centuries the language of logic as standardized, in common gate types to allow mass production



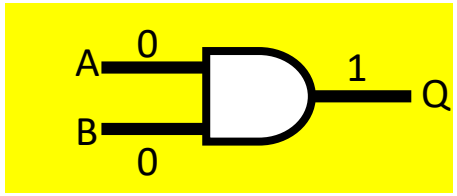
https://en.wikipedia.org/wiki/Logic_gate#Symbols

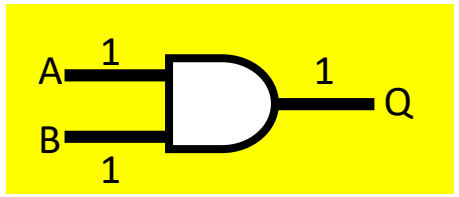
https://en.wikipedia.org/wiki/Mathematical_logic

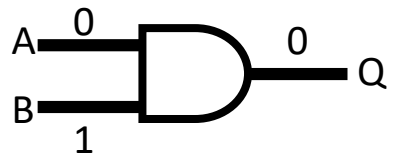
Logic Gates 101

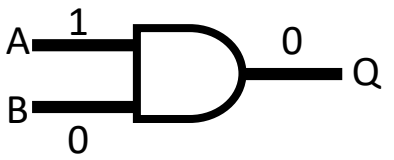
INPUT		OUTPUT
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

AND



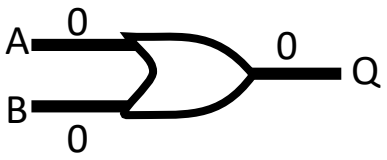


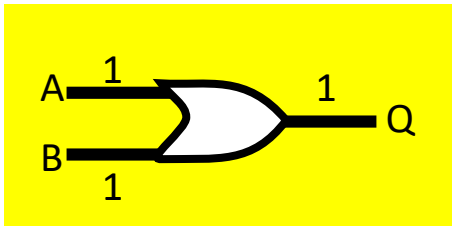


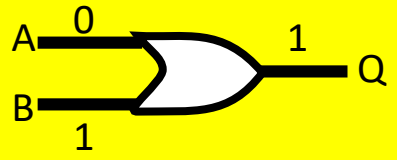


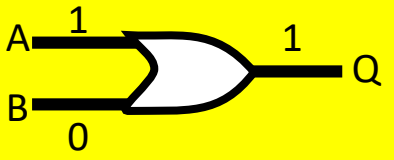
INPUT		OUTPUT
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

OR



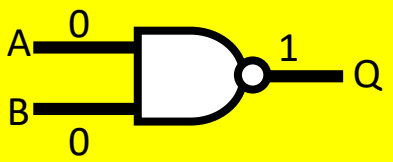


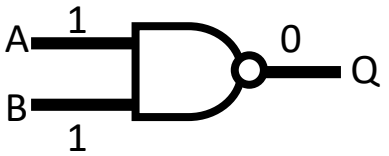


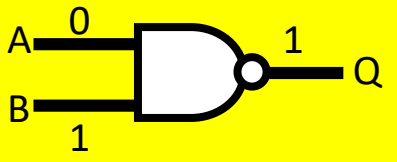


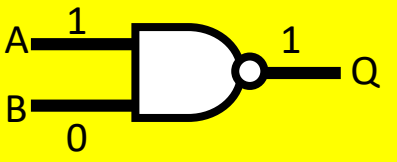
INPUT		OUTPUT
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

NAND



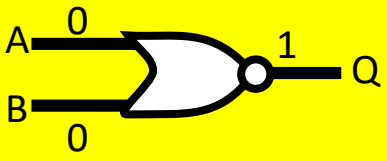


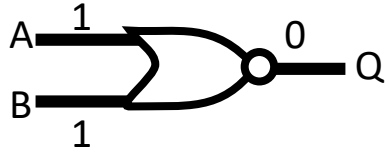


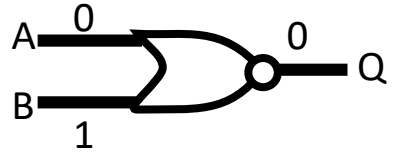


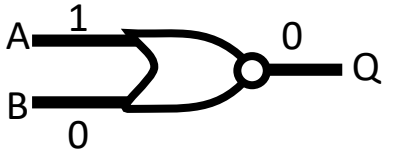
INPUT		OUTPUT
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

NOR



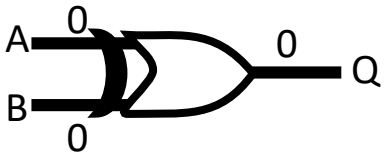


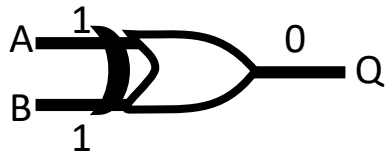


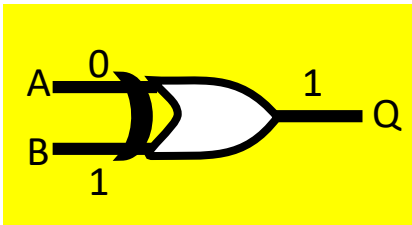


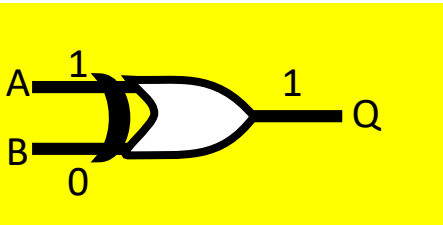
INPUT		OUTPUT
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

XOR



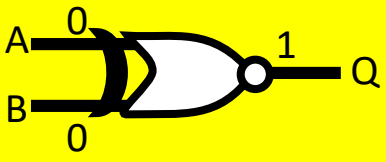


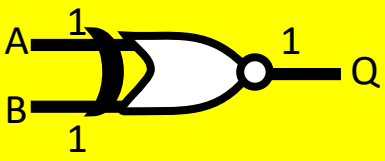


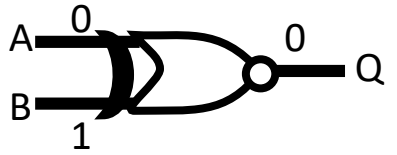


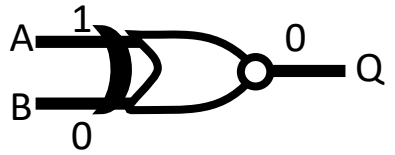
INPUT		OUTPUT
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

