Vintage Computing with FPGAs

Stephen A. Edwards

VCF East, May, 2018

MOS Technology KIM-1
C. 1976

Altera Cyclone II FPGA
C. 2004
void mos6502::Op_ADC(uint16_t src)
{
    uint8_t m = Read(src);
    unsigned int tmp = m + A + (IF_CARRY() ? 1 : 0);
    SET_ZERO(!(tmp & 0xFF));
    if (IF_DECIMAL()) {
        if (((A & 0xF) + (m & 0xF) + (IF_CARRY() ? 1 : 0)) > 9)
            tmp += 6;
        SET_NEGATIVE(tmp & 0x80);
        SET_OVERFLOW(!((A ^ m) & 0x80) && ((A ^ tmp) & 0x80));
        if (tmp > 0x99)
            tmp += 96;
        SET_CARRY(tmp > 0x99);
    } else {
        SET_NEGATIVE(tmp & 0x80);
        SET_OVERFLOW(!((A ^ m) & 0x80) && ((A ^ tmp) & 0x80));
        SET_CARRY(tmp > 0xFF);
    }
    A = tmp & 0xFF;
}

Source: https://github.com/gianlucag/mos6502

Emulation software tricks original software into thinking it is running on original hardware
What is an FPGA?

A Field Programmable Gate Array:
A configurable circuit; not a stored-program computer

LUT: 16-element lookup table

Source: http://evergreen.loyola.edu/dhhoe/www/HoeResearchFPGA.htm
Using a LUT to Implement a Circuit

If the circuit has fewer than 4 inputs, write the truth table and load it in the LUT:

<table>
<thead>
<tr>
<th>M</th>
<th>L</th>
<th>R</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Pong, Atari, 1972
The Main FPGA Players

formerly known as

intel® FPGA

ALTERA

XILINX®

Classic duopoloy, good for customers: neck-and-neck technology
Altera Cyclone II
EP2C5T144
c. 2004
Very-low end; completely obsolete
Dirt cheap: $16 board
“Only” 4608 LEs
14 KB memory
144 pins (89 I/O)
Minimal EP2C5T144 Development Board

- Altera FPGA Cyclone II
- EP2C5T144 Minimum System Board Development Board
- $16 on Banggood.com. Also Amazon, eBay, dx.com, AliExpress, etc.

- 50 MHz oscillator
- JTAG programming connector
- EPCS4SI8 4 Mb serial configuration flash
- Active Serial connector for programming
- +5V power jack
- 3.3V (I/O) and 1.2V (core) voltage regulators
- 3 LEDs, 1 pushbutton switch
The Design Flow

```systemverilog
module comb1(
    input logic [3:0] a, b,
    input logic s,
    output logic [3:0] y);

    always_comb
        if (s)
            y = a + b;
        else
            y = a & b;

endmodule
```

System Verilog
// Full adder
module full_adder(input logic a, b, c,
                 output logic sum, carry);

assign sum = a ^ b ^ c;
assign carry = a & b | a & c | b & c;
endmodule
An XOR Built Hierarchically

```verilog
module mynand2(input logic a, b,
               output logic y);
    assign y = ~(a & b);
endmodule

module myxor2(input logic a, b,
               output logic y);
    logic abn, aa, bb;

    mynand2 n1(a, b, abn),
            n2(a, abn, aa),
            n3(abn, b, bb),
            n4(aa, bb, y);
endmodule
```

Declare internal wires

n1: A mynand2 connected to a, b, and abn
A Decimal-to-Seven-Segment Decoder

always_comb: combinational logic in an imperative style

Multiway conditional

4’d5: decimal “5” as a four-bit binary number

Mandatory

"blocking assignment": use in always_comb

module dec7seg(input logic [3:0] a, output logic [6:0] y);

always_comb

```verilog
    case (a)
      4’d0: y = 7’b111_1110;
      4’d1: y = 7’b011_0000;
      4’d2: y = 7’b110_1101;
      4’d3: y = 7’b111_1001;
      4’d4: y = 7’b011_0011;
      4’d5: y = 7’b101_1011;
      4’d6: y = 7’b101_1111;
      4’d7: y = 7’b111_0000;
      4’d8: y = 7’b111_1111;
      4’d9: y = 7’b111_0011;
    default: y = 7’b000_0000;
    endcase
```
endmodule
module comb1(
    input logic [3:0] a, b,
    input logic s,
    output logic [3:0] y);

    always_comb
        if (s)
            y = a + b;
        else
            y = a & b;
    endmodule

