ECE 551

Digital Design And Synthesis

Fall '21

Digital Circuit Implementations:
- ✓Standard Cell
- ✓FPGA
- ✓Custom Logic

1

Functional & Synthesized…Now What?

- After synthesis, implement as hardware
  - FPGAs
  - Standard cells
  - Custom logic
- Choose implementation based on cost and performance requirements

2

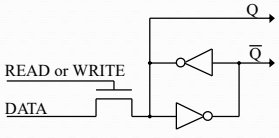
FPGAs

- Field Programmable Gate Array
  - Temporary (SRAM based)
  - Permanent (Flash) not as common
- Pros
  - Allow for very complex implementations
  - Generally reuseable (upgrades/bugfixes/prototype)
  - Low non-recurring engineering costs (NREs)
- Cons
  - Expensive per-unit (10s-100s of $)
  - Slower than gates
  - Need support circuits (configuration loading)
  - Higher power consumption than ASIC.
  - Not as robust (mission critical operations)

3

Programming an FPGA

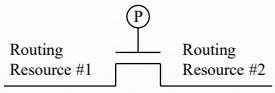
- Most designs based on SRAM
  - Writing to the SRAM "configures" device
  - Different circuits implemented based on values



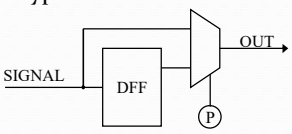
4

Configurable Routing Elements

- Programmable connection



- Programmable bypass

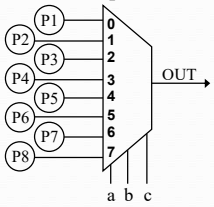


5

5

Logic Elements

- Look-Up Table (LUT)
  - Essentially a very small memory
  - Most common size is 4-input LUT (shown is 3-input)



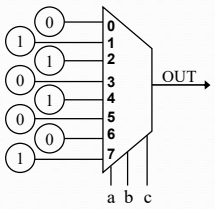
6

6

Logic Elements

- Look-Up Table (LUT)
  - OUT = a XOR b XOR c

abc	OUT
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	1



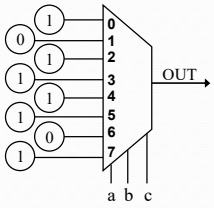
7

7

Logic Elements

- Look-Up Table (LUT)
  - OUT = $\bar{a}\bar{b} + ac + \bar{b}\bar{c}$

abc	OUT
000	1
001	0
010	1
011	1
100	1
101	1
110	0
111	1

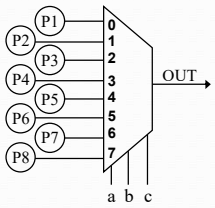


8

8

Logic Elements

- Look-Up Table (LUT)
  - Extremely flexible in implementing logic
  - Larger and slower than just using gates

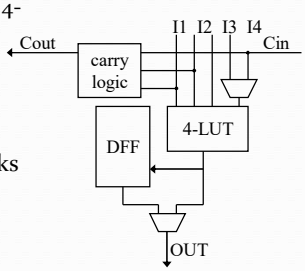


9

9

FPGA Logic Structure

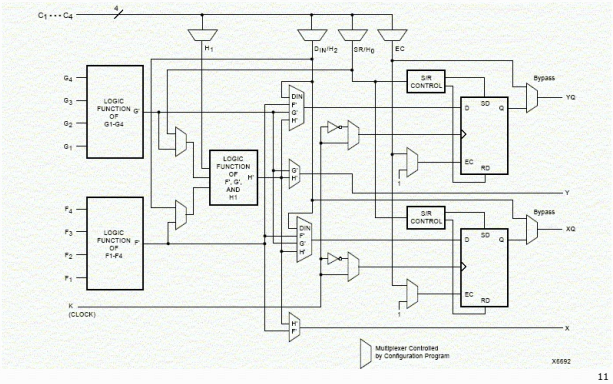
- "Cell" or "logic block":
  - 1 or more LUTs (generally 4-input)
  - At least one D flip-flop
  - Possibly fast carry logic
- Connect several logic blocks to form circuit



10

10

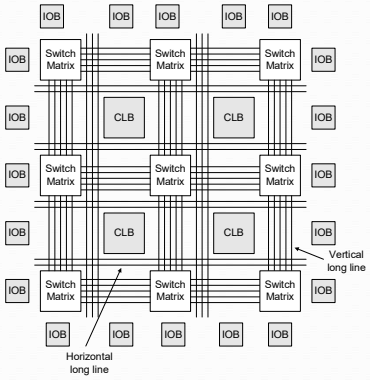
Xilinx 4000 Combinational Logic Block



11

11

Xilinx 4000 FPGA



Really much more than 4 CLB's and 9 configurable routing channels but this gives an idea of the structure

