

# 计算机组成原理 CPU

---

## CPU功能及使用说明

---

### 一、详细信息

CPI: 1

单周期

复位信号: sys\_rst\_n - 按键P2

接口:

INPUT

时钟: sys\_clk

UART: rx, start\_pg

七段数码管: control[0~7], cube\_data[0~7]

VGA: vga\_hs, vga\_vs

拨码开关: sw\_input[0~23]

OUTPUT

UART: tx

VGA: vga\_rgb[0~11]

LED: led[0~23]

支持uart

支持VGA

端口绑定

```
set_property PACKAGE_PIN P1 [get_ports sys_rst_n]
set_property PACKAGE_PIN Y18 [get_ports sys_clk]

set_property PACKAGE_PIN K17 [get_ports {led[23]}]
set_property PACKAGE_PIN L13 [get_ports {led[22]}]
set_property PACKAGE_PIN M13 [get_ports {led[21]}]
set_property PACKAGE_PIN K14 [get_ports {led[20]}]
```

```
set_property PACKAGE_PIN K13 [get_ports {led[19]}]
set_property PACKAGE_PIN M20 [get_ports {led[18]}]
set_property PACKAGE_PIN N20 [get_ports {led[17]}]
set_property PACKAGE_PIN N19 [get_ports {led[16]}]
set_property PACKAGE_PIN M17 [get_ports {led[15]}]
set_property PACKAGE_PIN M16 [get_ports {led[14]}]
set_property PACKAGE_PIN M15 [get_ports {led[13]}]
set_property PACKAGE_PIN K16 [get_ports {led[12]}]
set_property PACKAGE_PIN L16 [get_ports {led[11]}]
set_property PACKAGE_PIN L15 [get_ports {led[10]}]
set_property PACKAGE_PIN L14 [get_ports {led[9]}]
set_property PACKAGE_PIN J17 [get_ports {led[8]}]
set_property PACKAGE_PIN F21 [get_ports {led[7]}]
set_property PACKAGE_PIN G22 [get_ports {led[6]}]
set_property PACKAGE_PIN G21 [get_ports {led[5]}]
set_property PACKAGE_PIN D21 [get_ports {led[4]}]
set_property PACKAGE_PIN E21 [get_ports {led[3]}]
set_property PACKAGE_PIN D22 [get_ports {led[2]}]
set_property PACKAGE_PIN E22 [get_ports {led[1]}]
set_property PACKAGE_PIN A21 [get_ports {led[0]}]
```

```
set_property PACKAGE_PIN AB8 [get_ports {sw_input[23]}]
set_property PACKAGE_PIN AA8 [get_ports {sw_input[22]}]
set_property PACKAGE_PIN V8 [get_ports {sw_input[21]}]
set_property PACKAGE_PIN V9 [get_ports {sw_input[20]}]
set_property PACKAGE_PIN Y8 [get_ports {sw_input[19]}]
set_property PACKAGE_PIN Y9 [get_ports {sw_input[18]}]
set_property PACKAGE_PIN W9 [get_ports {sw_input[17]}]
set_property PACKAGE_PIN Y7 [get_ports {sw_input[16]}]
set_property PACKAGE_PIN AB6 [get_ports {sw_input[15]}]
set_property PACKAGE_PIN AB7 [get_ports {sw_input[14]}]
set_property PACKAGE_PIN V7 [get_ports {sw_input[13]}]
set_property PACKAGE_PIN AA6 [get_ports {sw_input[12]}]
set_property PACKAGE_PIN Y6 [get_ports {sw_input[11]}]
set_property PACKAGE_PIN T6 [get_ports {sw_input[10]}]
