



**Universidade Federal da Paraíba
Centro de Informática**

Concepção Estruturada de Circuitos Integrados

Sarah Andrade Toscano de Carvalho - Matrícula: 20170022975

João Pessoa, 2019.



Universidade Federal da Paraíba
Centro de Informática

Concepção Estrutural do ADDAC

Relatório da parte 1 do primeiro projeto da disciplina de Concepção Estrutural de Circuitos Integrados, ministrada pelo professor Antônio Carlos Cavalcante, no Centro de Informática da Universidade Federal da Paraíba.

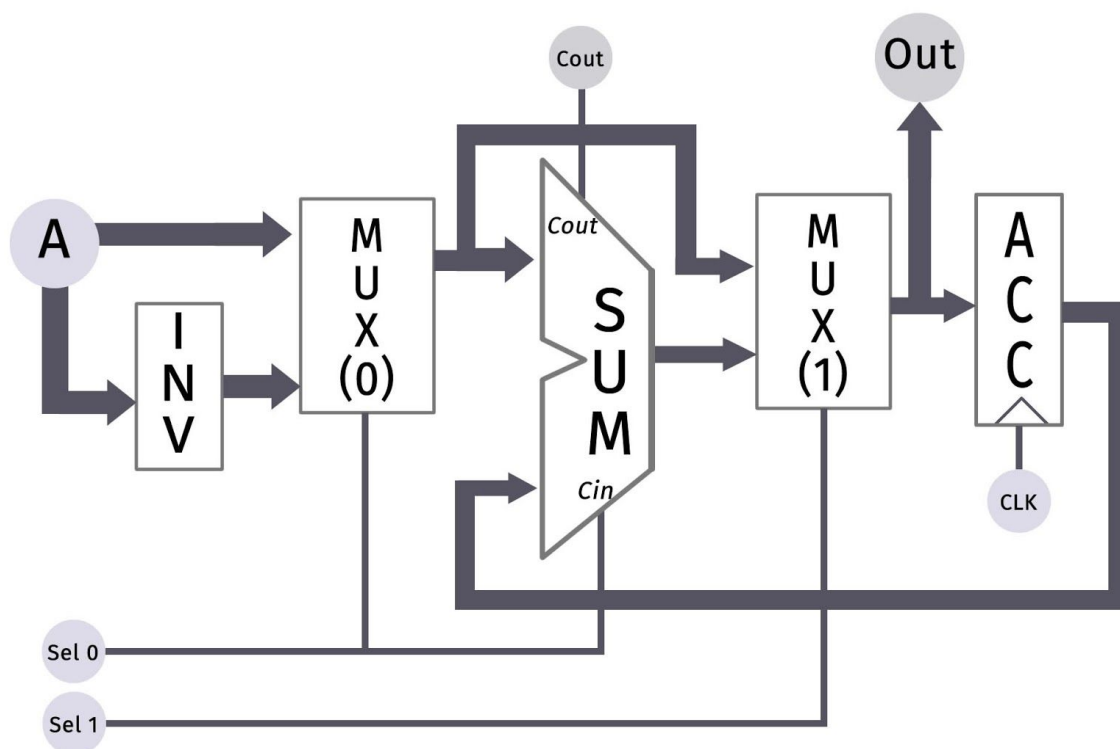
João Pessoa, 2019.

Sumário

1.1 Introdução	4
1.2 Implementação	5
1.2.1 Golden Model	5
1.2.1.1 Inversor.h	6
1.2.1.2 Mux.h	6
1.2.1.3 Somador.h	7
1.2.1.4 Carry Out.h	7
1.2.1.5 Acumulador.h	8
1.2.2 Simulação ModelSim - Arquivos .tv	9
1.2.2.1 Inversor.tv	9
1.2.2.2 Mux.tv	9
1.2.2.3 Somador.tv	10
1.2.2.4 Acumulador.tv	10
1.2.2.4 ADDAC.tv	11
1.2.3 Definição dos Blocos em System Verilog	12
1.2.3.1 Inv.sv	12
1.2.3.2 Mux.sv	12
1.2.3.3 Somador.sv	13
1.2.3.4 Acc.sv	13
1.2.3.4 ADDAC.sv	14
1.2.4 TestBench em System Verilog	14

