



C o m p u t e r O r g a n i z a t i o n



Lab12 Practice on Uart port with CPU

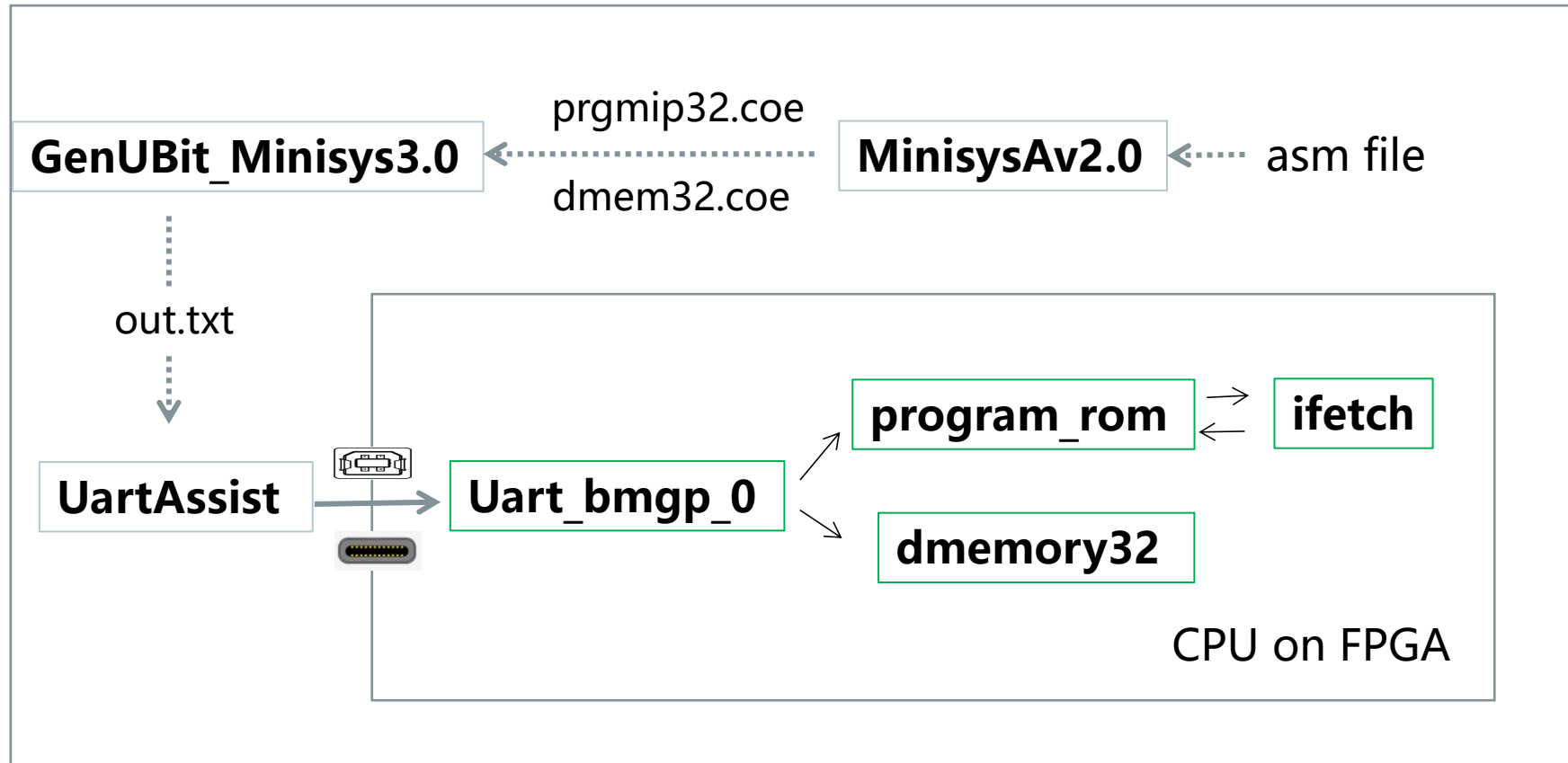


2 Topics

➤ CPU

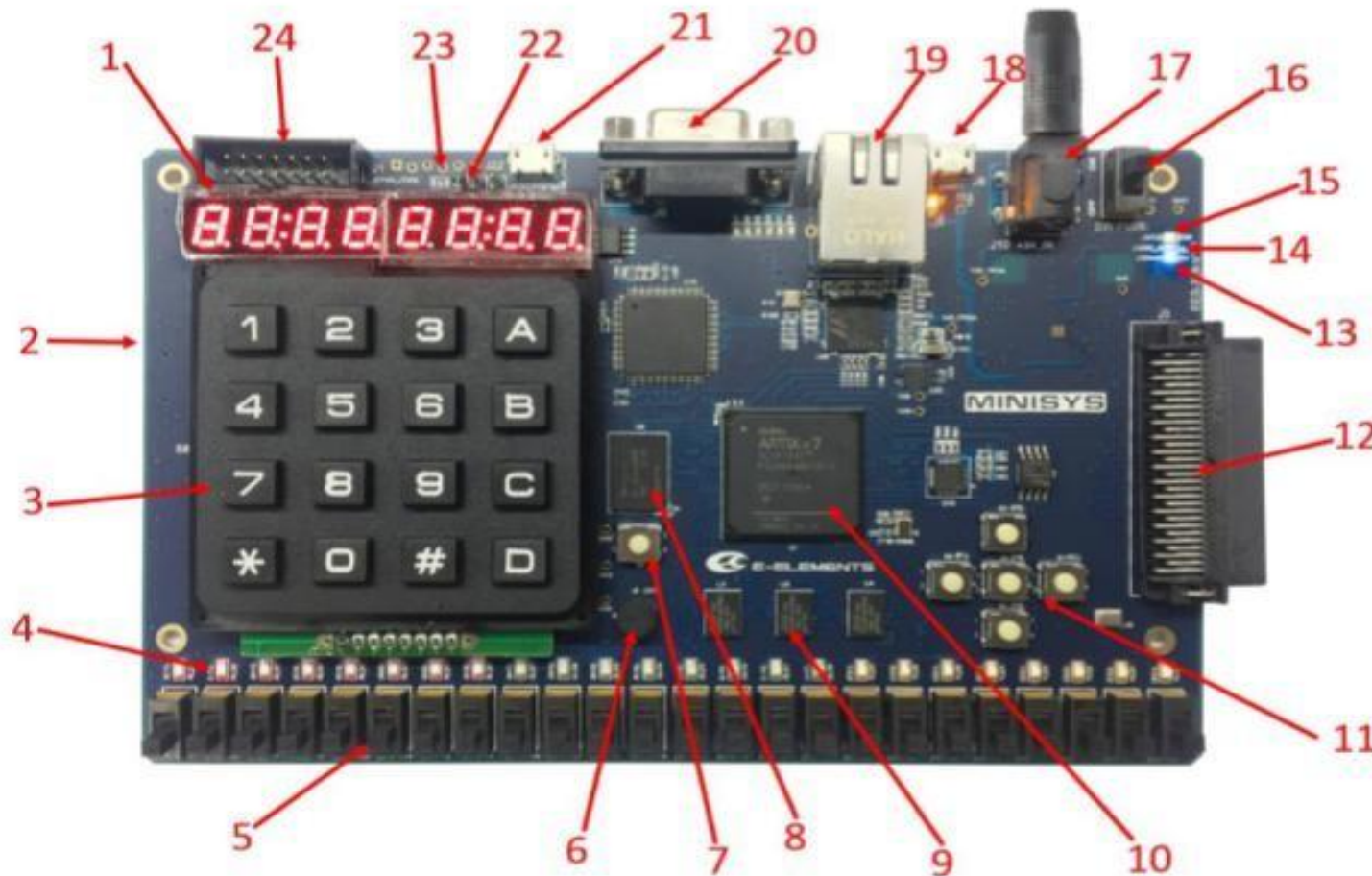
- What is a CPU (General purpose processor)
- How to make CPU work on a new program
 - Communicate with UART port to get coe file
 - Rewrite ProgramRAM and DataRAM
- Let's Do it
 - 2 tools
 - Modification on CPU code

Update The Data In Instruction And Data Memory



"GenUBit_Minisys3.0" , "UartAssist" and "UARTCoe_v3.0" could be found in the "Uart tools" of bb site:
https://bb.sustech.edu.cn/webapps/blackboard/content/listContentEditable.jsp?content_id=_288483_1&course_id=_3609_1

Uart Interface on Minisys Board



For Minisys board(old version, the type of USB_Jtag interface is typeB), port 18 is the USB to UART interface.

