



# Computer Organization

---

## Lab11 CPU Design(3)

‘single’ cycle CPU,  
clock, I/O(MMIO)



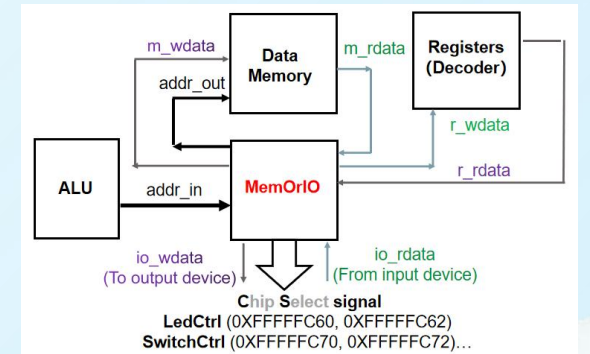
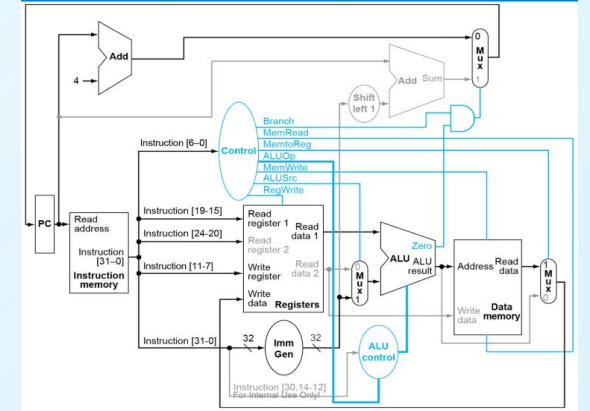
# Topic

## ➤ CPU Design(3)

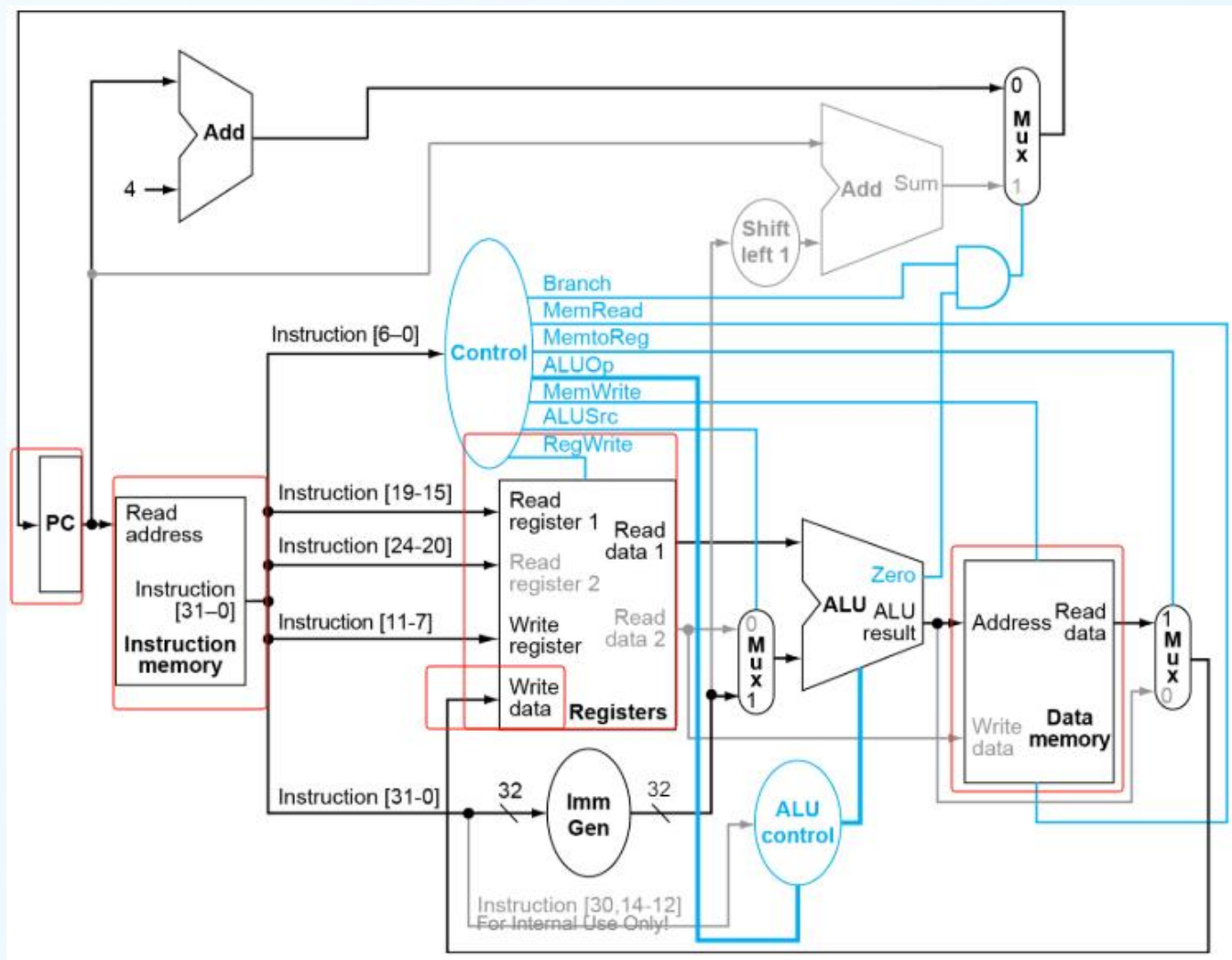
- A 'single' cycle CPU
- Clock (IP core)

