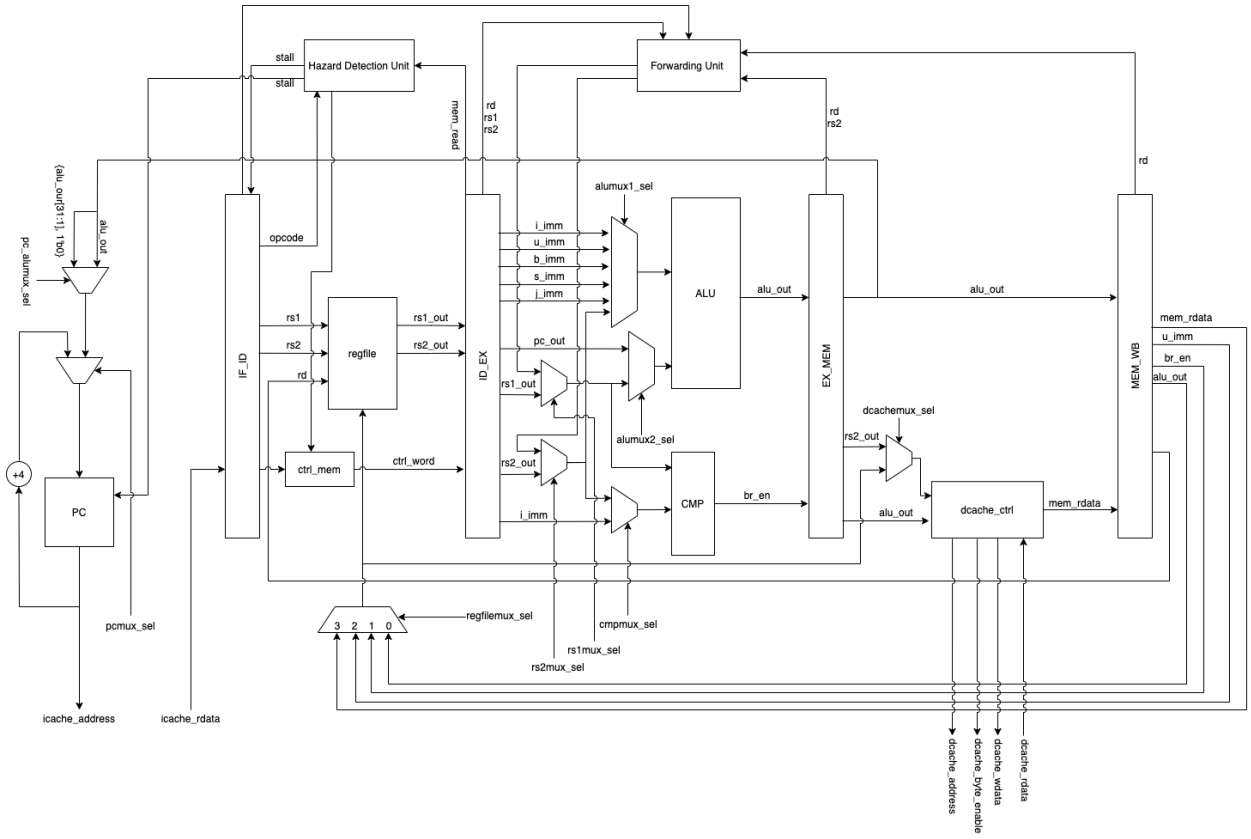
CP1 Progress Report/Roadmap

Nicholas Logan, Himanshu Bhadoria, Ching-Chia Kuo

3/28/22

Himanshu worked on the Arbiter design, Nick worked on the Forwarding and Hazard detection design and verification of the implementation, and Ching-Chia worked on the implementation of the basic pipelined datapath.



