

Centro de Informática

Universidade Federal da Paraíba

**Disciplina:** Concepção Estruturada de Circuitos Integrados

**Professor:** Antônio Carlos C.

ATIVIDADE 2

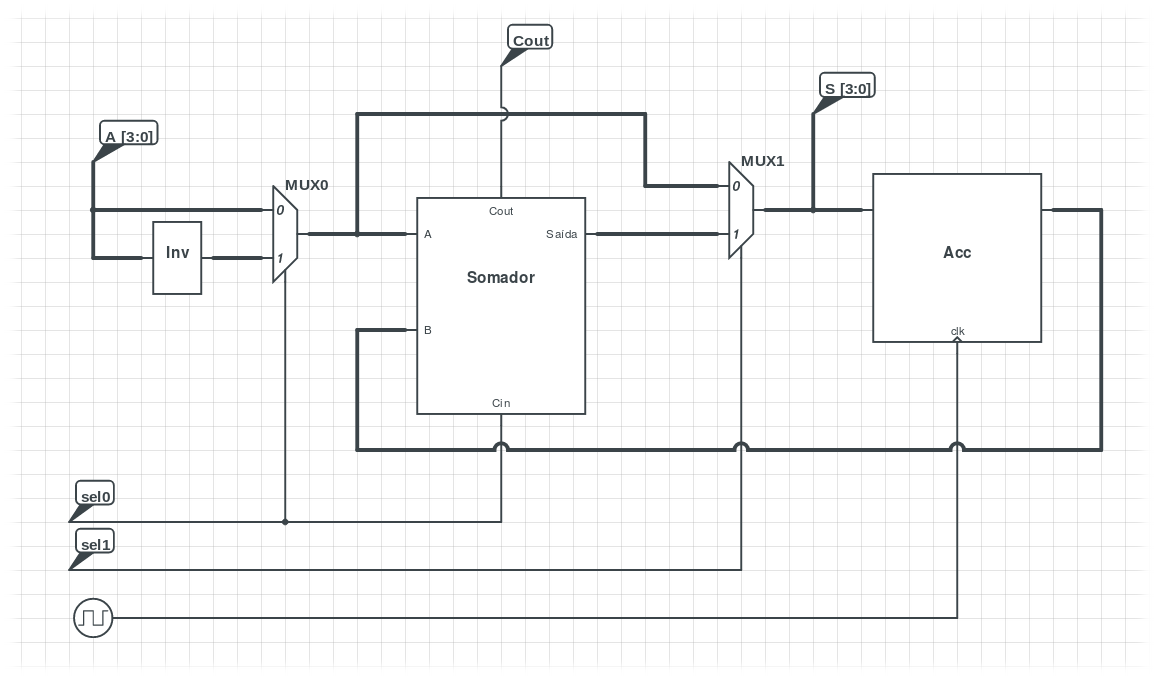
Vanessa Bonifácio

João Pessoa, 08 de Dezembro de 2019

**Introdução**

Esse projeto foi criado para a disciplina de Concepção Estruturada de Circuitos Integrados, ministrada pelo Prof. Antonio Carlos Cavalcanti, no período 2019.2, UFPB. A disciplina teve como objetivo gerar um circuito somador de 4 bits com acumulador totalmente funcional, seguindo a filosofia Top Down, partindo do Modelo de Referência de Ouro até o circuito completo.

**Diagrama do Somador de 4 Bits com Acumulador**



O circuito possui apenas três entradas, que são a entrada de valores de 4 bits A e os os seletores de modo de operação sel0 e sel1, com 1 bit cada, e duas saídas, que são a saída de 4 bits da operação atual S e o indicador de overflow Cout, que é ativado toda vez que uma operação resulta em um número maior que 4 bits.

|  |  |  |  |
| --- | --- | --- | --- |
| **sel0** | **sel1** | **Resposta** | **Operação** |
| 0 | 0 | Copia A para ACC | ACC <= A |
| 0 | 1 | Soma A com ACC e grava em ACC | ACC <= ACC + A |
| 1 | 0 | Copia Inv(A) para ACC | ACC <= Inv(A) |
| 1 | 1 | Subtrai A de ACC e grava em ACC | ACC <= - A |

**Modelos de Referência de Ouro (Golden Model)**

O Modelo de Referência de Ouro, do inglês Golden Reference Model, trata-se de um modelo de referência em alto nível de um dado circuito capaz de descrever seu comportamento, isto é, suas saídas, em função das suas entradas. Esse modelo é usado para produzir um vetor de casos de teste que exaustivamente compara se o resultado produzido pelo circuito está de acordo com o vetor de testes gerado.

