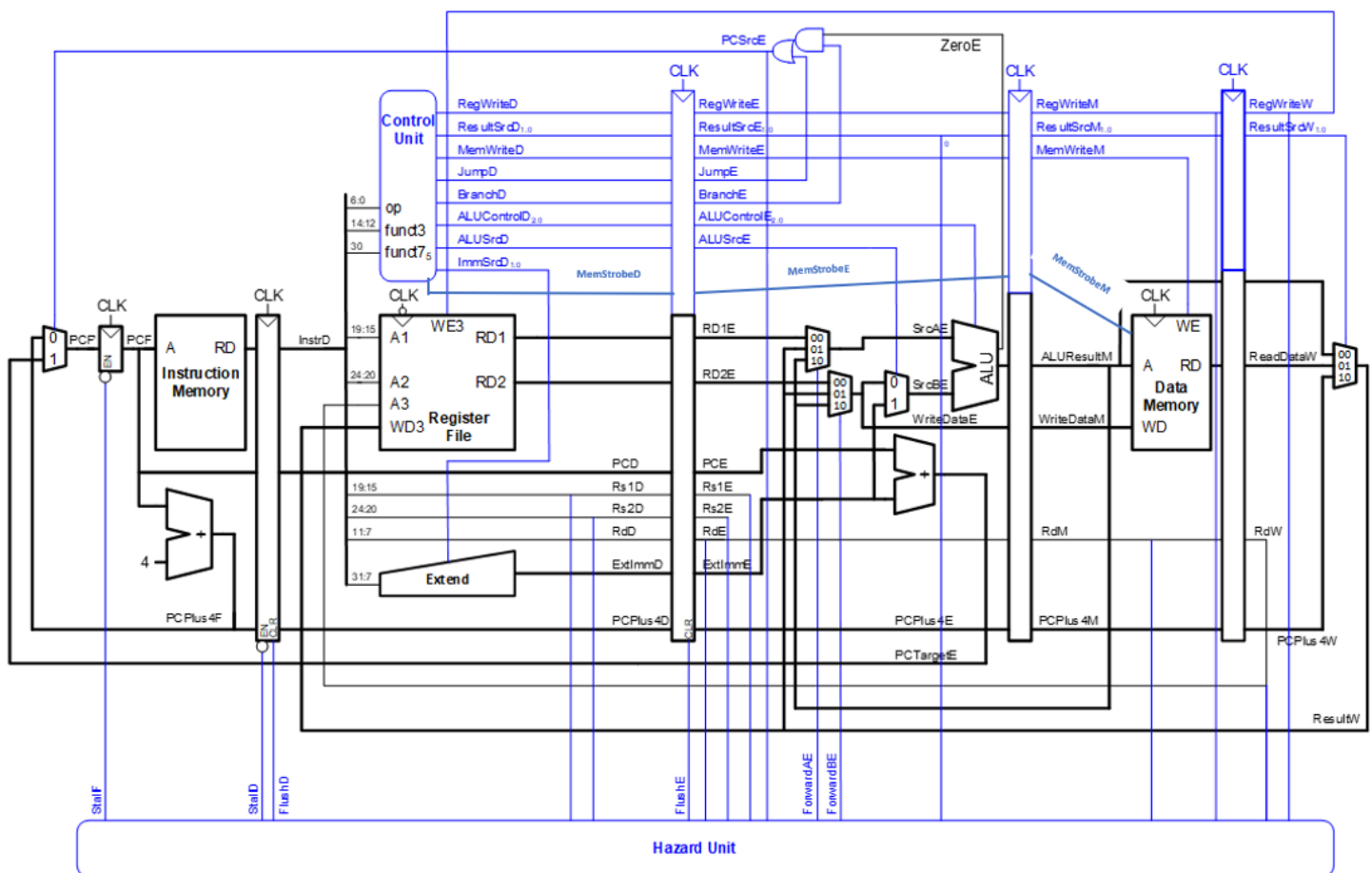


SoC Project Report
IEEE ASUB Advanced
FPGA (Final Project)

