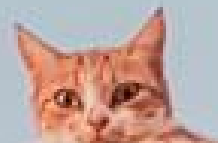


Computer Organization

Lab11 CPU(3) clock, I/O, Controller+

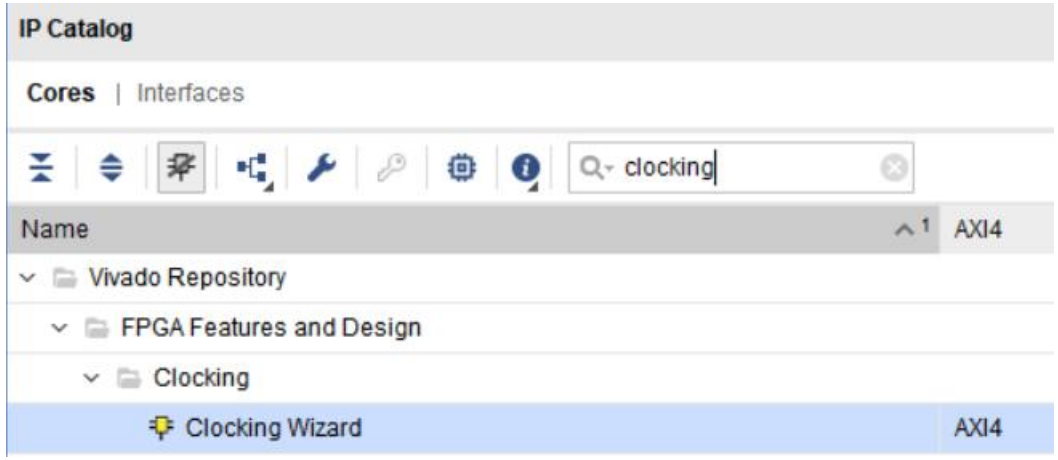
Topics

- CPU (3)
  - Clock
  - Build a Single Cycle CPU， Test it
- CPU with I/O
  - I/O
  - Controller+

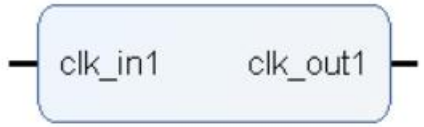
3 Clock

1. Add **PPL clock IP core** to generate a clock

    1. The clock on the Minisys development board is 100Mhz
    2. A clock of **23Mhz** is needed for the single clock CPU



Clock continued



2. Set the name and features of clock IP core

Component Name: cpuclk

Clocking Options | Output Clocks | Port Renaming | PLLE2 Settings | Summary

Clock Monitor

- [ ] Enable Clock Monitoring

Primitive

- ( ) MMCM (•) PLL

Clocking Features

- [x] Frequency Synthesis
- [ ] Minimize Power
- [x] Phase Alignment
- [ ] Dynamic Reconfig
- [ ] Safe Clock Startup

Jitter Optimization

- (•) Balanced
- ( ) Minimize Output Jitter
- ( ) Maximize Input Jitter filtering

Clocking Options | Output Clocks | Port Renaming | PLLE2 Settings

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)	
		Requested	Actual
[x] clk_out1	clk_out1	23.000	23.000

Component Name: cpuclk

Clocking Options | Output Clocks | Port Renaming | PLLE2 Settings | Summary

Enable Optional Inputs / Outputs for MMCM/PLL

- [ ] reset
- [ ] power_down
- [ ] locked

Reset Type: (•) Active High ( ) Active Low

The Function Verification of “cpuclk”

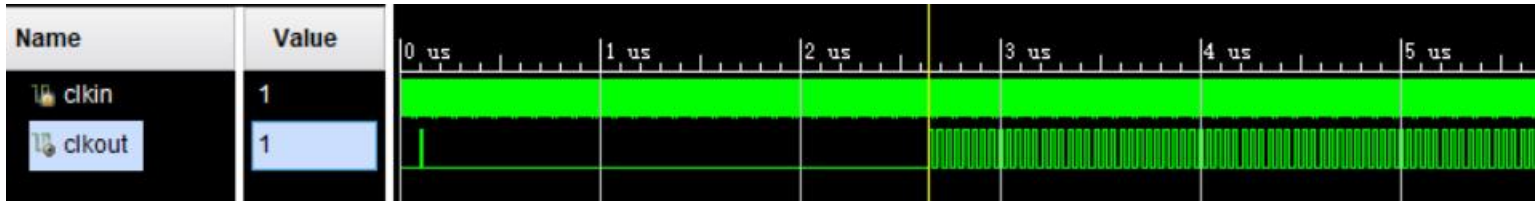
NOTE: The output of IP core 'cpuclk' need to work for a period of time to achieve stability.

