DIY soft-core uP
Microprocessor design using an FPGA
"made simple"

Jim Brakefield

Introduction

- FPGAs: digital circuit with its resources connected by programmable/configurable wiring
- uP: mechanism to sequentially execute instructions (at high speed). Has addressable memory and IO.
- Soft-core uP: uP implemented using FPGA resources. Written in VHDL or Verilog.

Table of Contents

Caveats

- Presentation may conflict with your course material?
- Am not a professor or even a career FPGA designer
- This talk summarizes how I have learned to go at the problem efficiently
- Computer architecture is one of my hobbies

Why design & implement a microprocessor from scratch?

- Useful skill set
  - Broaden your design capability from simple state machines to high performance uP
- Term project
  - Talk directed towards fast start & clear path
- Architectural exploration
  - Instructions & addressing modes of your choosing
- High performance real-time

FPGA resources

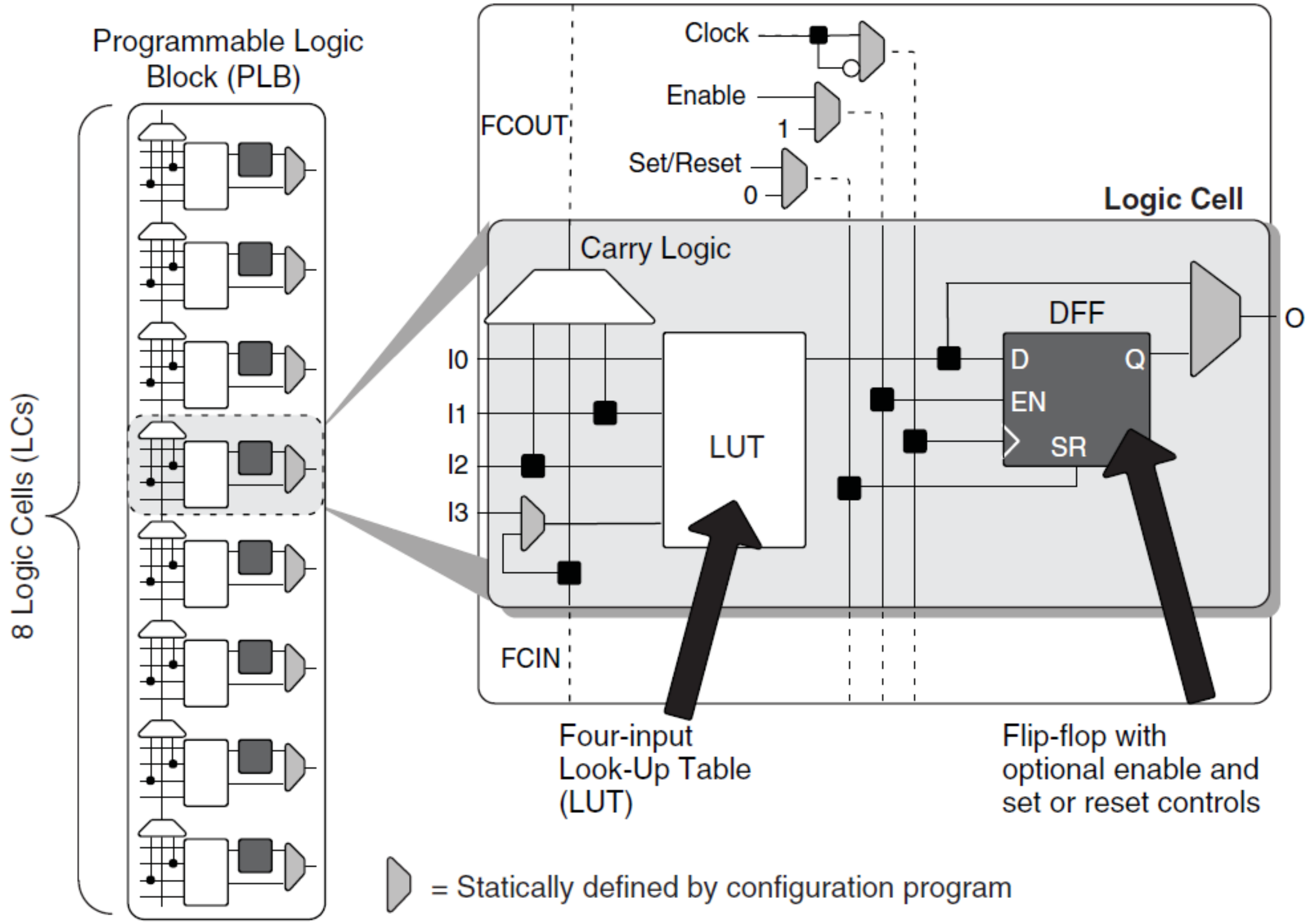
- LUT Lookup table 3 to 6 inputs, 1 to 2 outputs
  - Include carry chain for building adders
- DFF D flip-flop 1 to 2 per LUT
- IO Tristate transceiver with optional DFFs
- RAM Variable aspect ratio, usually dual port
  - LUT RAM: 16x1, 64x1
  - Small block RAM: 32x18, 32x20, 64x18
  - Block RAM: 128x36 to 1024x36
  - Large block RAM: 2Kx72, 4Kx72, 4Kx144
  - External DRAM: Built-in controller
- Wiring Fabric
  - Horizontal and vertical wire segments of various lengths

