



Computer Organization

Lab12 CPU4 work with Uart



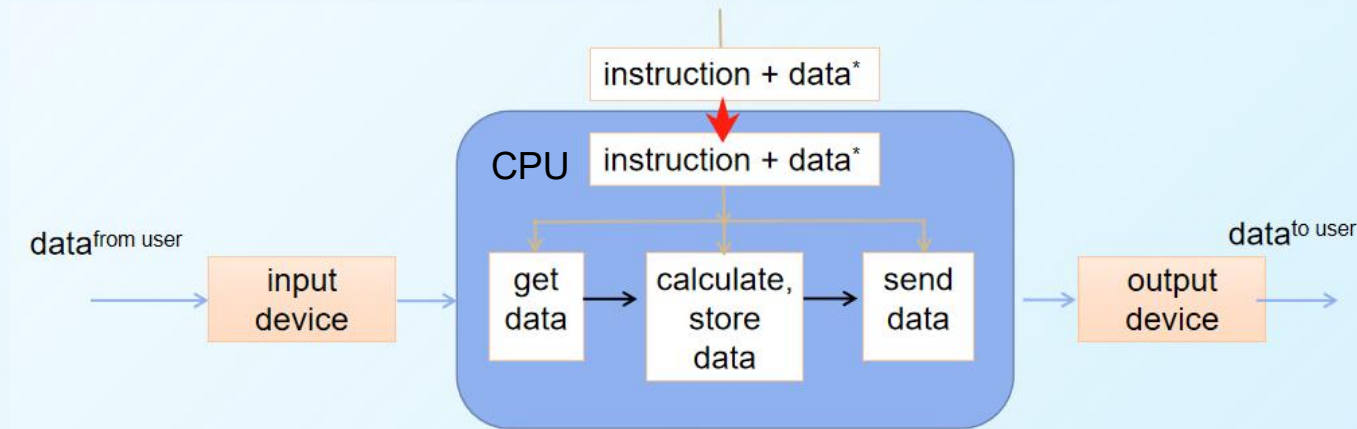
Load program on CPU



Topic

Load program on CPU

- Re-program the FPGA chip with updated bitstream file
- (*) No need to reprogram the FPGA chip, obtain the updated coe file through the Uart port and distribute it to CPU





How to make CPU work on a new program?

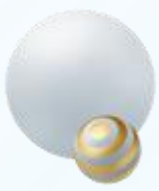
➤ new program -> new machine code and initial data (new coe file(s))

➤ **Solution1:**

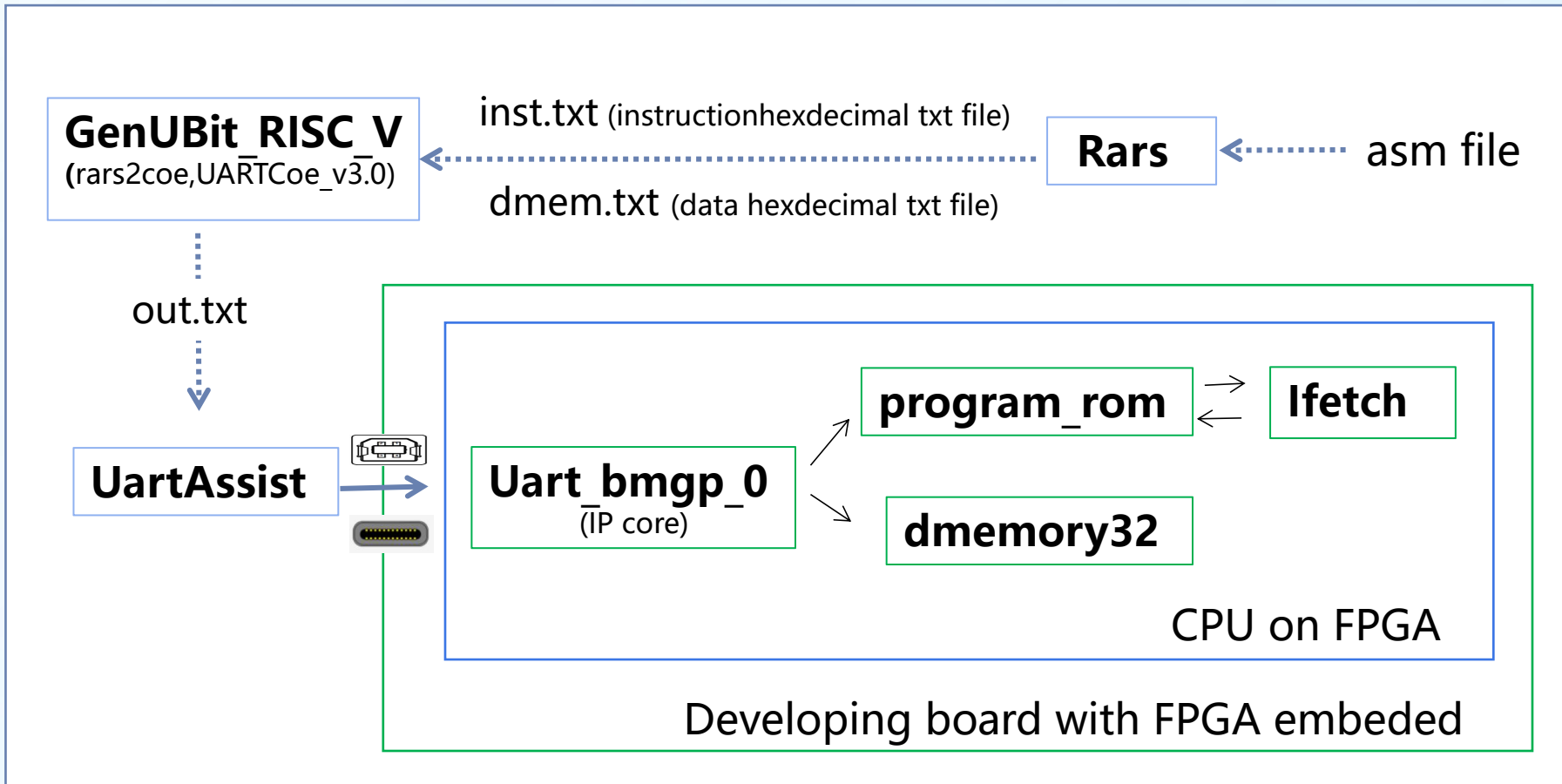
- 1) update the **ProgramRom** and **DataRAM** of CPU with new coe file(s) (re-generate output products of IP cores (**ProgramRom** and **DataRAM**) with the updated coe file(s))
- 2) re-generate bitstream of updated CPU
- 3) re-program the FPGA chip by updated bitstream file of CPU