XNOR









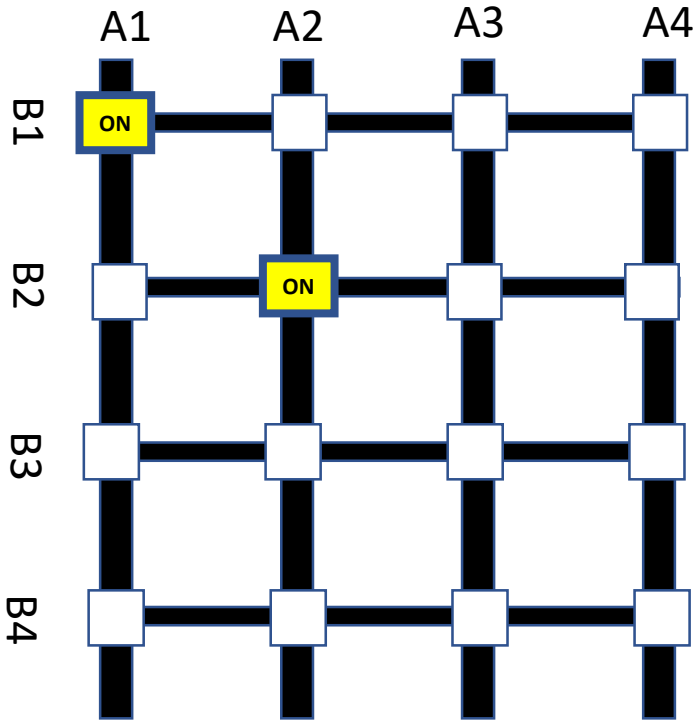
See: ANSI/IEEE 91 (and 91A)

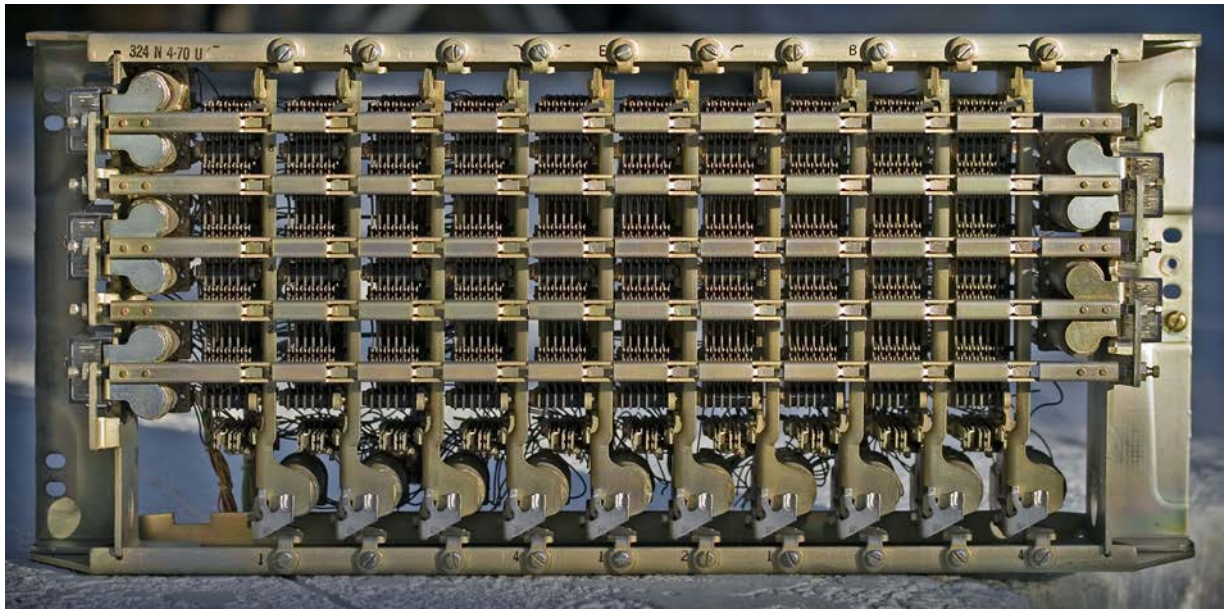
https://en.wikipedia.org/wiki/Logic_gate

Crossbar Switches

- Crossbar switches are arrays of electro-mechanical switches, to allow many things to link to many other things…emerged with telephone networks.
- Linking grid of wires are pulled or pushed to connect to other wires

A1 is connected to B1 because switch A1-B1 is on
A2 connected to B2 because switch A2-B2 is on.
Rest all off.

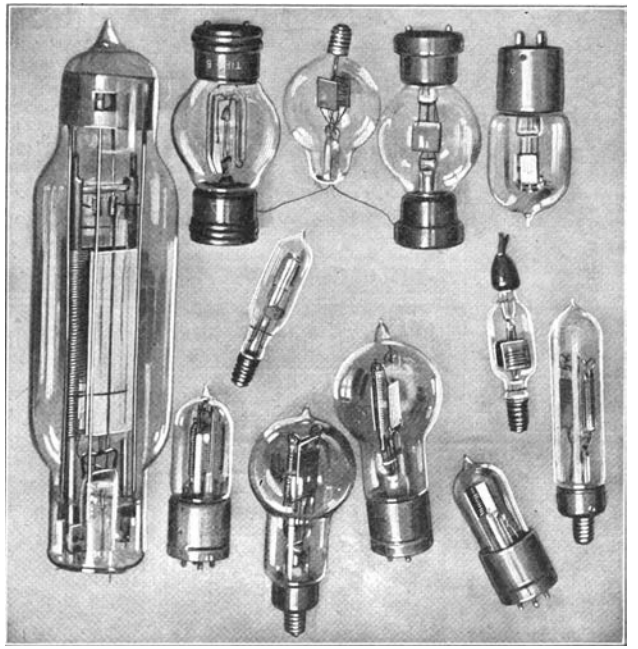




https://en.wikipedia.org/wiki/Crossbar_switch

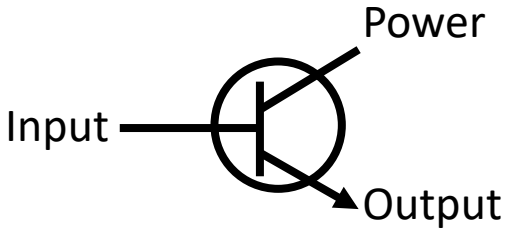
History of Logic: Vacuum Tubes

- Vacuum Tubes allow the same kinds of functions and gates as the electro-mechanical switches, but without sparking and physical movement.
- Combinations of Tubes can be made into standard gates.
- First Computers used vacuum tubes

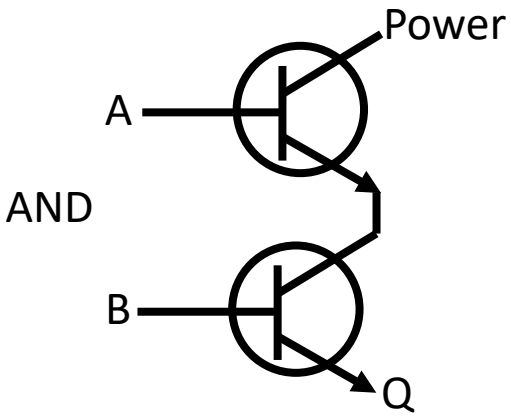


https://en.wikipedia.org/wiki/Vacuum_tube

History of Logic: Transistors

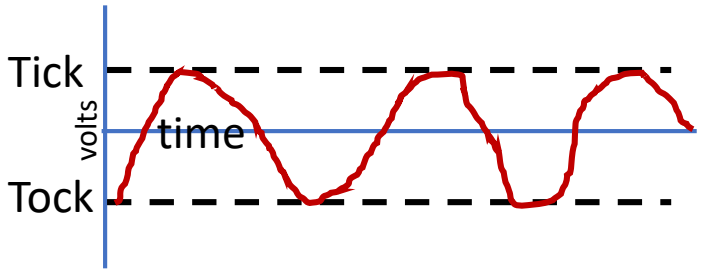


- 1926-1956: Transistors developed and patented.
- A Transistor is a solid state (i.e. no vacuum) component that is essentially a switch for our context (yes, more complicated then that but here, a switch)
- Combinations of transistors result in logic gates