**Somador 1-bit**

* **Golden Model**

Código em python que gera um arquivo de saída somador1bit.tv.

|  |
| --- |
| def somador1bit(a, b, cin):  saida = 0  cout = 0    if cin == 0:  if a + b == 0:  saida = 0  cout = 0  if a + b == 1:  saida = 1  cout = 0  if a + b == 2:  saida = 0  cout = 1  else:  if a + b == 0:  saida = 1  cout = 0  if a + b == 1:  saida = 0  cout = 1  if a + b == 2:  saida = 1  cout = 1  return saida, cout  def escrever(a, b, cin, cout, saida):  file = open("somador1bit.tv", "a")    file.write(str(a))  file.write("\_")    file.write(str(b))  file.write("\_")    file.write(str(cin))  file.write("\_")    file.write(str(saida))  file.write("\_")  file.write(str(cout))  file.write('\n')  file.close()  s0,c0 = somador1bit(0,0,0)  s1,c1 = somador1bit(0,0,1)  s2,c2 = somador1bit(0,1,0)  s3,c3 = somador1bit(0,1,1)  s4,c4 = somador1bit(1,0,0)  s5,c5 = somador1bit(1,0,1)  s6,c6 = somador1bit(1,1,0)  s7,c7 = somador1bit(1,1,1)  escrever(0,0,0,c0,s0)  escrever(0,0,1,c1,s1)  escrever(0,1,0,c2,s2)  escrever(0,1,1,c3,s3)  escrever(1,0,0,c4,s4)  escrever(1,0,1,c5,s5)  escrever(1,1,0,c6,s6)  escrever(1,1,1,c7,s7) |

* **TestVector**

|  |
| --- |
| 0\_0\_0\_0\_0  0\_0\_1\_1\_0  0\_1\_0\_1\_0  0\_1\_1\_0\_1  1\_0\_0\_1\_0  1\_0\_1\_0\_1  1\_1\_0\_0\_1  1\_1\_1\_1\_1 |

* **Descrição do Somador 1-bit**

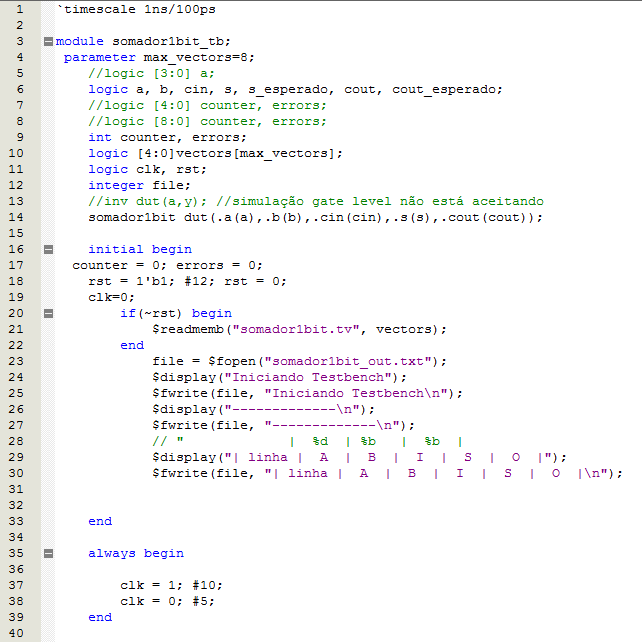
O módulo somador1bit possui uma entrada e saída de 1bit e a linguagem