M.B: Kareem Atef

1. The Design of the original RV32I

Full Datapath



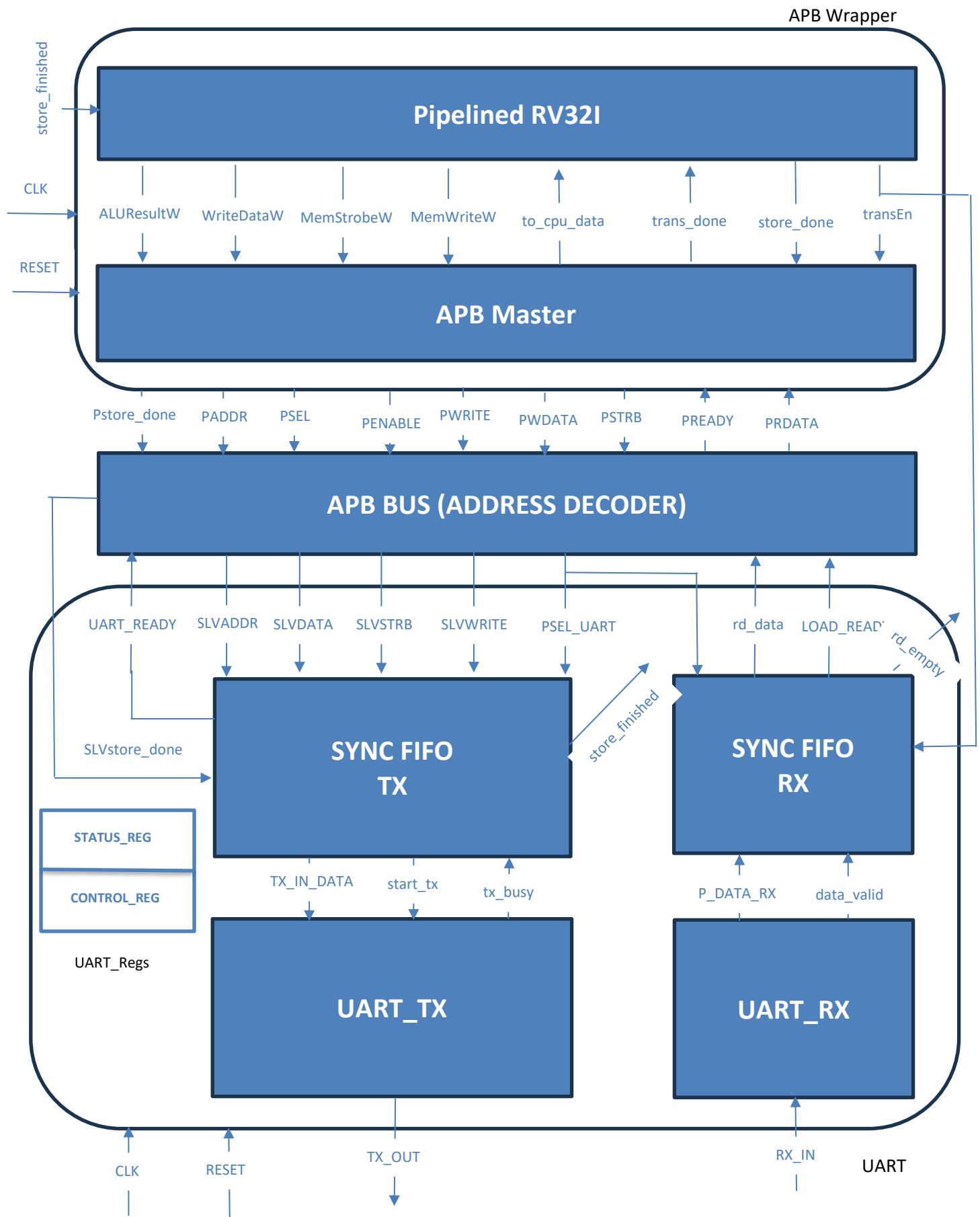
Note: I added MemStrobe Signal to handle (load/store) byte, half word and word.