➤ **Solution2:** (Needs Uart tools and modifications on CPU)

- 1) Set CPU work on **Communicate mode**: CPU get the new coe file(s) by uart port, then rewrite its '**ProgramROM**' and **DataRAM**
- 2) Set CPU work on **Normal mode**: work on the updated program



Solution2 on load program on CPU



"GenUBit_RISC_V" ("rars2coe" and "UARTCoe_v3.0") and "UartAssist" could be found in the "uart_tools.rar" of "tools" on BlackBoard site

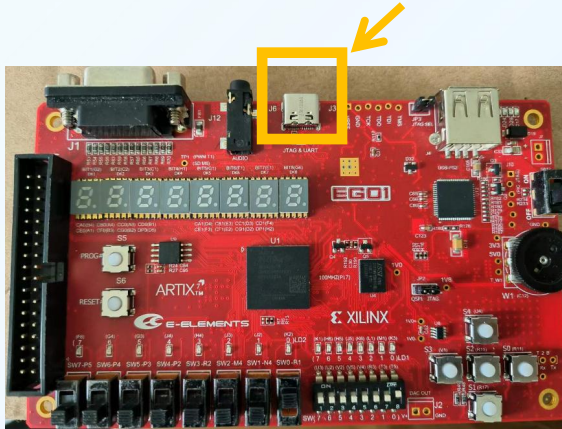
1. Modification on CPU:

- 1-1. new ports
(communication port, input port to switch the work mode)
- 1-2. two work mode:
 - 1. communication mode
(load new program on CPU)
 - 2. normal mode(execute new program)

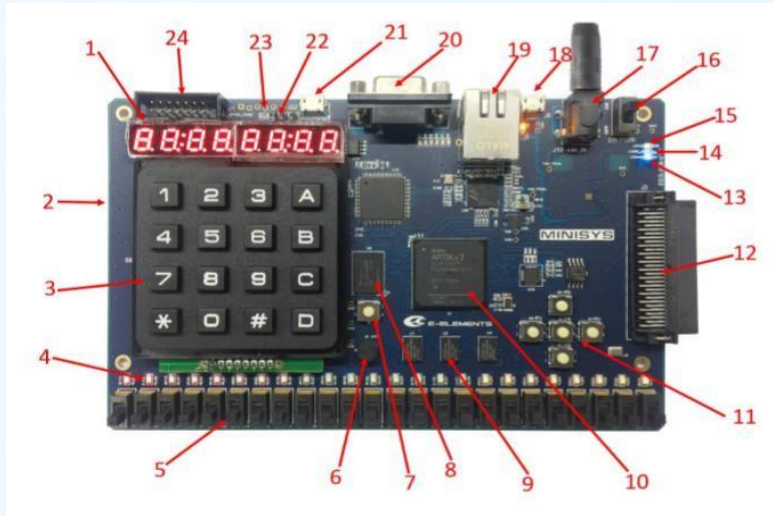
2. Tools:

- 2-1. Tools to change new program(on RISC-V) to file which is suitable for communication
- 2-2. Communication tools to send the file generated on step2-1 to the CPU which is work on communication mode.

Uart Interface work as communication port



For **EGO1 board**, USB_Jtag and USB(**typeC**) to UART interface share the same port.



For **Minisys board(new version)**, the type of USB_Jtag interface is **typeC**, USB_Jtag and USB to UART interface share the same port.

For **Minisys board(old version)**, the type of USB_Jtag interface is **typeB**, **port 18**(as shown on the left hand) is the USB to UART interface.

The handbook of Minisys board and EGO1 board could be found in the "Handbook_of_Minisys_EGO1" of "labs" on BlackBoard site

Changes on Single Cycle CPU

1. Two working modes on the CPU

- Normal mode vs Uart Communication mode

2. A new module(Uart_bmcp_0) which works as Uart interface

3. A new clock for uart communication

4. Changes

- 3-1) CPU top:

New module, new ports, new internal connection and new logic

- 3-2) Changes on Data-memory

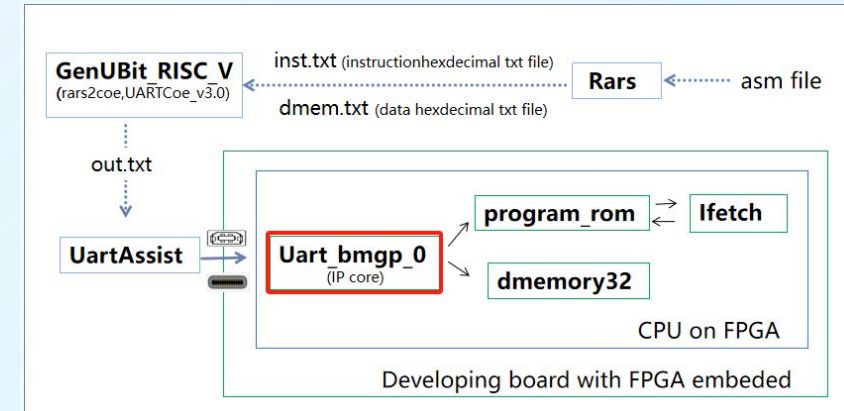
Working mode: Normal mode vs Uart Communication mode

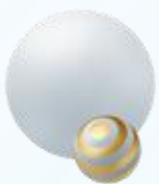
- 3-3) Changes on IFetch

- Change IP core "prgrom" from ROM to RAM

- Working mode: Normal mode vs Uart Communication mode

- Separate "prgrom" from IFetch (optional)