1.1 Introdução

O Sistema lógico do ADDAC realiza 4 funções: cópia, soma, subtração e inversão, através da acumulação de 4 bits. Para isso são utilizados conexões entre 2 multiplexadores, 1 inversor, 1 somador completo e 1 acumulador, como ilustra a Figura. Neste relatório, pretende-se descrever a modelagem da primeira parte do projeto referente a concepção estrutural do ADDAC, que consiste em apenas um inversor.



Sel 0	Sel 1	Resposta
0	0	Copia A para ACC
0	1	Soma A com ACC e grava em ACC
1	0	Copia !A para ACC
1	1	Subtrai A de ACC e grava em ACC

Figura 1 - Estrutura lógica do ADDAC.

1.2 Implementação

O método utilizado para implementar a concepção estrutural do addac consiste em 7 passos:

- 1 - Elaboração do modelo de ouro (em inglês, *Golden Model*) em qualquer linguagem de programação;
- 2 - Descrição do modelo em System Verilog;
- 3 - Estruturação do *Test Bench*;
- 4 - Geração do arquivo *RTL*, através da compilação do arquivo .sv
- 5 - Solicitação de simulação do *RTL Level*;
- 6 - Solicitação de simulação do *Gate Level*;
- 7 - Alteração na temporização.

Neste relatório será modelado inicialmente um modelo de todos os blocos descritos na Figura 1 com sinais de entrada de 1 bit.

1.2.1 Golden Model

Para validar o funcionamento do circuito modelado, neste caso, todos os blocos lógicos que compõem o addac é gerado seu modelo de referência comportamental, através de um programa que testa todas as suas possíveis entradas e obtém um sinal de saída, para cada uma delas. Desse modo foi elaborado alguns programas em extensão .h na linguagem C que ilustram o comportamento individual desses componentes. Nas Figura 2-6 são ilustradas respectivamente as estruturas do inversor, multiplexador, somador e acc (acumulador) de 1 bit, respectivamente.

1.2.1.1 Inversor.h

```
1  #ifndef INV_H_INCLUDED
2  #define INV_H_INCLUDED
3
4  #include <stdbool.h>
5
6  int inversor(int a){
7
8      FILE *arquivo;
9      int aux=!a;
10
11      arquivo = fopen("inv.tv", "a");
12      //fprintf(arquivo, "//Inversor\n//Entrada_Saida\n");
13      fprintf(arquivo, "%d_%d\n", a, aux);
14      return aux;
15      fclose(arquivo);
16  }
```

Figura 2 - Modelo de Referência do Inversor - inv.h.

1.2.1.2 Mux.h

```
1  #ifndef MUX_H_INCLUDED
2  #define MUX_H_INCLUDED
3
4  int mux(int a, int b, int Sel){
5
6      FILE *arquivo;
7      arquivo = fopen("mux.tv", "a");
8
9      //fprintf(arquivo, "//MUX\n//Sel_Saida\n");
10     fprintf(arquivo, "%d_%d_%d_", Sel, a, b);
11
12     if(!Sel){
13         fprintf(arquivo, "%d\n", a);
14         return a;
15     }
16     else{
17         fprintf(arquivo, "%d\n", b);
18         return b;
19     }
20     fclose(arquivo);
21 }
22 #endif // MUX_H_INCLUDED
```

Figura 3 - Modelo de Referência do Mux - mux.h.

