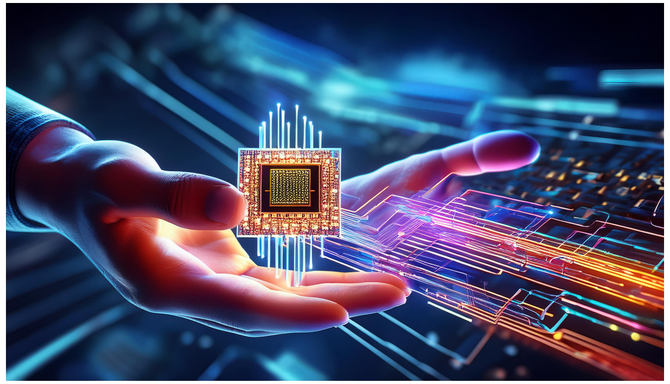
CSCI 250
Introduction to Computer Organisation
Lecture 2: Computer Memory II



Jetic Gū

Overview

- Focus: Course Introduction
- Architecture: Logical Circuits
- Textbook: v4: 8.3; v5: 7.3
- Core Ideas:
    1. RAM in FPGA
    2. Building RAM Modules
    3. Lab 2

RAM in Verilog

RAM Modules

- Two primary types of memory for FPGA:
  - On-die (On-chip): the amount of memory located on the FPGA chip. Often SRAM
  - Onboard: the amount of memory available to the chip, but located on the board. Often DDR RAM
- On-die RAM modules are automatically synthesised and created in Verilog. This will be our focus today
- Onboard RAM chips vary depending on manufacturer and model