set_property PACKAGE_PIN R6 [get_ports {sw_input[9]}]
set_property PACKAGE_PIN V5 [get_ports {sw_input[8]}]
set_property PACKAGE_PIN U6 [get_ports {sw_input[7]}]
set_property PACKAGE_PIN W5 [get_ports {sw_input[6]}]
set_property PACKAGE_PIN W6 [get_ports {sw_input[5]}]
set_property PACKAGE_PIN U5 [get_ports {sw_input[4]}]
set_property PACKAGE_PIN T5 [get_ports {sw_input[3]}]
set_property PACKAGE_PIN T4 [get_ports {sw_input[2]}]
set_property PACKAGE_PIN R4 [get_ports {sw_input[1]}]
set_property PACKAGE_PIN W4 [get_ports {sw_input[0]}]
```

```
set_property PACKAGE_PIN F15 [get_ports {cube_data[0]}]
set_property PACKAGE_PIN F13 [get_ports {cube_data[1]}]
set_property PACKAGE_PIN F14 [get_ports {cube_data[2]}]
set_property PACKAGE_PIN F16 [get_ports {cube_data[3]}]
set_property PACKAGE_PIN E17 [get_ports {cube_data[4]}]
set_property PACKAGE_PIN C14 [get_ports {cube_data[5]}]
set_property PACKAGE_PIN C15 [get_ports {cube_data[6]}]
set_property PACKAGE_PIN E13 [get_ports {cube_data[7]}]
set_property PACKAGE_PIN C19 [get_ports {control[0]}]
set_property PACKAGE_PIN E19 [get_ports {control[1]}]
set_property PACKAGE_PIN D19 [get_ports {control[2]}]
set_property PACKAGE_PIN F18 [get_ports {control[3]}]
```

```

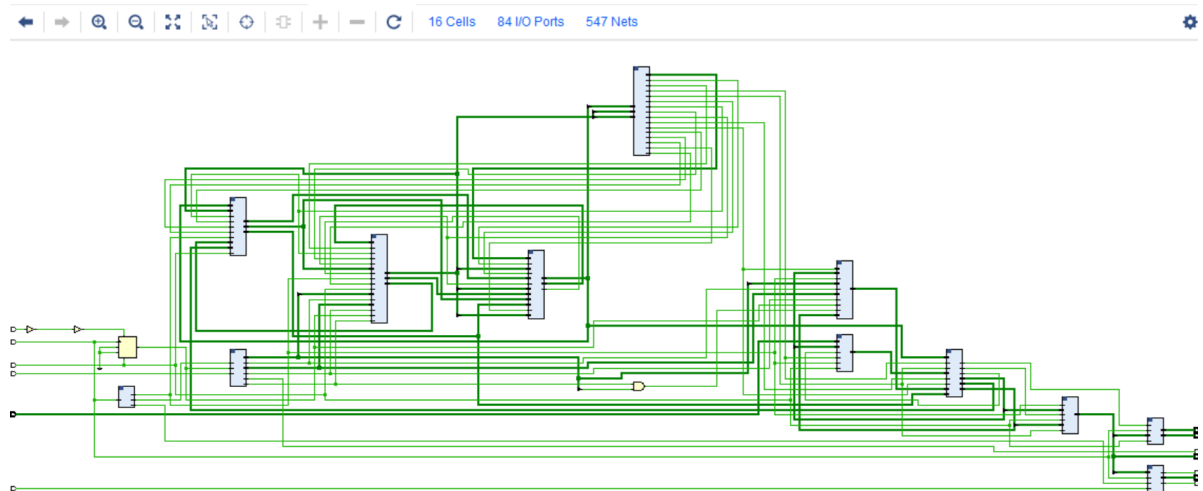
set_property PACKAGE_PIN E18 [get_ports {control[4]}]
set_property PACKAGE_PIN B20 [get_ports {control[5]}]
set_property PACKAGE_PIN A20 [get_ports {control[6]}]
set_property PACKAGE_PIN A18 [get_ports {control[7]}]
set_property PACKAGE_PIN Y19 [get_ports rx]
set_property PACKAGE_PIN V18 [get_ports tx]
set_property PACKAGE_PIN P20 [get_ports start_pg]

set_property PACKAGE_PIN M21 [get_ports vga_hs]
set_property PACKAGE_PIN L21 [get_ports vga_vs]
set_property PACKAGE_PIN H20 [get_ports {vga_rgb[0]}]
set_property PACKAGE_PIN G20 [get_ports {vga_rgb[1]}]
set_property PACKAGE_PIN K21 [get_ports {vga_rgb[2]}]
set_property PACKAGE_PIN K22 [get_ports {vga_rgb[3]}]
set_property PACKAGE_PIN H17 [get_ports {vga_rgb[4]}]
set_property PACKAGE_PIN H18 [get_ports {vga_rgb[5]}]
set_property PACKAGE_PIN J22 [get_ports {vga_rgb[6]}]
set_property PACKAGE_PIN H22 [get_ports {vga_rgb[7]}]
set_property PACKAGE_PIN G17 [get_ports {vga_rgb[8]}]
set_property PACKAGE_PIN G18 [get_ports {vga_rgb[9]}]
set_property PACKAGE_PIN J15 [get_ports {vga_rgb[10]}]
set_property PACKAGE_PIN H15 [get_ports {vga_rgb[11]}]

