FPGAs for bits & giggles

(or, "As much of a hacky guide to how FPGAs work and how to use them as will fit into 30-60 minutes")

Matt Evans

Make, Hack, Void

May 30, 2011

What is an FPGA?

Definition

Field-Programmable Gate Array

- A flexible logic device used in building digital electronic circuits
- A *gate array* is an array of 'generic' digital logic that can be configured into many different designs.
- Fixed gate arrays exist: The ZX Spectrum's "ULA" was an early example.
- Here, *field-programmable* means it arrives from the factory 'blank', and the connections between gates are designed & configured by the designer/user.
- You do not actually *have* to be in a field, but they do work on farms as well.

What is an FPGA? (2)

Ways of building digital electronic circuits include:

- Valves (Hot, unreliable, huge)
- Discrete transistors (Better, but still pretty huge)
- Silicon chips
  - Mass-produced discrete logic, e.g. 74xxx-series logic (You'll need a lot of wire)
  - Make your own custom silicon chip (Great! But you'd better be a millionaire)
  - Make your own ULA/Gate Array (You'll still need to make thousands)
  - Buy a programmable mass-produced chip, and customise it (Cheap and good!)

What is an FPGA? (2)

Ways of building digital electronic circuits include:

- Valves (Hot, unreliable, huge)
- Discrete transistors (Better, but still pretty huge)
- Silicon chips
  - Mass-produced discrete logic, e.g. 74xxx-series logic (You'll need a lot of wire)
  - Make your own custom silicon chip (Great! But you'd better be a millionaire)
  - Make your own ULA/Gate Array (You'll still need to make thousands)
  - Buy a programmable mass-produced chip, and customise it (Cheap and good!)

Custom logic can solve probs microcontrollers can't; you can't use a microcontroller to bit-bang a PCI bus.Custom circuits tend to be faster than general circuits programmed to perform tasks.Can design custom peripherals to interface to whatever you like, process whatever you like. BCD :-)

Programmable Logic Devices (PLDs)

- Mass-produced ($\therefore$ cheap) chips designed to be 'soft-configured' by the designer.
- *Programmable* = configurable logic functions, **not** software-driven

Device	**Era**	**Capacity**	**Cost**	**Volatile**
PAL/PLA/GAL	1970s-1990s	Low	Low	$\times$
CPLD	1980s-	Medium	Low	$\times$
FPGA	1980s-	High	Medium	Mostly ✓

- Smaller PLDs often used for glue logic
- Larger PLDs often used for core datapath logic

Why use an FPGA?

As a low-to-medium volume replacement for designing an ASIC:

- Chip manufacture is v. expensive
- VLSI design tools are v. expensive too! (e.g. $100K per person per year)
- *Not worth it unless you ship a million!*
- Xilinx/Altera/Lattice/etc. make FPGAs in high volumes
- FPGA unit cost ∴ lower than ASIC for small to medium volumes
- FPGA design tools free (beer) or cheap (ish $3K)

Why else?

Their reprogrammability opens up new uses!

- Excellent for prototyping future products (e.g. ARM/IBM/Intel prototype new CPUs on FPGAs)
- FPGAs in the actual product
  - Media equipment – new firmware can implement new video processing/smoothing/scaling algorithms
  - Fixing bugs in products
  - Field upgrades/added features
  - Reconfigurable computing (Hot research topic!)
- Fun and retrocomputing:
  - Open Graphics Project (Open video card)
  - Minimig (C= Amiga recreation)
  - fpgaarcade.com (Retro arcade machines)
  - opencores.org

Some example uses for FPGAs

- SGI Imagesync PCI card (Low requirements, lowish volume)
- My oscilloscope (First production run, not worth making an ASIC – yet)
- CERN's LHC detector/acquisition circuitry
- Jodrell Bank radio telescope
- SGI Altix RASC (Supercomputing with algorithm-in-FPGA accelerator)
- Crypto cruncher systems (Parallel brute-force work)
- Network switching (Reconfigure protocol support)

Some example uses for FPGAs

- SGI Imagesync PCI card (Low requirements, lowish volume)
- My oscilloscope (First production run, not worth making an ASIC – yet)
- CERN's LHC detector/acquisition circuitry
- Jodrell Bank radio telescope
- SGI Altix RASC (Supercomputing with algorithm-in-FPGA accelerator)
- Crypto cruncher systems (Parallel brute-force work)
- Network switching (Reconfigure protocol support)

