

Problema 2: ... Rega Automatizada (Melhorias)!

Cláudio Daniel Figueredo Peruna ^{*} Paulo Gabriel da Rocha Costa Silva [†]
Paulo Henrique Dantas Barreto [‡]

2024

Resumo

Este estudo apresenta a evolução de um sistema automatizado de irrigação, inicialmente proposto, com a implementação de um circuito aprimorado. O novo design incorpora funcionalidades avançadas, incluindo um cronômetro e uma interface de visualização aprimorada das informações do sistema. As ferramentas de desenvolvimento *Verilog* e *Quartus* foram empregadas para a implementação na placa *CPLD MAX II*. Os resultados demonstram a eficácia do uso de recursos como contadores síncronos e assíncronos, *flip-flops*, *latches* e manipulação de *clocks*, evidenciando o potencial do sistema para otimizar processos de irrigação.

Palavras-chaves: irrigação automatizada. circuitos digitais. verilog. quartus. CPLD MAX II. sistemas digitais. contadores síncronos. contadores assíncronos. flip-flops. latches. manipulação de clocks. automação agrícola. otimização de processos.

Introdução

A evolução tecnológica é um processo contínuo e inexorável. Como observa Bill Gates, "Nós sempre superestimamos as mudanças que ocorrerão nos próximos dois anos e subestimamos as mudanças que ocorrerão na próxima década. Não se deixe levar pela inércia. Este é um momento emocionante" ([GATES, 1995](#)). Este dinamismo exige que inovações sejam constantemente atualizadas e melhoradas para acompanhar as demandas emergentes. No contexto da agricultura, a eficiência e a sustentabilidade dos processos são fundamentais.

Com isso em mente, este estudo visa aprimorar um sistema automatizado de irrigação previamente desenvolvido, incorporando um novo circuito com funcionalidades avançadas. Este avanço foi implementado utilizando as ferramentas de desenvolvimento

^{*}danielperuna2012@gmail.com

[†]paulogrcsilva@gmail.com

[‡]rickdeuxvult@gmail.com

Verilog e *Quartus* na plataforma *CPLD MAX II*. As melhorias incluem a adição de um cronômetro e uma interface de visualização mais sofisticada, com as imagens sendo mostradas na matriz de LEDs da placa, e não mais no seu *display* de 7 segmentos. Dessa forma, esses enriquecimentos do projeto atendem melhor às necessidades dos usuários e otimizam o uso dos recursos hídricos.

1 Metodologia

A metodologia adotada seguiu uma sequência de etapas bem definidas, conforme descrito a seguir:

1. Definição da Visão Geral e Diagramação de Alto Nível do Projeto: Inicialmente, elaborou-se uma visão abrangente do projeto, seguida pela criação de diagramas de alto nível, os quais delinearam as interações e os principais componentes do sistema.
2. Listagem dos Módulos Necessários: Posteriormente, realizou-se a identificação e a especificação detalhada dos módulos funcionais para a consecução do projeto.
3. Escolha das Ferramentas Utilizadas: Em seguida, selecionaram-se as ferramentas mais adequadas para o desenvolvimento do projeto, com base em critérios técnicos específicos.
4. Implementação da Projeção Utilizando *Verilog*: Por fim, procedeu-se à implementação do projeto, empregando a linguagem de descrição de hardware *Verilog*, assegurando a conformidade com as especificações estabelecidas nas etapas anteriores.

1.1 Ferramentas Utilizadas

No contexto do presente estudo, as metodologias e ferramentas empregadas são uma continuação das práticas adotadas no projeto anterior, com a intenção de aprimorar a eficiência e a integração do desenvolvimento de sistemas digitais. A placa de desenvolvimento eletrônica escolhida para este projeto é a *EPM240T100C5N*, pertencente à família *MAX II*, que foi selecionada por suas capacidades robustas e compatibilidade com o ambiente de desenvolvimento.

O código que descreve o circuito foi escrito utilizando o *Verilog HDL*, uma linguagem de descrição de hardware amplamente utilizada na indústria, conhecida por sua capacidade de modelar circuitos digitais. A compilação do código foi realizada no *Quartus II*, uma ferramenta de design de circuitos integrados da *Intel*, versão 20, que oferece uma plataforma poderosa para a implementação de designs em *hardware*.

Para o desenvolvimento da descrição de hardware, foi preferido o uso da IDE do *Visual Studio Code*, uma ferramenta moderna e flexível que proporciona uma experiência de desenvolvimento integrado otimizada. Mesmo que o código seja compilado no *Quartus II*, o *Visual Studio Code* foi utilizado para sua escrita e edição, aproveitando-se das funcionalidades avançadas da IDE para melhorar a produtividade do desenvolvedor.

A transferência e atualização do repositório contendo o código foram facilitadas pelo uso do *Git* e do serviço *GitHub*, que permitem uma colaboração eficiente entre diferentes máquinas, garantindo a integridade e a continuidade do projeto.

1.2 Escrita do código e Documentação

Para garantir que o código atingisse o máximo nível de legibilidade e intuitividade, adotou-se o uso de variáveis autoexplicativas e de nomenclatura clara. Além disso, incorporaram-se comentários detalhados em todas as seções do código, o que não apenas facilita o entendimento dos processos implementados, mas também elimina ambiguidade. Partes do código consideradas redundantes foram removidas. Adicionalmente, foram inseridas tabelas e informações relevantes na forma de comentários, proporcionando um referencial imediato que enriquece a compreensão contextual do código por parte dos leitores. Essa abordagem sistemática visa a otimização do código, assegurando que este seja compreensível e acessível mesmo para aqueles que não participaram diretamente de sua elaboração.

1.3 Visão Geral

Para uma compreensão aprofundada do funcionamento do circuito projetado, recomenda-se a análise do fluxograma ilustrado na Figura 1.

