

Universidade de São Paulo

Escola Politécnica, Engenharia de Computação
PCS3645 - Laboratório Digital II
Turma 3 - Professor Paulo Cugnasca
Bancada B6



DoseMinder

Semana 1.0 do projeto - Relatório

Nome Completo	N USP
Henrique Freire da Silva	12555551
Mariana Dutra Diniz Costa	12550841

São Paulo, 07 de Novembro de 2023

1 PROPOSTA INICIAL

Em prescrições de uso contínuo de fármacos, é natural que possa se esquecer de ingerir o medicamento correto na hora adequada. Essa condição, porém, tende a ser problemática para um público com condições especiais, tal como idosos e indivíduos que sofrem do mal de Alzheimer.

Propõe-se, então, o desenvolvimento de um dispensador de medicamentos para controle de receita prescrita através de interface por aplicativo mobile e comunicação por protocolo MQTT. Dessa forma, poderia se facilitar a descarga de pílulas/comprimidos sem depender unicamente da recordação do cronograma por parte do paciente, bem como de sua capacidade de distinguir os medicamentos que deve ingerir. Os principais componentes que descrevem o sistema são como segue:

1. Interface gráfica desenvolvida em Flutter;
2. Sensor ultrassônico HC-SR04;
3. Micro servomotor;
4. ESP32;
5. Botão;
6. LEDs e/ou buzzer ativo;
7. LED IR (emissor infravermelho);
8. Fototransistor IR.

De acordo com os recursos de tempo e execução disponíveis, alguns requisitos da ideia de projeto inicial foram restringidos. Ao acionar o sistema manualmente via interface, o usuário define quantos comprimidos precisa tomar. Uma notificação é enviada ao dispositivo do usuário e uma mensagem é publicada no respectivo tópico MQTT. Com isso, a informação estruturada e agora contida na ESP32 é serializada e enviada à FPGA para registro das dosagens de cada fármaco disponível. O escopo do protótipo foi definido para dispor de um tipo de fármaco, no entanto o circuito será desenvolvido de forma generalizada, para que seja possível expandi-lo em outro momento.

Logo após o registro da dosagem e a verificação da presença de medicamento no reservatório, há o acionamento de sinais sonoros e visuais via LEDs e buzzers da maquete, para indicar ao usuário que o circuito está esperando para liberar o medicamento.

O sistema espera, durante certo tempo de timeout, pela proximidade de um recipiente ao sensor ultrassônico e, quando essa aproximação é efetivada, o servomotor responsável pelo reservatório é, então, reposicionado de forma a empurrar o próximo comprimido da fila. Esse processo é repetido até que a dose completa seja despejada. Da mesma forma, caso seja removido o recipiente de forma precipitada, deve-se interromper o processo de dispensa e avisar ao usuário que ainda há remédios a se coletar. O fluxo descrito pode ser verificado nas interconexões apresentadas na Figura 1.

O conteúdo dos reservatórios será organizado em pilha, sendo adequado para um determinado intervalo de tamanho de comprimido e permitindo que o servomotor "empurre" apenas um por vez. Este conteúdo será sempre monitorado via sensor de presença infravermelho para que, quando se chegue ao fim da disponibilidade de um fármaco, seja acionada uma notificação para que o usuário recarregue o dispositivo de forma correta. O sistema ficará travado e voltará a funcionar apenas após a realização da recarga.

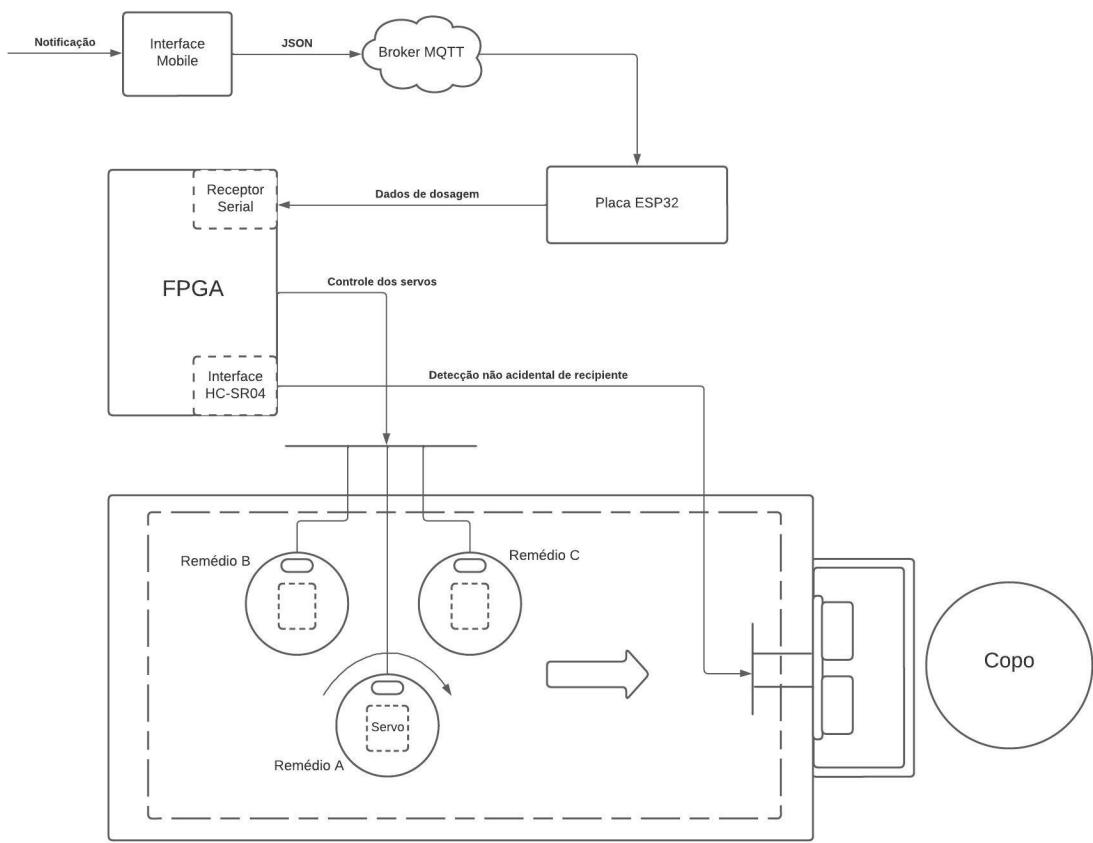


Figura 1: Diagrama de blocos de alto nível.

1.1 REQUISITOS

1.1.1 Requisitos Funcionais

1. Despejo do medicamento quando o sistema for acionado.
2. Capacidade do dispositivo para 1 tipo de medicamento.
3. Sinalização ao usuário no momento de despejo.
4. Interface de monitoramento via *software*.
5. Protótipo em maquete do produto final.
6. Acionamento manual do sistema pelo aplicativo com definição da dosagem.