1. SGI card: Low volume, low requirements, simple FPGA is cheap
2. LHC detectors: Hundreds of them, lowish-volume, massive parallel DSP
3. Jodrell Bank has custom DSP/receiver circuitry. Very low-volume... Maybe other telescopes to, too.
4. SGI RASC: FPGA brick; an HPC application could move some of the algorithm into hardware which appeared in the shared-memory fabric, could stream data through it.
5. Even though absolute perf not as high as real silicon (x00MHz) can still process massive amounts of data by repeating the circuit 10x, 100x. (parallelism)
6. Low-med volume appliances, sometimes even high-value appliances (where flexibility outweighs cost), e.g. those requiring reconfiguration flexibility

How do they work?

Several different vendors but most FPGAs have similar principles of operation:

- Matrix structure of Configurable Logic Blocks (CLBs)
- Configurable interconnect between them
- Power up 'blank', configure from flash ROM
- Very flexible I/O pads (LVTTL, LVDS, PCI, HSTL, gigabit transceivers)
- Dedicated 'primitive' blocks: Dual-port RAM, multipliers, etc.
- Clock distribution networks
- Regular delays – Tools can analyse timing

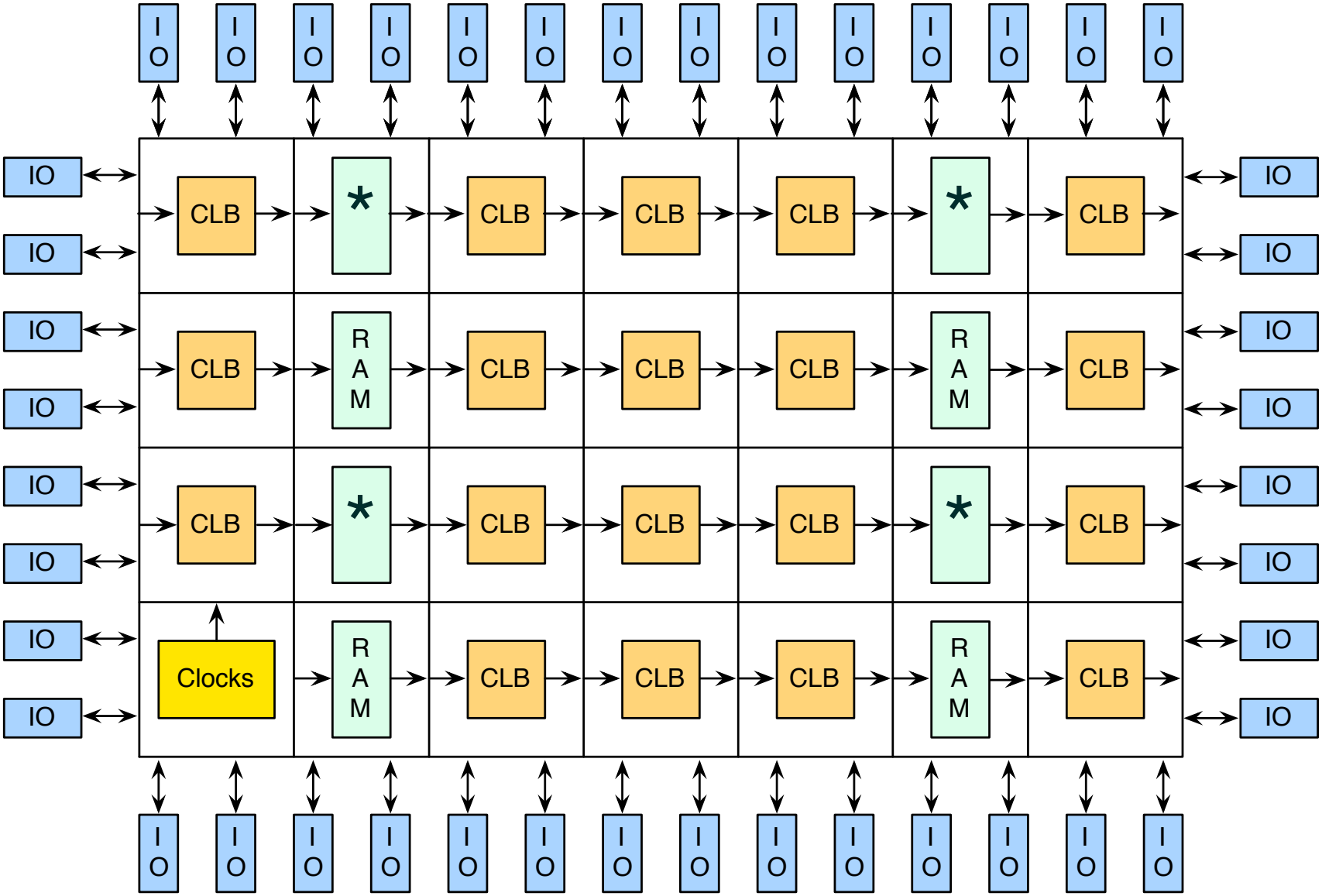
How do they work?

