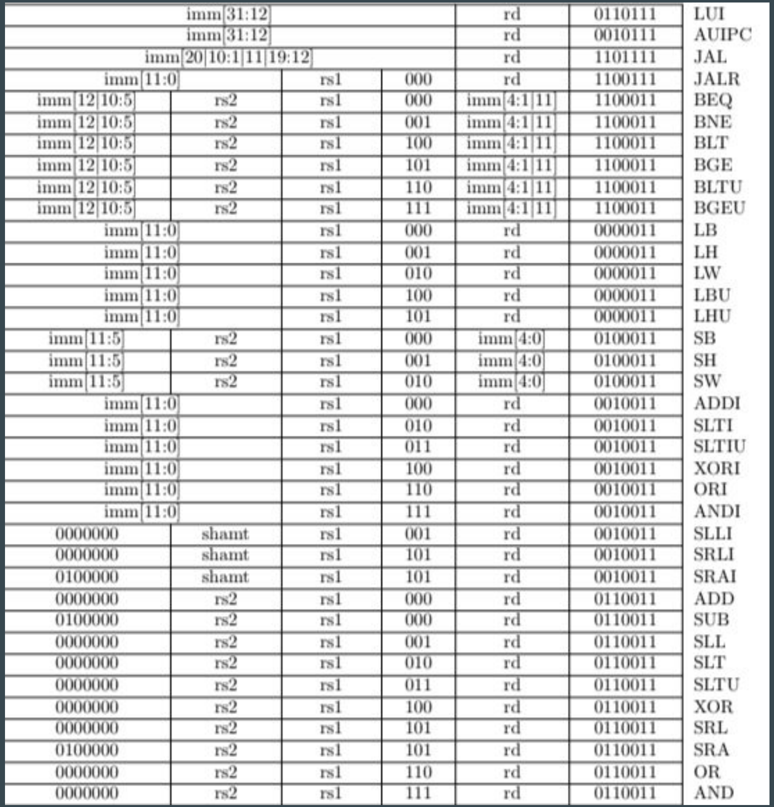
Battleship Micro Architecture Review

•••

By: Jaime Campos, Josh Cobian, Jacob Edmundson, Alan Ekstrand, Zach Simons

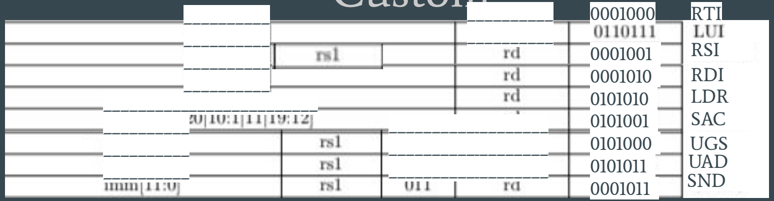
ISA

RV32I



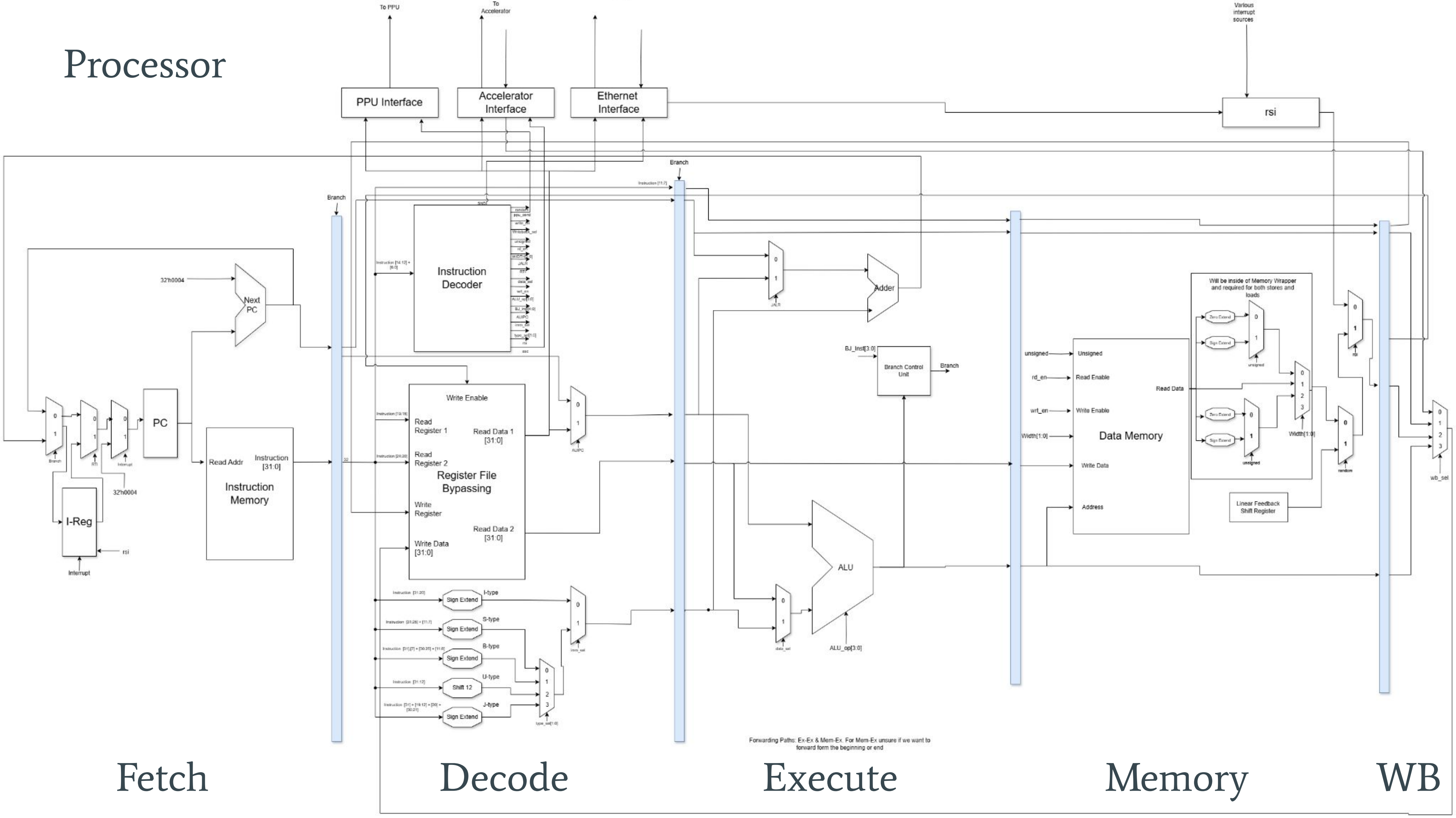
imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
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[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