1.1.2 Requisitos Não Funcionais

1. Usabilidade da Interface.
2. Tratamento da indisponibilidade do medicamento no dispenser.
3. Tempo de tolerância de até 3 minutos entre o acionamento e o despejo da dose de medicamento.
4. Desenvolvimento descentralizado e integração estável entre os componentes.

2 INTRODUÇÃO

No presente relato, será apresentado o trabalho planejado para a primeira semana do desenvolvimento do projeto. Nesse momento, será realizada a implementação de requisitos obrigatórios, iniciando o projeto do circuito lógico em VHDL para o sistema e a conexão de alguns dos componentes físicos. Nessa etapa, portanto, será realizada a comunicação via porta serial para controle de dosagem e a movimentação do atuador servomotor para o despejo do medicamento.

Com o objetivo de síntese e desenvolvimento integrado desse sistema, as seguintes seções compreendem uma etapa lógica do princípio de funcionamento do circuito, seguida da aplicação em VHDL, simulação com o *ModelSim*, síntese com o *Quartus Prime* e, finalmente, verificação por meio de testes práticos.

2.1 Requisitos Funcionais

Os requisitos funcionais obrigatórios que serão implementados nesta semana são os seguintes:

1. Despejo do medicamento quando o sistema for acionado.
2. Capacidade do dispositivo para 1 tipo de medicamento.
3. Sinalização ao usuário no momento de despejo.

É importante destacar que o despejo do medicamento será verificado pela movimentação do atuador, enquanto a maquete ainda não é desenvolvida, e que o acionamento do sistema será representado via comunicação serial pelo terminal TeraTerm, enquanto não há a interface em software.

2.2 Requisitos Não Funcionais

A arquitetura inicial proposta deve descrever um modelo estrutural com módulos de fluxo de dados e unidade de controle. O circuito parcial deve ser de fácil depuração, em que há sinais duplicados para verificação por osciloscópio de sinais emitidos e recebidos. O envio serial deve ser codificado em ASCII e o projeto deve ser descrito em VHDL e sintetizado pelo *Quartus Prime* para a placa DE0-CV.

3 DESCRIÇÃO DO PROJETO

O projeto discorrido nas seguintes seções descreve as conexões expostas pelo circuito que combina a transmissão serial e o servomotor, bem como a lógica adotada em etapas de processamento entre os componentes mencionados e demais fluxos para a construção parcial do sistema de dispenser de medicamentos.

3.1 Projeto do circuito

A entidade principal do sistema desenvolvido, de nome *pill_dispenser.vhd*, contém em sua arquitetura a interconexão entre sua unidade de controle, seu o fluxo de dados e demais portas de interface declaradas, além de sinais de depuração (Apêndice B.1).

A entrada "serial" é recebida via porta serial para definir a dosagem e a saída "pwm" provém do circuito controlador do atuador de forma a reger sua posição. Os sinais "trigger", "echo" e "alert" são deixados para a interface com o sensor ultrassônico, que ainda não faz parte do escopo de desenvolvimento para esta semana.

O projeto dispõe de um pacote *pill_package* que declara os componentes reutilizados nos módulos do design e a tipagem que modela o tema genericamente, tal que possa se centralizar mudanças que afetarão diversos arquivos. O pacote é tal como o Apêndice A.

O conjunto "db_switch" faz a seleção do sinais de depuração de estado dos módulos internos através de multiplexação, conforme C.3, em que "s_state" designa o estado da unidade de controle de cada componente. Os sinais de depuração apresentados a partir da linha 12 de B.1 são conectados diretamente a portas de módulos internos ao fluxo de dados.

3.1.1 Unidade de controle

A unidade de controle desenvolvida reage, inicialmente, ao recebimento serial pelo port "serial", aguardando um tempo definido (C.3, linha 2) para garantir que o recipiente permanece no lugar, porém com um *delay* incondicional até posterior implementação da interface de sensor; e persistindo em um ciclo de acionamento do servo, aguardo de um tempo de segurança (C.3, linha 4, em ciclos de *clock*), retorno para a posição de repouso do servo e decremento de registrador(es) aplicável(is) do fluxo de dados (discorridos na seção 3.1.2). A entidade desse módulo segue o Apêndice B.2.

A transição de estados de execução segue o algoritmo seguinte, descrito em linguagem natural:

```
1 Aguarda detecção de dosagem por DETECTED;
2 Enquanto DETECTED alto :
3     Espera pelo CHECK_TIMEOUT;
4     Avanca o motor para despejo do remedio pelo sinal MOVE
5     Espera pelo SAFETY_TIMEOUT;
6     Retorna o motor a posição inicial e levanta DISCOUNT;
```

Esse mesmo comportamento pode ser verificado na arquitetura apresentada no Apêndice C.2 entre as linhas 19 e 39, em que é descrita a lógica de transições e as saídas de cada estado (máquina de Moore).

O sinal de contagem (*count*, C.2, linha 44) está ativo em 3 estados do ciclo de execução: *check*, *dispense* e *await*. O compartilhamento desse sinal rege a contagem dos dois temporizadores utilizados no fluxo de dados e, para os estados vizinhos (*dispense* e *await*) a contagem do contador SAFETY_TIMER (Apêndice C.3, linhas 17-26) é repetida uma vez ao se concatenar ciclos de contagem com o fim circular sem desativar o sinal de contagem.

3.1.2 Fluxo de dados

O Fluxo de Dados (Apêndice B.3) realiza a integração entre diferentes módulos, que consistem atualmente no controle de pwm, a recepção serial, dois contadores de tempo e um contador regressivo de garantia da dosagem correta.

A transmissão serial foi padronizada no protocolo 7O1, onde os 3 bits menos significativos representam a quantidade de comprimidos que deve ser liberada naquela dose, os próximos 4 identificam o container de remédios e, por fim, um último bit de paridade ímpar. É importante salientar que a identificação de container foi uma decisão de projeto para generalizar o sistema e possibilitar expansão, porém, o escopo só abordará um único reservatório de medicamento.

Ao receber a palavra serial pelo sinal *s dosage*, verifica-se o container definido e, caso seja 0 (o único disponível nesse escopo), o contador inverso *downwards_counter* é habilitado. Seu objetivo é realizar uma contagem regressiva, partindo dos 3 bits menos significativos recebidos até o zero. A realização de uma contagem, ou seja, o decremento de um valor, é indicado pelo sinal de controle *discount* vindo da Unidade de Controle no momento oportuno do ciclo. Assim que a contagem chega ao fim, um sinal de alerta é ativado e enviado para a Unidade de Controle pela interface principal do circuito.