Our current datapath design supports all functionalities required for a basic rv32i CPU. The RISC-V pipeline is broken down into 5 stages, with one step for each stage. The steps are as follows: IF: Instruction fetch from memory; ID: Instruction decode & register read; EX: Execute operationor calculate address; MEM: Access memory operand; WB: Write result back into register. We insert four intermediate holding registers between stages that we called “stage_register”: IF_ID, ID_EX, EX_MEM, and MEM_WB for holding signals passing between each stage. We separate the signals passing between each stage into two data structures: rv32i_control_word and rv32i_data_word.

rv32i_control_word:

Signal type	**Signal name**
rv32i_opcode	opcode
alu_ops	aluop
regfilemux_sel_t	regfilemux_sel
logic	load_regfile
alumux1_sel_t	alumux1_sel
cmpmux_sel_t	cmpmux_sel
branch_funct3_t	cmpop
logic	dcache_read
logic	dcache_write
logic [3:0]	dcache_byte_enable
logic [2:0]	funct3
logic [6:0]	funct7
logic	br_en

rv32i_data_word:

Signal type	**Signal name**
rv32i_word	pc
rv32i_word	imm
rv32i_reg	rs1
rv32i_reg	rs2
rv32i_reg	rd
rv32i_word	rs1_out
rv32i_word	rs2_out

rv32i_word	alu_out
rv32i_word	mdr_out

To be more specific, the required signals for each stage will be fetched from the previous intermediate stage register's output and the result signals of each stage will be output to the input port of the later intermediate stage. In addition, the unused signals in rv32i_control_word and rv32i_data_word will be bypassed to the later intermediate stage register.

MUX Select Logic

pcmux

pcmux_sel	= {opcode==jal || opcode==jalr, (br_en && opcode == op_br)}
pcmux::pc_plus4	pcmux_out = pc_out + 4;
pcmux::alu_out	pcmux_out =.alu_out;
pcmux::alu_mod2	pcmux_out = {alu_out[31:1], 1'b0};
pcmux::br_en	pcmux_out = pc_out + 4;

regfilemux

regfilemux_sel	= MEM_WB_output.control_word.regfilemux_sel
regfilemux::alu_out:	regfilemux_out = MEM_WB_output.data_word.alu_out;
regfilemux::br_en:	regfilemux_out = {31'd0, MEM_WB_output.control_word.br_en};
regfilemux::u_imm:	regfilemux_out = MEM_WB_output.data_word.imm;
regfilemux::pc_plus4:	regfilemux_out = MEM_WB_output.data_word.pc + 4;
regfilemux::lw, lh, lhu, lb, lbu	regfilemux_out = MEM_WB_output.data_word.mdr_out;

alumux1

alumux1_sel	= ID_EX_output.control_word.alumux1_sel
0	alumux1_out = ID_EX_output.data_word.rs1_out;
1	alumux1_out = ID_EX_output.data_word.pc;

alumux2

alumux2_sel	= (ID_EX_output.control_word.opcode == op_reg)
0	alumux2_out = ID_EX_output.data_word.imm;
1	alumux2_out = ID_EX_output.data_word.rs2_out;