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	SLLI
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

Custom



				0001000	RTI
				0110111	LUI
	rs1		rd	0001001	RSI
			rd	0001010	RDI
			rd	0101010	LDR
20|10:1|11|19:12]				0101001	SAC
	rs1			0101000	UGS
	rs1			0101011	UAD
imm[11:0]	rs1	011	rd	0001011	SND

Processor



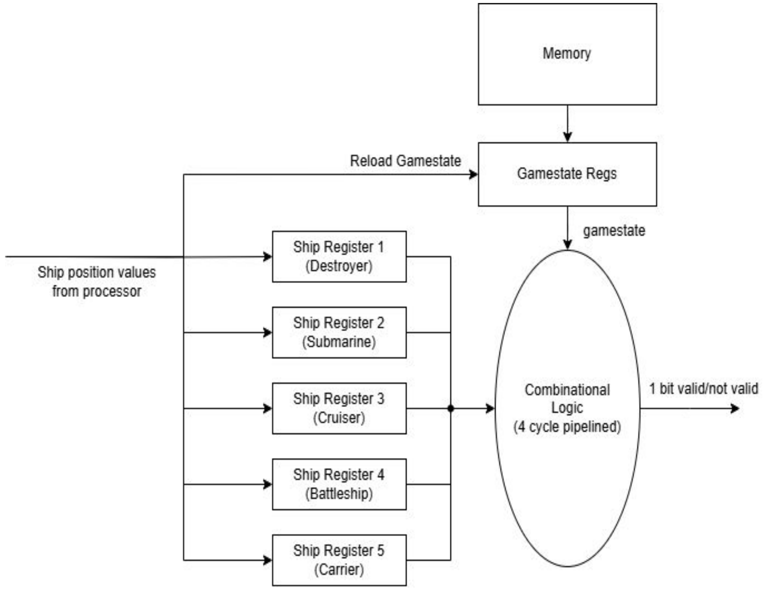
Forwarding Paths: Ex-Ex & Mem-Ex. For Mem-Ex unsure if we want to forward form the beginning or end