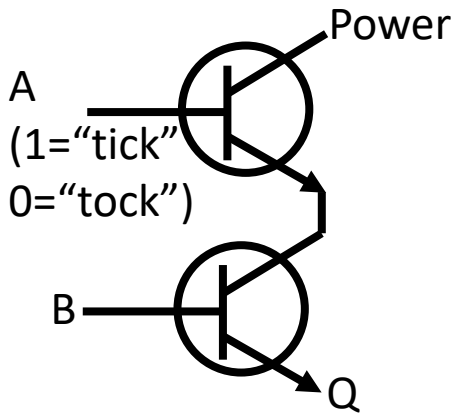


https://en.wikipedia.org/wiki/Transistor

Clocks and Timing



- Solid-State Clock signals were originally piezoelectric crystals that oscillated at a standard rate.
  - If the clock ‘ticks’, an accumulator circuit can count the ticks to tell how much time has elapsed since switched on
  - Ticks can trigger switches on or off
- Capacitors charge and discharge at a known amount of time
  - Can be used to provide a delay signal, i.e. switch on or off after a certain time
- Combinations provide system time for logic functions



Q=on if B=on and A=tick

Integrated Circuits

- In the 1960's+ Combinations of Transistors and other parts could be combined on a single chip.
  - If the chip is made to accept a language of instructions then provide a variety of responses, for general purposes you get the CPU (Central Processing Unit).
  - If the chip is crafted to take a specific set of inputs, discrete or in language, then output a specific set of responses, optimized for a specific purpose you get ASICs (Application Specific Integrated Circuits).
  - GPU (Graphics Processing Units) are both: They are optimized for vector graphics computation, but take a variety of inputs to get a variety of outputs
  - Field Programmable Gate Arrays (FPGAs) Fall in between CPUs and ASICs, sort of.

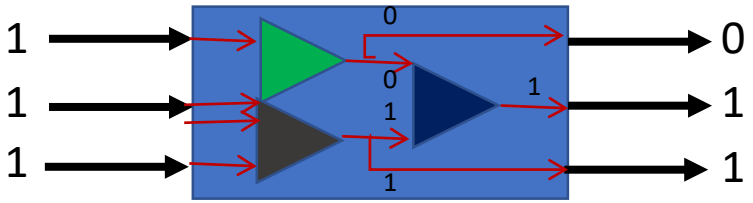
What are CPUs, ASICs, FPGAs

- Central Processing Unit or **CPU** (in this sense, a microprocessor chip like a Intel Core i7)
  - Programmable using common languages, which are compiled or interpreted into instructions, which are then computed into outputs
  - Benefits: Versatile
  - Costs: Speed and complexity of instruction software
- Application Specific Integrated Circuits or **ASIC**s
  - Etched or photolithographed to have the logic needed for a specific task. Simple input instruction set. Once made, response to instructions is invariant but fast.
  - Benefits: Speed and simplicity of instructions
  - Costs: Fixed behavior (Not Versatile), time to develop the circuit.
- Field Programmable Gate Arrays or **FPGA**s
  - Has a simple instruction set and response like an ASIC, BUT its logic can be reconfigured using a description language. Often used to prototype ASICs.
  - Benefits: Versatility, Some of the speed of ASICs, simple instructions once logic is set
  - Costs: Complexity to describe logic, loss of speed compared to ASICs.
- **N.B. Most modern ASICs and FPGAs have some complex features borrowed from CPUs**

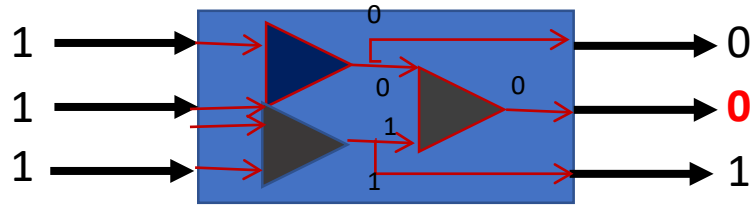
Note: FPGAs are used to prototype ASIC designs!

Logic Gates

Gate Structure A



Gate Structure B



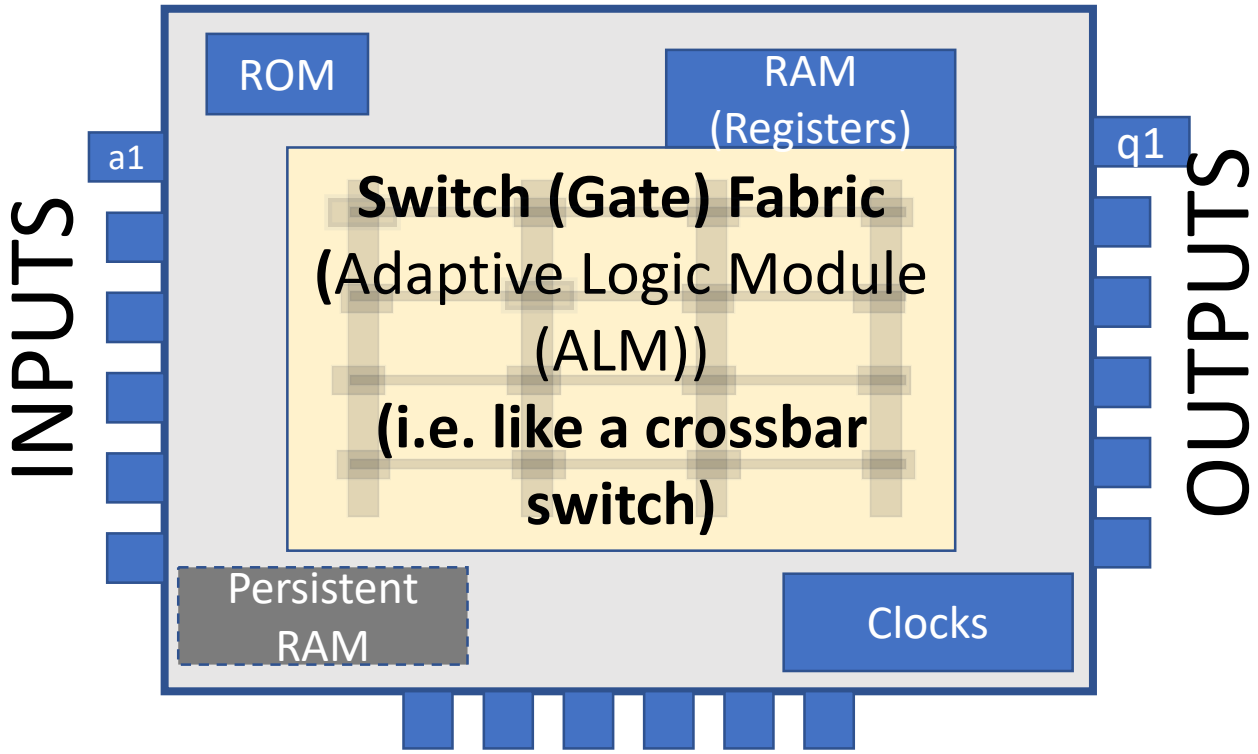
IF inputs are the same, 0 , else 1

IF inputs are different , 1 else 0

IF both 1, then 1, else 0

- In CPUs and ASICs the gates are the fixed, behavior is dictated by inputs only. CPUs have far more gates.
- To how a CPU behaves with the same inputs, just change the software.
- **To change how an ASIC behaves with the same inputs, it requires a change in hardware.**
- **In FPGAs, instructions (in Hardware Description Language, HDL) can change the gates (as A changes to B), so the same inputs in A and B, provide different outputs from A as in B.**
  - **The hardware is the same, but special gates, in a logical crossbar structure, allow changes in behavior dictated by the HDL commands.**

FPGA Top level



The Logic Module is configured by Hardware Description Language (HDL)
It can be parsed into Look Up Tables (LUTs) of various sizes. These correspond to the gate structure implemented inside the fabric for each input.