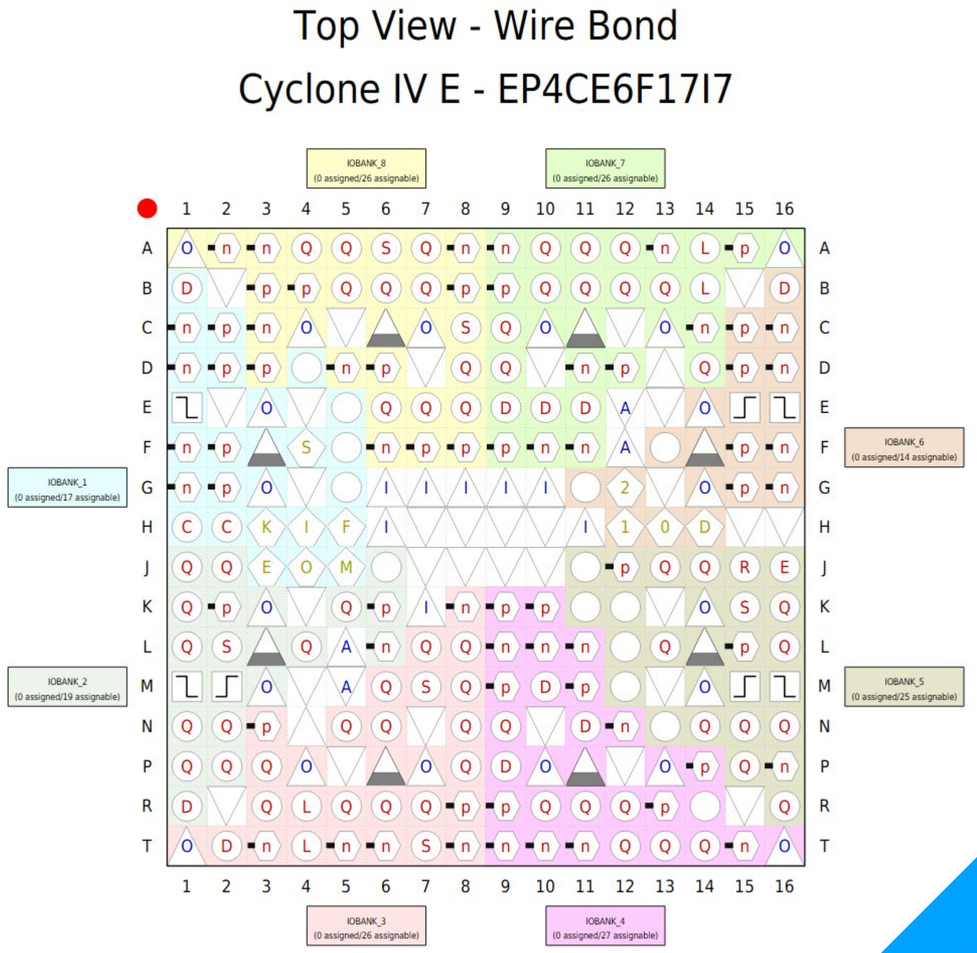
Concept

RAM Modules

- For example: my board (PIAX301-V2 dev board, using Cyclone IV EP4CE6F17I7
  - On-die: 270 Kbit. These cover all of the registers, flip-flops, latches, etc. If you want on-die SRAM, this is also what they use.
  - Onboard: 256 Mbit. These are dynamic RAM, and the chip is external to the FPGA die. They have their own interfaces

