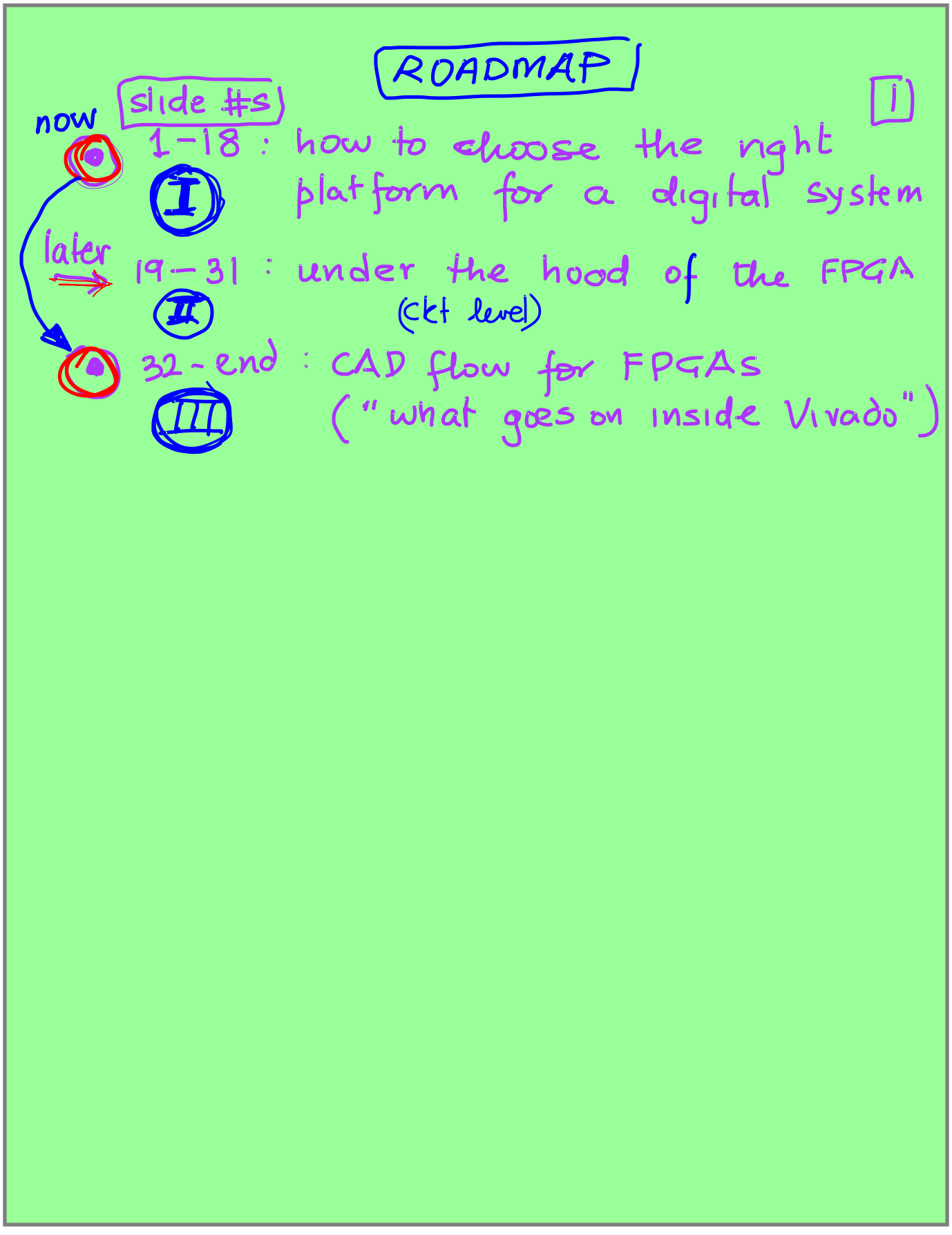
ECEN 449 – Microprocessor System Design

FPGAs and Reconfigurable Computing

Some of the notes for this course were developed using the course notes for ECE 412 from the University of Illinois, Urbana-Champaign


ROADMAP
1
now
slide #s
1-18 : how to choose the right platform for a digital system
I
later
19-31 : under the hood of the FPGA (ckt level)
II
32-end : CAD flow for FPGAs ("what goes on inside Vivado")
III

1

- ✓ FPGA = Field Programmable Gate Array
- ✓ ASIC = Application Specific Integrated Ckt (automated design process)
- ✓ Custom IC = more manual than ASIC
- ✓ NRE = non-recurring engineering expense

(1)

Which platform to use?

Criteria	Software	FPGA	ASIC	Custom IC
Delay	100+	10-15	2	1
time to market / design	1	2	10	30
power	100	10	1.5	1
area	1000+	10	1.5	1
design team size LB	1	2	10	50
cost = $\alpha\beta + \gamma$	2	9	120/200	1600/2500
NRE LB	1	5	20/100	100/1000
ease of upgrade ability	1000	100	1	1
recurring cost	1	2	10	50

fabless
- AMD
- Nvidia
- Apple

fab-based
- intel
- samsung

Objectives of this Lecture Unit

- Get a feel for the different technologies that can be used to implement a design (digital system)
  - Flavors of hardware technologies
  - Flavors of implementation methods
- Understand the basics of how FPGAs work
  - So that the CAD tools in the lab make sense to you

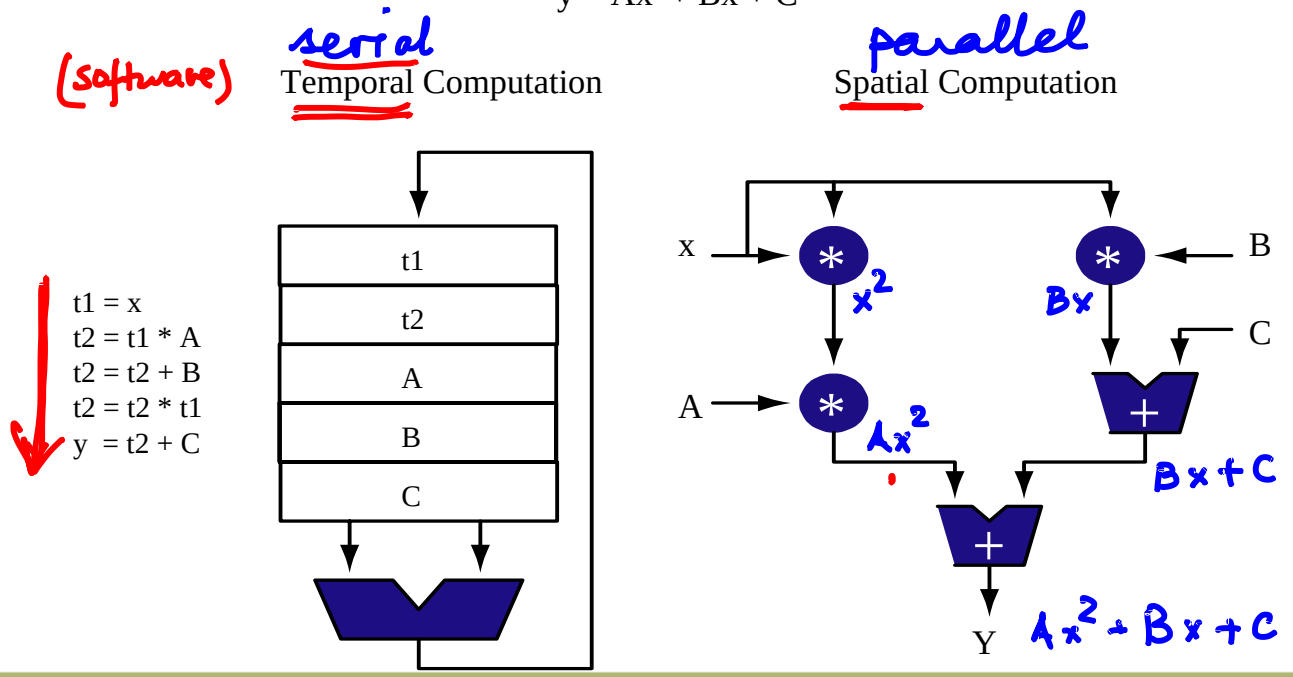
Software, Custom Hardware or Reconfigurable Hardware?

- When should we use software, “custom” hardware, or reconfigurable hardware?
- Software based systems are easiest to implement
  - But there is a huge performance gap between software and hand-designed (custom) hardware systems
  - Often 100-to-1 ratio of performance (speed) or performance/area
- But custom hardware systems not so good for general computing
  - Big design effort (time, cost) are barriers to implementation
  - Not practical to buy a new machine every time you want to run a different program
- Reconfigurable systems offer best-of-both-worlds
  - Run-time programmability (in the field)
  - Hardware-level performance (although lower than custom hardware)
  - FPGAs and CPLDs are the vehicles for reconfigurable systems.

Why is Hardware Faster Than Software?

- Spatial vs. Temporal Computation
  - Processors divide computation across time, dedicated hardware divides across space
  - But dedicated hardware is hardwired for a specific task.

$y = Ax^2 + Bx + C$



Why is Hardware Faster Than Software?

- Specialization:
  - Instruction set may not provide the operations your program needs
  - Processors provide hardware that may not be useful in every program or in every cycle of a given program
    - Multipliers
    - Dividers
- Instruction Memory
  - Processors need lots of memory to hold the instructions that make up a program and to hold intermediate results.
- Bit Width Mismatches
  - In general, processors have a fixed bit width, and all computations are performed on that many bits
    - Multimedia vector instructions (MMX) a response to this

So why not just use Hardware?

- Dedicated hardware is
  - Dedicated (not flexible)
  - Takes long to design and develop (typical processor takes a handful of years to design, with design teams of a few hundred engineers)
  - This is expensive!
  - Only way to justify such an effort is if the customer demand guarantees high volume sales
- So there is a strong need for a design approach which has performance comparable to dedicated hardware, with ease-of-programmability comparable to software.
- Answer? Reconfigurable computing (FPGAs, CPLDs and their cousins)

PLA

complex prog logic devices

Good Applications for Reconfigurable Computing

- Data Parallelism
  - Execute same computations on many independent data elements
  - Pipeline computations through the hardware
- Small and/or varying bit widths
  - Take advantage of the ability to customize the size of operators
- Low-volume applications which require rapid design turn-around time and hardware-like speeds
  - Several telecom, DSP (filters), radar, genomics (DNA sequence matching), processor emulation, neural network and similar applications.

Will FPGAs Defeat CPUs?

- Capacity: Instructions are very dense representation, logic blocks aren't
- Tools: Compilers for reconfigurable logic aren't very good
  - Some operations are hard to implement on FPGAs
  - C-for-FPGA technology is improving fast, though

One approach to capacity is to exploit the 90-10 rule of software

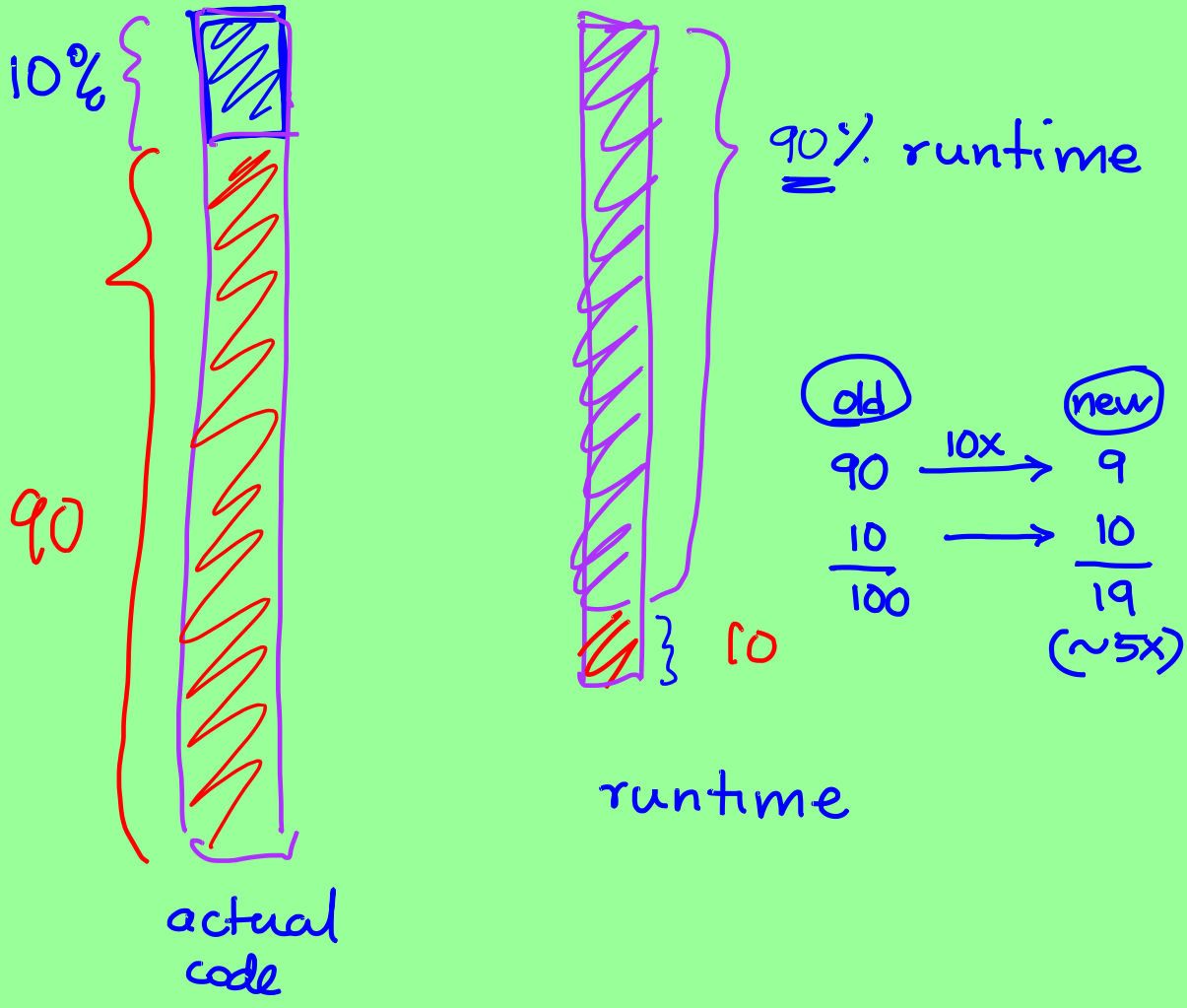
  - Run the 90% of code that takes 10% of execution time on a conventional processor
  - Run the 10% of code that takes 90% of execution time on reconfigurable logic

- But the temptation to merge the two worlds is real
  - Programmable-reconfigurable processors

8.1

90-10 rule

Digital system in software



8.2

Amdahl's law

Serial code (red) = S time units

Parallelizable code (blue) = P time units

Old runtime was S+P

Suppose we accelerate the parallelizable part by a factor F

New runtime = $S + \frac{P}{F}$

8.3

speedup is

$$\frac{S+P}{S+P/F} = \frac{\text{old runtime}}{\text{new runtime}}$$

for the 90-10 example (F = 10 assumed)

red blue

$$\frac{10+90}{10+90/10} = \frac{100}{19} \approx 5$$

red

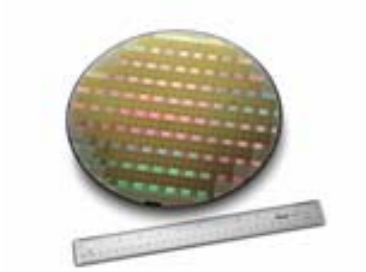
max speedup = $\frac{100}{10+90/\infty} \approx 10$ ✓

A Peek Under the Hood

- In the next few slides, we will peek under the hood of some of competing hardware based digital system design platforms
- We will cover
  - Application Specific ICs (ASICs).
    - Examples are IP routing ICs
    - SSI/MSI/LSI/VLSI
  - Reconfigurable (also sometimes called programmable) ICs.
    - Examples are FPGAs, CPLDs
  - Full custom Integrated Circuits (ICs).
    - Examples are processors, GPUs, network processors, DSP processors.