O circuito de pwm, definido para gerar pulsos de 4 diferentes larguras, foi instanciado e seu parâmetro *width* foi modulado de forma a assumir sempre um de dois valores: 0 ou 75000 (representativo de um pulso de 1500 ns). Sendo assim, ao receber o sinal de controle *move*, o servomotor é movimentado, empurrando consigo a pílula que deve ser despejada. Da mesma forma, quando o despejo é finalizado, ele retorna para a posição inicial até que o próximo comprimido precise ser empurrado.

Por fim, os dois contadores *check_timer* e *safety_timer* ficam ativos durante os estados de "check", "dispense" e "await" e são garantidores da segurança do sistema. O primeiro deles, que será posto à prova nas semanas subsequentes, está relacionado ao sensor ultrassônico e existe para garantir que a detecção de um recipiente à espera de medicamento foi efetivamente realizada durante 2 segundos, tempo considerado suficiente para realizar essa verificação com estabilidade. Já o temporizador de segurança conta 0,5 segundo e assegura que o atuador não correrá risco de ser forçado por instantes curtos demais, sendo sempre ativado por um tempo mínimo de funcionamento.

3.2 Estratégia de testes

O *testbench* do circuito de interface foi configurado para adotar os casos da Tabela 1. Para cada iteração rodada, a largura de pulso é definida de acordo com a segunda coluna da tabela.

Com a simulação do envio do sinal serial, o circuito deve realizar a movimentação do atuador, ou seja, enviar o pulso pwm de acordo com a quantidade de vezes requisitada. Isso só deve ocorrer, porém, quando o container selecionado for o "0", ou seja, o único atualmente disponível. A nível de teste laboratorial, a estratégia será verificar efetivamente o movimento do servomotor para cada reprodução desses envios via terminal TeraTerm.

Teste	Recepção Serial (big endian)	Largura de pulso (ns)	Ciclos de <i>dispense</i>
1	10000000	1500	1
2	11000001	1500	3
3	11101001	0	0
4	01100001	1500	6

Tabela 1: Casos de teste definidos no *testbench*.

3.3 Simulação e Síntese do Design

Depois de definidos os casos de testes em *testbench*, conforme apresentados na Tabela 1, o projeto foi integrado ao *software ModelSim* para fins da simulação final e geral. Desse modo, foi possível simular os cenários propostos e acompanhar o comportamento de cada sinal de entrada, saída e depuração ao longo do funcionamento do circuito.

É importante ressaltar que, apenas para fins de simulação, os tempos de *check_timeout*, *safety_timeout*, foram alterados para otimizar o processo de testagem e serão reconfigurados para os testes laboratoriais. No entanto, não foi possível adequar o período e a largura do pulso PWM para parâmetros tangíveis de simulação, que conta então com o sinal "move" para verificação. A movimentação do atuador será, portanto, testada em laboratório.

Foram realizados, portanto, 4 casos de teste, estando o primeiro representado na Figura 2. Nele, inicia-se recebendo uma entrada serialmente e, assim que a leitura é finalizada, o circuito entra no estado "check", enquanto o sinal de dosagem e o contador de pílula são configurados inicialmente. Depois que ocorre o timeout desse estado, inicia-se o "dispense", em que o sinal de "move" é ativado e, assim que termina o "safety timeout", a quantidade de pílulas é descontada em um por meio da ativação do sinal de controle "discount". Assim, com a contagem regressiva atingindo zero, nenhuma pílula mais deve ser liberada e o circuito entra em "idle" até que receba uma nova transmissão serial, já no caso de teste seguinte. É importante perceber, também, o sinal de saída "alert" que, além de indicar para o usuário que o circuito está pronto para liberar um comprimido, ajudará no controle do funcionamento do sensor ultrassônico.

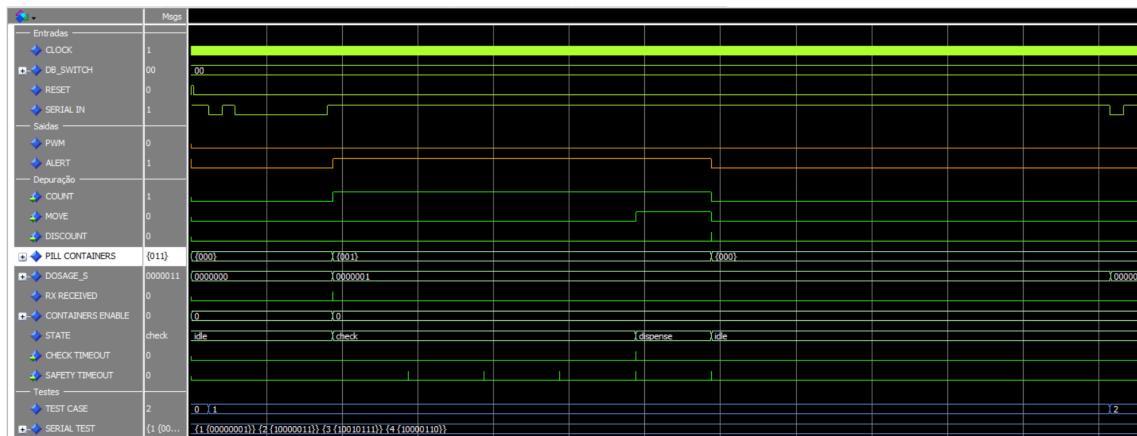


Figura 2: Formas de onda da simulação do circuito para o caso de liberar 1 pílula.

Os 3 casos de teste seguintes, dispostos entre as Figuras 3 e 5, seguem as conformidades do primeiro, registrando sempre o sinal recebido serialmente e contando regressivamente a liberação de cada comprimido, enquanto se completa os ciclos dos estados de "check", "dispense" e "await". É importante destacar que, na Figura 4, a entrada serial seleciona um reservatório inexistente, visto que o projeto abrange apenas o dispensador 0. É por isso que, nesse caso, o circuito se comporta de modo a não movimentar o atuador, não acionar sinais de atenção e não dispensar nenhum comprimido.

