**ELEC6234**

**Embedded Processor Design**

FPGA synthesis of a microprocessor core
Synthesis of registers and memory

---

**Sequential logic synthesis**

- Registers and memory in processors
  - Program Counter
  - General Purpose Register File
  - Instruction Register (not in picoMIPS)
  - RAM – sequential RAM in modern FPGAs
- when writing code for sequential logic, use always_ff blocks with incomplete conditional statements;
  - use always_latch if you want transparent latches; not recommended in modern FPGAs
- avoid temptation to mix complex combinational and sequential logic in one block; partition your design and use multiple always blocks
- for state machines use edge-triggered flip-flops (always_ff)

---

**Edge-triggered flip-flops**

- rising edge triggered D-type flip-flop with no reset module
```verilog
module ff1 (input logic clk, d, output logic q);
    always_ff @ (posedge clk) q <= d;
endmodule
```

- falling edge triggered D-type flip-flop with active-high asynchronous clear module
```verilog
module ff2 (input logic clk, D, Clear, output logic Q);
    always_ff @ (negedge clk or posedge Clear)
        if (Clear) Q <= 1'b0;
        else Q <= D;
endmodule
```
### Edge-triggered flip-flops – cont.

```verilog
module ff3 (input logic clk, D, Preset, output logic Q);
always_ff @ (posedge clk)
if (Preset)
Q <= 1'b1;
else
Q <= D;
endmodule
```

### Rising edge triggered D-type flip-flop with clock enable

```verilog
module ff4 (input logic clk, D, CE, output logic Q);
always_ff @ (posedge clk)
if (CE)
Q <= D;
endmodule
```

### RAMs used in FPGA designs are typically arrays of latches with a separate input and a separate output data bus, an address bus and a Write Enable signal; read is asynchronous

- Modern FPGAs (e.g. Cyclone V) support only synchronous memory, where both read and write is synchronous
- Note that there many types of RAMs, eg.
  - RAM with synchronous read
  - RAM with one Enable controlling both ports
  - RAM with separate Enables controlling each port
  - Multiple-Port RAMs
- Synthesis tools are usually able to infer the correct type of RAM supported by the target FPGA, regardless of the SystemVerilog description, but it is useful to be familiar with the specific types of RAM supported by the target FPGA.

### Standard RAM with asynchronous read

```verilog
module ram128x8 (input logic we, input logic [6:0] address, input logic [7:0] din, output logic [7:0] dout);
logic [7:0] ram [127:0]; // this 2-dimensional array defines the ram memory
  // write block
  always_latch if (we)
    ram[address] <= din;
  // asynchronous read block
  assign dout = ram[address];
endmodule
```

### Quartus 2 synthesis summary

<table>
<thead>
<tr>
<th>Family</th>
<th>Cyclone IV GX</th>
<th>Cyclone V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total logic elements</td>
<td>1,854</td>
<td>1,512</td>
</tr>
<tr>
<td>Total combinational functions</td>
<td>1,854</td>
<td>1,512</td>
</tr>
<tr>
<td>Dedicated logic registers</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total registers</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total pins</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Total virtual pins</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total memory bits</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: no memory bits have been used in the synthesis!

### Synchronous RAM blocks in modern FPGAs

Altera Cyclone V devices feature memory structures that consists M9K memory blocks that can be configured to provide RAM, shift registers, ROM, and FIFO buffers. An M9K block contains 8,192 memory bits and is synchronous, i.e. requires a clock.

1. M9K blocks—10-kilobit (Kb) blocks for larger memory configurations.
2. Memory logic array blocks (MLABs)—640-bit memory blocks for small memories. Each MLAB can be configured as ten 32 x 2 blocks, giving one 32 x 20 simple dual-port SRAM.

### RAM synthesis

- RAMs used in FPGA designs are typically arrays of latches with a separate input and a separate output data bus, an address bus and a Write Enable signal; read is asynchronous
- Modern FPGAs (e.g. Cyclone V) support only synchronous memory, where both read and write is synchronous
- Note that there many types of RAMs, eg.
  - RAM with synchronous read
  - RAM with one Enable controlling both ports
  - RAM with separate Enables controlling each port
  - Multiple-Port RAMs
- Synthesis tools are usually able to infer the correct type of RAM supported by the target FPGA, regardless of the SystemVerilog description, but it is useful to be familiar with the specific types of RAM supported by the target FPGA.
For details see file regs.sv

Write is synchronous
### Program Memory

// File name: prog.sv
// Function: Program memory reads x size - reads from file prog.hex
// Author: tjk
// Last rev.: 24 Oct 2012

module prog #(parameter psize = 6, isize = 24) // psize - address width, isize - instruction width
    (input logic [psize-1:0] address, // input logic [psize-1:0] address
     output logic [isize:0] instr), // instr - instruction code
    // program memory declaration, note: 1<<n is same as 2^n
    logic [isize:0] progMem [1<<psize-1:0]; // logic [isize:0] progMem [1<<psize-1:0]

module // end of module prog

// get memory contents from file iniGal$
readmemh ("prog.hex", progMem);

// program memory read
assign instr = progMem[address];

endmodule // end of module prog

Notes:
1) program file name must be prog.hex
2) use 1<<n to calculate 2^n

### Sample prog.hex

1) syntax is simple: each line contains instruction code in hex followed by a comment
2) this is a simple test that uses NOP, LDI (Load Immediate to Register) and ADD instructions
3) LDI is a 'synthetic' instruction: LDI %d, imm is implemented as ADD %d, %0, imm

### Summary

- **Progress so far:**
  - Data path
    - ALU, General Purpose Registers,
  - Control path
    - Program Counter, Program Memory,
  - Next step:
    - Develop a simple decoder and a CPU encapsulating module to allow integration of modules developed so far and simple program execution

### Notes:

1) syntax is simple: each line contains instruction code in hex followed by a comment
2) this is a simple test that uses NOP, LDX (Load Immediate to Register) and ADD instructions