1.2.1.3 Somador.h

```
1  #ifndef SOMADOR_COMPLETO_H_INCLUDED
2  #define SOMADOR_COMPLETO_H_INCLUDED
3  #include "carry_out_function.h"
4  #include <string.h>
5
6  #define BIT 2
7
8  int somador_completo(int a, int b, int carry_in){
9
10     FILE *arquivo;
11     arquivo = fopen("somador.tv", "a");
12     //fprintf(arquivo, "//Somador\n//A_B_Cin_Cout_Soma\n");
13     fprintf(arquivo, "%d_%d_%d_", a,b,carry_in);
14
15     int soma = a+b+carry_in;
16     char soma_bin[BIT];
17     int carry_out = carry_out_function(soma);
18     itoa(soma, soma_bin, 2); //transforma a soma p binario
19
20     if((a==0 & b==0))
21     {
22         fprintf(arquivo, "0_%d\n", carry_in);
23     }
24     else if (carry_in==0 & ((a==0 & b==1)|(a==1 & b==0)))
25     {
26         fprintf(arquivo, "0_1\n");
27     }
28     else{
29         fprintf(arquivo, "%c_%c\n", soma_bin[BIT-2], soma_bin[BIT-1]);
30     }
31     fclose(arquivo);
32     soma = 0;
33     return a+b+carry_in; //corrigir p main
34 }
35 #endif // SOMADOR_COMPLETO_H_INCLUDED
```

Figura 4 - Modelo de Referência do somador - somador_completo.h.

1.2.1.4 Carry Out.h

```
1  #ifndef CARRY_OUT_FUNCTION_H_INCLUDED
2  #define CARRY_OUT_FUNCTION_H_INCLUDED
3
4  int carry_out_function(int soma){
5      if(soma>1){
6          return 1;
7      }
8      else return 0;
9  }
10 #endif // CARRY_OUT_FUNCTION_H_INCLUDED
```

Figura 5 - Estrutura Auxiliar do Modelo de Referência do carry out - carry_out.h.

1.2.1.5 Acumulador.h

```
1  #ifndef ACC_H_INCLUDED
2  #define ACC_H_INCLUDED
3
4  int acc(int clk_a, int clk, int s, int acumulado){
5      FILE *arquivo;
6      arquivo = fopen("acc.tv", "a");
7      fprintf(arquivo, "%d_%d_%d_", clk_a, clk, s);
8
9      //fprintf(arquivo, "//acc\n//clk_a clk_saida\n");
10     if(clk_a==1 & clk==0){//borda de descida
11         fprintf(arquivo, "a%d\n", s);
12         acumulado = s; //atualiza
13         return s;
14     }
15     else{
16         fprintf(arquivo, "%d\n", acumulado);
17         return acumulado; //valor anterior
18     }
19     fclose(arquivo);
20
21 }
22 #endif // ACC_H_INCLUDED
```

Figura 6 -Modelo de Referência do acumulador - acc.h.

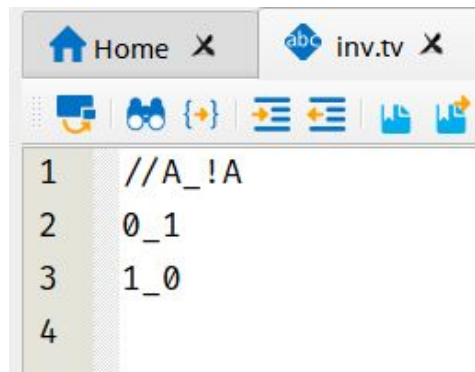
No escopo desses programas .h há o processo de escrita de um arquivo salvo na extensão “.tv” que futuramente irá ser utilizada como o arquivo de verificação para o comportamento do circuito implementado em verilog.

Para obter todas as possibilidades de saída para cada bloco na função principal deste programa (main.c) foi realizada a variação dos parâmetros de todas as funções implementadas em .h. Nas Figuras 7-10 são ilustrados os vetores de testes escritos nos arquivos .tv.

1.2.2 Simulação ModelSim - Arquivos .tv

1.2.2.1 Inversor.tv

O modelo de referência do arquivo inversor.tv foi escrito da seguinte forma: dado o bit de entrada, é adicionado um underscore para separá-lo do bit de saída, e em seguida o bit é invertido de acordo com a função descrita no arquivo inv.h.