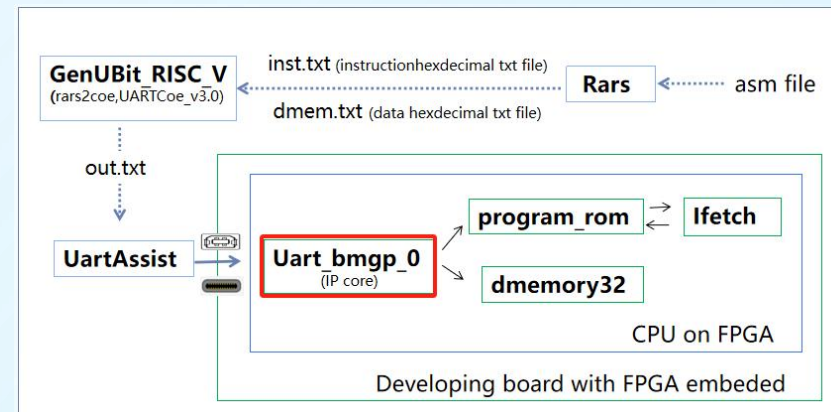




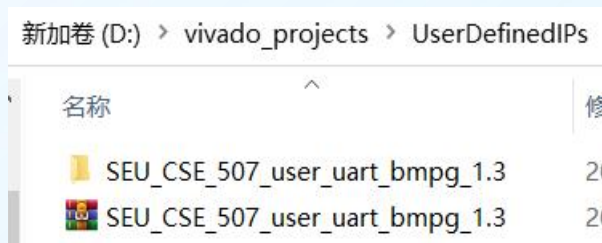
Add an IP core Which Processes Uart Data

Step1: Add the **IP core** to IP catalog(User Repository) of vivado.

- The Communication between this IP core and Uart port:
 - **Receive** data from Uart port and forward to data-memory and instruction-memory
 - **Send** data back to uart port to info that all the data has been received.

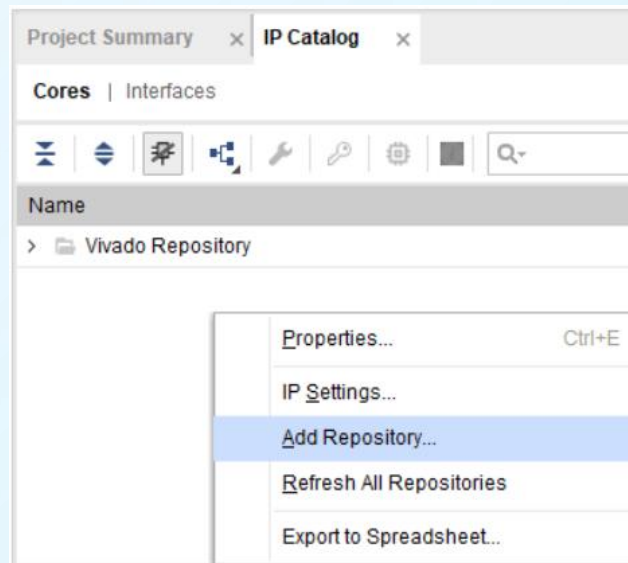


Step1-1: download the ziped IPcore file, then unzip it.

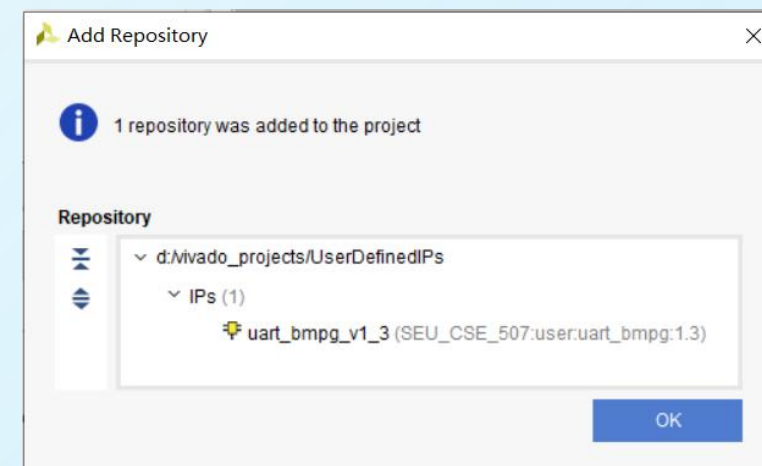


The IP core could be found in the "labs/lab12_uart" on BlackBoard site

Step1-2: open "IP Catalog" pane, right click on the blank space and select "Add Repository".