12

12

FPGAs

- Allow for complex implementations
- Generally reuseable (upgrades/bugfixes/prototype)
- Low non-recurring engineering costs (NREs)

- Expensive per-unit (10s-100s of $)
- Slower than gates (Programming points)

- Newer FPGAs often incorporate memories, multipliers, peripherals, and even processors all on the same chip

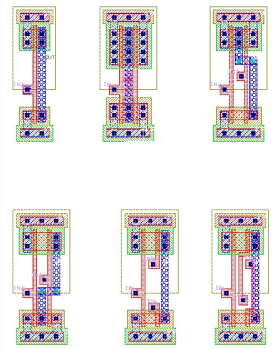
13

13

Standard Cells

- Gates and other small structures
- Can also use macroblocks
  - Groups of pre-optimized cells
  - Larger custom-layout structures
  - More efficient when possible

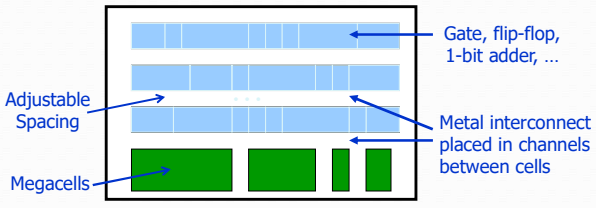
From: http://www.zuraleff.com/layout



14

14

Standard Cell Layouts



15

15

Standard Cells

- Cheap per-unit pricing ($0.50-$10s)
- Fast compared to FPGAs

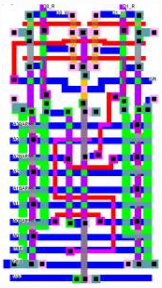
- High NREs (design time, mask fabrication…)
  - $150K-$10M
  - Cost of a respin is very high in time & $$
- Only makes sense for
  - Large quantities and/or
  - Critical operations

16

16

Custom Logic

- Manual layout
- Very high NREs
  - Huge design time!
  - Design changes painful.
- Human is still better than the CAD tool at tight layout.
- There exists a compromise between standard cell APR and full custom. These datapath synthesis tools do APR on a bitslice basis exploiting symmetry in the netlist.



17

17

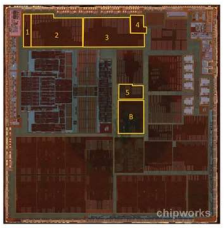
Apple A6 Utilizes Full Custom Design

Quote:

Die photographs of the A6 and A6X that were first published by Chipworks, with our added annotations, are reproduced below. Probably the single most striking feature of both is the in-house designed CPU. Going further, it was a custom design where the layout was performed manually, instead of using the common automated "place and route" approach. In their discussion of the A6 ChipWorks commented: "This is a more expensive and time-consuming method of layout. However it usually results in a faster maximum clock rate, and sometimes results in higher density." They go on to say: "In fact, with the exception of Intel CPUs, it's one of the first custom laid out digital cores we've seen in years!"

The fact that Intel and AMD would be the only other full custom design entities out there should show you how expensive and resource intensive a full custom layout is.

This commitment to true custom design shows Apple's devotion to the ARM ISA and platform long term. It also shows their unwillingness to compromise on "good enough" performance and to always be the best they can be, even if the basis for comparison across platforms doesn't always exist in an easily interpretable way due to ecosystem and UI differences.



A6 Die Photo with Digital Block References

18

18

Hardware Implementations

- There is no one "best" method
- Tradeoffs between cost and speed
  - Design complexity matters, too
- Standard cells are getting more expensive…
- FPGAs are getting faster and bigger…
  - This will affect future design choices
  - For a given design, current best choice may not == future best choice!