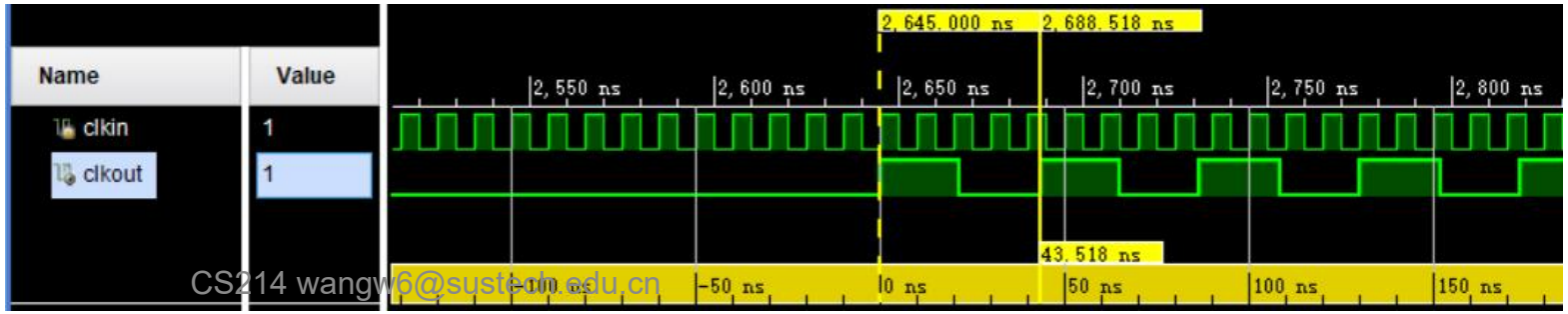
```
// a reference testbench for 'cpuclk'
module cpuclk_tb(    );
    reg clkin;
    wire clkout;
    cpuclk clk1( .clk_in1(clkin), .clk_out1(clkout) );
    initial       clkin = 1'b0;
    always #5 clkin=~clkin;
endmodule
```

Functional Verification

1) Create a verilog design module to perform instance and port binding on the IP core.

2) Set up testbench to verify whether the output signal is a 23Mhz clock signal while the input signal is 100Mhz



