Vivado ZedBoard:

32-bit Computer using MIPS Architecture

Ahmad Malik

ECE-311 Hardware design

Fall 2022

MIPS and ARM

- Both, ARM and MIPS are based on Reduced Instruction Set Computing (RISC).
- Both support instructions sets that are 32 bit/64 bit long, and both the instruction sets can be configured to big endianness as well as little endianness.
- The architectures of both ARM and MIPS are used in processors of smart phones and tablet computers such as iPhones, android and Windows RT tablets. However, modern computers typically use architectures like X86 and X64

MIPS vs. ARM

•ARM is the most widely used instruction set architecture in the world. ARM architectures are used in smart phones and tablet computers. ARM chips are also used in Raspberry Pi and other single-board computers because of their low power consumption and low cost.

•MIPs architecture is used in making smart phones, supper computers, embedded systems, routers, residential gateways, and video consoles like PlayStation.

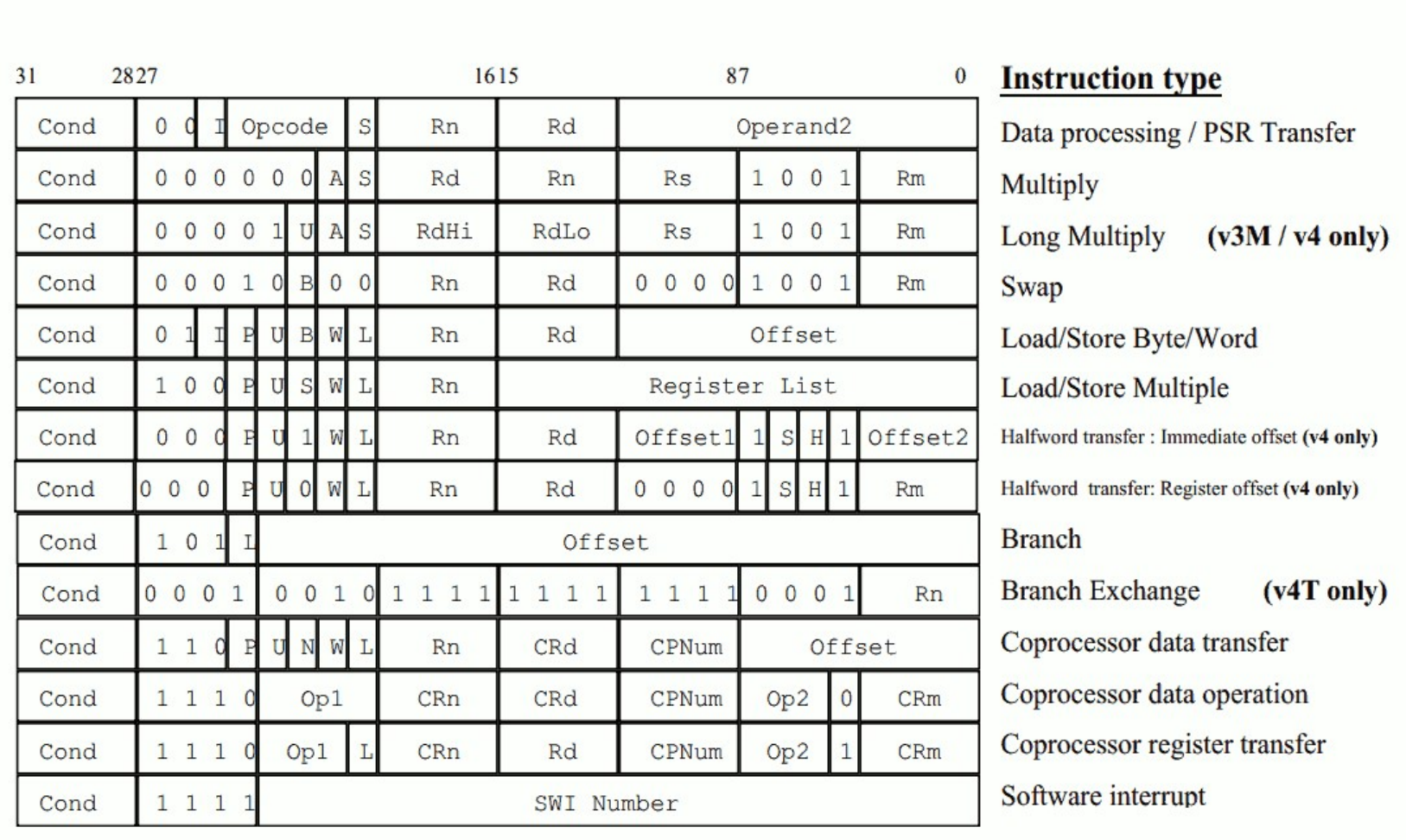
•MIPS supports 32 different Registers while ARM supports 16 registers.

