



**Universidade Federal de Campina Grande**  
**Centro de Engenharia Elétrica e Informática**  
**Departamento de Engenharia Elétrica**

**Relatório de Projeto**

**Título**

Sistema de Segurança Residencial

**Disciplina**

Laboratório de Circuitos Lógicos

**Professora**

Fernanda Cecília Correia Lima Loureiro

*fernanda@dee.ufcg.edu.br*

**Equipe**

**Aluno 1: Alysson Machado de Oliveira Barbosa**

**Aluno 2: Matheus Victor Alves Nascimento**

*alysson.barbosa@ee.ufcg.edu.br*

*matheus.nascimento@ee.ufcg.edu.br*

Campina Grande – PB

Dezembro de 2020

## Lista de Ilustrações

Figura 1 - Diagrama lógico do contador síncrono módulo 10	4
Figura 2 - Implementação em verilog do contador síncrono módulo 10.	5
Figura 3 - Diagrama lógico do comparador binário de 4 bits	5
Figura 4 - Implementação em verilog do comparador binário de 4 bits.	6
Figura 5 - Diagrama lógico do display de 7 segmentos para visualização da senha	7
Figura 6 - Implementação em verilog do display de 7 segmentos.	8
Figura 7 - Implementação do Flip-Flop JK em Verilog.	9
Figura 8 - Implementação do contador síncrono módulo 6 em verilog.	10
Figura 9 - Implementação do contador síncrono módulo 3 em verilog.	10
Figura 10 - Implementação do contador síncrono módulo 2 em verilog.	11
Figura 11 - Diagrama Lógico do Temporizador de 3 minutos.	11
Figura 12 - Implementação do temporizador de 3 minutos em verilog.	12
Figura 13 - Diagrama lógico do sistema de senhas para habilitação/desabilitação.	13
Figura 14 - Implementação do sistema de senhas em verilog.	14
Figura 15 - Netlist do sistema de senhas.	15
Figura 16 - Planta baixa da residência.	17
Figura 17 - Tabela verdade do sistema de acionamento do alarme.	18
Figura 18 - Diagrama lógico do sistema de acionamento de alarme.	19
Figura 19 - Diagrama lógico do projeto inteiro.	20
Figura 20 - Implementação em verilog do sistema de acionamento de alarme.	20

# Sumário

Introdução	3
Objetivos	3
Sistema de Habilitação/Desabilitação por Senha	3
Sistema de Controle de Alarme	16
Melhorias e Dificuldades	21
<b>Referências</b>	<b>22</b>

# 1. Introdução

Fazendo uso de conceitos (e práticas) desenvolvidos desde o primeiro experimento, foi implementado, utilizando a linguagem Verilog, um Sistema de Segurança Residencial Automático através da detecção de violação na residência e um sistema de senhas.

## 2. Objetivos

O sistema de segurança residencial implementado com linguagem de descrição de hardware deve funcionar da seguinte maneira:

a) Através de uma senha de 4 dígitos o usuário pode habilitar/desabilitar o sistema de segurança. A senha deve ser mostrada em um display.

b) Implementar uma forma simples do usuário digitar a senha, como um botão para cada dígito em um intervalo de 0 a 9, no sistema de numeração decimal.

c) Caso a senha esteja correta, é necessário adicionar um tempo de espera antes que o alarme fique ativado.

d) A casa deve possuir uma planta arquitetônica definida, em que nas portas, janelas ou qualquer local que permita a entrada de pessoas, um sensor deve ser instalado. Caso o alarme esteja ativado, a violação de qualquer um desses locais deve disparar um alarme, teremos também sensores no portão principal na porta de entrada da casa, além de ter a cerca elétrica que funcionará como sensor de detecção de invasores.

e) Caso o alarme seja ativado, o dono só poderá desativar através da senha, caso disparado o alarme irá soar um som de aviso e enviará uma mensagem para o celular da vítima sobre o ocorrido.

## 3. Sistema de Habilitação/Desabilitação por Senha

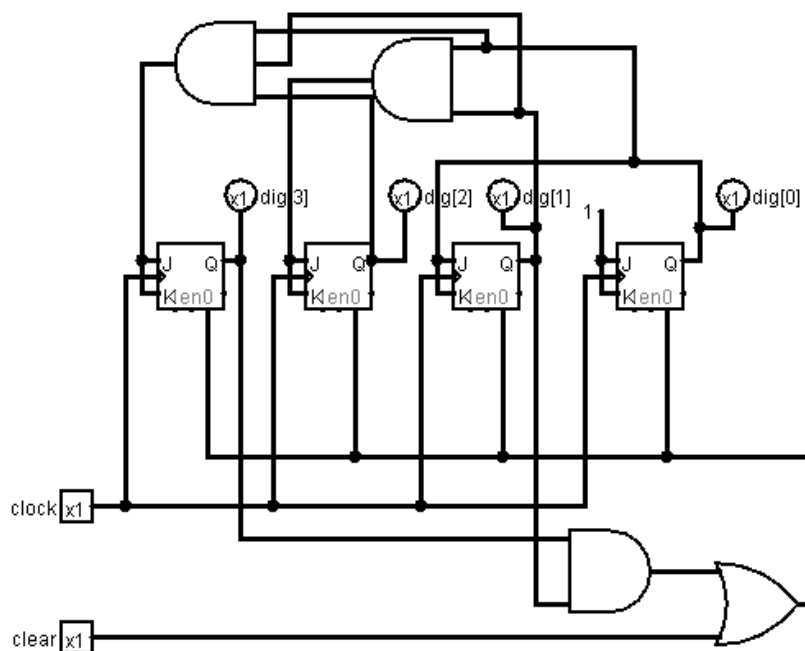
Através de uma senha de 4 dígitos o usuário pode habilitar/desabilitar o sistema de segurança. A senha deve ser mostrada em um display. Para implementar essa ideia, será utilizada botões para cada um dos dígitos da senha, visando facilitar a interação com o usuário. Cada um desses botões, deverá ativar o clock na borda de subida de um contador síncrono decádico (módulo 10). Desse modo, podemos implementar um sistema de entrada em que os dígitos estão na representação em decimal, mas a cada clock, os dígitos são representados por binários de 4 bits, gerando duas vantagens específicas:

- A senha que o usuário interage em decimal e com saída binária de 4 bits para cada um dos 4 dígitos pode ser utilizada em um módulo comparador binário. Desse modo, é possível verificar se o dígito fornecido é condizente com a senha cadastrada no sistema de segurança através de um comparador de 4 bits.
- Obtendo os dígitos em binário de 4 bits, será utilizado um display de 7 segmentos para que o usuário possa visualizar os dígitos escolhidos em decimal da senha. Ou seja, a cada vez que ele pressionar um dos 4 botões da senha de entrada, o display será atualizado no intervalo de 0-9, até que os 4 dígitos sejam iguais a senha cadastrada e o display mostre 0000 (por questões de segurança, a senha correta não deve ficar salva no display, ela é apagada logo após que a senha correta for informada).

Para essa finalidade, alguns módulos foram implementados utilizando o Logisim e, em seguida, foram feitos no Quartus usando a linguagem de descrição de hardware verilog:

- Contador decádico síncrono;
- Comparador binário de 4 bits;
- Display de 7 segmentos;

**Figura 1 - Diagrama lógico do contador síncrono módulo 10 para gerar números em decimal de 0-9 em binários de 4 bits**



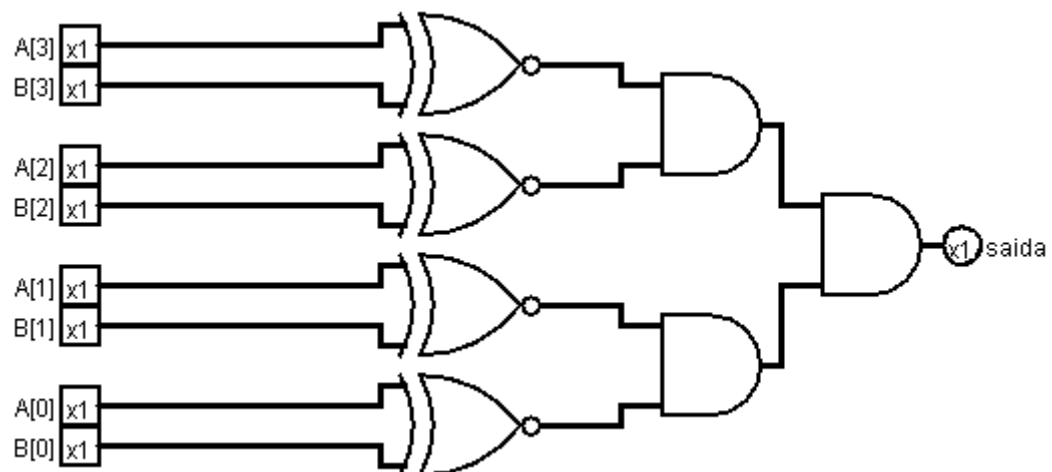
Fonte: Imagem Autoral

**Figura 2 - Implementação em verilog do contador síncrono módulo 10**

```
projeto-quartus > ≡ contador_decadico.v
1  // modulo do contador de decada sincrono
2  module contador_decadico(clk, clear, s);
3
4      input clk, clear;
5      output [3:0]s;
6
7      wire and1;
8      wire and2;
9      wire and3;
10     wire clr_aux;
11
12     // parte combinacional para o contador
13     assign and3 = (s[3] & s[1]);
14     assign clr_aux = and3 | clear;
15
16     // usando flip-flops jk para montar um contador de modulo 10 (parte sequencial)
17     ff_jk ff1(clr_aux, clk, 1, 1, s[0]);
18     ff_jk ff2(clr_aux, clk, s[0], s[0], s[1]);
19     assign and1 = s[0] & s[1];
20     ff_jk ff3(clr_aux, clk, and1, and1, s[2]);
21     assign and2 = s[0] & s[1] & s[2];
22     ff_jk ff4(clr_aux, clk, and2, and2, s[3]);
23
24
25 endmodule
26 // sistema de segurança residencial
27 // alysson machado e matheus victor
```

Fonte: Imagem Autoral

**Figura 3 - Diagrama lógico do comparador binário de 4 bits para verificar a senha do usuário com relação a senha do sistema**