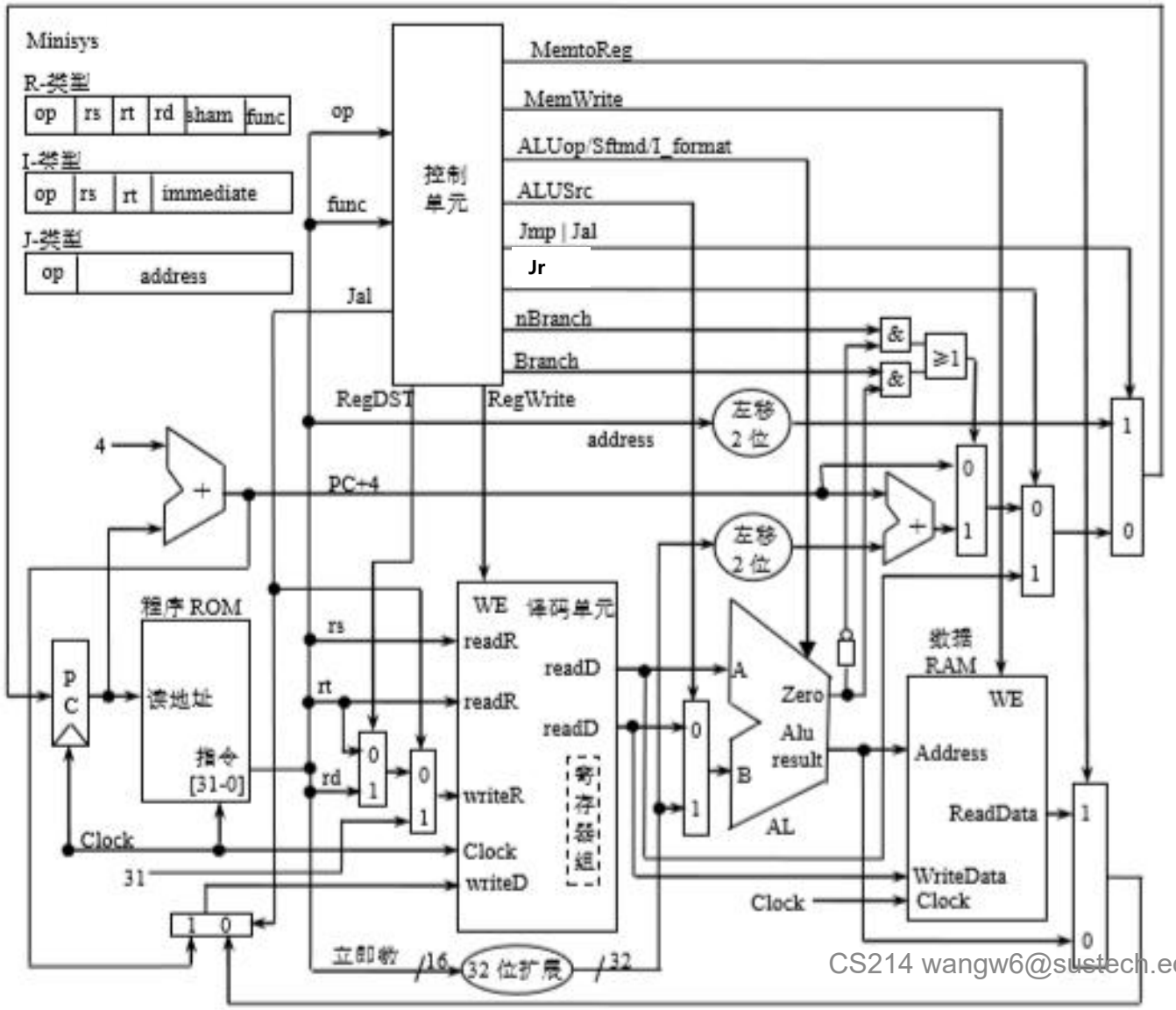


Single Cycle CPU

- Determine the input and output ports of the CPU
  - **Clock signal, reset signal**
  - **Input port ?**
  - **Output port ?**
- Determine the sub-module inside the CPU
  - **Clock module**
  - **IFetch, Controller, Decoder, Execution/ALU, Data-Memory** (IO processing
- Build CPU top-level module
  - **Notes the relationship on the sub-modules inside the CPU, especially the control signal, address, data, instruction between sub-modules.**
  - **Build CPU by using Structural Design in verilog or Block Design in Vivado.**

- NOTE: **Case sensitivity** in Verilog syntax.