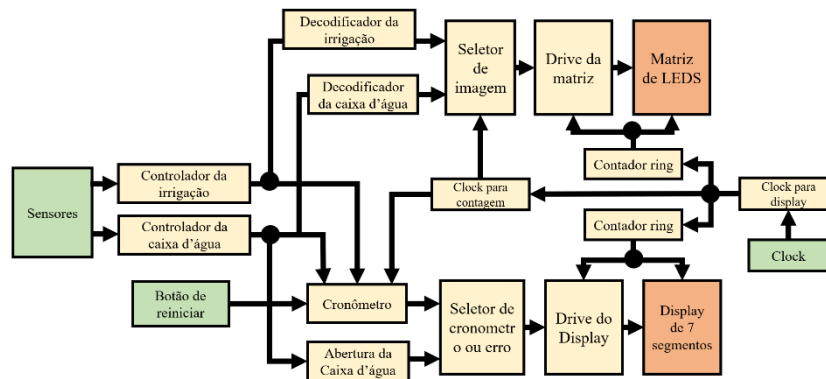


Figura 1 – Fluxograma geral do projeto

Os sensores empregados englobam tanto os sensores da caixa d'água quanto os sensores de ambiente. Os sensores da caixa d'água são categorizados em quatro níveis: crítico, baixo, médio e alto. Os sensores ambientais medem a umidade do solo, a umidade do ar e a temperatura, com a lógica inversa aplicada à temperatura.

Os sensores da caixa d'água fornecem dados para o codificador, que converte esses dados em valores binários de dois bits, categorizando-os como crítico, baixo, médio ou alto. Além disso, esses dados são encaminhados para o sistema de erro, que verifica a regularidade entre os sensores. Caso seja detectada uma irregularidade, o sistema emite um sinal que altera a imagem do cronômetro e o estado da válvula para uma mensagem de erro exibida no *display*.

A codificação gerada é transmitida ao decodificador da caixa d'água, cuja função é exibir a imagem correspondente na matriz de LEDs. Para monitorar o estado da válvula da caixa d'água, o sensor de nível alto, junto com o sistema de erro, é enviado a um módulo responsável por definir a exibição de "A"(aberto) ou "F"(fechado) no cronômetro, especificamente no *display* mais à esquerda do multiplexador.

O multiplexador, que atua como *driver* do *display*, seleciona qual dos quatro *displays* será ativado em cada instante, com o *display* mais à esquerda exibindo a informação mencionada.

No sistema de irrigação, as chaves de irrigação são processadas por um codificador que determina o modo de irrigação: aspersão ou gotejamento. Este codificador de um bit também avalia se as condições são adequadas para a irrigação, considerando o nível da caixa d'água para determinar se está em estado crítico. O codificador envia o modo de irrigação e a confirmação da irrigação ao decodificador de irrigação, que então gera a imagem correspondente na matriz de LEDs.

Para a intercalação das imagens, utiliza-se um *clock* integrado à placa. Inicialmente, este *clock* é enviado a um divisor de *clock*, responsável por intercalar as colunas, tanto do *display* quanto da matriz de LEDs, em uma frequência maior. Este *clock*, então, alimenta os contadores: um *ring counter* de 3 bits e um de 4 bits, sendo o de 4 bits designado para o *display* e o de 3 bits para a matriz. A escolha de 3 bits para a matriz se deve ao fato de que a imagem é espelhada, exibindo duas colunas por vez, com a coluna central sendo exibida individualmente, resultando em uma exibição dual.

Posteriormente, o sinal do *clock* passa por um segundo divisor de *clock*, que regula a alternância entre as imagens da matriz de LEDs, dividindo o tempo igualmente entre a representação da irrigação e da caixa d'água.

A matriz utiliza dois decodificadores, um para a caixa d'água e outro para a irrigação, que intercalam as imagens conforme o *clock* mais lento. A seguir, o sinal é direcionado para o seletor, que alimenta o drive da matriz. Este drive seleciona qual coluna será exibida em cada ciclo. O contador de três bits, após passar por um decodificador que realiza a redistribuição de três entradas para cinco saídas, organiza as colunas da seguinte maneira: a saída do contador dois controla as colunas quatro e zero, a saída do contador um controla as colunas três e um, e a saída do contador zero controla a coluna central, coluna dois. Esta redistribuição é então alimentada ao drive da matriz, que ativa e desativa as colunas conforme necessário. Para o funcionamento correto, a coluna deve estar desativada enquanto a linha está ativada, permitindo uma seleção precisa.

No sistema de decodificação, a lógica é configurar quais linhas de cada coluna serão desativadas, uma vez que o padrão é estar ativado. Assim, a operação principal é a desativação seletiva das linhas.

1.4 Módulos

Para a concretização do circuito conforme delineado no planejamento inicial, foram concebidos diversos módulos especializados, cada um com a finalidade de atender a objetivos técnicos específicos para o funcionamento do sistema. Esses módulos abarcam uma gama diversificada de funcionalidades. São eles: Módulos para Abastecimento de Água (*Water Supply*); Módulos Utilitários (*Utils*); Módulos para Matriz de LEDs (*Matrix*); Módulos para Temporização (*Timer*); e Módulos para Irrigação (*Irrigation*).

Os módulos para a irrigação, ou tipo de rega, gerenciam os ciclos de irrigação, controlando a ativação e desativação dos mecanismos de rega conforme os requisitos de umidade do ar e do solo, além da temperatura. Estes não foram alterados na atual melhoria do projeto. Assim, não precisarão ser mencionados novamente.

No total, foram desenvolvidos 29 módulos. Para evitar a extensão excessiva do

presente relatório, alguns módulos serão abordados de forma mais concisa, especialmente aqueles já discutidos no projeto anterior. Portanto, não serão repetidas as tabelas verdade e simplificações para esses módulos. O foco será nos módulos recém-criados e nos que sofreram modificações.

1.4.1 Water Supply, ou Válvula de abastecimento de água

Responsáveis pelo controle e gerenciamento do nível de água. Os módulos *water_encoder* e o *water_sensors_checker* não tiveram sua lógica alterada. A única mudança no *water_encoder* é o uso de barramento de 2 bits em sua saída de dados, o que facilita seu uso durante o código principal.

- *water_supply_controller*

O módulo *water_supply_controller* controla a válvula de suprimento de água com base em duas condições: presença de valores conflitantes dos sensores e nível alto de água no tanque.

