文档版本： Ver1.0
最后修改日期： 2015-05-30
修改人： William

E 课网 – UVM 实战培训

www.eecourse.com

klin@eecourse.com

SVV 实验 – DUT 手册

1. 简介

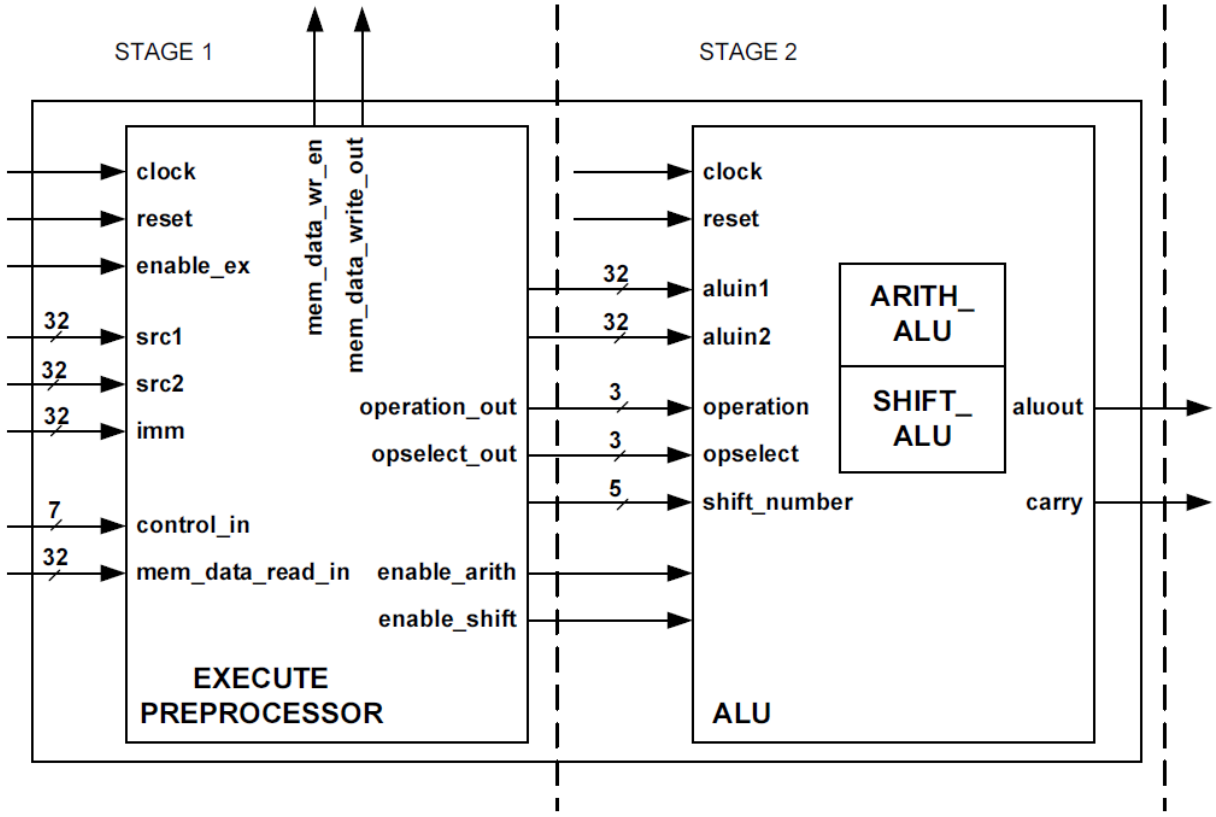


图 1 – DUT 顶层示意图

如图 1 所示，该 DUT 是由两个子模块构成的具有两级流水的系统模块，第一级为一个“预处理器”（EXECUTE PREPROCESSOR）模块，第二级为一个“算术逻辑运算单元”（ALU）模块：

- 预处理器 – 对外部输入的数据和控制信号进行处理，并产生对应的 ALU 可接受的数据和控制信号。
- 算术逻辑运算单元 – 根据输入的 opselect，operation 和 shift_number 信号执行相关算术逻辑运算。

这两个子模块都可以具有各自独立的时钟和复位信号，在该试验中，这两个子模块的时钟和复位信号是相同的。

DUT 代码文件：

```
data_defs.v top.v  Ex_Preproc.vp ALU.vp Shift_ALU.vp Arith_ALU.vp
```