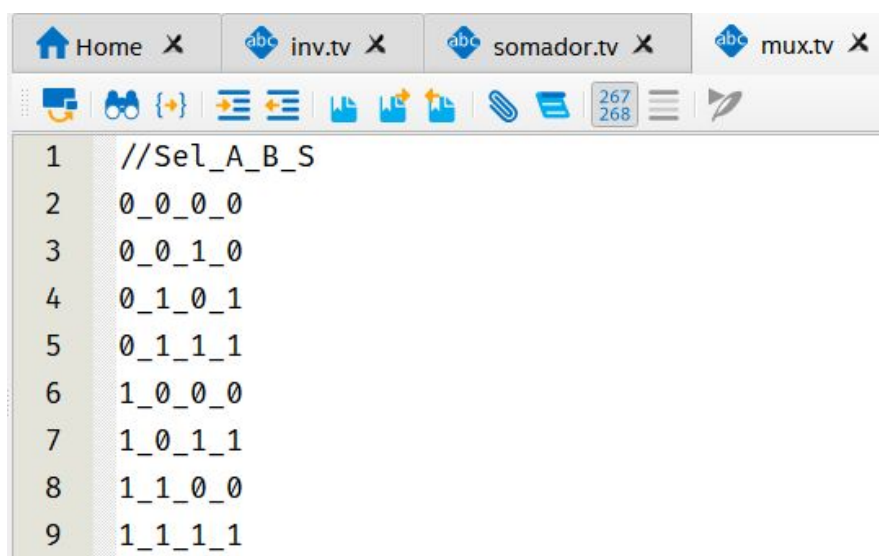


```
1 //A_!A
2 0_1
3 1_0
4
```

Figura 7 - Arquivo.tv do Inversor - inv.tv.

1.2.2.2 Mux.tv

O modelo de referência do arquivo mux.tv foi escrito com o bit equivalente a chave de seleção seguido das duas variáveis de entrada do mux e finalmente a variável de saída do bloco lógico. Dessa forma, sempre que o primeiro bit for equivalente a 0 a saída (o último bit) tem que ser igual ao bit equivalente a entrada A, ou seja, o segundo bit. Assim, a saída é igual ao segundo bit. Porém, se a chave de seleção estiver em 1 a saída é igual ao B.

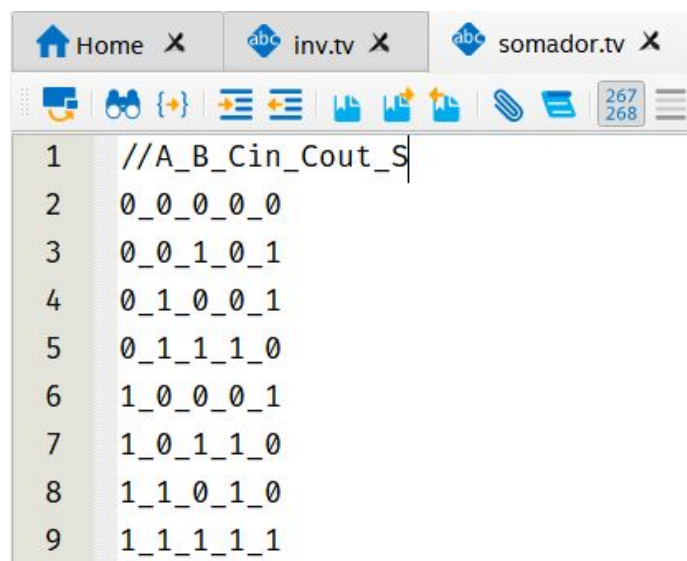


```
1 //Sel_A_B_S
2 0_0_0_0
3 0_0_1_0
4 0_1_0_1
5 0_1_1_1
6 1_0_0_0
7 1_0_1_1
8 1_1_0_0
9 1_1_1_1
```

Figura 8 - Arquivo.tv do mux - mux.tv.