For Minisys board(new version, the type of USB_Jtag interface is typeC), USB_Jtag and USB to UART interface share the same port: port 21.

Changes on Single Cycle CPU

1. Two working modes on the CPU

- Normal mode vs Uart Communication mode

2. A new module which works as Uart interface

3. A new clock for uart communication

4. Changes

- 3-1) CPU top:

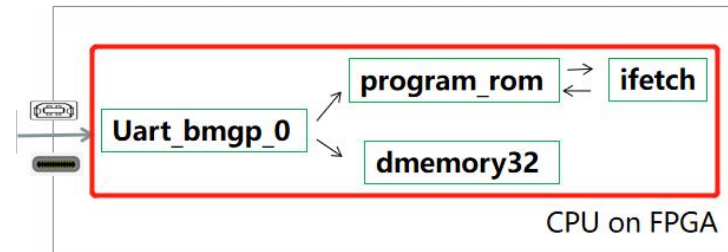
New module, new ports, new internal connection and new logic

- 3-2) Changes on Data-memory

Working mode: Normal mode vs Uart Communication mode

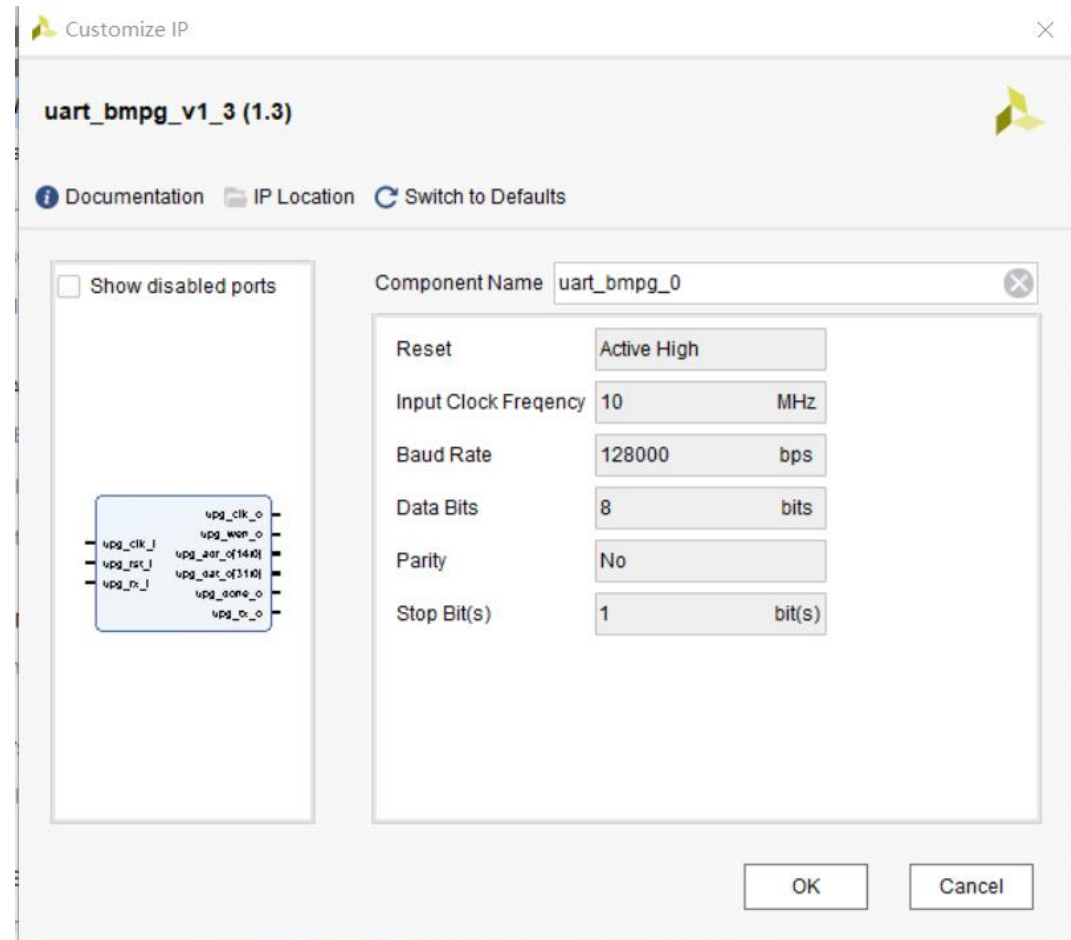
- 3-3) Changes on IFetch

- Change IP core "prgrom" from ROM to RAM
- Working mode: Normal mode vs Uart Communication mode
- Separate "prgrom" from IFetch (optional)