Quando a válvula deve ser fechada (indicada por 4'b1111):

1. Há valores conflitantes provenientes dos sensores (*water_sensors_conflicting*).
2. O nível da água no tanque está alto (*high_water_level*).

A válvula é aberta (indicada por 4'b1110) somente quando não há conflitos nos sensores e o nível da água não está alto.

A lógica de controle funciona da seguinte forma:

- Os três bits superiores de *valvule* são sempre ativados (1) se o nível da água estiver alto.
- O bit menos significativo de *valvule* é ativado (1) se houver valores conflitantes dos sensores ou se o nível da água estiver alto. Caso contrário, ele é desativado (0).

Isso garante que a válvula esteja fechada em condições críticas e aberta quando seguro.

$$valvule[0] = water_sensors_conflicting + high_water_level \quad (1)$$

water_sensors_conflicting	high_water_level	valvule[0]
0	0	0
0	1	1
1	0	1
1	1	1

Tabela 1 – Tabela verdade dos sensores de água

1.4.2 *Utils*, ou Utilitários

Inclui diversas funções auxiliares e de suporte que contribuem para a operação geral do sistema, ao englobar desde os redutores de *clocks*, até *flip-flops* e multiplexadores. Como todos são ferramentas predefinidas pela literatura, não houve a necessidade de representar as tabelas e expressões deles. Não houve alteração no *multiplexer_2x1*, o multiplexador 2x1 que foi utilizado no projeto anterior. No entanto, mais dois multiplexadores foram criados, além de 6 outros módulos. Dado que todos os módulos foram desenvolvidos com base em materiais didáticos, não se faz necessário a representação de tabelas verdade e expressões lógicas detalhadas para cada um deles. A fundamentação teórica e prática fornecida pelos materiais didáticos já garante a correta implementação dos circuitos e funcionalidades.

- Multiplexadores novos

Ambos os multiplexadores funcionam de maneira semelhante, baseados na literatura de (FLOYD, 2007), selecionando uma das entradas de dados com base no valor binário do contador (*ring_counter*). O valor selecionado é então transmitido para a saída *out*, permitindo a escolha dinâmica de qual dado será enviado adiante no circuito. Os novos multiplexadores são os *multiplexer_3x1* e *multiplexer_4x1*, que representam respectivamente multiplexadores 3x1 e 4x1, respectivamente, adaptados para a saída do *contador ring*, onde apenas um dos bits está em nível lógico alto por vez.

- *always_on*

Foi projetado para garantir um sinal positivo constante independentemente das condições de entrada. A operação começa com a inversão da entrada *a*, realizada pela porta NOT, que produz a *_not*. A propriedade lógica fundamental utilizada aqui é que qualquer valor lógico *A* somado com sua negação *A'* resulta em 0 (zero). Em termos da álgebra de *Boole*, isso é expresso como $A + A' = 0$.

A	A'		Saída
0	0		x
0	1		1
1	0		1
1	1		x

Tabela 2 – Tabela-verdade do módulo *always_on*

- *clock_divisor*

É responsável pela redução da frequência do *clock* original. Este módulo utiliza uma implementação que emprega cinco *flip-flops* do tipo T conectados assincronamente, permitindo assim uma divisão do sinal de *clock*. Sua saída, *new_frequency*, é definida como a saída do último *flip-flop*, que agora representa a frequência reduzida do sinal de entrada. Vale ressaltar que múltiplas instâncias desse módulo foram declaradas para alcançar as frequências desejadas. A equação 1.4.2 e a Figura 2 representa a sua nova frequência de saída em relação à de entrada.

$$NovaFrequência = \frac{clock}{32}$$

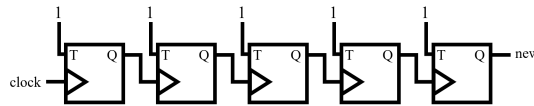


Figura 2 – Esquema do divisor de clock usando Flipflops T

- Flip-flops

Existem dois módulos de *flipflops* no projeto, o primeiro é o *flipflop_d*, usando nos contadores, tanto os decrescentes quanto os contadores em anéis, e o segundo é o *flipflop_t*, usado para a divisão do *clock*. Como o nome de ambos sugere, o primeiro é do tipo *D* e o segundo do tipo *T*. O primeiro contém entradas para *set* e *reset*, úteis para o uso de contadores, e o tipo *T* contém um *reset*, para caso seja necessário reiniciá-lo em alguma mudança de projeto futura.

- level_to_pulse

Este módulo converte um nível de sinal (0 ou 1) em um pulso curto na borda de subida do *clock*. O módulo é usado duas vezes no projeto, para que um pulso seja enviado toda vez que houver comutação no modo de aspersão. Portanto, quando atualizar para Aspersão ou Gotejamento, a recontagem do temporizador é iniciada automaticamente. O *level to pulse* está esquematizado na Figura 3, e funciona da seguinte maneira:

1. Armazenamento do nível: Um *flip-flop* do tipo *D* (*flipflop_d*) é utilizado para armazenar o nível atual do sinal de entrada *level*. O *clock* controla quando o *flip-flop* atualiza seu valor.
2. Negação do nível armazenado: A saída do *flip-flop* (*q*) é negada através de uma porta *NOT* (*not_q*). Isso significa que se *q* for 1, *not_q* será 0, e vice-versa.
3. Geração do pulso: Uma porta *AND* combina o nível negado (*not_q*) com o nível original (*level*). O pulso é gerado apenas quando o nível original muda de 0 para 1 (borda de subida) e o *flip-flop* ainda armazena o valor antigo (0). Isso garante que o pulso seja curto e preciso.

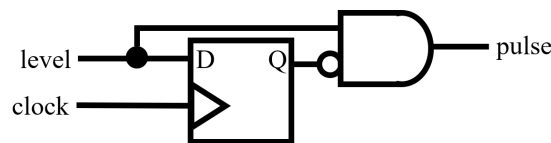


Figura 3 – Esquema do *level_to_pulse*

- pipe

Direciona os dados de entrada para saídas específicas do circuito. Este módulo facilita a gestão de fluxos de dados dentro do circuito digital, otimizando o processamento e a utilização dos recursos disponíveis. Como o projeto utiliza de *Verilog* estrutural ao invés de comportamental, com exceção dos *flipflops*, então o pipe funciona como um *assign*, onde a sinal de saída é idêntica ao sinal de entrada.