Application Specific Integrated Circuits

- Very high capacity today -- 10-100M transistors
- Very high speed – 500MHz+
  - Integration
  - Specificity
- Can use any design style below (or a hybrid)
  - Full Custom
  - Standard-cell (synthesized) – dominating methodology due to manufacturing considerations
- Long fabrication time
  - Weeks-months from completed design to product
- Only economical for high-volume parts
  - Making the masks required for fabrication is becoming very expensive, in the order of $1M per design

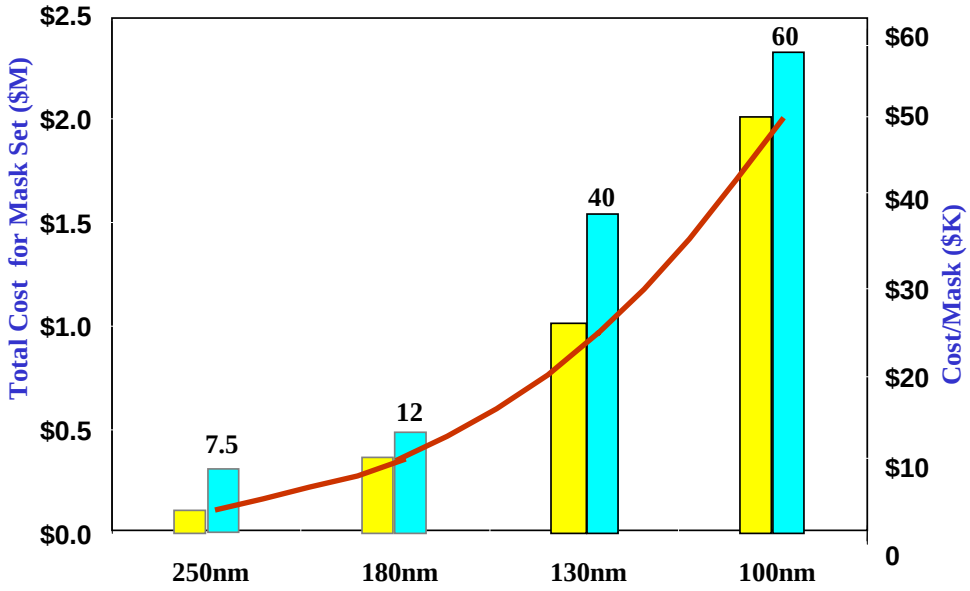


Deep Submicron Design Challenges

- This slide discusses why ASICs are becoming less popular in recent times (compared to reconfigurable ICs)
- Physical effects are increasingly significant
  - Parasitics, reliability issues, power management, process variation, etc.
- Design complexity is high
  - Multi-functionality integration
  - Design verification is a major limitation on time-to-market
- Cost of fabrication facilities and mask making has increased significantly

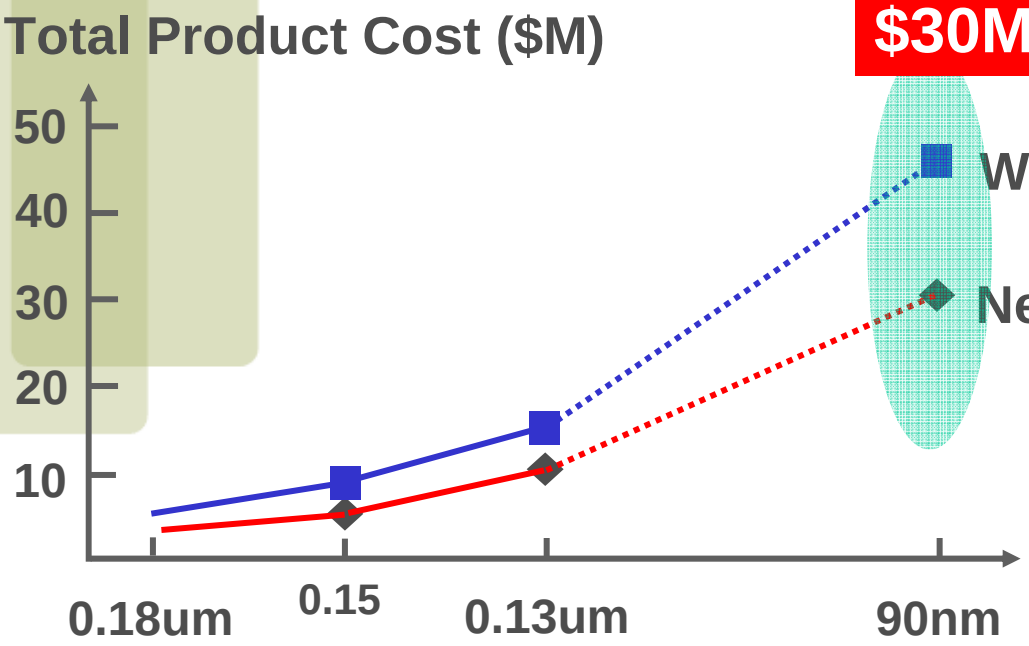
Rapid Increase in Manufacturing Cost

Process (um)	2.0	…	0.8	0.6	0.35	0.25	0.18	0.13	0.10
Single Mask cost ($K)	1.5		1.5	2.5	4.5	7.5	12	40	60
# of Masks	12		12	12	16	20	26	30	34
Mask Set cost ($K)	18		18	30	72	150	312	1,000	2,000



Source: EETimes

The Cost of Next Generation Product



$30M ~ $50M @ 90nm

Source:
IBS Inc.

Product Cost

- Engineering Cost – 60% up
- Manufacturing Cost – 40% up
- NRE/Mask Cost – 100% up
- Respin cost – 78% up

Programmable Logic Devices

- Early version: Mask-Programmable Gate Arrays
  - Build standard layout of transistors on chip
  - Customer specifies wiring to connect transistors into gates/system
  - Only has to go through last few mask steps of fabrication process, so faster than full chip fabrication
  - May become popular again in the near future
- Newer version: Programmable Logic Devices (PLD)
  - Use AND-OR array to implement arbitrary Boolean functions
  - Programmed by burning fuses that define connection from input wires to gates
  - Customer site programming allows rapid prototyping
  - Limited capacity, functionality
    - Generally have to be used in conjunction with other parts to hold state
    - Used to implement logic with moderate number of inputs (< 20)

PLA [248]

Programmable Logic Device Advantages

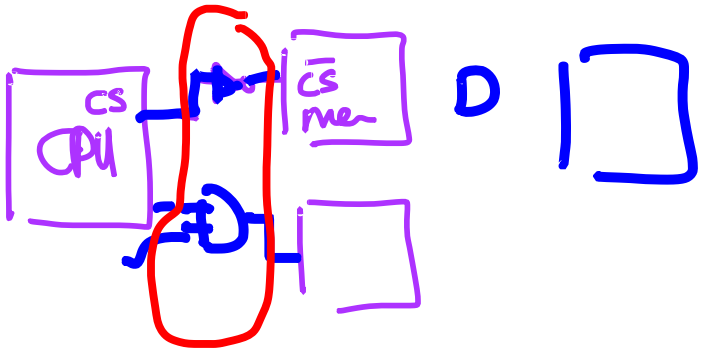
↳ Intel

↳ AMD

- Short TAT (total turnaround time)
- No or very low NRE (non-recurring engineering) costs.
- Field-reprogrammable
- Platform-based design

Today - Two Major Types of Programmable Logic

- CPLD (complex programmable logic device)
  - coarse-grained two-level AND-OR programmable logic arrays (PLAs)
  - fast and more predictable delay
  - simpler interconnect structures
- FPGA (field programmable gate array)
  - fine-grained logic cells
  - high logic density
  - good design flexibility (field programmable), easy redesign (just reprogram the chip!)
  - arguably more popular
- ***Increasing ASIC design costs are making FPGAs more popular. This technology is therefore important to learn about. Hence this course.***
  - ***Enables "garage" technology companies to thrive. This has a huge impact.***

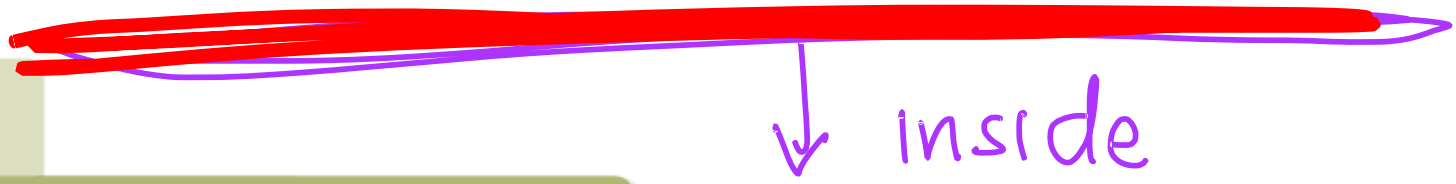


Evolution of the FPGA

- Early FPGAs used mainly for "glue logic" between other components
  - Simple CLBs, small number of inputs
  - Focus was on implementing "random" logic efficiently
- As capacities grew, other applications emerged
  - FPGAs as alternative to custom IC's for entire applications
  - Computing with FPGAs
- FPGAs have changed to meet new application demands
  - Carry chains, better support for multi-bit operations
  - Integrated memories, such as the block RAMs in the devices we'll use
  - Specialized units, such as multipliers, to implement functions that are slow/inefficient in CLBs
  - ***Newer devices incorporate entire CPUs: Xilinx Virtex II Pro has 1-4 Power PC CPUs (we will use such a device in our lab!!!)***
    - Devices that don't have CPU hardware generally support synthesized CPUs

Full Custom ICs

- These have captured an important niche in hardware implementation of systems
- Microprocessors, GPUs, network processors, DSP processors are key examples.
  - HIGH sale volume (required to justify huge development cost and time)
  - These are often the flagship products of many semiconductor companies (Intel, IBM, AMD, TI, Freescale, etc)
  - These designs are “custom” designed, to do a specific task extremely fast, with minimum area and power.

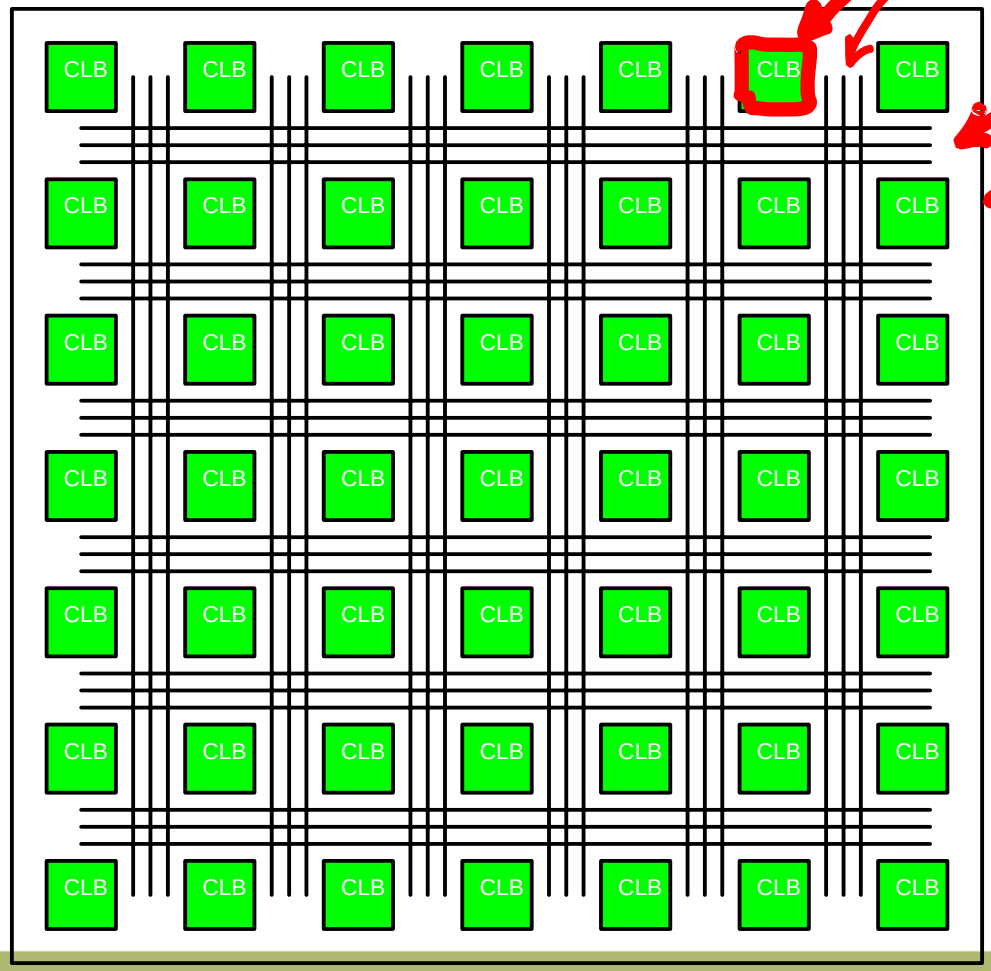


FPGAs in Detail

- Now in the next few slides, we will look at the technology that is inside an FPGA IC.
- This will allow us to understand how the FPGA works
- After this, we will be able to make sense of the design flow that is used to design a FPGA based circuit.