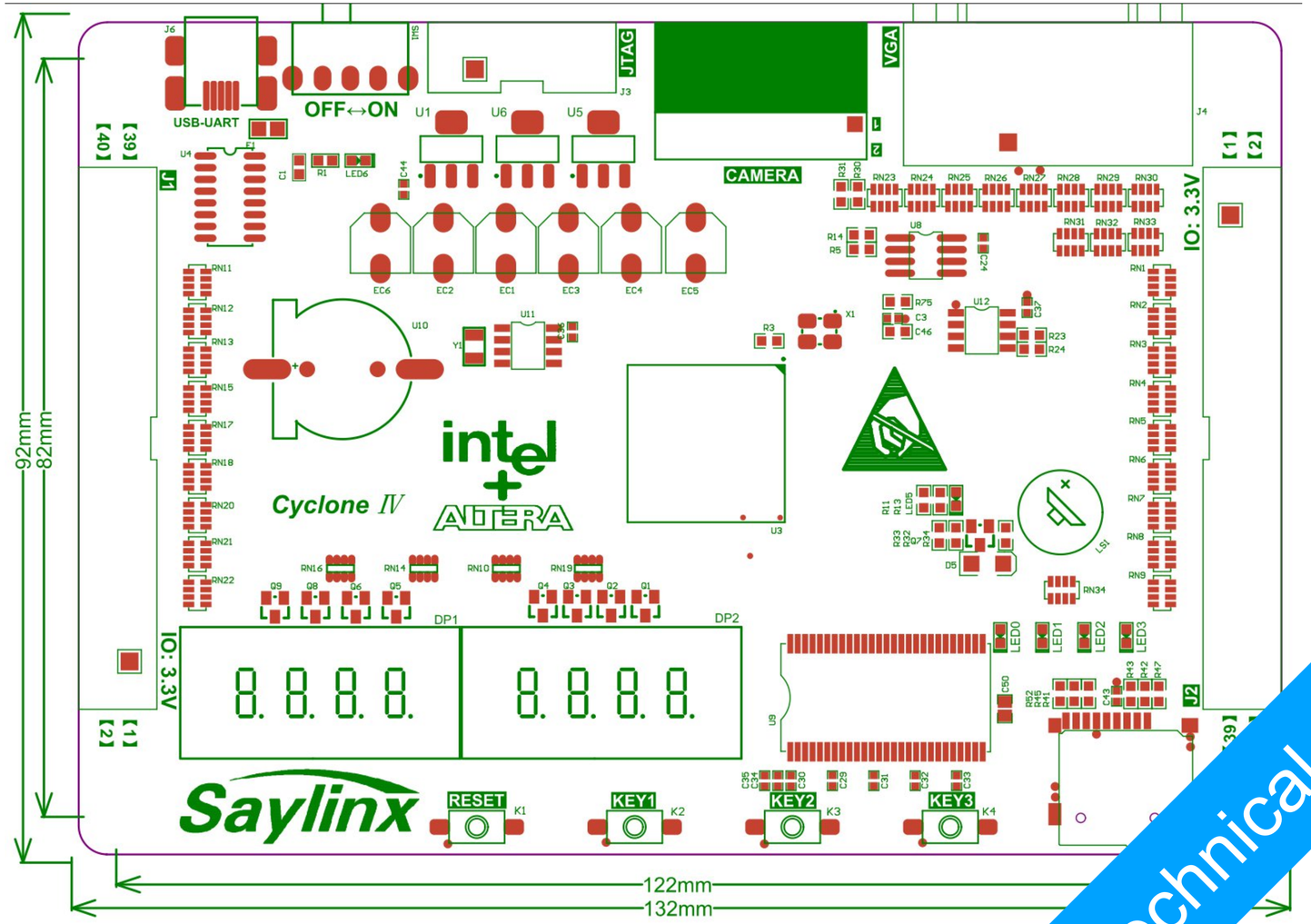
RAM Modules

- This is the `Pin Planner`. You can find it under `Assignments` in Quartus Prime
- When you use any IO external to the FPGA die
  - LED bulbs, 7 segment displays, USB ports, and of course RAM chips
  - you need to look at the board specs and assign these pins



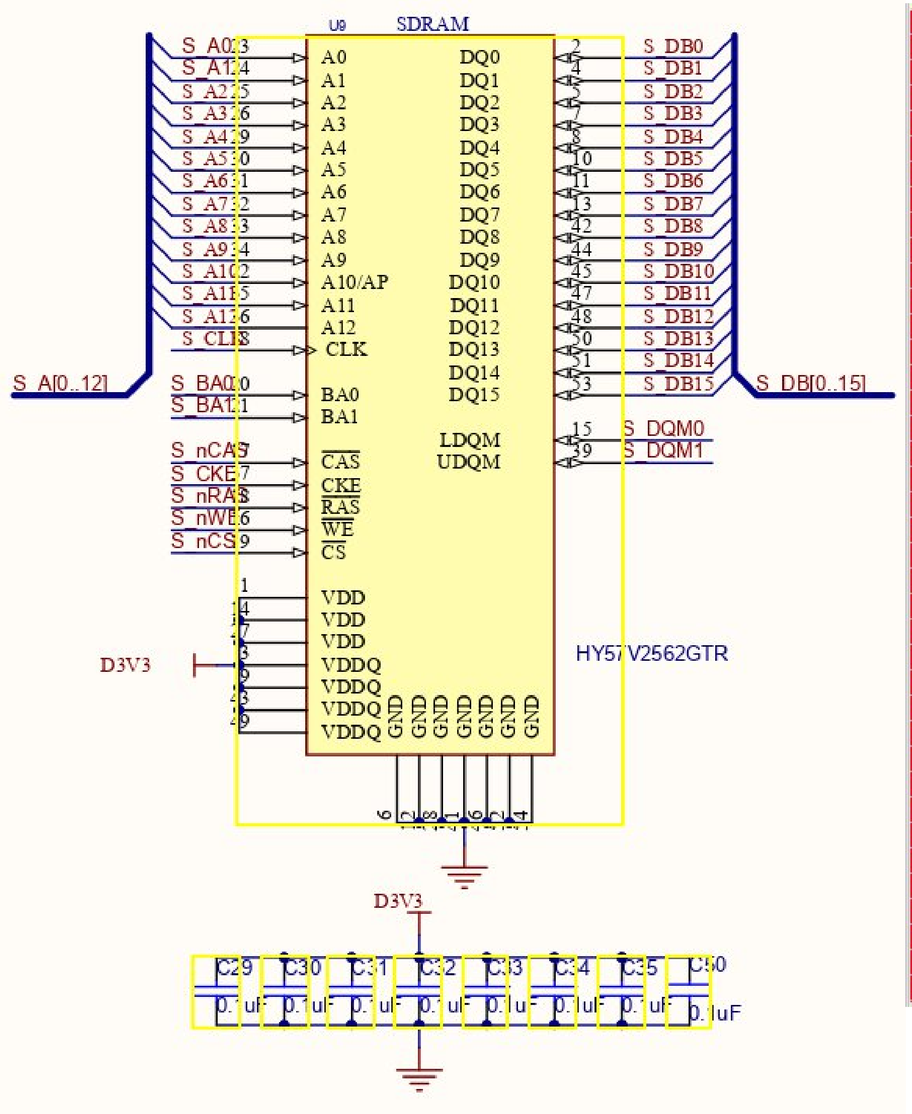
RAM Modules

- This is my board, this is taken from the dev manual of my board
- You can see the names of each pin of each IO on this board