7 Single Cycle CPU continued



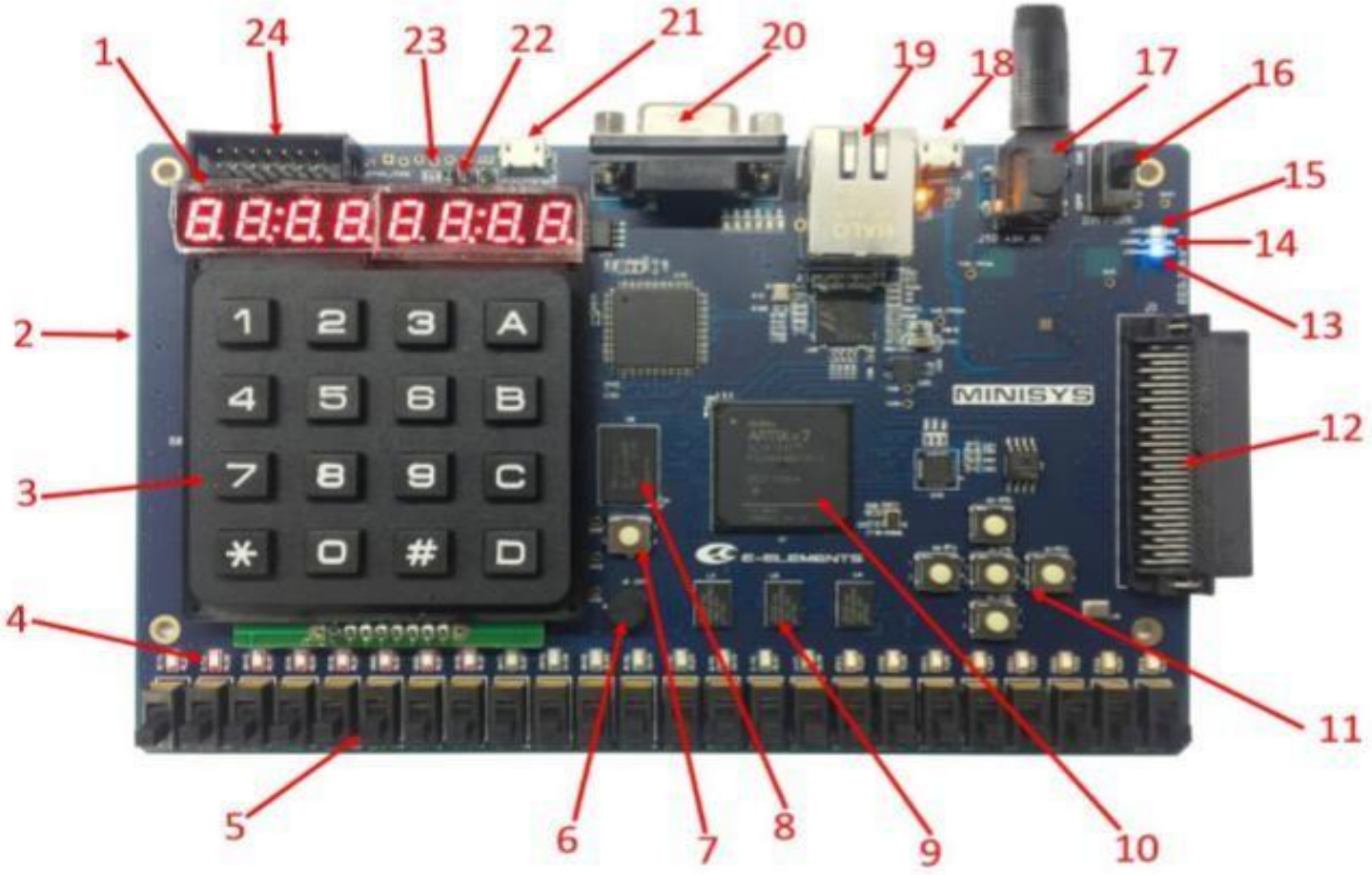
Build a **CPU top** module

1) **Instantiating** the sub-modules: **clock**, **Decoder**, execution unit/**ALU**, **IFetch, Controller** and **Data-Memory**.

2) Complete the inter-module connection inside the CPU and the **binding** to the CPU **port.**

Q. How to test the CPU ? how to determine the program, the data, how to check the result?

I/O Interface

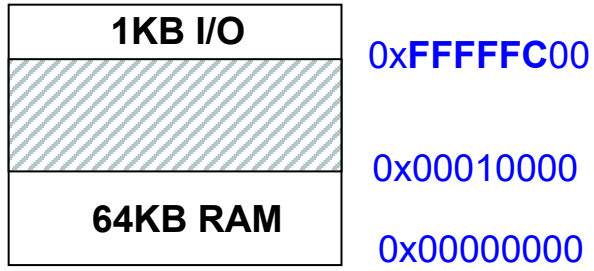


A Simple Design on the I/O Interface

This part mainly accomplishes the following work:

1. Add I/O function
2. 16-bit LED design
3. 16-bit DIP Switch design

I/O Share Part of the Data Bus Address



The space of **32** bits address bus is **4GB**(0x0000_0000~0xFFFF_FFFF)

1024 bytes(0xFFFF_FC00~0xFFFF_FFFF) is designed to be allocated for the **I/O**.
Chip **S**elect and **address** are specified by specifying **10** IO port lines.

Here is an example for **24** LED lights and **24** DIP switches on Minisys board, both of them are divided into two groups, all the ports in one group share the same address.

1. The CS(Chip Select) signal of the LED light is **ledCtrl**
2. The CS(Chip Select) signal of the DIP switch is **switchCtrl**