cmpmux1 wire to rs1 for CP1

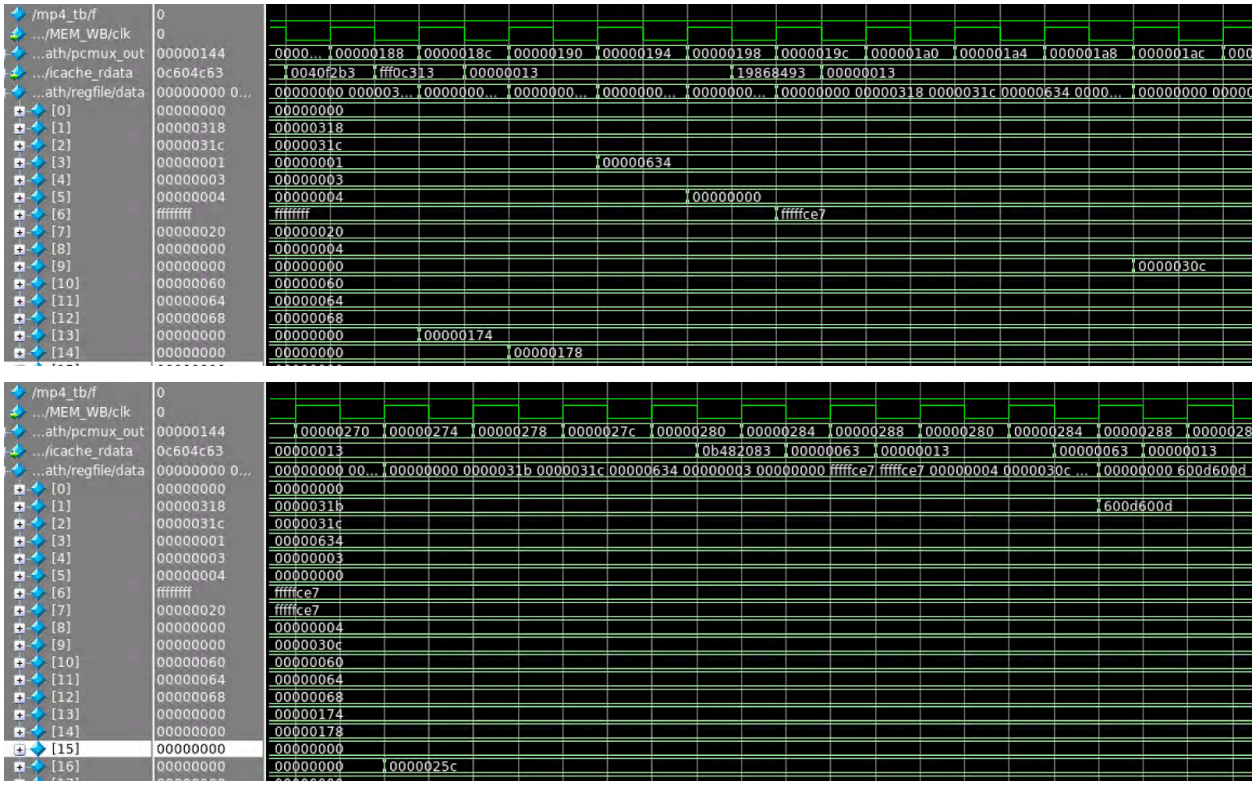
cmpmux2

cmpmux2_sel	ID_EX_output.control_word.cmpmux_sel
0	cmpmux2_out = ID_EX_output.data_word.rs2_out;
1	cmpmux2_out = ID_EX_output.data_word.imm;

Testing

Our testing strategy was done using the provided test code, mp4-cp1.s, and using ModelSim to verify the correctness of the waveform.

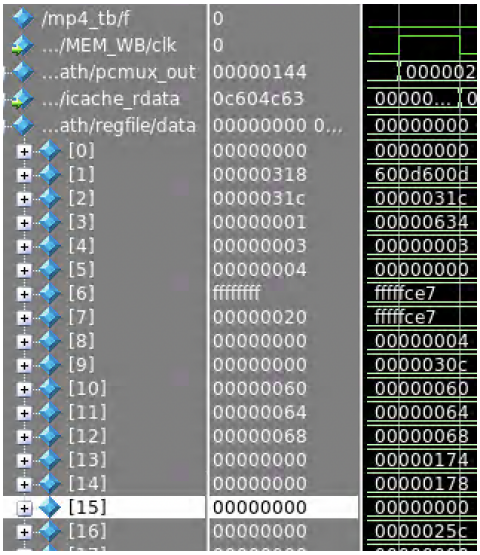

.../MEM_WB/clk
...ath/pcmux_out
.../icache_rdata
...ath/regfile/data
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]
[11]
[12]
/mp4_tb/f



As we can see in the last waveform above, 600d600d is loaded into register 1. Looking into the asm code for mp4-cp1.s, we can see the following is one of the last instructions:

```
lw x1, %pcrel_lo(pcrel_GOOD)(x16)
```