•Despite this, ARM has a greater throughput and efficiency than MIPS because ARM processors support 64-bit data bus, so more data is processed per clock cycle

ARM ISA

- The ARM instruction set can be divided into six broad classes of instructions: Branch, Data-processing, Load and store, Coprocessor, and Exception-generating instructions.
- Different ARM instructions can be identified using the opcode and the conditional flags.
- ARM has 16 general purpose registers

ARM Instruction Format



Credit: https://www.cs.uaf.edu/courses/cs301/2014-fall/notes/arm-asm/

ARM Registers

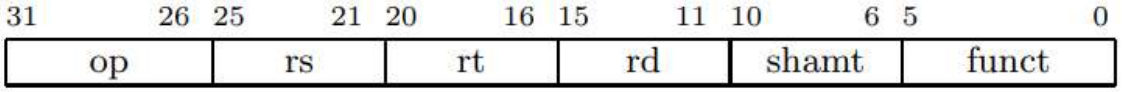
Registers	Use	Comment
R0	ARG 1	Used to pass arguments to subroutines. Can use them as scratch registers. Caller saved.
R1	ARG 2	
R2	ARG 3	
R3	ARG 4	
R4	VAR 1	Used as register based variables. Subroutine must preserve their data. Callee saved. Must return intact after call.
R5	VAR 2	
R6	VAR 3	
R7	VAR 4	
R8	VAR 5	
R9	VAR 6	Variable or static base
R10	VAR 7 / SB	Variable or stack limit
R11	VAR 8 / FP	Variable or frame pointer
R12	VAR 9 / IP	Variable or new static base for interlinked calls
R13	SP	Stack pointer
R14	LR	Link back to calling routine.
PC	PC	Program counter

MIPS ISA

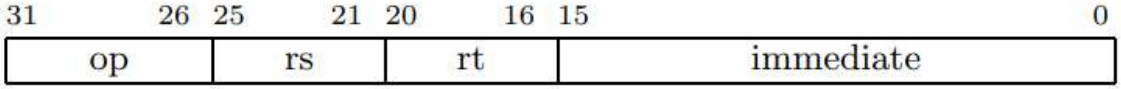
- There are three types of MIPS instructions: R, I and J Type.
- MIPS supports 32 Registers! Register $0 holds binary 0, and register 1 is reserved for the assembler

MIPS Instruction Format

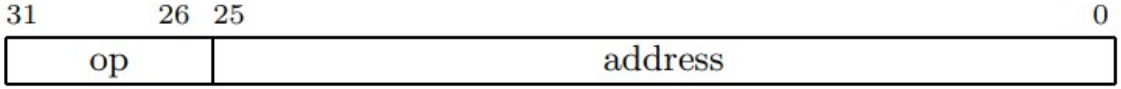
R-format (Register)



I-format (Immediate)



J-format (Jump)