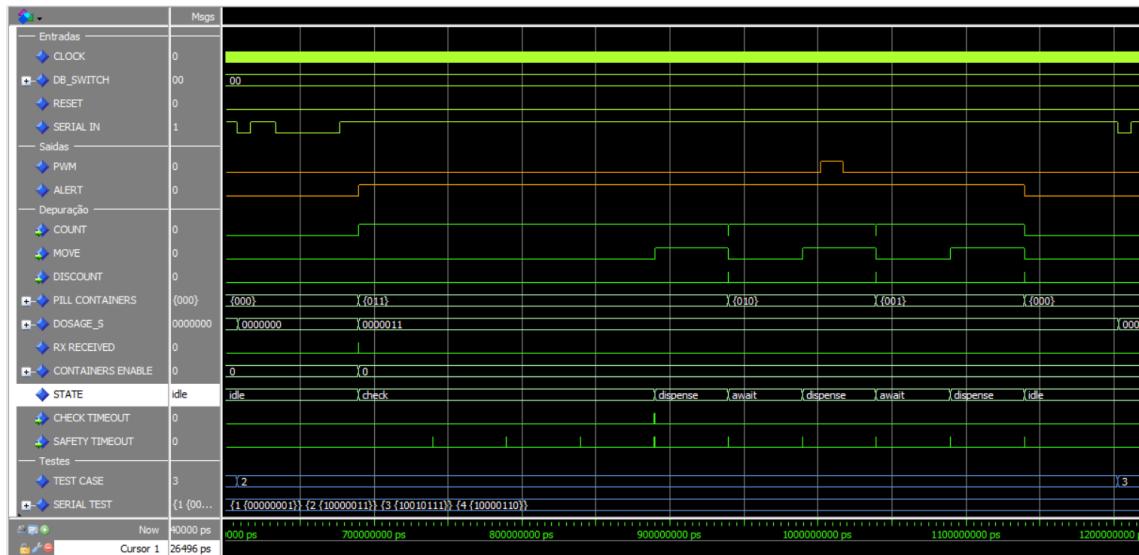


Figura 3: Formas de onda da simulação do circuito para o caso de liberar 3 pílulas.

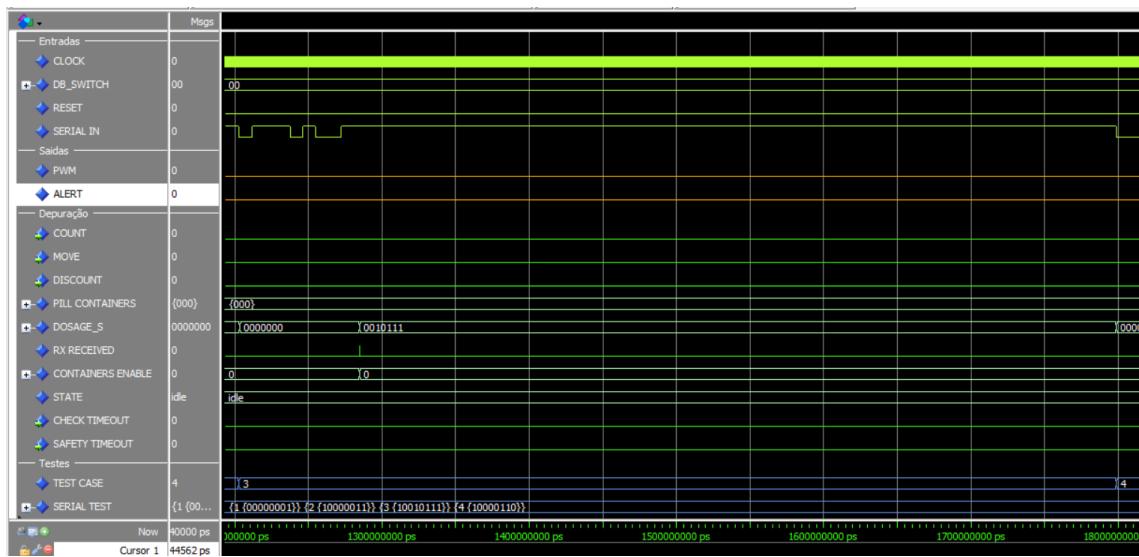


Figura 4: Formas de onda da simulação do circuito para o caso de selecionar um dispenser inexistente.

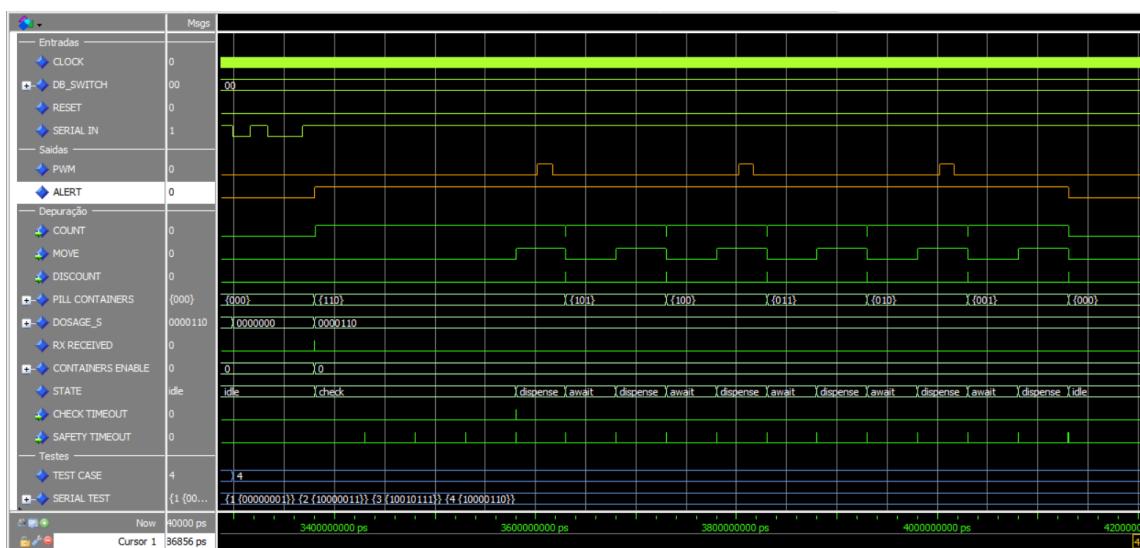


Figura 5: Formas de onda da simulação do circuito para o caso de liberar 6 pílulas.

Após realizar as simulações e confirmar, de fato, o bom funcionamento do circuito projetado para a integração do servomotor com a porta serial, os módulos do projeto foram importados para um novo arquivo na ferramenta *Intel Quartus Prime*, onde foi possível analisar cada componente e sintetizar o circuito final.

Partindo dessa síntese, foram criadas visualizações por meio da ferramenta *RTL viewer*, através da qual foi possível verificar os diagramas de bloco do circuito e confirmar sua compatibilidade com as especificações definidas no capítulo 3. Nesse sentido, foram ilustrados o detalhamento de implementação do novo componente contador regressivo (Figura 6), os componentes formadores do Fluxo de Dados (Figura 7) e o circuito completo do dispensador (Figura 8).