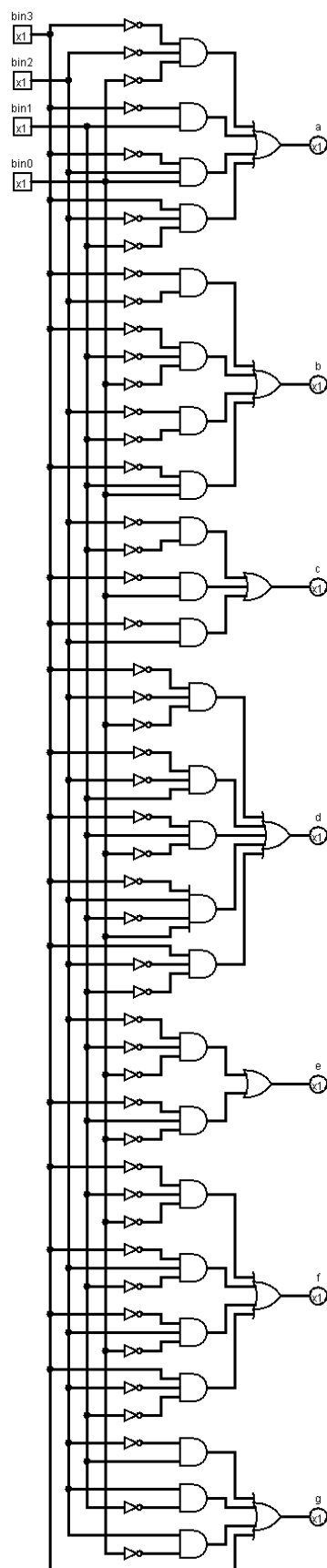
Fonte: Imagem Autoral

Figura 4 - Implementação em verilog do comparador binário de 4 bits

```
projeto-quartus > ≡ comparador_4bits.v
1  // modulo comparador de 4 bits em binario
2  module comparador_4bits(A, B, out);
3
4      input [3:0]A, B;
5      output out;
6
7      wire xnor1, xnor2, xnor3, xnor4;
8      wire and1, and2;
9
10     // so e verdade quando os bits sao iguais
11     assign xnor1 = ~(A[3] ^ B[3]);
12     assign xnor2 = ~(A[2] ^ B[2]);
13     assign xnor3 = ~(A[1] ^ B[1]);
14     assign xnor4 = ~(A[0] ^ B[0]);
15
16     // garante que todos os bits sejam iguais
17     assign and1 = xnor1 & xnor2;
18     assign and2 = xnor3 & xnor4;
19     assign out = and1 & and2;
20
21 endmodule
22 // sistema de segurança residencial
23 // alysson machado e matheus victor
24
```

Fonte: Imagem Autoral

**Figura 5 - Diagrama lógico do display de 7 segmentos para visualização da senha**



Fonte: Imagem Autoral



**Figura 6 - Implementação em verilog do display de 7 segmentos**

```
projeto-quartus > ≡ display_7segmentos.v
1  // modulo display de 7 segmentos
2  module display_7segmentos(dados, segmentos);
3
4      input [3:0]dados;
5      output reg [6:0]segmentos;
6
7      always @(*)
8          // aciona os segmentos a cada digito binario com 4 bits
9          case(dados)
10             0: segmentos = 7'b1111110;
11             1: segmentos = 7'b0110000;
12             2: segmentos = 7'b1101101;
13             3: segmentos = 7'b1111001;
14             4: segmentos = 7'b0110011;
15             5: segmentos = 7'b1011011;
16             6: segmentos = 7'b1011111;
17             7: segmentos = 7'b1110000;
18             8: segmentos = 7'b1111111;
19             9: segmentos = 7'b1110011;
20             default: segmentos = 7'b0000000;
21         endcase
22     endmodule
23 // sistema de segurança residencial
24 // alysson machado e matheus victor
```

Fonte: Imagem Autoral

A segunda parte do sistema de senhas é um temporizador de 3 minutos. Caso a senha seja digitada corretamente, o circuito deve ter um tempo de espera de 3 minutos. Caso a senha esteja incorreta, nada deverá acontecer e caso a senha salva no sistema seja 0000, o temporizador também não deve funcionar.

O temporizador deve gerar 3 tipos de saídas diferentes, cada uma delas com quantidades de bits diferentes, dado que:

- Uma das saídas para o temporizador representará o minuto, usando um contador síncrono módulo 3 para gerar números em binário entre 0-2;
- Uma das saídas para o temporizador representará o primeiro dígito do segundo, usando um contador síncrono módulo 6 para gerar números em binário entre 0-5;
- Uma das saídas para o temporizador representará o segundo dígito do segundo, usando um contador síncrono módulo 10 para gerar números em binário entre 0-9;

Os diversos contadores presentes no temporizador devem estar em sincronia para que o sistema funcione. Desse modo, o nosso temporizador irá começar em 0 minutos e 00 segundos e terminará a contagem em 2 minutos e 59 segundos, reiniciando toda vez que chegar em 3 minutos e 00 segundos. Além disso, será necessário um gerador de frequência (4 hz) que esteja funcionando constantemente, gerando clocks de subida e fazendo o temporizador funcionar. Mais detalhes serão apresentados consequentemente.

Para essa finalidade, alguns módulos foram implementados utilizando o Logisim e, em seguida, foram feitos no Quartus usando a linguagem de descrição de hardware verilog:

- Contador Síncrono módulo 10 (Figura 1);
- Flip-Flop JK;
- Contador Síncrono módulo 6;
- Contador Síncrono módulo 3;
- Contador Síncrono módulo 2;
- Temporizador de 3 min;

**Figura 7 - Implementação do Flip-Flop JK em Verilog**

```
projeto-quartus > ff_jk.v
1  // modulo flip-flop jk
2  module ff_jk(clear, clk, j, k, q);
3
4      input clear, clk, j, k;
5      output reg q;
6
7
8      always @ (posedge clk, posedge clear)
9          begin
10             // entrada sincrona do clear
11             if(clear) q <= 1'b0;
12             else
13                 begin
14                     case({j, k})
15                         // casos de estado do flip-flop jk
16                         0: q = q;
17                         1: q = 0;
18                         2: q = 1;
19                         3: q = ~q;
20                     endcase
21                 end
22             end
23
24 endmodule
25 // sistema de segurança residencial
26 // alysson machado e matheus victor
```

