Projeto Somador Serial Implementado no SystemVerilog e no Logisim

Josué Rodrigues
Instituto Federal do Ceará(IFCE)
Av. Parque Central, 1315 - Distrito Industrial I, Maracanau - Ceará, 61939-140, Brazil

João Victor de Lima Pereira
Instituto Federal do Ceará(IFCE)
Av. Parque Central, 1315 - Distrito Industrial I, Maracanau - Ceará, 61939-140, Brazil

João Pedro Moreira
Instituto Federal do Ceará(IFCE)
Av. Parque Central, 1315 - Distrito Industrial I, Maracanau - Ceará, 61939-140, Brazil

***Resumo*— Este trabalho demonstra a construção de um Somador Serial utilizando Logisim e SystemVerilog. Utilizando-se de apenas um Somador Completo e um Flip-Flop tipo D para realizar a adição binária bit a bit. Os resultados demonstraram o funcionamento esperado do circuito, demonstrando a eficácia do design sugerido e utilizando poucos componentes lógicos.**

Pelavras-Chaves—Somador Serial, Somador Completo, Flip-Flop D)

I. Introdução

Em circuitos digitais, a operação aritimética de soma é importantíssima por atuar como base para demais operações. Nesse contexto, o somador paralelo é aquele que os bits são precessados simultaneamente na entrada do somador [1]. Alternativamente, temos o somador serial, em que os bits são somados um par de cada vez através de um único circuito de somador completo (FA), o carry out do somador completo é transferido para um flip-flop D. E a saída desse flip-flop é então usada como a entrada de carry para o próximo par de bits significativos [2].

II. Objetivo

Projetar um Somador Serial em SystemVerilog(SV) e em Logisim com as seguintes características: utilizar um somador completo e um Flip-Flop D, com entradas A, B, Clock e Reset e possuir saídas S e Cout.

III. Metodologia

Por meio de pesquisas e leitura de livros, como [1] e [2], primeiramente, desenvolvemos o circuito em Logisim, utilizando apenas componentes lógicos básicos como exigido no objetivo. Com a implementação do circuito no software ficou facilitado desenvolver o código em SystemVerilog e após isso criamos o cógio da Testbench para se conseguir retirar as formas de ondas e a tabela de transição de estados.

IV. Resultados

1. Circuito Logisim

Conseguimos desenvolver o seguinte circuito, veja figura 1. Temos entradas **A** e **B** que representam os bits que serão somados no somador completo e terão saída **S**. O somador completo adiciona esses bits junto com o Carry in. Além disso, o Carry Out(**Cout)** gerado em cada pulso(**clock**) é armazenado no **Flip-Flop D** que então alimenta o Carry In na próxima repetição. Por fim, há um sinal de **reset** para reiniciar os valores armazenados no Flip-Flop.

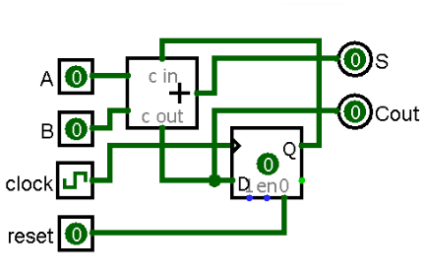

Figura 1. Circuito no Logisim do Somador Serial.

2. Código SystemVerilog

Com o circuito no Logisim, torna-se mais facilitado a implementação em SystemVerilog. Veja a figura 2.

```
module serialadder(
  input logic a, b,
  input logic clock, reset,
  output logic s, cout
);
  logic cin;
  always_ff @(posedge clock, posedge reset) begin
    if (reset) begin
      cin <= 0;
    end
    else begin
      cin <= cout;
    end
  end
  assign s = a ^ b ^ cin;
  assign cout = (a & b) | (a & cin) | (b & cin);
endmodule
```

Figura 2. Código escrito em SystemVerilog para o Somador Serial.

Aqui definimos o módulo **serialadder**, que inicia o encapsulamento lógico do circuito, com entradas **a**, **b**, **clock** e **reset**, e saídas **S** e **Count**. Também declaramos a variável **cin**. Na próxima linha declaramos um bloco **always_ff**, que descreve o flip-flop e que responde ao sinal do clock e do reset. Depois começamos uma estrutra condicional **if** que verifica se o reset está em 1, caso sim, "cin" recebe o valor zero. Se não, o "cout" da operação anterior fica guardado em cin para ser usado na próxima soma, é o que define o somador serial. Após isso, definimos a saída da soma e do "cout" como:

$$s = a \wedge b \wedge cin$$

$$cout = (a \,\&\, b) | \,(a \,\&\, cin) | \,(b \,\&\, cin)$$

3. Código da Testbench

Para testar o nosso código em SV precisamos desenvolver uma Testbench que teste todos os valores possíveis de entrada e saída. Com isso, foi desenvolvido o seguinte código da Testbench. Veja figura 3.

```
module tb_serialadder;
  logic a, b;
  logic clock, reset;
  logic s, cout;

  localparam N = 8; // Quantidade de casos teste
  logic [4:0] testvectors [N-1:0];
  logic tmp_a, tmp_b, cin, expected_s, expected_cout;

  serialadder dut(a, b, clock, reset, s, cout);
  always #5 clock = ~clock;

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, a, b, clock, reset, s, cout);
    $readmemb("testvectors", testvectors, 0, N-1);

    clock = 1; reset = 0; a = 0; b = 0; cin = 0;
    reset = 1; @(posedge clock); reset = 0;

    for (int i = 0; i < N; i++) begin
      {tmp_a, tmp_b, cin, expected_s, expected_cout} = testvectors[i];
      if (cin) begin
        // Faz com que o carry que vai para a soma seguinte seja 1
        a = 1; b = 1; @(posedge clock);
      end
      a = tmp_a; b = tmp_b; @(posedge clock);
      assert((s === expected_s) && (cout === expected_cout));
        else $error(
          "Erro no %do caso: s=%b expected_s=%b | cout=%b expected_cout=%b",
          i + 1, s, expected_s, cout, expected_cout
        );
      reset = 1; @(posedge clock); reset = 0;
    end
    $finish;
  end
endmodule;
```

Figura 3. Testbench para testar o circuito implementado em SV.