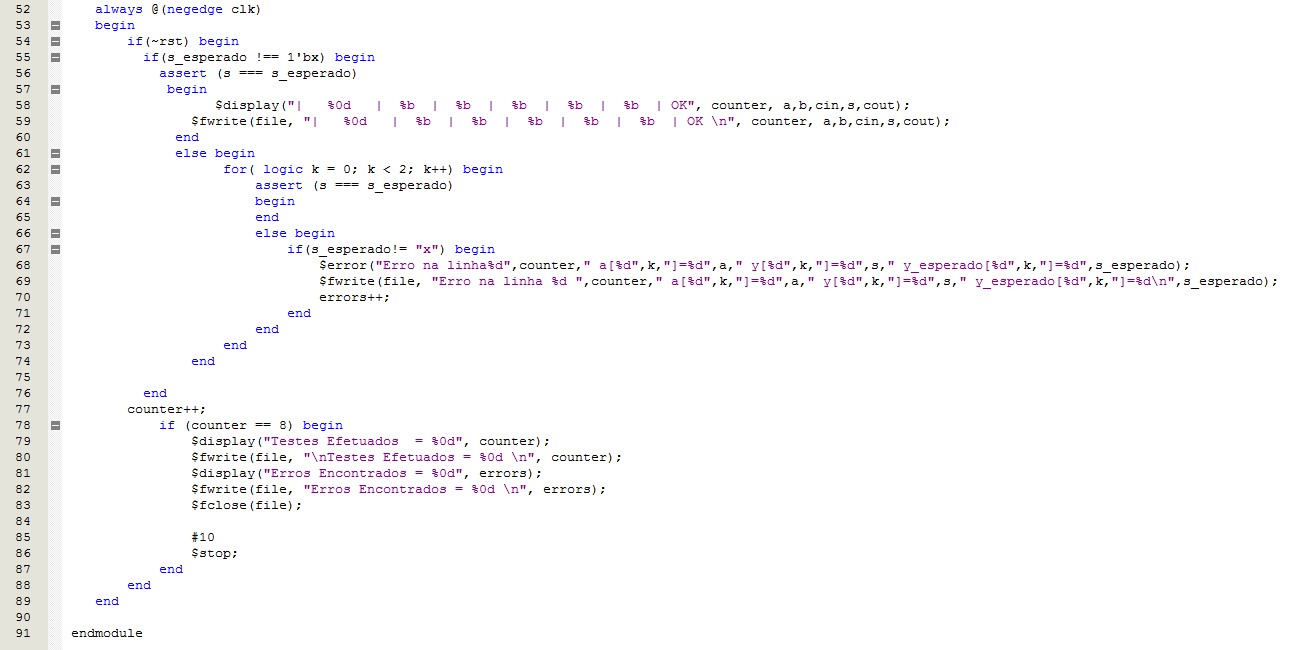
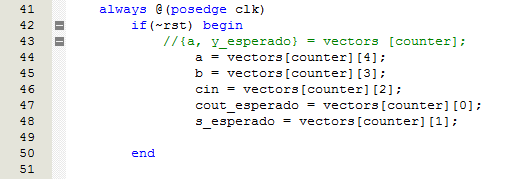
utilizada foi a System Verilog.

|  |
| --- |
| module somador1bit(input logic a, b, cin, output logic s,cout);  logic p, g;  assign p = a ^ b;  assign g = a & b;  assign s = p ^ cin;  assign cout = g | (p & cin);  endmodule |

* **Testbench**

Definidos golden model e somador 1 bit, é necessário criação de arquivo de validação do circuito, também em system verilog. Utilizando as entradas que estão nos vetores de testes do arquivo somador1bit.tv, aplicando ao módulo somador, gerando uma saída. Saída esta que é comparada com a saída do vetor de teste para verificação se há erros.





**Somador 4-bits**

* **Golden Model**

Código em python que gera um arquivo de saída somador4bit.tv.