1.2.2.3 Somador.tv

O modelo de referência do arquivo somador.tv foi escrito da seguinte forma: dado o bit referente a chave de seleção, em seguida são adicionadas as entradas A, B e o Carry in seguidas de underscore e por fim, é ilustrado o bit de estouro ou carry out e a saída equivalente ao resultado da operação realizada pelo somador. Para o último caso de análise, em que todas as entradas estão em nível lógico alto, ambas variáveis da saída também são 1.



1	//A_B_Cin_Cout_S
2	0_0_0_0_0
3	0_0_1_0_1
4	0_1_0_0_1
5	0_1_1_1_0
6	1_0_0_0_1
7	1_0_1_1_0
8	1_1_0_1_0
9	1_1_1_1_1

Figura 9 - Arquivo.tv do Somador - somador.tv

1.2.2.4 Acumulador.tv

O acumulador tem por função receber o sinal de entrada e só atualizar sua saída quando o clock estiver na borda de descida. Para este golden model foram utilizadas duas variáveis auxiliares que permitem controlar a borda do clock. A borda de descida é atingida quando o clock anterior está em 1 e o clk (atual) está em 0. Desse modo, apenas quando o acumulador recebe o clk_a como 0 e clk como 1 ele possui a sua saída igual a entrada. Na Figura 10, os parâmetros indicados estão na sequência descrita no comentário. A variável ACC se refere ao valor que está na entrada do acumulador porém a saída do bloco lógico só será igual a ele nas linhas 6 e 7, visto que o clock está na borda de descida.

```

1 //Clk_a Clk Acc Saida
2 0_0_0_0
3 0_0_1_0
4 0_1_0_0
5 0_1_1_0
6 1_0_0_0
7 1_0_1_1
8 1_1_0_0
9 1_1_1_0
10
11 //borda de descida 1_0

```

Figura 10 - Arquivo.tv do acumulador- acc.tv.

1.2.2.4 ADDAC.tv

O modelo de referência foi elaborado da seguinte forma: Sel_0, Sel_1, entrada, CLK, acumulado e saída.

```

1 0_0_0_0_0_0_0
2 0_0_1_0_1_1
3 0_0_0_1_0_1
4 0_0_1_1_1_1
5 0_1_0_0_1_1
6 0_1_1_0_0_0
7 0_1_0_1_0_0
8 0_1_1_1_1_0
9 1_0_0_0_1_1
10 1_0_1_0_0_0
11 1_0_0_1_1_0
12 1_0_1_1_0_0
13 1_1_0_0_0_0
14 1_1_1_0_1_1
15 1_1_0_1_1_1
16 1_1_1_1_0_1
17

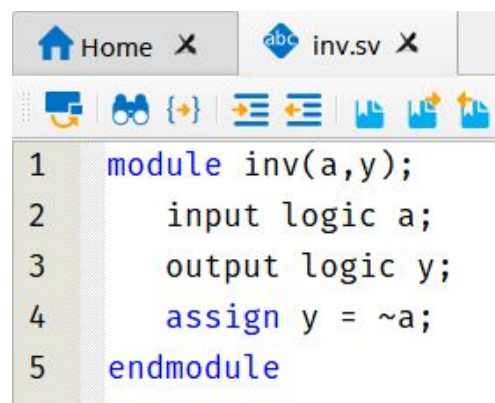
```

Figura 10 - Arquivo.tv do addac- addac.tv.