Field-Programmable Gate Arrays

- Based on Configurable Logic Blocks (CLB)

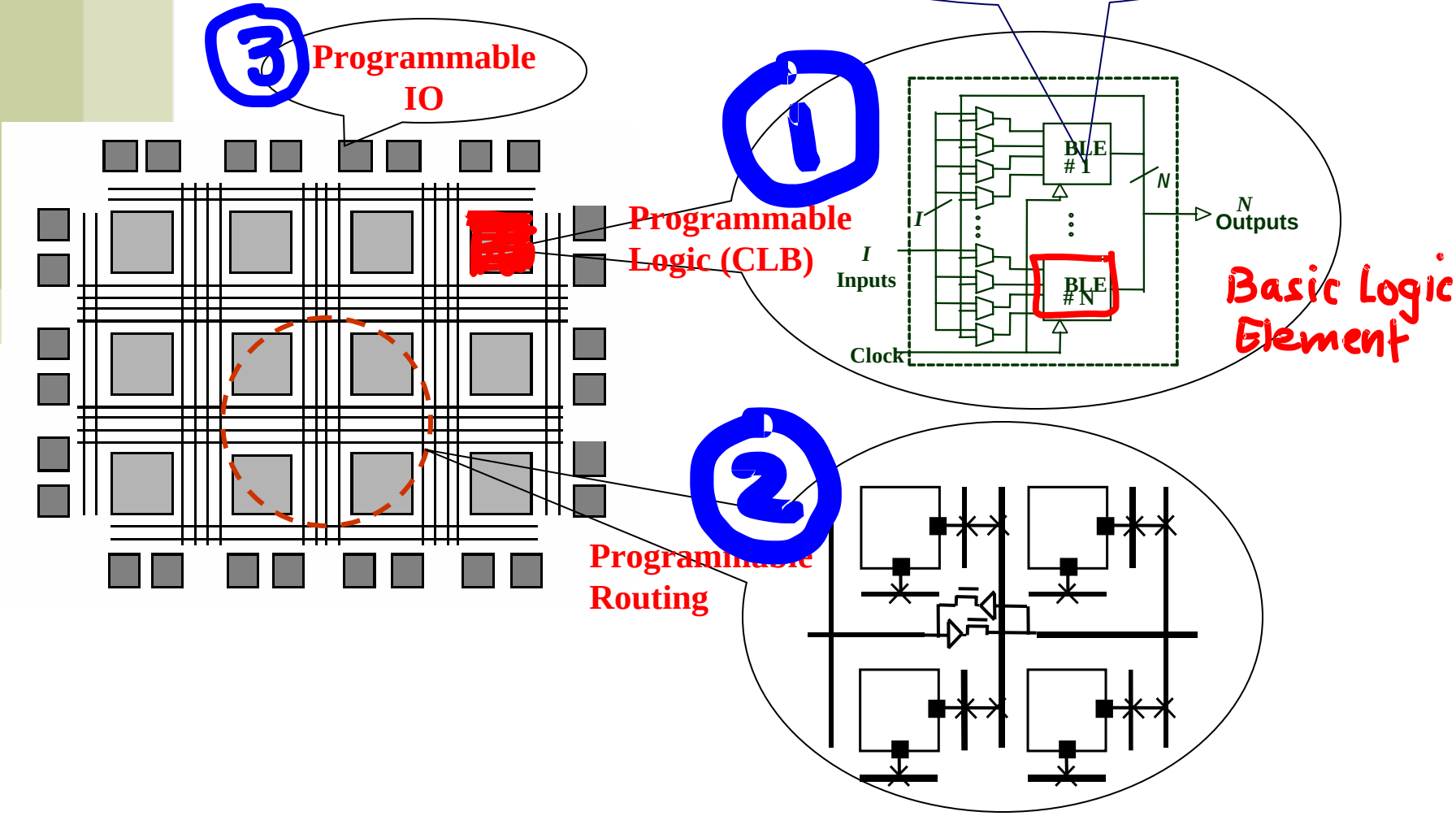


columns of uncommitted wires

rows of uncommitted wire

FPGA chip

A Generic FPGA Architecture

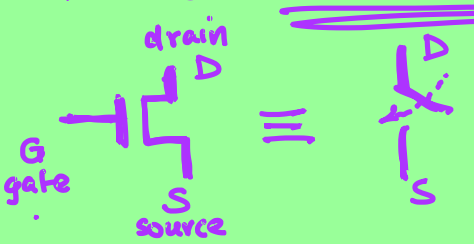


21.1

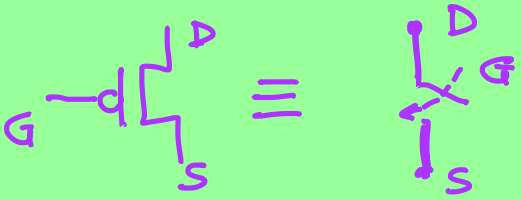
The circuits in an FPGA for programming are based on MOSFETs (Metal Oxide Semiconductor Field Effect Transistor)
aka transistors
aka MOS devices
aka FETs

There are 2 kinds of MOSFETs:
PMOSFETs
NMOSFETs

Both are switches (electrically)

(N) 

if V_G = VDD, the switch is closed (D & S shorted)
else
switch is open (D & S open ckt)

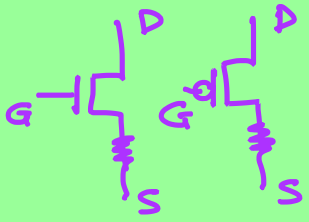
(P) 

if V_G = GND, the switch is closed
else
switch is open

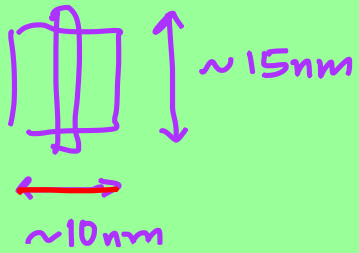
21.2

In reality

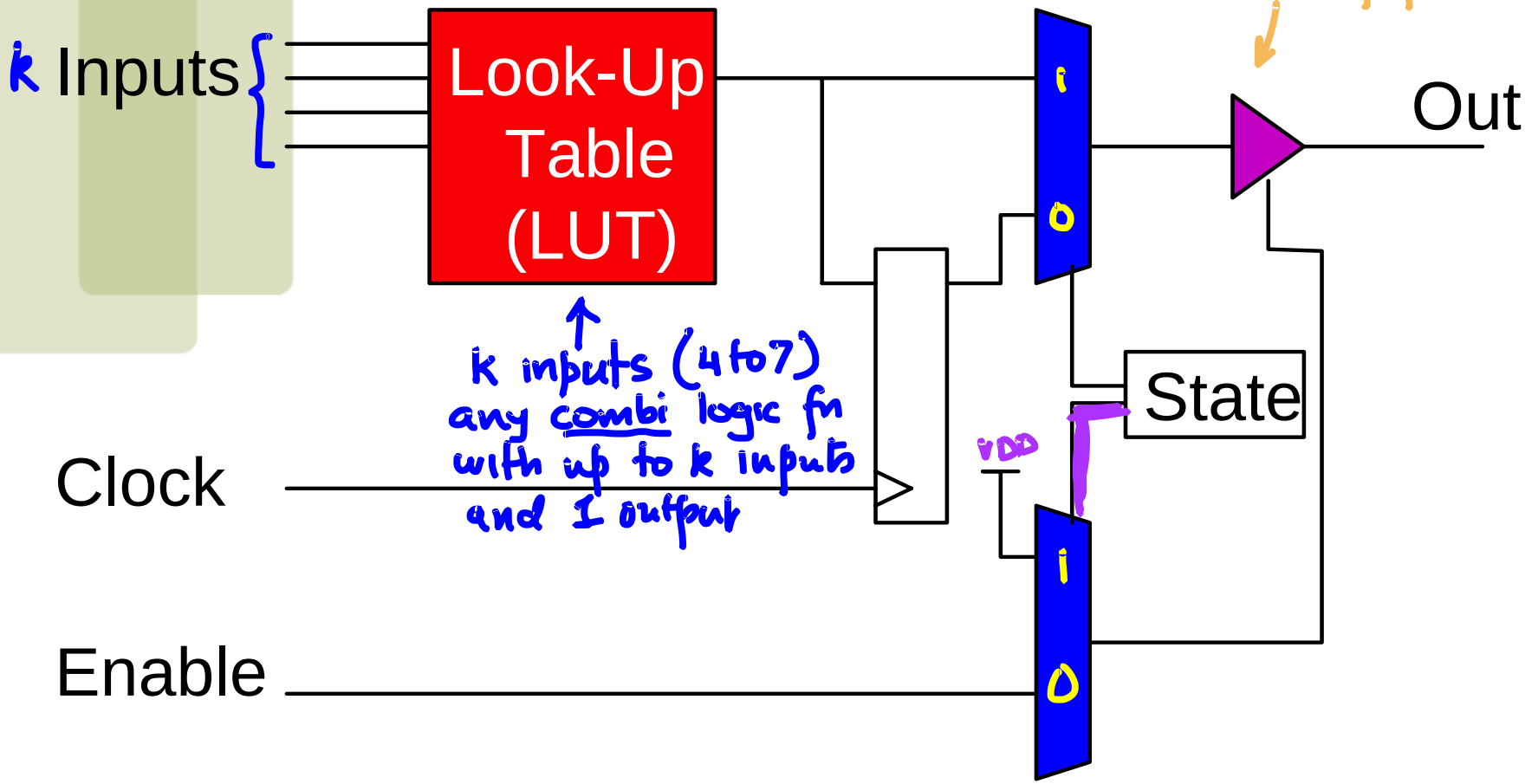
- PMOS & NMOS FETs have a series D-S resistance (not short circuited!)



- PMOS/NMOS device have on-resistance ~ 5KΩ and off resistance of >1GΩ
- They are tiny ("5nm technology" has P or N devices slightly larger than 5nm in each dimension



What's in a ~~CLB~~ BLE?



Xilinx CLB – a.k.a. “Slice”