Supported Instruction set

- My Design supports 28 different instructions here are the following supported instructions formats:

imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	LLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
imm[20 10:1 11 19:12]				rd	1101111	JAL
imm[12 10:5]	rs2	rs1	000	imm[4:1 11]	1100011	BEQ
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[31:12]				rd	0110111	LUI

2. SoC Diagram



3. Memory Map

Logic element		Start Address	End Address
Instruction Memory		0x0000_0000	0x0000_00FF
Reserved		0x0000_0100	0x0FFF_FFFF
Data Memory		0x1000_0000	0x1000_01FF
Reserved		0x1000_0200	0x1FFF_FFFF
Bridge	UART	0x2000_0000	0x2000_000C
	Other Perips ..	0x2000_0200	0x2000_03FF
		0x2000_0400	0x2000_05FF

4. Working idea of this SoC

- The pipelined RV32I deals with other peripherals with load/store instructions in this SoC we only deal with a UART Peripheral.
- The CPU access the UART upon its allowed addresses in the memory map which is the function of the APB Bus to send the signals from/to CPU to the required peripheral which here is the UART.
- This UART when it comes to storing the CPU sends the data with one specific address to store the data in a FIFO so according to the CPU it is talking to a specific register then the TX can take this data and send it outside in case it's not busy and the FIFO is not empty.
- When it comes to loading, once any data is sent to the UART from outside The RX receives it in serial transmit it in parallel to the FIFO to be stored then it can be loaded to the CPU and store it in the register file.

5. Signals Deceleration

1. APB Wrapper:

- ALUResultW: Address in the WB stage for which in case of a store instruction in the UART memory its base indicates which peripheral and the offset indicates which memory location in that peripheral, in case of a load its only function determines whether I want to load this data from the data memory or the FIFO and the base helps with that.
- WriteDataW: Data to be written in case of a store instruction to the UART.
- MemStrobeW: Signal indicates whether I store or load (byte, half word or word).
- MemWriteW: Considered a write enable in case of a store instruction to the UART.
- to_cpu_data: it carries the data coming from the FIFO to be stored in the Reg file in case of a load instruction from the UART.
- trans_done: Signal asserted to 1 when the CPU receives the to_cpu_data.
- transEn: Signal asserted to 1 in case of a load/store instructions to alert the SoC that the CPU may interact with the UART according to the base of the sent address.
- store_done: asserted in case of a store instruction which plays a role in stalling the cpu.
- store_finished: asserted when the data send to UART stored in the FIFO.
- APB Signals.

2. APB BUS:

- PSEL_UART: signal takes the value of PSEL in case of the address coming targeting the UART according to the memory map
- Other APB Signals is passed directly to the targeted peripheral according to the PSEL.

3. SYNC FIFO TX:

- TX_IN_DATA: data is sent from the FIFO to the TX to be sent outside the SoC in case the TX is not busy and the FIFO is not empty.

4. UART TX:

- P_DATA: Data sent to the UART to be stored in FIFO for a later use by the CPU.
- start_tx: Start bit deasserted to 0 when a data is ready to be transmitted to the RX.
- TX_OUT: Serial out data.
- tx_busy: Asserted to 1 each time the TX is busy with transferring data.

5. UART RX:

- RX_IN: Serial input data to the SoC.
- data_valid: Asserted to 1 when all the bits transferred successfully.

6. SYNC FIFO RX:

- rd_data: data extracted from the FIFO.

7. UART Regs: responsible for the status and control registers that stores and informations or data may be needed by the cpu as the status data from the uart or a data or information needed by the UART as the control data

6. The Changes in the Pipelined RV32I to adapt within SoC

1. As mentioned above I added a (MemStrobe) signal to help me with store/load (byte/half_word/word).
2. During the implementation of LUI instruction I added a mux before the Imm Extend to choose whether I want to extend the 25 bits according to instruction type or to extend the 20 bits of upper immediate and the selector for that mux was a signal (lui_en) which is asserted only when I have a LUI instruction.
3. At the (ReadDataW) I also added a mux that chooses between (ReadDataW) or (to_cpu_data) which is coming from the UART in case I am loading from it and for the selector signal is (trans_done) which when it is asserted means that the data I brought from the UART has arrived so I make the reg_file load the (to_cpu_data).
4. For that I added some Stalls to Stall the CPU in case of a load instruction from UART added (StallE,StallM,StallW) Which makes the CPU stalls according to the following condition:

[StallE=StallM=StallW=
(~trans_done&(peripheral_load===2)&(~opcode5|(store_done&~store_finished))&transEn)].