顶层信号列表：

信号名称	位宽	输入/输出	时钟域	信号描述
clock	1	Input	-	时钟信号
reset	1	Input	clock	复位信号，高电平有效
enable_ex	1	Input	clock	使能信号，高电平有效
src1	REGISTER_WIDTH	Input	clock	输入数据信号 1
src2	REGISTER_WIDTH	Input	clock	输入数据信号 2
imm	REGISTER_WIDTH	Input	clock	多功能数据&控制信号
control_in	7	Input	clock	指令信号
mem_data_read_in	REGISTER_WIDTH	Input	clock	由 memory 输出的数据信号
mem_data_write_out	REGISTER_WIDTH	Output	clock	输出到 memory 的数据信号
mem_write_en	1	Output	clock	写 memory 使能信号，低电平有效
aluout	REGISTER_WIDTH	Output	clock	ALU 计算结果输出数据信号
carry	1	Output	clock	ALU 计算结果进位信号

2. 系统功能描述

2.1 组合逻辑的输出

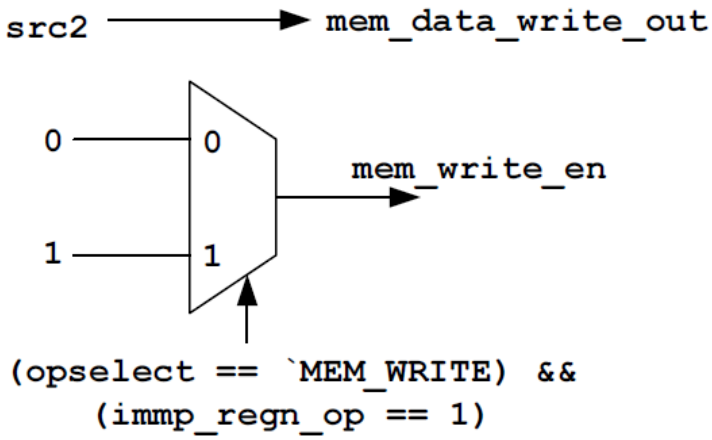


图 2 – Memory 写操作逻辑

图 2 所示的逻辑是为了实现 memory 的写操作功能，**当进行 memory 写操作时，ALU 阶段将会被跳过，**也就是说，只有预处理器参与该操作，除此之外的所有其他合法的操作指令，均需要经过预处理器和 ALU。

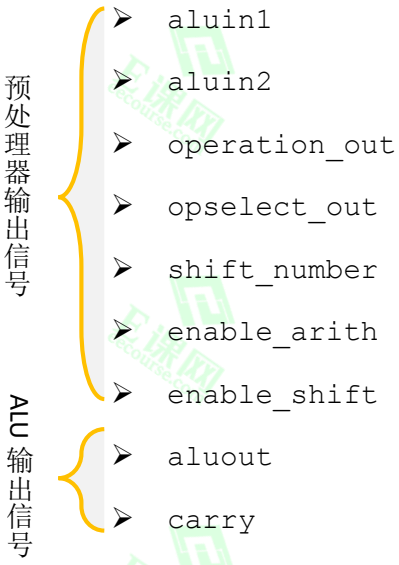
图 2 中的 operation 和 immp_regn_op 是输入信号 control_in 中的域，如下表所示：

control_in	[6:4]	3	[2:0]
域名	operation	imp_regn_op	opselect
Memory 读操作条件	XXX	1	MEM_WRITE(3'b100)

也就是当 control_in = 7'bxxx1100 时，表示进行 Memory 的写操作，写的数据来自于 src2 的输入值。在执行 Memory 的写操作时，ALU 不会参与该操作。注意，mem_write_en 信号为低电平有效，当为 0 时，表示 Memory 写操作。