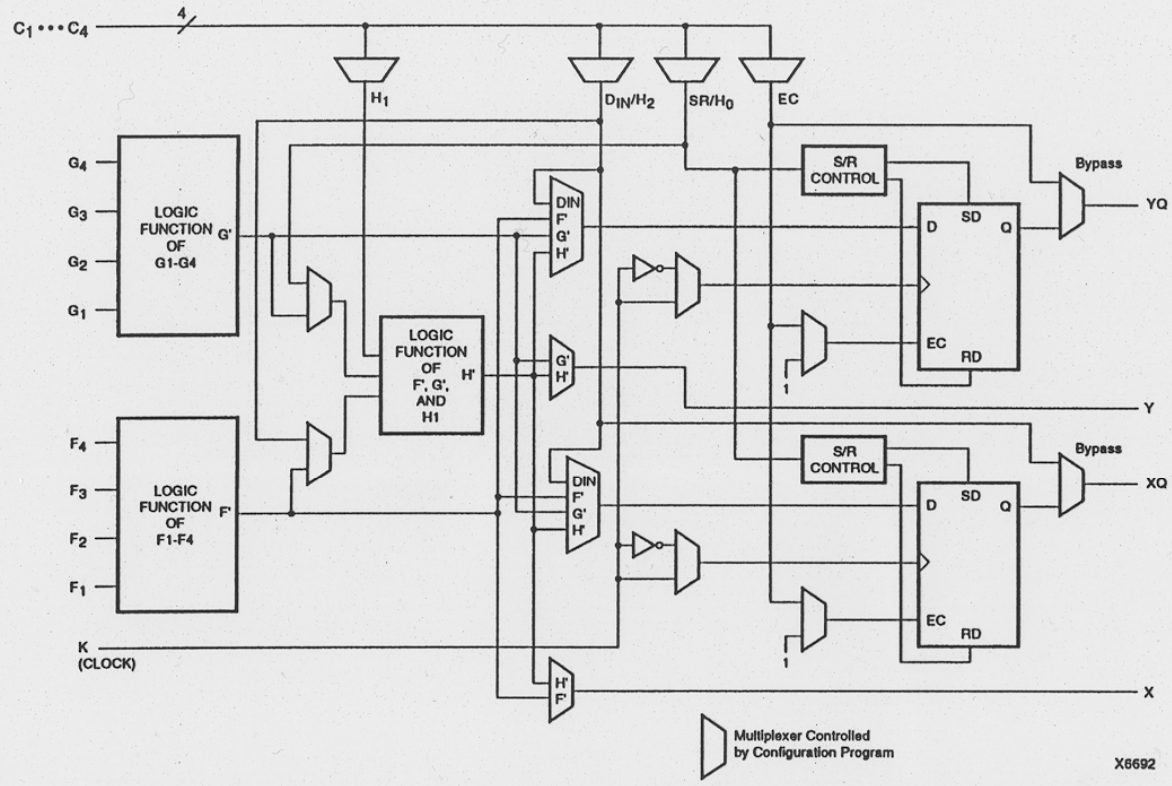
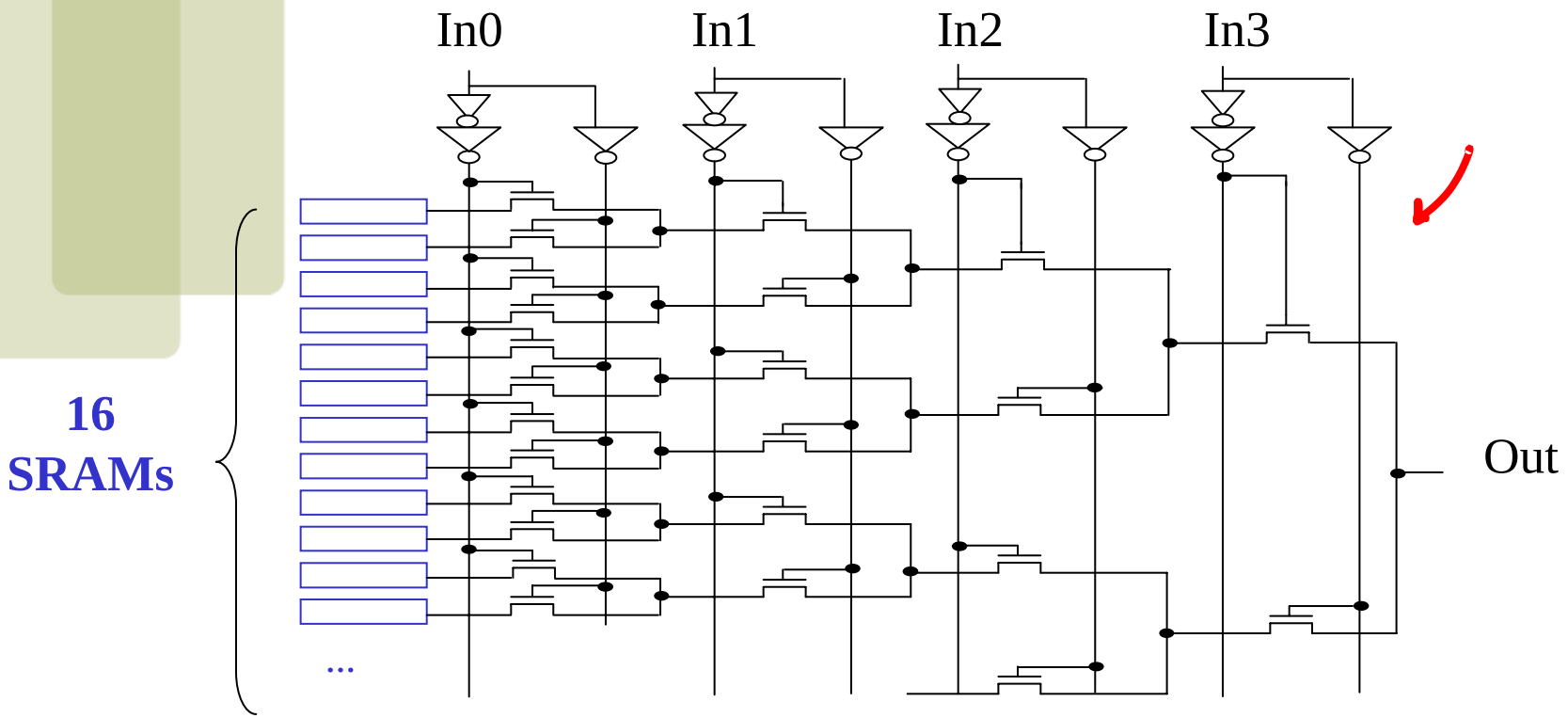


Figure 1: Simplified Block Diagram of XC4000-Series CLB (RAM and Carry Logic functions not shown)

Page 4-12, Xilinx XC400 Series Field Programmable Gate Arrays Product Specification

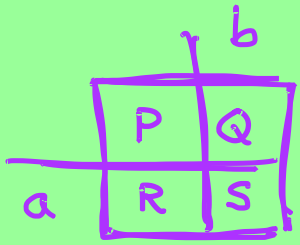
An Implementation of a 4-input Look-up Table (4-LUT)



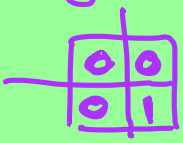
Out = f (in0, in1, in2, in3)

24.1

2-input LUT

- Implements any function of ≤ 2 inputs
- 

[if P=Q=R=0, S=1 we get AND gate



]

- # of squares in K-map = $2^2 = 4$

[if n-inputs, then # of squares in Kmap is 2^n]

- # of possible functions of ≤ 2 inputs?

$= 2^{\#\text{ of squares in Kmap}}$

$= 2^{2^2} = 2^4 = 16$

[if n-inputs then # of possible functions is $2^{\#\text{squares in Kmap}} = 2^{2^n}$]

24.2

Current LUTs use 4-6 inputs

functions they can implement

is

4: $2^{2^4} = 2^{16} = 64K$

5: $2^{2^5} = 2^{32} = 4G$

6: $2^{2^6} = 2^{64} = 64 \times \underbrace{2^{30}}_{1G} \times \underbrace{2^{30}}_{1G}$ WOW

K

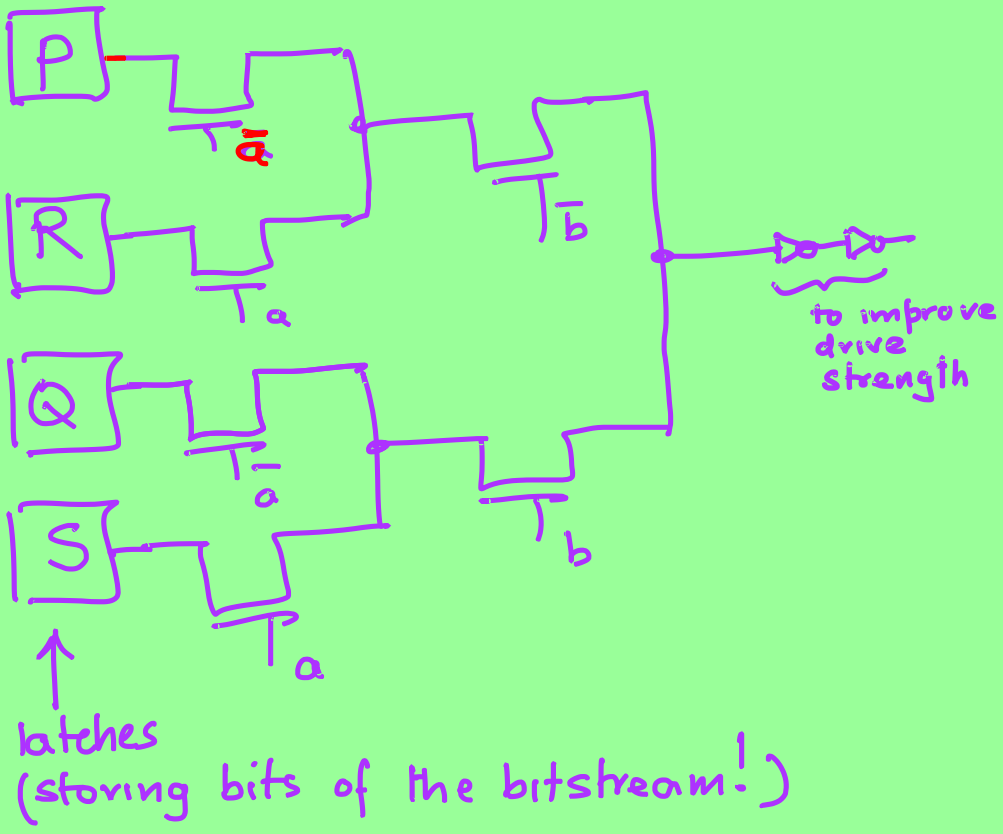
7: $2^{2^7} = 2^{128} = 256$ G G G G

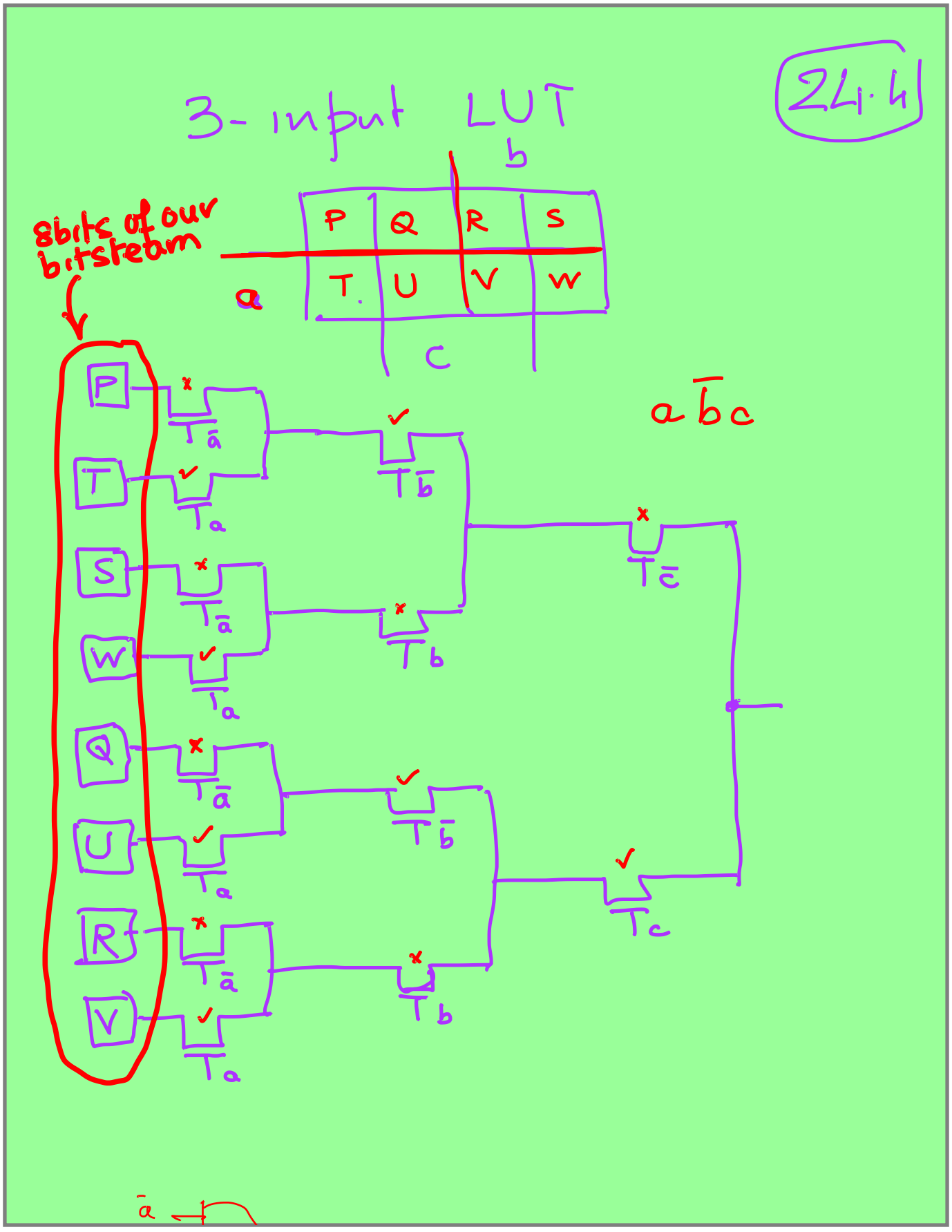
24.3

Lets create a 2-input LUT:

a \ b	$\bar{b}$	b
$\bar{a}$	P	Q
a	R	S

"if $\bar{a}\bar{b}$ then P
elsif $a\bar{b}$ then R
elsif $\bar{a}b$ then Q
elsif ab then S"




24.4
3-input LUT
b
P Q R S
T U V W
a
c
8bits of our bitstream
P
T
S
W
Q
U
R
V
abc

24.5

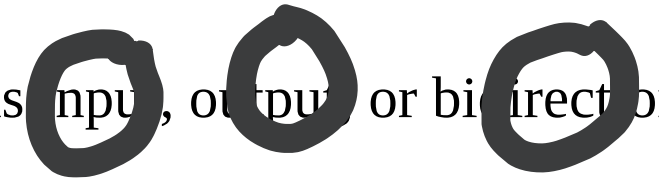
# inputs	# device	# functions
2	4+2=6	$2^{2^2} = 16$
3	8+4+2=14	$2^{2^3} = 256$
4	16+14=30	$2^{2^4} = 64K$
5	32+30=62	$2^{2^5} = 4G$
6	64+62=126	$2^{2^6} = 16\ G \cdot G$

w/o counting output

cost << benefit

Input-Output Blocks

- One IOB per FPGA pin
  - Allows pin to be used as input, output or bidirectional (tri-state)



- Inputs
  - Direct
  - Registered
  - Drive dedicated decoder logic for address recognition

- IOB may also include logic for boundary scan (JTAG)