```

## 二、顶层模块

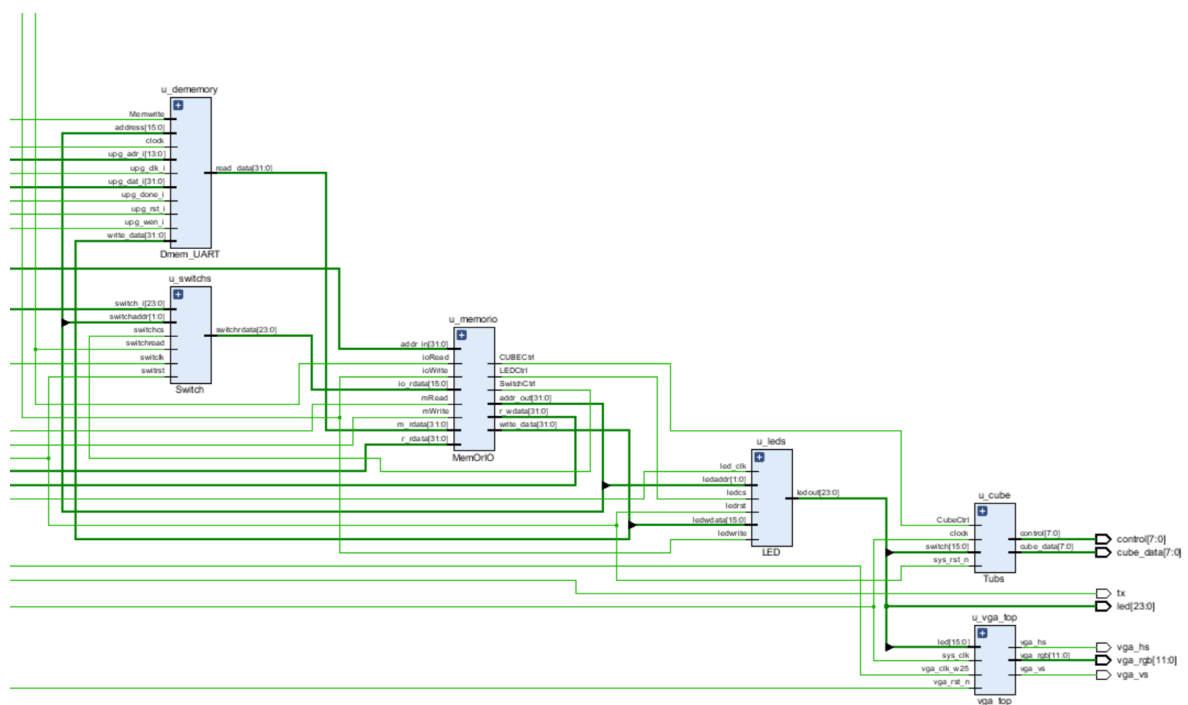
### 总览图



### 左下方:

由输入端口及时钟分频，复位信号处理与构成





### 三、子模块的设计说明

#### I. 外设模块：

功能：实现将从输入（uart、拨码开关）得到的数据，送予CPU处理，并将CPU得到的数据由外设（数码管、LED、显示器）输出

#### VGA:

##### VGA\_top

——VGA模块的顶层文件

VGA\_hs: 行同步时序

VGA\_vs: 场同步时序

VGA\_rgb[11:0]: 输出到显示器上的颜色

##### VGA\_driver

——驱动VGA显示，设置分辨率、行场后沿、坐标等参数

pixel\_data: 颜色数据

H\_DISP: 行像素点

V\_DISP: 列像素点

时钟：由于分辨率为640\*480，时钟频率须为25MHZ，敏感于上升沿

## VGA\_display

——处理从CPU得到的数据，并将对应颜色传递到VGA\_driver模块中显示

pixel\_xpos: 行像素颜色数据

pixel\_ypos: 列像素颜色数据

时钟: 由于分辨率为640\*480, 时钟频率须为25MHZ, 敏感于上升沿

## UART:

——从电脑或其他设备读取数据作为输入交由CPU处理, 并可以将CPU得到的数据作为输出发送到其他设备

upg\_wen\_o: uart写使能信号

upg\_done\_o: 数据接收完成信号

upg\_dat\_o: uart输出数据

rx: 接受数据端口

tx: 发送数据端口

时钟: 10MHZ

## 开关:

switchclk: 时钟信号

switrst: 复位信号

switchcs: 从memorio来的switch片选信号

switchaddr: 到switch模块的地址低端

switchrea: 读信号

[15:0] switchrdata: 送到CPU的拨码开关值注意数据总线只有16根

[23:0] switch\_i: 从板上读的24位开关数据

## 数码管:

CubeCtrl: 控制数码管是否显示

switch: 输入数据

control, cube\_data: 七段数码管输出信号

时钟须分频, 敏感于上升沿

## LED:

led\_clk: 时钟信号

ledrst: 复位信号

ledwrite: 写信号

ledcs: 从memorio来的LED片选信号 !!!!!!!!!!!!!!!

ledaddr: 到LED模块的地址低端 !!!!!!!!!!!!!!!

ledwdata: 写到LED模块的数据, 注意数据线只有16根

ledout: 向板子上输出的24位LED信号

## II. CPU模块:

### Decoder

——读取，写入寄存器

output[31:0] Read_data_1	寄存器1
output[31:0] Read_data_2	寄存器2
output[31:0] Imme_extend	32位拓展立即数
input [31:0] Instruction	指令
input [31:0] read_data	从内存或端口读到的数据
input [31:0] ALU_Result	ALU得到的结果写回到寄存器