Fonte: Imagem Autoral

**Figura 8 - Implementação do contador síncrono módulo 6 em verilog**

```
1 // modulo contador modulo 6
2 module contador_modulo6(clk, clear, s);
3
4     input clk, clear;
5     output [2:0]s;
6
7     wire and1, and2;
8     wire clear_aux;
9
10    // parte sequencial do contador sincrono modulo 6
11    ff_jk ff1(clear_aux, clk, 1, 1, s[0]);
12    ff_jk ff2(clear_aux, clk, s[0], s[0], s[1]);
13    assign and1 = s[0] & s[1];
14    ff_jk ff3(clear_aux, clk, and1, and1, s[2]);
15
16    // parte combinacional do contador sincrono modulo 6
17    assign and2 = s[1] & s[2];
18    assign clear_aux = clear | and2;
19
20 endmodule
21 // sistema de segurança residencial
22 // alysson machado e matheus victor
```

Fonte: Imagem Autoral

**Figura 9 - Implementação do contador síncrono módulo 3 em verilog**

```
projeto-quartus > ≡ contador_modulo3.v
1 // modulo contador modulo 3
2 module contador_modulo3(clk, clear, s);
3
4     input clk, clear;
5     output [1:0]s;
6
7     wire and1;
8     wire clear_aux;
9
10    // parte sequencial do contador sincrono modulo 3
11    ff_jk ff1(clear_aux, clk, 1, 1, s[0]);
12    ff_jk ff2(clear_aux, clk, s[0], s[0], s[1]);
13
14    // parte combinacional do contador sincrono modulo 3
15    assign and1 = s[0] & s[1];
16    assign clear_aux = and1 | clear;
17
18 endmodule
19 // sistema de segurança residencial
20 // alysson machado e matheus victor
```

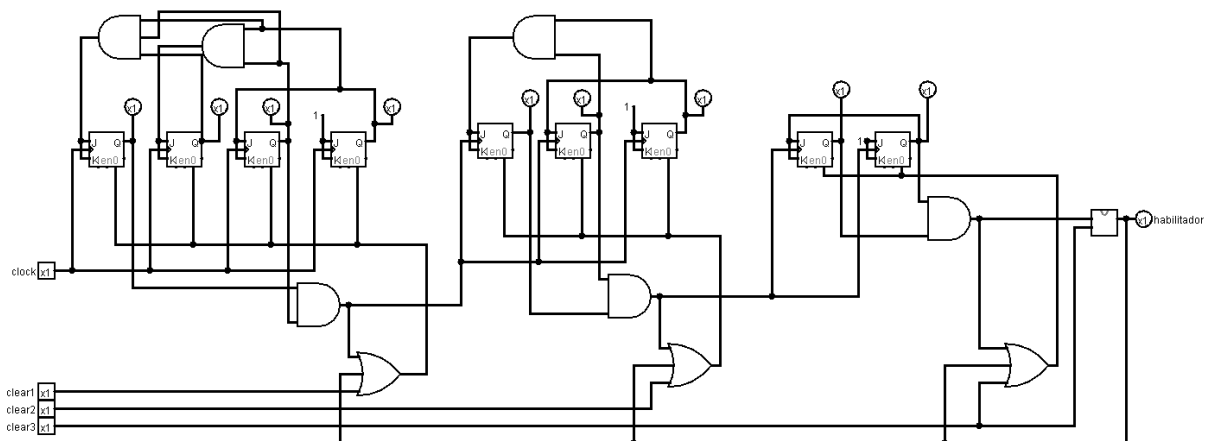
Fonte: Imagem Autoral

**Figura 10 - Implementação do contador síncrono módulo 2 em verilog**

```
projeto-quartus > ≡ contador_modulo2.v
1  // modulo contador modulo 2 sincrono
2  module contador_modulo2(clk, clear, s);
3
4      input clk, clear;
5      output s;
6
7      // parte sequencial do contador sincrono modulo 2
8      ff_jk ff1(clear, clk, 1, 1, s);
9
10 endmodule
11 // sistema de segurança residencial
12 // alysson machado e matheus victor
```


Fonte: Imagem Autoral

**Figura 11 - Diagrama Lógico do Temporizador de 3 minutos**



Fonte: Imagem Autoral

**Figura 12 - Implementação do temporizador de 3 minutos em verilog**

```
projeto-quartus >  temporizador_3min.v
1  // modulo temporizador de 3 minutos
2  module temporizador_3min(clk, clr, segmentos, hab);
3
4      input clk;
5      input [2:0]clr;
6      output [8:0]segmentos;
7      output hab;
8
9      wire and1, and2, and3;
10     wire clr_aux1, clr_aux2, clr_aux3;
11
12     assign clr_aux1 = clr[2] | hab;
13     // gera o primeiro digito dos segundos 0-9
14     contador_decadico mod10(clk, clr_aux1, segmentos[8:5]);
15
16     assign and1 = segmentos[8] & segmentos[6];
17
18     assign clr_aux2 = clr[1] | hab;
19     // gera o segundo digito dos segundos 0-5
20     contador_modulo5 mod5(and1, clr_aux2, segmentos[4:2]);
21
22     assign and2 = segmentos[4] & segmentos[3];
23
24     assign clr_aux3 = clr[0] | hab;
25     // gera o digito do minuto 0-2
26     contador_modulo3 mod3(and2, clr_aux3, segmentos[1:0]);
27
28     assign and3 = segmentos[1] & segmentos[0];
29
30     // armazena o estado de habilitacao/desabilitacao em 2:59
31     contador_modulo2 mod2(and3, clr[0], hab);
32
33 endmodule
34 // sistema de segurança residencial
35 // alysson machado e matheus victor
```