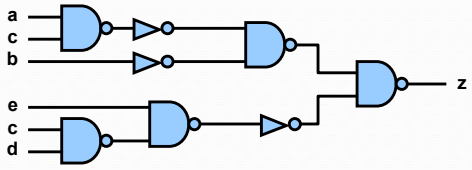
Another choice to implement your digital design might be a programmable µController with off the shelf support hardware

19

19

Tech Mapping: Std. Cells

- Example boolean equation:
  - $z = a\,\bar{b}\,c + c\,d + \bar{e}$
- Example cell library:
  - 2-input NAND, INV
- Resulting Tech Mapped Circuit:



Been there…Done that…

This is what we are doing with our designs when synthesizing to the 35nm tsmc library

20

20

Tech Mapping: FPGAs

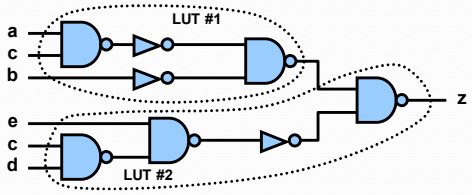
- Need to know building blocks of the FPGA
  - LUT size (if uses LUTs)
  - Any special resources (Multipliers, RAMs)
- Tech mapping then implements your netlist in terms of those building blocks

21

21

Tech Mapping: FPGAs

- Example boolean equation:
  - $z = a\,\bar{b}\,c + c\,d + \bar{e}$
- Example basic block:
  - 4-input LUT
- Resulting tech-mapped circuit:

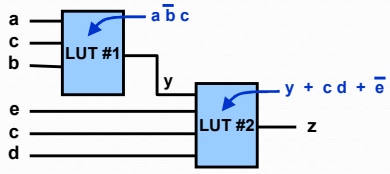


22

22

Tech Mapping: FPGAs

- Example boolean equation:
  - $z = a\,\bar{b}\,c + c\,d + \bar{e}$
- Example basic block:
  - 4-input LUT
- Resulting tech-mapped circuit:



23

23

Placement

- Standard Cells:
  - Choosing a row for each cell
  - Choosing a location within the row for each cell
  - Cost function based
  - Minimize ratsnest (interconnection)
  - CTS will have priority in placing clock buffers
- FPGAs:
  - Choosing which physical LUTs implement each netlist LUTs

24

24

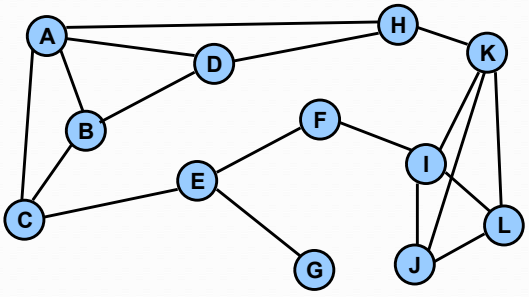
Partitioning & Floorplanning (can be common to both StdCell and FPGA)

- Sometimes you have BIG circuits
  - Makes placement take a long time
  - Yields poor results (too large a solution space)
- Use partitioning and floorplanning
  - Partitioning: Divide netlist into partitions
  - Floorplanning: Assign partitions to chip regions
  - Place regions separately
  - Benefit: Small problems are easier to solve well than large ones

25 25

25

Partitioning Example

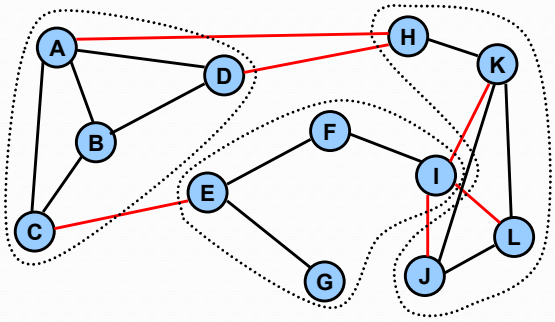


How might we choose to form 3 partitions?

26

26

Partitioning Example - Bad



27

27

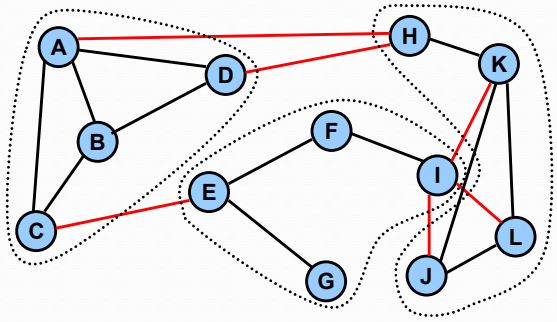
Partitioning

- We want to try to make our partitions as independent as possible.
- Independent = fewer outside connections
- Why?
  - Want to keep wires short
  - Try to place partitions adjacent to the partitions they interconnect with
  - If we have a lot of interconnections, this may not be easy/possible

