Universidade Federal de São Carlos - UFSCar

Lorhan Sohaky de Oliveira Duda Kondo 740951

Experimento 04 - Implementação de um completo com 4 bits utilizando Verilog

São Carlos - SP

2017

Universidade Federal de São Carlos - UFSCar

Lorhan Sohaky de Oliveira Duda Kondo 740951

Experimento 04 - Implementação de um completo com 4 bits utilizando Verilog

Orientador: Fredy João Valente

Universidade Federal de São Carlos - UFSCar

Departamento de Computação

Ciência da Computação

Laboratório de Circuitos Digitais

São Carlos - SP

2017

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

FPGA	*Field Programmable Gate Array* - Arranjo de Portas Programáveis em Campo

Sumário

1 Resumo

O experimento tem o objetivo de entender como implementar um somador completo de 4 bits. Para tal, dividiu-se o experimento em 3 (três) etapas para facilitar o aprendizado.

A primeira etapa é para entender como utilizar um display de 7 (sete) segmentos, como dispositivo de saída do circuito, e como implementar algo similar ao componente TTL 7449[1].

A segunda etapa serve para entender como implementar um meio-somador de 1 (um) *bit* e a saída sendo apresentada em um display de 7 (sete) segmentos.

A terceira etapa tem o objetivo de implementar um somador completo de 4 (quatro) *bits*, tendo a saída apresentada em dois display de 7 (sete) segmentos.

[1] Para mais detalhes sobre o TTL 7449 acesse o Apêndice A.

2 Descrição da execução do experimento

Para a realização deste experimento, foram utilizados o programa Quartus 13.0 SP 1 e a placa *Field Programmable Gate Array* - Arranjo de Portas Programáveis em Campo (FPGA) Cyclone II - EP2C20F484C7.

2.1 ETAPA 1 – Display de 7 segmentos

Para representar um número de 4 *bits* na placa, utilizou-se 4 *switch*, cada um representando um bit do número. Como um segmento do *display* pode ser acendido em mais de um número, motou-se uma expressão lógica para cada segmento do *display*.

Tabela 1 – Tabela verdade utilizada para chegar na expressão de cada um dos segmentos.

A (SW[4])	**B (SW[3])**	**C (SW[2])**	**D (SW[1])**	**Saída em base decimal**
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Para o segmento 0 montou-se a expressão

$$\overline{A}.\overline{B}.\overline{C}.D + \overline{A}.B.\overline{C}.\overline{D}$$

para o segmento 1 montou-se a expressão

$$\overline{A}.B.\overline{C}.D + \overline{A}.B.C.\overline{D}$$

para o segmento 2 montou-se a expressão

$$\overline{A}.\overline{B}.C.\overline{D}$$

para o segmento 3 montou-se a expressão

$$\overline{A}.B.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.B.C.D$$

para o segmento 4 montou-se a expressão

$$\overline{A}.D + \overline{A}.B.\overline{C} + \overline{B}.\overline{C}.D$$

para o segmento 5 montou-se a expressão

$$\overline{A}.\overline{B}.D + \overline{A}.C.D + \overline{A}.\overline{B}.C$$

para o segmento 6 montou-se a expressão

$$\overline{A}.\overline{B}.\overline{C} + \overline{A}.B.C.D$$

.

Com tais expressões, montou-se um módulo em Verilog conforme Listing 2.1.

```
module decodificacao ( A, B, C, D, h0, h1, h2, h3, h4, h5, h6);
  input A, B, C, D;
  output h0, h1, h2, h3, h4, h5, h6;

  assign h0 = (~A&~B&~C&D) | (~A&B&~C&~D);
  assign h1 = (~A&B&~C&D) | (~A&B&C&~D);
  assign h2 = ~A&~B&C&~D;
  assign h3 = (~A&B&~C&~D) | (~A&~B&~C&D) | (~A&B&C&D);
  assign h4 = (~A&D) | (~A&B&~C) | (~B&~C&D);
  assign h5 = (~A&~B&D) | (~A&C&D) | (~A&~B&C);
  assign h6 = (~A&~B&~C) | (~A&B&C&D);

endmodule
```

Listing 2.1 – Módulo de decodificação.

Depois foram realizadas simulações e execução do circuito na placa FPGA.

2.2 ETAPA 2 – Meio-somador 1 bit

A operação aritmética mais simples é a soma de dois dígitos binários. Um circuito combinacional que implementa a adição de dois bits é chamado de meio-somador (half adder ou HAD). A Figura 1 ilustra um esquema de entradas e saída de um meio-somador. Um meio-somador de 1 bit deve respeitar a Tabela 2.

Figura 1 – Ilustração de um meio somador.

Tabela 2 – Tabela verdade de um meio somador.

A	**B**	**S (soma)**	**CarryOut**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

O circuito deve guardar o *CarryOut* (o "vai um") da soma, representando sua existência ou ausência através de um LED, ligando-o quando houver o carry, e mantendo-o desligado quando o carry não ocorrer.

Então montou-se o módulo de meio-somador, conforme Listing 2.2.

```
module meio_somador2 ( a, b, soma, cout);
    input a, b;
    output soma, cout;

  assign soma = a ^ b;
    assign cout = a * b;

endmodule
```

Listing 2.2 – Módulo do meio somador.

2.3 ETAPA 3 – Meio-somador 4 bits

Foi montato o somador completo, ou seja,levando em consideração o "vai um", depois fez-se os testes. Por fim, dividiu-se o resultado em dois display conforme o Listing 2.3.

```
module meio_somador2 ( a, b, soma, cout);
    input a, b;
    output soma, cout;
```