Several different vendors but most FPGAs have similar principles of operation:

- Matrix structure of Configurable Logic Blocks (CLBs)
- Configurable interconnect between them
- Power up 'blank', configure from flash ROM
- Very flexible I/O pads (LVTTL, LVDS, PCI, HSTL, gigabit transceivers)
- Dedicated 'primitive' blocks: Dual-port RAM, multipliers, etc.
- Clock distribution networks
- Regular delays – Tools can analyse timing

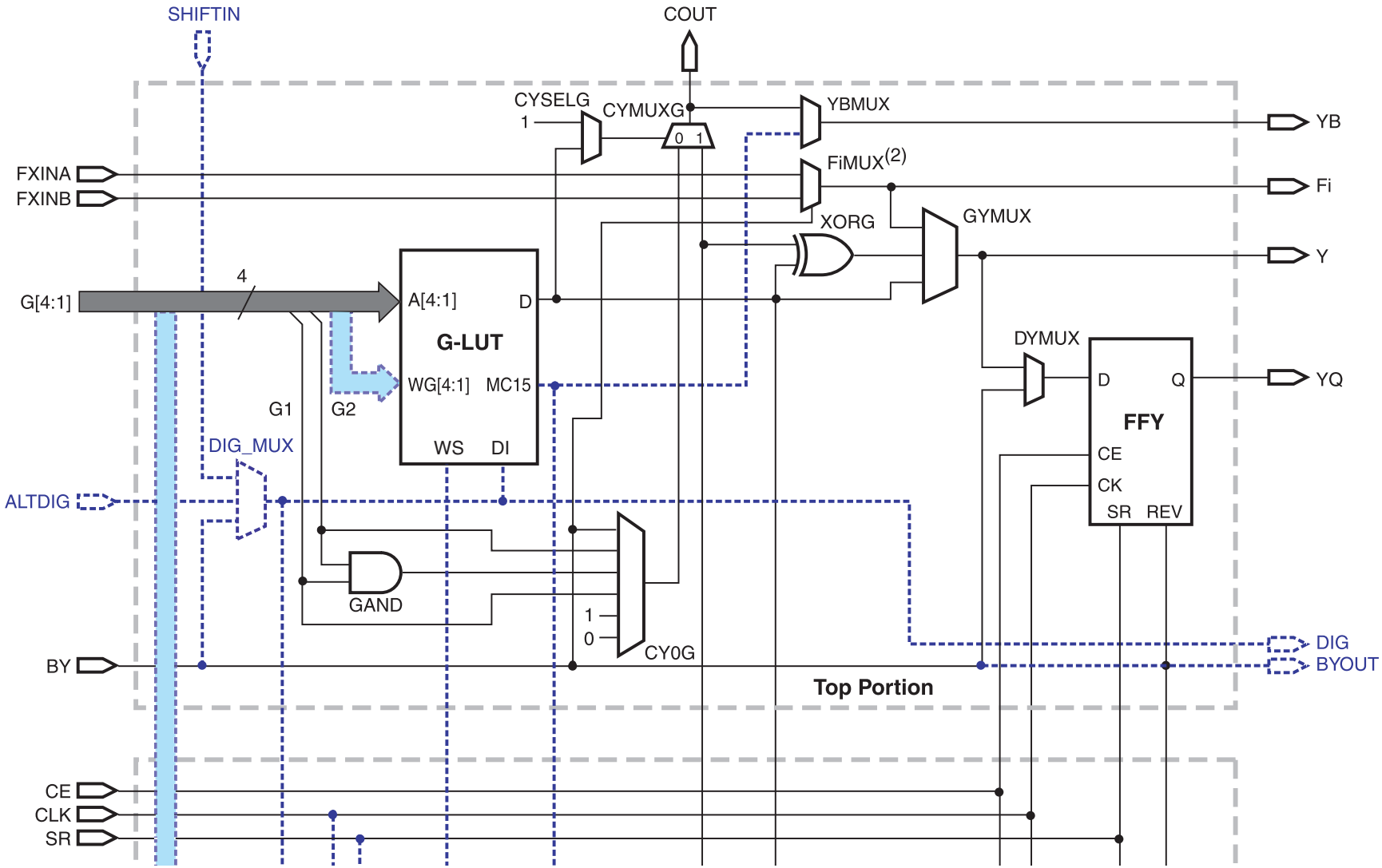
1. This talk is Xilinx-centric! Principles are common though.
2. CLBs perform the logic functions, contain flip-flops (STORAGE)
3. Some have internal flash, others external ROM chips, can upload from host processor too.
4. Excellent interfacing possibilities; can drive fast buses (e.g. video panels, PCI) even with modest FPGAs. With more exotic you can do PCIe, multi-gigabit ethernet, etc.
5. On ASIC you'd make custom blocks; either impossible or v. inefficient to make with CLBs so vendors usually have dedicated 'primitives' scattered around. FASTER, SMALLER.
6. Talk about regular (well-characterised) propagation delays, setup/hold times on latches; tools will tell you how fast you can run your design. OR specify constraints.