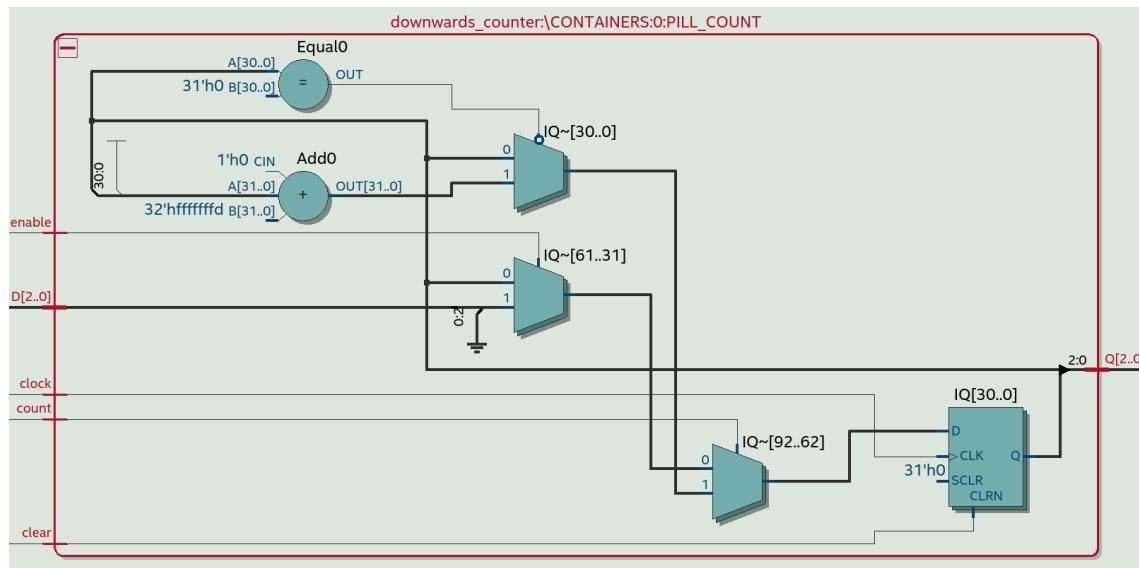


Figura 6: Diagrama de blocos RTL do novo componente "downwards counter".

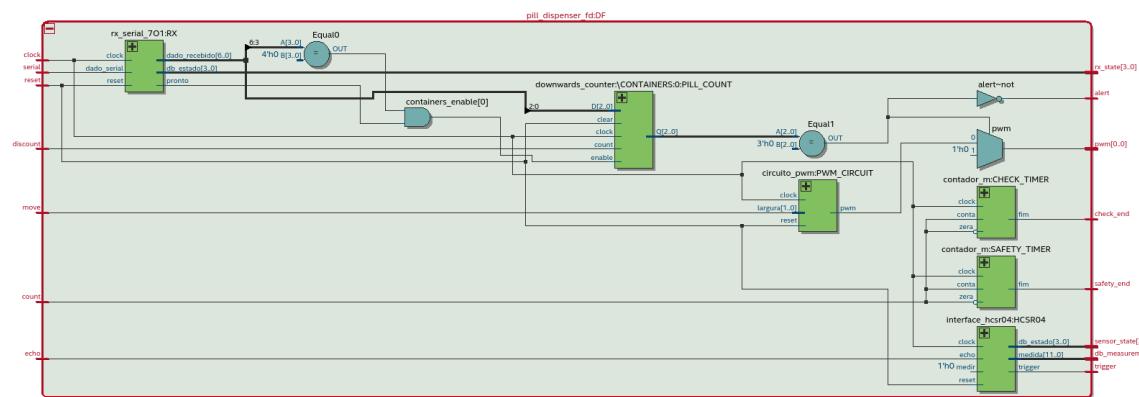


Figura 7: Diagrama de blocos RTL para o fluxo de dados do circuito.

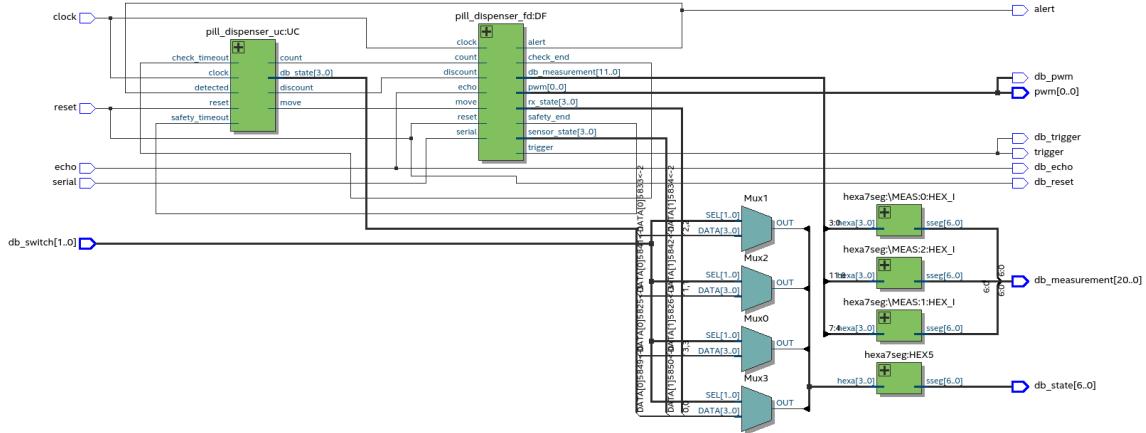


Figura 8: Diagrama de blocos RTL para a entidade principal do circuito.

Com isso, a ferramenta interna *State Machine Viewer* foi utilizada para conferir a estrutura da máquina de estados que rege a Unidade de Controle do projeto, assim como as transições entre estados e os sinais de condição que influenciam no seu funcionamento. Esses aspectos estão representados na Figura 9.

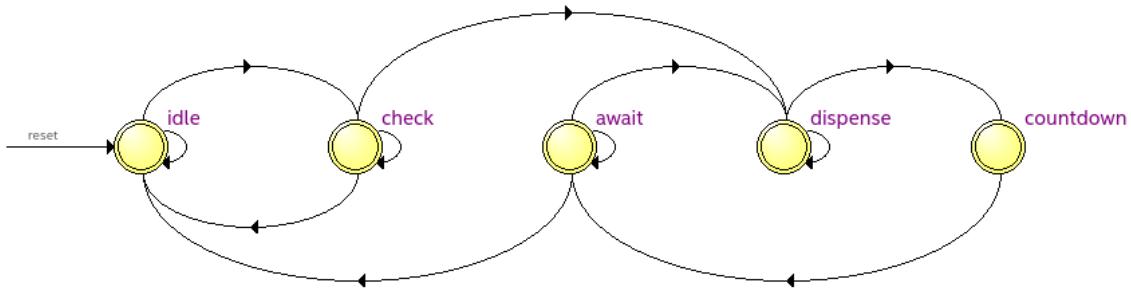


Figura 9: Diagrama da máquina de estados que rege a Unidade de Controle do circuito.