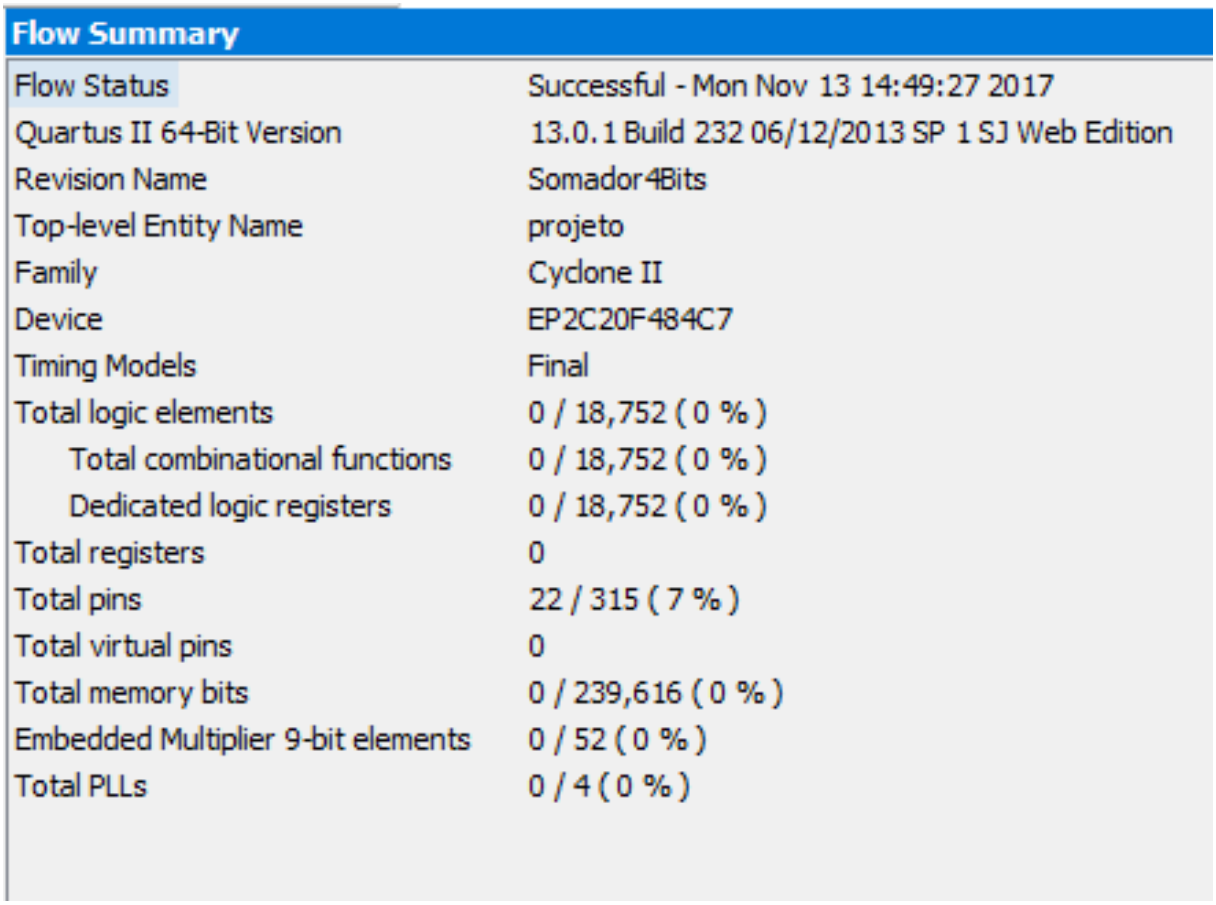
```
    assign soma = a ^ b;
      assign cout = a * b;

endmodule

module meio_somador3 ( a, b, c, soma, cout);
    input a, b, c;
    output soma, cout;

  wire carry_1, carry_2, soma_1;

    meio_somador2 m1 ( a, b, soma_1, carry_1);
    meio_somador2 m2 (soma_1, c, soma, carry_2);

  or o (cout, carry_1, carry_2);

endmodule

module somador_4_bits (a1, a2, b1, b2, c1, c2, d1, d2, soma1, soma2,
soma3, soma4, cout);
  input a1, a2, b1, b2, c1, c2, d1, d2;
  output soma1, soma2, soma3, soma4, cout;

  wire carry_1, carry_2, carry_3;

  meio_somador2 s1 ( a1, a2, soma1, carry_1);
  meio_somador3 s2 ( b1, b2, carry_1, soma2, carry_2);
  meio_somador3 s3 ( c1, c2, carry_2, soma3, carry_3);
  meio_somador3 s4 ( d1, d2, carry_3, soma4, cout);

endmodule

module separa ( A, B, C, D, E, z0, z1, z2, z3, z4);
  input A, B, C, D, E;
  output z0, z1, z2, z3, z4;

  assign z0 = (~A&E) | (~B&~C&~D&E);
  assign z1 = (~A&~B&D) | (~A&B&C&~D) | (A&~B&~C&~D);
  assign z2 = (~A&~B&C) | (~A&C&D) | (A&~B&~C&~D);
  assign z3 = (~A&B&~C&~D) | (A&~B&~C&D&~E);
  assign z4 = (~A&B&D) | (~A&B&C) | (A&~B&~C&~D) | (A&~B&~C&~E);

endmodule

module decodificacao ( A, B, C, D, h0, h1, h2, h3, h4, h5, h6);
  input A, B, C, D;
```

```
    output h0, h1, h2, h3, h4, h5, h6;

    assign h0 = (~A&~B&~C&D) | (~A&B&~C&~D);
    assign h1 = (~A&B&~C&D) | (~A&B&C&~D);
    assign h2 = ~A&~B&C&~D;
    assign h3 = (~A&B&~C&~D) | (~A&~B&~C&D) | (~A&B&C&D);
    assign h4 = (~A&D) | (~A&B&~C) | (~B&~C&D);
    assign h5 = (~A&~B&D) | (~A&C&D) | (~A&~B&C);
    assign h6 = (~A&~B&~C) | (~A&B&C&D);

endmodule

module projeto (SW, HEX0, HEX1);
    input [7:0]SW;
    output [6:0] HEX0,HEX1;
    wire z0, z1, z2, z3, z4, z5, z6, z7;

    wire s1, s2, s3, s4, cout;

    somador_4_bits soma ( SW[0], SW[4], SW[1], SW[5], SW[2], SW[6], SW
[3], SW[7], s1, s2, s3, s4, cout);
    separa s (cout, s4, s3, s2, s1, z0, z1, z2, z3, z4);

    assign z5 = 0;
    assign z6 = 0;
    assign z7 = 0;

    decodificacao dec1 ( z3, z2, z1, z0, HEX0[0], HEX0[1], HEX0[2], HEX0
[3], HEX0[4], HEX0[5], HEX0[6]);
    decodificacao dec2 ( 0, 0, 0, z4, HEX1[0], HEX1[1], HEX1[2], HEX1[3],
 HEX1[4], HEX1[5], HEX1[6]);

endmodule
```

Listing 2.3 – Módulo do somador completo e suas dependências.