Xilinx IOB

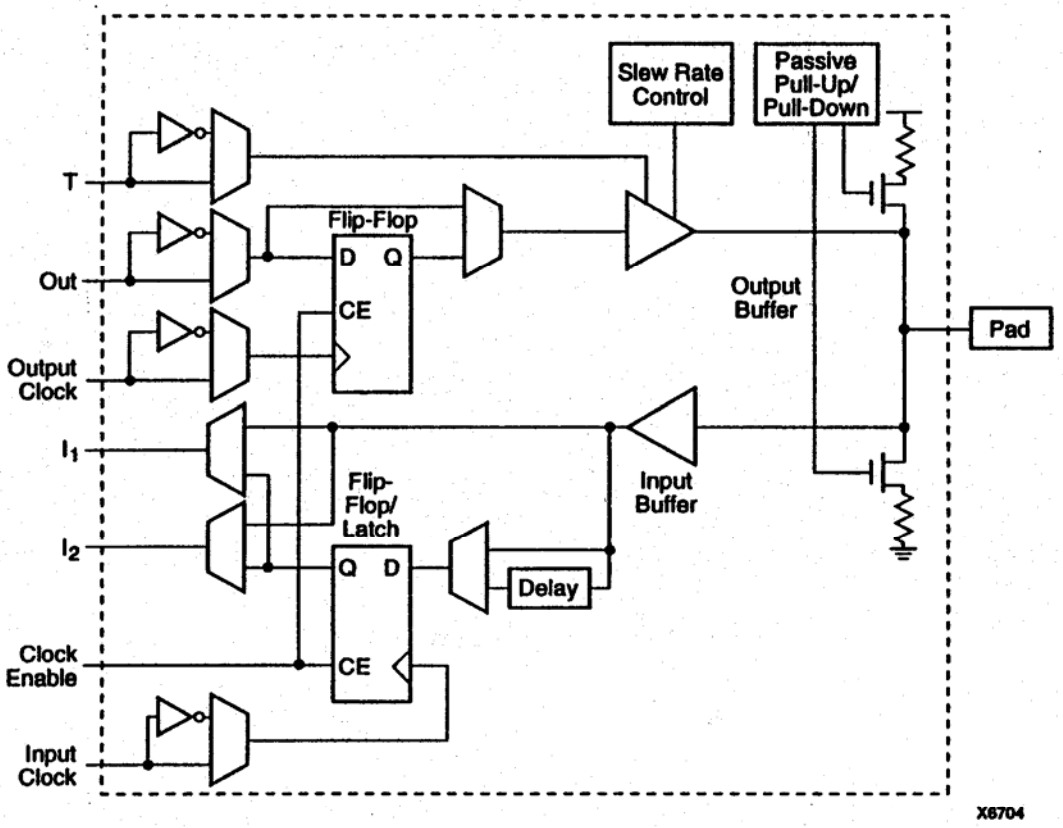


Figure 16: Simplified Block Diagram of XC4000E IOB

Figure 16, Xilinx XC400 Series Field Programmable Gate Arrays Product Specification

Interconnect

- 2-Dimensional mesh of wires, with switching elements at wire crossings to control routing
  - Bit patterns stored into the switch FFs determine routing
  - Switch connections programmed as part of configuring array

- To optimize for speed, many designs include multiple lengths of wire
  - Single-length (connect adjacent switches)
  - Double-length (connect to switches two hops away)
    - Long lines (run entire length/width of array)

Interconnect Diagram

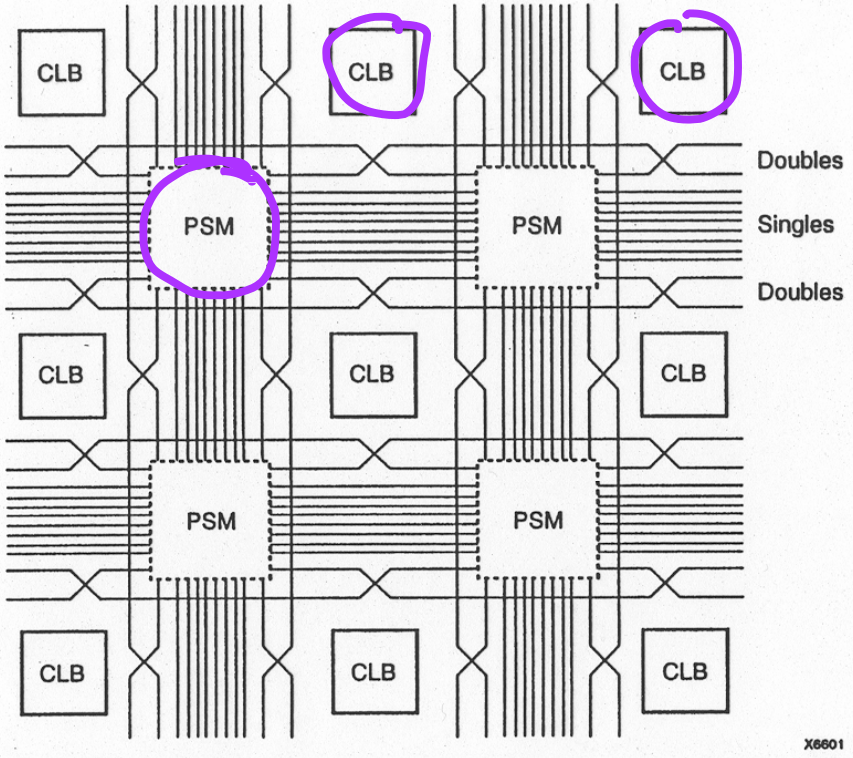
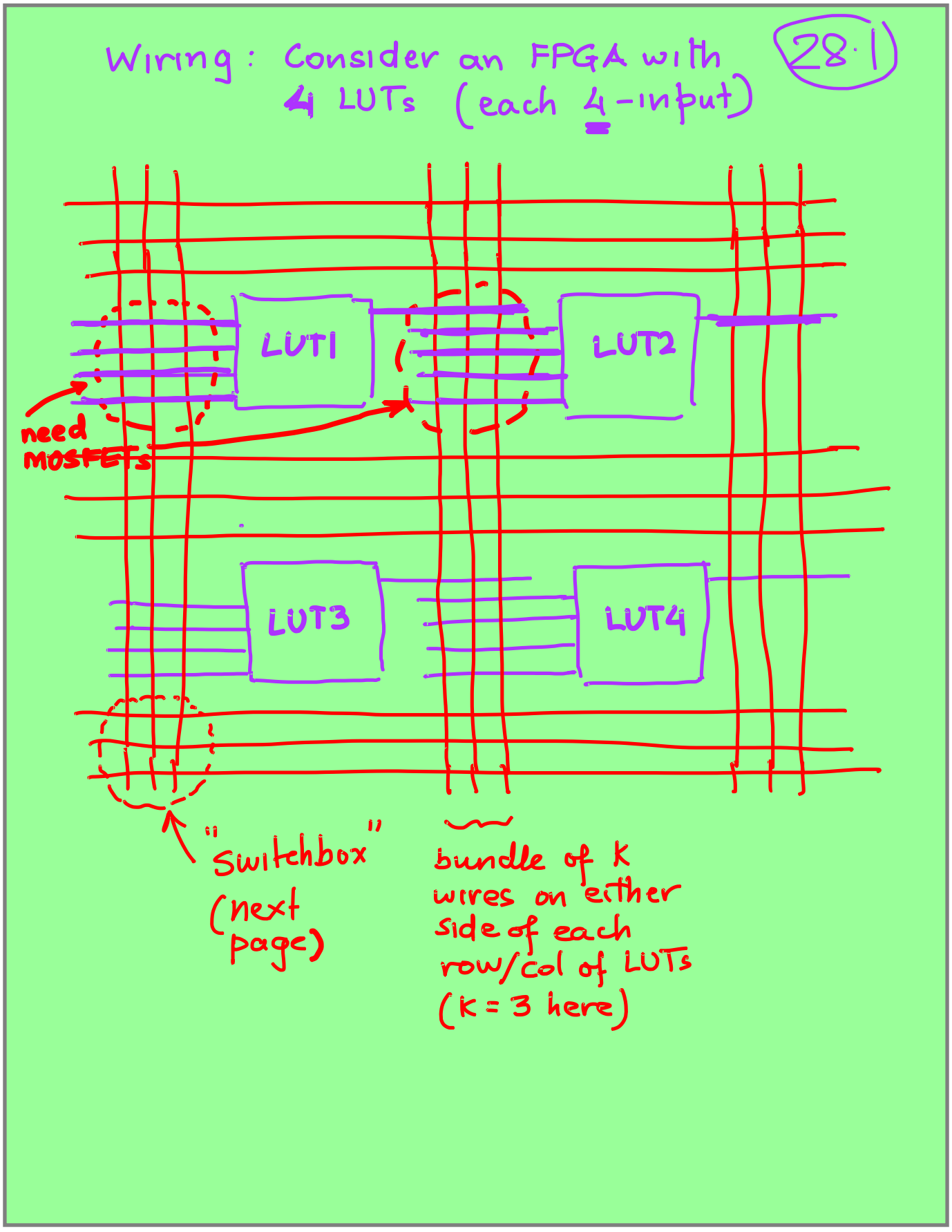


Figure 29: Single- and Double-Length Lines, with Programmable Switch Matrices (PSMs)

Figure 29, Xilinx XC400 Series Field Programmable Gate Arrays Product Specification


Wiring: Consider an FPGA with 4 LUTs (each 4-input)
28.1
LUT1
LUT2
LUT3
LUT4
need MOSFETs
"switchbox" (next page)
bundle of k wires on either side of each row/col of LUTs (k = 3 here)

28.2

switchbox detail

A, B, C, D, .. I are K^2
(9 in our example)
bits of our bitstream

28.3

Consider FPGA with 160000 5-input LUTs. Switchboxes size is 10x10. The FPGA has 200 I/O pins, which can be set in one of 12 ways. How long is the bitstream?

I/O : $\lceil \log_2 12 \rceil \times 200 = 800b$

LUTs : $(2^5+2) \times 160000 = 5.44 \times 10^6 b$

Wires: The FPGA is 400x400 LUTs,

Switchbox = $(401) \times (401) \times 100 = 16.08 \times 10^6$

(401) × (401) : # switchbox

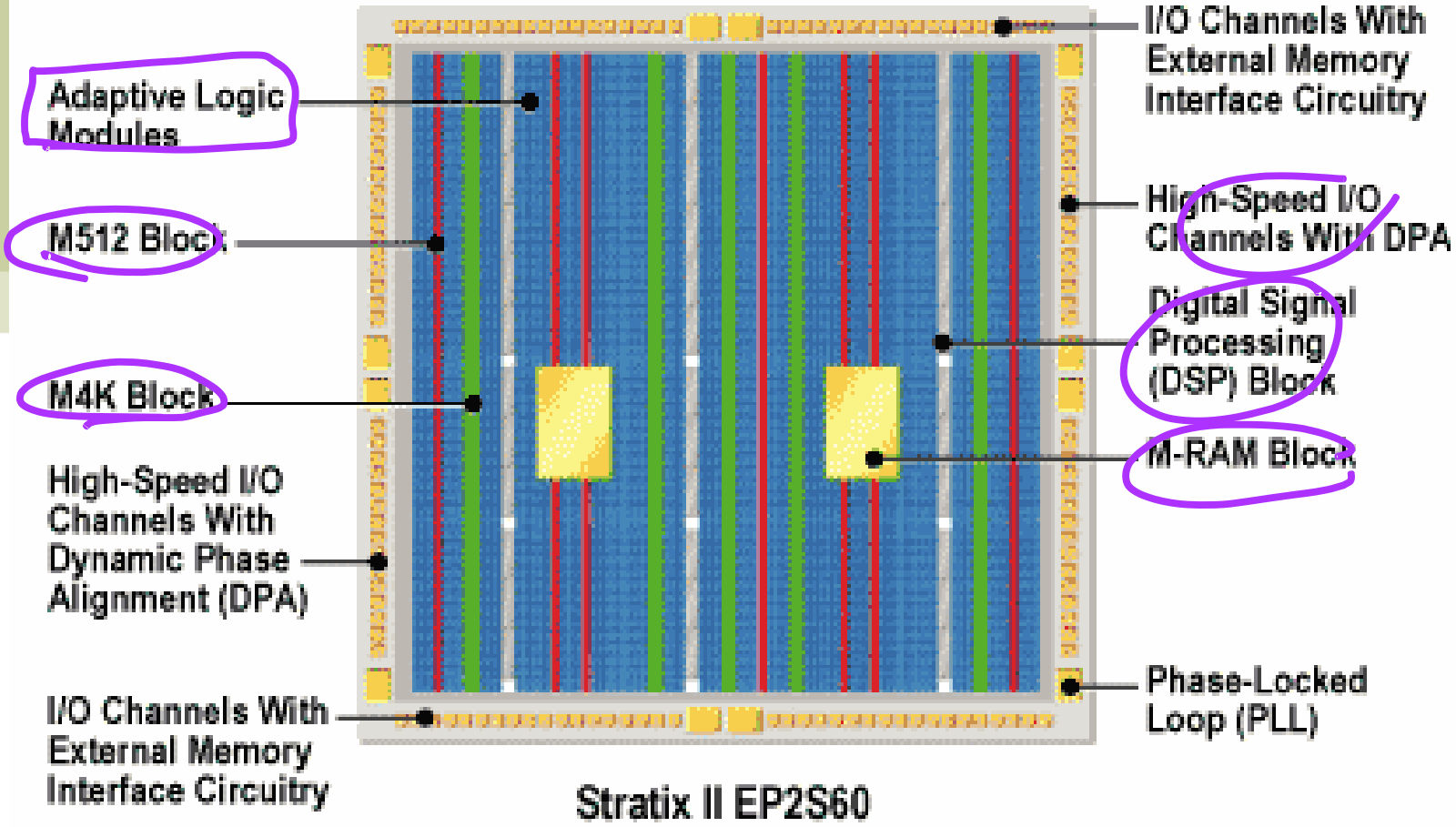
Wires to LUTs = $(160000) \times (5+1) \times 10 = 9.6 \times 10^6$