The bigger the LUTs, the slower the performance

Inputs			Outputs	
a1	a2	b1	q1	q2
0	1	1	0	1
1	0	0	1	0
0	0	1	0	0

https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf

FPGA: How does it work?

- To Implement the switch architecture, the switches can be these types:
  - SRAM based: Static RAM cells make up each gate. Programming in each cell determines gate structure, switch, etc. POWER OFF=ERASED (but can be reprogrammed).
  - Antifuse: Antifuse CMOS(Complementary metal–oxide–semiconductor) for each switch cell…WORM. i.e. you write it once, then it keeps the architecture power or not, but cannot be reprogrammed.
  - Flash RAM: Uses flash memory cells for each gate/switch. Persists when powered off, and can be reprogrammed.
- Groups of gates/switchs are aggregated into Logic Modules, each with LUTs (Look Up Tables, i.e. Truth Tables)

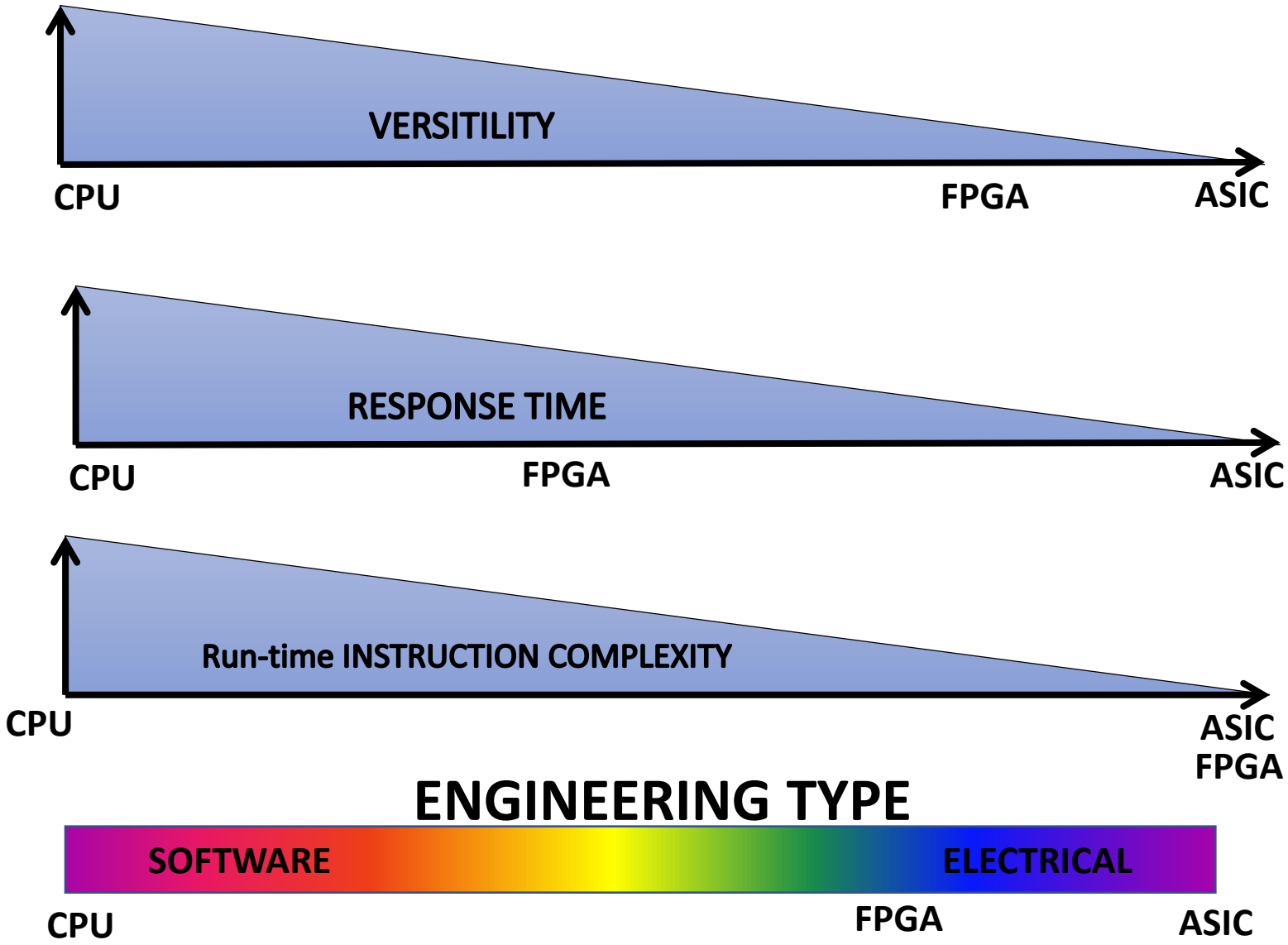
https://www.electronicdesign.com/technologies/fpgas/article/21801527/the-principles-of-fpgas

Side Benefit of FPGA: No Persistence

- Originally, FPGAs had no persistence (i.e. SRAM).
  - I.e. Loss of power erases switch architecture
- Very useful if you don’t want someone to know how your algorithms work, and need fast performance
- As a result, FPGAs became very useful in battlefield systems on the front line or in weapons.
- Modern FPGAs however, often have a bit of Flash memory that can persist the switch architecture (i.e. store the compiled HDL), or use Antifuse or Flash technologies.

CPU vs ASIC vs FPGA

CPUs have a high degree of versatility, limited mostly by programmer skill, But take longer to make decisions (i.e. gate)
And have high instruction complexity (i.e. software).
Changes are simply changes in sopftware.



ASICs have high performance, and simple inputs, but are limited to the designed purpose only. Most changes result in new hardware.

FPGAs have almost the same performance as ASICs, and have flexibility, but require more complex instructions, and are more expensive then ASICs or CPUs for the same purposes.
Some changes are simply HDL changes.