O módulo utiliza uma porta lógica *AND* para realizar essa operação. Por exemplo, em “ $A * A = A$ ”, quando o sinal de entrada “A” é multiplicado por si mesmo, o resultado é igual ao próprio sinal “A”. Portanto, o sinal de saída “out” será igual ao sinal de entrada “a”. É uma operação simples, mas útil em circuitos digitais.

A	A		Saída
0	0		0
0	1		x
1	0		x
1	1		1

Tabela 3 – Tabela-verdade do módulo *pipe*

1.4.3 *Matrix*, ou Matriz de LEDs

Como no desenvolvimento do projeto anterior, a matriz de LEDs não foi utilizada, todos os módulos presentes no atual projeto que são relacionados à matriz, são novos. A ideia por de trás dos módulos da matriz é que os decodificadores enviem informações das linhas de cada coluna para o controlador, o qual chamamos de *drive*, que mude a imagem mostrada conforme um contador *ring de 3bits*. A escolha do contador de *3 bits* para uma matriz de 5 colunas se deve ao fato da escolha das imagens mostradas serem simétricas no eixo vertical, o que reduz consideravelmente a quantidade de *elementos lógicos*, já que são decodificadas apenas 3 colunas para a imagem do modo de irrigação e 2, para o nível da caixa d’água.

- *column_selector*

É um contador de *3 bits* responsável por selecionar qual coluna será exibida por vez. O contador ter *3 bits* é devido ao fato de ter sido escolhido imagens espelhadas no eixo vertical para a matriz de LEDs, reduzindo assim a quantidade de recursos utilizados na sua implementação. Assim, cada decodificador só precisa enviar informações das linhas (um barramento de 7 bits) de apenas 3 colunas. Seu circuito está representado na Figura 4.

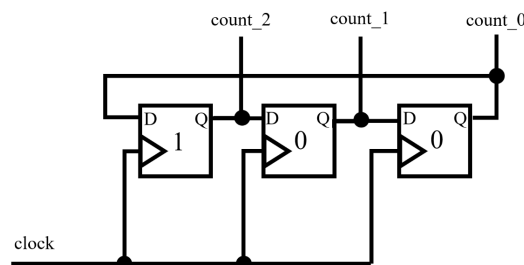


Figura 4 – Circuito do *column_selector*

Atual	Próximo		count ₂	count ₁	count ₀
100	010		0	1	0
010	001		0	0	1
001	100		1	0	0

Tabela 4 – Tabela de estados do contador ring de 3 bits

- *matrix_driver*

Seu funcionamento é de um multiplexador que utiliza barramentos, ao invés de sinais individuais. Assim, recebe-se 3 barramentos de 7 bits, um para cada coluna, contendo informações para ativação ou não de cada linha dessa coluna, e então retorna um desses barramentos por vez, de acordo com um seletor. Sua multiplexação é feita diretamente pelo *contador ring de 3 bits*, (*column_selector*) portanto apenas um dos *bits* pode estar em nível lógico alto por vez. Seu circuito se encontra na Figura 5.

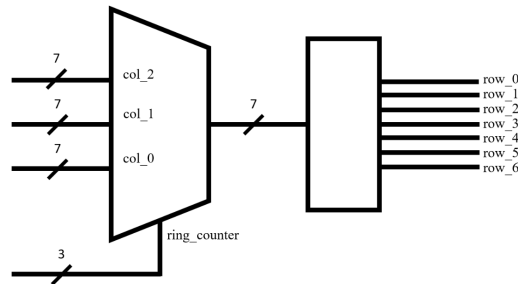


Figura 5 – Circuito do *matrix_driver*

- *matrix_ring_decoder*

Desempenha a função de transformar entradas de 3 bits em saídas de 5 bits. Este decodificador possui uma característica especial, já pode ser escrito sem a presença de portas lógicas, a pesar de terem sido usadas para a implementação em *Verilog* estrutural, por meio de instâncias do módulo *pipe*. Sua única função é a de redistribuir cada *bit* da saída do *column_selector* (3 bits) para as colunas da matriz de LEDs (5 bits), espelhando-os verticalmente. Seu esquema pode ser visto na Figura 6.

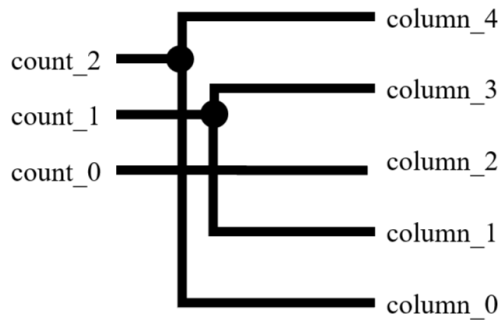


Figura 6 – Circuito do *matrix_ring_decoder*

- *irrigation_mode_decoder*

Decodifica os sinais provenientes do sistema de irrigação, representando modos como aspersão, gotejamento, ou a ausência de irrigação na matriz de LEDs. Este módulo recebe entradas específicas dos sinais de *splinker* e *irrigation*, ajustando a exibição conforme o modo de irrigação ativo. Caso exista irrigação e o modo de aspersão esteja ativo, será decodificado a imagem de um aspersor, já caso esteja no modo de gotejamento (quando *splinker* estiver em nível lógico baixo, pois são complementares), a imagem de uma gota será decodificada. Do contrário, caso não haja irrigação, será decodificado uma exclamação, independente do sinal do modo de irrigação. A tabela verdade de cada linha e coluna pode ser vista em 7, enquanto a simplificação de suas expressões pode ser vista nos mapas de Karnaugh 8, 9, 10, 11. Pode ser visto em cada mapa-K a presença da numeração de sua coluna e linha no formato "coluna:linha".