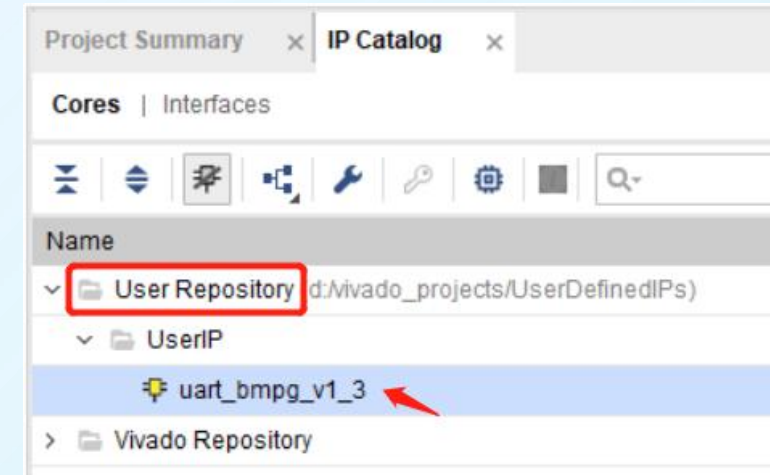
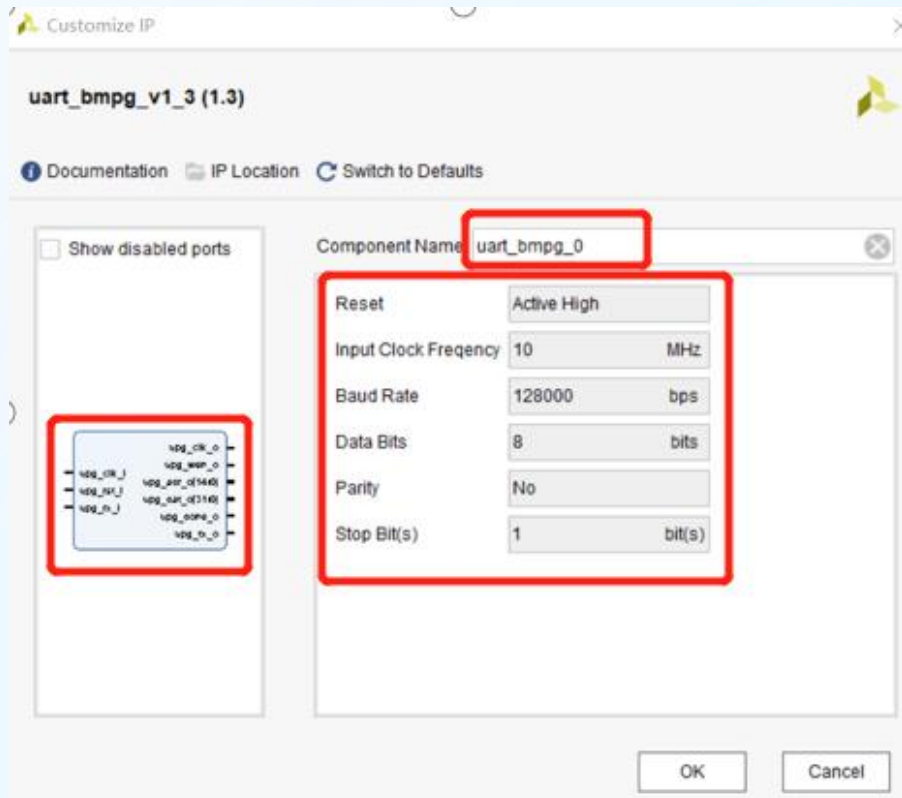
Step1-3: in "Add Repository" pane select the directory where the upzipped IPcore is placed. vivado would detect the IPcore and pop up the following prompt window, click "OK".





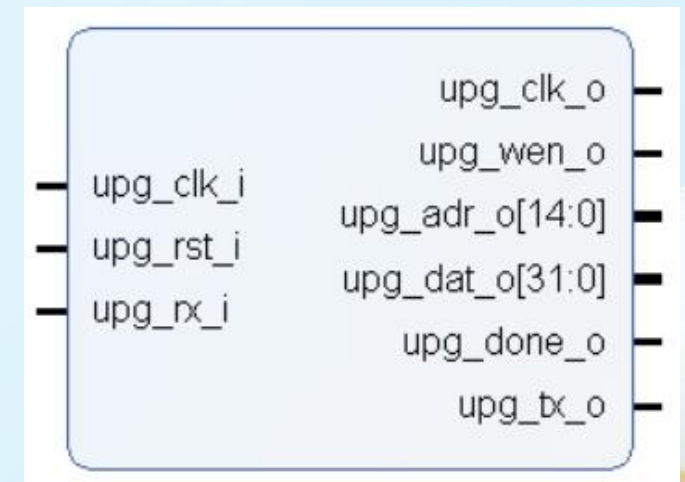
Add an IP core Which Processes Uart Data continued

Step2: Add the IP core(**uart_bmpg_0**) from IP catalog into vivado project: in “**IP catalog**” pane, the IPcore(**uart_bmpg_v1_3**) could be found in the “**User Repository**” , click it to add it to your vivado project.



NOTE:
Don't change the settings of this IP core.


While using the IPcore, **its name, features on uart communication and ports are important !**

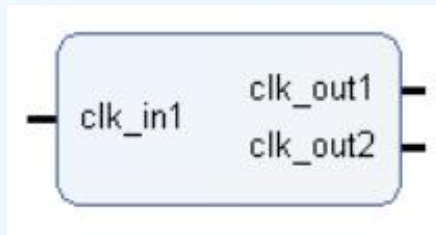




Add a New Clock For The New IP core

- Reset the “cpuck” IP core to make a new clock
 - Add a new clk_out (**clk_out2**) whose frequency is **10 Mhz** for the **IP core(uart_bmpg_0)** which is used for Uart communication(on last page)

>  cpuck : cpuck (cpuck.xci)



Component Name: cpuck

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

The phase is calculated relative to the active input clock.

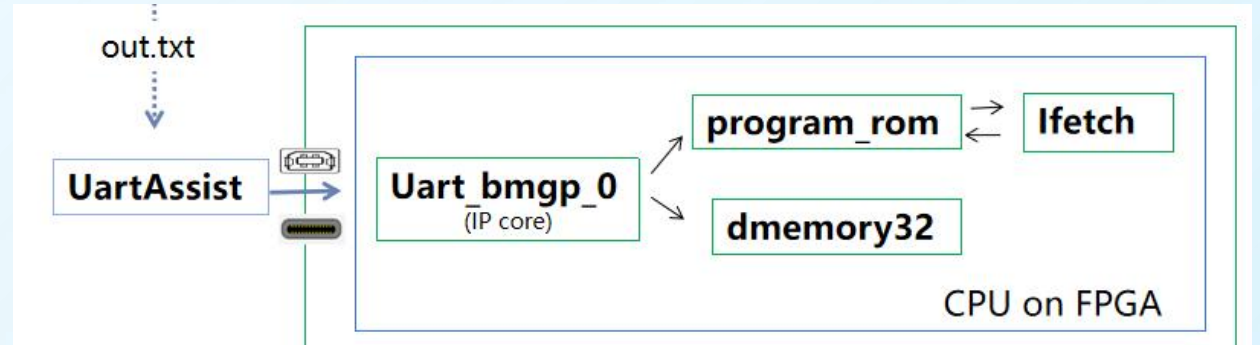
Output Clock	Port Name	Output Freq (MHz)		Phase (deg)
		Requested	Actual	
<input checked="" type="checkbox"/> clk_out1	clk_out1	23.0	23.000	0.000
<input checked="" type="checkbox"/> clk_out2	clk_out2	10.0	10.000	0.000

Handwritten red annotations: "single_cycle_cpu_clk" with an arrow pointing to the 23.0 MHz value, and "uart clk" with an arrow pointing to the 10.0 MHz value.