Finalmente, foi configurada a pinagem conforme a Tabela 2 descrita na Seção 4 e o projeto foi exportado para um arquivo de extensão *qar* que poderá ser diretamente utilizado em laboratório para elaboração na placa FPGA.

4 PLANEJAMENTO DA AULA PRÁTICA

Em laboratório, será realizada a síntese do projeto arquivado (extensão *qar*) na placa FPGA DE0-CV segundo a pinagem da Tabela 2.

A montagem experimental será, então, realizada em etapas intermediárias para testar o correto funcionamento de todos os componentes do projeto e suas relações. Esse processo seguirá de acordo com o aqui descrito Plano de Execução Experimental.

4.1 Plano de Execução Experimental

Inicialmente, será testada a comunicação pela porta serial - USB, conectando o GND, TX e RX nos pinos correspondentes da placa MAX3232 e deixando em curto os terminais TD e RD. Do outro lado, a porta USB será conectada à entrada devida no computador. Com isso, a ferramenta *TeraTerm* será configurada de acordo com a codificação 7O1 utilizada e serão realizados testes de eco ao digitar um caractere ASCII no terminal, verificando o correto retorno dele pela porta.

Em seguida, será realizada a integração com o módulo do servomotor acoplado ao sensor ultrassônico. Para isso, ele será apropriadamente alimentado pelos terminais VCC e GND da placa FPGA, além de conectado ao sinal de saída *pwm* do circuito de controle do sonar por meio dos acessos GPIO. É importante recordar, desde o início, de referenciar todos os componentes experimentais a um mesmo potencial terra (GND).

A partir desse momento, a placa será ligada e programada com o projeto em questão para a realização dos testes propostos, sempre de forma a se atentar aos sinais de depuração definidos. Além da depuração definida na placa, ainda há os pinos GPIO que apresentam a possibilidade de serem conectados ao Analog Discovery para verificação das ondas, que pode ser bem utilizada na depuração da recepção serial, por exemplo.

Sinal	Ligaçāo na placa FPGA	Pino na FPGA	Analog Discovery
clock	CLOCK_50	PIN_M9	-
reset	chave SW0	PIN_U13	-
serial	GPIO_0_D1	PIN_B16	-
pwm	GPIO_0_D35	PIN_T15	-
alert	led LEDR[0]	PIN_AA2	-
db_switch	chaves SW8 e SW9	PIN_AB13 PIN_AB12	-
db_state	display HEX5	PIN_N9 PIN_M8 PIN_T14 PIN_P14 PIN_C1 PIN_C2 PIN_W19	-
db_reset	led LEDR[3]	PIN_Y3	-
db_pwm	GPIO_1_D11	PIN_J18	CH1+ (Scope)
db_partida	led LEDR[2]	PIN_W2	-

Tabela 2: Pinagem para a montagem experimental.

5 ATIVIDADES EXPERIMENTAIS

Em laboratório, o circuito final do projeto foi sintetizado e programado na placa FPGA DE0-CV. A montagem foi realizada em etapas intermediárias, seguindo o que havia sido anteriormente definido no Plano de Execução Experimental. Primeiramente, foi realizada a montagem intermediária para testar a comunicação serial. Nesse sentido, o pino GND foi conectado à referência de Terra da FPGA, os terminais RD e TD da placa MAX3232 foram colocados em curto e a porta serial foi conectada à entrada USB computador. Com isso, o terminal TeraTerm foi utilizado e com ele verificada a transmissão de caracteres via serial no protocolo 7O1. Com isso, foi possível conectar o terminal TD ao pino GPIO do sinal *serial* de entrada do circuito para possibilitar, enfim, a transmissão da dosagem requisitada.

O próximo passo foi integrar o módulo do servomotor, por meio da conexão de seus terminais GND e VCC (5V), além do sinal de controle *pwm* vindos da GPIO. Com isso, foi realizada a integração dos dois módulos e finalizada a montagem do sistema atual, conforme apresentado na Figura 10. Nela, o display hexadecimal mais à esquerda reflete o estado do componente principal do circuito (o que pode ser multiplexado pelas chaves SW9 e SW8) e o LED0, mais à direita, foi utilizado como saída do sinal de alerta.

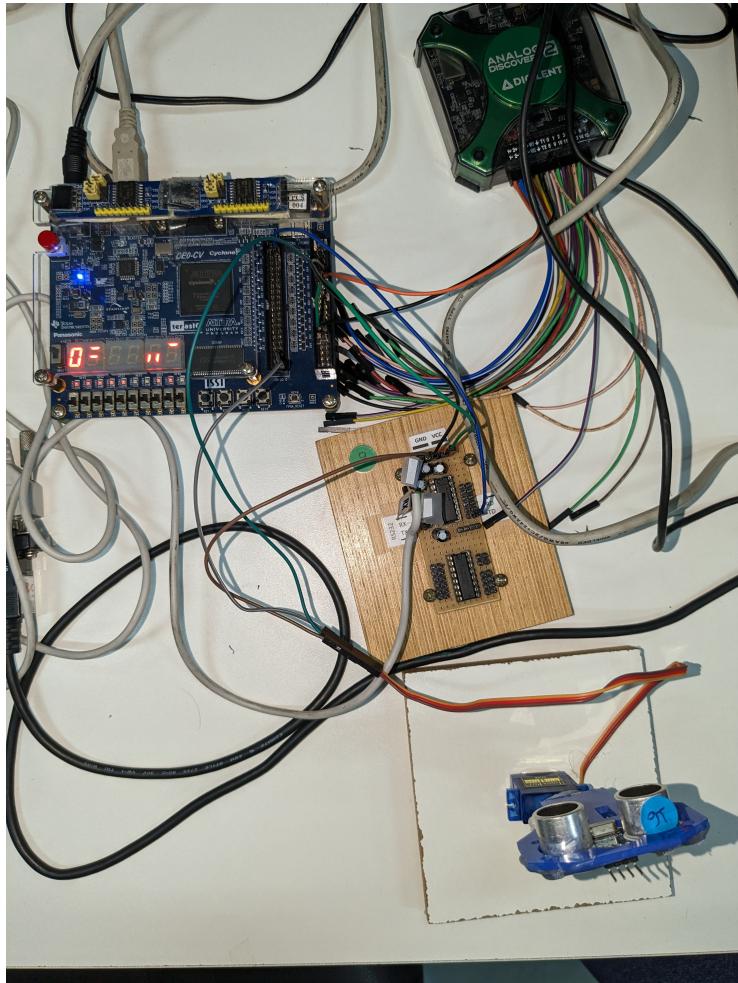


Figura 10: Montagem experimental do sistema atual.

