UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CENTRO MULTIDISCIPLINAR DE PAU DOS FERROS
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA

LABORATÓRIO DE CIRCUITOS DIGITAIS

Verilog – Abordagem por Fluxo de Dados (Parte 1)

Prof.: Pedro Thiago Valério de Souza
UFERSA – Campus Pau dos Ferros
pedro.souza@ufersa.edu.br

Descrição por Fluxo de Dados

- Descreve o circuito em termos de expressões (lógicas e aritméticas) que relacionam entradas e saídas;
- Utiliza a diretiva **assign**;

```
assign <saida ou net> = (constante, segmento de vetor ou
                         expressão);
```

- Não pode-se utilizar o **assign** para mudar valores de dados do tipo **reg**.

Descrição por Fluxo de Dados

- **Atribuição de Constantes:**
  - Declaração:

```
<tamanho>'<base><valor>
```

  - Tamanho: Quantidade de *bits*;
  - Base:
    - `d` ou `D`: Decimal;
    - `h` ou `H`: Hexadecimal;
    - `b` ou `B`: Binário;
    - `o` ou `O`: Octal.

Descrição por Fluxo de Dados

- **Atribuição de Constantes:**
  - Exemplos:

```
assign x = 16'd255; //Decimal de 16 bits com valor 255
assign y = 8'hA0;   //Hexadecimal de 8 bits com valor A0
assign z = 4'b1001; //Binário de 4 bits com valor 1001
assign w = 6'o12;   //Octal de 6 bits com valor 12
```

Descrição por Fluxo de Dados

- **Atribuição de Constantes:**
  - Observações:
    - Quando não explicitado o tamanho e o tipo, suponha-se um valor do tipo **decimal** de **32 *bits***;

```
assign x = 12;      //Decimal de 32 bits com valor 12
```

    - Para representar valores negativos (em complemento de dois) coloca-se um sinal de menos **antes da especificação do tamanho**;

```
assign y = -8'd12;  //Decimal negativo de 8 bits (em
                    //complemento de dois)
```

Descrição por Fluxo de Dados

- **Atribuição de Constantes:**
  - Caracteres Especiais:
    - *Undeline* (_): pode ser utilizado para melhorar a legibilidade das atribuições;

```
assign x = 8'b1001_0011;   //Binário de 8 bits
```

    - *X ou x*: Valor desconhecido;

```
assign y = 8'hAx;          //Nibble menos significativo
                           //desconhecido
```

    - *Z ou z*: Alta impedância.

Descrição por Fluxo de Dados

- **Atribuição de um segmento de vetor:**
  - Declaração:

```
assign x = y[inicio:fim];
```

  - No caso, *x* tem tamanho `(inicio-fim)+1`;
  - Exemplos:

```
assign x = y[5:2];
assign z = y[1:0];
```

Descrição por Fluxo de Dados

- **Atribuição a partir de Expressões:**
  - Operadores Lógicos;
    - *Bit-wise*;
    - Redução;
    - Lógicos;
  - Operadores Aritméticos;
  - Operadores de Concatenação e Replicação;
  - Operadores de Deslocamento;
  - Operadores Relacionais;

Descrição por Fluxo de Dados

- **Operações Lógicas:**
  - Operadores *Bit-wise*:
    - São aplicados *bit*-a-*bit* dos operandos;

Operador	Operação Realizada	Exemplo: *a* = 4'b1010 *b* = 4'b0011
~	NOT	~*a* = 4'b0101
&	AND	*a* & *b* = 4'b0010
|	OR	*a* | *b* = 4'b1011
^	XOR	*a* ^ *b* = 4'b1001
~^ ou ^~	XNOR	*a* ~^ *b* = 4'b0110

Descrição por Fluxo de Dados

- **Operações Lógicas:**
  - Operadores *Bit-wise*:
    - Exemplos:

```
assign x = ~a;                  //O operador NOT admite
                                //apenas um operando!
assign y = a & b;
assign z = a | b;
assign q = a & b & c;
assign w = a ^ b;
assign r = a ~^ b;
```

Descrição por Fluxo de Dados

- **Operações Lógicas:**
  - Operadores *Bit-wise*:
    - Observações:
      - Os operadores *Bit-wise* valem mesmo que os operandos tenham apenas um *bit*;
      - A quantidade de *bits* da saída de um operador *Bit-wise* é igual a quantidade de *bits* do operando mais longo;
        - Preenchimento com zeros;