Additional FPGA resources not used herein

- PLL/DLL Generation of additional clock frequencies
- Differential IO High speed interfaces
- SERDES High speed serial IO (gigabits/sec)
- Hard core uP ARM Cortex M3, R5, A9, A53
- Vendor soft cores (32-bit, full tool chains)
  - Altera NIOS II
  - Xilinx microBlaze
- www.opencores.org processor soft cores

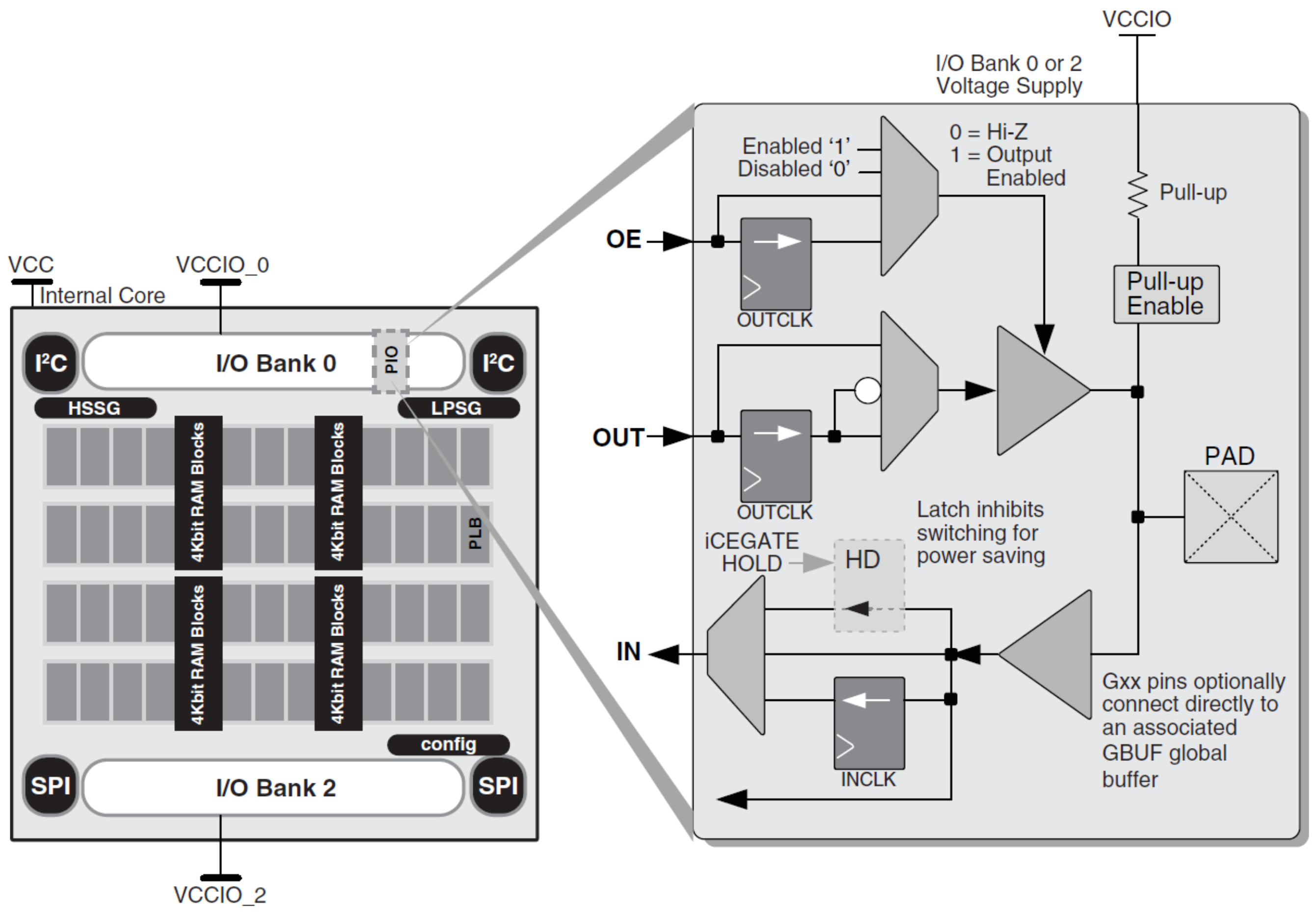
LUT + DFF

Lattice Semiconductor iCE40FamilyHandbook.pdf pg6-2


Programmable Logic Block (PLB)
8 Logic Cells (LCs)
FCOUT
Clock
Enable
1
Set/Reset
0
Logic Cell
Carry Logic
DFF
I0
I1
I2
I3
LUT
D
Q
EN
SR
O
FCIN
Four-input Look-Up Table (LUT)
Flip-flop with optional enable and set or reset controls
= Statically defined by configuration program