## ➤ CPU work with I/O device

- MMIO(Memory-Mapped Input Output)
- Modification on CPU
  - CPU TOP(add new input and output port)
  - MemOrIO
  - Controller +



# A 'single' cycle CPU



Q1. Does it **take time** for signals to be processed and transmitted within the module, as well as between modules?

Q2. Which sub modules within CPU **need the trigger from the clock**? When does the following event occur in a clock cycle?

1-1) **IFetch: update** the value of PC register  
1-2) **IFetch: fetch** the instruction according to the value of PC

2-1-1) **Controller: generate the control signals**  
2-2-1) **Decoder: get** the value of register(s)  
2-2-2) **Decoder: generate** the extended immediate

3-1) **ALU: get** the operands  
3-2) **ALU: generate** the calculation result

4-1) **Dmemory: get** the **address**(from ALU) and **data**(from Decoder)  
4-2) **Dmemory: read out** the data

5-1) **Decoder: write back** the data

# Clock

The clock on the Minisys/EGO1 development board is **100Mhz** , is the **100Mhz** (cycle **10ns**) clock work ok for the 'single cycle' CPU?

**Solution:** testing by vivado:

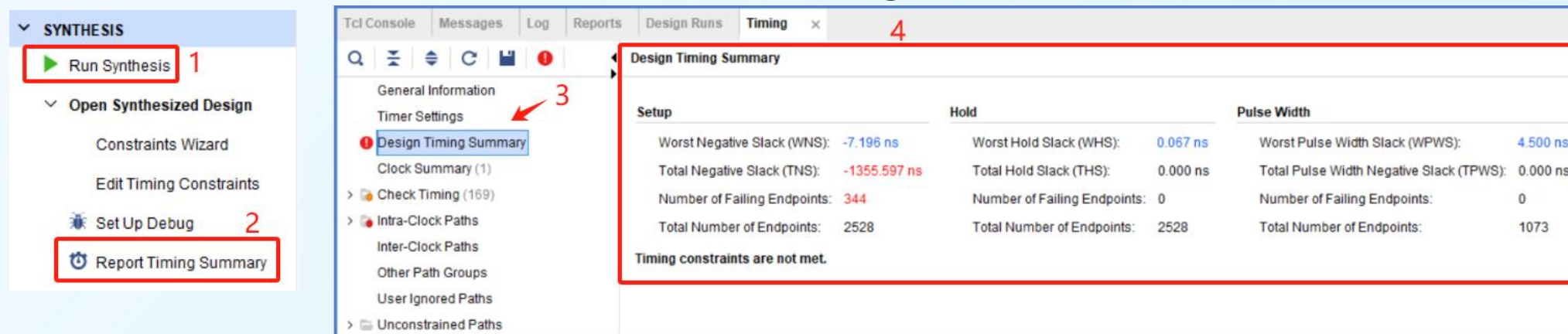
✓ step1. **add a constraint description on clock to the constraint file to test**

```
module CPUv1(  
input clk, rst  
);
```

```
create_clock -period 10.000 -name clk [get_ports clk]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports clk]  
set_property IOSTANDARD LVCMOS33 [get_ports rst]  
set_property PACKAGE_PIN P17 [get_ports clk]  
set_property PACKAGE_PIN P15 [get_ports rst]  
create_clock -period 10.000 -name clk [get_ports clk]
```

✓ step2. **run synthesis, then chek the Report Timing Summary in vivado**



The image shows two screenshots from the Vivado IDE. The left screenshot shows the 'SYNTHESIS' tab with 'Run Synthesis' (1) and 'Report Timing Summary' (2) highlighted. The right screenshot shows the 'Timing' tab with the 'Design Timing Summary' report (3) displayed. The report shows timing violations, including a Worst Negative Slack (WNS) of -7.196 ns and a Total Negative Slack (TNS) of -1355.597 ns. A red box highlights the 'Timing constraints are not met.' message.

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): -7.196 ns	Worst Hold Slack (WHS): 0.067 ns	Worst Pulse Width Slack (WPWS): 4.500 ns
Total Negative Slack (TNS): -1355.597 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 344	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 2528	Total Number of Endpoints: 2528	Total Number of Endpoints: 1073

Timing constraints are not met.