Descrição por Fluxo de Dados

- **Operações Lógicas:**
  - Operadores de Redução:
    - Possuem apenas um operando e reduzem um vetor a um único *bit*;
    - A operação é feita com os *bits* do operando;

Operador	Operação Realizada	Exemplo: *a* = 4'b1010
&	AND	&*a* = 1'b0
~&	NAND	~&*a* = 1'b1
|	OR	|*a* = 1'b1
~|	NOR	~|*a* = 1'b0
^	XOR	^*a* = 1'b0
~^ ou ^~	XNOR	~^*a* = 1'b1

Descrição por Fluxo de Dados

- **Operações Lógicas:**
  - Operadores de Redução:
    - Exemplos:

```
assign x = &a;
assign y = ~&a;
assign z = |a;
assign q = ~|a;
assign w = ^a;
assign e = ~^a;
```

Descrição por Fluxo de Dados

- **Operações Lógicas:**
  - Operadores Lógicos:
    - Resultam em um único *bit*, independente da quantidade de *bits* das entradas;
    - Se o dado é diferente de zero, a saída é 1;
    - Se o dado é igual a zero, a saída é 0;

Operador	Operação Realizada	Exemplo: *a* = 3'b101 (=1b) *b* = 3'b000 (=0b)
!	NOT	!*a* = 1'b0 !*b* = 1'b1
&&	AND	*a* && *b* = 1'b0
||	OR	*a* || *b* = 1'b1

Descrição por Fluxo de Dados

- **Operações Lógicas:**
  - Operadores Lógicos:
    - Exemplos:

```
assign x = a && b;
assign y = a || b;
assign z = !a;
```

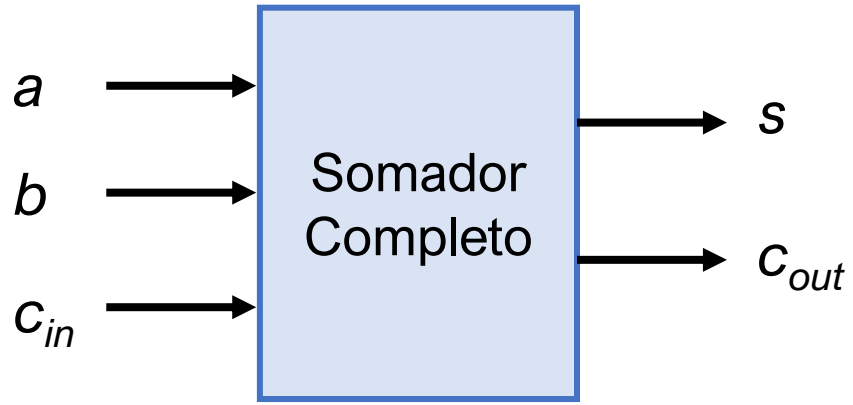
Descrição por Fluxo de Dados

Projeto 03 – Somador completo (revisto)

Descreva o circuito corresponde ao somador completo em Verilog utilizando a abordagem por fluxo de dados.

$$s = a \oplus b \oplus c_{in}$$

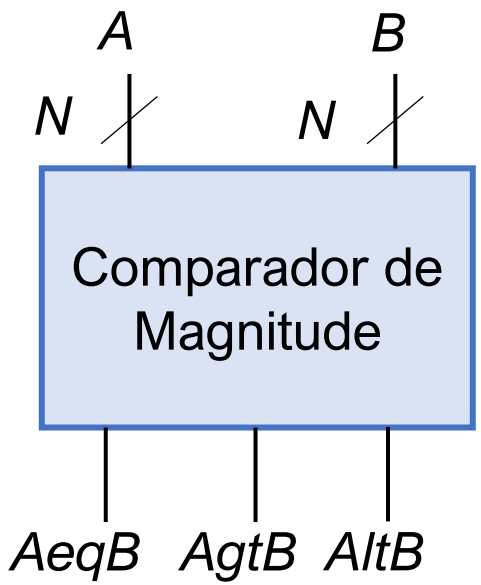
$$c_{out} = ab + ac_{in} + bc_{in}$$



Descrição por Fluxo de Dados

Projeto 04 – Comparadores de magnitude

Projete, utilizando o Verilog, um circuito que se comporte como um comparador de magnitude de 4 *bits*. O circuito possui duas entradas, denominadas de *A* e *B*, ambas de 4 *bits*. O circuito possui três saídas, denominas de *AeqB*, *AgtB* e *AltB*, que indicam quando, respectivamente, *A*=*B*, *A*>*B* e *A*<*B*.