Fonte: Imagem Autoral

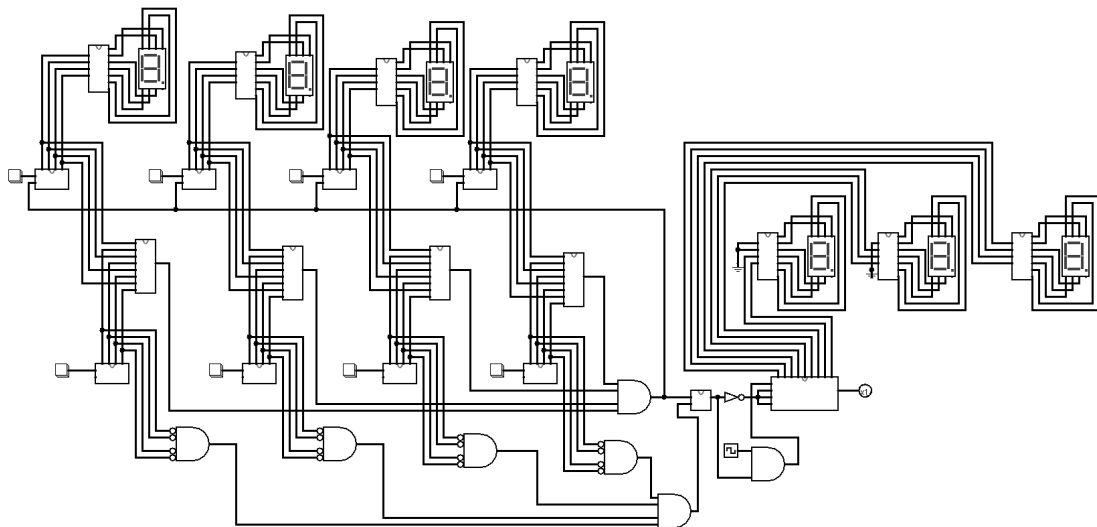
Agora que foi estabelecido todos os módulos auxiliares do sistema de senha, poderemos implementar no projeto a parte de entrada de senhas interativa para o usuário. Lembrando que:

- O usuário deverá usar um botão para digitar os 4 dígitos da senha, tendo uma visualização em decimal no display de 7 segmentos;
- Para iniciar o processo de ativação/desativação do circuito, o usuário deverá informar a senha correta, os 4 bits de entrada devem ser iguais aos 4 bits cadastrados no sistema;
- Caso a senha cadastrada no sistema seja 0000, nada deverá ser acionado, forçar para 0, utilizando o clear, no contador que registra o estado de ativação do sistema;

- Caso a senha de entrada seja correta e o circuito esteja desabilitado, antes de habilitar o circuito, haverá um temporizador de 3 minutos (com auxílio de um gerador de frequência de 3 minutos, na frequência de 4 hz);
- Caso a senha cadastrada seja correta, forçar para 0, utilizando o clear dos contadores de entrada de senha do usuário, de modo que a senha desapareça imediatamente por questões;
- Caso a senha de entrada seja correta e o circuito esteja habilitado, desabilitar imediatamente o alarme, forçando para 0, através do clear, no contador que registra o estado de ativação do sistema;

Abaixo é possível visualizar toda a parte de segurança por senha do Sistema de Alarme Residencial. Para esse fim, será apresentado Diagrama Lógico, Códigos em linguagem de descrição de hardware em verilog, Netlist do sistema e descrição das entradas e saídas, respectivamente.