3 Avaliação dos resultados do experimento

Figura 2 – Resultado da compilação do projeto inteiro.

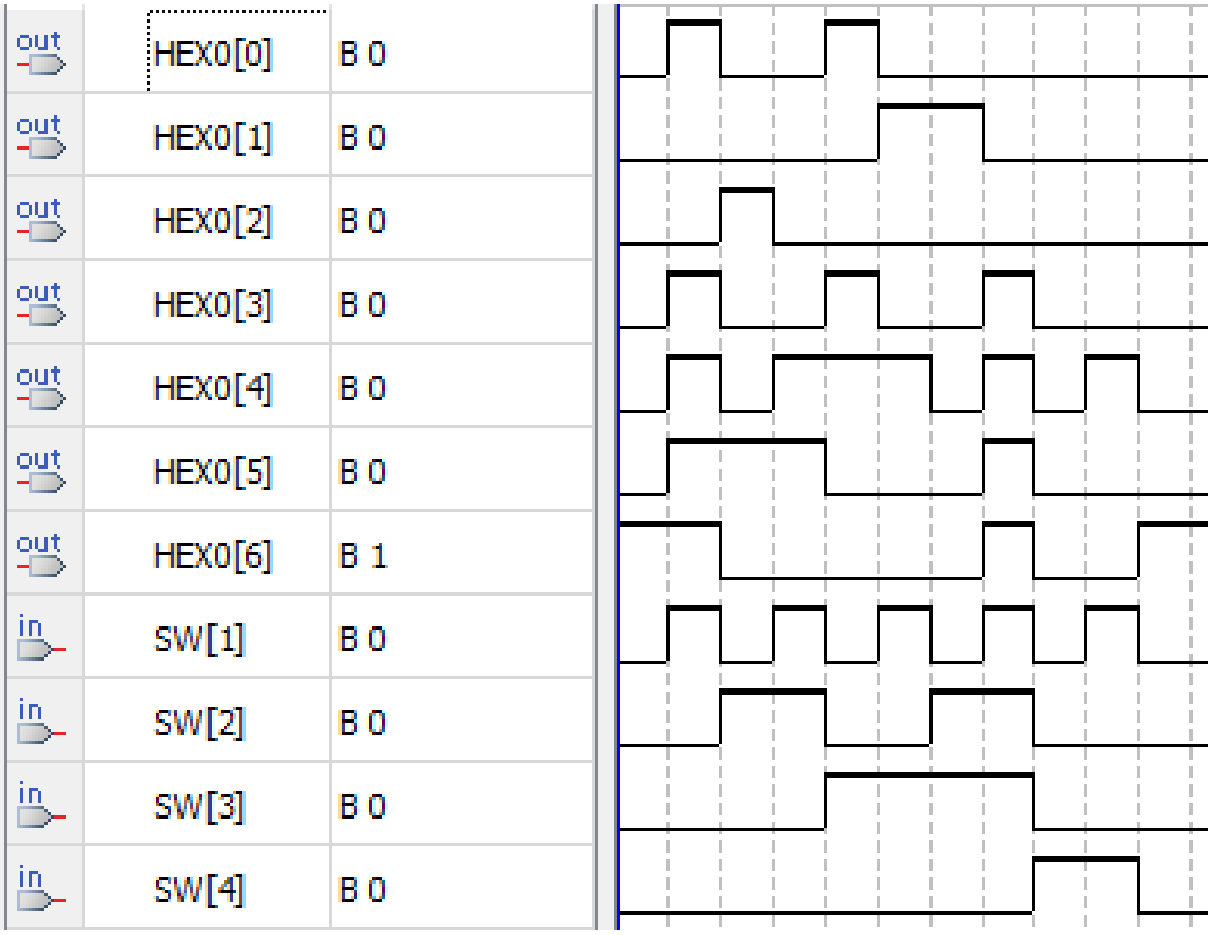


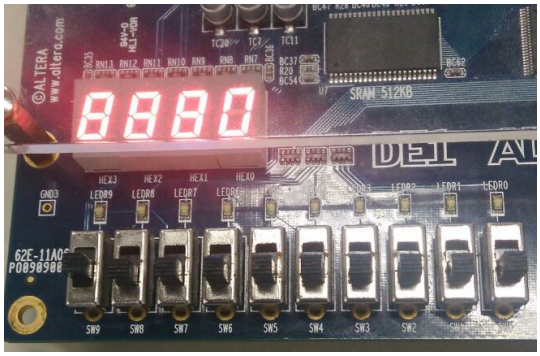
Flow Summary	
Flow Status	Successful - Mon Nov 13 14:49:27 2017
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Somador4Bits
Top-level Entity Name	projeto
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	0 / 18,752 (0 %)
Total combinational functions	0 / 18,752 (0 %)
Dedicated logic registers	0 / 18,752 (0 %)
Total registers	0
Total pins	22 / 315 (7 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

3.1 ETAPA 1 – Display de 7 segmentos

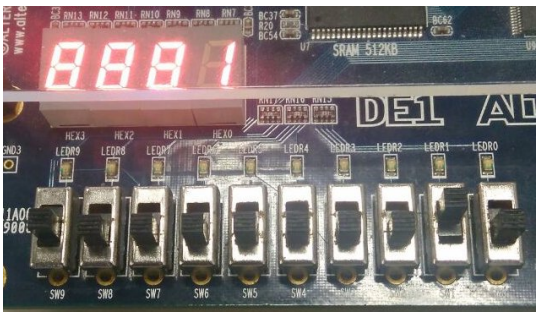
Verificou-se, para todos os casos de entrada, que o valor previsto pela Tabela 1 como saída era válido, demonstrando sucesso na implementação do experimento. Isso pode ser visualizado tanto pela simulação, como na execução na placa.

Figura 3 – Resultado da simulação da etapa 1.