28

28

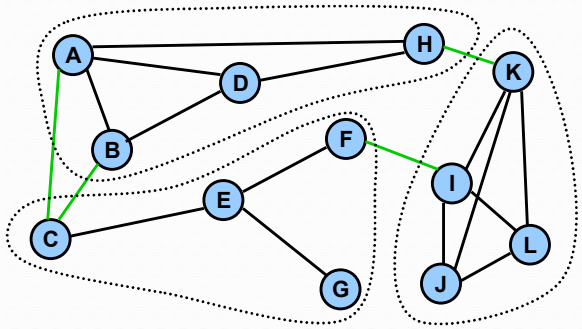
Partitioning Example - Bad



29

29

Partitioning Example - Better



30

30

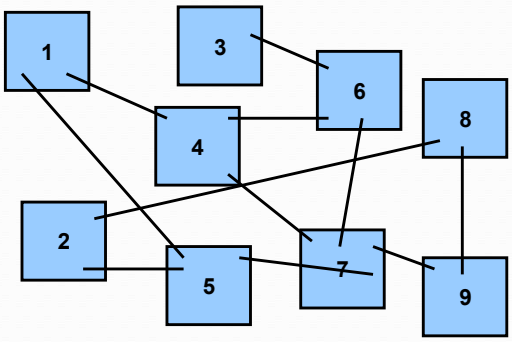
Floorplanning

- OK, so we've divided our problem up into partitions
- Now, figure out where partitions should be placed relative to one another
- Assign partitions to regions of the silicon / FPGA
- Try to avoid long wires between partitions
- Don't want to have to route wires through too many other partitions
  - Wastes area in those partitions

31

31

Floorplanning Example

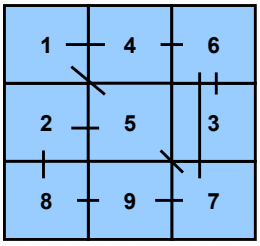


32

32

Floorplanning Example

- Try to arrange partitions to minimize cross-partition routing



Eat your heart out, Sudoku.

33

33

Routing

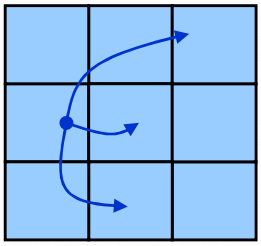
- Have locations for all the cells/LUTs in the netlist
- Now need to connect them together to actually make the circuit
- Different techniques for std. cell vs. FPGA
- Divided into:
  - Global
  - Detailed (local)

34

34

Global Routing

- Find a rough path for each net
- Figure out what areas a signal passes through

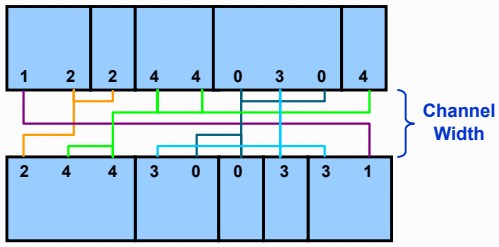


35

35

Detailed Routing: Std. Cells

- Connect the cells within the global regions
- Goal: minimize channel width



36

36

Detailed Routing: FPGAs

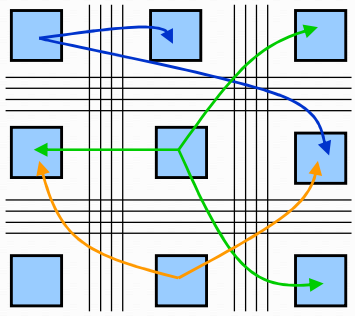
- Assign signals in netlist to:
  - Wires
  - Switchbox points
- Fixed set of available resources
  - Can't "widen" routing channels
- Goal: Reduce *congestion*
  - Congestion is the ratio of signals:wires
  - By keeping areas "open", more likely to be able to route later signals

37

37

Detailed Routing: FPGAs

- Goal: Reduce *congestion*

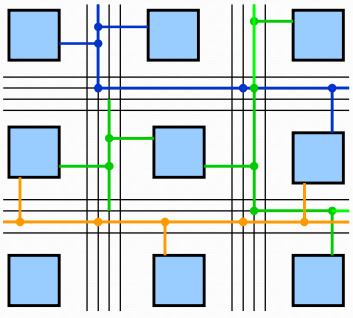


38

38

Detailed Routing: FPGAs

- Goal: Reduce *congestion*



39

39

Detailed Routing: FPGAs

- Frequently start with an "idealized" routing
  - Signals can share wires
- Repeatedly "rip up" and reroute
  - One or more nets (signals)
- Stop when no wires are shared

40

40

Conclusion

- Synthesis isn't the end of the process!
  - Many steps after it
- Choose target implementation
  - Examine cost/performance tradeoffs
- Use CAD tools to implement synthesized circuit on FPGA or std. cells
  - Place & Route
  - Generate bitstream or layout masks
- See **ECE556** for more details on CAD algorithms

41