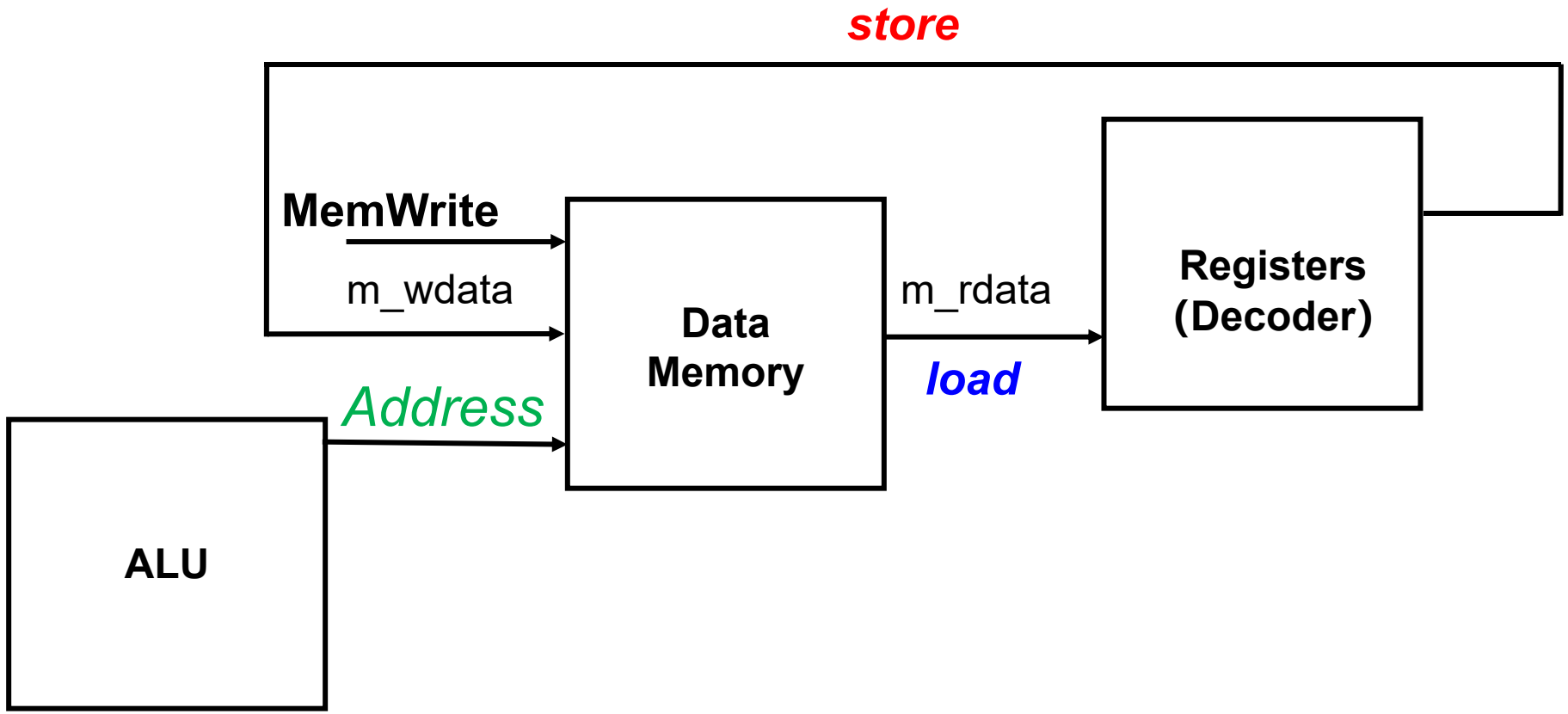
Range	LED(1~16)	LED(17~24)	Switch(1~16)	Switch(17~24)
Address	0x**FFFFFC60**	0x**FFFFFC62**	0x**FFFFFC70**	0x**FFFFFC72**

Note:

*1. In the computer field, there are usually two schemes for I/O address space design: I/O and memory **unified addressing** or **I/O independent addressing**. However there is no dedicated I/ O instruction in current Minisys-1. Here, both LW and SW instructions are used for RAM access and I/O access, which means Minisys-1 can only use I/O unified addressing.*

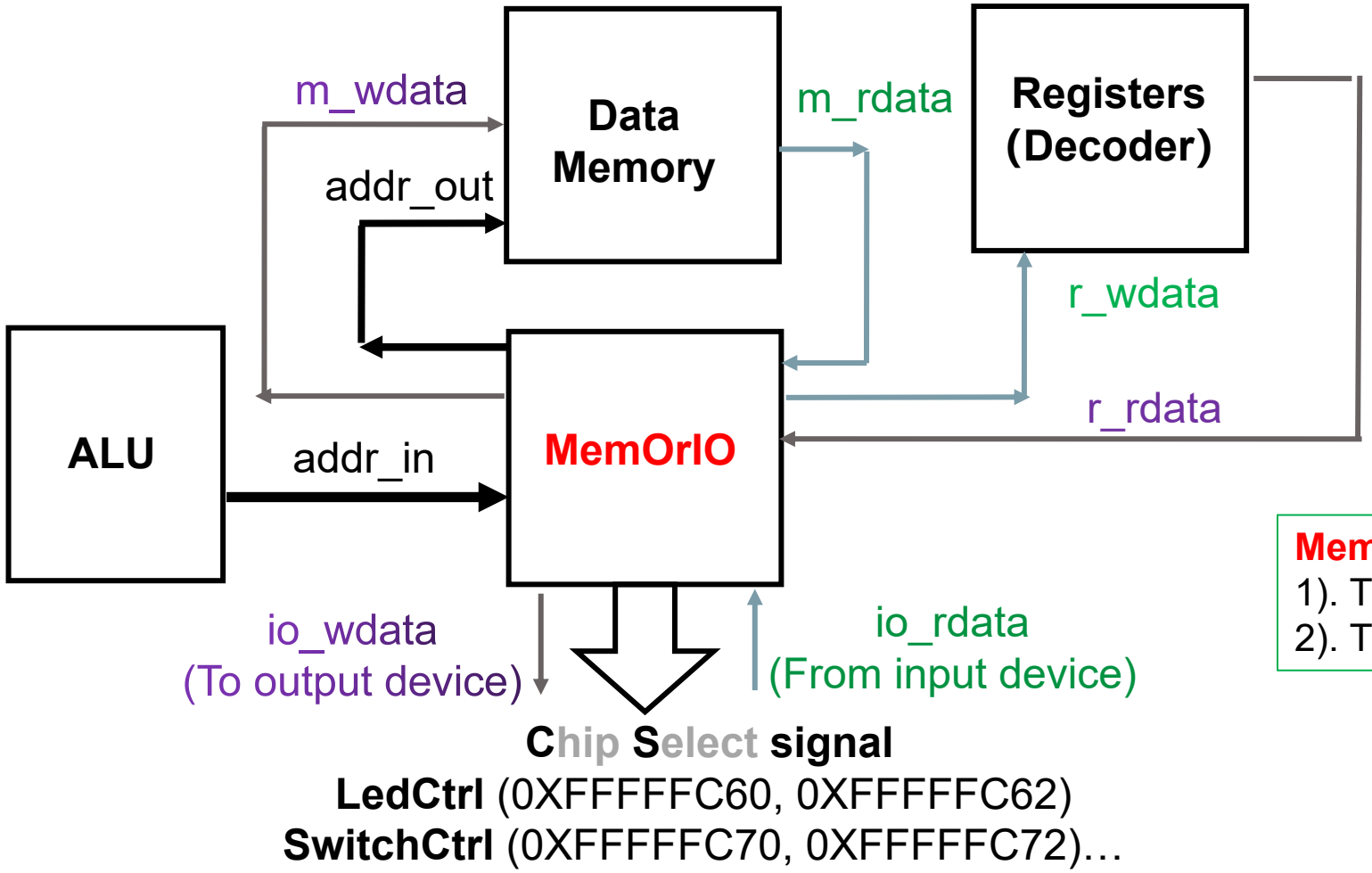
2. It is just a way for IO address implementation, but not the only choice.