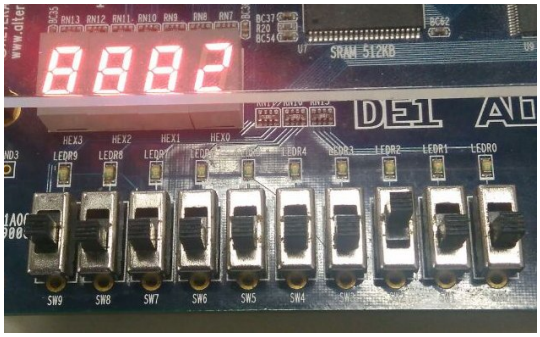




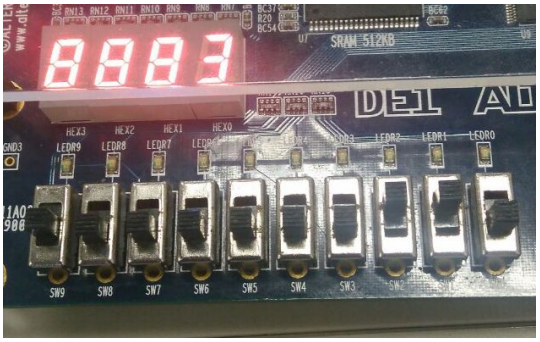
(a) Número 0



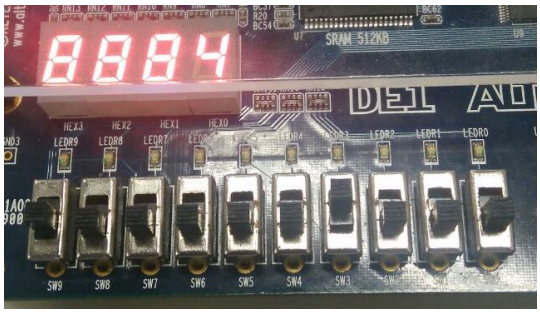
(b) Número 1



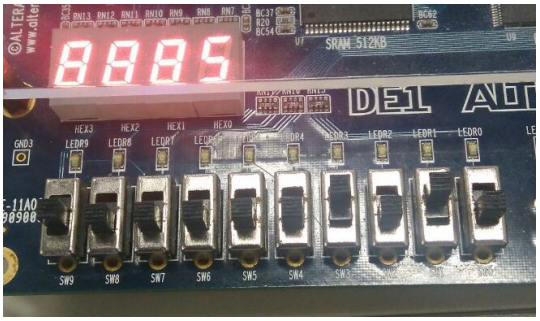
(c) Número 2



(d) Número 3

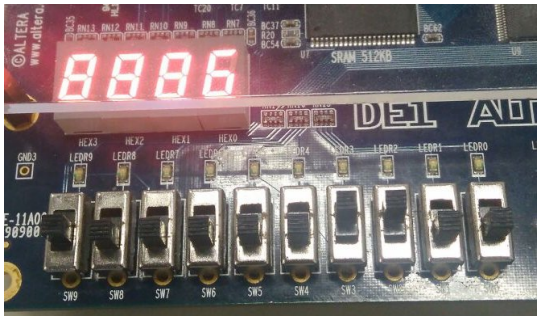


(e) Número 4

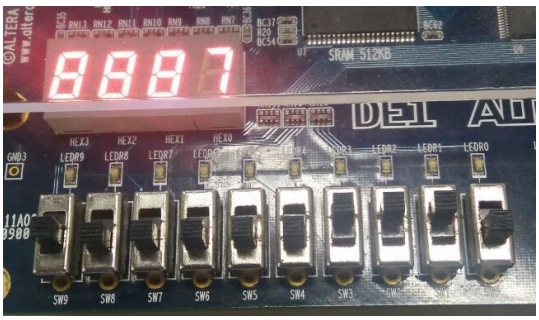


(f) Número 5

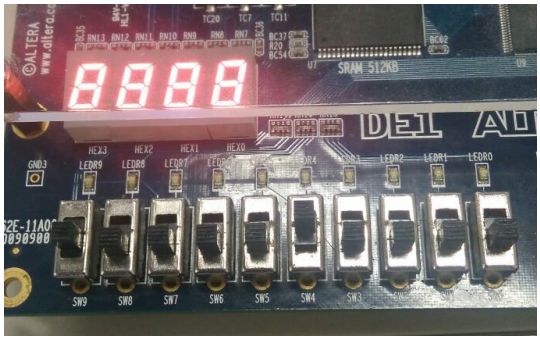
Figura 4 – Teste do circuito rodando na placa, no intervalo de 0-5.



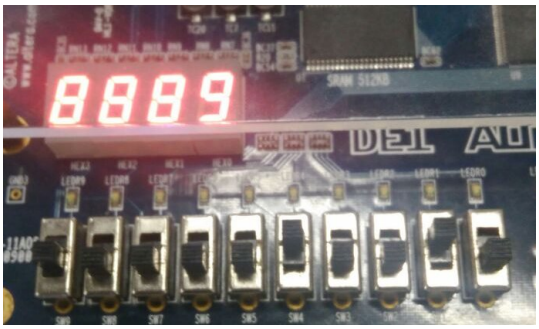
(a) Número 6



(b) Número 7



(c) Número 8



(d) Número 9

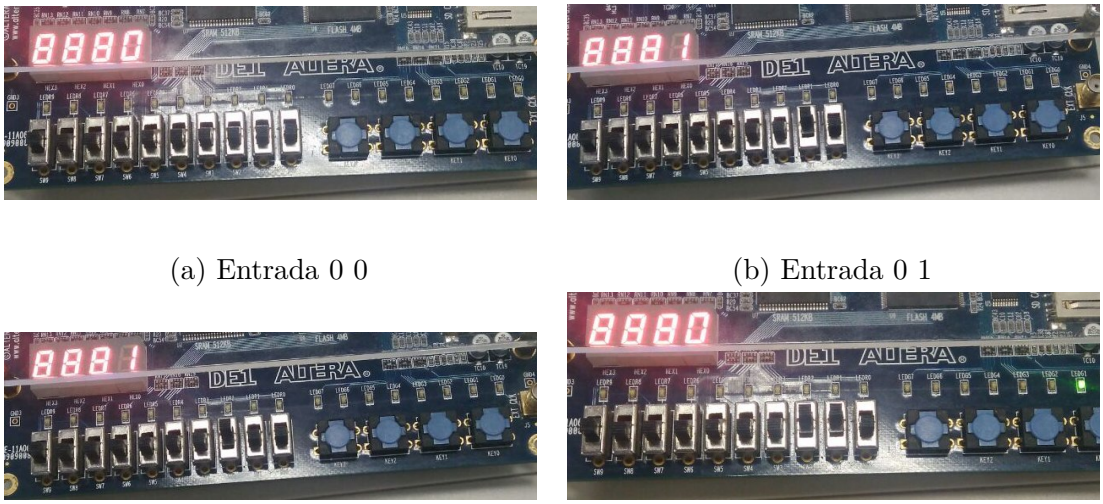
Figura 5 – Teste do circuito rodando na placa, no intervalo de 6-9.