2.2 复位

复位时，所有的时序逻辑的输出都为 0。这些信号包括：



2.3 预处理器功能描述

所有的时序逻辑的输出只有在 enable_ex==1 的条件下才会发生变化。该子模块的所有时序逻辑的输出真值表如下：

aluin1

enable_ex	aluin1
0	保持不变
1	src1

aluin2

enable_ex	opselect	control_in[3]	aluin2
0	X	X	保持不变
1	ARITH_LOGIC	0	src2
1	ARITH_LOGIC	1	imm
1	MEM_READ	0	保持不变
1	MEM_READ	1	mem_data_read_in
对于其他的信号组合，**aluin2** 均保持不变			

operation_out

enable_ex	operation_out
0	保持不变
1	control_in[6:4]

opselect_out

enable_ex	opselect_out
0	保持不变
1	control_in[2:0]

shift_number

enable_ex	opselect	imm[2]	shift_number
0	X	X	保持不变
1	SHIFT_REG	0	imm[10:6]
1	SHIFT_REG	1	src2[4:0]
对于其他的信号组合，**shift_number** 为 0			

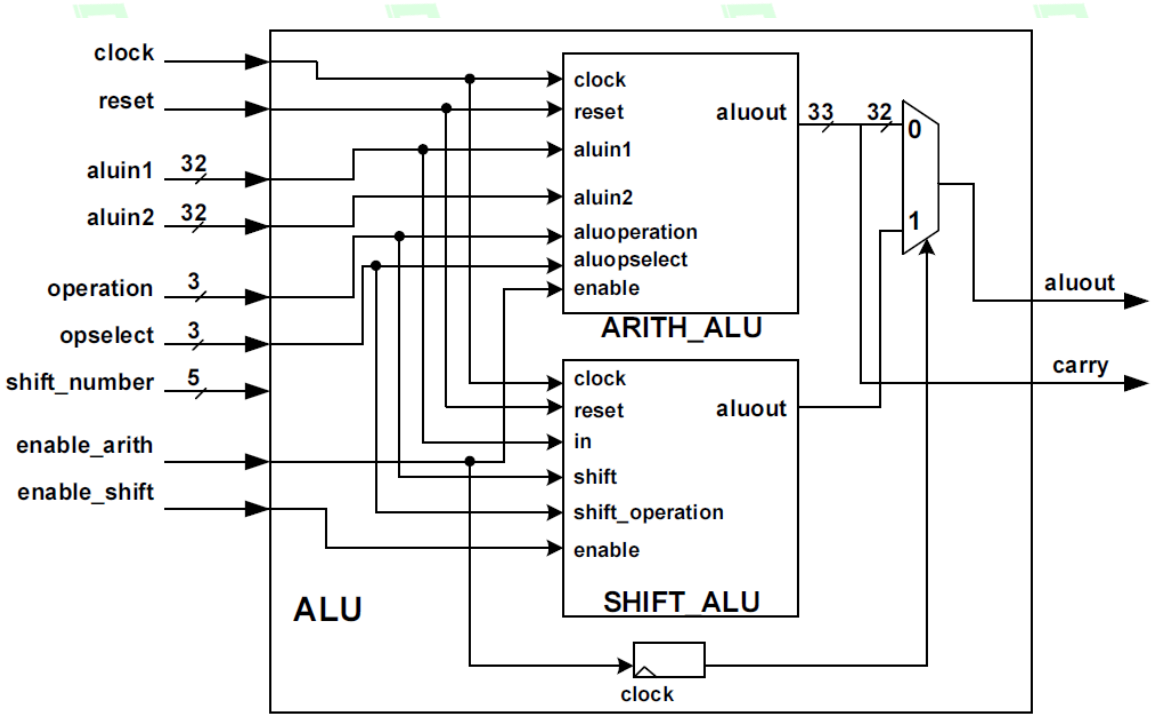
enable_arith

enable_ex	opselect	control_in[3]	enable_arith
0	X	X	0
1	ARITH_LOGIC	0	1
1	ARITH_LOGIC	1	1
1	MEM_READ	0	0
1	MEM_READ	1	1
对于其他的信号组合，**enable_arith** 为 0			

enable_shift

enable_ex	opselect	enable_shift
0	X	0
1	SHIFT_REG	1
对于其他的信号组合，**enable_number** 为 0		