input Jal, RegWrite, MemOrIOtoReg, RegDst	控制信号
input clock, reset,	
input [31:0] opcplus4	JAL连接地址

寄存器读写敏感于时钟上升沿

### Controller

——分析指令，生成控制信号用于其他模块

input [5:0] Opcode;	由指令产生
input [5:0] Function_opcode;	由指令产生
output RegDST, ALUSrc, RegWrite, MemWrite;, Branch, nBranch, Jr	生成的各种控制信号
output Jmp, Jal	用于jump指令
output I_format	I格式, R格式
output [1:0]ALUOp;	

```
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 MemRead; // 1 indicates that the instruction needs to read from the memory
output IORead; // 1 indicates I/O read
output IOWrite; // 1 indicates I/O write
```

组合逻辑

### ALU

——逻辑计算单元

```
// from decoder
input[31:0] Read_data_1 the source of Ainput
input[31:0] Read_data_2 one of the sources of Binput
input[31:0] Imme_extend one of the sources of Binput
// from ifetch
input[5:0] Function_opcode, instructions[5:0]
input[5:0] Opcode, instruction[31:26]
input[4:0] Shamt, instruction[10:6], the amount of shift bits
input[31:0] PC_plus_4, pc+4
// from controller
input[1:0] ALUOp
input ALUSrc, 1 means the 2nd operand is an immedite (except beq, bne)
input I_format 1 means I-Type instruction except beq, bne, LW, SW
```

input Sftmd                      1 means this is a shift instruction  
input Jr,                        1 means this is a jr instruction  
  
output Zero,                    1 means the ALU\_result is zero, 0 otherwise  
output reg [31:0] ALU\_Result    the ALU calculation result  
output[31:0] Addr\_Result        the calculated instruction address