3.2 ETAPA 2 – Meio-somador 1 bit

Na etapa 2, o experimento demonstrou os resultados esperados, de acordo com a Tabela 2.

Figura 6 – Resultado da simulação da etapa 2.

Após o *deploy* na placa no kit DE1, o kit educacional da Altera, o circuito apresentou os resultados esperados, representando o resultado da soma no display de 7 segmentos HEX0, conforme Figura 7.



(a) Entrada 0 0

(b) Entrada 0 1

(c) Entrada 1 0

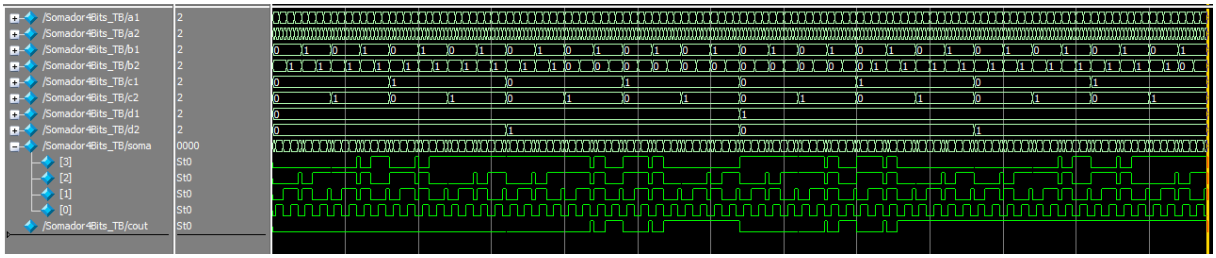
(d) Entrada 1 1

Figura 7 – Teste do circuito rodando na placa.

3.3 ETAPA 3 – Meio-somador 4 bits

O resultado foi como o esperado, conforme pode ser visto na simulação, execução na placa e no testbench.

Figura 8 – Simulação do projeto.



```
#       SOMADOR COMPLETO 1 bit
#
#   A  +   B  =   Soma    | Cout
# -----------------------------
# 0000 + 0000 =  0000     |  0
# 0000 + 0001 =  0001     |  0
# 0001 + 0000 =  0001     |  0
# 0001 + 0001 =  0010     |  0
# 0000 + 0010 =  0010     |  0
# 0000 + 0011 =  0011     |  0
# 0001 + 0010 =  0011     |  0
# 0001 + 0011 =  0100     |  0
# 0010 + 0000 =  0010     |  0
# 0010 + 0001 =  0011     |  0
# 0011 + 0000 =  0011     |  0
# 0011 + 0001 =  0100     |  0
# 0010 + 0010 =  0100     |  0
# 0010 + 0011 =  0101     |  0
# 0011 + 0010 =  0101     |  0
# 0011 + 0011 =  0110     |  0
# 0000 + 0100 =  0100     |  0
# 0000 + 0101 =  0101     |  0
# 0001 + 0100 =  0101     |  0
# 0001 + 0101 =  0110     |  0
# 0000 + 0110 =  0110     |  0
# 0000 + 0111 =  0111     |  0
# 0001 + 0110 =  0111     |  0
```

```
# 0001 + 0111 =  1000     |  0
# 0010 + 0100 =  0110     |  0
# 0010 + 0101 =  0111     |  0
# 0011 + 0100 =  0111     |  0
# 0011 + 0101 =  1000     |  0
# 0010 + 0110 =  1000     |  0
# 0010 + 0111 =  1001     |  0
# 0011 + 0110 =  1001     |  0
# 0011 + 0111 =  1010     |  0
# 0100 + 0000 =  0100     |  0
# 0100 + 0001 =  0101     |  0
# 0101 + 0000 =  0101     |  0
# 0101 + 0001 =  0110     |  0
# 0100 + 0010 =  0110     |  0
# 0100 + 0011 =  0111     |  0
# 0101 + 0010 =  0111     |  0
# 0101 + 0011 =  1000     |  0
# 0110 + 0000 =  0110     |  0
# 0110 + 0001 =  0111     |  0
# 0111 + 0000 =  0111     |  0
# 0111 + 0001 =  1000     |  0
# 0110 + 0010 =  1000     |  0
# 0110 + 0011 =  1001     |  0
# 0111 + 0010 =  1001     |  0
# 0111 + 0011 =  1010     |  0
# 0100 + 0100 =  1000     |  0
# 0100 + 0101 =  1001     |  0
# 0101 + 0100 =  1001     |  0
# 0101 + 0101 =  1010     |  0
# 0100 + 0110 =  1010     |  0
# 0100 + 0111 =  1011     |  0
# 0101 + 0110 =  1011     |  0
```

```
# 0101 + 0110 =  1011     |  0
# 0101 + 0111 =  1100     |  0
# 0110 + 0100 =  1010     |  0
# 0110 + 0101 =  1011     |  0
# 0111 + 0100 =  1011     |  0
# 0111 + 0101 =  1100     |  0
# 0110 + 0110 =  1100     |  0
# 0110 + 0111 =  1101     |  0
# 0111 + 0110 =  1101     |  0
# 0111 + 0111 =  1110     |  0
# 0000 + 1000 =  1000     |  0
# 0000 + 1001 =  1001     |  0
# 0001 + 1000 =  1001     |  0
# 0001 + 1001 =  1010     |  0
# 0000 + 1010 =  1010     |  0
# 0000 + 1011 =  1011     |  0
# 0001 + 1010 =  1011     |  0
# 0001 + 1011 =  1100     |  0
# 0010 + 1000 =  1010     |  0
# 0010 + 1001 =  1011     |  0
# 0011 + 1000 =  1011     |  0
```

```
# 0001 + 1110 =  1111     |  0
# 0001 + 1111 =  0000     |  1
# 0010 + 1100 =  1110     |  0
# 0010 + 1101 =  1111     |  0
# 0011 + 1100 =  1111     |  0
# 0011 + 1101 =  0000     |  1
# 0010 + 1110 =  0000     |  1
# 0010 + 1111 =  0001     |  1
# 0011 + 1110 =  0001     |  1
# 0011 + 1111 =  0010     |  1
# 0100 + 1000 =  1100     |  0
# 0100 + 1001 =  1101     |  0
# 0101 + 1000 =  1101     |  0
# 0101 + 1001 =  1110     |  0
# 0100 + 1010 =  1110     |  0
# 0100 + 1011 =  1111     |  0
# 0101 + 1010 =  1111     |  0
# 0101 + 1011 =  0000     |  1
# 0110 + 1000 =  1110     |  0
```