6 Add an IP core Which Processes Uart Data

- Add the **IP core** to IP catalog of vivado
- Add the IP core(**uart_bmpg_0**) from IP catalog into vivado project, then instance it
- **NOTICE: Don't change the settings of this IP core**
- 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.




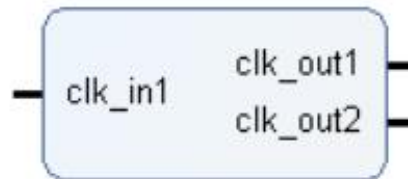
Get the IP core from:

https://bb.sustech.edu.cn/webapps/blackboard/content/listContentEditable.jsp?content_id=_288483_1&course_id=_3609_1

7 Add a New Clock For The New IP core

- Reset the “cpucclk” IP core to make a new clock
 - Add a new clk_out (clk_out2) whose frequency is **10 Mhz** for the IP core which is used for **Uart communication**(page 6)

>  cpucclk : cpucclk (cpucclk.xci)



Component Name cpucclk

Clocking Options Output Clocks Port Renaming PLLE2 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

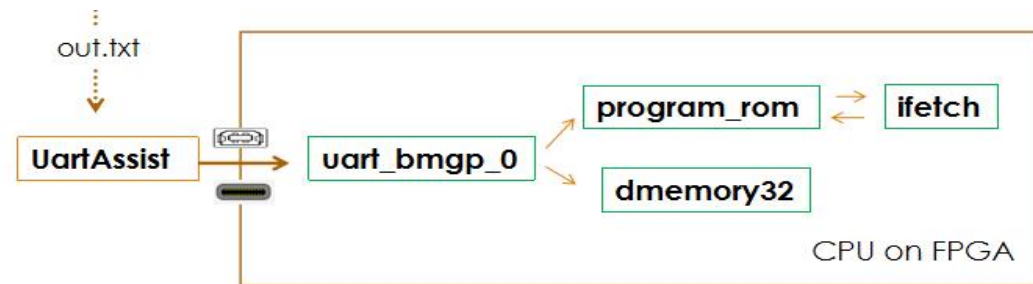
single cycle_cpu_clk

uart_clk

8 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  
);
```

```
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN Y19} [get_ports rx]  
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN V18} [get_ports tx]
```



Here the usage of “fpga_rst” and “start_pg” are only one type of implements, not the request.

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

9 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  
);
```

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"?

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

```
// 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;
```

```
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;
```

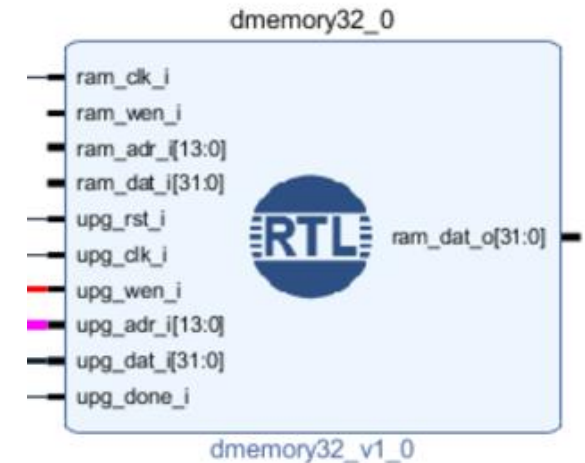
10 Changes on Demeory32

```

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);