NOTE: The 23MHz(single_cycle_cpu_clk) is just for a single cycle CPU demo, and this value needs to be adjusted according to actual design requirements and implements.

Changes on CPU Top Module

```
module CPU_TOP(  
    input  fpga_rst,    /*Active High */  
    input  fpga_clk,  
  
    input[23:0]  switch2N4,  
    output[23:0] led2N4,  
  
    /* UART Programmer Pinouts  
    / start Uart communicate at high level */  
    input  start_pg,    /* Active High*/  
    input  rx,          /* receive data by UART*/  
    output tx          /* send data by UART*/  
);
```



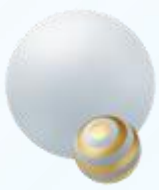
```
// For Minisys, the package_pin relationship in the constraints file  
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN Y19} [get_ports rx]  
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN V18} [get_ports tx]  
  
// For EGO1, the package_pin relationship in the constraints file  
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN T4} [get_ports tx]  
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN N5} [get_ports rx]
```

NOTE: There are errors about the description on pins of uart's rx and tx port on the EGO1's handbook, please use the **T4** to band with **uart's tx** port, use **N5** to band with **uart's rx** port.

Here the usage of "fpga_rst", "start_pg" and the bitwidth of IO are only one type of implements, not the request.

The **Y19**(UART_RX) and **V18**(UART_TX) are the USB-UART pins of the FPGA chip(**Artix7 fgg484**) on **Minisys** Board.

The **N5**(UART_RX) and **T4**(UART_TX) are the USB-UART pins of the FPGA chip(**Artix7 csg324**) on **EGO1** Board



Changes on CPU Top Module continued

```
module CPU_TOP(  
    input  fpga_rst,    //Active High  
    input  fpga_clk,  
    input[23:0] switch2N4,  
    output[23:0] led2N4,  
  
    // UART Programmer Pinouts  
    // start Uart communicate at high level  
    input  start_pg,    // Active High  
    input  rx,          // receive data by UART  
    output tx          // send data by UART  
);
```

```
// UART Programmer Pinouts  
wire upg_clk, upg_clk_o;  
wire upg_wen_o;    //Uart write out enable  
wire upg_done_o;   //Uart rx data have done  
  
//data to which memory unit of program_rom/dmemory32  
wire [14:0] upg_adr_o;  
  
//data to program_rom or dmemory32  
wire [31:0] upg_dat_o;
```

Q1. How many types of working mode on the CPU which support uart communication to download the program and the data?

How to identify different types of working mode?

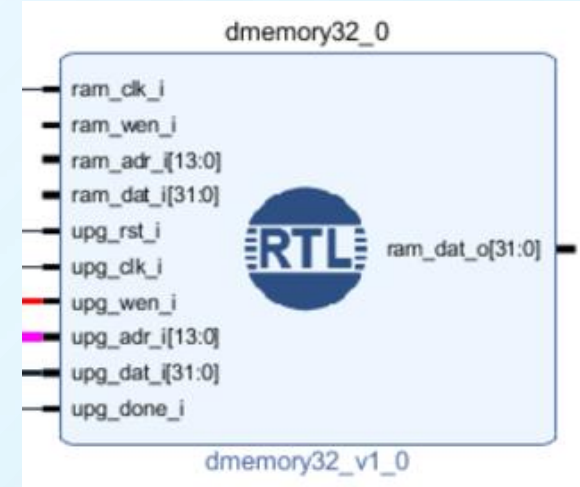
Q2. What's the relationship between the working mode and the "fpga_rst" and "start_pg"?

TIPS: Here the usage of "fpga_rst", "start_pg" and the bitwidth of IO are only one type of implements, not the request.

```
wire spg_bufg;  
BUFG U1(.I(start_pg), .O(spg_bufg)); // de-twitter  
// Generate UART Programmer reset signal  
reg upg_rst;  
always @ (posedge fpga_clk) begin  
    if (spg_bufg) upg_rst <= 0;  
    if (fpga_rst) upg_rst <= 1;  
end  
//used for other modules which don't relate to UART  
wire rst;  
assign rst = fpga_rst | !upg_rst;
```

Changes on Demeory32

```
module dmemory32 (  
    input      ram_clk_i,           // from CPU top  
    input      ram_wen_i,           // from Controller  
    input [13:0] ram_adr_i,          // from alu_result of ALU  
    input [31:0] ram_dat_i,         // from read_data_2 of Decoder  
    output [31:0] ram_dat_o,        // the data read from data-ram  
  
    // 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  
);
```



```
wire ram_clk = !ram_clk_i;
```