CPU vs ASIC vs FPGA: Development 1

- CPU:
  - Specify need then buy chips or whole boards/processing system.
  - Software Engineering Process
  - Compile or Interpret → Instruct → Calculate → Format → Output
- ASIC:
  - Circuit Engineering Process then build or buy circuits
  - Build = Burn, Etch i.e. implement purpose in hardware
  - (Compile or Interpret→)Instruct → Output
- FPGA
  - Circuit Engineering Process + Describe Circuit Change Logic (in HDL, similar to a Software Engineering Process)
  - (Compile or Interpret→)Instruct → Output then **Change**→(Compile or Interpret→)Instruct → Output
  - HAS THE ABILITY TO WIPE, and is not reverse engineerable like an ASIC

Process

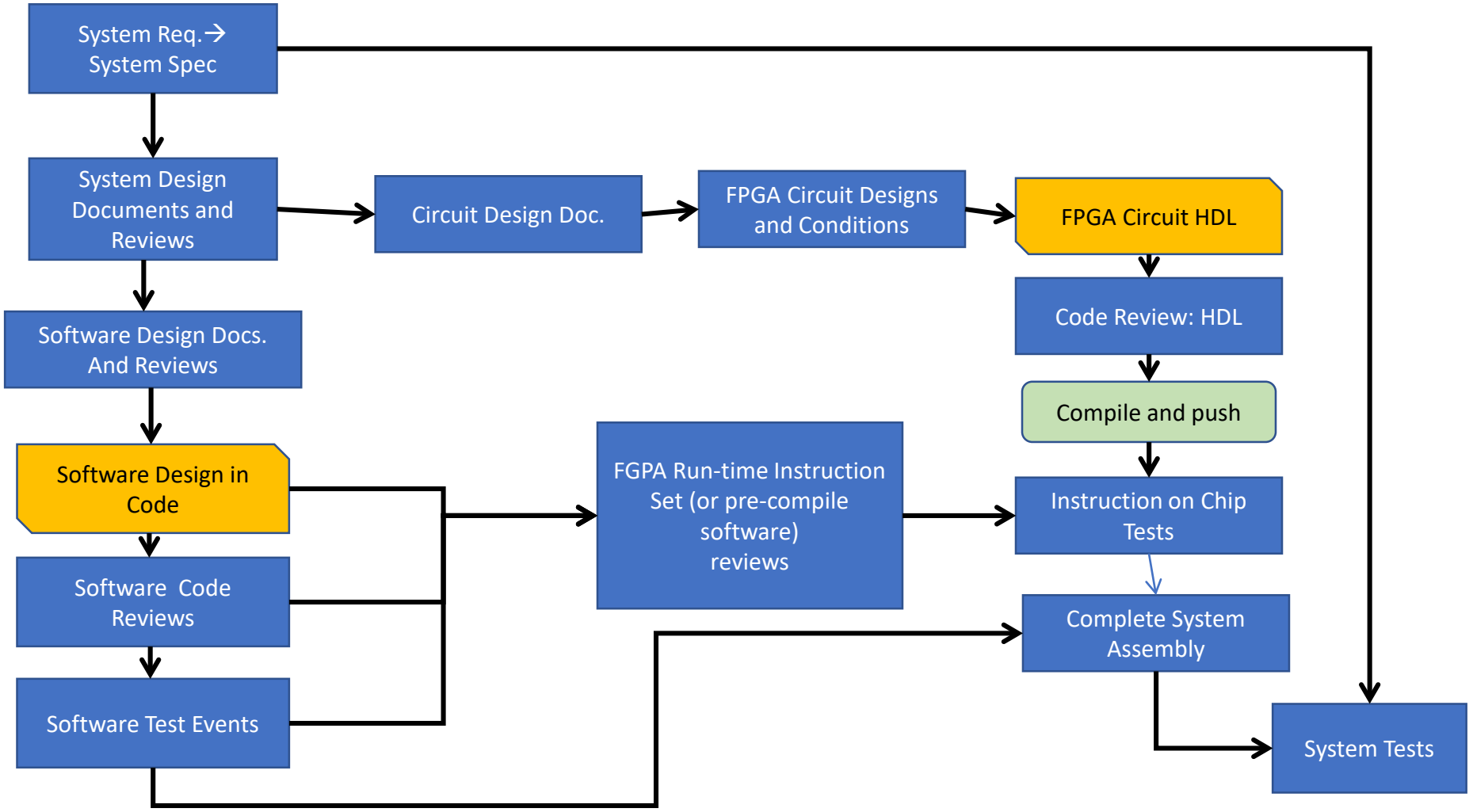
- CPUs: Once a processor is selected, Software Engineering is the dominant concern. Processor is static.
- ASICs: Circuit Engineering and Circuit building, and simplified instruction set design (I.e. simplified software Engineering) are core elements. Testing and redesign of the circuit, with costs to rebuild the circuit, are key. ASIC hardware may be rebuilt many times (incurring costs, reduced somewhat using simulation).
- FPGAs: Circuit Engineering and Software Engineering on two areas: Circuit Description and Instruction set design. FPGAs can shift circuit design quickly and cheaply in each test cycle since the Processor hardware is static (as in CPUs), though its logical circuit is changed. A core trade is if the instruction set is reloaded to the FPGA (assuming NOT CMOS)

https://www.intel.com/content/www/us/en/products/programmable/fpga/new-to-fpgas/resource-center/overview.html

Developing for an FPGA

- FPGAs have two software trains:
  - FPGA Circuit Design
    - Circuit (Hardware) Description Language that results in the configuration appropriate for the system software.
    - This skill set is mostly electrical engineering, with software practices and circuit practices
  - System Software Design
    - i.e. the software that will run using FPGA Components, CPU components, and ASICs combined in the system.
    - These components need to follow the organizational software process.
- Remaining components follow the software engineering process for the organization.

High level process to use an FPGA



HDLs: VHDL vs Verilog

VHDL

```
-- (this is a VHDL comment)
-- import std_logic from the IEEE library
library IEEE;
use IEEE.std_logic_1164.all;
entity ANDGATE is
  port (
    A : in std_logic;
    B : in std_logic;
    Q  : out std_logic);
end entity ANDGATE;
architecture RTL of ANDGATE is
begin
  Q <= A and B;
end architecture RTL;
```

Inputs		Outputs
a	**b**	**q**
1	**0**	**0**
0	**1**	**0**
1	**1**	**1**

Verilog

```
module andgate (a, b, q);
input a, b;
output q;
assign q = a & b;
endmodule
```

https://en.wikipedia.org/wiki/VHDL

https://en.wikipedia.org/wiki/Verilog

Quartus

- Quartus (Prime) is an Intel package that allows graphical design for FPGAs, runs on LINUX and Windows.
  - Made by Intel, 3 Versions: Intel® Quartus® Prime Lite Edition (free), Intel® Quartus® Prime Standard Edition (paid), Intel® Quartus® Prime Pro Edition (paid, most expensive version).
  - Lite version supports most early versions of FPGAs, but is very limited in analysis and how it can push to the FPGA board.
    - Alterra MAX10 is supported in Lite
      - Need at least 1GB RAM, 50GB Drive space….recommend some sporty graphics processors!
  - The pay versions have most modern FPGAs, and supports troubleshooting and push functions.
  - There is a very good course in Coursera…if you want to use Quartus do the trial course…