```

```

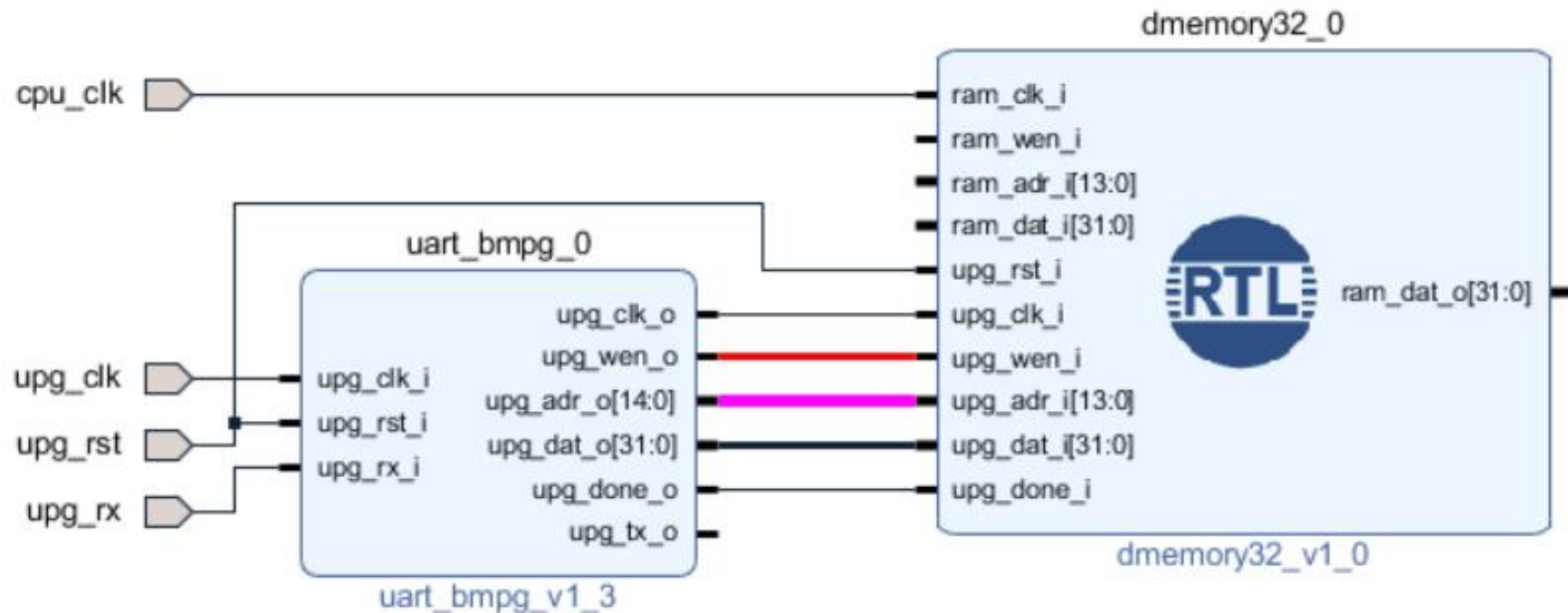
ram ram (
    .clka (kickOff ? ram_clk : upg_clk_i),
    .wea (kickOff ? ram_wen_i : upg_wen_i),
    .addra (kickOff ? ram_adr_i : upg_adr_i),
    .dina (kickOff ? ram_dat_i : upg_dat_i),
    .douta (ram_dat_o)
);