(2)

Coluna 2								
Aspersão	Irrigação	row_6	row_5	row_4	row_3	row_2	row_1	row_0
0	0	1	1	1	1	1	1	1
0	1	1	1	1	1	0	0	1
1	0	1	1	1	1	1	1	1
1	1	1	0	0	1	1	1	0

Coluna 1								
Aspersão	Irrigação	row_6	row_5	row_4	row_3	row_2	row_1	row_0
0	0	1	1	1	1	1	1	1
0	1	1	1	0	0	0	0	0
1	0	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	0

Coluna 0								
Aspersão	Irrigação	row_6	row_5	row_4	row_3	row_2	row_1	row_0
0	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1
1	1	0	1	1	1	1	0	1

Figura 7 – Tabelas verdade do *irrigation_decoder*

Karnaugh Map

2:6, 2:3, 1:5, 0:5, 0:4, 0:3, 0:2, 0:0 Irrigação

		0	1
Aspersão	0	1	1
	1	1	1

Figura 8 – Expressão: $Y = 1$

Karnaugh Map

2:5, 2:4, 2:0, 1:6 Irrigação

		0	1
Aspersão	0	1	1
	1	1	0

Figura 9 – Expressão: $Aspersão' + Irrigação'$

Karnaugh Map

2:2, 2:1, 1:4, 1:3, 1:2, 1:1 Irrigação

		0	1
Aspersão	0	1	0
	1	1	1

Figura 10 – Expressão: $Irrigação' + Aspersão$

Karnaugh Map

2:0, 0:1 Irrigação

		0	1
Aspersão	0	1	0
	1	1	0

Figura 11 – Expressão: $Irrigação'$

- *water_level_decoder*

Recebe o nível da água codificado e verifica a presença de erros. Este módulo é projetado para retornar imagens apropriadas à matriz de LEDs, com uma característica especial que envolve o uso de um barramento de 2 bits para as colunas do meio, que são idênticas. Em caso de detecção de erro, nenhuma imagem é exibida, alertando o sistema para a necessidade de correção. A tabela verdade contendo as linhas e colunas do decodificador da matriz está disponível na Figura 12, enquanto isso, a simplificação das expressões por meio do Mapa-K pode ser encontrado nas Figuras 13, 14, 15 e 16. A posição da expressão a ser usada na matriz é representada logo acima da tabela usando o formato "coluna:linha".

Coluna 1									
erro	nível_1	nível_0	row_6	row_5	row_4	row_3	row_2	row_1	row_0
0	0	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1
0	1	0	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

Coluna 0									
erro	nível_1	nível_0	row_6	row_5	row_4	row_3	row_2	row_1	row_0
0	0	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	0	1
0	1	0	1	1	0	0	0	0	1
0	1	1	0	0	0	0	0	0	1
1	0	0	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

Figura 12 – Tabelas verdade do *water_level_decoder*

Karnaugh Map

1:6, 1:5, 1:4, 1:3, 1:2, 1:1, 1:0, 0:0 n_1, n_0

	00	01	11	10
Erro 0	1	1	1	1
1	1	1	1	1

Figura 13 – Expressão: $Y = 1$

Karnaugh Map

0:6, 0:5 n_1, n_0

	00	01	11	10
Erro 0	1	1	0	1
1	1	1	1	1

Figura 14 – Expressão: $Y = n'_1 + n'_0 + Erro$

Karnaugh Map

0:4, 0:3 n_1, n_0

	00	01	11	10
Erro 0	1	1	0	0
1	1	1	1	1

Figura 15 – Expressão: $Y = n'_1 + Erro$

Karnaugh Map

		00	01	11	10
0:2, 0:1	$n1, n0$				
Erro 0		1	0	0	0
1		1	1	1	1

Figura 16 – Expressão: $Y = n'_1n'_0 + Erro$

- *select_matrix_display_mode*

É um multiplexador que seleciona entre as imagens representativas do modo de irrigação e as do nível da caixa d'água. Este módulo alterna entre os modos com base em um seletor controlado por um clock, permitindo uma transição fluida e sincronizada entre as diferentes informações exibidas na matriz de LEDs. Seu circuito está representado na Figura 17.

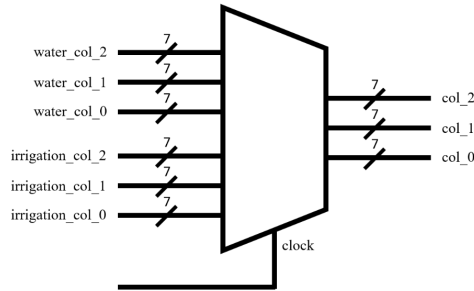


Figura 17 – Circuito do *select_matrix_display_mode*

1.4.4 Timer, ou Temporizador

Implementa as funcionalidades de temporização, necessárias para visualização no display.

- Contadores

A lógica de representação desses contadores utiliza 4 bits de saída para representar um BCD (Binary-Coded Decimal). Este código binário é então direcionado ao decodificador, que é responsável por interpretar e exibir os valores no display. Para a exibição no display, apenas os valores binários de 0 a 9 são utilizados, enquanto o valor 10 é descartado. Os valores binários excedentes são empregados para representar letras, proporcionando uma flexibilidade adicional na exibição de informações alfanuméricas no display.

Os flipflops dos contadores são síncronos, porém os contadores são assíncronos entre si. Dependendo dos contadores anteriores para realizar a contagem decrescente. Ao alcançar zero, o *clock* que entra no contador mais à esquerda (o *down_from_5*) é inibido, até que haja um pulso de atualização pelo módulo *timer_reseter*. O uso de três contadores que usam a quantidade mínimo necessária de *flipflops* para a

representação de cada dígito é importante para a redução do uso de recursos da placa. As tabelas verdade dos três contadores se encontram na Figura 18.

1. *down_from_9*

O contador "down_from_9" é projetado para realizar contagens decrescentes de 9 a 0. Este módulo é composto por quatro flip-flops do tipo D, que juntos representam quatro bits. A saída combinada dos quatro flip-flops é utilizada para alimentar o clock do contador subsequente, o "down_from_3", enquanto isso, seu *clock* vem da saída combinada do *down_from_5*. Esta interconexão garante uma sincronização adequada entre os módulos de contagem.