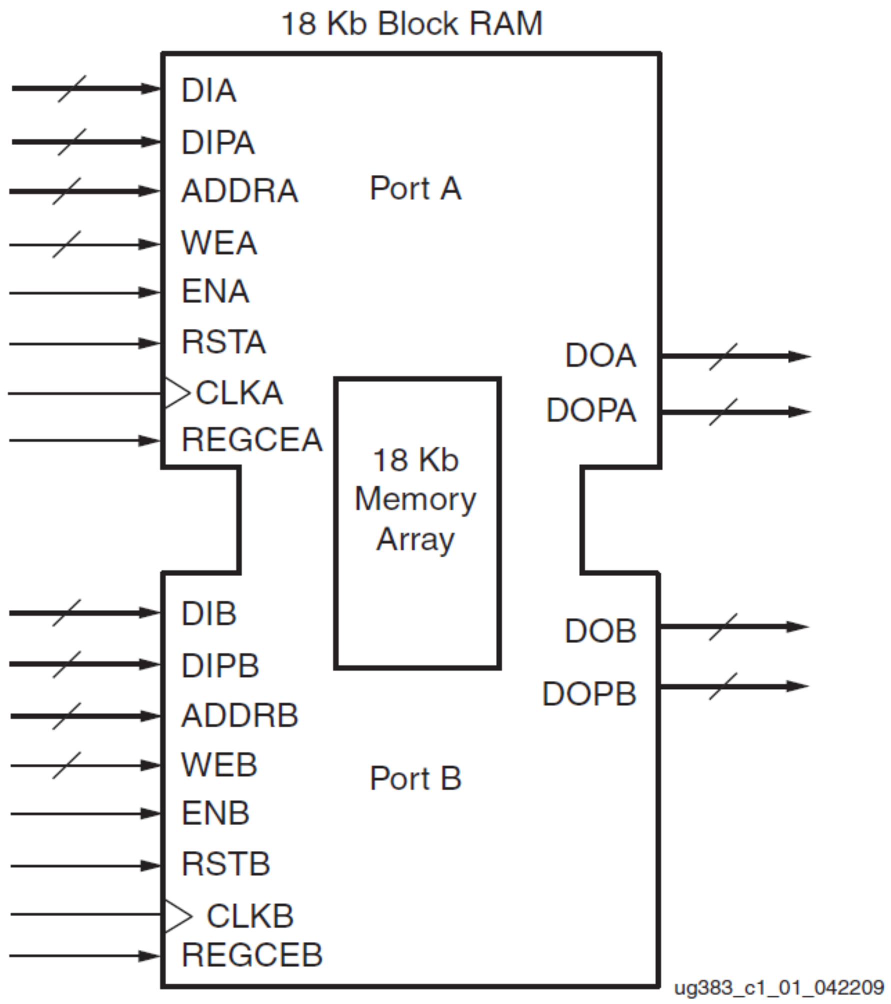
Input – Output pins

Lattice Semiconductor iCE40FamilyHandbook.pdf pg6-7



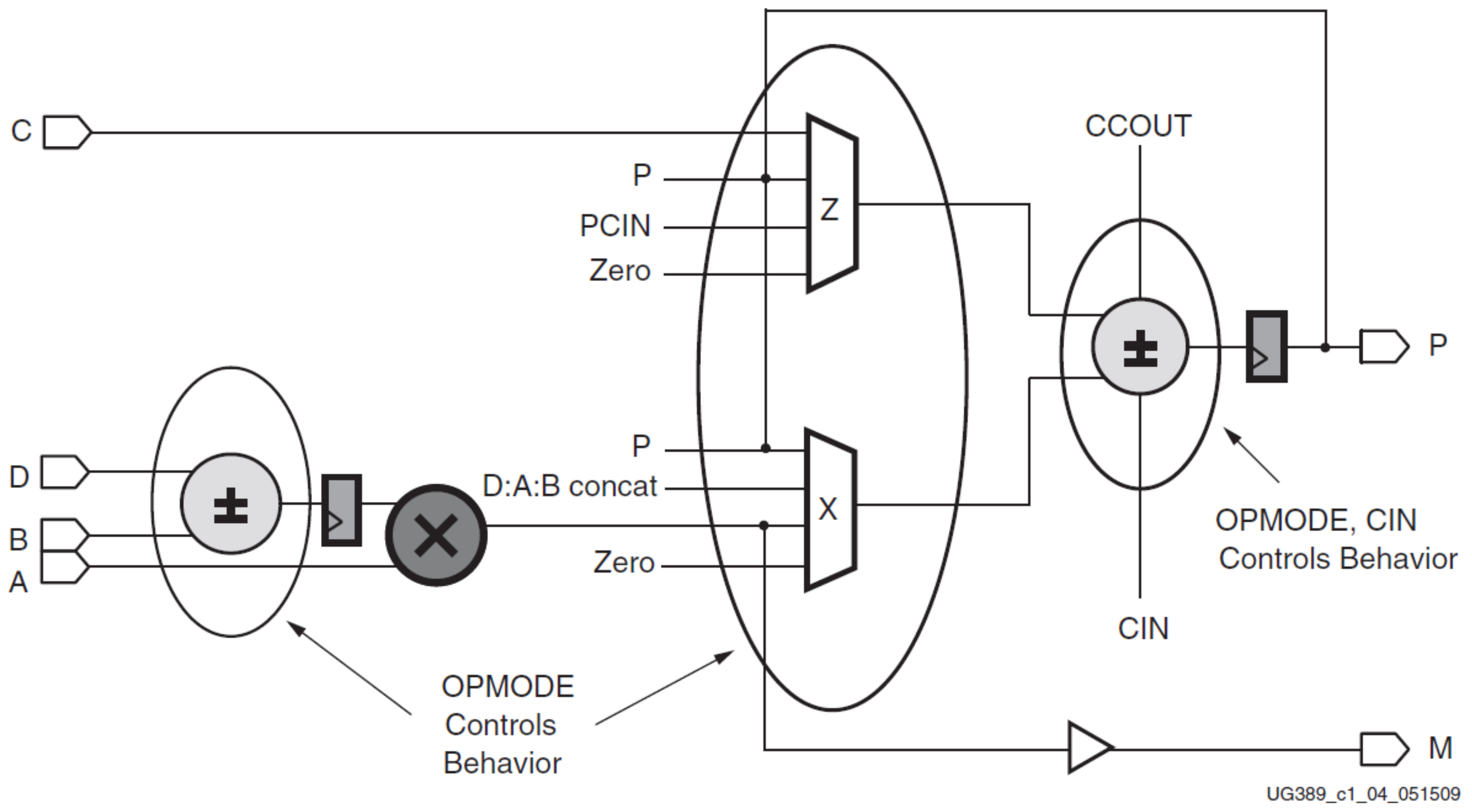
Block RAM

Xilinx Spartan-6 FPGA Block RAM Resources User Guide pg12



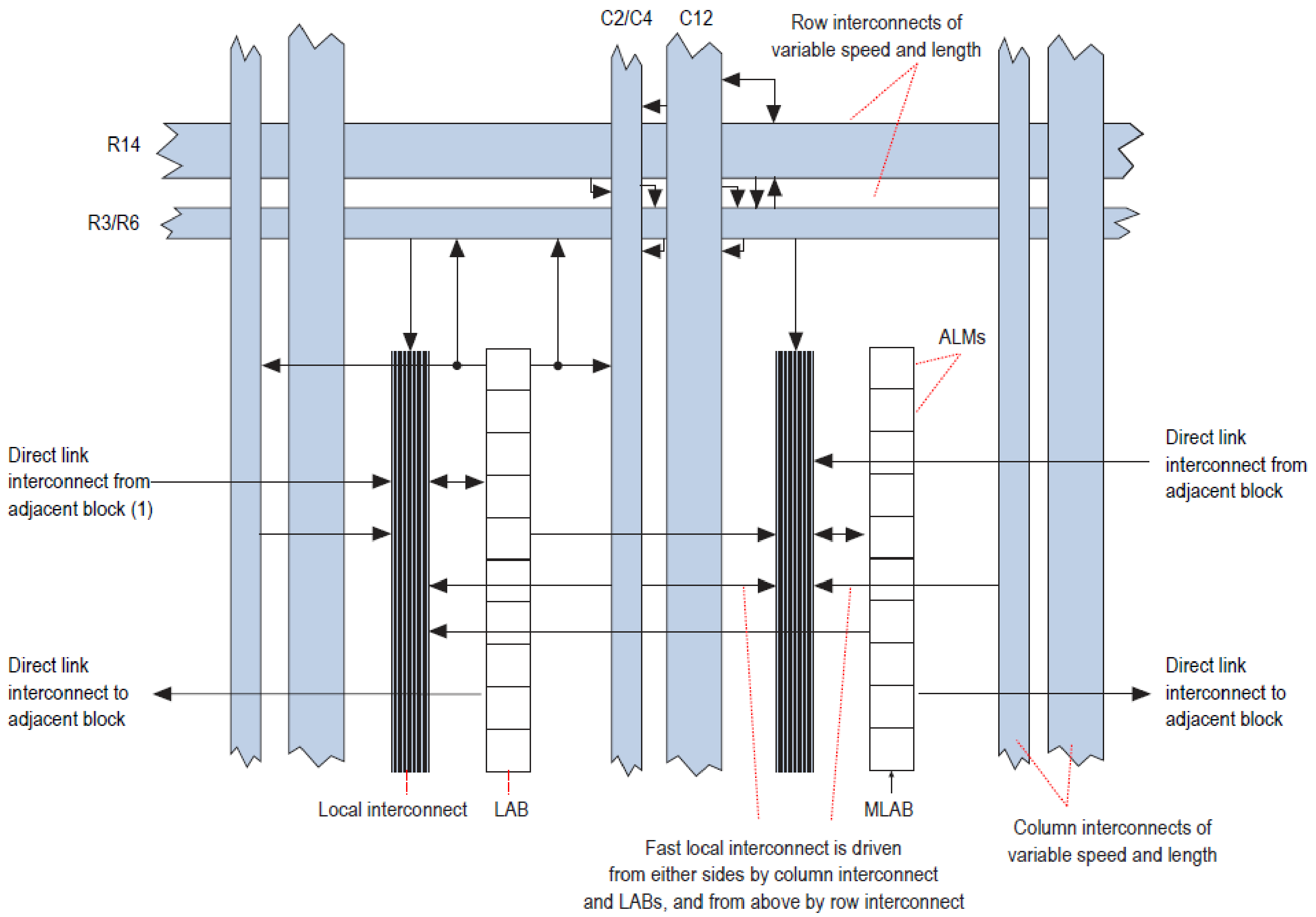
Simplified DSP48A1 slice

Xilinx Spartan-6 FPGA DSP48A1 Slice pg 17



Wiring Fabric

Altera CycloneV Device Handbook pg1-2


C2/C4
C12
Row interconnects of variable speed and length
R14
R3/R6
ALMs
Direct link interconnect from adjacent block (1)
Direct link interconnect from adjacent block
Direct link interconnect to adjacent block
Direct link interconnect to adjacent block
Local interconnect
LAB
MLAB
Column interconnects of variable speed and length
Fast local interconnect is driven from either sides by column interconnect and LABs, and from above by row interconnect

FPGA Vendors

Name	web site	market share	revenue
Xilinx	Xilinx.com	51%	$2.4B
Intel	Altera.com	34%	$1.9B
Lattice Semiconductor	Latticesemi.com	~7%	$0.4B
Microsemi	Actel.com	~7%	$0.4B?
Cypress	Cypress.com	?	?