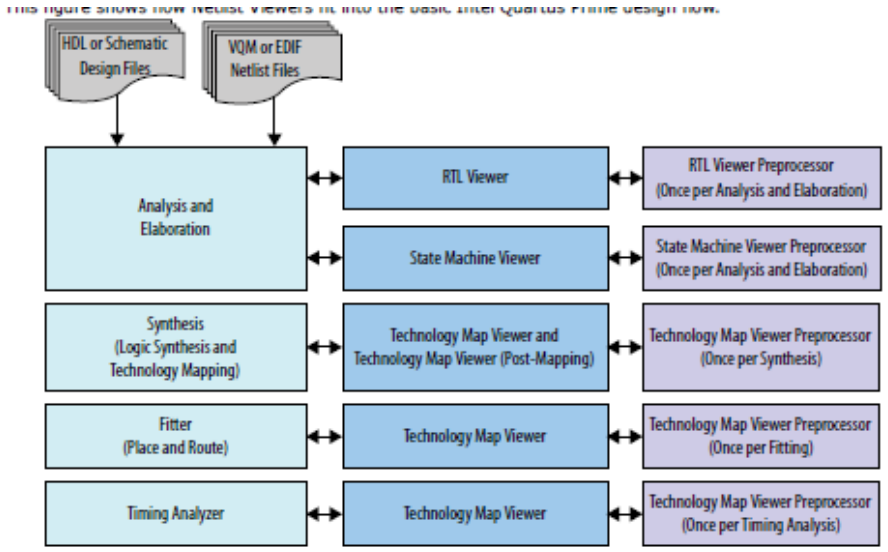
https://fpgasoftware.intel.com/?edition=lite https://fpgasoftware.intel.com/requirements/19.1/

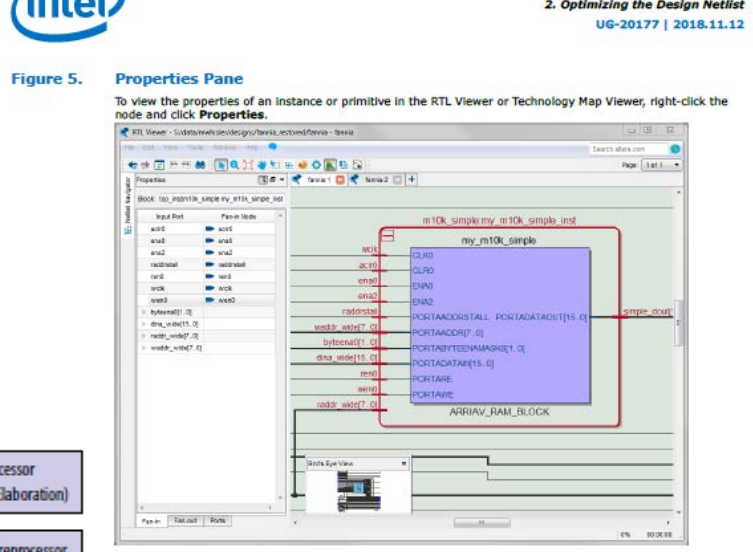
https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html

Ref: Intel® Quartus® Prime Standard Edition User Guide Design Optimization
Updated for Intel® Quartus® Prime Design Suite: **18.1 UG-20177 | 2018.11.12**

Using Quartus

- Step 1: Make sure your FPGA board and version is supported.
  - Download Module if needed
  - Make sure it works!!!!
- Step 2: Graphically Design Circuit
  - Define details like timing, truth tables, etc. in various windows
  - Save as a project.
- Step 3: Run Test and Compile
  - Pick hardware to run against and compile for.
  - Set parameters particular to hardware
- Step 4: Test Timing against simulated FPGA
- Step 5: Optimize and retest
- Step 6: Produce HDL
- Step 7: Push to your board
- Step 8: Test in real world
- Step 9: Fix is needed





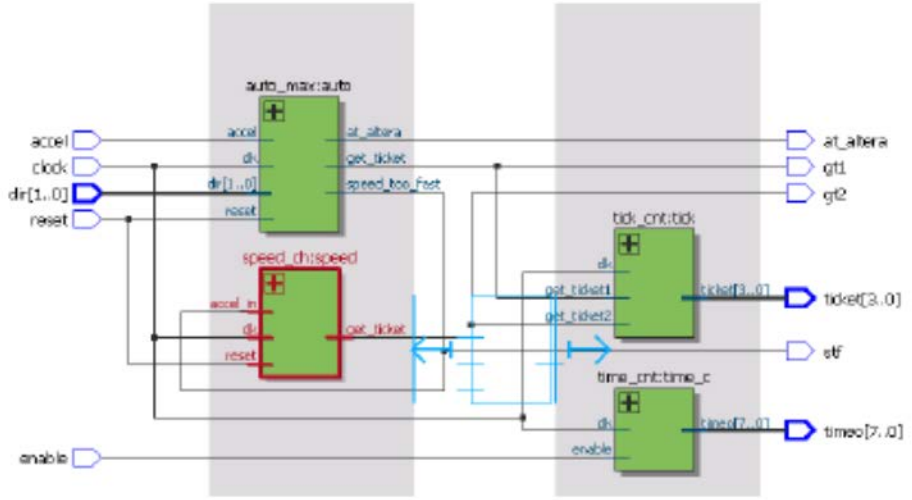
https://www.intel.com/content/www/us/en/programmable/documentation/yoq1529444104707.html

https://www.intel.com/content/www/us/en/programmable/products/design-software/fpga-design/quartus-prime/user-guides.html

Schematic View

- Works a lot like Visio or other CAD packages in Schematic View
  - Chip Diagram is on screen, drag and drop parts like: gates, tables, registers, timers, accumulators etc.
  - Use connectors to wire up inputs and outputs

Figure 10. Legal Placement when Moving Nodes



To restore the schematic view to its default arrangement, right-click and click **Refresh**.

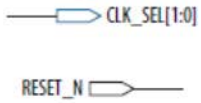
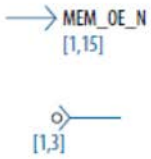
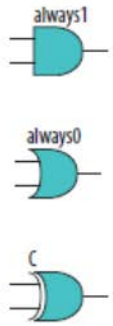
2.7.2. Schematic Symbols

The symbols for nodes in the schematic represent elements of your design netlist. These elements include input and output ports, registers, logic gates, Intel primitives, high-level operators, and hierarchical instances.

Note: The logic gates and operator primitives appear only in the RTL Viewer. Logic in the Technology Map Viewer is represented by atom primitives, such as registers and LCELLs.

Table 5. Symbols in the Schematic View

This table lists and describes the primitives and basic symbols that you can display in the schematic view of the RTL Viewer and Technology Map Viewer.

Symbol	Description
I/O Ports 	An input, output, or bidirectional port in the current level of hierarchy. A device input, output, or bidirectional pin when viewing the top-level hierarchy. The symbol can also represent a bus. Only one wire is shown connected to the bidirectional symbol, representing the input and output paths. Input symbols appear on the left-most side of the schematic. Output and bidirectional symbols appear on the right-most side of the schematic.
I/O Connectors 	An input or output connector, representing a net that comes from another page of the same hierarchy. To go to the page that contains the source or the destination, double-click the connector to jump to the appropriate page.
OR, AND, XOR Gates 	An OR, AND, or XOR gate primitive (the number of ports can vary). A small circle (bubble symbol) on an input or output port indicates the port is inverted.
MULTIPLEXER	A multiplexer primitive with a selector port that selects between port 0 and port 1. A multiplexer with more than two inputs is displayed as an operator.

continued...