2. *down_from_5*

O contador "down_from_5" desempenha a função de contar de 5 a 0 de forma decrescente. Este módulo é constituído por três flip-flops do tipo D, representando três bits. A saída combinada dos três flip-flops é utilizada para alimentar o clock do contador "down_from_9", criando uma cadeia lógica de contagem decrescente.

3. *down_from_3*

O contador "down_from_3" conta de 3 a 0 de forma decrescente e também é composto por dois flip-flops do tipo D, associados sincronicamente. Esse contador recebe como *clock* um pulso do contador anterior (o *down_from_9*), para quando este alcançar zero, fazendo o *down_from_3* mudar para o próximo estado.

Contador Decrescente de 4 bits						
Atual	Próximo	Q3	Q2	Q1	Q0	Decimal
1001	1000	1	0	0	0	9
1000	0111	0	1	1	1	8
0111	0110	0	1	1	0	7
0110	0101	0	1	0	1	6
0101	0100	0	1	0	0	5
0100	0011	0	0	1	1	4
0011	0010	0	0	1	0	3
0010	0001	0	0	0	1	2
0001	0000	0	0	0	0	1
0000	1001	1	0	0	1	0
Contador Decrescente de 3 bits						
Atual	Próximo	Q2	Q1	Q0	Decimal	
101	100	1	0	0	5	
100	011	0	1	1	4	
011	010	0	1	0	3	
010	001	0	0	1	2	
001	000	0	0	0	1	
000	101	1	0	1	0	
Contador Decrescente de 2 bits						
Atual	Próximo	Q1	Q0	Decimal		
11	10	1	0	3		
10	01	0	1	2		
01	00	0	0	1		
00	11	0	1	0		

Figura 18 – Tabelas verdade dos contadores

- *display_decoder*

Este módulo converte um código binário de 4 bits em sinais de controle para um *display* de sete segmentos. O decodificador também suporta a exibição das letras 'A', 'F', 'E', 'r', e 'o', além dos dígitos de 0 a 9. Vale ressaltar que a escolha do 'A' e 'F' como '1110' e '1111', respectivamente, não fora por acaso, mas uma forma de simplificar o uso, pois apenas o bit menos significativo muda entre ambos.

A entrada de 4 bits (*data*) é usada para determinar quais segmentos do *display* devem ser acesos para formar o caractere correspondente. Cada segmento do *display* (*a* até *g*) é controlado por uma série de condições lógicas que dependem dos bits da entrada. Quando um bit específico da entrada está presente, os segmentos necessários são ativados para formar o caractere correspondente no *display*. Sua tabela verdade se encontra na Tabela 5 e o mapa K de cada segmento pode ser visto nas figuras 19 a 25.

d ₃	d ₂	d ₁	d ₀		a	b	c	d	e	f	g		Rep.
0	0	0	0		0	0	0	0	0	0	1		0
0	0	0	1		1	0	0	1	1	1	1		1
0	0	1	0		0	0	1	0	0	1	0		2
0	0	1	1		0	0	0	0	1	1	0		3
0	1	0	0		1	0	0	1	1	0	0		4
0	1	0	1		0	1	0	0	1	0	0		5
0	1	1	0		0	1	0	0	0	0	0		6
0	1	1	1		0	0	0	1	1	1	1		7
1	0	0	0		0	0	0	0	0	0	0		8
1	0	0	1		0	0	0	0	1	0	0		9
1	0	1	0		x	x	x	x	x	x	x		
1	0	1	1		0	1	1	0	0	0	0		E
1	1	0	0		1	1	1	1	0	1	0		r
1	1	0	1		1	1	0	0	0	1	0		o
1	1	1	0		0	0	0	1	0	0	0		A
1	1	1	1		0	1	1	1	0	0	0		F

Tabela 5 – Tabela da decodificação do *display* de 7 segmentos

Karnaugh Map

<i>a</i>		<i>d</i> ₁ , <i>d</i> ₀			
		00	01	11	10
<i>d</i> ₃ , <i>d</i> ₂	00	0	1	0	0
	01	1	0	0	0
	11	1	1	0	0
	10	0	0	0	-

Figura 19 – Segmento *a*

$$a = d'_3 d'_2 d'_1 d_0 + d_2 d'_1 d'_0 + d_3 d_2 d'_1$$

Karnaugh Map

<i>b</i>		<i>d1,d0</i>			
		00	01	11	10
<i>d3,d2</i>	00	0	0	0	0
	01	0	1	0	1
	11	1	1	1	0
	10	0	0	1	-

Fig. 20. Karnaugh map for the function b

Fig. 20

Figura 20 – Segmento b

$$b = d_2 d_1' d_0 + d_3' d_2 d_1 d_0' + d_3 d_2 d_1' + d_3 d_1 d_0$$

Karnaugh Map

<i>c</i>		<i>d1,d0</i>			
		00	01	11	10
<i>d3,d2</i>	00	0	0	0	1
	01	0	0	0	0
	11	1	0	1	0
	10	0	0	1	-

Figura 21 – Segmento c

$$c = d_2' d_1 d_0' + d_3 d_2 d_1' d_0' + d_3 d_1 d_0$$

Karnaugh Map

<i>d</i>		<i>d1,d0</i>			
		00	01	11	10
<i>d3,d2</i>	00	0	1	0	0
	01	1	0	1	0
	11	1	0	1	1
	10	0	0	0	-

Fig. 22. Karnaugh map for the function d

Figura 22 – Segmento d

$$d = d_3' d_2' d_1' d_0 + d_2 d_1' d_0' + d_2 d_1 d_0 + d_3 d_2 d_0'$$

Karnaugh Map

<i>e</i>		<i>d1,d0</i>			
		00	01	11	10
<i>d3,d2</i>	00	0	1	1	0
	01	1	1	1	0
	11	0	0	0	0
	10	0	1	0	-