## IFetch

——从.coe文件或串口获取指令的模块

```
output[31:0] Instruction, // the instruction fetched from this module
output[31:0] branch_base_addr, // (pc+4) to ALU which is used by branch type instruction
output reg[31:0] link_addr, // (pc+4) to decoder which is used by jal instruction
output reg[31:0] pco,
input clock,reset, // Clock and reset
// from ALU
input[31:0] Addr_result, // the calculated address from ALU
input Zero, // while Zero is 1, it means the ALUresult is zero
// from Decoder
input[31:0] Read_data_1, // the address of instruction used by jr instruction
// from controller
input Branch, // while Branch is 1,it means current instruction is beq
input nBranch, // while nBranch is 1,it means current instruction is bnq
input Jmp, // while Jmp 1,it means current instruction is jump
input Jal, // while Jal is 1,it means current instruction is jal
input Jr, // while Jr is 1,it means current instruction is jr
// UART Programmer Pinouts
input upg_rst_i, // UPG reset (Active High)
input upg_clk_i, // UPG clock (10MHz)
input upg_wen_i, // UPG write enable
input[13:0] upg_adr_i, // UPG write address
input[31:0] upg_dat_i, // UPG write data
input upg_done_i // 1 if program finished
```

读取指令时敏感于时钟下降沿

## Data memory

——内存，用来存储数据，可读可写

```
input clock,
input [0:0] Memwrite ,
input [15:0]address,
input [31:0]write_data,
output [31:0]read_data,
// UART Programmer Pinouts
input upg_rst_i, // UPG reset (Active High)
input upg_clk_i, // UPG ram_clk_i (10MHz)
input upg_wen_i, // UPG write enable
input [13:0] upg_adr_i, // UPG write address
input [31:0] upg_dat_i, // UPG write data
input upg_done_i // 1 if programming is finished
```



# MemOrData

---

——选择读取内存中的数据或输入端口数据；选择将写入内存或由输出端口输出

```
input mRead; // read memory, from control32
input mWrite; // write memory, from control32
input ioRead; // read IO, from control32
input ioWrite; // write IO, from control32
input[31:0] addr_in; // from alu_result in executs32
output[31:0] addr_out; // address to memory

input[31:0] m_rdata; // data read from memory
input[15:0] io_rdata; // data read from io, 16 bits
output reg[31:0] write_data; // data to memory or I/O???m_wdata, io_wdata???

output reg [31:0] r_wdata; // data to idecode32(register file)
input[31:0] r_rdata; // data read from idecode32(register file)

output CUBEctrl;
output LEDctrl; // LED Chip Select
output Switchctrl; // Switch Chip Select
```

## 四、 问题及总结

---

在实现cpu功能和vga和uart扩展功能时我们遇见了许多困难和bug，但是最后都被我们一一解决了，最终实现了完整功能的project。

我们遇见的第一大问题就是如何通过写mips汇编程序来设计cpu，在明白了led显示便是向led对应地址sw数据后我们解决了这个问题，在仿真文件中对应led的寄存器已经能存放输出的数据了。但是仍然不能在开发板上正确显示，我们认为是频率的问题，于是引入了pll来更改始终频率，可是仍然不能显示。

于是我们在汇编文件中增加空语句循环，并且去除了pll，但是仍然不能显示。最后我们同时使用pll和空循环终于能让开发板正确显示了并且将变化周期调到了一秒中。后来我们发现了数据在不同测试场景下并不能共通，我们通过优化代码解决了这个问题，减少了通过读取内存获取数据的操作，优化了代码流程，提高效率和简洁度。

在七段数码管的显示中，我们最初将cpu和七段数码管直接对接，然后发现不能显示，最后我们发现了问题，将输出给led的数据同时输出给七段数码管而并不是将cpu的数据输出给七段数码管，这样既解决了七段数码管不能显示的问题又能保证七段数码管和led灯同时显示同样的内容，我们在后续设计vga显示的时候也是这样设计的。

我们在uart和vga设计中也遇见了很多问题。由于我们在数字逻辑课程中也使用到了uart和vga内容于是我们便借鉴了上学期project写的内容，但是却因此也遇到了一些问题。在uart的参数设定上，我们一开始使用的是上学期的波特率9600，在调试中却一直不能正确连接，后来才发现了老师给的代码的波特率为12800，调整后正常运行，vga显示中我们建立了显示模块和输入数据之间的连接，使得vga能用黑白条的方式来显示每一个bit。最终我们将vga，七段数码管，led三种io输出同时显示，并且可以通过uart来传输24bit数据显示。

最后诚挚感谢我和队友两人的互相理解和配合。没有任何一个人，我们不可能实现这个project，也不可能基础功能实现后还继续努力完成连接两个学期project的想法，实现最后的成品，感谢一年的陪伴，再见Verilog!

---

**Farewell ViVado!**

---