**Figura 13 - Diagrama lógico do sistema de senhas para habilitação/desabilitação do sistema de segurança**



Fonte: Imagem Autoral

Figura 14 - Implementação do sistema de senhas em verilog

```
projeto-quartus > entrada_senha.v
1 // modulo de cadastro de senha
2 module entrada_senha(escolhe_digitos, senha_original, gerador_frequencia, habilitador);
3
4     input [3:0]escolhe_digitos, senha_original;
5     input gerador_frequencia;
6     output habilitador;
7
8     wire and1, and2, and3;
9     wire and_bar1, and_bar2, and_bar3, and_bar4;
10    wire [3:0]verifica_bits;
11
12    wire [6:0]segmentos1_usuario;
13    wire [3:0]digito1_usuario;
14    // entrada do usuario para escolha do digito[3] em decimal usando um binário de 4 bits
15    contador_decadico dig1_usuario(escolhe_digitos[3], and1, digito1_usuario);
16    // saída em um display de 7 segmentos do digito[3] escolhido
17    display_7segmentos dig1_display(digito1_usuario, segmentos1_usuario);
18    wire [3:0]digito1_sistema;
19    // entrada do sistema para cadastro do digito[3] em decimal usando um binário de 4 bits
20    contador_decadico dig1_sistema(senha_original[3], 1'b0, digito1_sistema);
21    // compara o digito[3] do sistema com o do usuario
22    comparador_4bits digito1(digito1_usuario, digito1_sistema, verifica_bits[3]);
23
24    wire [6:0]segmentos2_usuario;
25    wire [3:0]digito2_usuario;
26    // entrada do usuario para escolha do digito[2] em decimal usando um binário de 4 bits
27    contador_decadico dig2_usuario(escolhe_digitos[2], and1, digito2_usuario);
28    // saída em um display de 7 segmentos do digito[2] escolhido
29    display_7segmentos dig2_display(digito2_usuario, segmentos2_usuario);
30    wire [3:0]digito2_sistema;
31    // entrada do sistema para cadastro do digito[2] em decimal usando um binário de 4 bits
32    contador_decadico dig2_sistema(senha_original[2], 1'b0, digito2_sistema);
33    // compara o digito[2] do sistema com o do usuario
34    comparador_4bits digito2(digito2_usuario, digito2_sistema, verifica_bits[2]);
35
```

```
projeto-quartus > entrada_senha.v
35
36    wire [6:0]segmentos3_usuario;
37    wire [3:0]digito3_usuario;
38    // entrada do usuario para escolha do digito[1] em decimal usando um binário de 4 bits
39    contador_decadico dig3_usuario(escolhe_digitos[1], and1, digito3_usuario);
40    // saída em um display de 7 segmentos do digito[1] escolhido
41    display_7segmentos dig3_display(digito3_usuario, segmentos3_usuario);
42    wire [3:0]digito3_sistema;
43    // entrada do sistema para cadastro do digito[1] em decimal usando um binário de 4 bits
44    contador_decadico dig3_sistema(senha_original[1], 1'b0, digito3_sistema);
45    // compara o digito[1] do sistema com o do usuario
46    comparador_4bits digito3(digito3_sistema, digito3_usuario, verifica_bits[1]);
47
48    wire [6:0]segmentos4_usuario;
49    wire [3:0]digito4_usuario;
50    // entrada do usuario para escolha do digito[0] em decimal usando um binário de 4 bits
51    contador_decadico dig4_usuario(escolhe_digitos[0], and1, digito4_usuario);
52    // saída em um display de 7 segmentos do digito[0] escolhido
53    display_7segmentos dig4_display(digito4_usuario, segmentos4_usuario);
54    wire [3:0]digito4_sistema;
55    // entrada do sistema para cadastro do digito[0] em decimal usando um binário de 4 bits
56    contador_decadico dig4_sistema(senha_original[0], 1'b0, digito4_sistema);
57    // compara o digito[0] do sistema com o do usuario
58    comparador_4bits digito4(digito4_sistema, digito4_usuario, verifica_bits[0]);
59
60    assign and1 = verifica_bits[3] & verifica_bits[2] & verifica_bits[1] & verifica_bits[0];
61
62    assign and_bar1 = ~digito1_sistema[3] & ~digito1_sistema[2] & ~digito1_sistema[1] & ~digito1_sistema[0];
63    assign and_bar2 = ~digito2_sistema[3] & ~digito2_sistema[2] & ~digito2_sistema[1] & ~digito2_sistema[0];
64    assign and_bar3 = ~digito3_sistema[3] & ~digito3_sistema[2] & ~digito3_sistema[1] & ~digito3_sistema[0];
65    assign and_bar4 = ~digito4_sistema[3] & ~digito4_sistema[2] & ~digito4_sistema[1] & ~digito4_sistema[0];
66
67    assign and2 = and_bar1 & and_bar2 & and_bar3 & and_bar4;
```

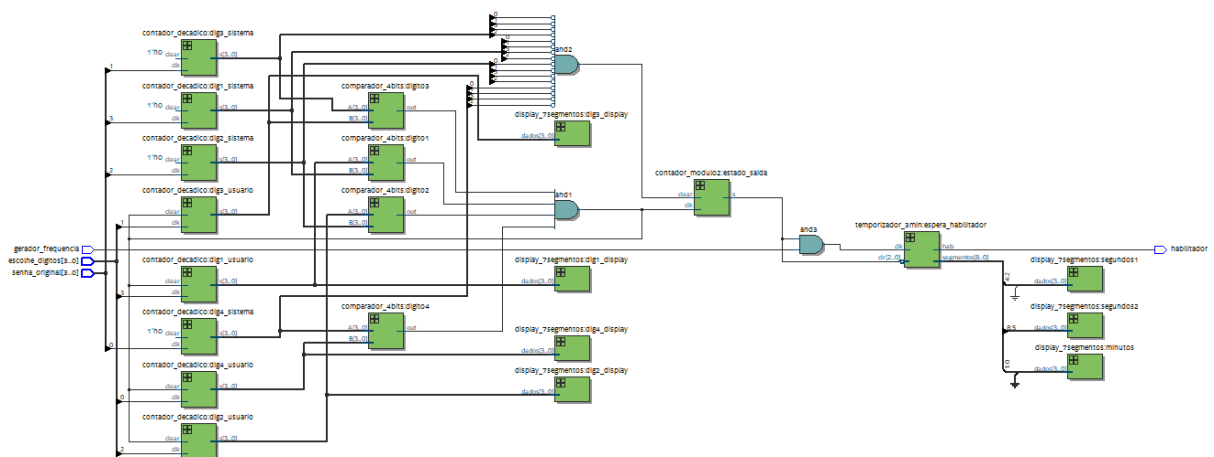
```

projeto-quartus > entrada_senha.v
68
69 wire saida_estado;
70 // memoriza o estado do circuito se esta habilitado ou nao
71 contador_modulo2 estado_saida(and1, and2, saida_estado);
72
73 assign and3 = gerador_frequencia & saida_estado;
74
75 wire [8:0]segmentos_tempo;
76 // temporizador de 3 min de espera antes de habilitar o sistema de seguranca
77 temporizador_3min espera_habilitador(and3, ~(saida_estado, saida_estado, saida_estado), segmentos_tempo,
78 habilitador);
79
80 wire [6:0]segmentos_minuto;
81 wire [6:0]segmentos_segundo1;
82 wire [6:0]segmentos_segundo2;
83 // visualiza o digito de minuto do temporizador
84 display_7segmentos minutos({1'b0, 1'b0, segmentos_tempo[1:0]}, segmentos_minuto);
85 // visualiza o primeiro digito de segundo do temporizador
86 display_7segmentos segundos1({1'b0, segmentos_tempo[4:2]}, segmentos_segundo1);
87 // visualiza o segundo digito de segundo do temporizador
88 display_7segmentos segundos2(segmentos_tempo[8:5], segmentos_segundo2);
89
90 endmodule
91 // sistema de segurança residencial
92 // alysson machado e matheus victor

