Trabalho de Verificação – INF01194

1. Tarefa: Monte um ambiente de verificação para um ULA
2. Circuito a ser verificado.

- O circuito a ser verificado será uma Unidade Lógica Aritmética (ULA/ALU) capaz de realizar 8 operações, as quais estão definidas na Tabela 1 deste documento.
- A ULA deverá receber 2 valores de entrada de dados com larguras configuráveis (utilize **16** bits para teste).
- A ULA deverá retornar, através do seu sinal de saída de dados, o resultado da operação aritmética solicitada através do seletor.
- Sinais de validade (i.e., indicadores dados válidos) foram colocados nas entradas e saídas do circuito.

A. Mapa do Seletor

Tag	Equation	Name	Code
ALU_ADD	data_1 + data_2	Sum	000
ALU_SUB	data_1 – data_2	Subtraction	001
ALU_MULT	data_1 * data_2	Multiplication	010
ALU_LSH	data_1 << data_2	Left shift	011
ALU_RSH	data_1 >> data_2	Right shift	100
ALU_INCR	data_X + 1	Increment	101
ALU_DECR	data_X – 1	Decrement	110

B. Interfaces

i. Parameters

Parameter Name	Description
DATA_WIDTH	Width for the data input signals.
SEL_WIDTH	Width for the ALU selector.

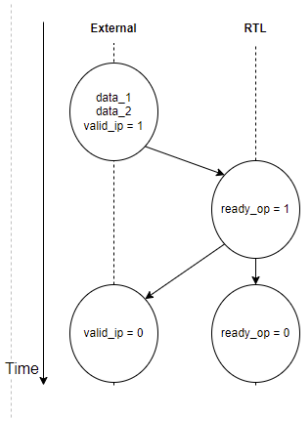
ii. Signals

Signal Name	Width (bits)	Direction	Description
Control Signals			
clk	1	Input	General clock signal.
rst	1	Input	General reset signal.
Input Interface			
valid_ip	1	Input	Flag for valid input data.
sel_ip	SEL_WIDTH	Input	ALU operation selector.
data_ip_1	DATA_WIDTH	Input	Input for Data 1.
data_ip_2	DATA_WIDTH	Input	Input for Data 2.
ready_op	1	Output	Ack for data received.
parity_ip	1	Input	Parity for data + sel.
Output Interface			
valid_op	1	Output	Flag for valid output data.
data_op	DATA_WIDTH*2	Output	Output for final data.

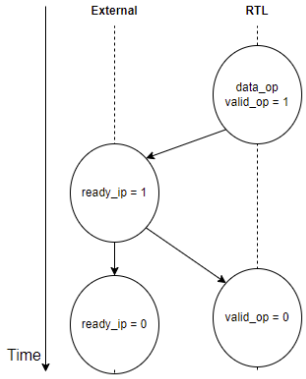
ready_ip	1	Input	Ack for data collected externally.
err_op	1	Ouptup	Flag for error detected.

C. Comportamento

iii. Input Data



iv. Output Data



v. Descrição

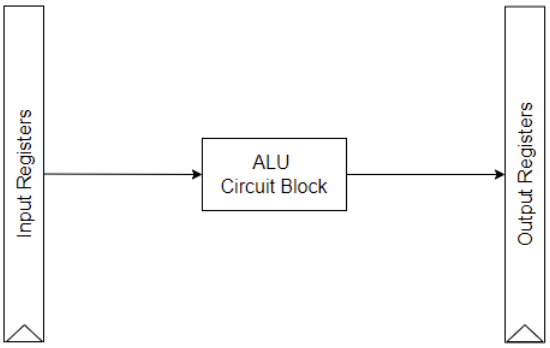
A ULA utilizada para este projeto é equipada com um sistema de validação de recebimento. Para isso, sempre que um componente externo estiver com os sinais de entrada corretamente definidos, esse componente deverá colocar o valor ‘1’ no sinal “valid_ip”.

O RTL, por sua vez, colocará o valor ‘1’ no seu sinal de saída “ready_op” apenas quando estiver pronto para receber os valores de entrada. Em outras palavras, caso o RTL tenha recebido valores em rodadas anteriores e ainda não conseguiu processar tais valores (i.e., os valores ainda estão salvos nos registradores de entrada), então o sinal “ready_op” será mantido em ‘0’.

Da mesma forma que a entrada, a saída apresenta um mecanismo de validação semelhante. Sendo assim, um valor só será retirado dos registradores de saída (i.e., que serão liberados para os próximos valores) quando o sinal de “ready_ip” for definido para ‘1’.

Este circuito não possui qualquer tipo de buffer interno, o que significa que uma entrada é processada apenas quando uma saída for lida.

O circuito interno, que processa as operações, é um circuito combinacional baseado em multiplexadores. O diagrama abaixo apresenta a estrutura interna da ULA.