Fetch Decode Execute Memory WB

Accelerator

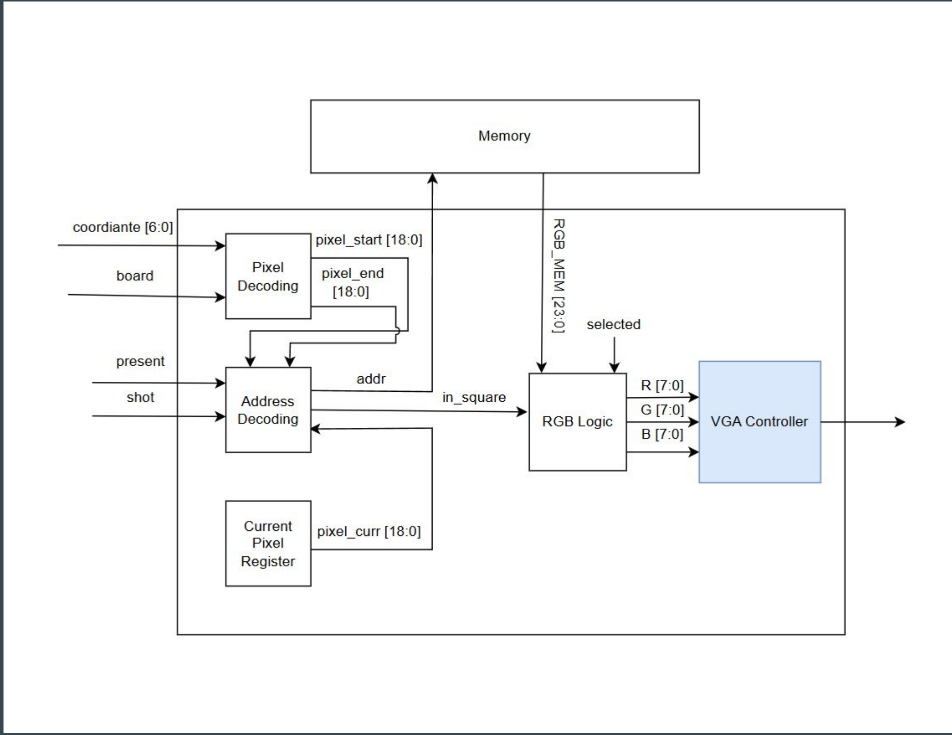
- Input/Output
  - Ship positions/reload gamestate
  - valid/invalid
- Process
  - Loads random ship locations into regs
    - Grid location
    - Orientation
  - Gets gamestate from memory
    - Loads into gamestate regs
  - Checks ship placements in parallel
  - Outputs if ship placement is valid or not

```
for each ship {
        select one random possible ship location
        if location is incompatible with any
other selected locations, continue loop A
    }
    if this configuration conflicts with the
current board state…
```

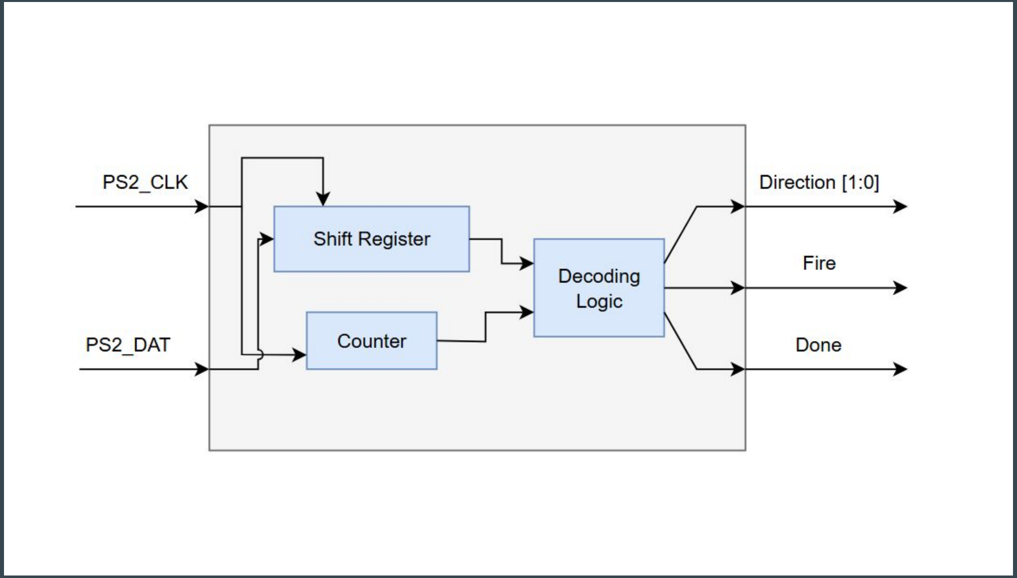

Memory
Reload Gamestate
Gamestate Regs
gamestate
Ship position values from processor
Ship Register 1 (Destroyer)
Ship Register 2 (Submarine)
Ship Register 3 (Cruiser)
Ship Register 4 (Battleship)
Ship Register 5 (Carrier)
Combinational Logic (4 cycle pipelined)
1 bit valid/not valid