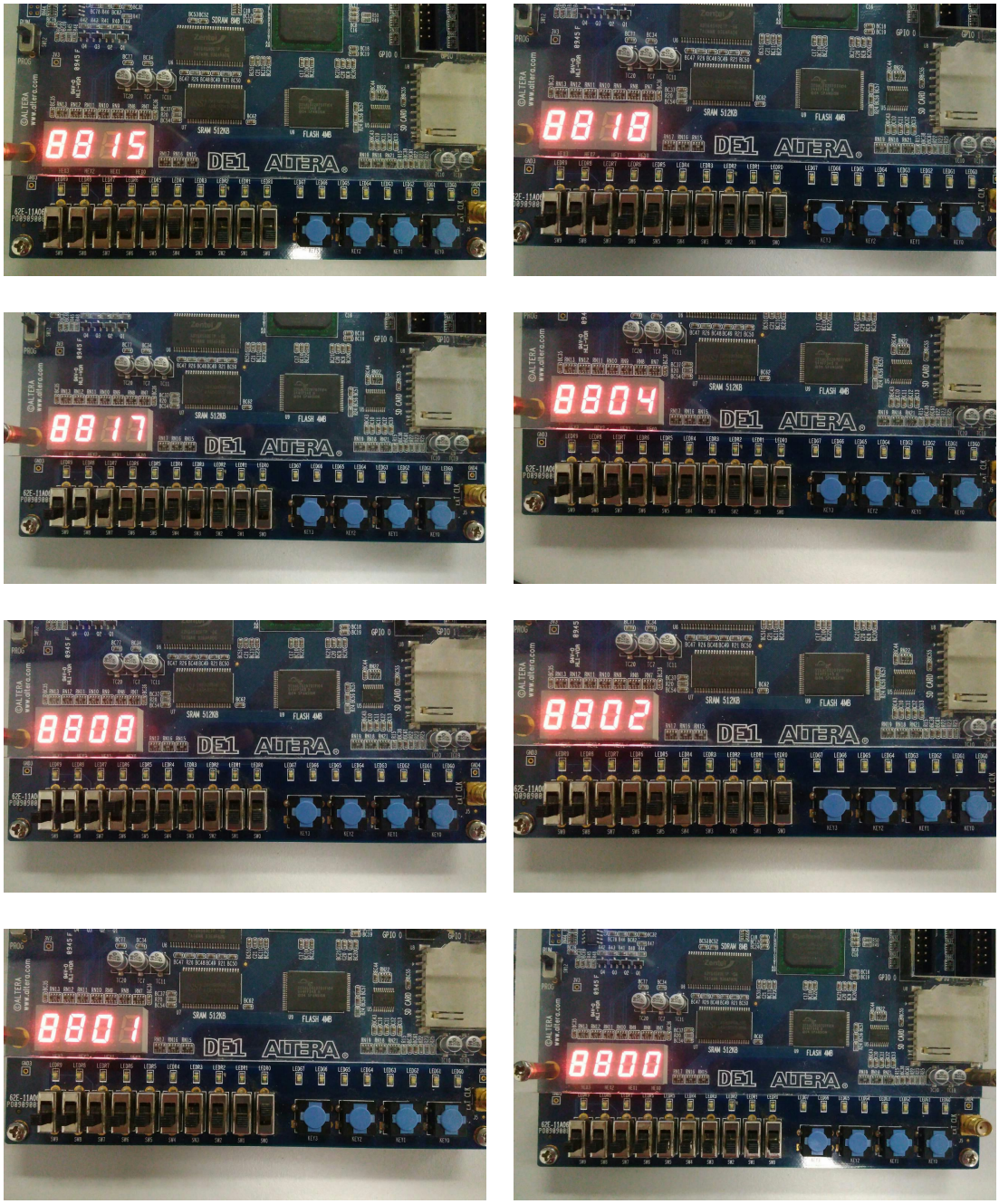


Figura 10 – Execução do projeto na FPGA.

Os testbench podem ser encontrados nos apêndices.

4 Análise crítica e discussão

Teve-se dificuldade para compreender o funcionamento do Verilog, como por exemplo laços de repetição , separação em módulo e simulação.

Apêndices

APÊNDICE A – Testbench meio-somador2

```
module meio_somador2_TB;

  integer a1, a2;
  wire soma, cout;

  meio_somador2 s( a1[0], a2[0], soma, cout);

  initial
  begin
    $display("\tMEIO SOMADOR\n");
    $display("A | B | Soma | Cout");
    $display("--------------------");
    for( a1 = 0; a1 < 2; a1 = a1+1) begin
      for( a2 = 0; a2 < 2; a2 = a2+1) begin
        #1
        $display("%x | %x |  %x   |  %x", a1[0], a2[0], soma, cout);
      end
    end
  end
endmodule
```

Listing A.1 – Testbench do módulo meio-somador2.

APÊNDICE B – Testbench meio-somador3

```
module meio_somador3_TB;
  integer a1, a2, a3;
  wire soma, cout;

  meio_somador3 s( a1[0], a2[0], a3[0], soma, cout);

  initial
  begin
    $display("\tSOMADOR COMPLETO 1 bit\n");
    $display("A | B | C | Soma | Cout");
    $display("--------------------");
    for( a1 = 0; a1 < 2; a1 = a1+1) begin
      for( a2 = 0; a2 < 2; a2 = a2+1) begin
        for( a3 = 0; a3 < 2; a3 = a3+1) begin
          #1
          $display("%x | %x | %x |  %x   |  %x", a1[0], a2[0], a3[0], soma,
    cout);
        end
      end
    end
  end
endmodule
```

Listing B.1 – Testbench do módulo meio-somador3.