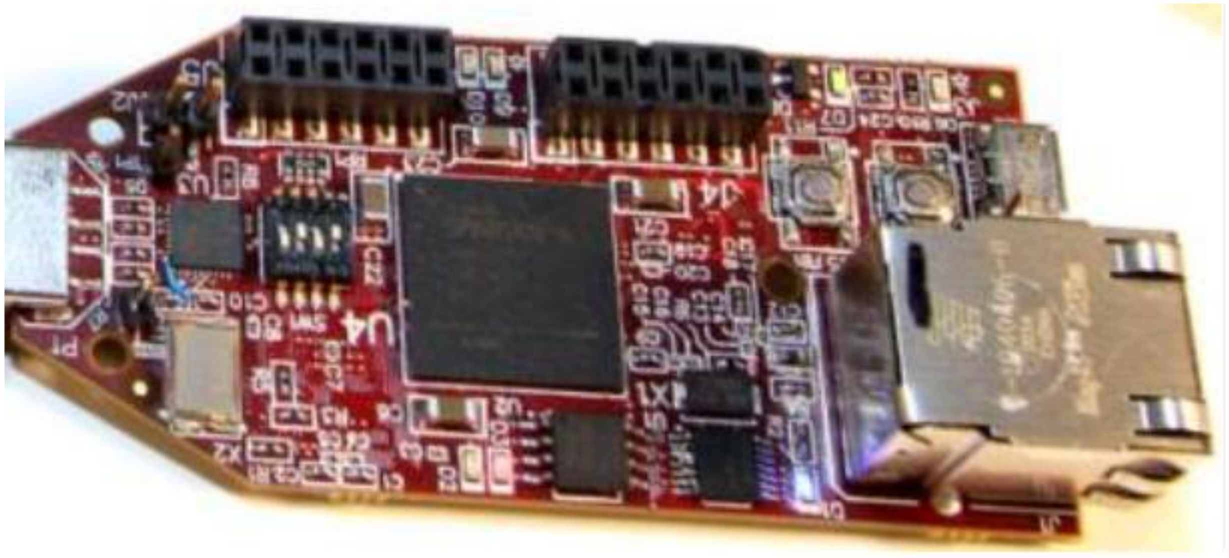
FPGA Tools

- HDL (Hardware Design Languages)
  - VHDL, Verilog, System Verilog
  - High Level Synthesis(HLS): C, C++, Matlab

Compilation	HDL into “gates” & DFF
Place & Route	Vendor proprietary
Timing analysis	How fast will it run?
Performance analysis	LUT count & Fmax
Simulation	Unit testing on a PC
Download	Transfer bit file to the chip

Spartan-6 resources (XC6LX9-3CSG324)

- Avnet Spartan-6 FPGA LX9 Micro-Board
- 4 LEDs, 4 switches, 1 push button, 2 PMOD (16 IO)
- Ethernet jack, 64MB DRAM, boot flash
- Power & download via USB cable
- 6K 6LUTs, 11K Dffs, 64 8K block ram, 200 IOs
- 16 18x18 MUL/DSP, 2 PLL, no SERDES
- 100MHz clock