|  |
| --- |
| def somador1bit(a, b, cin):  saida = 0  cout = 0    if cin == 0:  if a + b == 0:  saida = 0  cout = 0  if a + b == 1:  saida = 1  cout = 0  if a + b == 2:  saida = 0  cout = 1  else:  if a + b == 0:  saida = 1  cout = 0  if a + b == 1:  saida = 0  cout = 1  if a + b == 2:  saida = 1  cout = 1  return saida, cout  def somador4bit(a,b,cin):  saida = [0]\*len(a)  cout = [0]\*len(a)  s = 0  c = cin  tamanho = len(a) - 1    for i in range(len(a)):  s,c = somador1bit(a[tamanho - i],b[tamanho - i],c)  saida[tamanho - i] = s  cout[tamanho - i] = c  return saida, cout  def escrever(a, b, cin, saida, cout):  file = open("somador4bit.tv", "a")  for i in a:  file.write(str(i))  file.write("\_")  for j in b:  file.write(str(j))  file.write("\_")  file.write(str(cin))  file.write("\_")  for k in saida:  file.write(str(k))  file.write("\_")  for l in cout:  file.write(str(l))  file.write('\n')  file.close()  def decimalconv(a):  dec = 0  tamanho = len(a)- 1  for i in range(len(a)):  dec = (dec + ((2\*\*i)\*a[tamanho - i]))  return dec  a = [0,0,0,0]  b = [1,1,1,1]  c = [0,0,0,1]  d = [1,1,0,1]  e = [0,0,1,0]  f = [1,0,0,0]  s0,c0 = somador4bit(a,b,0) #1111 - 0000  s1,c1 = somador4bit(c,d,0) #1110 - 0001  s2,c2 = somador4bit(e,f,0) #1010 - 0000  s3,c3 = somador4bit(a,c,0) #0001 - 0000  s4,c4 = somador4bit(a,d,0) #1101 - 0000  s5,c5 = somador4bit(a,f,0) #1000 - 0000  s6,c6 = somador4bit(b,c,0) #0000 - 1111  s7,c7 = somador4bit(d,e,0) #1111 - 0000  escrever(a,b,0,s0,c0)  escrever(c,d,0,s1,c1)  escrever(e,f,0,s2,c2)  escrever(a,c,0,s3,c3)  escrever(a,d,0,s4,c4)  escrever(a,f,0,s5,c5)  escrever(b,c,0,s6,c6)  escrever(d,e,0,s7,c7) |

* **TestVector**

|  |
| --- |
| 0000\_1111\_0\_1111\_0000  0001\_1101\_0\_1110\_0001  0010\_1000\_0\_1010\_0000  0000\_0001\_0\_0001\_0000  0000\_1101\_0\_1101\_0000  0000\_1000\_0\_1000\_0000  1111\_0001\_0\_0000\_1111  1101\_0010\_0\_1111\_0000 |

* **Descrição do Somador 4-bits**

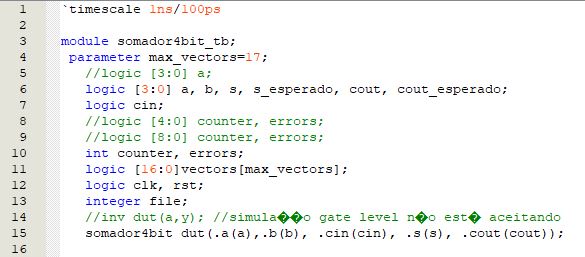
O módulo somador4bit possui uma entrada e saída de 4bits e a linguagem

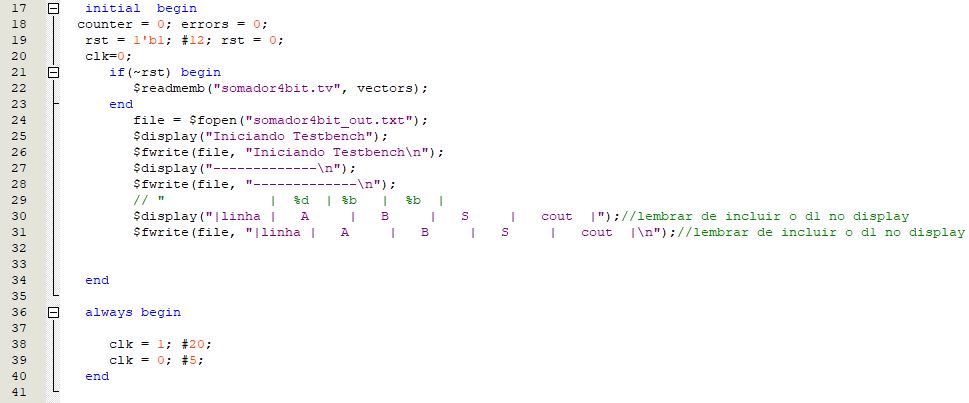
utilizada foi a System Verilog.