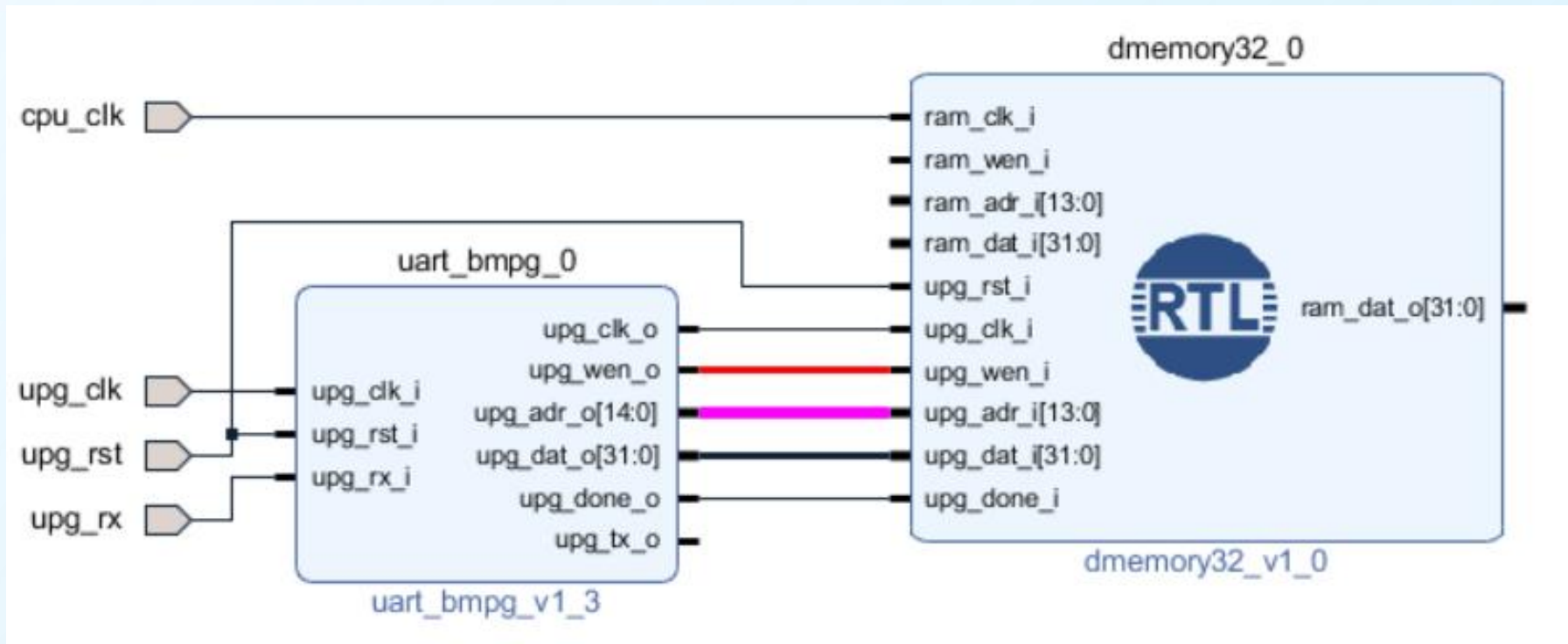
```
/* CPU work on normal mode when kickOff is 1.  
CPU work on Uart communicate mode when kickOff is 0.*/  
wire kickOff = upg_rst_i | (~upg_rst_i & upg_done_i);
```

```
ram ram (  
    .clka (kickOff ? ram_clk : upg_clk_i),  
    .wea (kickOff ? ram_wen_i : upg_wen_i),  
    .addra (kickOff ? ram_adr_i : upg_adr_i),  
    .dina (kickOff ? ram_dat_i : upg_dat_i),  
    .douta (ram_dat_o)  
);
```

Q. While “**kickOff**” is 1'b1, what's the working mode of the CPU ?
How about while “**kickOff**” 1'b0?

Changes on Demeory32 continued

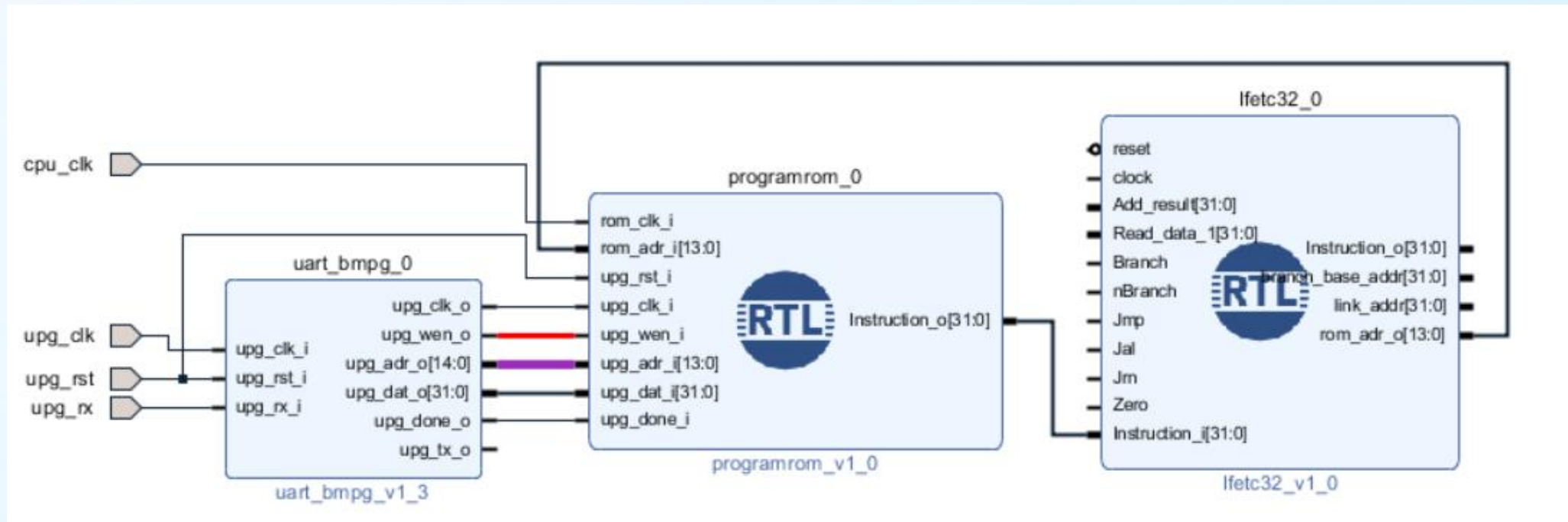
- **upg_wen_i** (uart write enable on **Dmemory32**) :
 - determined by: **upg_wen_o**(from uart_bmpg_0) & **upg_adr_o[14]** (from uart_bmpg_0)
- **upg_adr_i[13:0]** (uart write address on **Dmemory32**):
 - connect with: **upg_adr_o[13:0]** (from uart_bmpg_0)