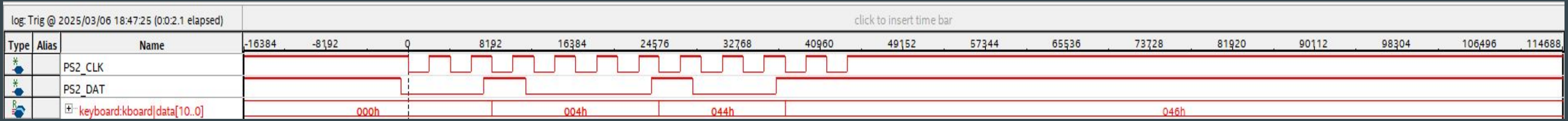
PPU

- board, xy, sel, e, m, v, type, seg, ai
- 1, 7, 1, 1, 1, 1, 3, 3, remaining
- board - which board to update
- xy - 7 bit coordinate (0-99)
- sel - 1 if square is selected, else 0
- e - 1 if ship is present in grid square, else 0
- m - 1 if grid square has been shot at, else 0
- v - 1 if ship is vertical, 0 if ship is horizontal
- type - type of ship encoding (0-destroyer, 1-submarine, 2-cruiser, 3-battleship, 4-carrier)
- seg - which square of the ship it is (0-front, max-back)



Keyboard (PS/2) Controller

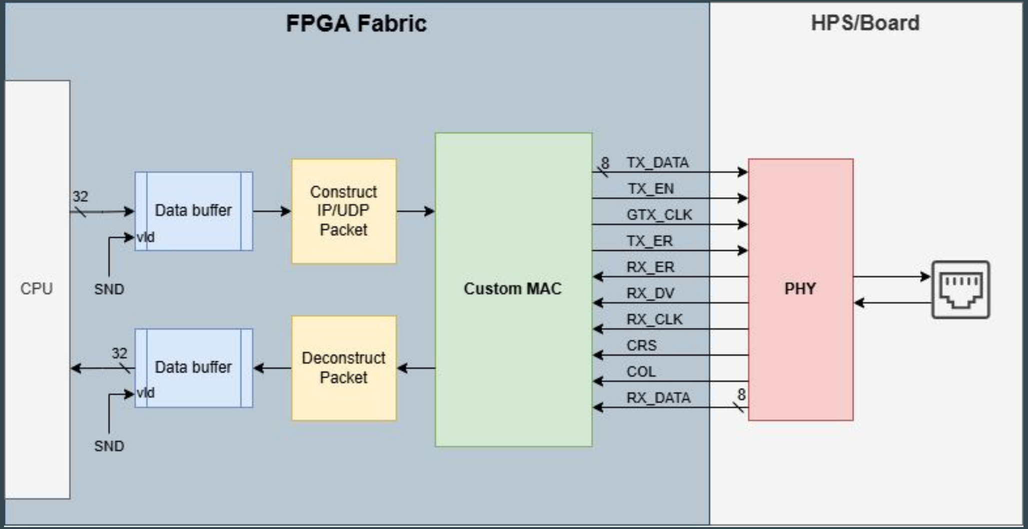



log: Trig @ 2025/03/06 18:47:25 (0:0:2.1 elapsed)

Ethernet Interface

Custom MAC in FPGA logic, communicate with PHY via GMII (1Gig) interface

- GMII to transmit data to PHY
- MDIO master to configure PHY
- UDP/IP packetizer/depacketizer to connect rest of design to MAC


FPGA Fabric
HPS/Board
CPU
32
Data buffer
vld
SND
Construct IP/UDP Packet
32
Data buffer
vld
SND
Deconstruct Packet
Custom MAC
8
TX_DATA
TX_EN
GTX_CLK
TX_ER
RX_ER
RX_DV
RX_CLK
CRS
COL
RX_DATA
8
PHY