Figura 23 – Segmento *e*

$$e = d'_3d_0 + d'_3d_2d'_1 + d'_2d'_1d_0$$

Karnaugh Map

<i>f</i>		<i>d1,d0</i>			
		00	01	11	10
<i>d3,d2</i>	00	0	1	1	1
	01	0	0	1	0
	11	1	1	0	0
	10	0	0	0	-

Figura 24 – Segmento *f*

$$f = d'_3d'_2d_0 + d'_3d_1d_0 + d_3d_2d'_1 + d'_3d'_2d_1$$

Karnaugh Map

<i>g</i>		<i>d1,d0</i>			
		00	01	11	10
<i>d3,d2</i>	00	1	1	0	0
	01	0	0	1	0
	11	0	0	0	0
	10	0	0	0	-

Figura 25 – Segmento *g*

$$g = d'_3d'_2d'_1 + d'_3d_2d_1d_0$$

- *display_selector*

O seletor de *display* funciona de forma muito similar ao *column_selector* descrito anteriormente para a matriz de LEDs, sendo esse um contador do tipo *ring de 4bits*. Sua função é a de selecionar qual dos dígitos será mostrado por vez no *display* de sete segmentos, o que permite que sejam mostradas todas as informações. Para isso, recebe-se um *clock* que seja alto suficiente para que pareça uma imagem estática aos olhos humanos, quando na verdade apenas um dos dígitos é exibido por vez. Seu esquema se encontra na Figura 26 e sua tabela de estados na Tabela 6.

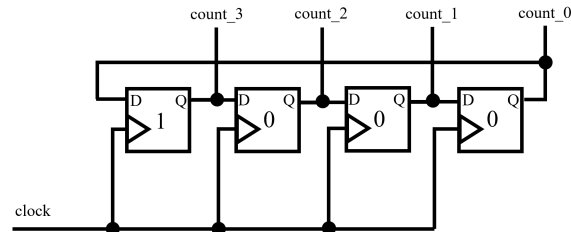


Figura 26 – Esquema do *display_selector*

Atual	Próximo		count ₃	count ₂	count ₁	count ₀
1000	0100		0	1	0	0
0100	0010		0	0	1	0
0010	0001		0	0	0	1
0001	1000		1	0	0	0

Tabela 6 – Tabela de estados do contador ring de 4 bits

- *display_driver*

seleciona um conjunto de dados de entrada entre quatro possíveis (*data₃*, *data₂*, *data₁*, *data₀*) com base no valor do *ring_counter* e envia o conjunto selecionado para a saída out. Utiliza quatro multiplexadores 4x1 para fazer essa seleção, um para cada bit dos dados de entrada, permitindo que diferentes dados sejam exibidos em diferentes momentos conforme o valor do contador de anel. O driver recebe informações de instâncias do *error_or_info*, responsável por intercalar entre uma mensagem de erro ou o representação padrão do cronômetro juntamente da abertura da caixa d'água.

Assim, *data₃* pode receber a informação correspondente aos caracteres 'A' ou 'E', *data₂* o dígito da dezena de minutos do contador ou o caractere 'r', *data₁* o dígito da unidade de minutos ou 'r', enquanto *data₀*, a dezena de segundos ou o caractere 'o'. Seu esquema se encontra na Figura 27.

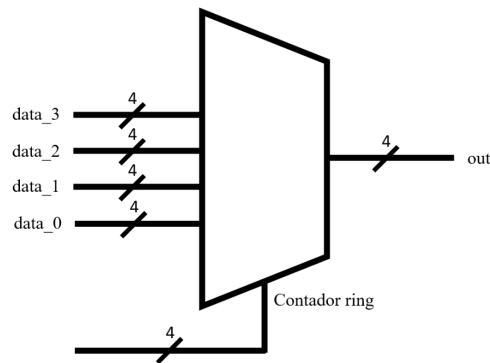


Figura 27 – Esquema do *display_driver*

- *error_or_info*

O módulo *error_or_info* seleciona entre dois conjuntos de dados de entrada (*info* e *error*) com base no sinal *has_error*. Se *has_error* for verdadeiro, a saída *out* exibirá os dados de *error*; caso contrário, exibirá os dados de *info*. Sua visualização se encontra na Figura 28. Ele usa multiplexadores 2x1 para fazer essa seleção para cada bit dos dados de entrada, permitindo a exibição de mensagens de erro ou informação conforme a necessidade. Quatro instâncias desse módulo são usadas no projeto, uma para cada dígito do mostrador de 7 segmentos.

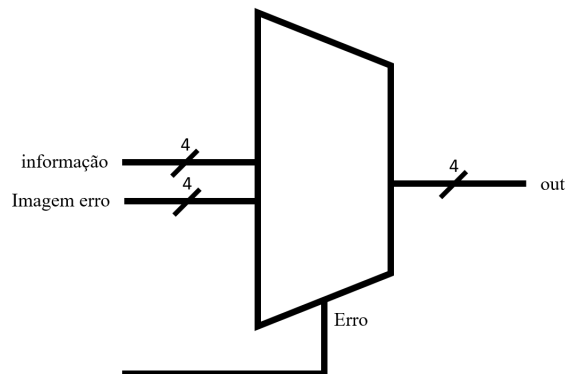


Figura 28 – Esquema do *error_or_info*

- *set_display_init*

O módulo *set_display_init* configura os valores iniciais dos *displays* de minutos e segundos dependendo do modo de operação (*splinker_mode_on*) e do sinal de reinício (*recount*).

1. Quando *splinker_mode_on* é verdadeiro, ele configura os *displays* de minutos e segundos para os valores correspondentes ao modo "Splinker"("0011 0000 0000", "3 0 0"em decimal).
2. Quando *splinker_mode_on* é falso (modo "Dripper"), ele configura os *displays* para os valores do modo "Dripper"("0001 0101 0000", "1 5 0"em decimal).

3. *recount* é utilizado para indicar quando os *flipflops* devem ser configurados para o valor inicial novamente.