2.4 ALU 功能描述



所有的指令信号只能在以下两个条件成立之一才是有效的：

- **enable_arith == 1** 或者
- **enable_shift == 1**

这些信号均在时钟的上升沿处进入 ALU 子模块。ALU 可以执行移位操作（SHIFT_ALU）和算术逻辑运算（ARITH_ALU）。操作指令以及输出如下表所示：

SHIFT_ALU

enable_shift	shift_operation[1:0]	aluout
0	X	保持不变
1	SHLEFTLOG （逻辑左移）	in << shift; 最低位补 0; carry = 0;
1	SHLEFTART （算术左移）	in << shift; 最低位补 0; 如果 in 是一个正数,carry = 0, 如果 in 是一个负数,carry = 1
1	SHRGHTLOG （逻辑右移）	in >> shift; 最高位补 0; carry = 0;
1	SHRGHTART （算术右移）	in >> shift; 最高位为符号位; carry = 0;
对于其他的信号组合，**aluout** 不变		

ARITH_ALU

enable_arith	aluopselect	operation	aluout
0	X	X	保持不变
1	ARITH_LOGIC	ADD	带符号加法 aluin1 + aluin2;
1	ARITH_LOGIC	HADD（Half Word add）	{h_carry, h_add[15:0]} = aluin1[15:0] + aluin2[15:0] carry = h_carry; aluout = **sxt**①(h_add);
1	ARITH_LOGIC	SUB	带符号减法 aluin1 – aluin2;
1	ARITH_LOGIC	NOT	对 aluin2 按位取反; carry = 0;
1	ARITH_LOGIC	AND	对 aluin1 和 aluin2 按位与; carry = 0;
1	ARITH_LOGIC	OR	对 aluin1 和 aluin2 按位或; carry = 0;
1	ARITH_LOGIC	XOR	对 aluin1 和 aluin2 按位异或; carry = 0;
1	ARITH_LOGIC	LHG	aluout[31:16] = aluin2[15:0]; aluout[15:0] = 16'h0; carry = 0;
1	MEM_READ	LOADBYTE	**signext**②{aluin2[7:0]}; carry = 0;
1	MEM_READ	LOADBYTEU	**zeropad**③{aluin2[7:0]}; carry = 0;
1	MEM_READ	LOADHALF	signext{aluin2[15:0]}; carry = 0;
1	MEM_READ	LOADHALFU	zeropad{aluin2[15:0]}; carry = 0;
1	MEM_READ	LOADWORD	aluin2; carry = 0;
1	MEM_READ	Ohters	aluin2; carry = 0;
对于其他的信号组合，**aluout** 和 **carry** 保持不变			

① sxt()：符号位扩展。

举个例子来说（其中 M > N）：如果 out[M:0] = sxt(in[N:0])，（注意 in 只有 N+1 位）则 out[N:0] = in[N:0]并且 out[M:N+1]每一位均为 in[N]。

② signext{}：符号位扩展。

举个例子来说（其中 M > N）：如果 out[M:0] = signext{in[N:0]}，（注意 in 可以大于 N+1 位）则 out[N:0] = in[N:0] 并且 out[M:N+1]每一位均为 in[N]。

③ zeropad{}：高位补 0。

举个例子来说（其中 M > N）：如果 out[M:0] = zeropad{in[N:0]}，（注意 in 可以大于 N+1 位）则 out[N:0] = in[N:0] 并且 out[M:N+1]每一位均为 0。

附录

常量宏定义

```
// 时钟周期定义
`define  CLK_PERIOD          10

// 系统指令和数据位宽定义
`define  REGISTER_WIDTH      32
`define  INSTR_WIDTE         32
`define  IMMEDIATE_WIDTH     16

// 系统操作类型定义
`define  MEM_READ            3'b101
`define  MEM_WRITE           3'b100
`define  ARITH_LOGIC         3'b001
`define  SHIFT_REG           3'b000

// 算术逻辑运算定义
`define  ADD                 3'b000
`define  HADD                3'b001
`define  SUB                 3'b010
`define  NOT                 3'b011
`define  AND                 3'b100
`define  OR                  3'b101
`define  XOR                 3'b110
`define  LHG                 3'b111

//移位运算定义
`define  SHLEFTLOG           3'b000
`define  SHLEFTART           3'b001
`define  SHRGHTLOG           3'b010
`define  SHRGHTART           3'b011

// 数据传输定义
`define  LOADBYTE            3'b000
`define  LOADBYTEU           3'b100
`define  LOADHALF            3'b001
`define  LOADHALFU           3'b101
`define  LOADWORD            3'b011
```