1.2.3 Definição dos Blocos em System Verilog

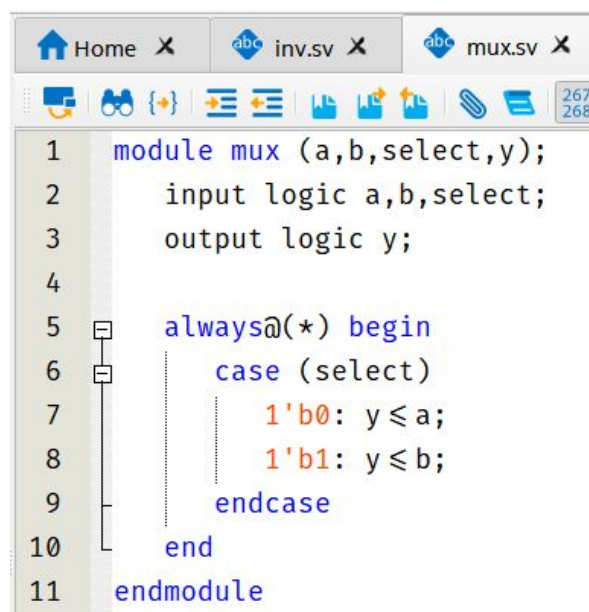
Após elaborar a síntese dos blocos lógicos na linguagem C, será definido os seus respectivos módulos de modo comportamental através de programas descritos em system verilog. Deste modo, eles terão o mesmo propósito e algoritmo lógico de funcionamento do aplicado no desenvolvimento do modelo de referência.

1.2.3.1 Inv.sv



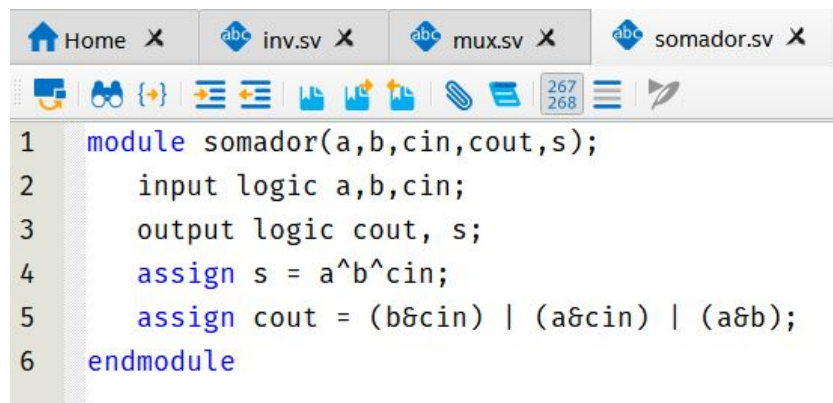
```
1 module inv(a,y);
2     input logic a;
3     output logic y;
4     assign y = ~a;
5 endmodule
```

1.2.3.2 Mux.sv



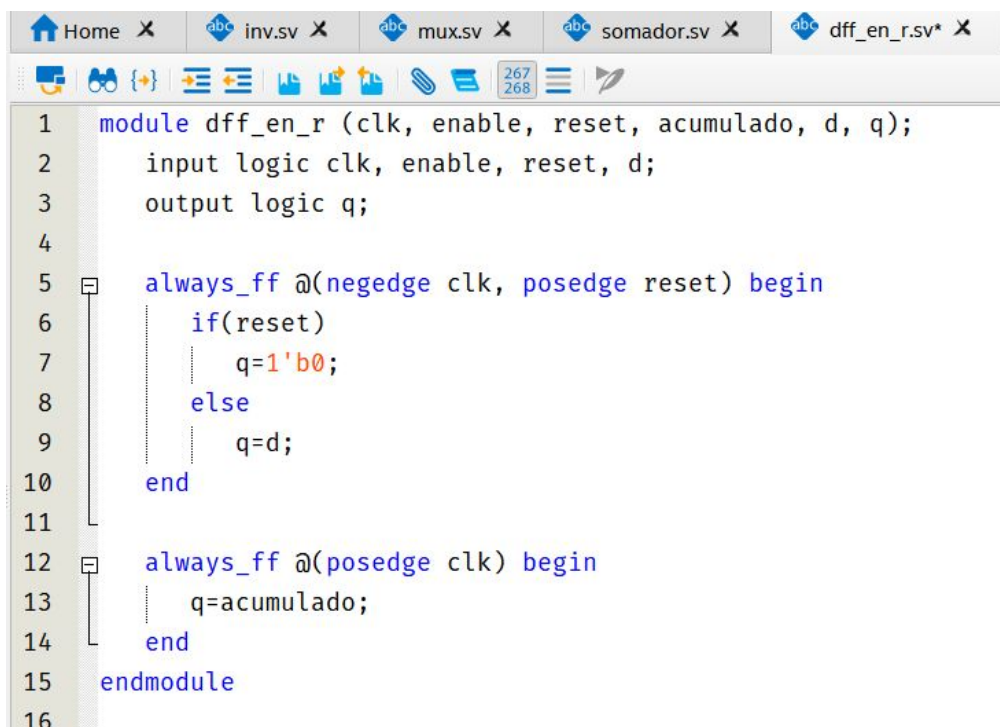
```
1 module mux (a,b,select,y);
2     input logic a,b,select;
3     output logic y;
4
5     always@(*) begin
6         case (select)
7             1'b0: y ≤ a;
8             1'b1: y ≤ b;
9         endcase
10    end
11 endmodule
```