Ref. Intel® Quartus® Prime Standard Edition User Guide Design Optimization
Updated for Intel® Quartus® Prime Design Suite: **18.1 UG-20177 | 2018.11.12**

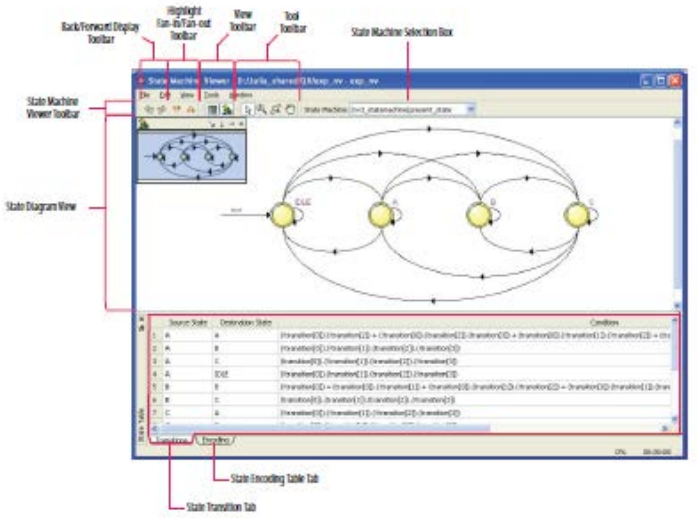
Ref: Intel® Quartus® Prime Standard Edition User Guide Design Optimization
Updated for Intel® Quartus® Prime Design Suite: **18.1 UG-20177 | 2018.11.12**

Other Views

- State Machine examines network and timing, truth tables, and overall math model
- Brid's Eye view get you the whole Schematic
  - Chip View show schematic for you intended target if you loaded the library
- RTL (Register Transfer Level) Viewer in Netlist Viewer: Shows how the model looks logically for variables (registers) and how they are connected in language

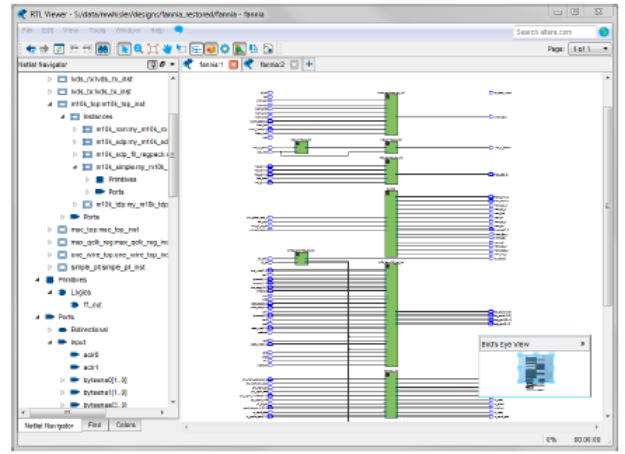
Figure 11. The State Machine Viewer

The following figure shows an example of the State Machine Viewer for a simple state machine and lists the components of the viewer.



of the design.

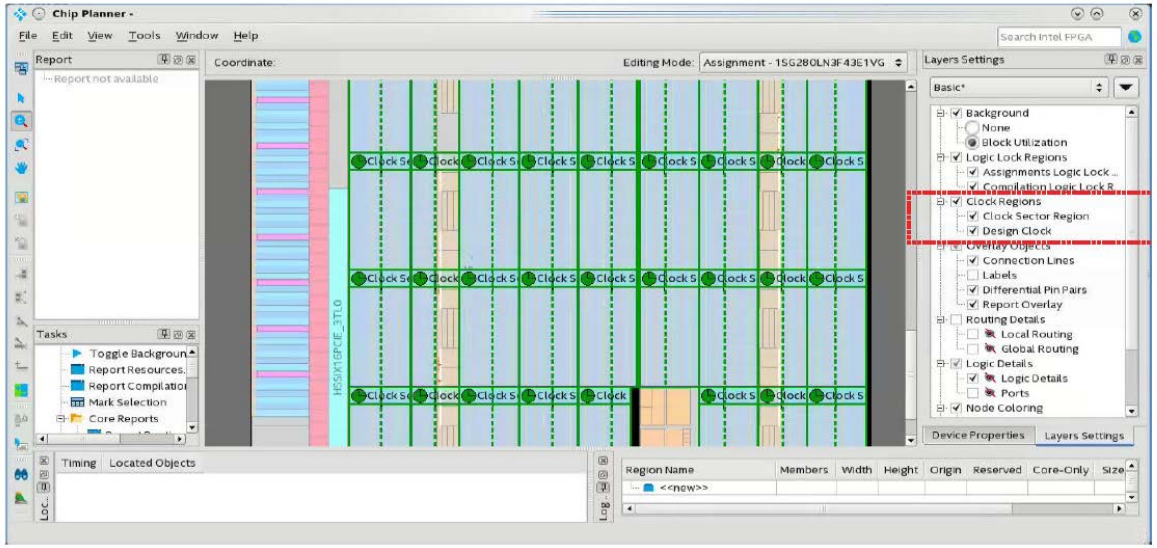
Figure 3. RTL Viewer



Timing and optimization

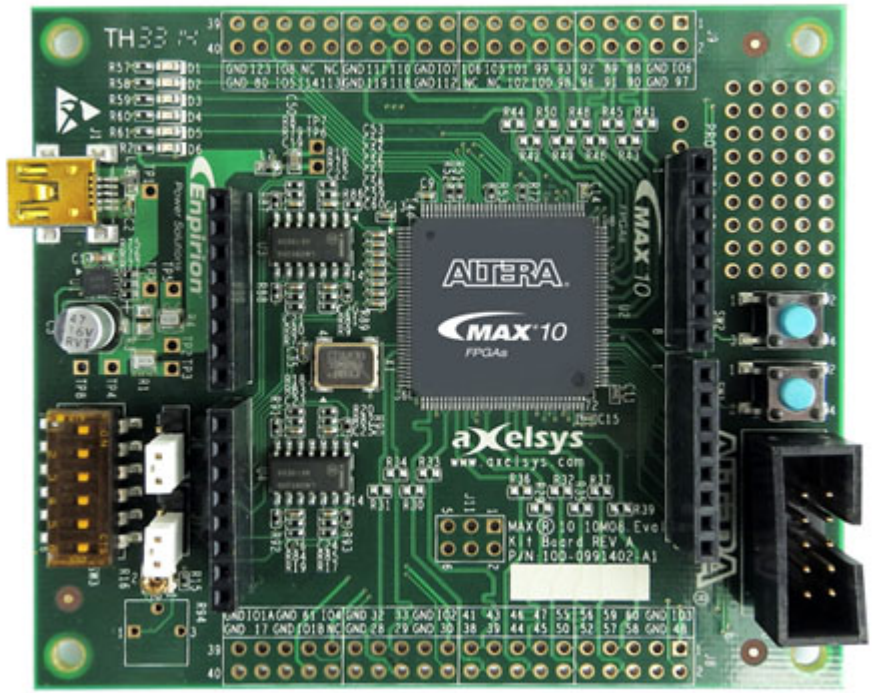
- Several timing tools are in Quartus, including critical path time calculators…i.e. where are you wasting latency..
- Requires the Chip Planner to be up for your chip.
- Also looks at resource utilization and other criteria