➤ **100Mhz is too fast for a 'single' clock CPU**(Worst Negative slack(WNS): Worst path violation, a negative value means design may not run, Setup Violation Usually solved by reducing logic latency or increasing clock cycles )



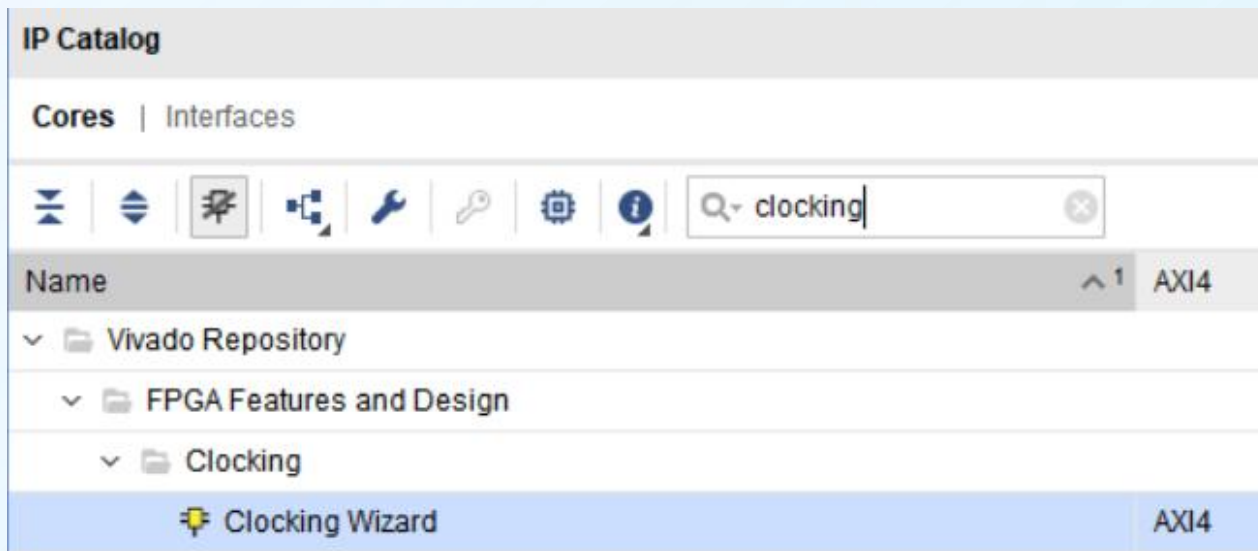


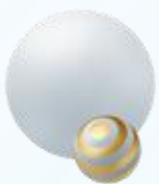
# Clock continued

➤ Add **PLL clock IP core** to generate the needed clock:

1. The clock on the Minisys/EGO1 development board is **100Mhz** *(clk\_in1)*  
➤ 100Mhz is too fast for a 'single' clock CPU

2. A clock of 23Mhz is more suitable for the 'single' clock CPU *(clk\_out1)*





# Clock continued

Custom the IP core, set its **name**, **Primitive**, **Output Freq** and **with out the reset and locked**. Then **generate** the IP core with the settings.

Component Name **cpuck**

**Clocking Options** | Output Clocks | Port Renaming | PLL2 Settings | Summary

**Clock Monitor**

☐ Enable Clock Monitoring

**Primitive**

☐ MMCM ☒ **PLL**

**Clocking Features**

☒ Frequency Synthesis ☐ Minimize Power

☒ Phase Alignment

☐ Dynamic Reconfig

☐ Safe Clock Startup

**Jitter Optimization**

☒ Balanced

☐ Minimize Output Jitter

☐ Maximize Input Jitter filtering

**Clocking Options** | **Output Clocks** | Port Renaming | PLL2 Settings

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)	
		Requested	Actual
<input checked="" type="checkbox"/> clk_out1	clk_out1	23.000	23.000

Component Name **cpuck**

**Clocking Options** | **Output Clocks** | Port Renaming | PLL2 Settings | Summary

**Enable Optional Inputs / Outputs for MMCM/PLL**

☐ **reset** ☐ power\_down

☐ **locked**

**Reset Type**

☒ Active High ☐ Active Low



# The Function Verification of “cpuck”

Functional Verification by **testbench** and **simulator**

1) Create a verilog **testbench** module to instance the IP core “**cpuck**” and bind its ports. set the frequency of the input on “cpuck” as **100Mhz**.