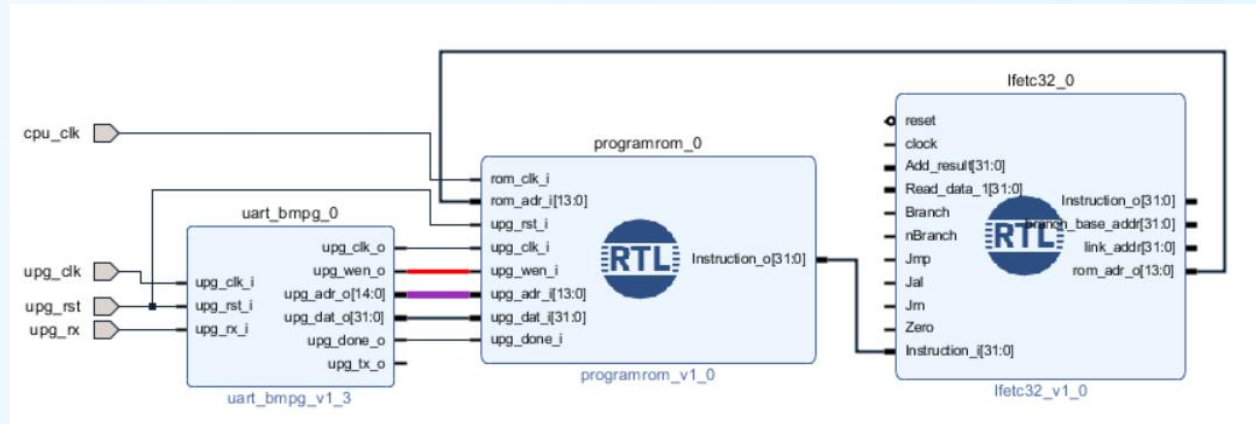


Changes on IFetch

- **Separate IP core("programrom_0")** which stores the Instruction from IFetch(optional)
- **Change program from ROM to RAM**(writeable while work on Uart communication mode, read only while work on normal mode)
- **upg_wen_i** (uart write enable on "programrom"), determined by: **upg_wen_o**(from **uart_bmpg_0**) & **(!upg_adr_o[14])** (from **uart_bmpg_0**)
- **upg_adr_i[13:0]** (uart write address on "programrom"), connect with **upg_adr_o[13:0]** (from **uart_bmpg_0**)



Changes on IFetch continued



```

module programrom (
    // Program ROM Pinouts
    input          rom_clk_i,          // ROM clock
    input[13:0]    rom_adr_i,          // From IFetch
    output [31:0]  Instruction_o,      // To IFetch
    // 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
);
    
```

/ if kickOff is 1 means CPU work on normal mode,
otherwise CPU work on Uart communication mode */*

wire kickOff = upg_rst_i | (~upg_rst_i & upg_done_i);

```

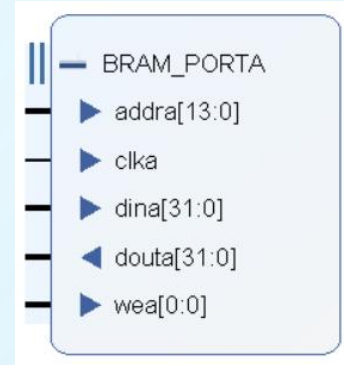
prgrom instmem (
    .clka (kickOff ? rom_clk_i : upg_clk_i ),
    .wea (kickOff ? 1'b0 : upg_wen_i ),
    .addra (kickOff ? rom_adr_i : upg_adr_i ),
    .dina (kickOff ? 32'h00000000 : upg_dat_i ),
    .douta (Instruction_o)
);
endmodule
    
```

Changes on IFetch continued

Make a **new** programrom(which is a **RAM** memory):

TIPS about the "programrom" :

- *While on CPU communication mode, "programrom" is writable.*
- *While on CPU normal mode, "programrom" is readOnly.*

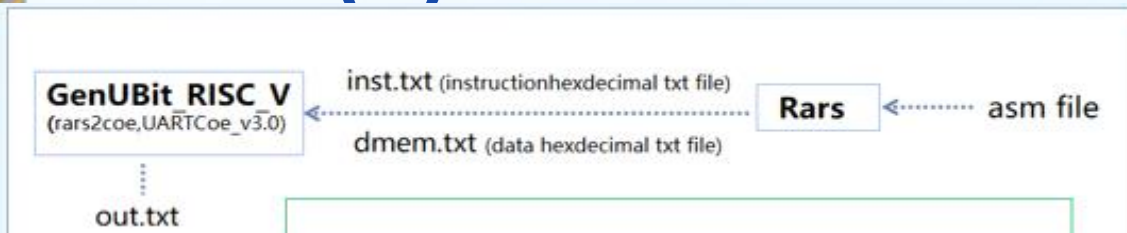


Basic	Port A Options	Other Options	Summary
Interface Type	Native	<input type="checkbox"/> Generate address interface with 32	
Memory Type	Single Port RAM	<input type="checkbox"/> Common Clock	
ECC Options			
ECC Type	No ECC		
<input type="checkbox"/> Error Injection Pins	Single Bit Error Injection		
Write Enable			
<input type="checkbox"/> Byte Write Enable			
Byte Size (bits)	9		
Algorithm Options			
Defines the algorithm used to concatenate the block RAM primitives. Refer datasheet for more information.			
Algorithm	Minimum Area		
Primitive	8kx2		

Basic	Port A Options	Other Options	Summary
Memory Size			
Write Width	32	Range: 1 to 4608 (bits)	
Read Width	32		
Write Depth	16384	Range: 2 to 1048576	
Read Depth	16384		
Operating Mode Write First Enable Port Type Always Enabled			
Port A Optional Output Registers			
<input type="checkbox"/> Primitives Output Register	<input type="checkbox"/> Core Output Register		
<input type="checkbox"/> SoftECC Input Register	<input type="checkbox"/> REGCEA Pin		
Port A Output Reset Options			
<input type="checkbox"/> RSTA Pin (set/reset pin)	Output Reset Value (Hex)	0	
<input type="checkbox"/> Reset Memory Latch	Reset Priority	CE (Latch or Register Enable)	
READ Address Change A			

Basic	Port A Options	Other Options	Summary
Pipeline Stages within Mux 0 Mux Size: 4x1			
Memory Initialization			
<input checked="" type="checkbox"/> Load Init File			
Coe File\minisys.srcs/sources_1/ip/prgrom/prgmip32.coe <input type="button" value="Browse"/>			
<input checked="" type="checkbox"/> Fill Remaining Memory Locations			
Remaining Memory Locations (Hex) 0			
Structural/UniSim Simulation Model Options			
Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.			
Collision Warnings All			
Behavioral Simulation Model Options			
<input type="checkbox"/> Disable Collision Warnings <input type="checkbox"/> Disable Out of Range Warnings			