$x_3 = (A_3 \oplus B_3)' = A_3'B_3' + A_3B_3$

$x_2 = (A_2 \oplus B_2)' = A_2'B_2' + A_2B_2$

$x_1 = (A_1 \oplus B_1)' = A_1'B_1' + A_1B_1$

$x_0 = (A_0 \oplus B_0)' = A_0'B_0' + A_0B_0$

$$AeqB = x_3x_2x_1x_0$$

$$AgtB = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

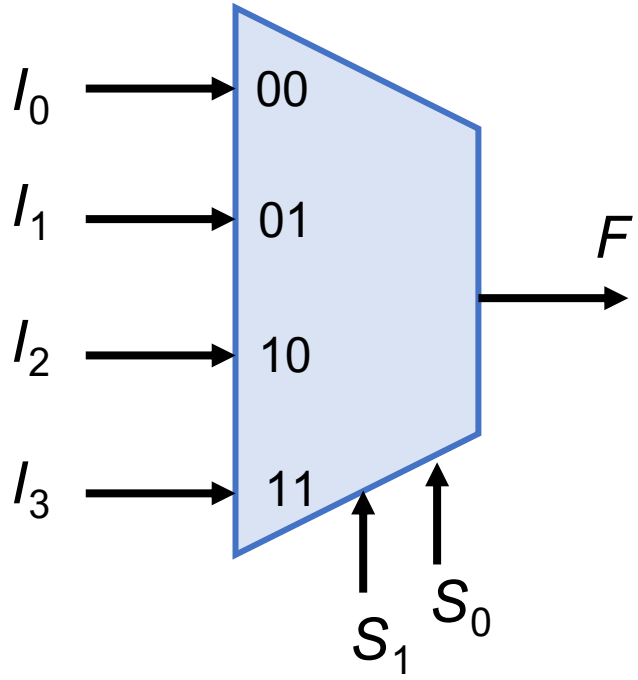
$$AltB = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

Descrição por Fluxo de Dados

Projeto 05 – Multiplexador 4x1

Descreva o circuito corresponde ao multiplexador 4x1 em Verilog utilizando a abordagem por fluxo de dados.

$$F = I_0S_1'S_0' + I_1S_1'S_0 + I_2S_1S_0' + I_3S_1S_0$$

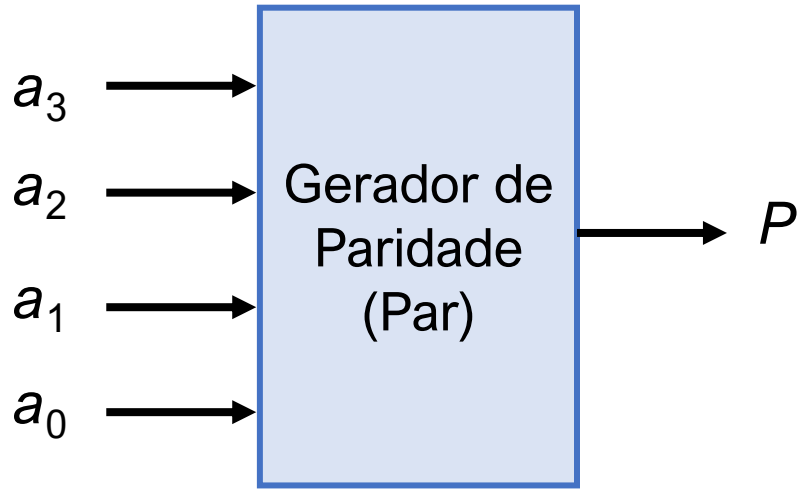


Descrição por Fluxo de Dados

Projeto 06 – Gerador de Paridade (Par)

Descreva o circuito corresponde a um gerador de paridade par em Verilog utilizando a abordagem por fluxo de dados.

$$P = a_3 \oplus a_2 \oplus a_1 \oplus a_0$$



Referências

- **PIMENTA, TALES CLEBER – Circuitos Digitais – Análise e Síntese Lógica: Aplicações em FPGA. Elsevier, 2017.**
- Altera’s Verilog HDL Basics (https://www.youtube.com/watch?v=PJGvZSIsLKs)
- VAHID, FRANK - Sistemas Digitais - Projeto, Otimização e HDLs. Bookman, 2008.