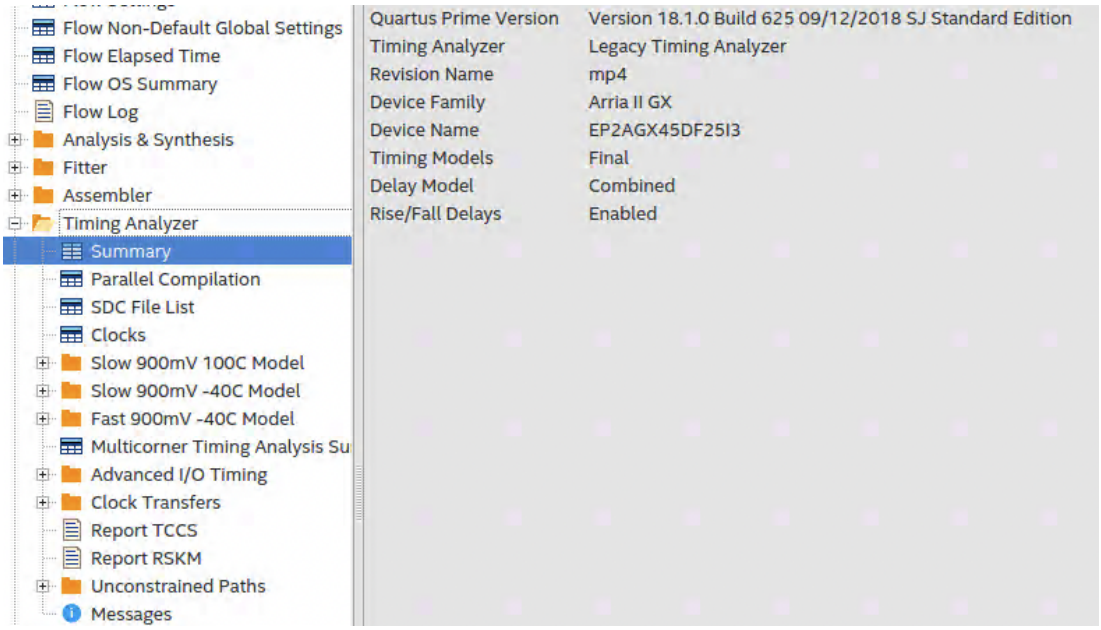
As it looks like all of the instructions are being reflected in the waveform, we can say with some level confidence that our pipelined design is working.



The above is the final results in the registers.

Timing Analysis + Energy Report

After writing the SDC file and running Timing Analysis, our compilation gave no errors or warnings on timing.



	Fmax	Restricted Fmax
Slow 900mV 100C Model	122.43 MHz	122.42 MHz
Slow 900mV -40C Model	129.74 MHz	129.74 MHz
Fast 900mV -40C Model	n/a	n/a

Here are the results after writing the Power Analyzer Tool.

Power Analyzer Status	Successful - Mon Mar 28 11:37:17 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Standard Edition
Revision Name	mp4
Top-level Entity Name	mp4
Family	Arria II GX
Device	EP2AGX45DF25I3
Power Models	Final
Total Thermal Power Dissipation	413.94 mW
Core Dynamic Thermal Power Dissipation	36.47 mW
Core Static Thermal Power Dissipation	318.72 mW
I/O Thermal Power Dissipation	58.76 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

```
        Running Quartus Prime Power Analyzer
        Command: quartus_pow --read_settings_files=on --write_settings_files=off mp4 -c mp4
 18236 Number of processors has not been specified which may cause overloading on shared ma
 21077 Low junction temperature is -40 degrees C
 21077 High junction temperature is 100 degrees C
332104 Reading SDC File: 'mp4.out.sdc'
332152 The following assignments are ignored by the derive_clock_uncertainty command
223000 Starting Vectorless Power Activity Estimation
222013 Relative toggle rates could not be calculated because no clock domain could be ident
223001 Completed Vectorless Power Activity Estimation
218000 Using Advanced I/O Power to simulate I/O buffers with the specified board trace mode
334003 Started post-fitting delay annotation
334004 Delay annotation completed successfully
215049 Average toggle rate for this design is 16.156 millions of transitions / sec
215031 Total thermal power estimate for the design is 413.94 mW
        Quartus Prime Power Analyzer was successful. 0 errors, 2 warnings
```

Roadmap

Himanshu will work on the L1 cache and Arbiter, Nick will work on the Hazard Detection and Forwarding, and Ching-Chia will work on the static branch predictor implementation/verification. We will all work on the advanced feature proposal and designs. In particular, we are planning to work on the L2+ cache system[2], 4-way associative cache[2], local branch history table[2], global 2-level branch history table[3], tournament branch predictor[5], RISC-V M extension[3], and the Eviction write buffer[4].

The features to implement next checkpoint are: Hazard Detection and Forwarding to deal with data hazards and stalling, Arbiter to interface the caches, and Static Branch Predictor to predict the outcome of the branch based on instruction content.