Tools(1): Generate the Data For Uart Port



- Step1: Using “Rars” to assemble the asm file and dump machine code and data to hexadecimal txt files(inst.txt and dmem.txt)
- Step2: Using “GenUBit_RISC_V” to generate coe files based on the hexadecimal txt files(generated in step1), and then merge the coe files into one file “out.txt” .
 - Tips on Step2:
 - put “inst.txt” and “dmem.txt” into the same directory with “UARTCoe_v3.0” , “rars2coe” and “GenUBit_RISC_V” , or you will need to make some modification on “GenUBit_RISC_V”

```
C:\Windows\System32\cmd.exe
C:\Users\sustech\risc_v_lab1\Rars_assemble_dump_files>dir
驱动器 C 中的卷没有标签。
卷的序列号是 743F-63DC

C:\Users\sustech\risc_v_lab1\Rars_assemble_dump_files 的目录
2024/02/16 21:27 <DIR> .
2024/02/16 21:27 <DIR> ..
2024/02/16 21:10 10,240 dmem.txt
2024/02/16 21:15 401 GenUBit_RISC_V.bat
2024/02/16 20:58 30 inst.txt
2024/02/07 21:32 84,666 rars2coe.exe
2022/05/04 16:57 2,144,754 UARTCoe_v3.0.exe
5 个文件 2,240,091 字节
2 个目录 271,312,105,472 可用字节

C:\Users\sustech\risc_v_lab1\Rars_assemble_dump_files>GenUBit_RISC_V.bat
2 files are read successfully
Hexadecimal file(s) detected.
Done.

C:\Users\sustech\risc_v_lab1\Rars_assemble_dump_files>dir
驱动器 C 中的卷没有标签。
卷的序列号是 743F-63DC

C:\Users\sustech\risc_v_lab1\Rars_assemble_dump_files 的目录
2024/02/16 21:28 <DIR> .
2024/02/16 21:28 <DIR> ..
2024/02/16 21:10 10,240 dmem.txt
2024/02/16 21:28 163,905 dmem32.coe
2024/02/16 21:15 401 GenUBit_RISC_V.bat
2024/02/16 20:58 30 inst.txt
2024/02/16 21:28 262,152 out.txt
2024/02/16 21:28 163,905 prgmip32.coe
2024/02/07 21:32 84,666 rars2coe.exe
2022/05/04 16:57 2,144,754 UARTCoe_v3.0.exe
8 个文件 2,830,053 字节
2 个目录 271,312,920,576 可用字节

C:\Users\sustech\risc_v_lab1\Rars_assemble_dump_files>
```

“GenUBit_RISC_V” , “UartAssist” and “UARTCoe_v3.0” could be found in the “uart_tools.rar” of “labs/lab12_uart” on BlackBoard site



Tools(2): Using “UartAssist”

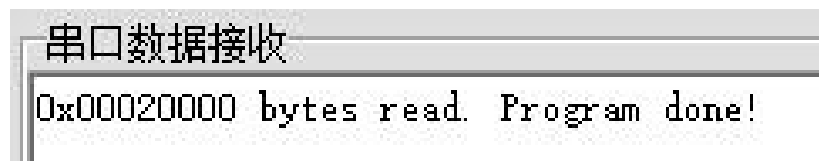
- **Step 1: Connect the Computer** which runs “UartAssist” with **EGO1/Minisysboard** on which your designed CPU has already been programed on its FPGA chip.
- **Step 2: Make sure the CPU on FPGA works on uart communication mode.**
- **Step 3: Double click on “UartAssist” to open it**
- **Step 4: Set** the items in “**串口设置**” as the settings of screen snap on the right hand, then click on “**打开**”
 - **NOTE1:** “**串口号**” could be an **serial port** other than “COM4” , which is **up to your Computer**. The port which you choose here and then click on “**打开**” hasn't report error is the right port.
 - **NOTE2:** “**波特率**” MUST be **128000** while using “**uart_bmpg_0**” in CPU to communicate with “**UartAssist**”



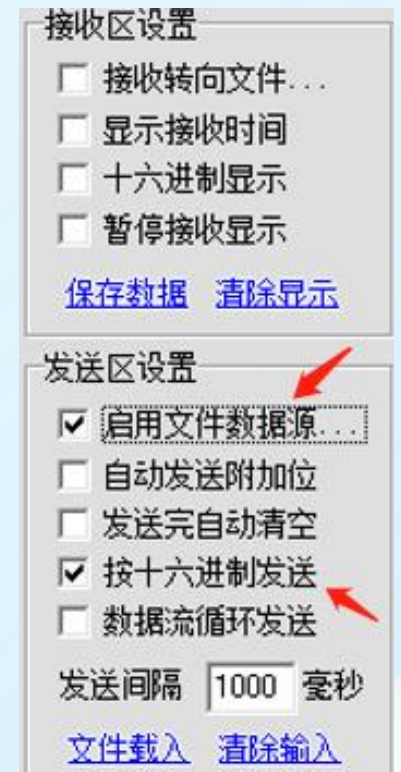


Tips: Using “UartAssist” continued

- **Step 5-1:** Click on the “按十六进制发送” (which means send in hexadecimal) in “发送区设置”
- **Step 5-2:** Click on “启用文件数据源” in “发送区设置” to find and specify the file(out.txt) which is to be transformed by uart port to FPGA chip.
- **Step 5-3:** Click on “发送” to send the file by the uart port.
- **Step 6:** Wait until a notice info “Program done!” has appeared in the “串口数据接收” window as the screen snap on the right hand.



- If “Program done” has appeared, it means the new program(out.txt) has been received by CPU and been loaded into the Instruction Memory and Data Memory in CPU, Next, we can shift to CPU working mode to test the new program on the CPU.





Practice(optional)

- 1. Modify the Data-memory module, do the unit test on the updated Data-memory.
- 2. Modify the IFetch, do the unit test on the updated IFetch.
- 3. Update the CPU with uart communication implemented.
- 4. Do the test: Programe FPGA chip only once, make the CPU run the different programs by using uart communication to update the instructions and data of new program.

It is strongly recommended to conduct unit test first, and then conduct integration test after passing the unit test.