FPGA chip structure in Lego^{TM}



Xilinx Spartan3E CLB detail

(Stolen from the Spartan3E datasheet)

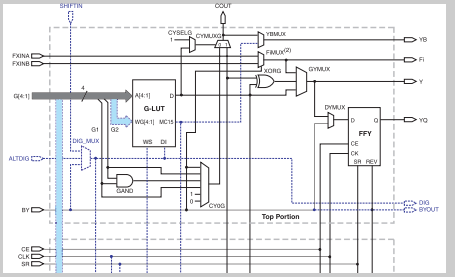


FPGAs for bits & giggles

Xilinx Spartan3E CLB detail

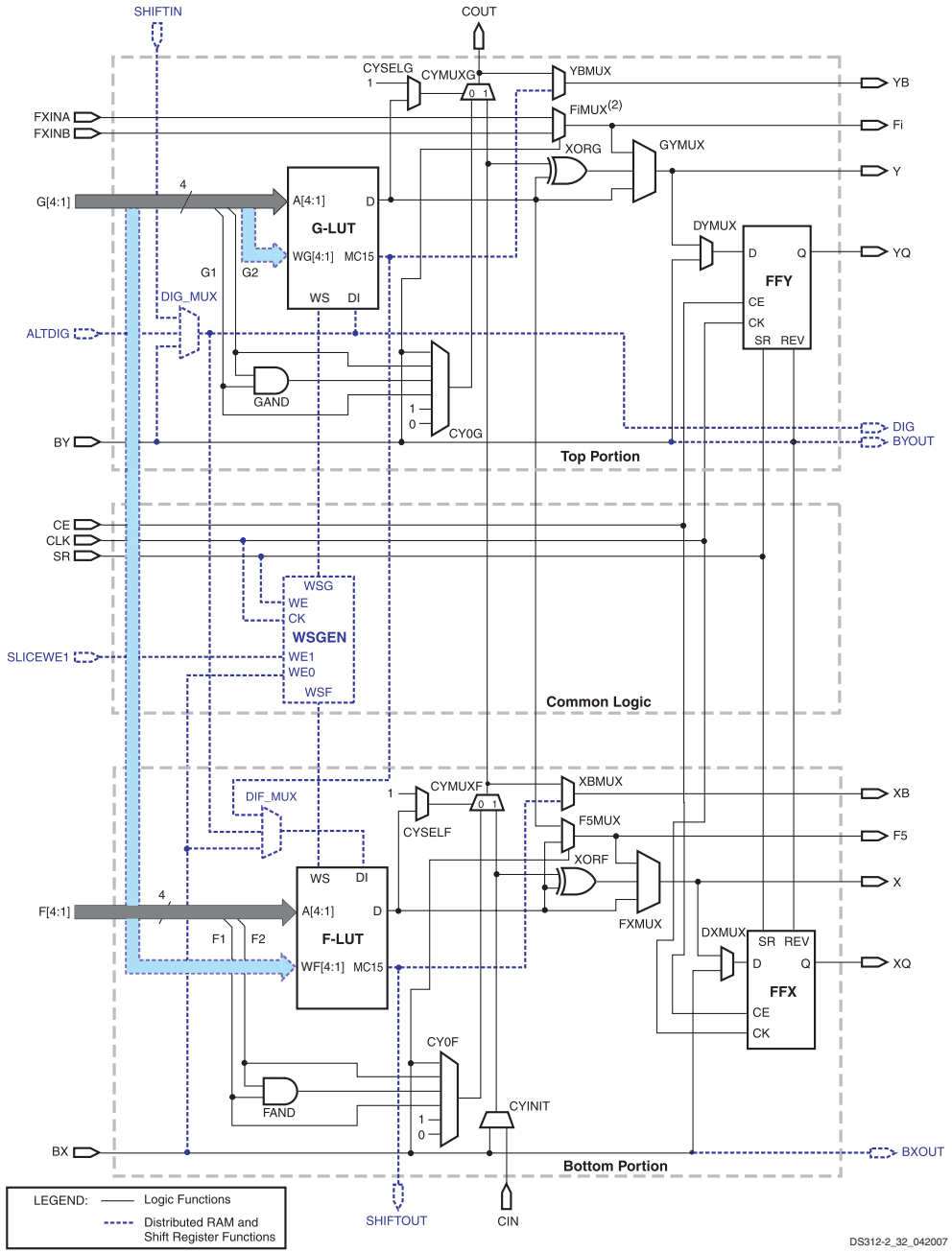
Xilinx Spartan3E CLB detail

(Stolen from the Spartan3E datasheet)



1. Will cover functions later– LUT = output function of the inputs
2. LUTs 4-input here, each CLB has 4x4input LUTs, 2 latches
3. Contemporary Spartan6/Virtex7 has 6 input LUTs
4. Wakes up 'blank' – "bitstream" configures CLB operation

Xilinx Spartan 3E CLB bird's eye view



Xilinx FPGA features

Over time, more specialised functional blocks have been added:

Family	**BRAM**	**DSP**	**PCIe**	**MemCtrl**	**Gigabit IO**
Spartan	×	×	×	×	×
Spartan-2E	✓	×	×	×	×
Spartan-3E	✓	✓	×	×	×
Virtex-4	✓	✓	×	×	✓
Virtex-5	✓	✓	✓	×	✓
Spartan-6	✓	✓	✓	✓	✓

More new features: ADCs, PowerPC & ARM Cortex CPU cores/SoCs

OK, so how do you *do stuff*?

1. Find your FPGA hardware (e.g. starter kit)
2. Sketch your design on paper. Do not just start hacking! :-)
3. Describe logic in:
   - Verilog
   - VHDL
   - Schematic
4. Simulate, simulate, simulate, go to step 2
5. Xilinx Webpack ISE used to build 'bitstream':
   - Verilog/VHDL compiled; functions inferred (e.g. adders, multipliers, RAM)
   - 'UCF' file maps physical pins to signal names
   - Logic Place And Routed, mapped to target device
   - Final layout condensed into configuration bitstream
6. Program FPGA in-situ. Finish, rejoice, or go to step 2 :-)

Let's make something

Quick example of designing something with:

- Spartan-3 Starter Board
- Verilog
- Xilinx's free Webpack ISE design software (on x86 Linux!)
- Iverilog simulator & GTKWave

...and a *very* short course on digital logic design :-)

This won't be a complete course in Verilog because:

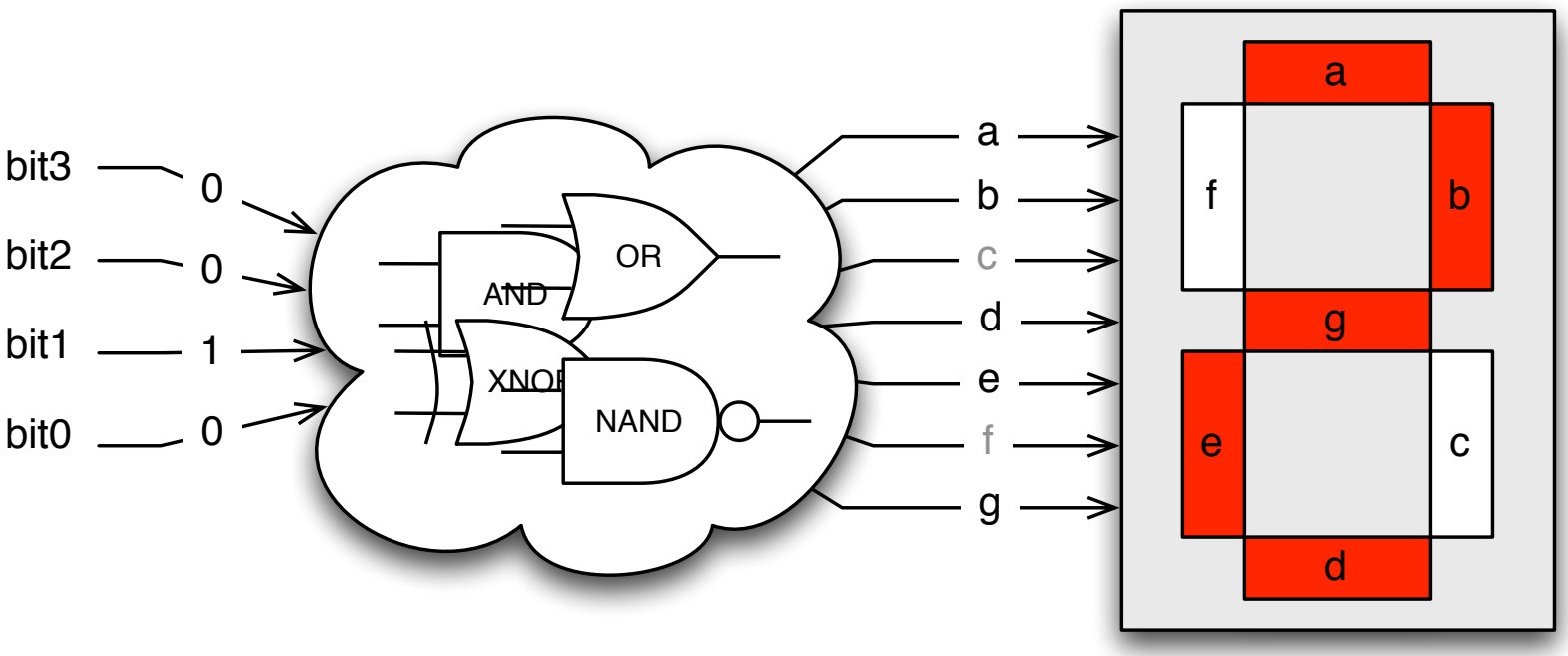
1. There isn't time
2. I don't know all of Verilog :-)

MHV201: Combinatorial logic

Class of logic where a boolean function on a set of signals produces another set of signals.

No storage (latches/flip-flops), only logic gates.

Example: Decode a 4-bit BCD digit (0-9) into 7 segment signals for an LED display



MHV201: Combinatorial logic

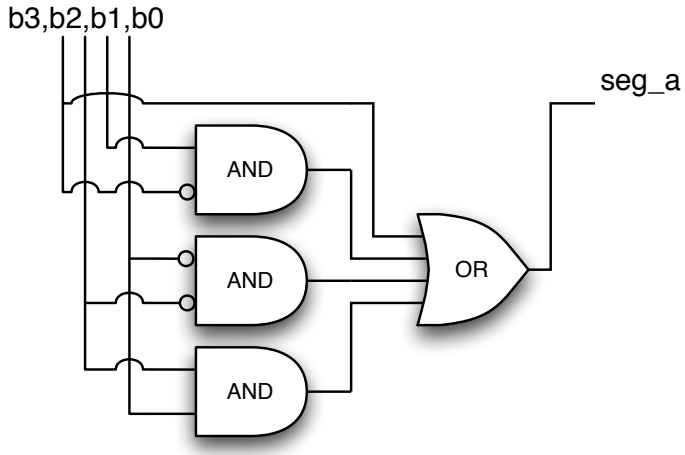
7 segment decoder truth table:

in[3:0]	a	b	c	d	e	f	g
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	1	0	1	1

For each output, minimise equations with a Karnaugh map...

$\overline{seg_a} = in[3] + in[1].\overline{in[3]} + \overline{in[0]}.\overline{in[2]} + in[2].in[0]$

...and draw schematic:



MHV201: Combinatorial logic in Verilog

Ha! Verilog makes this much easier.

The language describes wires, registers (later) and the connections between them.