APÊNDICE C – Testbench somador completo 4 bits

```
module Somador4Bits_TB;
  integer a1, a2, b1, b2, c1, c2, d1, d2;
  wire [3:0]soma;
  wire cout;

  somador_4_bits s( a1[0], a2[0], b1[0], b2[0], c1[0], c2[0], d1[0], d2[0],
    soma[0], soma[1], soma[2], soma[3], cout);

  initial
  begin
    $display("\tSOMADOR COMPLETO 1 bit\n");
    $display(" A + B = Soma | Cout");
    $display("-------------------------------");
    for( d1=0; d1<2; d1=d1+1) begin
      for( d2=0; d2<2; d2=d2+1) begin

        for( c1=0; c1<2; c1=c1+1) begin
          for( c2=0; c2<2; c2=c2+1) begin

            for( b1=0; b1<2; b1=b1+1) begin
              for( b2=0; b2<2; b2=b2+1) begin

                for( a1=0; a1<2; a1=a1+1) begin
                  for( a2=0; a2<2; a2=a2+1) begin
                    #1
                    $display("%x%x%x%x + %x%x%x%x = %x%x%x%x | %x", d1
    [0], c1[0], b1[0], a1[0], d2[0], c2[0], b2[0], a2[0], soma[3],soma[2],
    soma[1],soma[0], cout);
                  end
                end
              end
            end
          end
        end
      end
    end
  end
endmodule
```

Listing C.1 – Testbench do módulo de somador completo 4 bits.

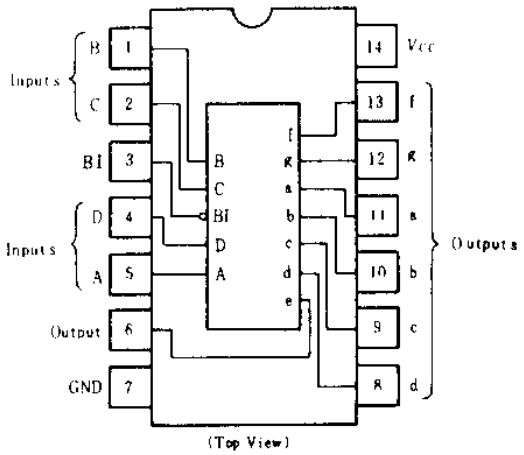
Anexos

ANEXO A – *Datasheet* do componente 7449

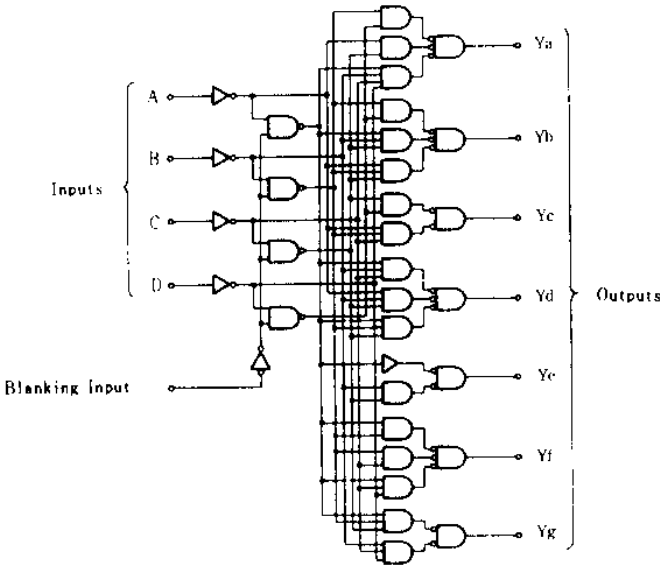
HD74LS49 ●BCD-to-Seven Segment Decoder/Driver (with Open collector outputs)

The HD74LS49 features active-high outputs for driving lamp buffer. This circuit incorporates a direct blanking input. Segment identification and resultant displays are shown below. Display patterns for BCD input counts above 9 are unique symbols to authenticate input conditions. It contains an over-riding blanking input (BI) which can be used to control the lamp intensity by pulsing or to inhibit the output. Inputs and outputs are entirely compatible for use with TTL or DTL logic outputs.

■PIN ARRANGEMENT



■BLOCK DIAGRAM



■ABSOLUTE MAXIMUM RATINGS

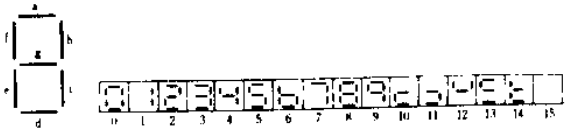
Item	Symbol	Ratings	Unit
Supply voltage	V_{CC}	7.0	V
Input voltage	V_{IN}	7.0	V
Output current (off state)	$I_{O(off)}$	1	mA
Operating temperature range	T_{opr}	− 20 ~ + 75	℃
Storage temperature range	T_{stg}	65 ~ + 150	℃

■FUNCTION TABLE

Decimal or Function	Inputs					Outputs							Note
	D	C	B	A	BI	a	b	c	d	e	f	g	
0	L	L	L	L	H	H	H	H	H	H	H	L	
1	L	L	L	H	H	L	H	H	L	L	L	L	
2	L	L	H	L	H	H	H	L	H	H	L	H	
3	L	L	H	H	H	H	H	H	H	L	L	H	
4	L	H	L	L	H	L	H	H	L	L	H	H	
5	L	H	L	H	H	H	L	H	H	L	H	H	
6	L	H	H	L	H	L	L	H	H	H	H	H	
7	L	H	H	H	H	H	H	H	L	L	L	L	
8	H	L	L	L	H	H	H	H	H	H	H	H	1
9	H	L	L	H	H	H	H	H	L	L	H	H	
10	H	L	H	L	H	L	L	L	H	H	L	H	
11	H	L	H	H	H	L	L	H	H	L	L	H	
12	H	H	L	L	H	L	H	L	L	L	H	H	
13	H	H	L	H	H	H	L	L	H	L	H	H	
14	H	H	H	L	H	L	L	L	H	H	H	H	
15	H	H	H	H	H	L	L	L	L	L	L	L	
BI	×	×	×	×	L	L	L	L	L	L	L	L	2

H; high level, L; low level, X; irrelevant

Notes: 1. The blanking input (BI) must be open or held at a high logic level when output functions 0 through 15 are desired.
2. When a low logic level is applied directly to the blanking input (BI), all segment outputs are low regardless of the level of any other input.