```

Fonte: Imagem Autoral

**Figura 15 - Netlist do sistema de senhas**



Fonte: Imagem Autoral



### **Entradas do sistema de senha:**

- Gerador de frequência de 4hz (responsável por fazer o temporizador funcionar);
- 4 dígitos de senha do usuário, sendo cada um representado por 4 bits em binário (parte em que o usuário usará botões para escolher cada um dos dígitos através do clock em contadores decádicos síncronos);
- 4 dígitos de senha do sistema, sendo cada um representado por 4 bits em binário (essa entrada deve ser definida apenas uma vez, sendo análoga a senha que deverá ser cadastrada ao sistema para verificação);

### **Saída do sistema de senha:**

- Habilitador (ativa caso a senha seja digitada corretamente e o temporizador bater 2 minutos e 59 segundos; desativa quando o usuário digitar a senha corretamente e a saída do reabilitador estiver ativa);

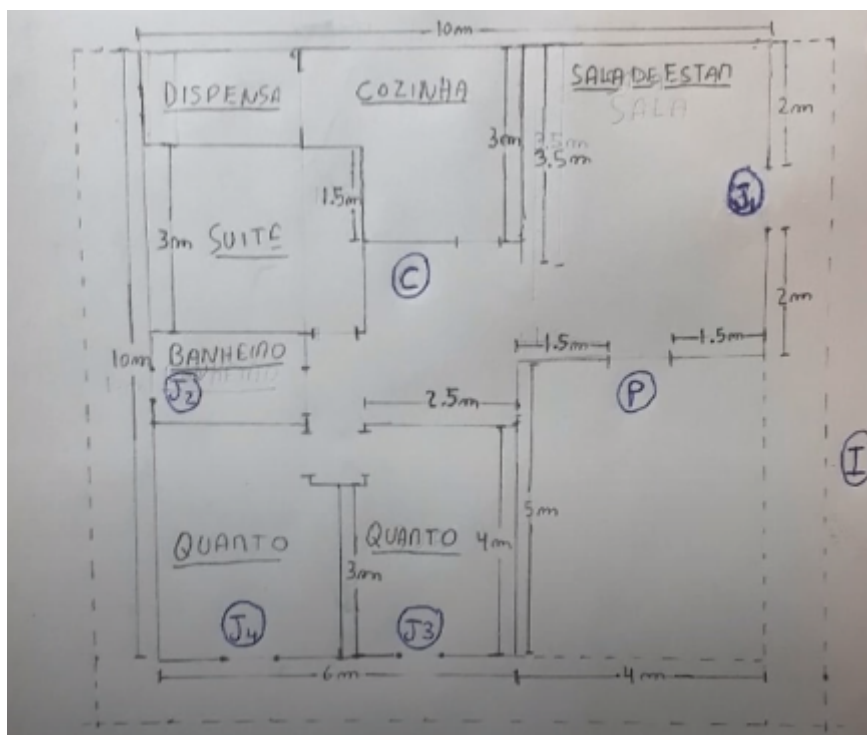
### **Parte interna do sistema de senha:**

- 4 displays de 7 segmentos que mostram ao usuário a senha de 4 dígitos fornecida;
- 3 displays de 7 segmentos que mostram a saída do temporizador, representado o dígito do minuto, e os dois dígitos do segundo;

## **4. Sistema de Controle do Alarme**

Na segunda parte precisamos integrar o sistema de senha exemplificado anteriormente no sistema de segurança geral, o sistema de senhas vai funcionar como uma chave habilitadora e teremos também os sensores integrados ao sistema, teremos um vetor J com quatro posições que representa os sensores nas janelas da casa, teremos a entrada P que representa o sensor que está na porta e teremos o disparo da cerca elétrica representado por I, para entender melhor confira a imagem da planta da casa a seguir.

**Figura 16 - Planta baixa da residência**



Fonte: Imagem Autoral

O próximo passo é elaborar a tabela verdade baseado nos seguintes fatos.

- Ter uma entrada para definir o tipo de segurança, se é segurança máxima ou mínima
- Se for segurança máxima, elaborar um sistema que, quando estiver ligado o sistema se qualquer sensor de janelas, portas e cerca forem acionados disparar imediatamente o alarme, o único caso que de gerar ser isento do acionamento é o caso em que a cerca elétrica for "disparada" com os sensores sem detectar imprudência, para o caso de ser algum animal andando no muro.
- Para o caso da segurança ser mínima, temos que soa cerca elétrica deve estar monitorando, para o caso das pessoas da casa queiram utilizar as janelas e porta, observação nesse caso com só a cerca funcionando o caso de animal no muro não deve se levar em consideração, para que não tenha como burlar o sistema.

**Figura 17 - Tabela verdade do sistema de acionamento do alarme**

C	Y	I	J1	J2	J3	J4	P	A
0	X	X	X	X	X	X	X	0
1	0	0	X	X	X	X	X	0
1	0	1	X	X	X	X	X	1
1	1	0	0	0	0	0	0	0
1	1	0	1	X	X	X	X	1
1	1	0	X	1	X	X	X	1
1	1	0	X	X	1	X	X	1
1	1	0	X	X	X	1	X	1
1	1	0	X	X	X	X	1	1
1	1	1	0	0	0	0	0	0
1	1	1	1	X	X	X	X	1
1	1	1	X	1	X	X	X	1
1	1	1	X	X	1	X	X	1
1	1	1	X	X	X	1	X	1
1	1	1	X	X	X	X	1	1