Nesse momento seriam, então, iniciados os testes propostos na Tabela 1. No entanto, ao codificar os valores seriais dos testes para ASCII, percebeu-se que se tratavam de caracteres especiais, o que dificultou o envio por meio do teclado. Por isso, foi feita uma adequação temporária no circuito, configurando o "dispenser" padrão (único existente) com o valor 14 ("1110") em vez de 0 ("0000"). Dessa forma, os casos de teste foram alterados (Tabela 3) de modo a abarcarem o novo valor de identificação do dispenser, assim possibilitando o trabalho com caracteres ASCII entre "q" e "w".

Teste	Recepção Serial (big endian)	Caractere	Largura de pulso (ns)	Ciclos de <i>dispense</i>
1	10001110	q	1500	1
2	11001111	s	1500	3
3	11101010	W	0	0
4	01101111	v	1500	6

Tabela 3: Casos de testes experimentais para envio de dosagens.

Com esses novos valores de teste, adequados à restrição dos caracteres, foi possível verificar o bom funcionamento do circuito projetado por meio da movimentação de ida e volta do servomotor na quantidade exata de vezes determinada pela entrada serial. Este movimento representa, por fim, o ato de empurrar uma pílula para a saída do dispenser e retornar para a posição inicial, seja para empurrar uma nova ou finalizar o processo daquela dose. Além disso, a indicação da saída de alerta no LED0 ocorreu conforme esperada, permanecendo ativa até que o despejo daquela requisição seja finalizado.

Como depuração, foram utilizados o display HEX5 para o estado e o canal 1 do osciloscópio do Analog Discovery para conferir o correto envio do sinal de PWM, conforme representa a Figura 11 a seguir.

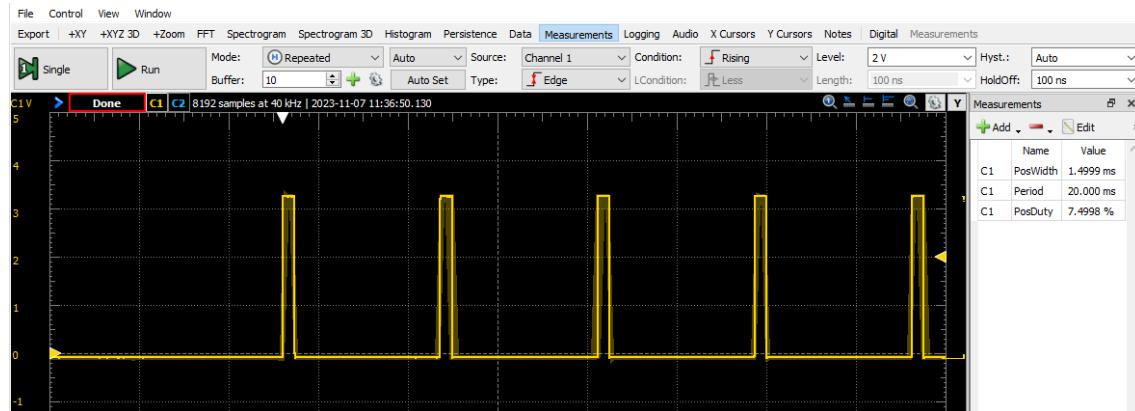


Figura 11: Verificação do envio e da largura do sinal PWM no osciloscópio.

6 CONCLUSÃO

A montagem e depuração em etapas graduais do projeto aqui discorrido, tal como o uso de osciloscópio pela ferramenta *Scope* na verificação da largura de pulso, bem como o teste unitário dos principais componentes do circuito: servomotor e barramento serial; auxiliou na validação do circuito desenvolvido e correções de falhas não previstas em simulação.

Além disso, para a primeira semana de desenvolvimento do projeto, percebeu-se a extrema importância de se realizar um planejamento completo e detalhado, tendo sido definidos desde o princípio os objetivos intermediários e os requisitos que devem ser atingidos naquela prática de laboratório. Por fim, foi muito relevante compreender a viabilidade dos testes práticos dentro dos recursos disponíveis na bancada e encontrar flexibilidade na readequação dos testes a serem realizados, quando necessário.

APÊNDICES

A Pacote do projeto

```
1 package pill_package is
2     constant CONTAINERS : natural range 1 to 16 := 1;
3
4     subtype containers_range is natural range 0 to CONTAINERS-1;
5     type pill_count is array (containers_range) of
6         std_logic_vector(2 downto 0);
7
8     component downwards_counter is
9         generic (constant N: integer);
10        port (
11             clock   : in  std_logic;
12             clear   : in  std_logic;
13             count   : in  std_logic;
14             enable  : in  std_logic;
15             D       : in  std_logic_vector(N-1 downto 0);
16             Q       : out std_logic_vector(N-1 downto 0)
17         );
18     end component downwards_counter;
19
20     component circuito_pwm is
21         generic (
22             conf_período          : integer;
23             largura_00, largura_01 : integer;
24             largura_10, largura_11 : integer
25         );
26         port (
27             clock    : in  std_logic;
28             reset    : in  std_logic;
29             largura : in  std_logic_vector(1 downto 0);
30             pwm      : out std_logic
31         );
32     end component circuito_pwm;
33
34     component contador_m is
35         generic (constant M, N : integer);
36         port (
37             clock : in  std_logic;
38             zera  : in  std_logic;
39             conta : in  std_logic;
40             Q     : out std_logic_vector(N-1 downto 0);
41             fim   : out std_logic;
42             meio  : out std_logic
43         );
44     end component contador_m;
45
46     component interface_hcsr04 is
```

```

47  port (
48    clock      : in  std_logic;
49    reset      : in  std_logic;
50    medir      : in  std_logic;
51    echo       : in  std_logic;
52    trigger    : out std_logic;
53    medida    : out std_logic_vector(11 downto 0);
54    pronto    : out std_logic;
55    db_reset   : out std_logic;
56    db_medir   : out std_logic;
57    db_estado  : out std_logic_vector(3 downto 0)
58  );
59 end component interface_hcsr04;
60
61 component rx_serial_7O1 is
62   generic (constant samples : natural := 16);
63   port (
64     clock      : in  std_logic;
65     reset      : in  std_logic;
66     dado_serial : in  std_logic;
67     dado_recebido : out std_logic_vector(6 downto 0);
68     paridade_recebida : out std_logic;
69     pronto    : out std_logic;
70     db_dado_serial : out std_logic;
71     db_estado  : out std_logic_vector(3 downto 0)
72  );
73 end component rx_serial_7O1;
74
75 component hexa7seg is
76   port (
77     hexa : in  std_logic_vector(3 downto 0);
78     sseg : out std_logic_vector(6 downto 0)
79  );
80 end component hexa7seg;
81
82 end package;