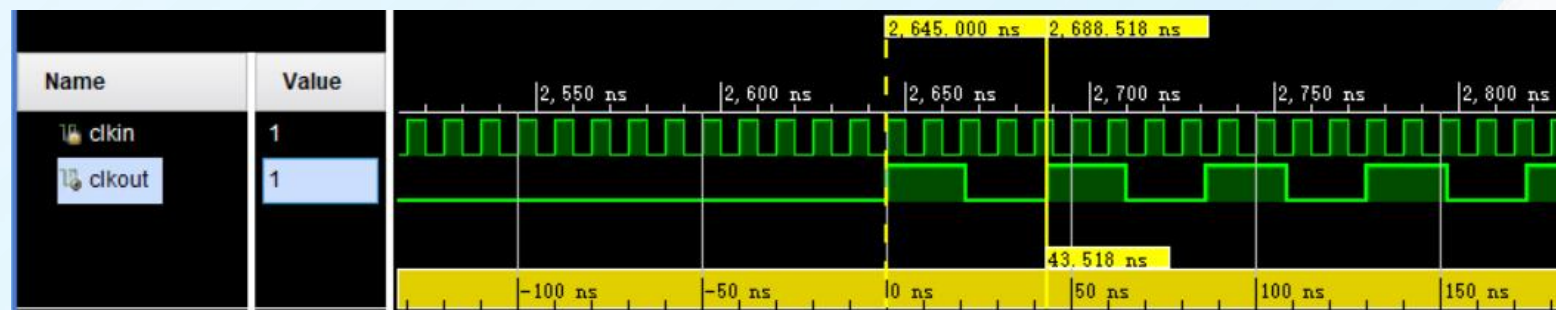
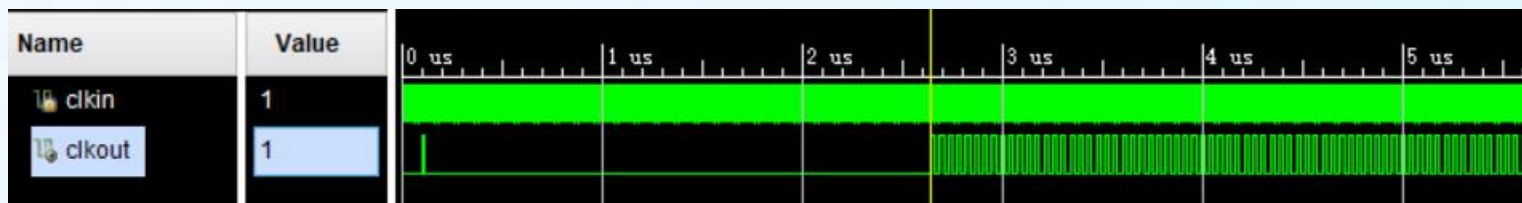
2) Do the simulation to verify whether the output signal is a **23Mhz** clock signal while the input signal is **100Mhz**.

```
module cpuck_tb( ); // a reference testbench for 'cpuck'

    reg clkkin;
    wire clkkout;
    cpuck clk1( .clk_in1(clkin), .clk_out1(clkout) );

    initial      clkkin = 1'b0;
    always #5 clkkin=~clkkin;
endmodule
```

**NOTE:** The output of IP core 'cpuck' need to work for a ‘long’ time to achieve stability.





# Test a 'single cycle' CPU

## Q. How to test the CPU ?

### Q1.how to determine the testcase and make it work on CPU?

**A1. asm file ->coe files->Instruction Memory & Data Memeory in CPU**

### Q2. how to check the testing result?

**A2-1. Check the internal signal in the CPU**

1)Do the simulation (learnt in lab10)

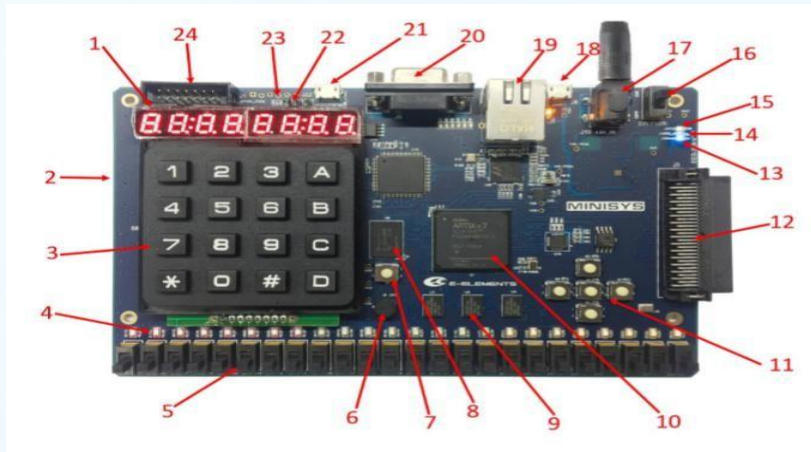
**A2-2. Make CPU work with I/O device, check the external signal of CPU**

- ✓ 1) Do the simulation
- ✓ 2) Do the testing on develop board(EGO1/Minisys) with FPGA chip embedded
  - ✓ Generate bitstream file of CPU -> 'program device' to the FPGA embed on EGO1/Minisys -> test on develop board(EGO1/Minisys) with FPGA chip embedded

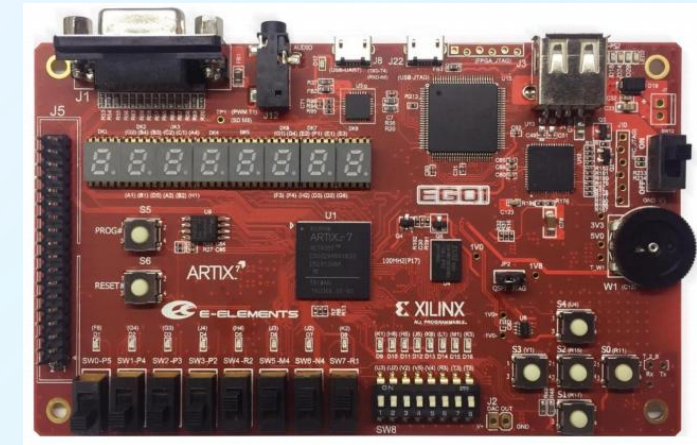


# I/O interface

Minisys board with FPGA(xc7a100tfgg484-1) chip embedded



EGO1 board with FPGA chip(xc7a35tcsg324-1) embedded



TIPS:

The handbook of board **Minisys** and **EGO1** could be found in the directory “**labs\Handbook\_of\_Minisys\_EGO1**” on the course **BlackBoard** sit

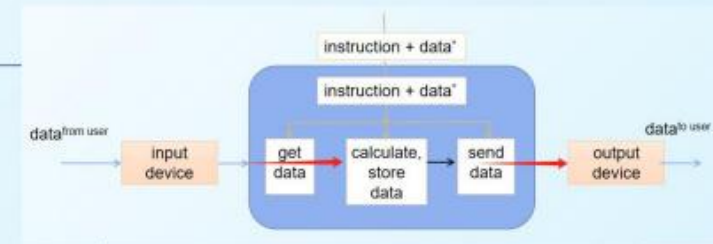
We have practiced a Cropped CPU on EGO1 in lab1, MMIO(**Memory-Mapped Input Output**) is used.

```
#assembly source file in RISC-V
.data          # data* is 4bytes, its initial value is 0
    buf: .word 0x0000

.text          # instructions
main:
    #in the 'tailored General purpose processor', the value in register x31 is 0xFFC0.

    lw x1, 8(x31)    #copy data from 0xFFC8(switches) to register x1
    sw x1, 4(x31)    #copy data from register x1 to 0xFFC4(leds)

    j main          # jump to the instructions labled by main
```





# A Simple Design on the I/O Interface

This part mainly accomplishes the following work:

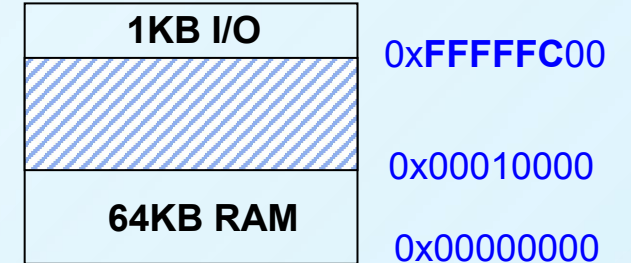
1. Add I/O function
2. 16-bit LED design
3. 16-bit DIP Switch design

This is only one of the design solutions for I/O related data bus. **Please develop a solution that suits your design needs!**

# MMIO: 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 **Select** 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	0xFFFFFC60	0xFFFFFC62	0xFFFFFC70	0xFFFFFC72

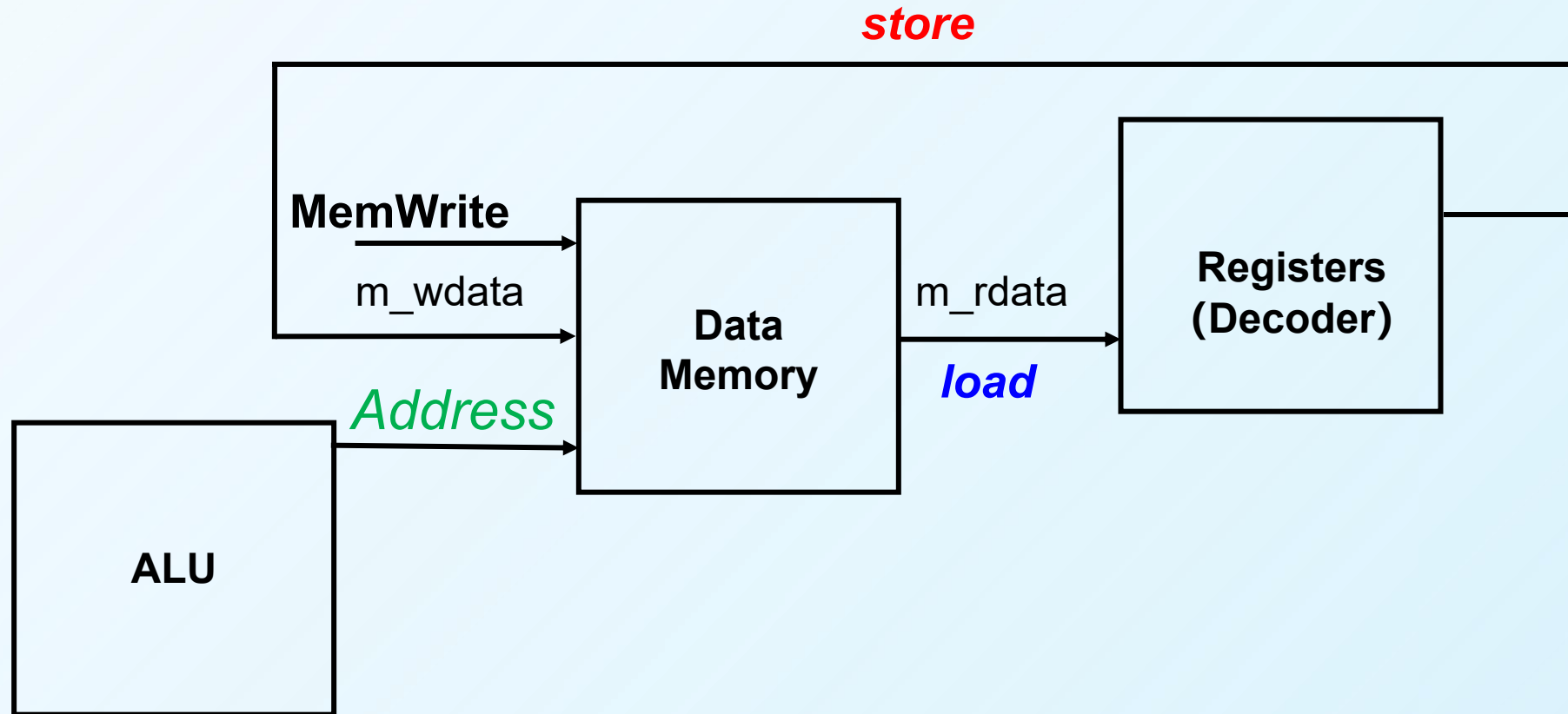
## Note:

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 (MMIO: Memory-Mapped Input Output) , but not the only choice.



# Corresponding Operation of LW/SW



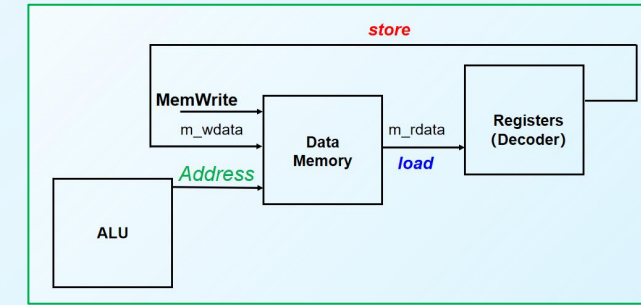
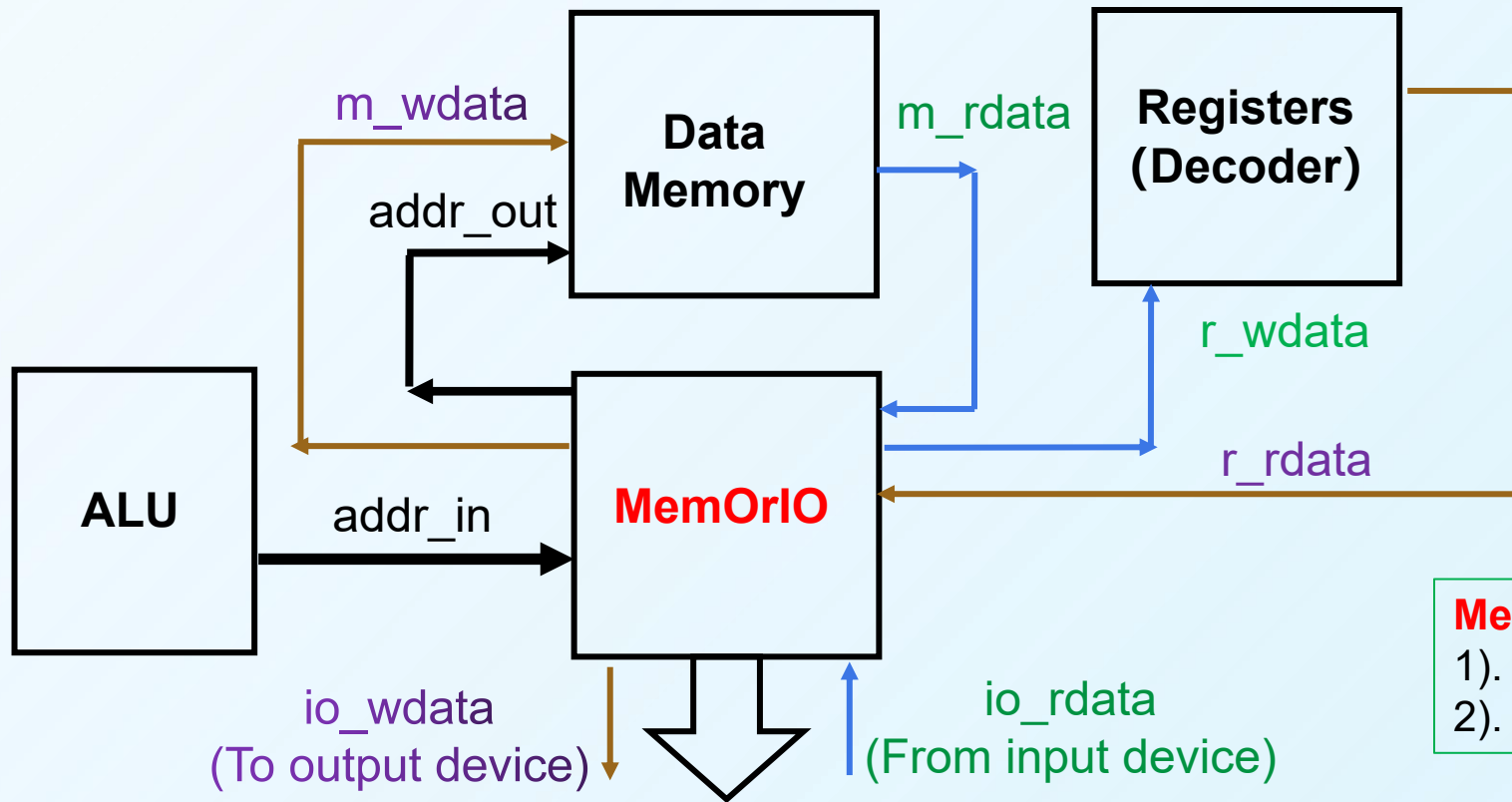
NOTE:

- 1) There is no specific instruction in **MiniRISC-VI**(**lw**, **sw**, **add**, **sub**, **and**, **or**, **beq**) 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 lw/sw instructions** in **MiniRISC-VI**.





# MemOrIO

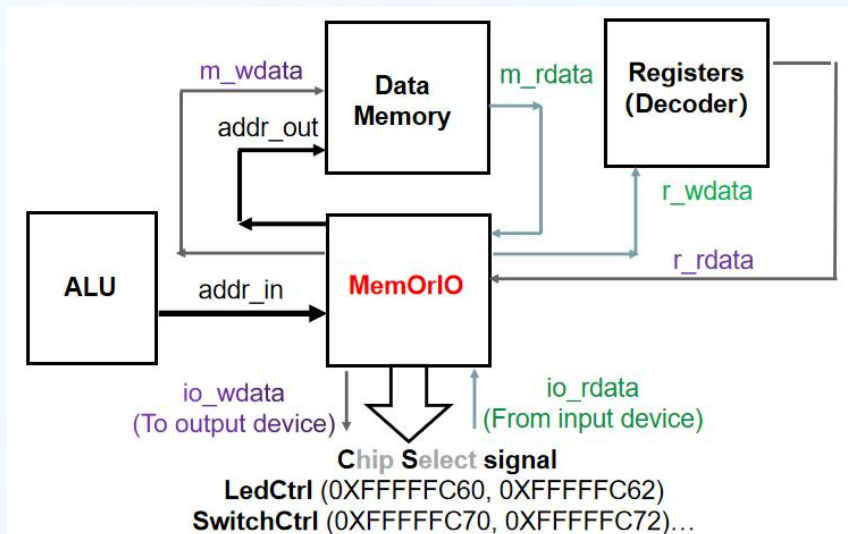


- MemOrIO** determine:
- 1). The source of **r\_wdata**
  - 2). The destination of **r\_rdata**

**Chip Select signal**  
**LedCtrl** (0xFFFFFC60, 0xFFFFFC62)  
**SwitchCtrl** (0xFFFFFC70, 0xFFFFFC72)...



# 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

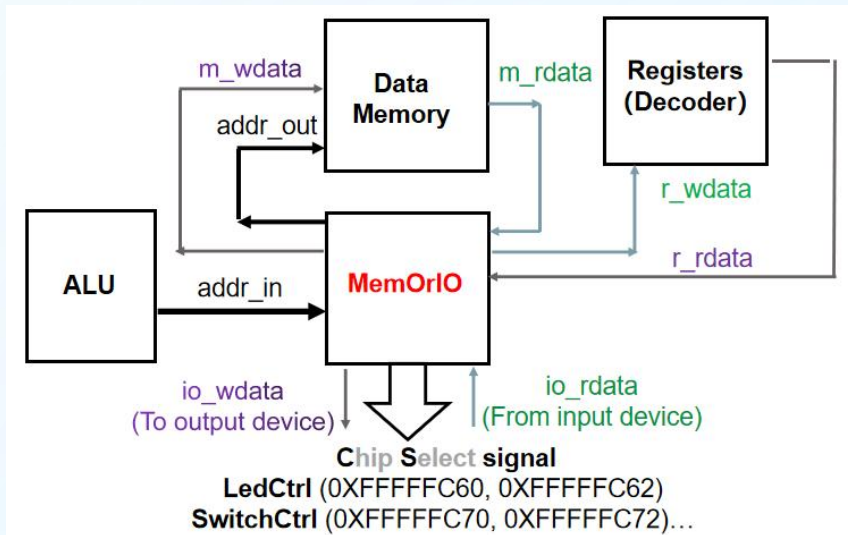
input[31:0] m_rdata;                        // data read from Data-Memory
input[15:0] io_rdata;                      // data read from IO, 16 bits
output[31:0] r_wdata;                      // data to Decoder(register file)

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
```

*Tips: A demo about how the **Chip Select** signals work on I/O could be found in **labs/lab11\_io** on course BlackBoard site*



# MemOrIO continued