Corresponding Operation of LW/SW



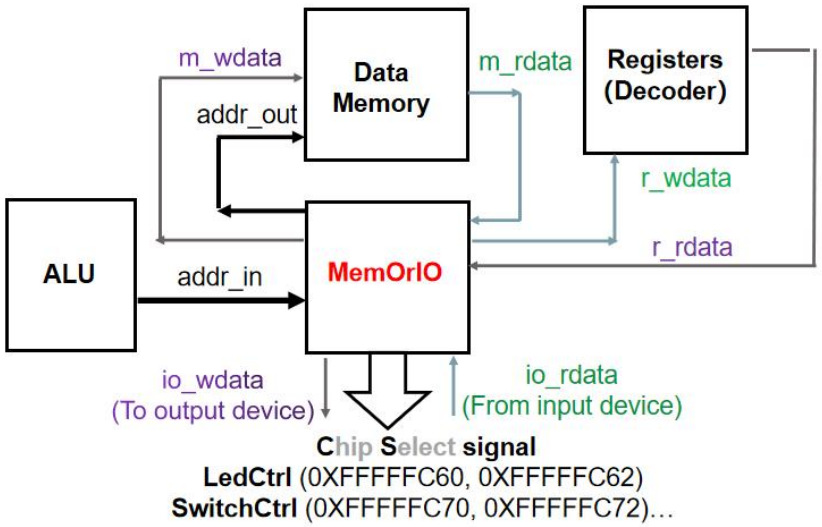
NOTE:
1) There is no specific instruction in Minisys to read data from input ports and write data to output ports.
2) To implement the read/write process on I/O, it needs to **share the load/store instructions** in Minisys.

MemOrIO



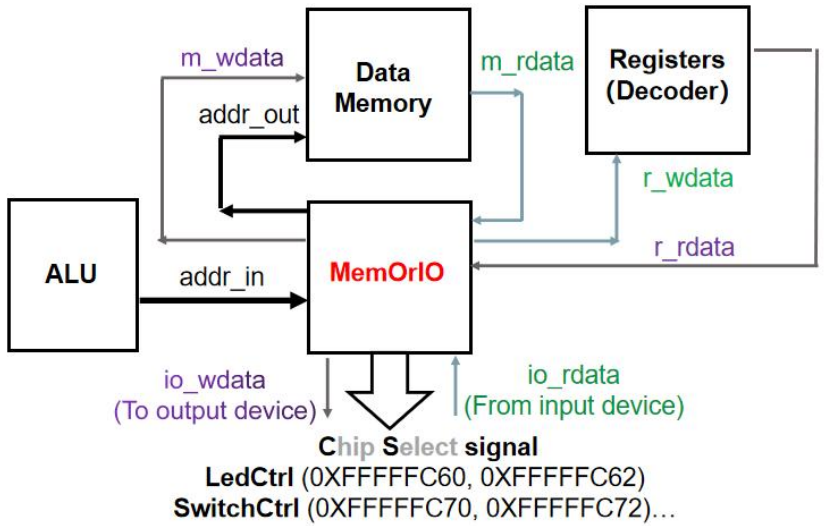
MemOrIO determine:
1). The source of r_rdata
2). The destination of r_wdata