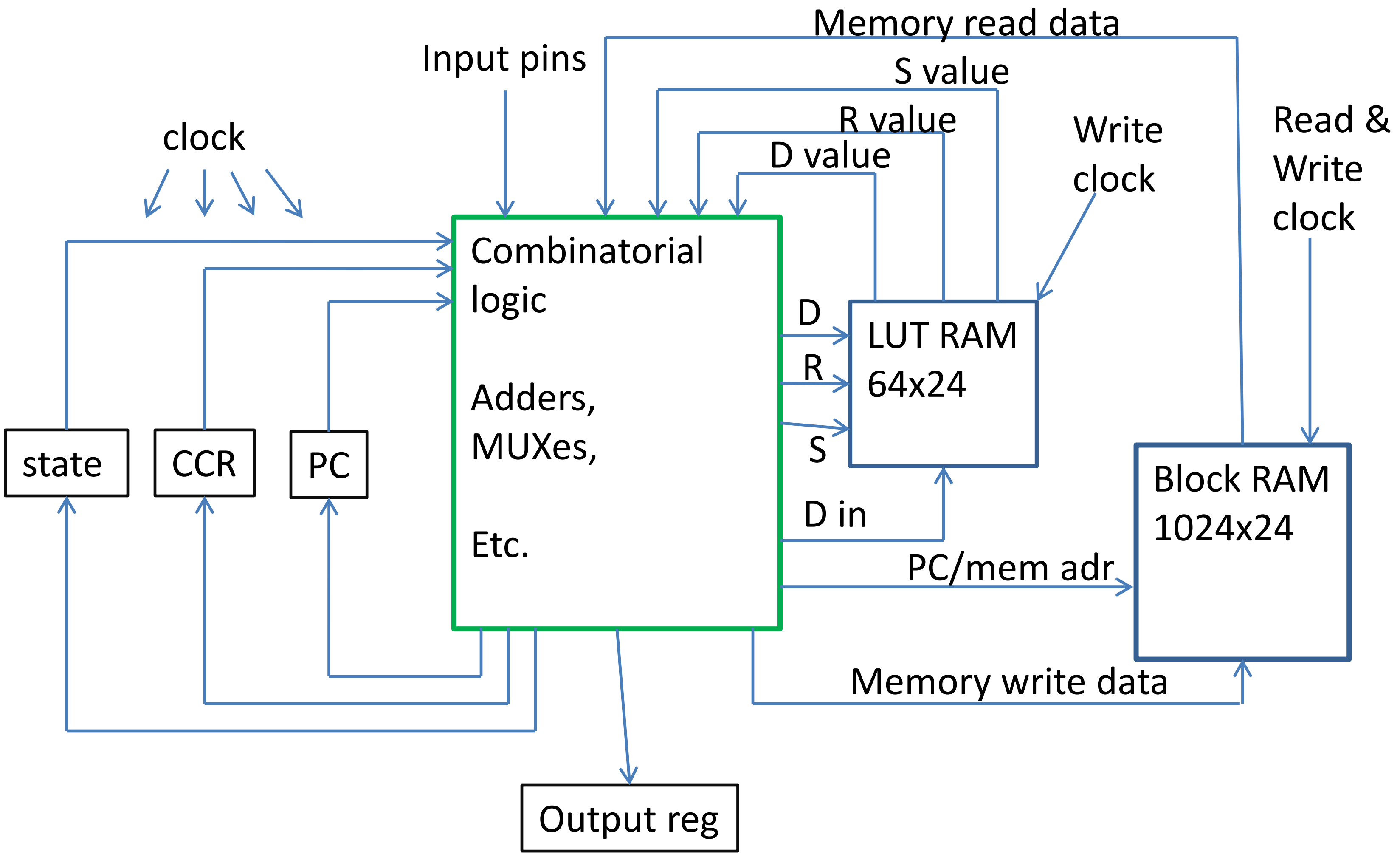


ROIS24_24uP

- 64 24-bit registers: uses 96 LUTs, quad ported
- 24 bit instruction with 6-bit op-code and three 6-bit register designators
  - XXXXXX DDDDDD RRRRRR SSSSSS (DRS)
  - XXXXXX DDDDDD RRRRRR sNNNNN (DRsN)
  - XXXXXX DDDDDD sNNNNN NNNNNN (DsNN)
  - XXXXXX sNNNNN NNNNNN NNNNNN (sNNN)
- 24-bit by 1024 word block RAM main memory
- IO ports directly connected to LEDs, Switches, Push button, 100MHz clock

rois24_24uP block diagram

write enables on all registers and RAM



Clock cycle events

- Read instruction out of block RAM
- Use R & S to read operands from LUT RAM
- Generate control signals
- Do the arithmetic
- Select appropriate result
- Set write enables
- Update PC, CCR, LUT RAM, post address (new PC) to block RAM

rois24_24uP instruction set

- Register zero always reads as zero
- <u>XXXXXX DDDDDD RRRRRRR SSSSSSS </u>(DRS)
  - Two operands & result registers
  - Add, add with carry, subtract, subtract with carry,
  - AND, ANDC(2nd operand complemented), OR, XOR
  - Call (Jump if D=0; branch to mem[R+S])
- <u>XXXXXX DDDDDD RRRRRR sNNNNN </u>(DRsN)
  - Sign extended 6-bit immediate combined with R value
  - Addi, adci, andi, ori, xori
  - Call (Jump if D=0; absolute call/jmp if R=0; return if sN=0)
  - In, Out

rois24_24uP instruction set cont’d

- Load and store: DRS & DRsN
  - Load to D, Store from D
- XXXXXX DDDDDD sNNNNN NNNNNN (DsNN)
  - Conditional branch relative (D is condition code)
  - Load immediate
  - Call relative
- XXXXXX sNNNNN NNNNNN NNNNNN (sNNN)
  - Prefix: sNNN will be prefixed to next sN inst
- 28 instructions currently, room for 64 inst.

The difficulties

- FPGA Complexity
  - FPGAs have a lot of features
  - Each family is different
- VHDL/Verilog Complexity
  - Must code with FPGA primitives in mind
  - HLS doesn’t handle control logic well
- uP Complexity
  - Large instruction sets, high performance

Crunch time

- Devise an instruction set
  - Text & spreadsheet versions
- Define file with op-code mnemonics
  - Lets one change op-code encoding easily
- Find a minimal initial set of instructions
  - Write a short program for blinking LEDs
- Place program into case statement
- Implement each instruction as a line in another case statement

Rois24_24uP minimal program

- At 100MHz need more than 24-bits to get blinking lights
- Use two registers:
  - Add one to first register
  - Add carry to second register
  - Output second register to LEDs
  - Branch back to first instruction
- Four instruction loop
  - 2nd register increments every 0.67 seconds

Op-code encoding

```
constant op_ADD                  : std_logic_vector(5 downto 0) :=
"100000";      -- R + S => D
constant op_SUB                  : std_logic_vector(5 downto 0) :=
"100001";      -- R - S => D
constant op_ADC                  : std_logic_vector(5 downto 0) :=
"100010";      -- R + S + carry => D
constant op_SBC                  : std_logic_vector(5 downto 0) :=
"100011";      -- R - S + carry => D
constant op_AND                  : std_logic_vector(5 downto 0) :=
"100100";      -- R and S => D
constant op_ANDC                 : std_logic_vector(5 downto 0) :=
"100101";      -- R and not S => D
. . .
```