Both $a + b$ and $a \& b$ computed, mux selects the result.
module adecode(input logic [15:0] address, 
               output logic RAM, ROM, 
               output logic VIDEO, IO);

always_comb begin
    {RAM, ROM, VIDEO, IO} = 4'b 0;
    if (address[15])
        RAM = 1;
    else if (address[14:13] == 2'b 00 )
        VIDEO = 1;
    else if (address[14:12] == 3'b 101)
        IO = 1;
    else if (address[14:13] == 2'b 11 )
        ROM = 1;
end
endmodule
A D-Flip-Flop

```verilog
module mydff(input logic clk, input logic d, output logic q);
  always_ff @(posedge clk)
  begin
    q <= d;
  end
endmodule
```

- **always_ff** introduces sequential logic
- Triggered by the rising edge of **clk**
- Copy **d** to **q**
- Non-blocking assignment: happens “just after” the rising edge of **clk**

**Diagram:**
- **q~reg0**
- **d** and **clk** inputs
- **D**, **Q**, and **q** outputs
module count4(input logic clk, output logic [3:0] count);

always_ff @(posedge clk)
  count <= count + 4'd 1;
endmodule
“Lite”/“Web edition” free good enough for our use
Quartus 13.0sp1 last to support the Cyclone II
Quartus 13.0sp1 on Ubuntu 16.04, 64 bit

Installer is a 32-bit binary; needs 32-bit libraries

# sudo apt install -y lib32z1
# sudo ./setup.sh

Install to /opt/altera/13.0sp1

Set up environment:

# PATH=$PATH:/opt/altera/13.0sp1/quartus/bin
# quartus --64bit
Ubuntu 16.04 USB Blaster
Permissions

Nothing today doesn’t require a programming dongle; JTAG, active serial

`# lsusb
Bus 001 Device 087: ID 09fb:6001 Altera Blaster`

Create the file `/etc/udev/rules.d/51-altera.rules`

`ATTR{idVendor}=="09fb", ATTR{idProduct}=="6001", MODE="0666"`

Fixes “Unable to lock chain (Insufficient port permissions)” errors from jtagconfig
“Hello World” for an FPGA

hello.qpf  Project file; mostly the name
hello.qsf  Settings file: device type, pin assignments
hello.sdc  Timing constraints: 50 MHz clock
hello.sv   System Verilog specification of “guts”

Use the GUI or run at the command line:

quartus_sh --64bit --flow compile hello
quartus_pgm -c 1 -m as -o "pv;hello.pof"
# hello.qpf
QUARTUS_VERSION = "13.0"
DATE = "11:17:07  May 17, 2018"
PROJECT_REVISION = "hello"

# hello.qsf
set_global_assignment -name FAMILY "Cyclone II"
set_global_assignment -name DEVICE EP2C5T144C8
set_global_assignment -name PROJECT_CREATION_TIME_DATE "11:17:07  MAY 17, 2018"
set_global_assignment -name ORIGINAL_QUARTUS_VERSION "13.0 SP1"
set_global_assignment -name LAST_QUARTUS_VERSION "13.0 SP1"
set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
set_global_assignment -name MAX_CORE_JUNCTION_TEMP 85
set_global_assignment -name TOP_LEVEL_ENTITY hello
set_global_assignment -name SYSTEMVERILOG_FILE hello.sv
set_location_assignment PIN_3 -to LED[0]
set_location_assignment PIN_7 -to LED[1]
set_location_assignment PIN_9 -to LED[2]
set_location_assignment PIN_17 -to clk50
set_location_assignment PIN_144 -to KEY
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to KEY

# hello.sdc
create_clock -name "clk50" -period "50 MHz" [get_ports {clk50}]
derive_pll_clocks -create_base_clocks
derive_clock_uncertainty
module hello(input clk50,
             input KEY,
             output [2:0] LED);

logic [26:0] count;
always_ff @(posedge clk50)
count <= count + (KEY ? 27'd1 : 27'd4); // KEY = 0 when pressed

assign LED = ~count[26:24];
endmodule
The KIM-1

MOS Technology, 1976

1 MHz MCS6502 processor

2 MCS6530 “RIOT” chips:
1K ROM + 64B RAM + GPIO + timer

1K Static RAM (8 × 6102)

24-key hexadecimal keyboard

6 7-segment LEDs

Serial port:
20 mA current loop

Cassette interface
Need SystemVerilog for these blocks

- **6502**: steal
- **1K RAM**: write
- **6530**: write
- **Keyboard**: build
- **Display**: build
- **TTY**: emulate
- **Audio**: ignore

Source: KIM-1 User Manual
Which 6502 Module To Use?