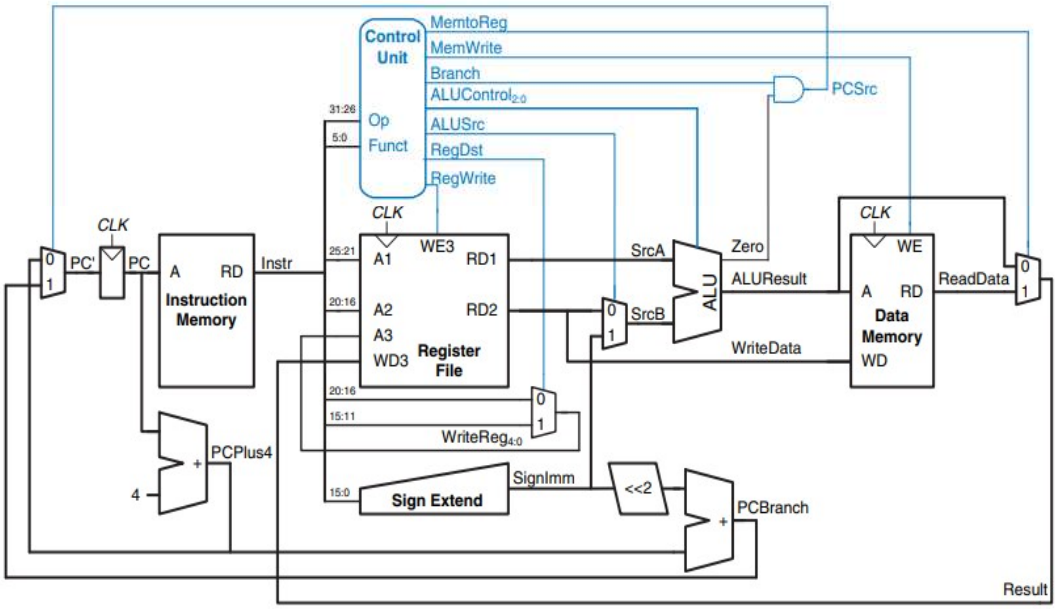


op	6-bit opcode
rs	5-bit source register specifier
rt	5-bit target register specifier
rd	5-bit destination register specifier
shamt	5-bit shift amount
funct	6-bit function field
immediate	16-bit immediate
address	26-bit absolute address