The program case statement

```
program_ROM: process(pc)
Begin
case pc(3 downto 0) is
--       location              op-code       D reg          R reg          sN or S reg
when "0000" => inst <= op_ADDI   & "000001" & "000001" & "000001";
when "0001" => inst <= op_ADCI    & "000010" & "000010" & "000000";
when "0010" => inst <= op_OUTsN & "000010" & "000000" & "000000";
when "0011" => inst <= op_JMPsN & "000000" & "000000" & "000000";
when others => inst <=                (others => '0');
end case;
end process;
```

Instruction evaluation case statement

```
--	instruction decode and implementation
decode: process(inst, pc, sN, R, RR, SS, CCR, Dloc,opcode,aluout)
begin
--	default signal values
pcN<=PC+1;
ALUout<=(others => '0');
LUTwe<='0'; CCRwe<='0'; outwe<='0';
RR<= '0'&R;	-- need one additional bit so can save carry out
SS<= '0'&sN;	-- the program only uses the DRsN mode
```

Instruction evaluation case statement cont'd

```
--  instruction implementation
-- (for each instruction specify non-default signal values)
case opcode is
   when op_JMPsN => ALUout<=RR+SS;         pcN<=ALUout(23 downto 0);
   when op_ADDI  =>   ALUout<=RR+SS;         LUTwe<='1'; CCRwe<='1';
   when op_ADCI  =>   ALUout<=RR+SS+CCR(24); LUTwe<='1'; CCRwe<='1';
   when op_OUTsN => ALUout<=RR+SS;          outwe<='1';
   when others => pcN<=pc;
               -- effectively a branch to itself, eg HALT
end case;
```

Instruction evaluation case statement cont’d

```
--      prohibit writes to register zero
if Dloc = "000000" then LUTwe<='0'; end if;
end process;

--      connect result to register file write port
Din     <= ALUout(data_size-1 downto 0);
--      condition code register is copy of ALU result
CCRN    <= "000" & ALUout;          -- no overflow for now
```

Register update process

```
update: process(clk)
begin
if (rising_edge(clk)) then
    pc<=pcN;              -- always update the PC
    if CCRwe = '1' then CCR<=CCRN; end if;
    if outwe = '1' then out0<=Dout; end if;
--  LUTwe does its enable at the LUT RAM
end if;
end process;
```

Is it working?

- Constraint file
  - Sets clock speed
  - Sets IO pin assignment
  - Evaluation kit usually has a sample constraint file
- Simulate/debug
  - Tools will generate a simple test bench
  - Need to relay observed signals out to test bench
  - Program acts as the test script
- Performance metrics & goals
  - Track LUT count and Fmax

Some results & experiments

Directory name	# of inst	Fmax MHz	KHz per LUT	LUT count	MUXCY count	Comments	Data path	Block RAM
rois24_24up_s6_noram	4	109	590	184	56	basic "Hello World": blinking LEDs	No	No
rois24_24up_s6_dpnoram	4	142	1043	136	32	experiment to test simple data path	Yes	No
rois24_24up_s6_bram	25	**83**	74	**1119**	264	all inst except PFX, IN & shifts; Hello World only	No	Yes
rois24_24up_s6_dpbram	28	105	206	512	52	all inst; experiment to test full set of data paths	Yes	Yes
rois24_24up_s6_dpbram	28	106	375	**283**	52	area mode, high effort	Yes	Yes
rois24_24up_k7_dpbram	28	176	471	373		determine best Fmax & KHz/LUT	Yes	Yes
rois24_24up_k7_dpbram	28	143	506	282		area mode, high effort	Yes	Yes

Moving forward

- Migrate to a data path?
- Getting block RAM running
- Add instructions
- Migrate to more pipeline stages?
  - Not that big of a gain!
- Adding modalities
  - Addressing modes
  - Whatever fits into your schedule/interests

Advanced features

- Floating-point: figure at least 2K LUTs
- External RAM:
  - Use block RAM for Caches
  - Use vendor’s DDR interface
- Writing an assembler
- Writing a compiler
- Adding peripherals (see www.opencores.org)
- Barrel processors
- Multiple dispatch

Insights

- A deliberate process:
- Get something bare bones working
  - Minimum instruction set, minimum program
- Add instructions
- Add to test program
- Migrate to block RAM and data-path?
  - Data-path logic is harder to debug
- Lots of soft-core uPs at www.opencores.org

My next steps

- Expand program with tests for each inst.
- Get the block RAM/data path version debugged
- Do two+ stage pipeline
- Add multiply, shift & floating-point instructions
- Do a variation with five-bit register designators
  - Two instruction flag bits: CCR update & return
  - Stack like usage of register file
- Do 12, 16, 32 and 48-bit data size versions
- Do 12/16-bit instruction format: D & R partially implied