No código temos o módulo **tb_serialadder** que testará o **serialadder** já implementado. Definimos localparam N igual a 8 para o número de casos testes, além de declarar uma matriz **testvectors** com "N" elementos de 5 bits cada para armazenar os valores de "a","b","cin","expected_s" e "expected_cout". Após isso, declaramos variáveis temporárias para "a","b" e "cin", sendo, respectivamente, **tmp_a**, **tmp_b** e **cin** e para os calores esperados para "s" e "cout", sendo, **expected_s** e **expected_cout**. Além disso, iniciamos um módulo **serialadder** com o nome **dut**(Device Under Test) para conectar as entradas e saídas já declaradas. Logo após, definimos o bloco **always #5 clock = ~clock** para a cada segundo sinal é invertido 5 vezes.

Após definimos as variáveis e as condições podemos iniciar o nosso teste com o bloco **initial**. Ademais, **$dumpfile("dump.vcd")** e **$dumpvars(0, a, b, clock, reset, s, cout)** gera um arquivo que armazena as mudanças das variáveis. **$readmemb("testvectors", testvectors, 0, N-1)** carrega os vetores para o teste dos arquivos, preenchendo com "N" vetores de teste. Após definir as condições iniciais das variáveis, um laço é iniciado para repitir "N" vezes, para cada pulso os valores do vetor de teste atual são atribuídos às variáveis temporárias. Caso "cin" seja 1, os valores de "a" e de "b" se tornam 1 e espera que o valor saia de 0 para 1(borda de subida) do sinal "clock", simulando o comportamento quando o "carry in" é 1. Na sequência, os valores de de "a" e "b" assumem, respectivamente, "tmp_a" e "tmp_b" e espera por uma borda de subida no clock, nesse ciclo os valores de entrada são aplicados ao módulo "dut".

A partir da linha 28, **assert((s === expected_s) && (cout === expected_cout))** verifica se os valores de "s" e "cout" gerados pelo "dut" correspondem aos valores esperados, caso sejam diferentes há um erro gerado. No erro, é impresso uma mensagem detalhando o número do caso de teste que houve a falha e os valores dos sinais gerados e esperados.

Já na linha 33, o módulo "dut" é resetado após cada teste para seja reiniciado antes do próximo caso. **$finish** termina os testes após todos os casos serem executados.

4. Forma de Onda, Digital Js e Tabela de Transição de Estados

Com tudo finalizado podemos analisar nossos resultados gerando uma forma de onda e desenvolvendo uma tabela de transição de estados. Ao se utilizar o site **EDA Playground**, temos a seguinte forma de onda, veja figura 4.

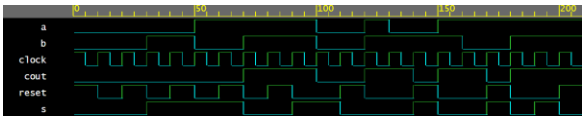

Figura 4. Forma de Onda gerada pelo EDA Playground.

Alem disso podemos gerar um circuito pelo site **Digital JS** a partir do código SV, figura 2, obtendo a seguinte imagem, veja figura 5.

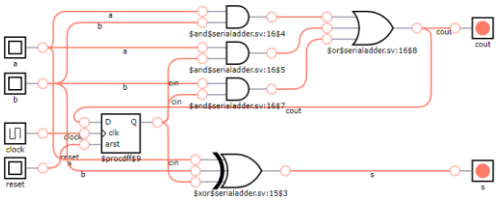

Figura 5. Circuito gerado apartir do código SV no site Digital JS.

Por fim, com os valores obtidos na testbench conseguimos desenvolver a seguinte tabela, veja tabela 1.

TABELA 1. RESULTADOS DA TESTBENCH

Bits de Entrada		Estado Atual	Estado Final	Bits de Saída	
A	**B**	**Cin**	**Cin⁺**	**S**	**Cout**
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	0	1	0
1	1	0	1	0	1
0	0	1	0	1	0
0	1	1	1	0	1
1	0	1	1	0	1
1	1	1	1	1	1

Formação da Tabela de Transição de Estados.

V. CONCLUSÃO

Portanto, por meio de planejamento e estudo, o projeto da implementação de um somador serial foi bem sucedido, visto que foi capaz de atender os requisitos iniciais e capaz de atribuir corretamente as saídas das somas bit a bit. O somador é capaz de resolver operação aritmética de soma utilizando poucos compononentes lógicos. Acresça-se ainda que, a importância de se estudar esse tipo de somador, apesar de ser mais lento em comparação ao somador paralelo, é que conseguimos criar um circuito de operação aritimética com poucas portas lógicas.

REFERÊNCIAS

[1] KUMAR, A. Anand. Fundamentals of digital circuits. PHI Learning Pvt. Ltd., 2016.

[2] Tocci, R. J., Widmer, N. S., and Moss, G. L. (2010). Sistemas digitais. Pearson Educacion. 22.v