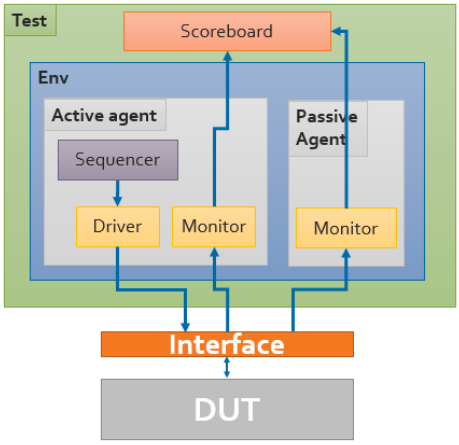


A ULA possuiu um sistema de verificação de paridade (**even parity**) das entradas data_1, data_2 e sel através do sinal parity_ip. Ela realiza um XOR de todos os bits de data_1, data_2 e sel, e caso o valor seja diferente da entrada parity_ip, a flag err_op é acionada. Alguns exemplos de casos:

data_1	data_2	sel	No 1-bits	parity_ip	err_op
00000000	00000000	000	0 (par)	0	0
00000000	00000000	000	0 (par)	1	1
00000000	00000000	010	1 (ímpar)	0	1
01100000	00000001	011	5 (ímpar)	1	0
01100000	00010001	011	6 (par)	1	1

3. Detalhes

A. Ambiente de Verificação



B. Configurações

i. O ambiente UVM a ser criado deverá ter agentes operando em **modos UVM_ACTIVE e UVM_PASSIVE**.

C. Detalhes dos Componentes

vi. Item (TX)

Deverá gerar, através de randomizações e *contraints*, todas as possibilidades de dados de entrada. As *constraints* que devem estar presentes são as seguintes:

- **data_2 <= data_1** para ALU_SUB.
- **data_2 <= DATA_WIDTH** para ALU_RSH e ALU_LSH
- **data_2 == 0 || data_2 == 1** para ALU_INCR e ALU_DECR
- **ALU_ADD** é 3x mais frequente que outras operações
- **data_2 <= 63** para ALU_MULT

vii. Sequência

Deverão ser feitas duas sequências (duas classes) uma **simple_seq** e uma **err_seq**. A diferença entre elas é que na **err_seq o bit de paridade DEVE estar errado**, na **simple_seq tanto faz** (totalmente randomizado).

Dica: Essa condição não deve adicionar nenhuma *constraint* adicional na classe do item. Recomenda-se utilizar uma **constraint in-line na err_seq**. Exemplo:

```
m_item = empty_tx::type_id::create("m_item");
start_item(m_item);
assert(m_item.randomize() with { /* CONSTRAINT DO BIT DE PARIDADE */}) else
  `uvm_fatal("SEQ_RAND", $sformatf("Unable to randomize for %s",
             get_full_name()))
// m_item.print();
m_item.print_2();
finish_item(m_item);
```

viii. Driver

a. *Active*

- O driver em modo **UVM_ACTIVE** deverá obter transações da sequência e definir os sinais de entrada do circuito através da interface.
- O driver em modo **UVM_ACTIVE** deverá manter os sinais de entrada válidos até o recebimento de ready_op = ‘1’.
- Um ciclo de clock após ready_op e valid_ip ficarem simultaneamente em ‘1’, valid_ip deverá ser definido como ‘0’.
- Nenhum novo dado poderá ser colocado nas entradas mesmo ciclo de clock em que valid_ip foi definido como ‘0’.

b. *Passive*

- O driver em modo UVM_PASSIVE deverá ler o sinal de saída valid_op e enviar ready_ip definido em ‘1’.
- Um ciclo de clock após ready_ip e valid_op ficarem simultaneamente em ‘1’, ready_op deverá ser definido para ‘0’.
- Não é esperado que nenhum novo dado seja colocado nas portas de saída do RTL no mesmo ciclo de clock em que ready_ip foi definido para ‘0’.

ix. Monitor

c. *Active*

- Deverá criar um item do tipo alu_tx e colocar os valores de **entrada** da arquitetura (i.e., Input Signals) dentro desse item. Para obter tais valores, a interface deverá ser lida a cada ciclo de clock.

- Apenas valores válidos deverão ser colocados na analysis port e, portanto, valid_ip deverá ser verificado.

d. Passive

- Deverá criar um item do tipo alu_tx e colocar os valores de **saída** da arquitetura (i.e., Output Signals) dentro desse item. Para obter tais valores, a interface deverá ser lida a cada ciclo de clock.
- Apenas valores válidos deverão ser colocados na analysis port e, portanto, valid_op deverá ser verificado.

x. Agente

- Em modo UVM_ACTIVE, **sequencer, driver e monitor** deverão ser criados.
- Em modo UVM_PASSIVE, apenas **driver e monitor** deverão ser criados.

xi. Env

- Deverá criar o **alu_agt_inputs** e **alu_agt_outputs** que operarão em modos, respectivamente, UVM_ACTIVE e UVM_PASSIVE.

i. Scoreboard

- Não será necessária a implementação de um Scoreboard para este trabalho.
- O objeto Scoreboard não precisará ser criado.