FPGA evaluation kits

all use USB download, check frequently for new boards

Vendor	Price	FPGA	“LUTs”	Mults	Comments
Cypress CY8CKIT-59	**$10**	Cypress PSoC5	384	0	ARM Cortex M3, digital & analog, schematic editor, not a true FPGA
Arrow BeMicro MAX10	**$30**	Altera Max 10M08	8K	24	oscillator, 12-bit A2D, flash, DRAM
Arrow BeMicroCV	**$49**	Altera Cyclone V	25K	50	Some versions of Cyclone V have (2) Cortex A9: DE0-Nano-SoC $104
Arrow SmartFusion Kick Start	**$60**	Actel SmartFusion2	12K	22	Cortex M3, security & reliability features
XESS XuLA2-LX9	**$69**	Xilinx Spartan-6	9K	16	40-pin DIP, SDRAM (might want to use $99 “Arty” Atrix-7 instead)
Digilent ZYBO(student $)	**$125**	Xilinx Zynq 7010	28K	80	(2) Cortex A9, 512MB, HDMI…
Adapteva Parallella-16	**$149**	Xilinx Zynq 7010	28K	80	(2) Cortex A9, 16-core uP

Videos & Training

- www.YouTube.com: hundreds of videos
- Xilinx

http://www.xilinx.com/training/free-video-courses.htm

http://www.xilinx.com/support/university.html

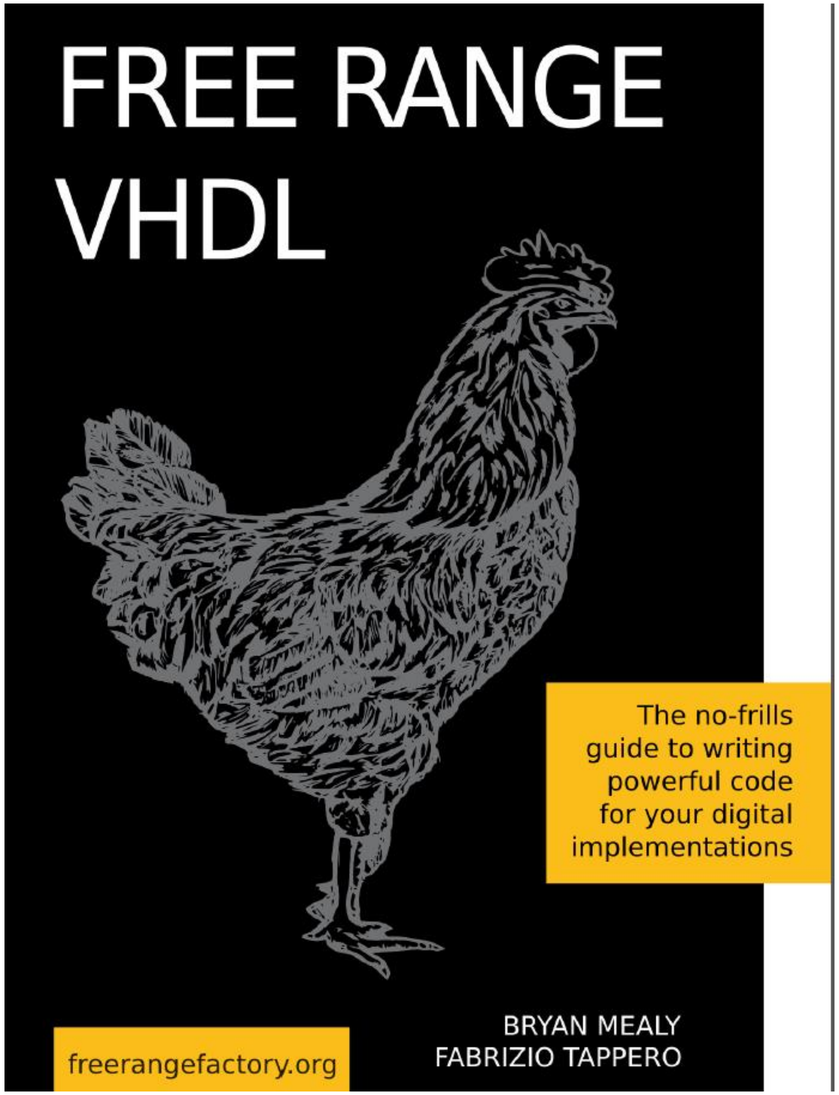
- Altera

https://www.altera.com/support/training/university/overview.html

- Altium

https://altiumvideos.live.altium.com/

Free Range VHDL


FREE RANGE VHDL

The no-frills guide to writing powerful code for your digital implementations

BRYAN MEALY
FABRIZIO TAPPERO

freerangefactory.org

Guidelines & PDFs

- "Fundamental mode": Single clock, no latches
- Two process VHDL:
  - Distinct combinational and sequential processes
  - Jiri Gaisler 2014: **A structured VHDL design method**
- VHDL, Verilog and System Verilog **Quick Reference Cards**
- Crockett etal 2014:

  The Zynq Book: ARM Cortex A9 + FPGA
- Mealy & Tappero 2012:

  Free Range VHDL: A to-the-point VHDL text

Books

- Harris & Harris 2013: *Digital Design and Computer Architecture, 2nd ed.*

  VHDL & System Verilog, all the way from 0s & 1s to x86

  Shows trade-off of performance versus pipe length
- Max Maxfield 2004: *The Design Warrior's Guide to FPGAs*

  Folksy, good coverage at the chip level
- Nazeih Botros 2006: *HDL Programming Fundamentals – VHDL and Verilog*

  Side by side VHDL & Verilog
- Peter Ashenden 2008: *The Designer's Guide to VHDL, 3rd ed.*

  Thee VHDL reference
- Steve Kilts 2007: *Advanced FPGA Design*