1.2.3.3 Somador.sv



```
1 module somador(a,b,cin,cout,s);
2     input logic a,b,cin;
3     output logic cout, s;
4     assign s = a^b^cin;
5     assign cout = (b&cin) | (a&cin) | (a&b);
6 endmodule
```

1.2.3.4 Acc.sv



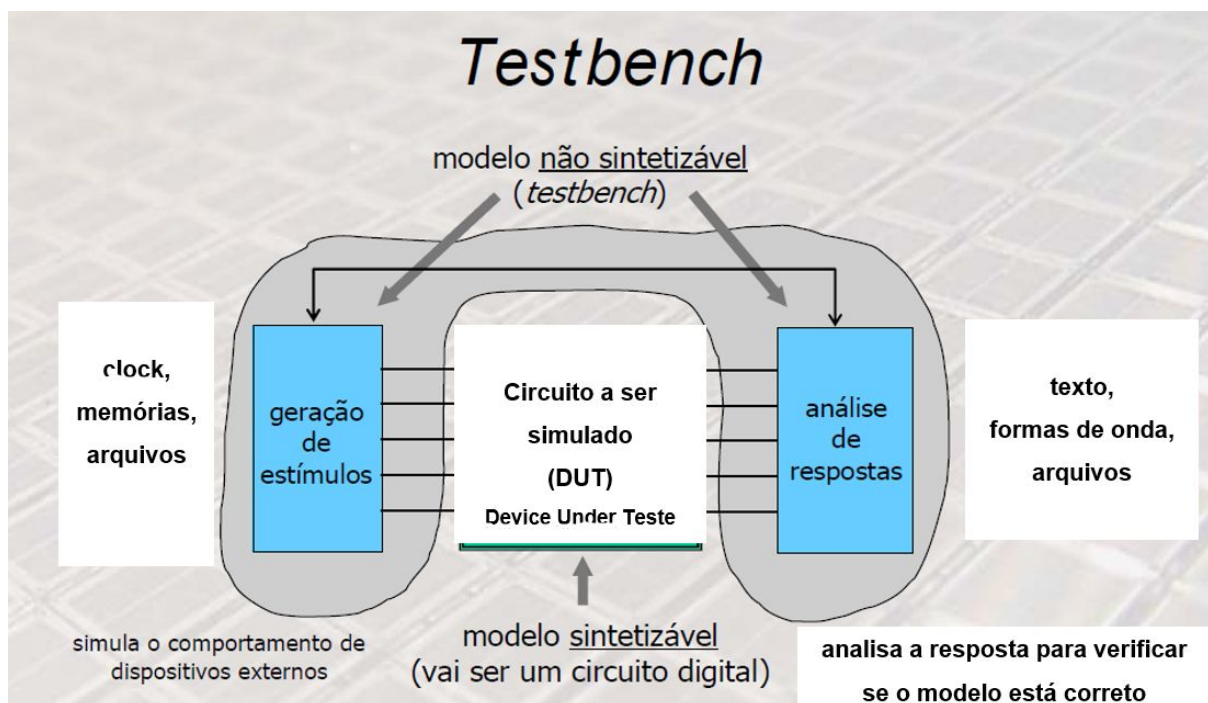
```
1 module dff_en_r (clk, enable, reset, acumulado, d, q);
2     input logic clk, enable, reset, d;
3     output logic q;
4
5     always_ff @(negedge clk, posedge reset) begin
6         if(reset)
7             q=1'b0;
8         else
9             q=d;
10    end
11
12    always_ff @(posedge clk) begin
13        q=acumulado;
14    end
15 endmodule
16
```