- Modular hierarchy: define something then instantiate it N times
- Performs logic minimisation
- Can infer Xilinx primitives (e.g. multiply)
- Used for both hardware description ("RTL") and test harness/general programming language ("behavioural Verilog")

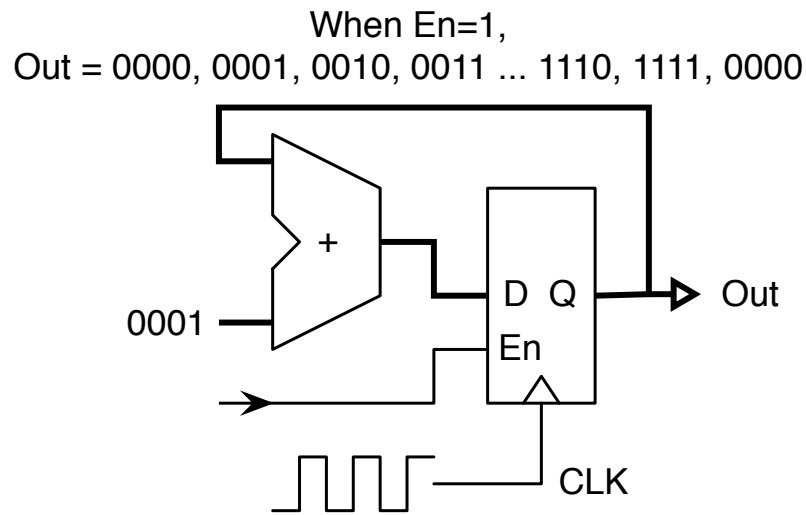
Verilog: 7-segment decoder

See `sources/bcd_to_7seg.v`, and simulate it with test harness `tb/tb_7seg.v`

MHV201: Sequential logic (state machines)

On a (usually) rising edge of the system clock, a flip-flop can grab its inputs and hold them until told to do it again.

A simple counter can be made by combining an incrementer (combinatorial) with flipflops, in a loop:

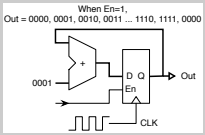


This is a *state machine*; it goes through a number of states with well-defined transitions between them.

MHV201: Sequential logic (state machines)

On a (usually) rising edge of the system clock, a flip-flop can grab its inputs and hold them until told to do it again.

A simple counter can be made by combining an incrementer (combinatorial) with flipflops, in a loop:



This is a *state machine*; it goes through a number of states with well-defined transitions between them.

1. A flipflop has an enable input. When the enable input goes to 1, the flip flop will LATER latch its input at the next clock edge. (Not at the enable edge!) When it's 0, the flip flop holds its output indefinitely.
2. Haven't said much about the clock so far, but the most common design style is "synchronous", where every change of flip flop state is referenced to a common global clock signal.

Verilog: Count to ten

See `sources/counter.v`, and resulting GTKWave!