(160000) : # LUT

(5+1) : 5 in & 1 out

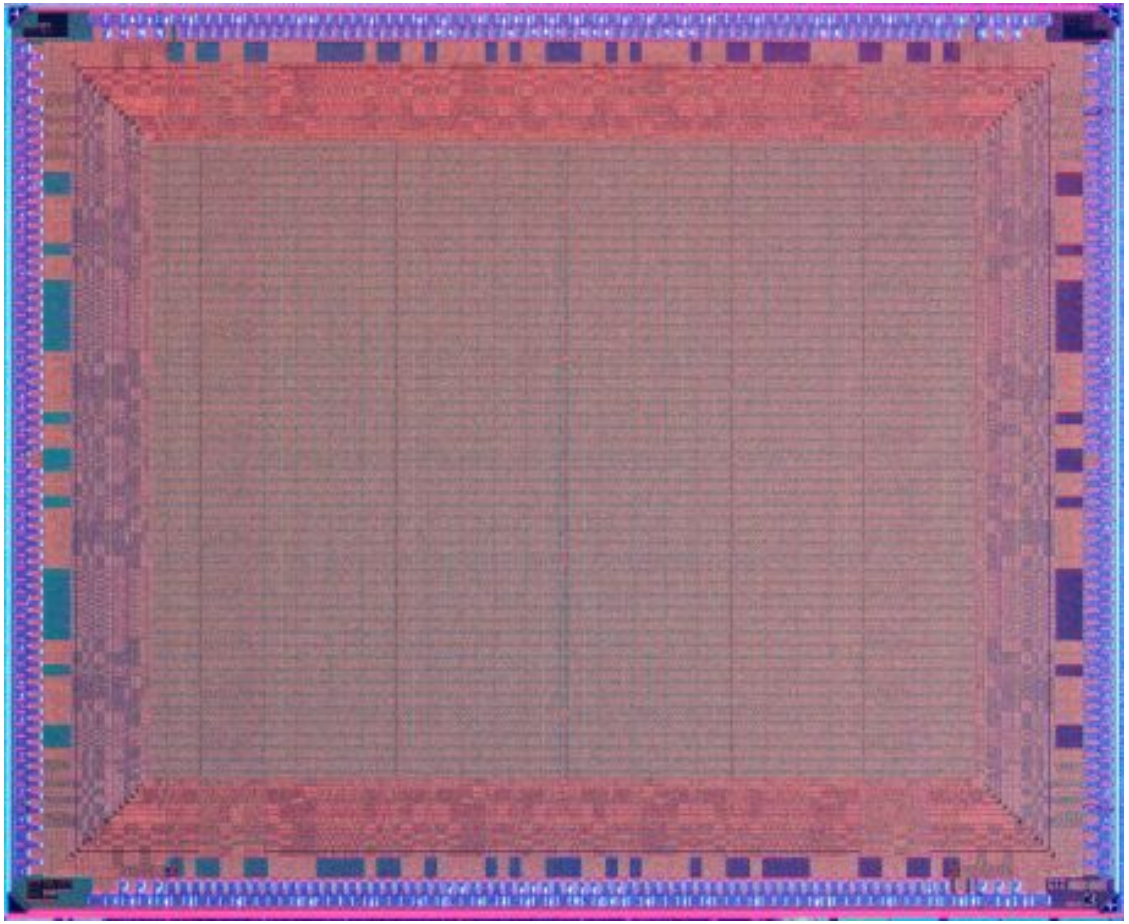
10 : crossings

One Commercial FPGA, Altera Stratix II

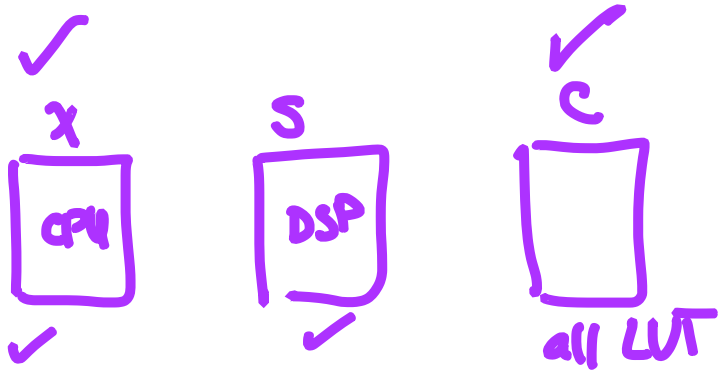


Chip Shot - Xilinx Spartan-3 die image

- Note the regularity…

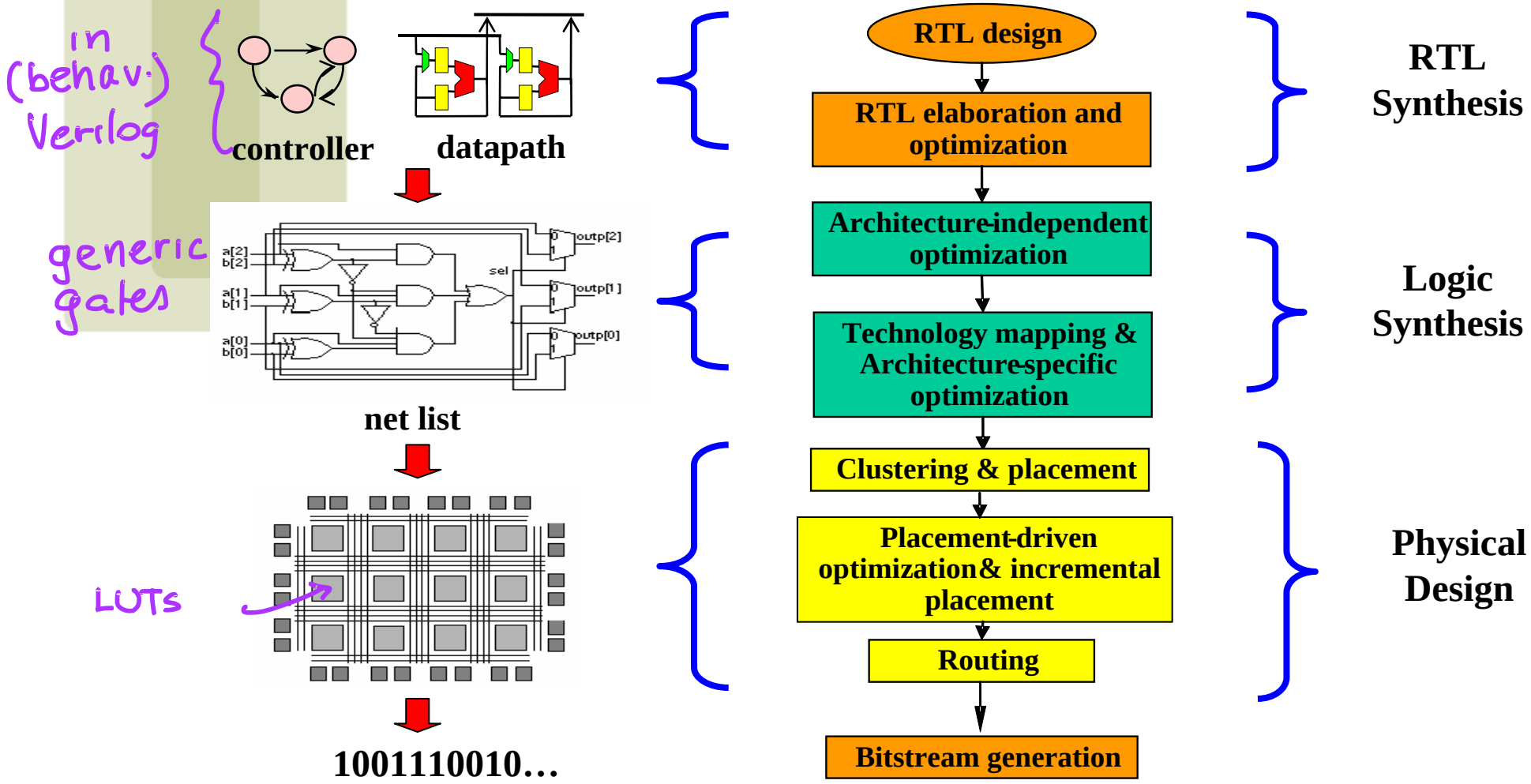


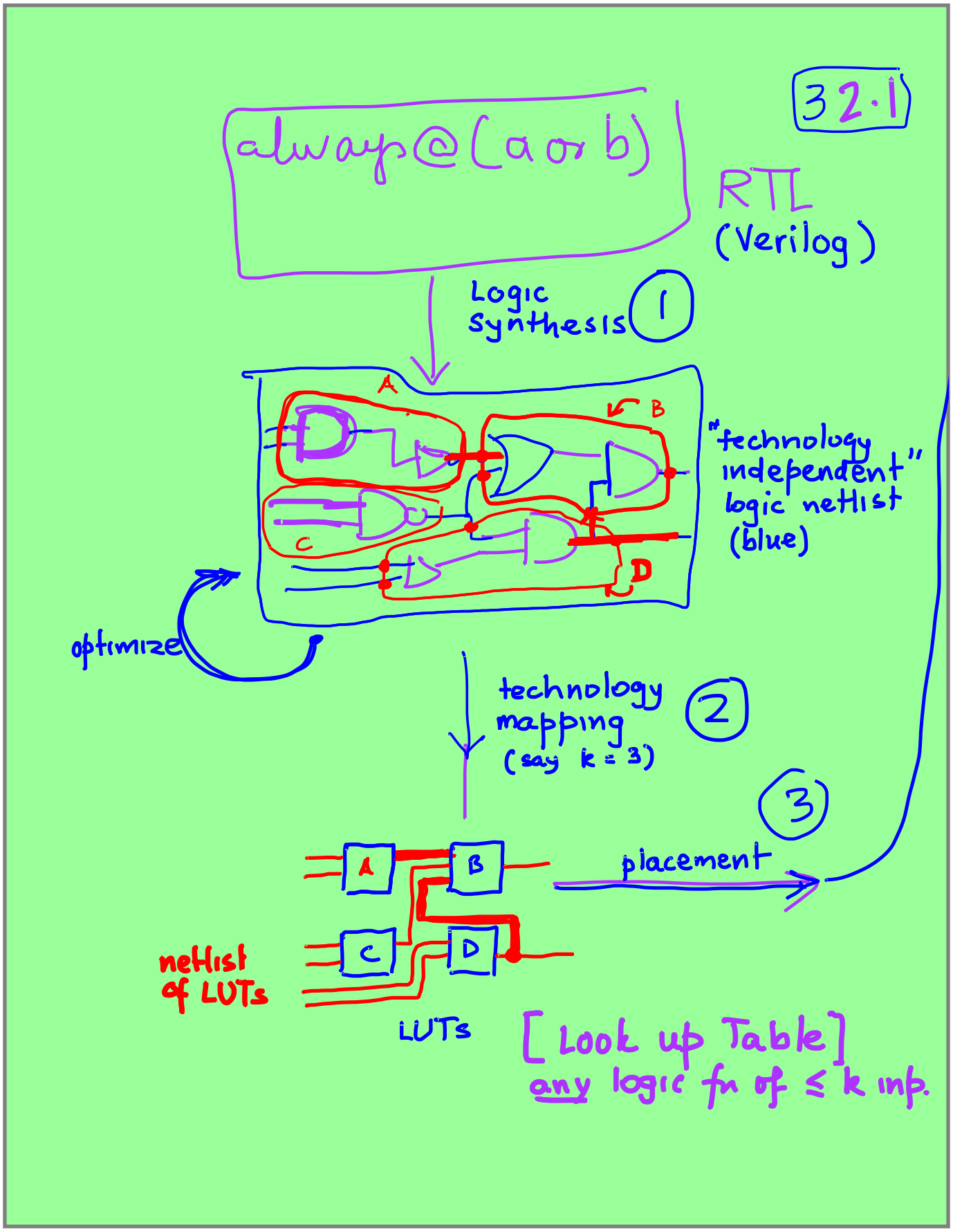
Design Variables

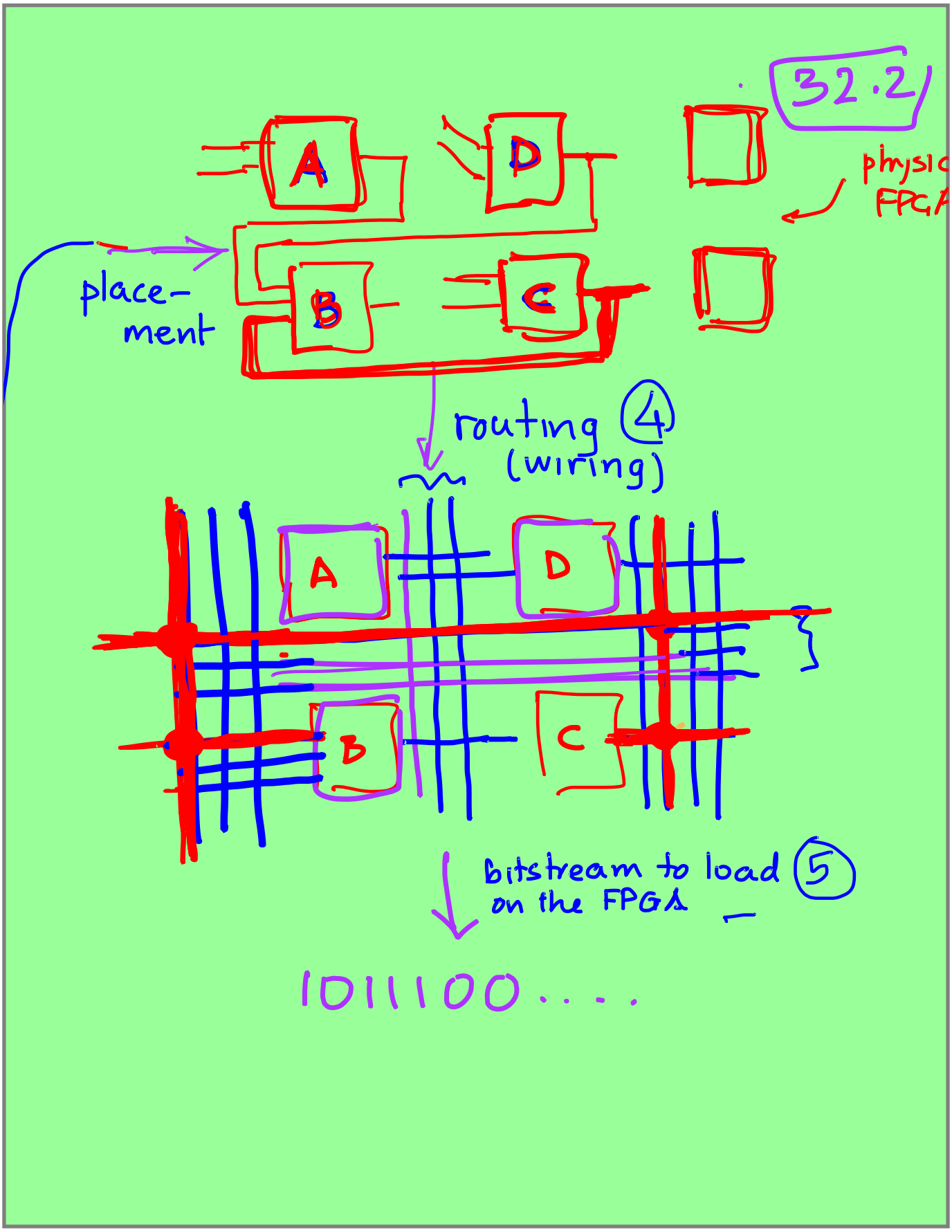


- The following issues are something that the company which designs the FPGA needs to worry about. The user of the FPGA is agnostic to these issues.
- # of inputs to LUT
  - Trade off number of CLBs required vs. size of CLB and routing area
- How is logic implemented
  - Switch based? Gate based?
  - SRAM configuration? Fuse burning configuration?
- Flip-flop in CLB?
- Additional Functionality
  - Carry chains, CPU's, block RAM files

Design Flow for Programmable Logic




32.1
always@(a or b)
RTL
(Verilog)
Logic synthesis
1
A
B
C
D
"technology independent" logic netlist (blue)
optimize
technology mapping
(say k = 3)
2
3
placement
A
B
C
D
netlist of LUTs
LUTs
[Look up Table]
any logic fn of ≤ k inp.