Fonte: Imagem Autoral

O próximo passo é a implementação no Logisim, fazendo o mapa de Karnaugh, ficamos com a expressão abaixo:

$$A = C (\sim Y) I + C Y P + C Y J4 + C Y J3 + C Y J2 + C Y J1$$

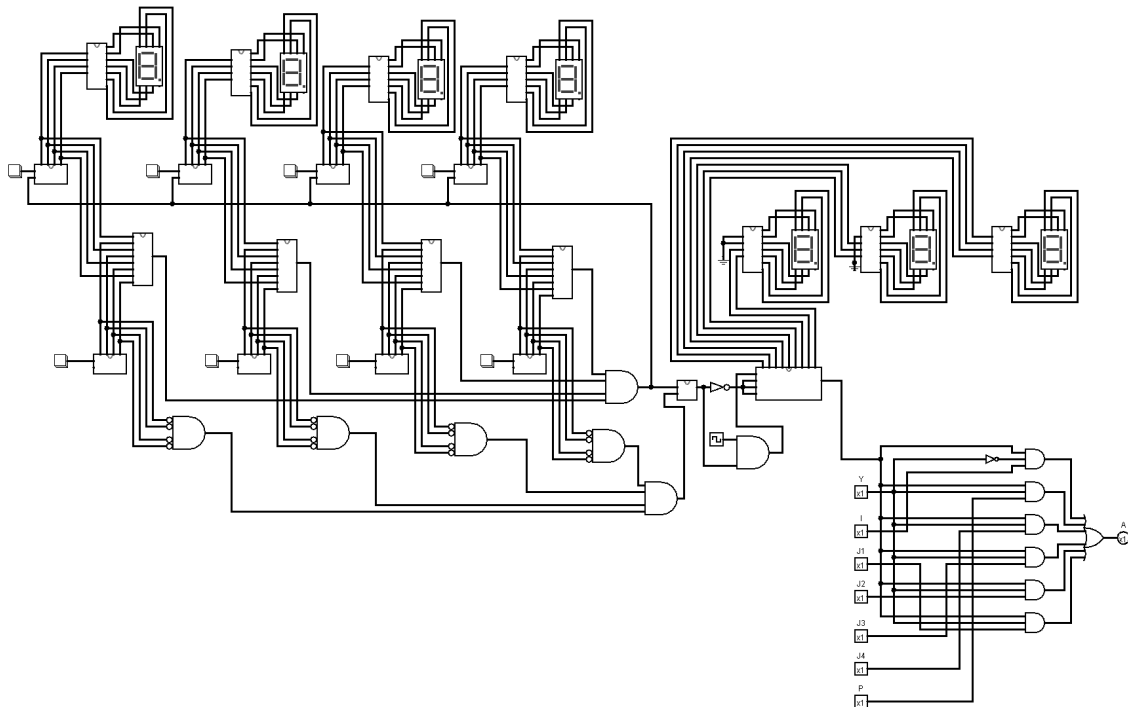
A análise das expressões lógicas acima possui as seguintes legendas:

- **A** = Acionamento do alarme;
- **C** = Estado de habilitação do alarme;
- **Y** = Modo de segurança baixa/alta;
- **I** = Sensor de violação da cerca elétrica;
- **P** = Sensor de violação da porta;
- **J[4:1]** = Sensores de violação das janelas;

A seguir temos a imagem da implementação no Logisim, no formato de diagrama lógico.



**Figura 19 - Diagrama lógico do projeto inteiro**



Fonte: Imagem Autoral

E por fim temos a implementação no Quartus, usando a linguagem de descrição de hardware verilog.

**Figura 20 - Implementação em verilog do sistema de acionamento de alarme**

```
projeto-quartus > ssr.v
1 // modulo do projeto de sistema de segurança residencial completo
2 module ssr
3 (
4     input [4:1]j,
5     input p, i, y,
6     input [3:0]senha_sistema, senha_usuario,
7     input gerador_frequencia,
8     output alarme
9 );
10
11
12     wire habilitador;
13     wire [5:0]fio;
14     wire and1, or1;
15
16     // instanciação do modulo de entrada de senha
17     entrada_senha es_ssr(senha_usuario, senha_sistema, gerador_frequencia, habilitador);
18
19     // verifica as possibilidades de violação da residencial
20     and_3entradas Z1(i,~y,habilitador,fio[0]);
21     and_3entradas Z2(p,y,habilitador,fio[1]);
22     and_3entradas Z3(j[1],y,habilitador,fio[2]);
23     and_3entradas Z4(j[2],y,habilitador,fio[3]);
24     and_3entradas Z5(j[3],y,habilitador,fio[4]);
25     and_3entradas Z6(j[4],y,habilitador,fio[5]);
26
27     // verifica se pelo menos uma das entradas da casa foi violada
28     or_6entradas K(fio[0],fio[1],fio[2],fio[3],fio[4],fio[5],alarme);
29
30 endmodule
31 // sistema de segurança residencial
32 // alysson machado e matheus victor
```

Fonte: Imagem Autoral

## **5. Melhorias e Dificuldades**

A principal dificuldade ao desenvolver esse projeto foi o não contato com placas FPGA para o desenvolvimento do projeto, dado que o mesmo foi desenvolvido em meio a uma pandemia. Entretanto, a implementação de simuladores na atualidade facilitou bastante o desenvolvimento de projetos a nível de linguagens de descrição de hardware. As possíveis melhorias que podem ser feitas podem surgir devido às dificuldades de uso e ao feedback dos primeiros usuários que utilizarem o projeto, sendo problemas que devem ser resolvidos através de atualizações constantes do projeto.

## **6. Referências**

- Sistemas Digitais: princípios e aplicações / Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss ; revisão técnica Renato Giacomini ; tradução Jorge Ritter. - 11. ed. - São Paulo : Pearson Prentice Hall, 2011.