```

B Declaração de entidades

B.1 Entidade principal

```
1 entity pill_dispenser is
2   port (
3     clock      : in  std_logic;
4     reset      : in  std_logic;
5     db_switch  : in  std_logic_vector(1 downto 0);
6     echo        : in  std_logic;
7     serial      : in  std_logic;
8     pwm         : out std_logic_vector(CONTAINERS-1 downto 0);
9     trigger     : out std_logic;
10    alert       : out std_logic;
11    — debug
12    db_reset    : out std_logic;
13    db_pwm      : out std_logic;
14    db_echo     : out std_logic;
15    db_trigger  : out std_logic;
16    db_state    : out std_logic_vector(6 downto 0);
17    db_measurement : out std_logic_vector(20 downto 0)
18  );
19 end entity;
```

B.2 Unidade de Controle

```
1 entity pill_dispenser_uc is
2   port (
3     clock      : in  std_logic;
4     reset      : in  std_logic;
5     check_timeout : in  std_logic;
6     safety_timeout : in  std_logic;
7     detected    : in  std_logic;
8     — control
9     count      : out std_logic;
10    move       : out std_logic;
11    discount   : out std_logic;
12    — debug
13    db_state   : out std_logic_vector(3 downto 0)
14  );
15 end entity;
```

B.3 Fluxo de Dados

```
1 entity pill_dispenser_fd is
2   port (
3     clock      : in  std_logic;
4     reset      : in  std_logic;
5     serial      : in  std_logic;
6     echo        : in  std_logic;
```

```

7   count      : in  std_logic;
8   move       : in  std_logic;
9   discount    : in  std_logic;
10  — outputs
11  trigger     : out std_logic;
12  check_end   : out std_logic;
13  safety_end  : out std_logic;
14  pwm         : out std_logic_vector(CONTAINERS-1 downto 0);
15  alert        : out std_logic;
16  — debug
17  db_measurement : out std_logic_vector(11 downto 0);
18  — states
19  rx_state    : out std_logic_vector(3 downto 0);
20  sensor_state: out std_logic_vector(3 downto 0)
21 );
22 end entity;

```

C Implementação de arquiteturas

C.1 Contador decrescente

```
1 entity downwards_counter is
2     generic (constant N: integer := 3);
3     port (
4         clock : in std_logic;
5         clear : in std_logic;
6         count : in std_logic;
7         enable : in std_logic;
8         D      : in std_logic_vector (N-1 downto 0);
9         Q      : out std_logic_vector (N-1 downto 0)
10    );
11 end entity downwards_counter;
12
13 architecture behavioral of downwards_counter is
14     signal IQ : natural;
15 begin
16
17     process(clock, clear, enable, IQ)
18     begin
19         if (clear = '1') then IQ <= 0;
20         elsif (rising_edge(clock)) then
21             if (count='1') then
22                 if (IQ /= 0) then IQ <= IQ - 1;
23                 end if;
24             elsif (enable='1') then IQ <= to_integer(unsigned(D));
25             else IQ <= IQ;
26             end if;
27         end if;
28         Q <= std_logic_vector(to_unsigned(IQ, Q'length));
29     end process;
30
31 end architecture behavioral;
```

C.2 Unidade de Controle

```
1 architecture fsm_arch of pill_dispenser_uc is
2     type state is (idle, check, dispense, countdown, await);
3     signal current_state, next_state : state;
4 begin
5
6     — state
7     process (reset, clock)
8     begin
9         if reset='1' then
10            current_state <= idle;
11        elsif clock'event and clock='1' then
12            current_state <= next_state;
13        end if;
```

```

14    end process;
15
16    — next state
17    process (current_state, detected, check_timeout, safety_timeout)
18    begin
19        case current_state is
20            when idle      => if detected='1' then next_state <= check;
21                            else                      next_state <= idle;
22                            end if;
23            when check     => if detected='0' then next_state <= idle;
24                            elsif check_timeout='1'
25                                then next_state <= dispense;
26                            else                      next_state <= check;
27                            end if;
28            when dispense   => if safety_timeout='1'
29                            then next_state <= countdown;
30                            else                      next_state <= dispense;
31                            end if;
32            when countdown  => next_state <= await;
33            when await      => if detected='0' then next_state <= idle;
34                            elsif safety_timeout='1'
35                                then next_state <= dispense;
36                            else                      next_state <= await;
37                            end if;
38            when others     => next_state <= idle;
39        end case;
40    end process;
41
42    — control signals
43    with current_state select
44        count    <= '1' when check | dispense | await, '0' when others;
45    with current_state select
46        move     <= '1' when dispense, '0' when others;
47    with current_state select
48        discount <= '1' when countdown, '0' when others;
49
50    with current_state select
51        db_state <= "0000" when idle,
52                            "0001" when check,
53                            "0010" when dispense,
54                            "0011" when countdown,
55                            "0100" when await,
56                            "1110" when others; — Error
57
58 end architecture;

```

C.3 Fluxo de Dados

```

1 ...
2 constant check_timeout      : natural := 100_000_000;
3 ...

```

```

4 | constant safety_timeout : natural := 25_000_000;
5 |
6 | ...
7 | CHECK_TIMER: contador_m
8 |   generic map (M => check_timeout, N => check_timeout_bits)
9 |   port map (
10 |     clock => clock,
11 |     zera => s_count_reset,
12 |     conta => count,
13 |     Q => open,
14 |     fim => check_end,
15 |     meio => open
16 |   );
17 |
18 | SAFETY_TIMER: contador_m
19 |   generic map (M => safety_timeout, N => safety_timeout_bits)
20 |   port map (
21 |     clock => clock,
22 |     zera => s_count_reset,
23 |     conta => count,
24 |     Q => open,
25 |     fim => safety_end,
26 |     meio => open
27 |   );
28 |
29 | — 4 bits to identify the container (6 downto 3) and
30 | — the 3 least significant bits to indicate the proper dosage of
31 | — the container (2 downto 0)
32 | CONTAINERS: for i in pill_containers'range generate
33 |   containers_enable(i) <= '1' when
34 |     i = to_integer(unsigned(s_dosage(6 downto 3))) and
35 |     s_rx_received='1'
36 |   else '0';
37 |   pwm(i) <= '0' when pill_containers(i) = "000" else s_pwm;
38 |
39 | PILL_COUNT: downwards_counter
40 |   generic map (N => 3)
41 |   port map (
42 |     clock => clock,
43 |     clear => reset,
44 |     count => discount,
45 |     enable => containers_enable(i),
46 |     D => s_dosage(2 downto 0),
47 |     Q => pill_containers(i)
48 |   );
49 | end generate;
50 |
51 | alert <= '0' when
52 |   pill_containers = (pill_containers'range => (others => '0'))
53 | else '1';

```