32.2
physic
FPGA
A
D
B
C
place-
ment
routing (wiring) 4
A
D
B
C
Bitstream to load on the FPGA 5
1011100...

32.3

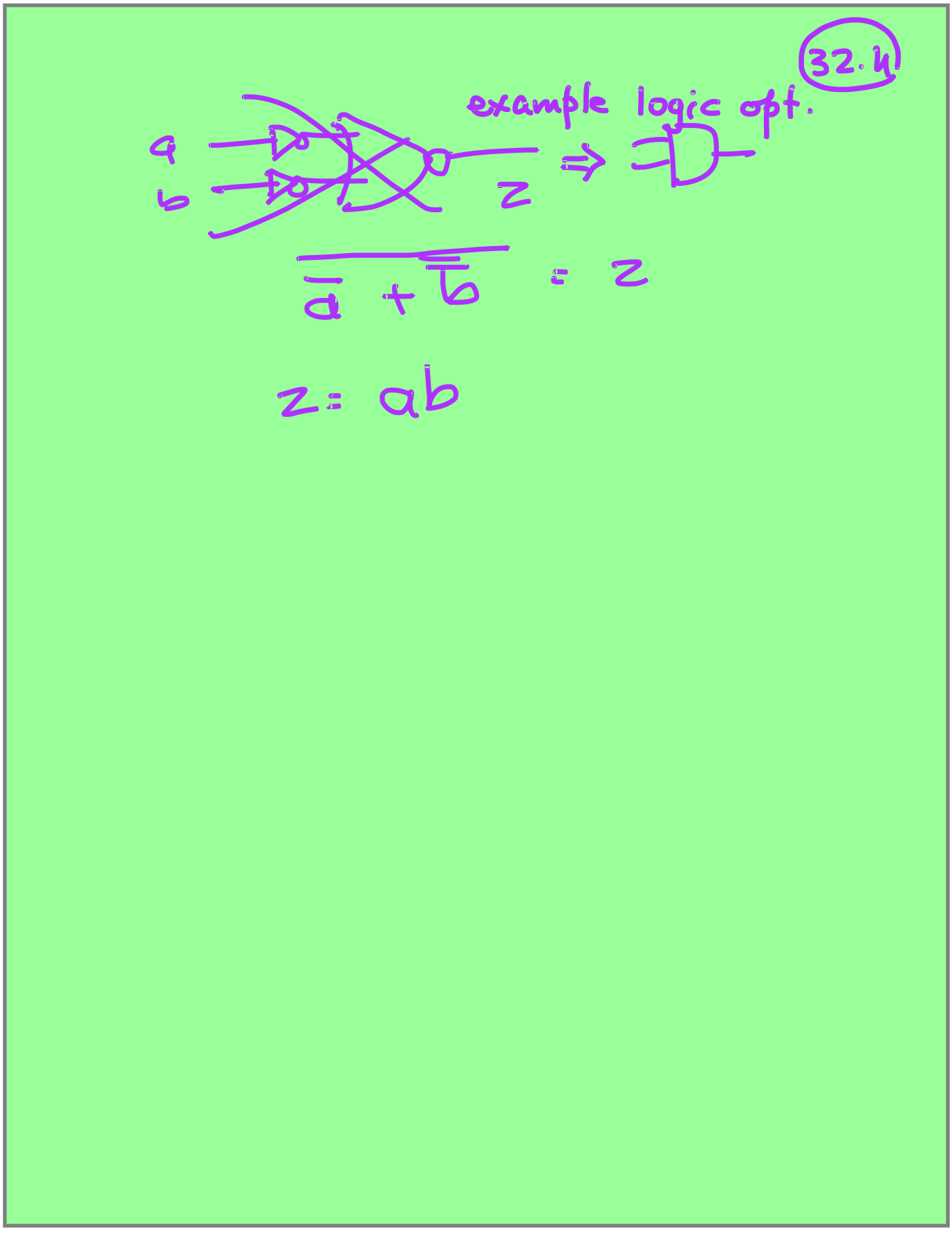
3 inp = $2^3 \equiv 8$ boxes in kmap
$2^8 = (2^{23})$ = # ways · can fill boxes

4 inp = $2^4 \equiv 16$ .. .. ..
$= 2^{16} = 2^{24}$ = unique fns

⋮

k inp = 2^k . . . . . .
2^{2^k} unique fns

6 inp $2^{2^6} = 2^{64}$ functions


32.4
example logic opt.
a
b
z
$\overline{\bar{a} + \bar{b}} = z$
z = ab

FPGAs – Pros (recap)

- Reasonably cheap at low volume
  - Good for low-volume parts, more expensive than IC for high-volume parts
  - Can migrate from SRAM based to fuse based when volume ramps up
- Short Design Cycle
- Reprogrammable (~1sec programming time)
  - Can download bug fix into units you've already shipped
- Large capacity (100 million gates or so, though we won't use any that big)
  - FPGAs in the lab are "rated" at ~1M gates for 30K LE's
- More flexible than PLDs -- can have internal state
- More compact than MSI/SSI

FPGAs – Cons (recap)

- Lower capacity, speed and higher power consumption than building an ASIC
  - Sub-optimal mapping of logic into CLB's – often 60% utilization
  - Much lower clock frequency that max CLB max toggle rate – often 40%
  - Less dense layout and placement and slower operation due to programmability
    - Overhead of configurable interconnect and logic blocks
- PLDs may be faster than FPGA for designs they can handle
- Need sophisticated tools to map design to FPGA. But the FPGA vendor typically provides these tools (at a cost).

FPGA Design Flow

- Now that we know what the circuit structure inside an FPGA is, lets see how we go about programming an FPGA.
- In other words, we will briefly cover the steps that we undertake between
  - The conception of a design idea
  - The decision-making step of whether an FPGA will be the correct hardware platform for the design
  - The design flow we follow to obtain an FPGA based hardware realization of this design.

From Concept to Circuit

- Need to specify your design and then implement it as a functioning system
- Trade-offs between time/cost and efficiency
  - Performance of final system
  - Amount of silicon area required (manufacturing cost)
  - Time to manufacture
  - Power consumption
- Need to think about a number of factors to make decision
  - Sales volume
  - Profit margin and how performance affects it
  - Time-to-market concerns, particularly if trying to be the first product in a new area

High-Level Design

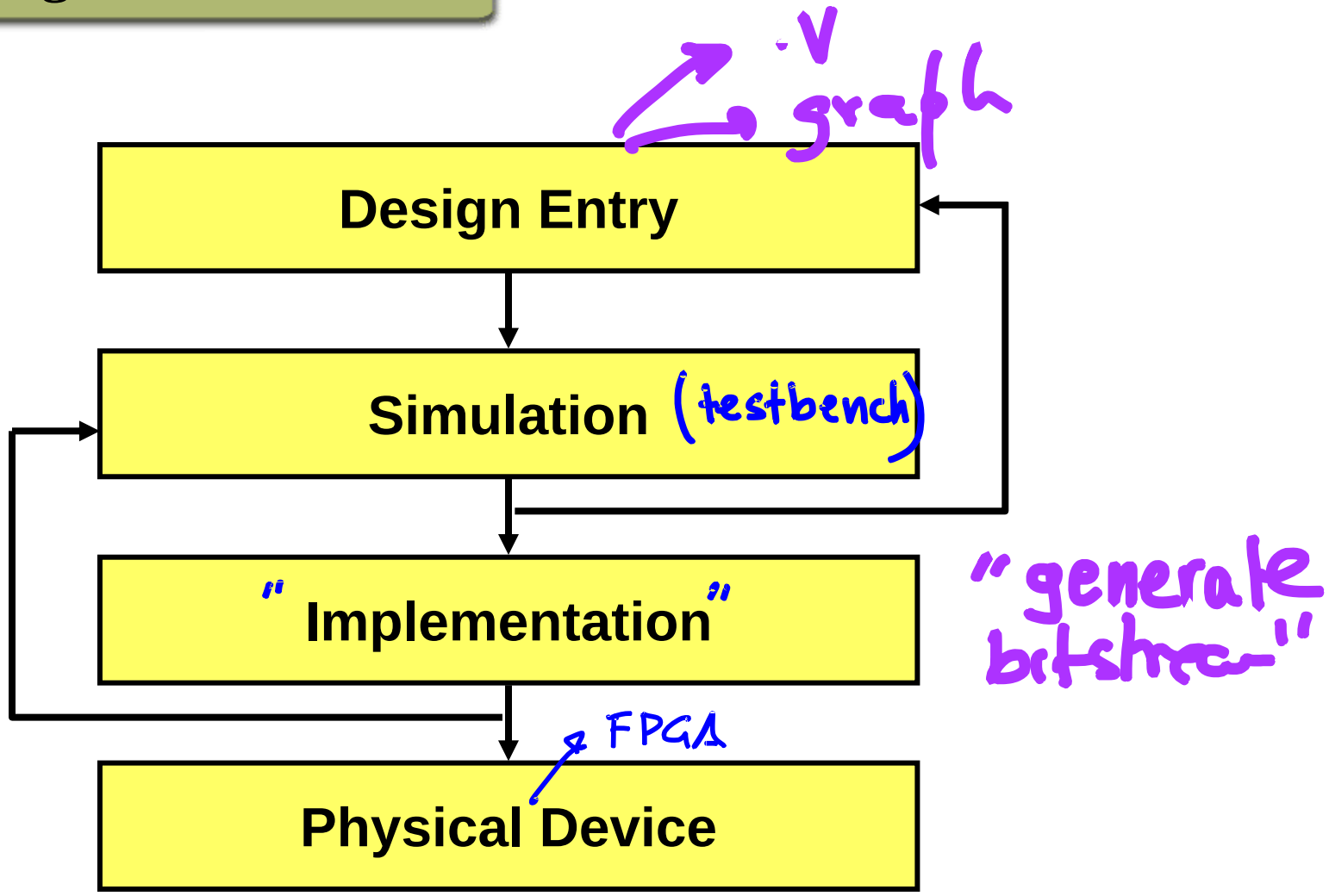
- Problem 1: modern designs are just too complex to keep in your head at one time
  - Custom chips approaching 100M transistors
  - FPGA designs approaching 1M gates
- Problem 2: Even if you could design complex systems by hand, it would take too long
  - 100M transistors at 10s/transistor = 133.5 person-years
  - Transistor counts and system speed increasing at 50% or so/year
  - Design time is critical
- # transistors per chip increasing at 50%/year
- # transistors per engineer-day increasing at 10%/year

- Need techniques that reduce the amount of human effort required to design systems
  - Let humans work at higher levels, rely on software to map to low-level designs

Increasing Design Abstraction

- Old way: specify/layout each device by hand
  - Early chips were laid out by cutting patterns in rubylith with knives
- Current State of the Art: Combination of synthesis and hand design
  - Specify entire system in HDL (Verilog or VHDL), simulate, and test
  - Use synthesis tools to convert non-performance-critical parts of the design to transistors/gates
  - Human designs critical components by hand for performance
- Where Things are Going: System-on-a-Chip Design
  - Specify design out of high-level components (cores)
  - Integrate sensors, transmitters, actuators, computers on a chip
  - Rely very heavily on tools to map design to software and hardware.
  - ***XUP (the board we will use in the lab) is an SoC design vehicle***

(FPGA) Design Flow



Design Entry

Two main methods:

- Text entry (VHDL/Verilog):
  - Compact format, no special tools required
  - Good for high-level designs and control logic

- Schematic Capture: Draw pictorial representation of circuit, tool converts into design (typically HDL description)
  - Traditionally used for low-level (transistor) designs, regular structures
  - Commonly used today in conjunction with text entry to provide visual viewing of overall structure of a design

Simulation

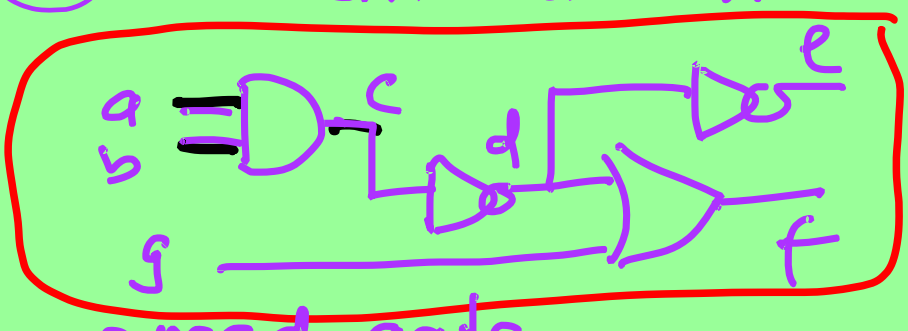
- Two types of HDL simulators
  - Interpreted: runs slower but more versatile and no compilation time
  - Compiled: runs faster but require compilation time and often not as versatile partly due to needs to compile all library components used.
  - emulation
- Both typically use Discrete-Event techniques
  - Divide time into discrete steps
    - User can select time step to trade accuracy vs. run-time
  - Keep lists of events that have to be resolved at each time step.
    - At each time step, resolve all events for the time step and schedule events for later time steps
- Output:
  - Text from output/print statements in your design
  - Errors from assert statements
  - Waveform traces
- Like any testing, the key is having good tests. The designer creates these!

event driven timing simulator

SIMULATION

41.1

(A) Interpreted event driven




```
  1 1 1 1
  10101100
+ 11010111
----------
 110000100
```