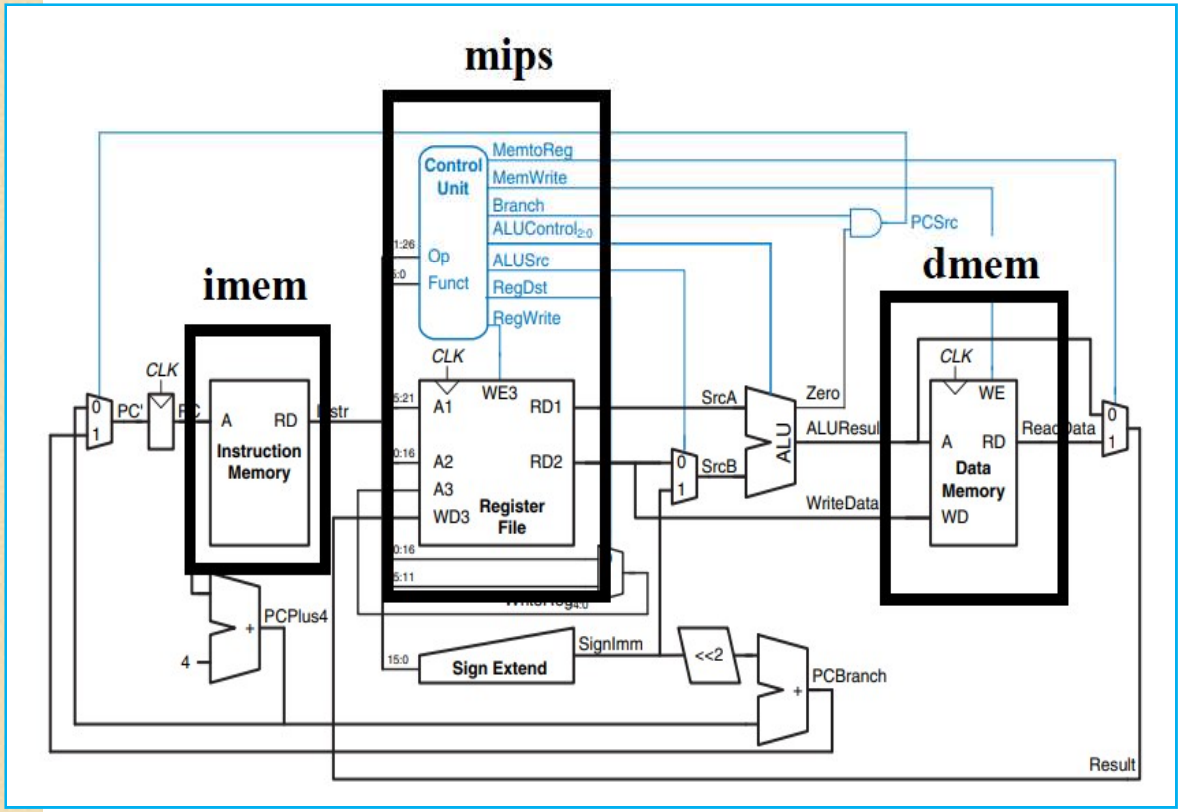
MIPS Registers

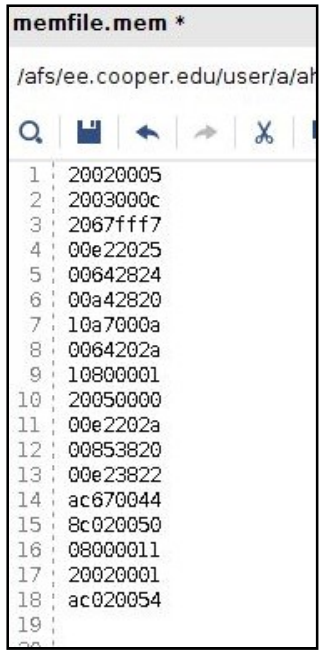
Name	Register #	Usage
$zero	0	The constant value 0
$at	1	Used by assembler
$vo-$v1	2-3	Values for results and expression evaluation
$a0-$a3	4-7	Arguments
$t0-$t7	8-15	Temporaries
$s0-$s7	16-23	Saved
$t8-$t9	24-25	More temporaries
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return pointer

MIPS Single-Cycle Full



MIPS Single Cycle: Main Blocks





Input/Output

```
module datapath(input clk, reset,
                input memtoreg, pcsrc,
                input  alusrc, regdst,
                input regwrite, jump,
                input [2:0] alucontrol,
                output zero,
                output [31:0] pc,
                input [31:0] instr,
                output [31:0] aluout, writedata,
                input [31:0] readdata,
                input [4:0] memAddr,
                output [31:0] memOut);
```