Build a counter thingy

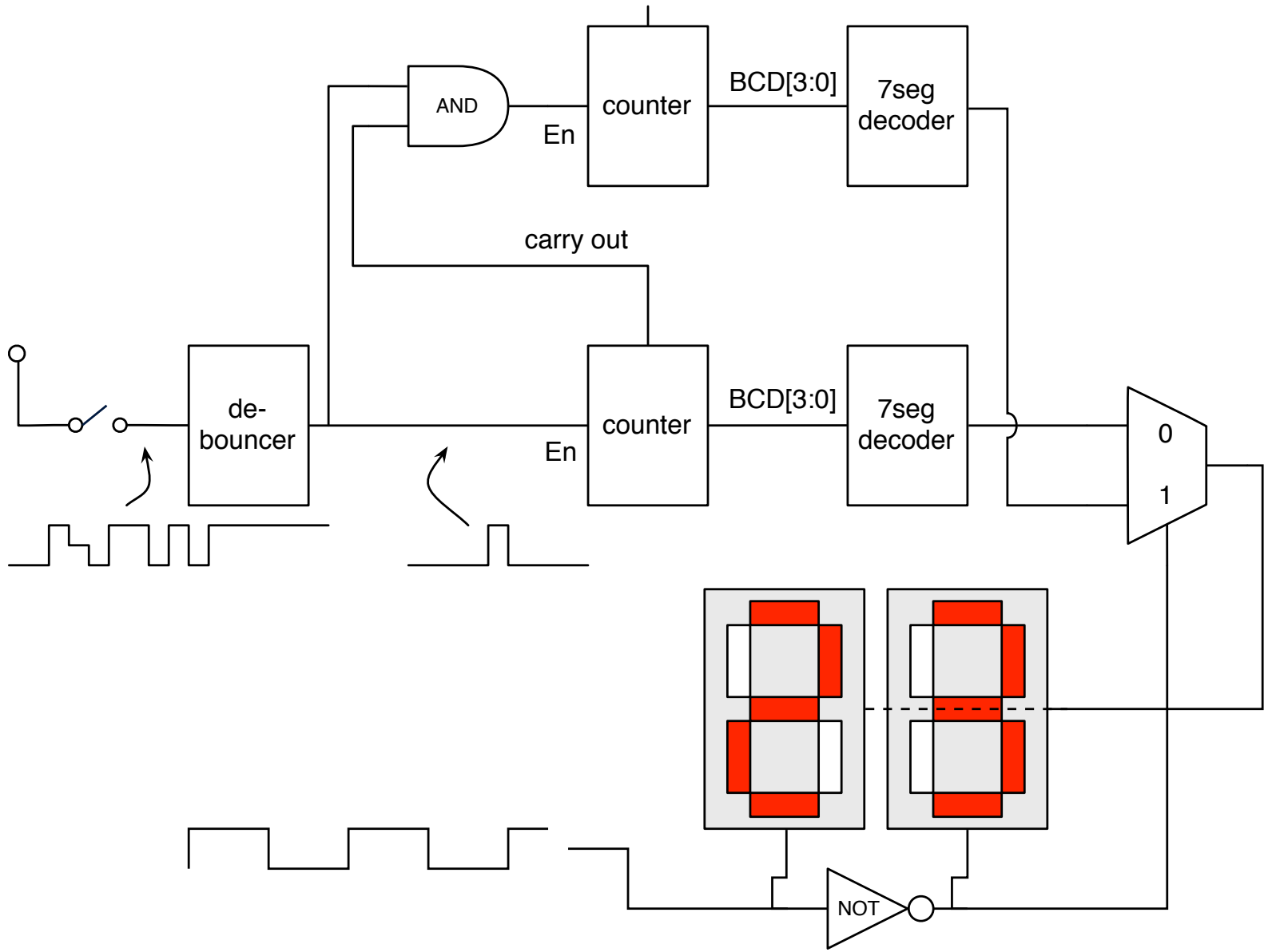
We can put the two together to make the S3 board count to 100, ZOMGZ!

To make it more interesting (!), we'll have two counters for units & tens, and a carry between them.

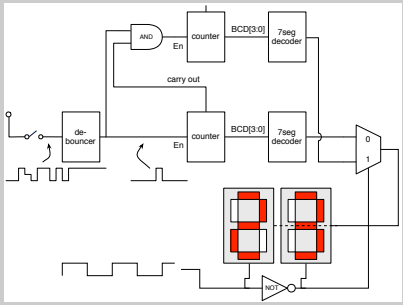
Other features:

- The 7-segment LED displays are *multiplexed* on the S3 board, so we have to alternate between digits quickly.
- Debounce a button input, to count just once when it's pressed.

Counter thingy



Counter thingy



1. Clock/reset are not shown on the diagram
2. Button input is all over the place; the debouncer creates a pulse one clock wide when the button is pressed, enabling the counters
3. When the bottom counter gets to 9, its 'carry out' output goes to 1 signalling that next time it's enabled & clocked it will roll over.
4. If the set of counters is enabled and carryout is present, the upper counter is enabled to 'catch' the roll-over from the bottom counter.
5. The LED displays are driven by simply switching the output between digits 0 and 1 (whilst enabling digit 0 or 1) fairly quickly. POV.

Counter thingy (2)

See `sources/top_level.v` and simulation with `tb/tb_toplevel.v`

Then, we can compile with ISE and download via JTAG to the board. Then enjoy the new toy.

Links

- Iverilog simulator: `http://iverilog.icarus.com/`
- GTKWave viewer for sim traces: `http://gtkwave.sourceforge.net/`
- Xilinx s/w (free reg. required): `http://www.xilinx.com/support/download/index.htm`
- Opencores, "sourceforge for hardware": `http://opencores.org/`
- FPGA arcade machine builds: `http://fpgaarcade.com/`
- Projects, tutorials: `http://fpga4fun.com/`