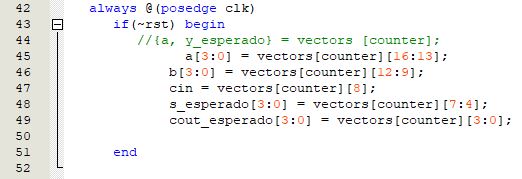
|  |
| --- |
| module somador4bit(input logic [3:0] a, b, input logic cin, output logic [3:0] s, cout);  somador1bit s0 (a[0],b[0],cin,s[0],cout[0]);  somador1bit s1 (a[1],b[1],cout[0],s[1],cout[1]);  somador1bit s2 (a[2],b[2],cout[1],s[2],cout[2]);  somador1bit s3 (a[3],b[3],cout[2],s[3],cout[3]);  endmodule |

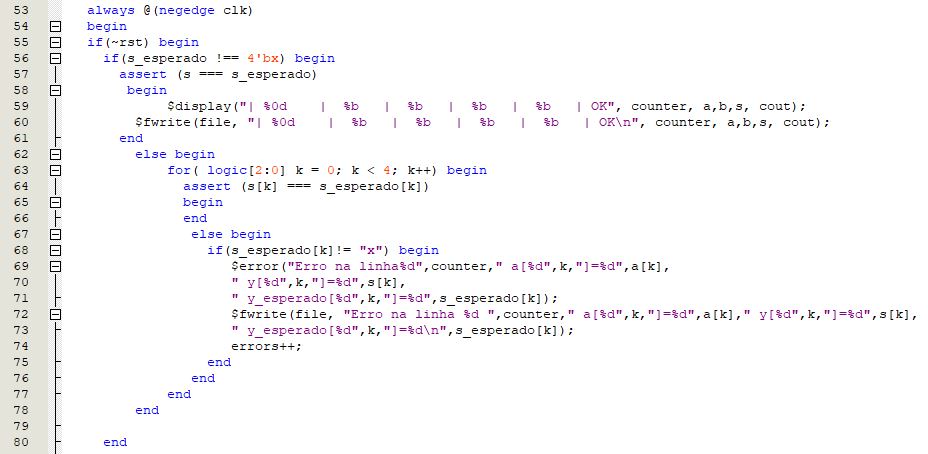
* **Testbench**

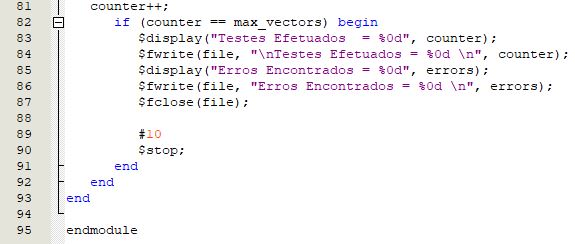
Definidos golden model e somador 4 bits, é necessário criação de arquivo de validação do circuito, também em system verilog. Utilizando as entradas que estão nos vetores de testes do arquivo somador4bit.tv, aplicando ao módulo somador4bit, gerando uma saída. Saída esta que é comparada com a saída do vetor de teste para verificação se há erros.











**Acumulador**

**Flopenr 1bit**

* **Golden Model**

Código em python que gera um arquivo de saída flopenr.tv.

|  |
| --- |
| def flopenr(d,en,ck,rt):  global flag,q    if rt == 1:  q = 0  return q  if ck == 0 and en == 0:  flag = 0  return q  if ck == 0 and en == 1:  flag = 1  return q  if ck == 1 and flag == 1:  q = d  return q  if ck == 1 and flag == 0:  return q    def write(d,en,ck,rt,q):    file = open("flopenr.tv","a")  file.write(str(d))  file.write("\_")  file.write(str(en))  file.write("\_")  file.write(str(ck))  file.write("\_")  file.write(str(rt))  file.write("\_")  file.write(str(q))  file.write('\n')  file.close()    def loop(d,en,ck,rt):  global q    for i in range(len(d)):  q = flopenr(d[i],en[i],ck[i],rt[i])  write(d[i],en[i],ck[i],rt[i],q)  ck = [0,1,0,1,0,1,0,1,0,1,0,1]  d = [0,0,0,0,1,1,1,1,0,0,1,1]  en = [0,0,1,1,0,0,1,1,0,0,1,1]  rt = [1,1,1,1,1,1,1,1,0,0,0,0]  q = 'x'  flag = 0  loop(d,en,ck,rt) |