- read gate
- perform event driven simulation (see verilog notes) with or without timing

- N gates
- $T = \frac{1}{f}$ = CPU clock period that runs the event driven simulator
- α = activity factor → 20-25% (logic), 90% arithmetic

time to simulate = $100 \cdot T \cdot \alpha \cdot N$ (sec)

- 100: cycles per event
- $\alpha \cdot N$: #events

41 2

- Compiled code

```
c = and(a,b)
d = complement(c)
e = complement(d)
f = or(d,g)
```

32b words

bitwise operator of ISA

↖ same ckt as on page 41.1

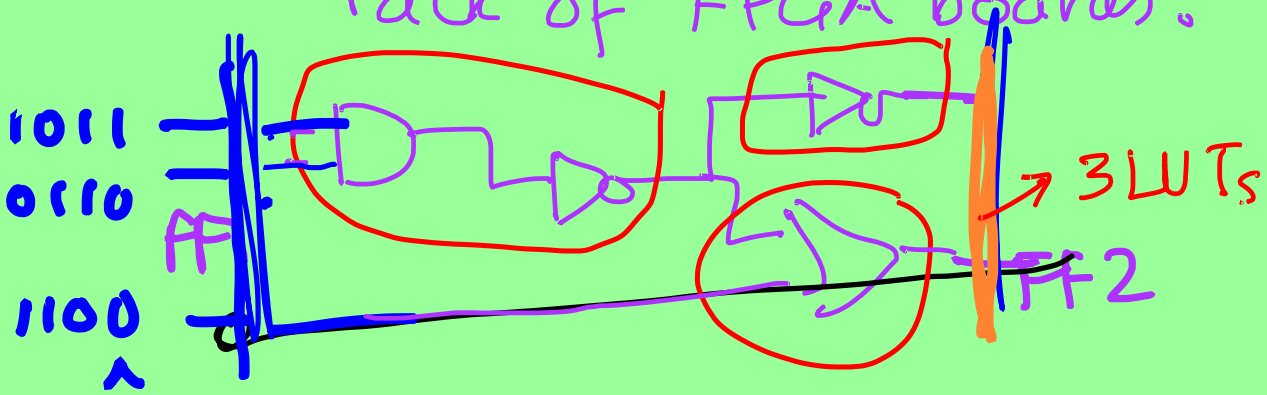
time to simulate = $\frac{N}{32} \cdot T$

41·3

emulation (hardware)

- take the netlist, map to FPGA & emulate on a "fridge-sized" rack of FPGA boards!

10 X 20



D: Logic depth (in LUTs) of the emulated circuit

τ = FPGA LUT delay

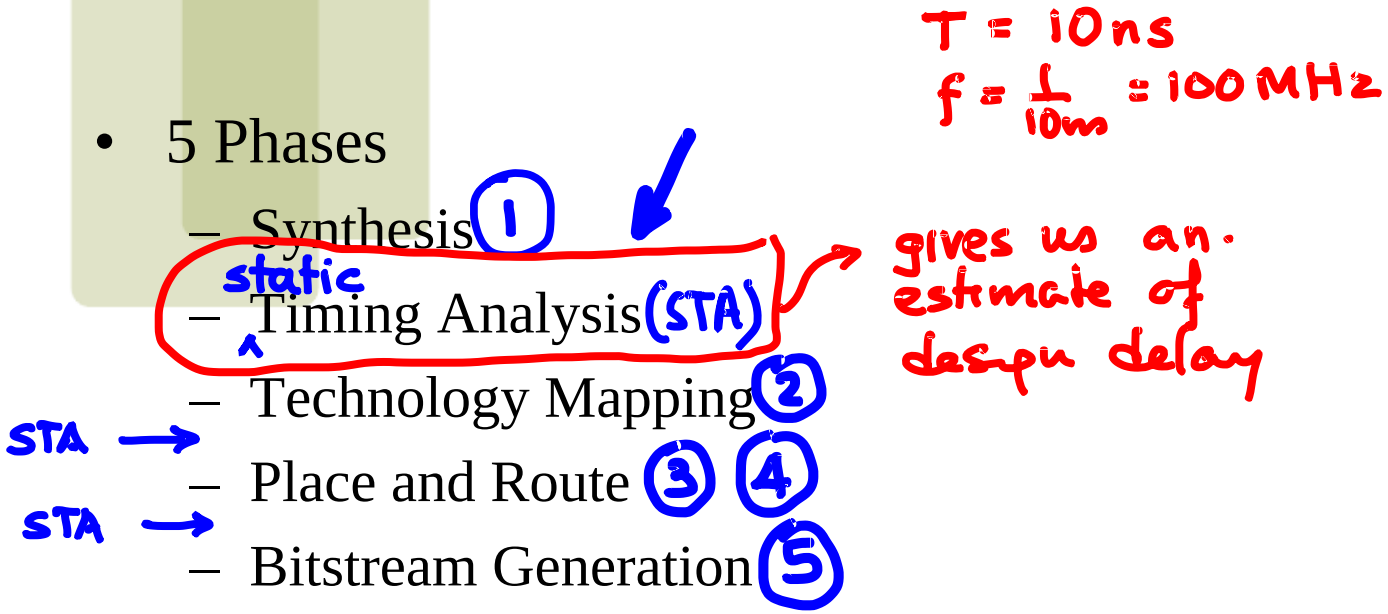
celaro

time to simulate = $D \cdot \tau$

Rx

Implementation of an FPGA Design

Going from simulated Verilog design to circuits



(Sometimes do additional timing analysis after place and route just to make sure that the timing is good)

Synthesis

Verilog or

Transforms program-like VHDL into hardware design (netlist)

- Inputs
  - HDL description
  - Timing constraints (When outputs need to be ready, when inputs will be ready, data to estimate wire delay)
  - Technology to map to (list of available blocks and their size/timing information)
  - Information about design priorities (area vs. speed)
  - effort level

For big designs, will typically break into modules and synthesize each module separately

  - 10K gates/module was reasonable size 5 years ago, tools can 50-100K gates now

Timing Analysis

Static timing analysis is the most commonly-used approach

- Calculate delay from each input to each output of all devices
- Add up delays along each path through circuit to get critical path
- Works as long as no cycles in circuit
  - Tools let you break cycles at registers to handle feedback
- Also, ignores *false paths* in the design.
- Trade off some accuracy for run time
  - Simulation tools like SPICE will give more accurate numbers, but take much longer to run
- If the netlist passes timing analysis tests, we proceed further

14.1

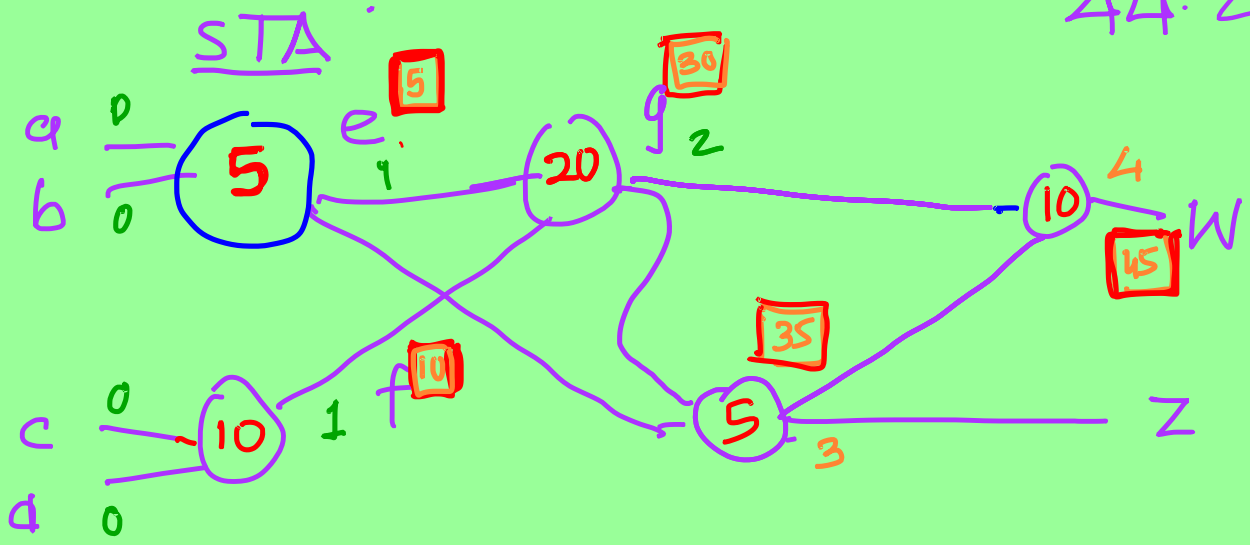
Timing analysis

- In: logic ckt (w/ N gate)
  delays of gates
- Out: Max delay of ckt
  - → static timing analysis (STA)
    - O(N)
    - fast
    - conservative (overestimate the true delay)
  - → exact timing analysis
    - $O(2^N)$
    - exact
    - not viable in practice

out: find max delay of any o/p

44.2

STA



① levelize gates (orange)

- level(g) = 1 + max_{fi ∈ fanin of g} (level(fi))
  (g: some gate)
- level(primary input) = 0

② Arrival time (AT) of a gate g (in level order)

→ $AT(g) = \left[\max_{fi \in \text{fanin of } g} AT(fi)\right] + \Delta(g)$

AT(e) = 5, AT(f) = 10

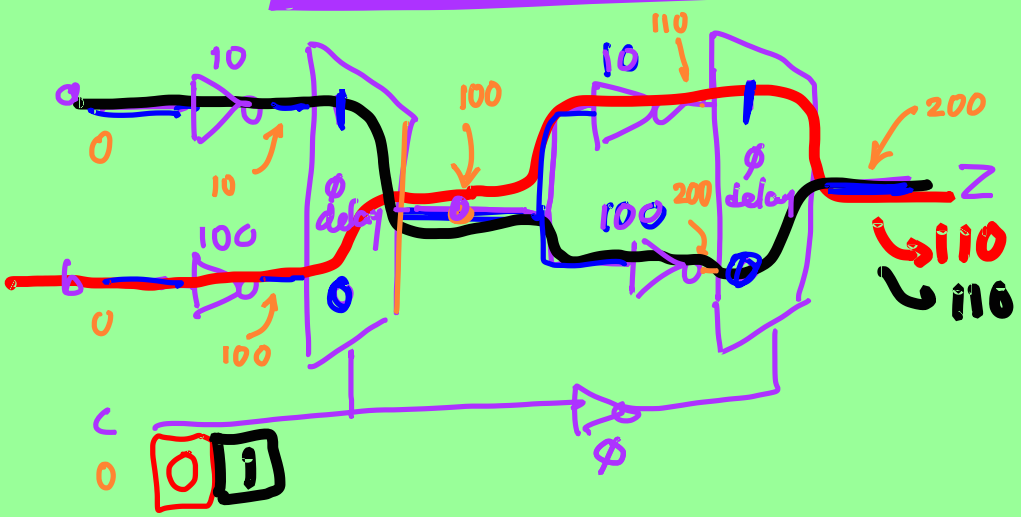
AT(g) = max(5,10) + 20 = 30
AT(z) = max(5,20) + 5 = 35
AT(w) = max(30,25) + 10 = 45

44.3

run
STA time

O(N), N = # gates

conservative
STA pessimism



AT(z) = 200

if real delay is Δ then STA result will be ≥ Δ	110 200

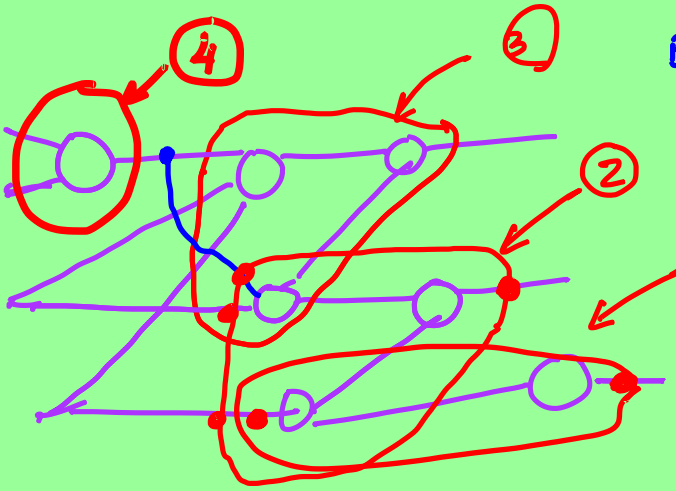
↑ conservative

Technology Mapping

- Technology mapping converts a given Boolean circuit (a netlist) into a functionally equivalent network comprised only of LUTs or PLAs
  - Basically, can divide logic into n-input functions, map each onto a CLB.
- Technology mapping is a crucial optimization step in the programmable logic design flow
- Direct impact on
  - Delay (number of levels of logic)
  - area/power (number of LUTs or PLAs)
  - Interconnects (number of edges)
- Harder problem: Placing blocks to minimize communication, particularly when using carry chains

45

Tech Mapping



Blue = tech. indep. netlist

assume 3-input LUT

- Suppose we use 3-input LUTs (implements any function of ≤ 3 inputs, and exactly 1 output)
  - → each island must have 1 output and ≤ 3 inputs
  - → each circle must be in at least 1 island
  - → islands (LUTs) labeled ①, ②, ③, ④..

Place and Route

Synthesis generates netlist -- list of devices and how they're interconnected

Place and route determines how to put those devices on a chip and how to lay out wires that connect them

Results not as good as you'd like -- ~~40-60~~% utilization of devices and wires is typical for FGPA

- Can trade off run time of tool for greater utilization to some degree, but there are serious limits
- Beyond 80% utilization, there is a good chance that routing will fail.

Bitstream Generation

- A bitstream in FPGA-speak is a sequence of bits, which
  - Determines how the FPGA fabric is customized in order to implement the design.
  - It determines how IOs, CLBs, and wiring are configured.
- This bitstream is loaded (serially) into the FPGA in a final step. Now the FPGA is customized to implement the design we wanted.
  - This loading typically takes a few seconds at most.
- Reprogramming simply means that we load a new bitstream on to the FPGA.