MemOrIO continued




```
module MemOrIO( mRead, mWrite, ioRead, ioWrite,addr_in, addr_out,
m_rdata, io_rdata, r_wdata, r_rdata, write_data, LEDCtrl, SwitchCtrl);

input mRead;                    // read memory, from Controller
input mWrite;                   // write memory, from Controller
input ioRead;                   // read IO, from Controller
input ioWrite;                  // write IO, from Controller

input[31:0] addr_in;            // from alu_result in ALU
output[31:0] addr_out;          // address to Data-Memory

input[31:0] m_rdata;            // data read from Data-Memory
input[15:0] io_rdata;           // data read from IO,16 bits
output[31:0] r_wdata;           // data to Decoder(register file)

input[31:0] r_rdata;        // data read from Decoder(register file)
output reg[31:0] write_data;    // data to memory or I/O  (m_wdata, io_wdata)
output LEDCtrl;                 // LED Chip Select
output SwitchCtrl;              // Switch Chip Select
```

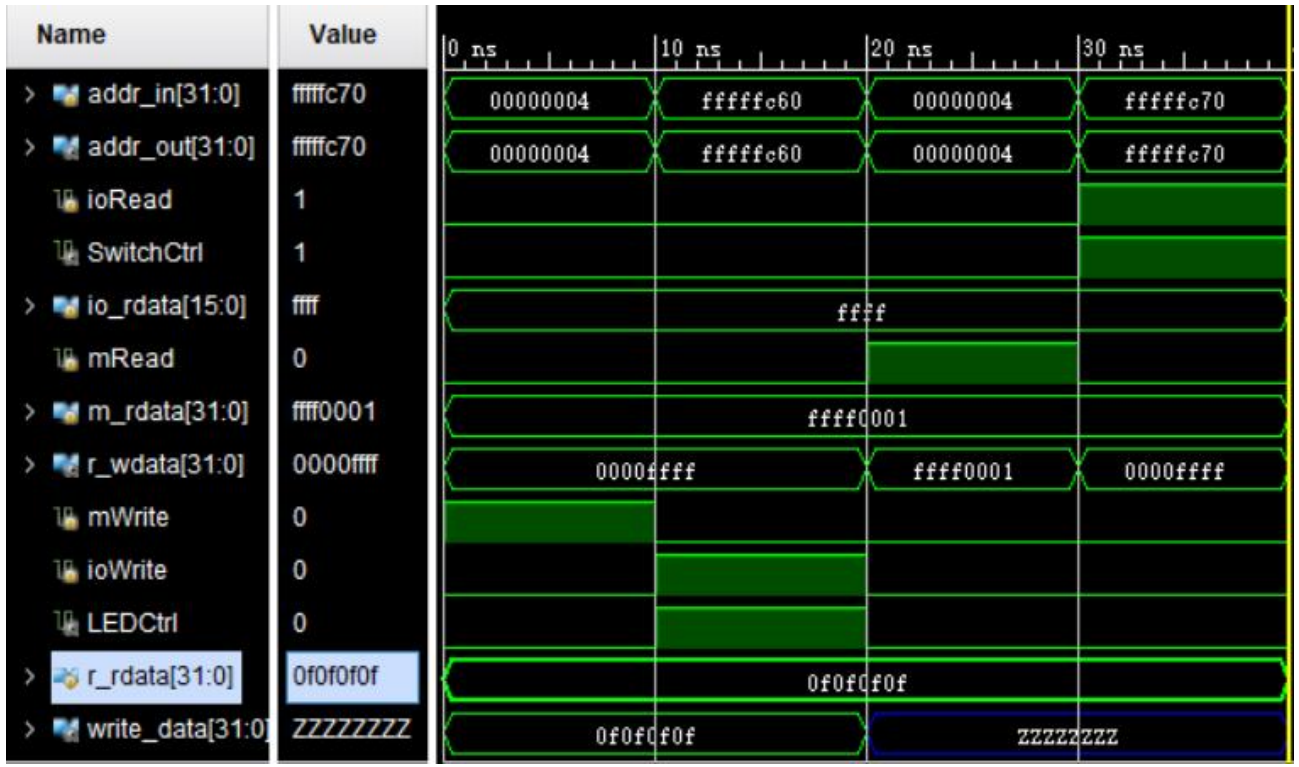
MemOrIO continued




```
assign addr_out= addr_in;
// The data wirte to register file may be from memory or io.
// While the data is from io, it should be the lower 16bit of r_wdata.
assign r_wdata = ? ? ?

// Chip select signal of Led and Switch are all active high;
assign LEDCtrl= ? ? ?
assign SwitchCtrl= ? ? ?

always @* begin
        if((mWrite==1)||(ioWrite==1))
            //wirte_data could go to either memory or IO. where is it from?
                write_data = ? ? ?
        else
                write_data = 32'hZZZZZZZZ;
end
endmodule
```