* **TestVector**

|  |
| --- |
| 0\_0\_0\_1\_0  0\_0\_1\_1\_0  0\_1\_0\_1\_0  0\_1\_1\_1\_0  1\_0\_0\_1\_0  1\_0\_1\_1\_0  1\_1\_0\_1\_0  1\_1\_1\_1\_0  0\_0\_0\_0\_0  0\_0\_1\_0\_0  1\_1\_0\_0\_0  1\_1\_1\_0\_1 |

* **Descrição do Flopenr 1-bit**

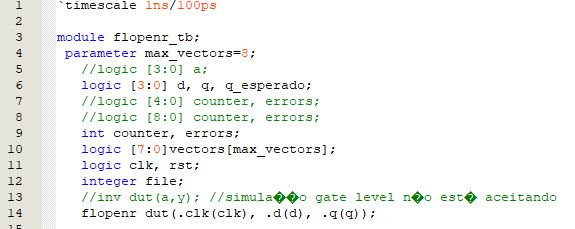
O módulo flopenr 1bit possui entrada, clock enable de controle e saída de 1bit,

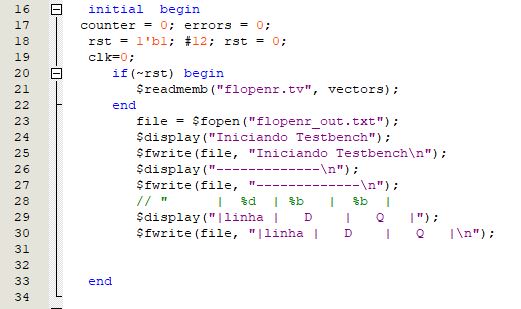
onde a linguagem utilizada foi System Verilog.

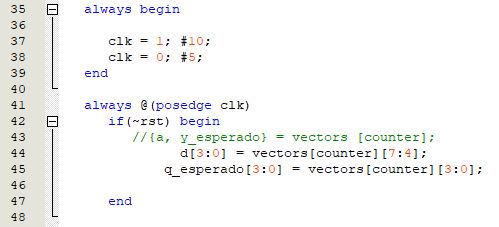
|  |
| --- |
| module registrador(input logic clk, input logic [3:0] d, output logic [3:0] q);  always\_ff @(posedge clk)  q <= d;  endmodule |

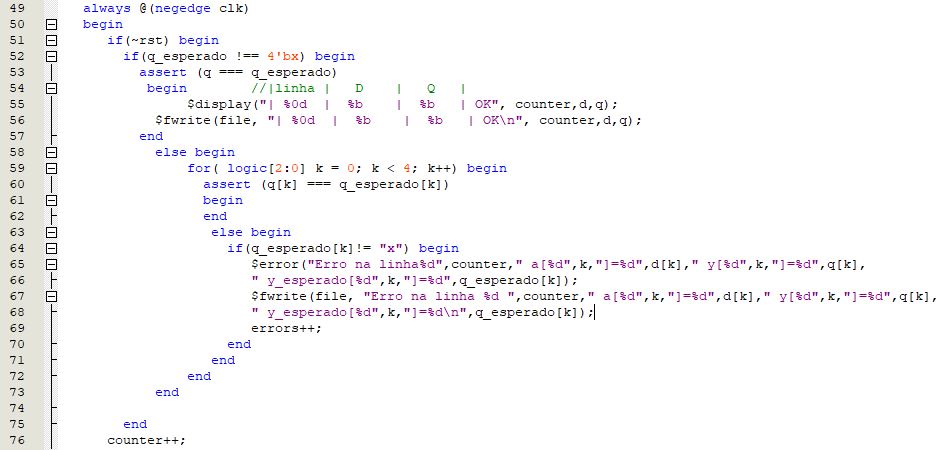
* **Testbench**

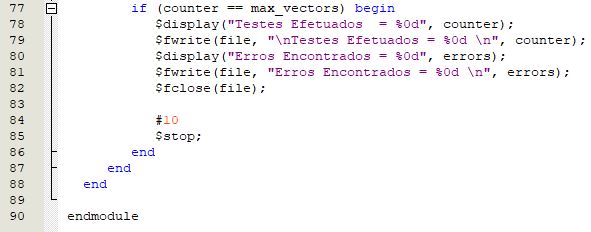
Definidos golden model e flopenr 1-bit, é necessário criação de arquivo de validação do circuito, também em system verilog. Utilizando as entradas que estão nos vetores de testes do arquivo flopenr.tv, aplicando ao módulo flopenr, gerando uma saída. Saída esta que é comparada com a saída do vetor de teste para verificação se há erros.