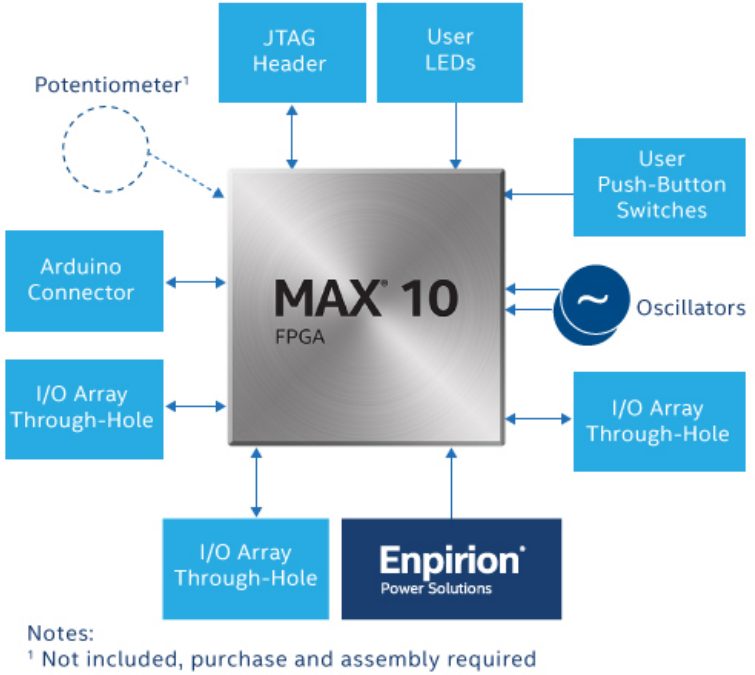
Figure 34. Clock Regions



Alterra MAX10

- Inexpensive FPGA board for learning FPGAs



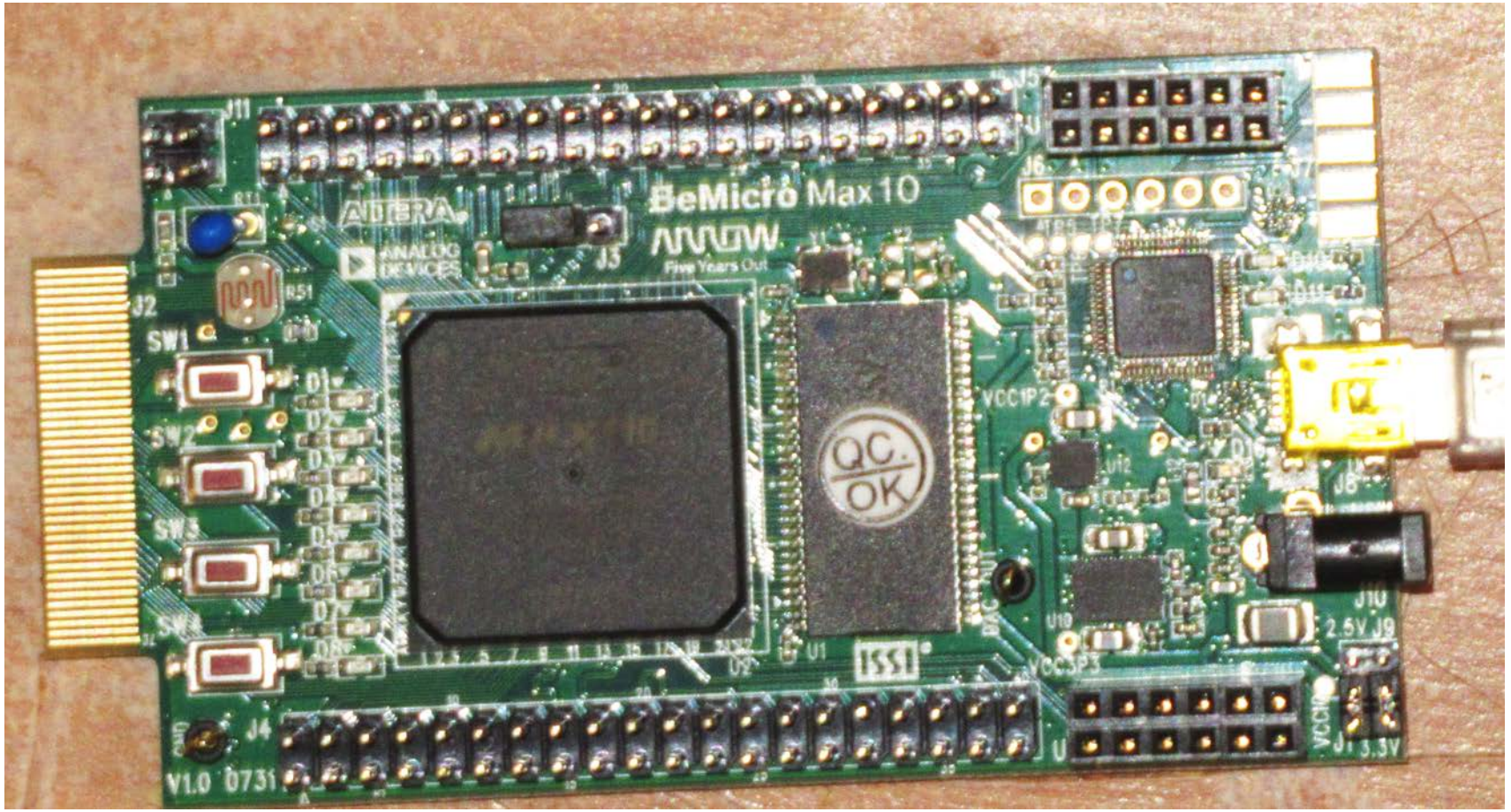


https://www.intel.com/content/dam/altera-www/global/en_US/support/boards-kits/max10_dk_schematic_revB_pcb.pdf

https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/kit-max-10-evaluation.html

https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/max-10-fpga-development-kit.html

BeMicroMAX10



USB powered and interfaced
Can be plugged into other boards, extended using pinouts
~$30.00

https://www.arrow.com/en/products/bemicromax10/arrow-development-tools

Synopsis and Links

This preso will cover the basics of what are Field Programmable Gate Arrays (FPGAs), past and current, when to use them versus ASICs (Application Specific Circuits) and CPUs/GPUs, and a show and tell on the Alterra BeMicro10 and a little on Quartus and HDLs (VHDL, Verilog) to program them.

An FPGA is a series of circuits that can be configured using a description language, then used as if they were burned for that use (as ASICs are), in lieu of CPUs which are full burned in but need software to run. FPGAs are often used to prototype circuitry before burning ASICs, used to test timing in real-time systems, or make one time use circuits, or used in systems that are configured, then 'forget' themselves if captured or lost (as in battlefield systems).

A few references (more in slides).

https://en.wikipedia.org/wiki/Field-programmable_gate_array

https://en.wikipedia.org/wiki/Application-specific_integrated_circuit

https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html

https://en.wikipedia.org/wiki/VHDL

https://en.wikipedia.org/wiki/Verilog

https://numato.com/blog/differences-between-fpga-and-asics/

https://www.arrow.com/en/products/bemicromax10/arrow-development-tools#page-1