Coding and verifying even an 8-bit processor is not trivial

Fortunately, others have done this. I found four in Verilog:

- Thomas Skibo

- Arlet Ottens
  https://github.com/Arlet/verilog-6502

- Oleg Odintsov
  http://opencores.org/project,ag_6502

- Rob Finch http://finitron.ca/Projects/Prj6502/bc6502_page.html

I selected Arlet Ottens’: simple and designed for FPGAs. OpenCores core complicated & modeled the 6502’s two-phase clock.
module KIM_1(input clk, 
           input reset, 
           input NMI, 
           output [15:0] AB, 
           output [7:0] DO, 
           output [7:0] DI, 
           output WE, 
           output RDY, 

           // .... 
           );

mcs6502 U1( .clk( clk ), 
            .reset( reset ), 
            .AB( AB ), 
            .DI( DI ), 
            .DO( DO ), 
            .WE( WE ), 
            .IRQ( 1'b0 ), 
            .NMI( NMI ), 
            .RDY( RDY ) );
Debounced RS (reset) and ST (stop/NMI) keys

Take from environment; assume good signals

Address decoder

Easily coded

1 MHz crystal

Derive from 50 MHz

Single-step (SST) switch and logic

6502 module doesn’t have sync; omitted

Source: KIM-1 User Manual
module SN74145(input [3:0] select, output [9:0] out);
    always_comb
    case (select)
        4'd0: out = 10'b11_1111_1110;
        4'd1: out = 10'b11_1111_1101;
        4'd2: out = 10'b11_1111_1011;
        4'd3: out = 10'b11_1111_0111;
        4'd4: out = 10'b11_1110_1111;
        4'd5: out = 10'b11_1101_1111;
        4'd6: out = 10'b11_1011_1111;
        4'd7: out = 10'b11_0111_1111;
        4'd8: out = 10'b10_1111_1111;
        4'd9: out = 10'b01_1111_1111;
        default: out = 10'b11_1111_1111;
    endcase
endmodule

output [7:0] K // Active-low address decoder output

logic [9:0] u4out;
SN74145 U4(.select ( {DECODE_ENABLE, AB[12:10]} ),
    .out ( u4out ) );
assign K = u4out[7:0]; // outputs 8,9 are not connected
The Clock

Oscillators have to come from outside an FPGA

EP2C5T144 includes a 50 MHz crystal oscillator on pin 17

The KIM-1 runs at 1 MHz

Solution: Divide-by-50 counter:

```verilog
logic [4:0] clkcount = 5’h0;
logic clk = 1’b0; // 1 MHz clock
always @(posedge clk50) begin // 50 MHz clock
    if (clkcount == 5’d24) begin // Every 25 cycles,
        clkcount <= 5’d0; // Reset the counter
        clk <= ~clk; // And toggle the clock
    end
    else
        clkcount <= clkcount + 5’d1;
end
```

FPGAs also have programmable PLLs; EP2C5’s could only divide down to 10 MHz
1K RAM

Originally 8 6102 SRAM chips. Easy to model:

// U5-U14: 1K Static RAM

logic [7:0] RAM1K [0:1023];
logic [7:0] RAM1K_DO; // Read data from 1K RAM
logic RAM1K_OE; // Data from the 1K RAM

always_ff @(posedge clk) begin
    {RAM1K_OE, RAM1K_DO} <= {1'b0, 8'bx};
    if (!K[0])
        if (WE) RAM1K[ AB[9:0] ] <= DO;
    else {RAM1K_OE, RAM1K_DO} <= { 1'b1, RAM1K[ AB[9:0] ] };
end

// Emulate the tri-state data bus being driven by the peripherals
always_comb
    if (RAM1K_OE) DI = RAM1K_DO;
    else if (RAM128_OE) DI = RAM128_DO;
    else if (ROM2K_OE) DI = ROM2K_DO;
    else if (RIOT002_OE) DI = RIOT002_DO;
    else if (RIOT003_OE) DI = RIOT003_DO;
    else DI = 8’bx;
2K ROM

6530s each have 1K ROM; easier to model a 2K ROM

// 2K ROM within the two 6530s

logic [7:0] ROM2K [0:2047];
initial $readmemh("ROM.hex", ROM2K); // Load the ROM from a file
logic [7:0] ROM2K_DO; // Data from the 2K ROM
logic ROM2K_OE; // 2K ROM selected

always_ff @(posedge clk) begin
    {ROM2K_OE, ROM2K_DO} <= {1'b0, 8'bx};
    if (!K[6] || !K[7])
        {ROM2K_OE, ROM2K_DO} <= {1'b1, ROM2K[ AB[10:0] ] };
end

Creating the .hex file:

# cat 6530-003.bin 6530-002.bin | \ 
    od -v -t x1 -An > ROM.hex

a9 ad 8d ec 17 20 32 19 a9 27 8d 42 17 a9 bf 8d 
43 17 a2 64 a9 16 20 7a 19 ca d0 f8 a9 2a 20 7a
logic IOT_SELECT;
always_ff @(posedge clk)
  if (reset) begin
    PAO <= 8'd0;  PAOE <= 8'd0;
    PBO <= 8'd0;  PBOE <= 8'd0;
  end else begin
    {OE, DO} <= {1'b0, 8'bx};
    if (IOT_SELECT)
      case ({RW, A[2:0]})
        4'b0_000 : PAO <= DI;  // Write port A
        4'b1_000 : {OE, DO} <= {1'b1, PAI_int};  // Read port A
        4'b0_001 : PAOE <= DI;  // Write DDRA
        4'b1_001 : {OE, DO} <= {1'b1, PAOE};  // Read DDRA
        4'b0_010 : PBO <= DI;  // Write port B
        4'b1_010 : {OE, DO} <= {1'b1, PBI_int};  // Read port B
        4'b0_011 : PBOE <= DI;  // Write DDRB
        4'b1_011 : {OE, DO} <= {1'b1, PBOE};  // Read DDRB
      default : ;
    endcase
  end
end
Multiplexed 6 common-anode LEDs
Anodes driven by a PNP transistor
‘145 selects one of the digits or one of the three keyboard rows. Open-collector output.
Cathodes with current-limiting resistors driven by keyboard columns through inverters
We have enough pins; use them:

**Emulated ’145:**

- KB_ROW[2:0] Row outputs (Open Collector)
- Inputs and outputs separated
- LED_SEG[6:0] Segment drivers (Active low)
- KB_COL[6:0] Column inputs (Weak pullup resistor)
Creating Constraints Algorithmically

Use the Quartus GUI or program in Tcl (run quartus_sh):

```tcl
project_new KIM_EP2C5 -overwrite

foreach svfile {
    KIM_1.sv
    mcs6530.sv
    KIM_EP2C5.sv
} {
    set_global_assignment -name SYSTEMVERILOG_FILE $svfile
}

foreach {signal pin} {
    clk50   PIN_17
    LED[0]  PIN_3
    LED[1]  PIN_7
    LED[2]  PIN_9
    KEY     PIN_144

    LED_DIG[4]  PIN_65
    LED_DIG[6]  PIN_69
    LED_DIG[7]  PIN_70
    LED_DIG[8]  PIN_71
    LED_DIG[9]  PIN_72
} {
    set_location_assignment $pin -to $signal
    set_instance_assignment -name OUTPUT_PIN_LOAD 20 -to $signal
}
```
Separate KIM-1-specific from board/chip-specific:

```verilog
module KIM_EP2C5(input clk50,
    inout [7:0] PA,
    inout [7:0] PB,
    output [3:0] KB_ROW,
    input [6:0] KB_COL,
    output [9:4] LED_DIG, // Active low: 4 is leftmost
    output [6:0] LED_SEG, // Active low: 0 is segment A
    // ...
);

KIM_1 TOP( .PAI ( PA ),
    .PBI ( PB ),
    .DECODE_ENABLE ( 1'b0 ),
    .RDY (),
    .K ( K ),
    // ...
);

/* Emulate open-collector outputs on the KB_ROW signals */
logic [3:0] KB_ROW_int;
generate
    for (i = 0 ; i < 4 ; i++ ) begin: KB_ROW_pins
        assign KB_ROW[i] = KB_ROW_int[i] ? 1'bz : 1'b0;
    end
endgenerate
```
Fitter Summary Report

Fitter Status : Successful - Thu May 17 23:19:56 2018
Quartus II 64-Bit Version : 13.0.1 Build 232 06/12/2013 SP
Revision Name : KIM_EP2C5
Top-level Entity Name : KIM_EP2C5
Family : Cyclone II
Device : EP2C5T144C8
Timing Models : Final
Total logic elements : 607 / 4,608 ( 13 % )
   Total combinational functions : 575 / 4,608 ( 12 % )
   Dedicated logic registers : 259 / 4,608 ( 6 % )
Total registers : 259
Total pins : 52 / 89 ( 58 % )
Total virtual pins : 0
Total memory bits : 28,160 / 119,808 ( 24 % )
Embedded Multiplier 9-bit elements : 0 / 26 ( 0 % )
Total PLLs : 0 / 2 ( 0 % )
Other FPGA Retro Sites

http://www.fpgaarcade.com

Featured Cores

AMIGA
The Amiga was based on the Motorola 68000 CPU and was sold by Commodore between 1985 and 1994. Its advanced graphics and sound made it popular for gaming and video production.

commodore 64
The Commodore 64 is a computer that was manufactured by Commodore between 1982 and 1994. It was very popular during the 80's and early 90's.

Calaga

VIC-20

Grant Searle
http://searle.hostei.com/grant/Multicomp/

MiSTer FPGA board
https://github.com/MiSTer-devel/Main_MiSTer/wiki