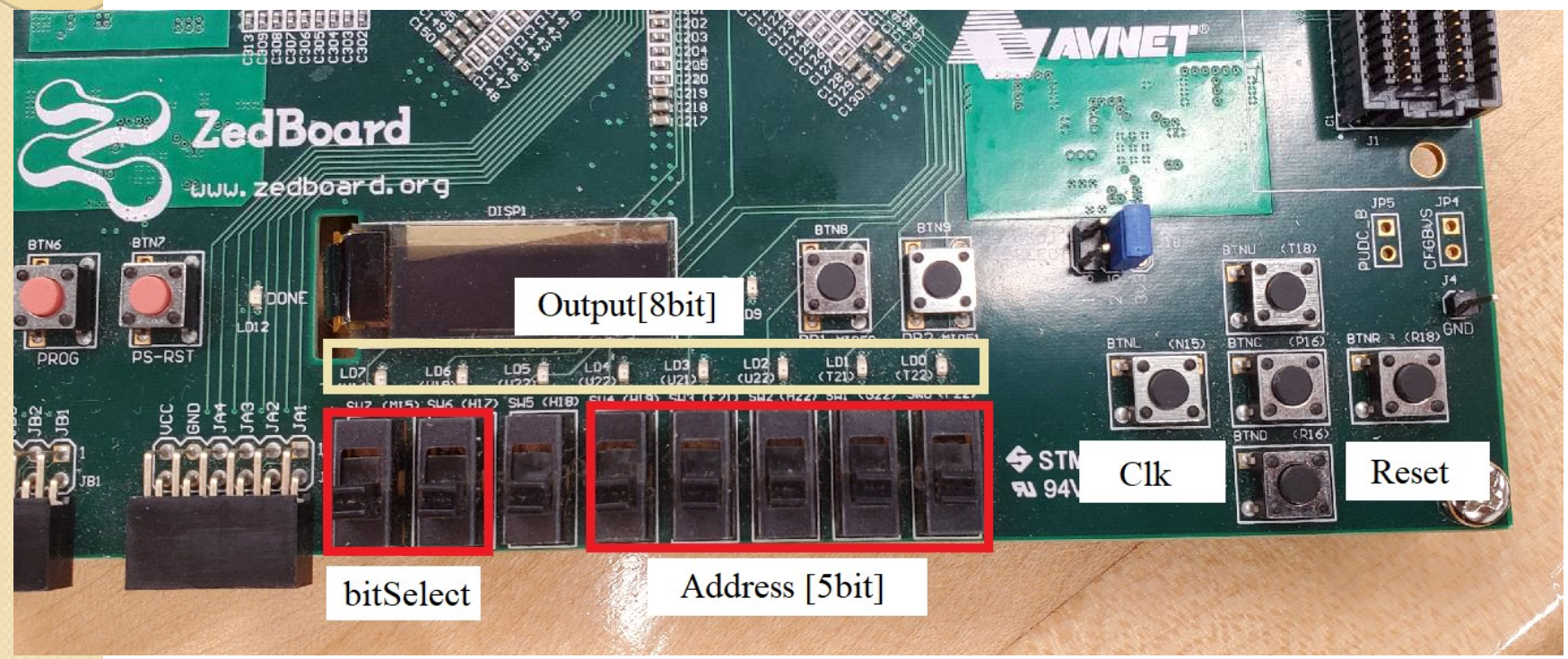
test.v *

/afs/ee.cooper.edu/user/a/ahmad.malik/Midterm/Midterm.srcs/sources_1/new/test.v

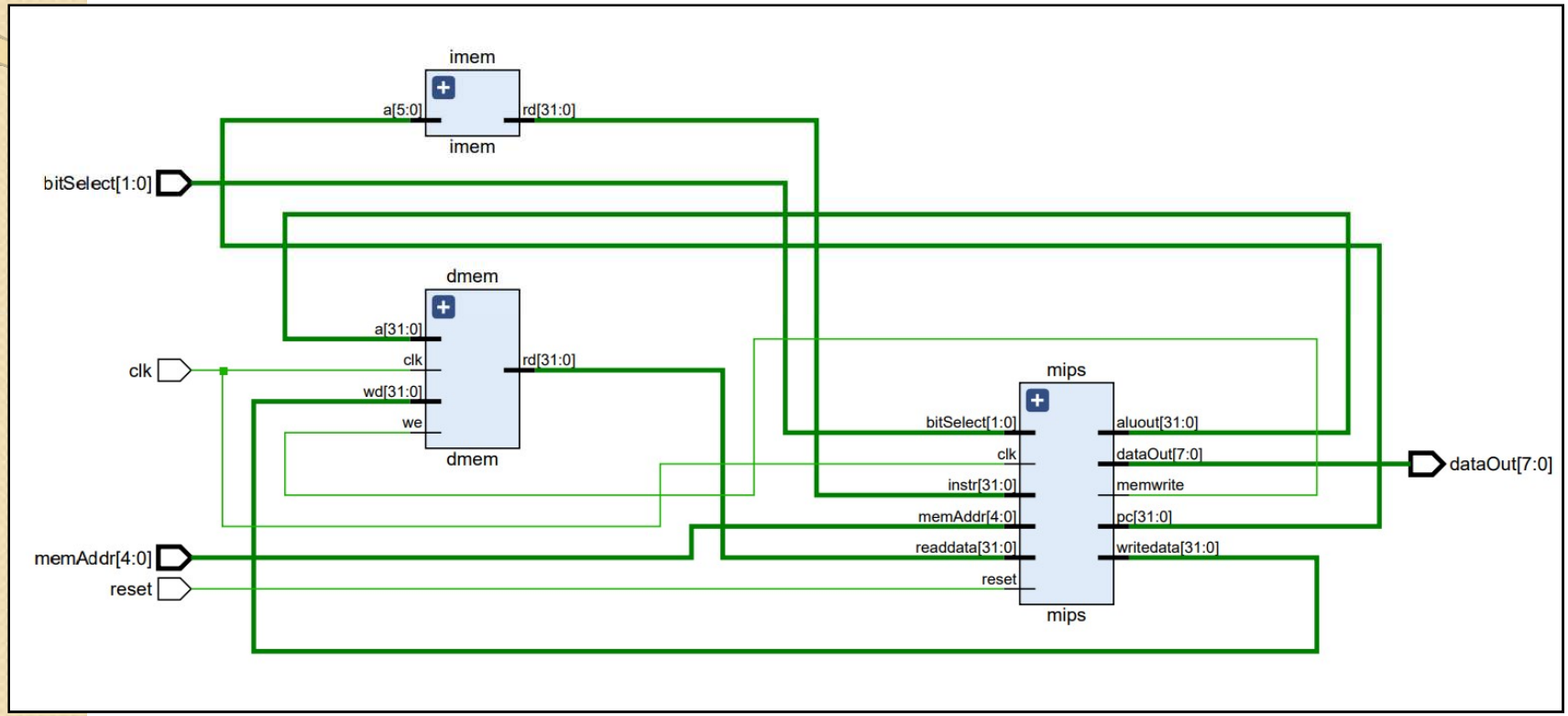
```
1  module test(input [1:0] bitSelect, input [31:0] memOut, output reg [7:0] dataOut);
2      always @(*)
3      begin
4          case(bitSelect)
5              2'b00:
6                  dataOut <= memOut[7:0];
7              2'b01:
8                  dataOut <= memOut[15:8];
9              2'b10:
10                 dataOut <= memOut[23:16];
11             2'b11:
12                 dataOut <= memOut[31:24];
13         endcase
14     end
15 endmodule
16
```