**Flopenr 4bits**

* **Golden Model**

Código em python que gera um arquivo de saída flopenr-4bit.tv.

|  |
| --- |
| def escrever(d,q):  file = open("flopenr-4bit.tv","a")  for i in d:  file.write(str(i))  file.write("\_")  for j in q:  file.write(str(j))  file.write('\n')  file.close()  def acumulador(clk,d):  q = ["x"]\*len(d)  for i in range(len(clk)):  if clk[i] > clk[i-1]:  escrever(d,q)  q = d  else:  q = q    clock = [0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1]  d = [0,0,1,0]  acumulador(clock,d) |

* **TestVector**

|  |
| --- |
| 0010\_xxxx  0010\_0010 |

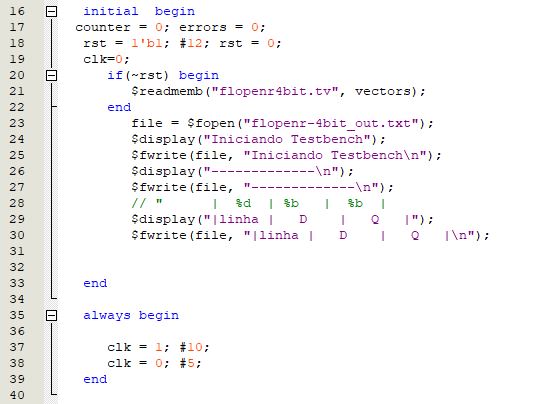
* **Descrição do Flopenr 4-bits**

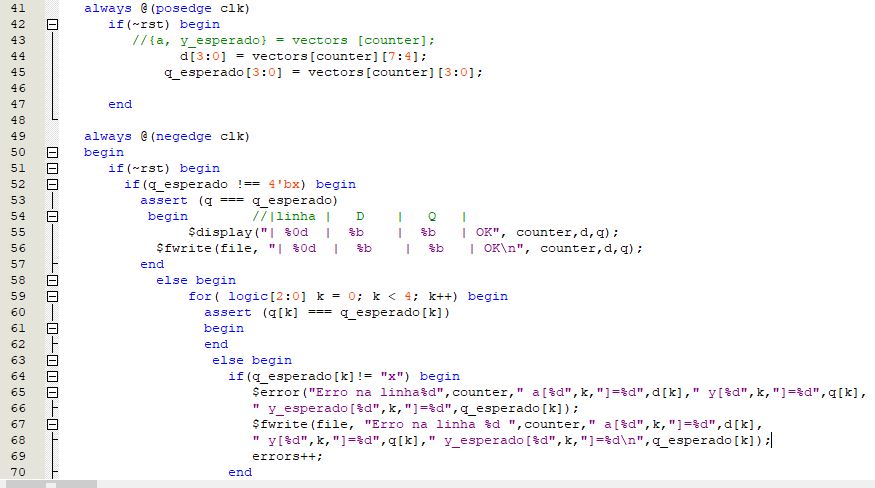
O módulo flopenr-4bit possui entrada, clock enable de controle de 1bit e saída de 4bits, a linguagem utilizada foi System Verilog.