```

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

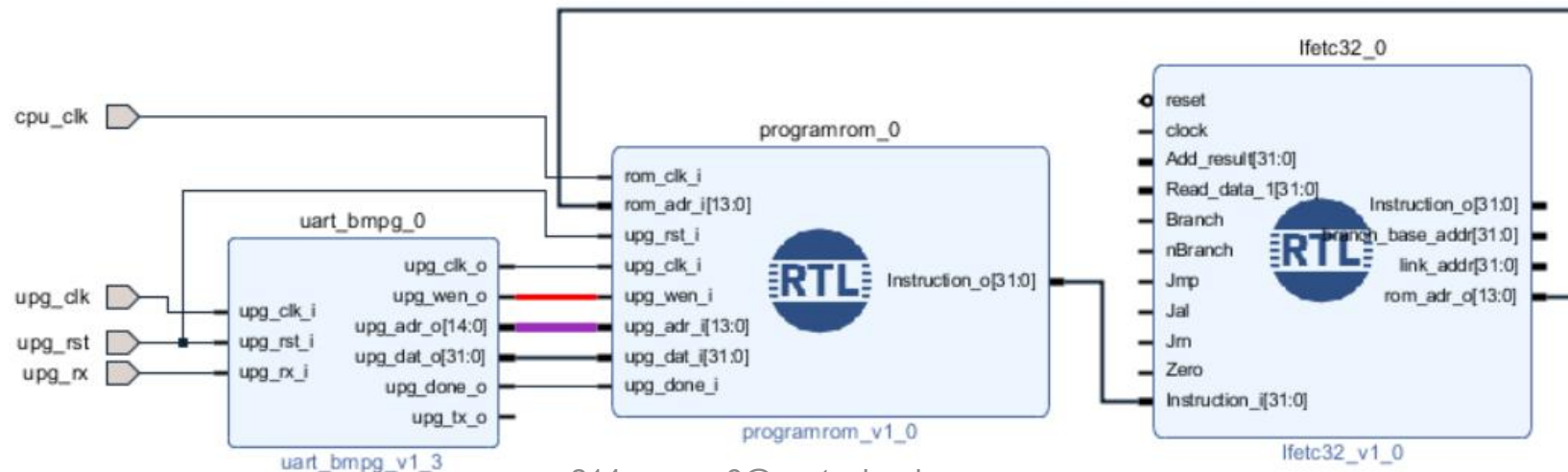
Changes on Dmemory32 continued

- **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)

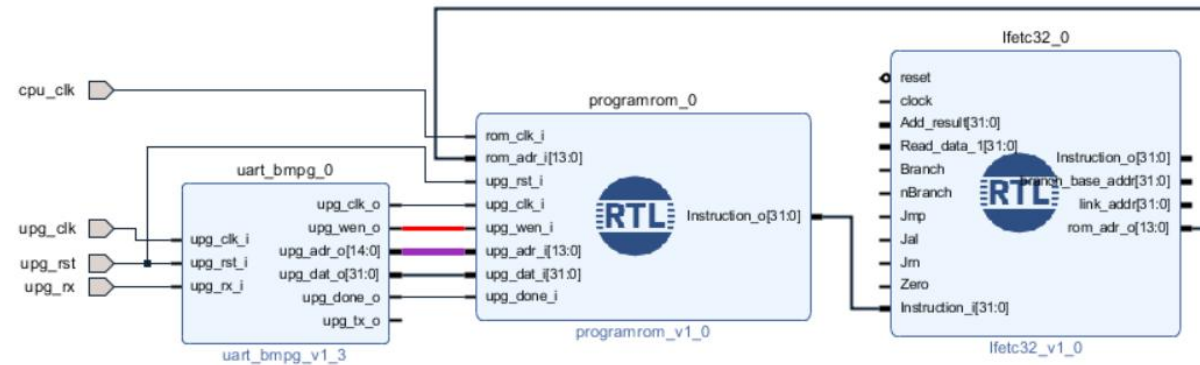


12 Changes on IFetch

- **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);

```

program 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

```

14 Changes on IFetch continued

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



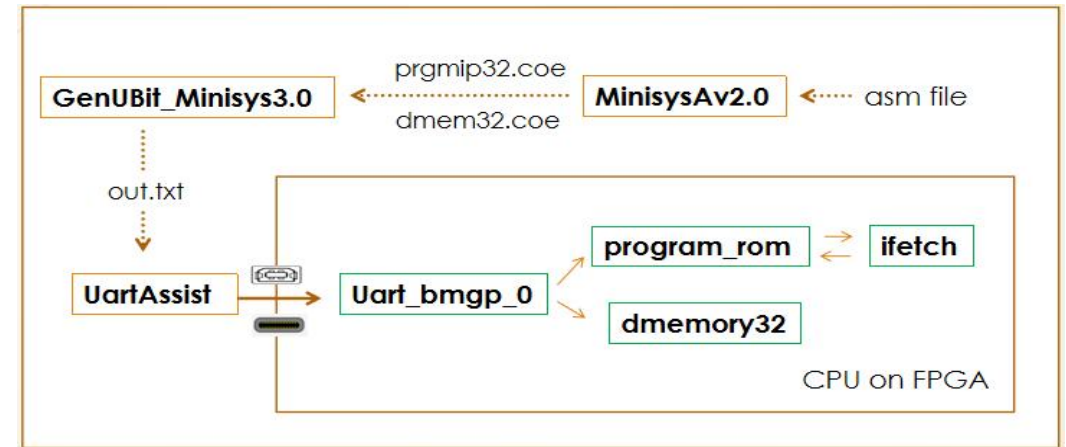
Basic	Port A Options	Other Options	Summary
Interface Type	Native	<input type="checkbox"/> Generate address interface with 32	
Memory Type	Single Port RAM	<input type="checkbox"/> Common Clock	
ECC Options			
ECC Type	No ECC		
<input type="checkbox"/> Error Injection Pins	Single Bit Error Injection		
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.srscs/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		