Modified the data path to support an extra two arguments that serve as a way to output the contents of a 32bit register given a 5bit address of a register. That way, we can see when the register has value 7 in register $2. Since Zedboard has 8 LEDS, we need to split 32bits into 4 sections so we can view it.

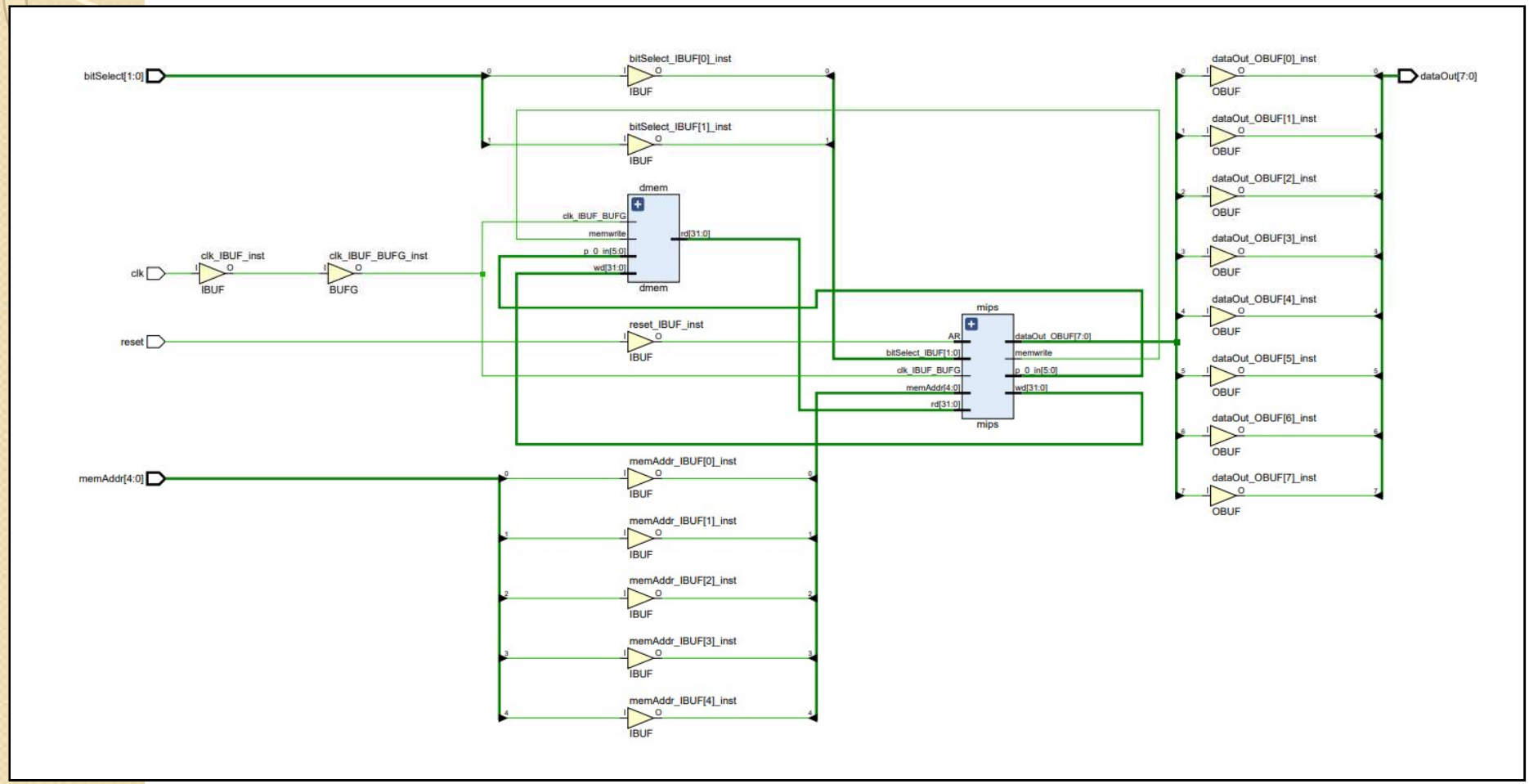
Input/Output



RTL Schematic



Synthesized/Implemented Schematic



Unable to Generate Bitstream!

Implementation (3 errors)

Place Design (3 errors)

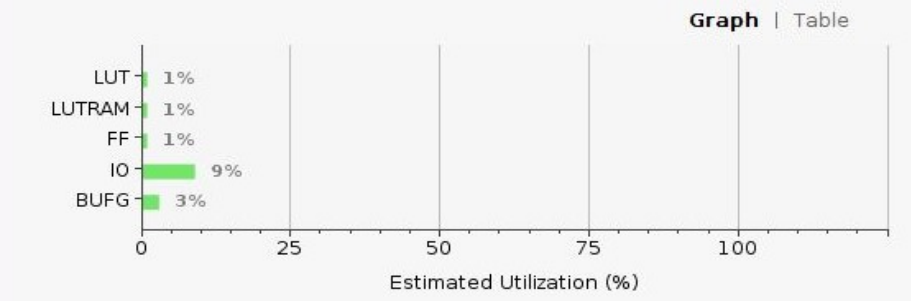
[Place 30-574] Poor placement for routing between an IO pin and BUFG. If this sub optimal condition is acceptable for this design, you may use the CLOCK_DEDICATED_ROUTE constraint in the .xdc file to demote this message to a WARNING. However, the use of this override is highly discouraged. These examples can be used directly in the .xdc file to override this clock rule.
< set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF] >

clk_IBUF_inst (IBUF.O) is locked to IOB_X1Y62
and clk_IBUF_BUFG_inst (BUFG.I) is provisionally placed by clockplacer on BUFGCTRL_X0Y31

[Place 30-99] Placer failed with error: 'IO Clock Placer failed'
Please review all ERROR, CRITICAL WARNING, and WARNING messages during placement to understand the cause for failure.

[Common 17-69] Command failed: Placer could not place all instances

Resource	Estimation	Available	Utilization %
LUT	431	53200	0.81
LUTRAM	104	17400	0.60
FF	6	106400	0.01
IO	17	200	8.50
BUFG	1	32	3.13



Single Cycle TestBench without I/O