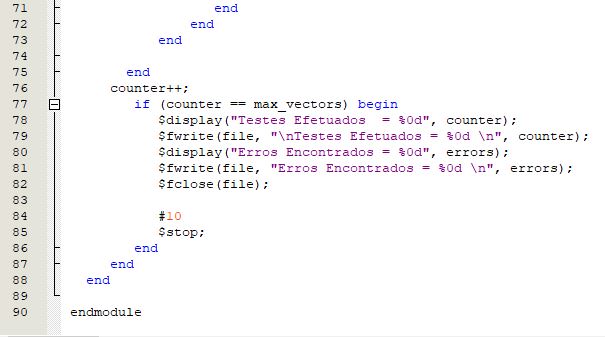
|  |
| --- |
| module flopenr-4bit(input logic clk, input logic [3:0] d, output logic [3:0] q);  flopenr flopenr\_1(clk,reset,en,d[0],q[0]);  flopenr flopenr\_2(clk,reset,en,d[1],q[1]);  flopenr flopenr\_3(clk,reset,en,d[2],q[2]);  flopenr flopenr\_4(clk,reset,en,d[3],q[3]);  endmodule |

* **Testbench**

Definidos golden model e flopenr-4bits, é necessário criação de arquivo de validação do circuito, também em system verilog. Utilizando as entradas que estão nos vetores de testes do arquivo flopenr-4bit.tv, aplicando ao módulo flopenr 4bits, gerando uma saída. Saída esta que é comparada com a saída do vetor de teste para verificação se há erros.







**Addac**

* **Golden Model**

Código em python que gera um arquivo de saída addac.tv

|  |
| --- |
| def somador1bit(a, b, cin):  saida = 0  cout = 0    if cin == 0:  if a + b == 0:  saida = 0  cout = 0  if a + b == 1:  saida = 1  cout = 0  if a + b == 2:  saida = 0  cout = 1  else:  if a + b == 0:  saida = 1  cout = 0  if a + b == 1:  saida = 0  cout = 1  if a + b == 2:  saida = 1  cout = 1  return saida, cout  def somador4bit(a,b,cin):  saida = [0]\*len(a)  cout = [0]\*len(a)  s = 0  c = cin  tamanho = len(a) - 1    for i in range(len(a)):  s,c = somador1bit(a[tamanho - i],b[tamanho - i],c)  saida[tamanho - i] = s  cout[tamanho - i] = c  return saida,cout[0]  def inversor(a):  saida = []  for i in a:  if i == 1:  saida.append(0)  elif i == 0:  saida.append(1)  return saida  def addac(a,sel0,sel1):  c = 0  s = 0  global acc    if sel0 == 0 and sel1 == 0:  acc = a  if sel0 == 1 and sel1 == 0:  acc = inversor(a)  if sel0 == 0 and sel1 == 1:  s,c = somador4bit(a,acc,sel0)  acc = s  if sel0 == 1 and sel1 == 1:  ainv = inversor(a)  s,c = somador4bit(ainv,acc,sel0)  acc = s  return c,acc  def escrever(a,sel0,sel1,acc):  file = open("addac.tv","a")  for i in a:  file.write(str(i))  file.write("\_")  file.write(str(sel0))  file.write("\_")  file.write(str(sel1))  file.write("\_")  for j in acc:  file.write(str(j))  file.write('\n')  file.close()  a = [0,0,1,1]  acc = [0,0,0,0]  c,acc = addac(a,0,0)  escrever(a,0,0,acc)  c,acc = addac(a,0,1)  escrever(a,0,1,acc)  c,acc = addac(a,1,0)  escrever(a,1,0,acc)  c,acc = addac(a,1,1)  escrever(a,1,1,acc) |

* **TestVector**

|  |
| --- |
| 0011\_0\_0\_0011  0011\_0\_1\_0110  0011\_1\_0\_1100  0011\_1\_1\_1001 |

* **Descrição Addac**

O módulo ADDAC une todos os módulos implementados até agora,

resultando em um bloco sum/sub de 4bits.

|  |
| --- |
| module addac(input logic [3:0] a, input logic sel0, sel1, clk, output logic [3:0] s, cout);  logic [3:0] na,m0out,accout,sum;  inversor inv(a, na);  mux m0(a,na,sel0,m0out);  somador4bit s4(m0out,accout,sel0,sum,cout);  mux m1(m0out,sum,sel1,s);  flopenr-4bit acc(clk,s,accout);  endmodule |

* **Testbench**

Definidos golden model e addac, foi necessária criação do arquivo de validação do circuito, também em system verilog. Utilizando as entradas que estão nos vetores de testes do arquivo addac.tv, aplicando ao módulo addac, gerando uma saída. Saída esta que é comparada com a saída do vetor de teste para verificação se há erros.