The Function Verification of MemOrIO

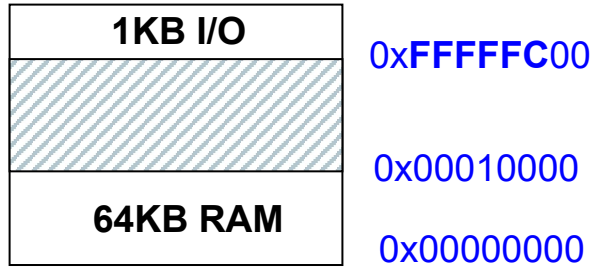
```
// a reference for the testbench of MemOrIO
module MemOrIO_tb( );
   reg mRead,mWrite,ioRead,ioWrite;
   reg[31:0] addr_in,m_rdata,r_rdata;
   reg[15:0] io_rdata;
   wire LEDCtrl,SwitchCtrl;
   wire [31:0] addr_out,r_wdata,write_data;

   MemoryOrIO umio(addr_out, addr_in,
mRead, mWrite, ioRead, ioWrite,
m_rdata, io_rdata, r_rdata, r_wdata, write_data,
LEDCtrl, SwitchCtrl );
```




```
 initial begin    // r_rdata -> m_wdata(write_data)
 m_rdata = 32'h0xffff_0001;   io_rdata = 32'h0xffff;   r_rdata = 32'h0x0f0f_0f0f;    addr_in = 32'h4;{mRead,mWrite,ioRead,ioWrite}= 4'b01_00;
 #10   addr_in = 32'hffff_fc60;   {mRead,mWrite,ioRead,ioWrite}= 4'b00_01;          // r_rdata -> io_wdata(write_data)
 #10   addr_in = 32'h0000_0004;    {mRead,mWrite,ioRead,ioWrite}= 4'b10_00;       // m_rdata -> r_wdata
 #10   addr_in = 32'hffff_fc70;     {mRead,mWrite,ioRead,ioWrite}= 4'b00_10;      // io_rdata -> r_wdata(write_data)
 #10 $finish;
   end
endmodule
```

Controller +

Add new ports to Controller for IO reading and writing support.




```
module  control32(Opcode,Function_opcode,Jr,Branch,nBranch,Jmp,Jal,
Alu_resultHigh,
RegDST, MemorIOtoReg, RegWrite,
MemRead, MemWrite,
IORead, IOWrite,
ALUSrc,ALUOp,Sftmd,I_format);
    ...
    input[21:0] Alu_resultHigh;     // From the execution unit Alu_Result[31..10]
    output MemorIOtoReg;    // 1 indicates that data needs to be read from memory or I/O to the register
    output RegWrite;        // 1 indicates that the instruction needs to write to the register
    output MemRead;         // 1 indicates that the instruction needs to read from the memory
    output MemWrite;        // 1 indicates that the instruction needs to write to the memory
    output IORead;          // 1 indicates  I/O read
    output IOWrite;         // 1 indicates  I/O write
    ...
```

Controller + continued

1) **Modify** the logic of the '***MemWrite***'
2) **Add '*MemRead*'**, '***IORead***' and '***IOWrite***' signals**,**
3) **Change '*MemtoReg*'** to '***MemorIOtoReg***'.

```
 // The real address of LW and SW is Alu_Result, the signal comes from the execution unit
// From the execution unit Alu_Result[31..10], used to help determine whether to process Mem or IO
   input[21:0]  Alu_resultHigh;

    output      MemorIOtoReg;        //1 indicates that read date from memory or I/O to write to the register
    output      MemRead;             // 1 indicates that reading from the memory to get data
    output      IORead;              // 1 indicates  I/O read
    output      IOWrite;             // 1 indicates  I/O write

    assign RegWrite = (R_format || Lw || Jal || I_format) && !(Jr) ;        // Write memory or write IO
    assign MemWrite = ((sw==1) && (Alu_resultHigh[21:0] != 22'h3FFFFF)) ? 1'b1:1'b0;
    assign MemRead = ? ? ?                  // Read memory
    assign IORead  = ? ? ?                  // Read input port
    assign IOWrite      = ? ? ?             // Write output port

// Read operations require reading data from memory or I/O to write to the register
    assign MemorIOtoReg = IORead || MemRead;
```

Practice

Complete the following modules, do the function verification:

- 1. Sigle cycle CPU
- 2. MemoryOrIO
- 3. Controller+
- 4. Sigle cycle CPU with I/O process