15 Tools(1): Generate the Data For Uart Port

- Step1: Using “MinisysAv2.0” to assemble the asm file and generate the coe files(prgmip32.coe and dmem32.coe)
- Step2: Using “GenUBit_Minisys3.0” to merge the coe files(prgmip32.coe and dmem32.coe) into one file “out.txt”
 - Tips on Step2:
 - put “prgmips32.coe” and “dmem32.coe” into the same directory with “UARTCoe_v3.0” and “GenUBit_Minisys3.0” , or you will need to make some modification on GenUBit_Minisys3.0



LENOVO (D:) > UartTools

名称

- dmem32.coe
- GenUBit_Minisys3.0
- prgmip32.coe
- UARTCoe_v3.0

```
d:\UartTools>GenUBit_Minisys3.0.bat
2 files are read successfully
Hexadecimal file(s) detected.
Done.
```

“GenUBit_Minisys3.0” , “UartAssist” and “UARTCoe_v3.0” could be found in the “Uart tools” of bb site:
https://bb.sustech.edu.cn/webapps/blackboard/content/listContentEditable.jsp?content_id=_288483_1&course_id=_3609_1

16 Tools(2): Using “UartAssist”

- **Step1: Connect the Computer** which runs “UartAssist” with **Minisysboard** on which your designed CPU has already been programed on its FPGA chip.
- **Step2:** Double click on “**UartAssist**” to **open** it
- **Step3: Set** the items in “**串口设置**” as the settings of screen snap on the right hand, then click on “**打开**”
 - **NOTICE:** “**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.



“GenUBit_Minisys3.0” , “UartAssist” and “UARTCoe_v3.0” could be found in the “Uart tools” of bb site:
https://bb.sustech.edu.cn/webapps/blackboard/content/listContentEditable.jsp?content_id=_288483_1&course_id=_3609_1

17 Tips: Using “UartAssist” continued

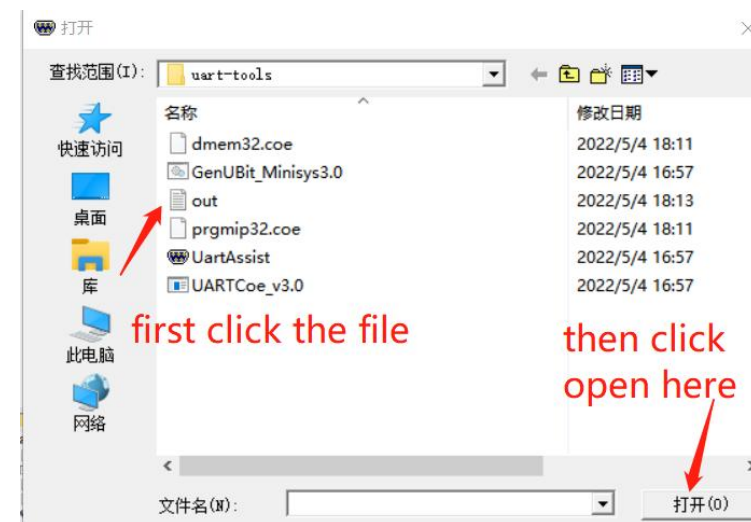
- **Step4:** Make sure the **CPU on FPGA works on uart communication mode.**



- **Step5:** Set the items in “发送区设置” as the screen snap on the right hand. Click on “启用文件数据源”, find the data file which is to be transformed by uart port to FPGA chip.



- **Step6:** 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.



串口数据接收

0x00020000 bytes read. Program done!

18 Practice

- 1. Is there a way to update the program to be executed on a CPU faster?
- 2. Modify the Data-memory module, do the unit test on the updated Data-memory.
- 3. Modify the IFetch, do the unit test on the updated IFetch.
- 4. Update the CPU with uart communication implemented. Do the test: Program 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.