```
assign addr_out= addr_in;  
// The data write 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= ? ? ?
```

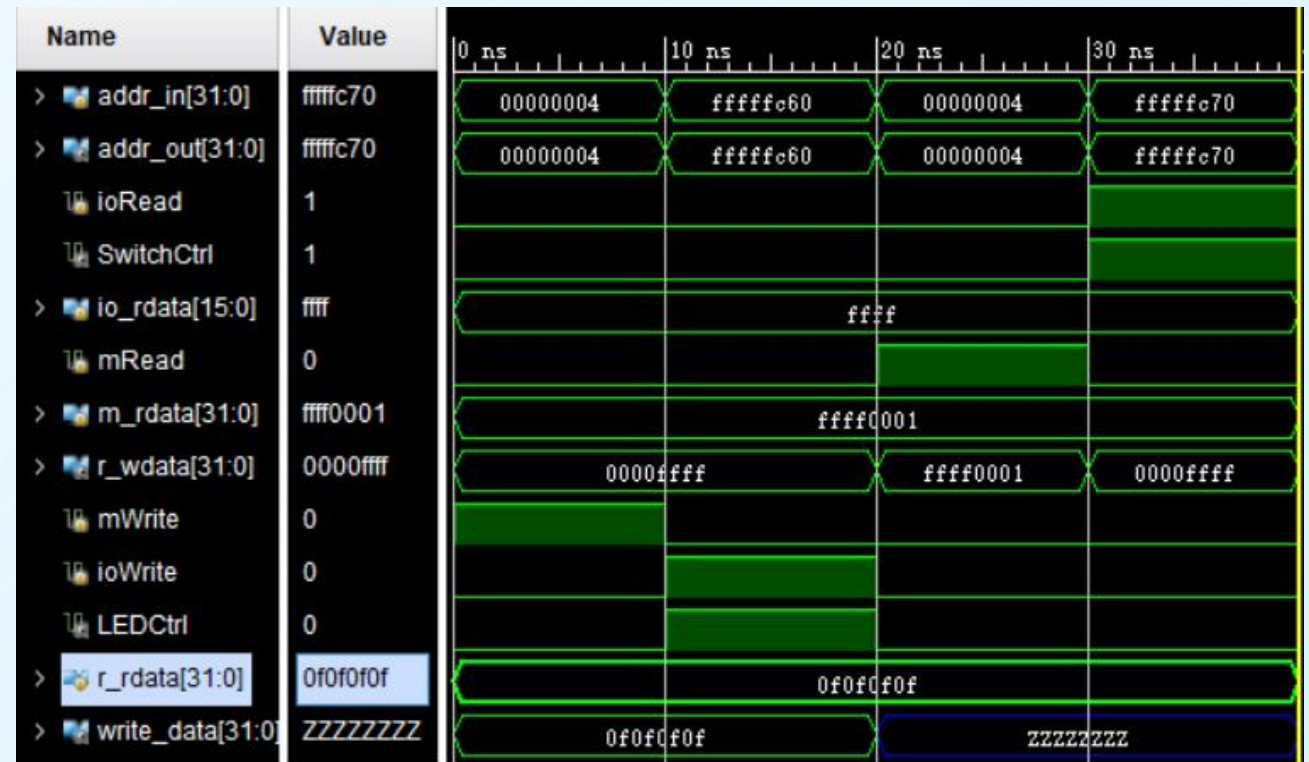
```
always @* begin  
    if((mWrite==1)||(ioWrite==1))  
        //write_data could go to either memory or IO. where is it from?  
        write_data = ? ? ?  
    else  
        write_data = 32'hZZZZZZZZ;  
end  
endmodule
```

# 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);
```



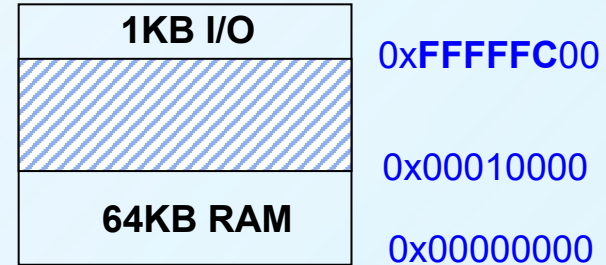
```
initial begin // r_rdata -> m_wdata(write_data)
    m_rdata = 32'h0xffff_0001; io_rdata = 16'h0xffff; r_rdata = 32'h0x0f0f_0f0f; addr_in = 32'h4;{mRead,mWrite,ioRead,ioWrite}= 4'b01_00;
    #10 addr_in = 32'hffff_fc60; {mRead,mWrite,ioRead,ioWrite}= 4'b00_01; // r_rdata -> io_wdata(write_data)
    #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 data 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 = ???      // Write memory or write IO
assign MemWrite = ((sw==1) && (Alu_resultHigh[21:0] != 22'h3FFFFFF)) ? 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

P1-1. Do the functional verification on the module `cpuclk`(which is introduced on the first part of this lab)

P1-2. Answer the Q2 on page 3 of this lab slides.

P2. Complete the following modules, do the function verification:

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

P3. Redesign and implement the solution about I/O data bus and I/O addressing that are suitable for your design. Build the single cycle CPU with the updated solution of I/O process and do the function verification.