- (peripheral_load===2) is for 4 bytes in the base part of the address that indicates I am communicating with a peripheral not the data memory.
- (transEn) which is asserted in case of a load/store instruction to ensure that the cpu will stall only in case if I needed in a load instruction from the UART.
- For the stalling of the cpu in case of storing once there a store a signal store_done is asserted and passed to the hazard unit in the WB stage and it moves with the data and the address till it reaches to the FIFO_TX then a signal store_finished is asserted and passed directly to the cpu to stop the stalling.

7. Program Test

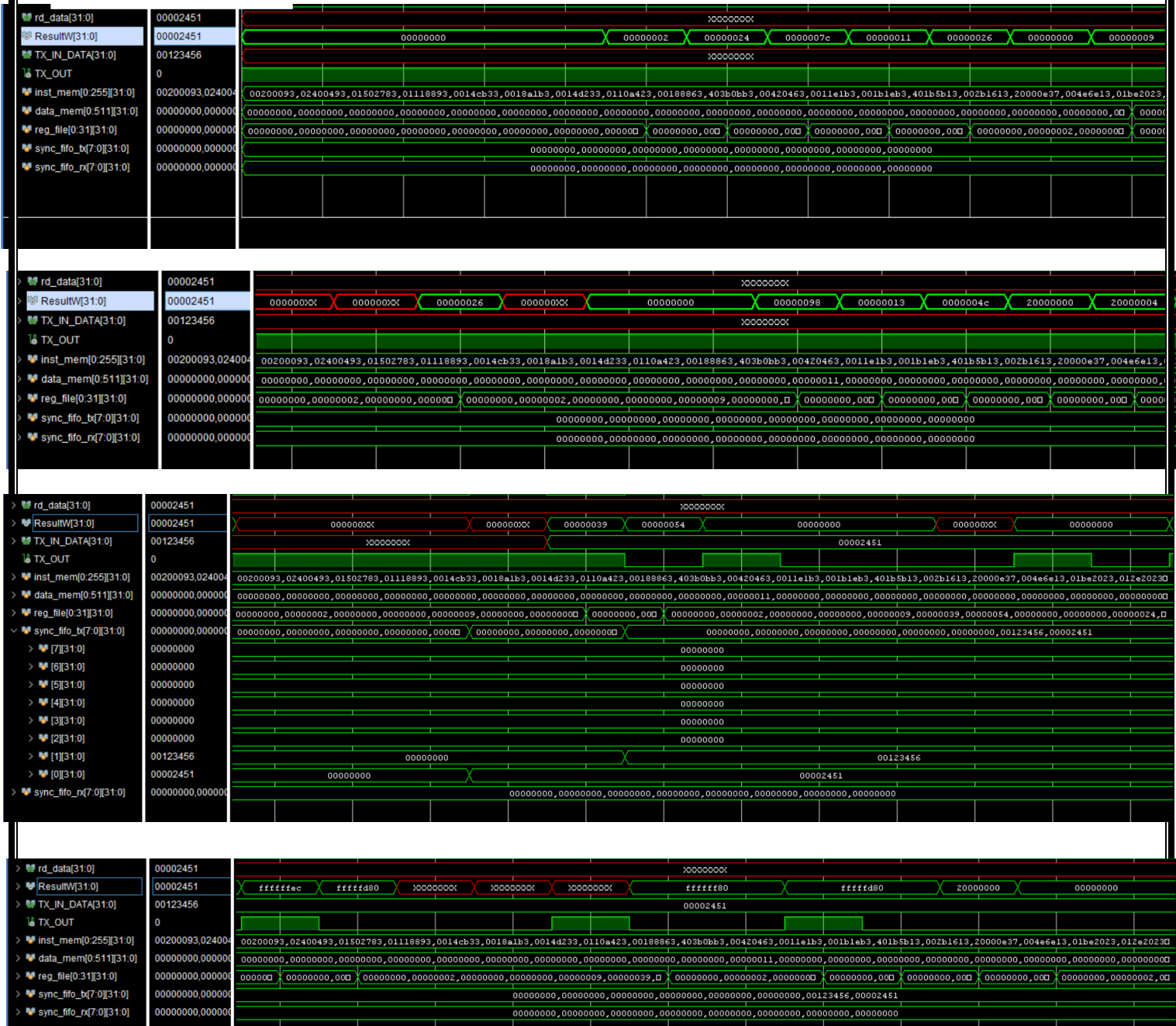
Address	Assembly	Comment	Machine Code
0x0	addi x1,x0,2	x1=2	00200093
0x4	addi x9,x0,36	x9=36	02400493
0x8	lw x15,21(x0)	x15=DM[21]=124	01502783
0xC	addi x17,x3,17	x17=17	01118893
0x10	xor x22,x9,x1	x22=38	0014CB33
0x14	slt x3,x17,x1	x3=0	0018A1B3
0x18	sra x4,x9,x1	x4=9	0014D233
0x1C	sw x17,8(x1)	DM[10]=17	0110A423
0x20	beq x17,x1,branch	not executed	00188863
0x24	sub x23,x22,x3	x22=38	403B0BB3
0x28	beq,x4,x4,branch	executed	00420463
0x2C	or x3,x3,x1	not executed	0011E1B3
branch: 0x30	sll x29,x22,x1	x29=152	001B1EB3 // The following 3 instruction to test data forwarding
0x34	srai x22,x22,1	x22=19	401B5B13
0x38	slli x12,x22,2	x12=76	002B1613
0x40	lui x28,131072	x28=8'h20000000	20000E37 // The following 4 instructions to test the store in the FIFO_TX
0x44	ori x28,x28,4	x28=8'h20000004	004E6E13
0x48	sw x27,0(x28)	x27=9297	01BE2023
0x4C	sw x18, 0(x28)	x18=(123456)hex	012E2023
0x50	sub x5,x12,x22	x5=57	416602B3
0x54	jal x6,test1	x6=pc+4=58	00C0036F