■ELECTRICAL CHARACTERISTICS (Ta= −20 ~ +75°C)

Item	Symbol	Test Conditions		min	typ*	max	Unit
Input voltage	V_{IH}			2.0	–	–	V
	V_{IL}			–	–	0.8	V
Output current	I_{OH}	V_{CC}=4.75V, V_{IH}=2V, V_{IL}=0.8V, V_{OH}=5.5V		–	–	250	μA
Output voltage	V_{OL}	V_{CC}=4.75V, V_{IH}=2V, V_{IL}=0.8V	I_{OL}=4mA	–	–	0.4	V
			I_{OL}=8mA	–	–	0.5	
Input current	I_{IH}	V_{CC}=5.25V, V_I=2.7V		–	–	20	μA
	I_{IL}	V_{CC}=5.25V, V_I=0.4V		–	–	−0.4	mA
	I_I	V_{CC}=5.25V, V_I=7V		–	–	0.1	mA
Supply current **	I_{CC}	V_{CC}=5.25V		–	8	15	mA
Input clamp voltage	V_{IK}	V_{CC}=4.75V, I_{IN}=−18mA		–	–	−1.5	V

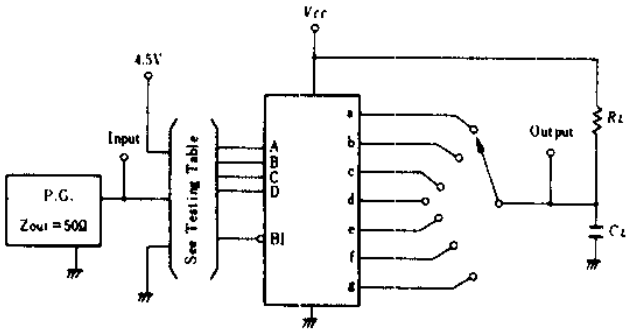
* V_{CC}=5V, Ta=25°C
** I_{CC} is measured with all outputs open and all inputs at 4.5V.

■SWITCHING CHARACTERISTICS (V_{CC}=5V, Ta=25°C)

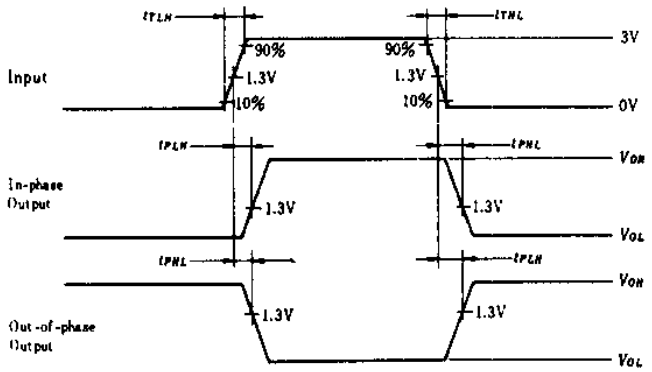
Item	Symbol	Input	Test Conditions	min	typ	max	Unit
Propagation delay time	t_{PHL}	A	C_L=15pF, R_L=2kΩ	–	–	100	ns
	t_{PLH}			–	–	100	
	t_{PHL}	BI	C_L=15pF, R_L=6kΩ	–	–	100	ns
	t_{PLH}			–	–	100	

■TESTING METHOD

1) Test Circuit



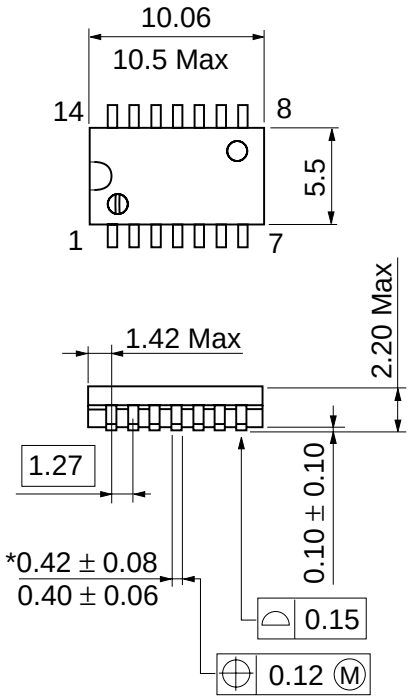
Waveform

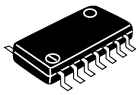


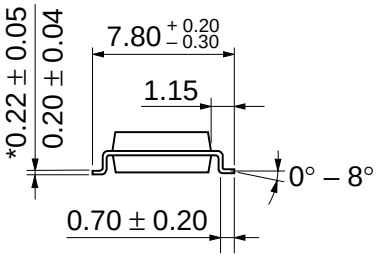
2) Testing Table

Item	Inputs					Outputs						
	BI	D	C	B	A	a	b	c	d	e	f	g
	4.5V	GND	GND	GND	IN	OUT	–	–	OUT	OUT	OUT	–
t_{PLH}	4.5V	GND	GND	4.5V	IN	–	–	OUT	–	OUT	–	–
t_{PHL}	4.5V	GND	4.5V	4.5V	IN	OUT	OUT	–	OUT	OUT	OUT	OUT
	IN	GND	GND	GND	GND	OUT	OUT	OUT	OUT	OUT	OUT	–

Unit: mm



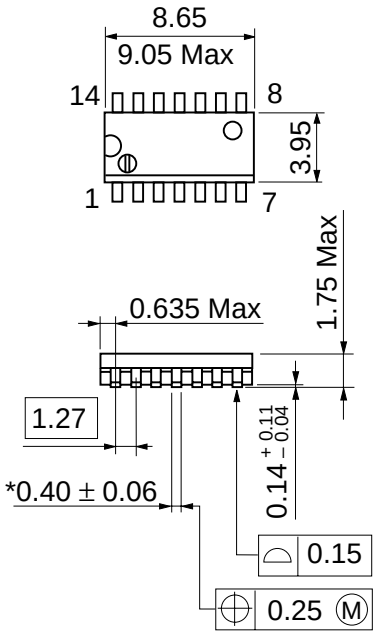


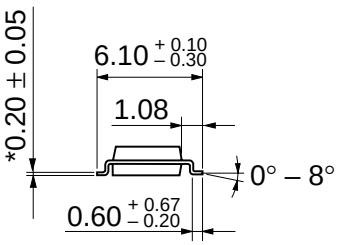


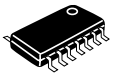
*Dimension including the plating thickness
Base material dimension

Hitachi Code	FP-14DA
JEDEC	—
EIAJ	Conforms
Weight (reference value)	0.23 g

Unit: mm







Hitachi Code	FP-14DN
JEDEC	Conforms
EIAJ	Conforms
Weight (reference value)	0.13 g

*Pd plating