1.2.3.4 ADDAC.sv

```
1 module addac(a,sel_0, sel_1, clk, cout, s);
2
3   input logic a, sel_0, sel_1, clk;
4   output logic cout, s;
5
6   logic mux_0_out, mux_1_out, somador_out;
7
8   mux mux_0_bloco(a, inversor(a), sel_0, mux_0_out);
9   somador_out somador_bloco(a, s, sel_0, cout, somador_out);
10  mux mux_1_bloco(mux_0_out, somador_out, sel_1, mux_1_out);
11
12  s = dff_en_r acc_bloco(clk, 1'b1, 1'b0, mux_1_out,s);
13
14  endmodule
```

1.2.4 TestBench em System Verilog

A lógica de execução de um testbench está ilustrada na Figura abaixo:



Dados os estímulos de entrada do circuito, será elaborado uma instanciação dos módulos definidos em 1.2.3 denominado DUV (design under verification). Desse modo, as entradas são inseridas nesse circuito, porém o sinal de resposta gerado por este sistema é comparado com o sinal obtido no modelo de referência, elaborado na linguagem C em 1.2.2.

Caso esses dados apresentem alguma inconsistência, o módulo do testbench emitirá uma mensagem de sinalização apontando a variável errada e seu bit de erro. Desse modo, conseguimos localizá-lo e consertá-lo.

Abaixo, é ilustrado um arquivo de testbench elaborado para um inversor de um bit, os demais circuitos apresentam a mesma lógica de implementação.

```
`timescale 1ns/100ps
module inv_tb ();
    logic a;
    logic clk, reset;
    logic y, y_esperado;
    logic [2:0]vector[1:0];
    int count, erro, aux_erro;

    inv DUV(a, y);

    initial begin
        $display("Iniciando Testbench");
        $display("| A | S |");
        $display("-----");
        $readmemb("../Simulation/ModelSim/inv.tv",vector);
        count=0; erro=0;
        reset=1'b1; #7; reset=1'b0;
    end

    always begin
        clk=1; #6;
        clk=0; #10;
    end

    always @(posedge clk) begin
        if(~reset) begin
            {a,y_esperado} = vector[count]; //atualiza os valores na borda de subida
        end
    end
end
```

```

1 always@(negedge clk) begin
1     if(~reset) begin
        aux_erro=erro;
        assert (y == y_esperado); //verifica se o resultado bateu com o esperado
    end else begin
        //Caso o assert dê erro... São printadas as mensagens de Erro
        $display("Linha [%d] -- Saída Esperada: %b -- Saída: %b",count+1, y_esperado, y);
        erro = erro + 1; //Incrementa contador de erros a cada bit errado encontrado
    end
    if(aux_erro == erro)
        $display("| %b | %b | OK", a, y);
    else
        $display("| %b | %b | ERRO", a, y);
        count = count+1;

    if(count == $size(vector)) begin //Finaliza os casos de testes
        $display("Testes Efetuados = %0d", count);
        $display("Erros Encontrados = %0d", erro);
        #10
        $stop;
    end
end
endmodule

```