0x58	addi x5,x5,10	not executed	00A28293
0x5C	sub x5,x5,x22	not executed	016282B3
test1: 0x60	sltiu x25,x4,-6	x25=0	FFA23C93
0x64	beq x25,x0,test2	executed	000C8463
0x68	xor x17,x17,x9	not executed	0098C8B3 // The remaining instructions to test lb,lh,lw,sb,sh,sw
test2: 0x6C	addi x16,x0,4076	x16=4076	FEC00813
0x70	slli,x19,x16,5	x19=(1FD80)hex	00581993
0x74	sb x19,38(x0)	DM[38]=(80)hex	03300323
0x78	sh x19,39(x0)	DM[39]=(FD80)hex	033013A3
0x7C	sw x19,40(x0)	DM[40]=(fffffd80)	03302423
0x80	lb x24,40(x0)	x24=DM[40]=(80)hex	02800C03
0x84	lh x25,40(x0)	x25=(FD80)hex	02801C83
0x88	lw x26,40(x0)	x26(fffffd80)	02802D03
0x90	lui x2,131072	x2=8'h20000000	20000137 // The following 2 instructions to test the load from the UART FIFO and the reason they are
0x94	lw x2,0(x2)	x2=8'h12345678	00012103 the last instructions to give the UART time till it already transmit data and store it in the FIFO
0x98	xor x22,x9,x1	x22=38	0014CB33

Note: The Machine Codes are stored in this order in the instruction memory.

8. Some of the Simulation Snippets

- I will show you the final stage of my instruction execution (mainly in ResultW) for an easy checkup instead of reviewing a whole bunch of signals XD

- WB Stage:



[illegible]

Note: From The program test section, the column (Comment) contains the final stage of each instruction where can be seen from the ResultW stage and the memories to ease the verification of the correct execution of each instruction.

Also, there is a folder attached called Memory Content contains the first stage of any mem file before writing anything and the final version of this mem file after the execution of the program.