RAM Modules

- This is my onboard DSRAM
- 16bits for data, bi-directional
- 13bits for address, these go out from the FPGA chip into the DSRAM module



sdram_we_n	Unknown	PIN_J13
sdram_ras_n	Unknown	PIN_K11
sdram_dqm[0]	Unknown	PIN_J14
sdram_dqm[1]	Unknown	PIN_G15
sdram_dq[0]	Unknown	PIN_P14
sdram_dq[1]	Unknown	PIN_M12
sdram_dq[2]	Unknown	PIN_N14
sdram_dq[3]	Unknown	PIN_L12
sdram_dq[4]	Unknown	PIN_L13
sdram_dq[5]	Unknown	PIN_L14
sdram_dq[6]	Unknown	PIN_L11
sdram_dq[7]	Unknown	PIN_K12
sdram_dq[8]	Unknown	PIN_G16
sdram_dq[9]	Unknown	PIN_J11
sdram_dq[10]	Unknown	PIN_J16
sdram_dq[11]	Unknown	PIN_J15
sdram_dq[12]	Unknown	PIN_K16
sdram_dq[13]	Unknown	PIN_K15
sdram_dq[14]	Unknown	PIN_L16
sdram_dq[15]	Unknown	PIN_L15
sdram_cs_n	Unknown	PIN_K10
sdram_clk	Unknown	PIN_B14
sdram_cke	Unknown	PIN_F16
sdram_cas_n	Unknown	PIN_J12
sdram_ba[0]	Unknown	PIN_G11
sdram_ba[1]	Unknown	PIN_F13
sdram_addr[0]	Unknown	PIN_F11
sdram_addr[1]	Unknown	PIN_E11
sdram_addr[2]	Unknown	PIN_D14
sdram_addr[3]	Unknown	PIN_C14
sdram_addr[4]	Unknown	PIN_A14
sdram_addr[5]	Unknown	PIN_A15
sdram_addr[6]	Unknown	PIN_B16
sdram_addr[7]	Unknown	PIN_C15
sdram_addr[8]	Unknown	PIN_C16
sdram_addr[9]	Unknown	PIN_D15
sdram_addr[10]	Unknown	PIN_F14
sdram_addr[11]	Unknown	PIN_D16
sdram_addr[12]	Unknown	PIN_F15

Building RAM Modules

Why On-Die RAM

- We need to simulate having a RAM chip
  - We will setup a synchronous sequential module with the same interface as our example SDRAM onboard chip
  - During simulation, we use this on-die RAM module
- Should we use on-die RAM modules or onboard RAM modules in production?

Code for RAM Module

- Single-port RAM: means we have one databus for both read and write, and one address bus
- Basic Interfaces:
  - `A[12:0]`: for memory address
  - `DB[15:0]`: for databus
  - `clk`: Clock, `oe`: Output Enabled, `we`: Write Enabled, `cs`: Chip Select;

Code for RAM Module

- #: **parameters** are like constants you declare as macros in C/C++. They are replaced by actual values at compile time
- Width: number of bits
- Depth here: number of unique memory addresses
- Note: this is **NOT** byte-addressable memory
  - Real byte-addressable memory requires more steering to enable individual byte read/write. Not necessary to us

```
module ram
  # (parameter ADDR_WIDTH = 13,
     parameter DATA_WIDTH = 16,
     parameter DEPTH = 8192
    )

  (    input                  clk,
       input [ADDR_WIDTH-1:0]   a,
       inout [DATA_WIDTH-1:0]   db,
       input                  cs,
       input                  we,
       input                  oe
  );

  reg [DATA_WIDTH-1:0]   tmp_data;
  reg [DATA_WIDTH-1:0]   mem [DEPTH:0];

  always @ (posedge clk) begin
    if (cs & we)
      mem[a] <= db;
  end

  always @ (posedge clk) begin
    if (cs & !we)
      tmp_data <= mem[a];
  end

  assign db= cs & oe & !we ? tmp_data : 'hz;
endmodule
```

Code

Code for RAM Module

- `inout`: bidirectional wire here, nothing to see

cs ChipSelect	we WriteEnable	oe OutputEnable	Behaviour
0	X	X	Output 'hz, high-Z
1	0	0	Read mode No output
1	0	1	Read mode Output db
1	1	0	Write mode No output
1	1	1	Write mode No output

```
module ram
  # (parameter ADDR_WIDTH = 13,
     parameter DATA_WIDTH = 16,
     parameter DEPTH = 8192
    )

  (   input                  clk,
      input [ADDR_WIDTH-1:0]  a,
      inout [DATA_WIDTH-1:0]  db,
      input                  cs,
      input                  we,
      input                  oe
  );

  reg [DATA_WIDTH-1:0]  tmp_data;
  reg [DATA_WIDTH-1:0]  mem [DEPTH:0];

  always @ (posedge clk) begin
    if (cs & we)
      mem[a] <= db;
  end

  always @ (posedge clk) begin
    if (cs & !we)
      tmp_data <= mem[a];
  end

  assign db = cs & oe & !we ? tmp_data : 'hz;
endmodule
```

Code

Code for RAM Module

- `mem`: main memory array
  - Brackets before: the contents for each address, 16bits
  - Brackets after: the number of words in the memory, 8192 words
- `tmp_data`: temporary buffer for data to go out
  - Think: why not go directly out?

```
module ram
  # (parameter ADDR_WIDTH = 13,
     parameter DATA_WIDTH = 16,
     parameter DEPTH = 8192
    )

  (   input                  clk,
      input [ADDR_WIDTH-1:0]  a,
      inout [DATA_WIDTH-1:0]  db,
      input                  cs,
      input                  we,
      input                  oe
  );

  reg [DATA_WIDTH-1:0]  tmp_data;
  reg [DATA_WIDTH-1:0]  mem [DEPTH:0];

  always @ (posedge clk) begin
    if (cs & we)
      mem[a] <= db;
  end

  always @ (posedge clk) begin
    if (cs & !we)
      tmp_data <= mem[a];
  end

  assign db = cs & oe & !we ? tmp_data : 'hz;
endmodule
```

Code

Code for RAM Module

- `always`: procedural block
  - Statements inside an always block are executed sequentially
- `@(posedge clk)`: trigger
  - This block is executed at the **positive edge** of CLK
- If just one statement, no need for `begin/end`

```
module ram
  # (parameter ADDR_WIDTH = 13,
     parameter DATA_WIDTH = 16,
     parameter DEPTH = 8192
    )

  (   input                  clk,
      input [ADDR_WIDTH-1:0]  a,
      inout [DATA_WIDTH-1:0]  db,
      input                  cs,
      input                  we,
      input                  oe
  );

  reg [DATA_WIDTH-1:0]  tmp_data;
  reg [DATA_WIDTH-1:0]  mem [DEPTH:0];

  always @ (posedge clk) begin
    if (cs & we)
      mem[a] <= db;
  end

  always @ (posedge clk) begin
    if (cs & !we)
      tmp_data <= mem[a];
  end

  assign db = cs & oe & !we ? tmp_data : 'hz;
endmodule
```

Code

Code for RAM Module

- Is this sequential or combinational?
- `expr? exprA: exprB`
  - This is called a Ternary Operator, same as in C/C++
  - Difference? This is combinational
  - You can write it as boolean operators only (How?)

```
module ram
  # (parameter ADDR_WIDTH = 13,
     parameter DATA_WIDTH = 16,
     parameter DEPTH = 8192
    )

  (   input                  clk,
      input [ADDR_WIDTH-1:0]  a,
      inout [DATA_WIDTH-1:0]  db,
      input                  cs,
      input                  we,
      input                  oe
  );

  reg [DATA_WIDTH-1:0]  tmp_data;
  reg [DATA_WIDTH-1:0]  mem [DEPTH:0];

  always @ (posedge clk) begin
    if (cs & we)
      mem[a] <= db;
  end

  always @ (posedge clk) begin
    if (cs & !we)
      tmp_data <= mem[a];
  end

  assign db = cs & oe & !we ? tmp_data : 'hz;
endmodule
```

Code

P3
Lab 2

Building RAM Modules

P2
Lab 2

What will you need to build?

- A Python Exercise (See LS7)
- An SRAM Cell using block diagram (See LS5)
- A RAM module using Verilog (LS6), with DQM (more to follow)[1]
- Waveform simulation required

1. https://www.alldatasheet.com/datasheet-pdf/view/333572/HYNIX/H57V2562GTR-75C.html

Technical