O circuito utiliza portas lógicas para determinar os valores específicos que devem ser enviados aos *displays* dos minutos (*minutes_d_setter*, *minutes_u_setter*, *minutes_d_resetter*, *minutes_u_resetter*) e segundos (*seconds_d_resetter*) dependendo do modo de operação e do sinal de reinício. As Tabelas 7 a 12 mostram as tabelas verdade dos Setters e Resetters de cada dígito do temporizador.

Dezena de minutos									
Recontar	Aspersão	Setter				Resetter			
		set ₃	set ₂	set ₁	set ₀	reset ₃	reset ₂	reset ₁	reset ₀
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	0	0
1	1	0	0	0	1	1	1	1	0

Tabela 7 – Tabela verdade para a dezena de minutos

Saída	set ₃	set ₂	set ₁	set ₀	reset ₃	reset ₂	reset ₁	reset ₀
Expressão	0	0	RA'	RA' + RA = R	RA' + RA = R	RA' + RA = R	RA	0

Tabela 8 – Sets e resets das unidades de minutos

Unidade de minutos									
Recontar	Aspersão	Setter				Resetter			
		set ₃	set ₂	set ₁	set ₀	reset ₃	reset ₂	reset ₁	reset ₀
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1
1	1	0	1	0	1	1	0	1	0

Tabela 9 – Tabela verdade para a unidade de minutos

Saída	set ₃	set ₂	set ₁	set ₀	reset ₃	reset ₂	reset ₁	reset ₀
Expressão	0	RA	0	RA	RA' + RA = R	RA'	RA' + RA = R	RA'

Tabela 10 – Sets e resets das unidades de minutos

Dezena de segundos									
Recontar	Aspersão	Setter				Resetter			
		set ₃	set ₂	set ₁	set ₀	reset ₃	reset ₂	reset ₁	reset ₀
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1
1	1	0	0	0	0	1	1	1	1

Tabela 11 – Tabela verdade da dezena dos segundos

Saída	set ₃	set ₂	set ₁	set ₀	reset ₃	reset ₂	reset ₁	reset ₀
Expressão	0	0	0	0	$RA' + RA = R$	$RA' + RA = R$	$RA' + RA = R$	$RA' + RA = R$

Tabela 12 – Sets e resets dos segundos

- *timer_reseter*

O módulo *timer_reseter* gera um sinal de *reset* para o temporizador com base em várias condições de entrada relacionadas ao sistema de irrigação. Ele é responsável por indicar que o temporizador deve refazer uma contagem, voltando ao estado inicial. Para isso, seu sinal deve ser enviado para o *set_display_init*, permitindo a atualização dos *flipflops* dos contadores.

O sinal *reset* é ativado se ocorrer uma das seguintes situações:

1. O sistema de irrigação está desligado (*irrigation_off*).
2. Um botão de *reset* foi pressionado e liberado (*button_released*).
3. Há valores conflitantes (*conflicting_values*).
4. Um pulso de comutação do sistema de irrigação é detectado (*irrigation_switch_pulse*).

Esse módulo é importante para que se possa retomar a contagem do temporizador, atualizando-o sempre que uma dessas condições acima sejam atendidas. Vale ressaltar que em alguns casos, ele é capaz de travar o temporizador no estado inicial, que pode ser de 30 minutos ou 15 minutos, como é o caso de valores conflitantes e irrigação inexistente. Seu esquema pode ser visualizado na Figura 29.

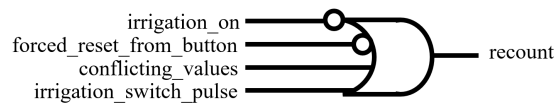


Figura 29 – Esquema do *timer_reseter*

1.5 Pinagem

Para o presente circuito, preservaram-se diversas entradas do projeto precedente, incluindo as chaves pinadas associadas aos níveis da caixa d'água, bem como os sensores de umidade e temperatura. Contudo, houve a introdução de novas entradas, destacando-se os sinais de *reset*, designados como Pulso 3 e Pulso 2, além do sinal de *clock*.

Toda a pinagem, com as devidas atualizações podem ser observadas na Tabela [13](#).

Componente	Nome	Pino
Entradas		
Switch	Temperatura	CH5 (30)
Switch	Umidade do Ar	CH4 (34)
Switch	Umidade da Terra	CH3 (36)
Switch	Nível Alto da Água	CH2 (38)
Switch	Nível Médio da Água	CH1 (40)
Switch	Nível Baixo da Água	CH0 (42)
Botão	Pulso 3	B0 (52)
Botão	Pulso 2	B1 (50)
Clock	Clock	12
Saídas		
LED	Indicador Válvula Abastecimento	LED0 (54)
LED	Contador 2	LED2 (57)
LED	Contador 1	LED3 (61)
LED	Contador 0	LED4 (67)
LED	Válvula Gotejadora	LED8 (75)
LED	Bomba Aspersora	LED9 (76)
LED	Alarme (Vermelho)	Red Led (86)
Display 7 Segmentos		
Cátodo	Segmento A	7 SEG-A (90)
Cátodo	Segmento B	7 SEG-B (70)
Cátodo	Segmento C	7 SEG-C (41)
Cátodo	Segmento D	7 SEG-D (98)
Cátodo	Segmento E	7 SEG-E (100)
Cátodo	Segmento F	7 SEG-F (92)
Cátodo	Segmento G	7 SEG-G (39)
Ânodo Comum	Display 0 (Dígito mais à direita)	7 SEG-D1 (37)
Ânodo Comum	Display 1	7 SEG-D2 (68)
Ânodo Comum	Display 2	7 SEG-D3 (66)
Ânodo Comum	Display 3 (Dígito mais à esquerda)	7 SEG-D4 (88)
Ponto Decimal		7 SEG-P (96)
Matriz		
Coluna	Coluna 0	97
Coluna	Coluna 1	99
Coluna	Coluna 2	95
Coluna	Coluna 3	82
Coluna	Coluna 4	78
Linha	Linha 0	85
Linha	Linha 1	83
Linha	Linha 2	84
Linha	Linha 3	87
Linha	Linha 4	81
Linha	Linha 5	91
Linha	Linha 6	89

Tabela 13 – Tabela de componentes e pinos

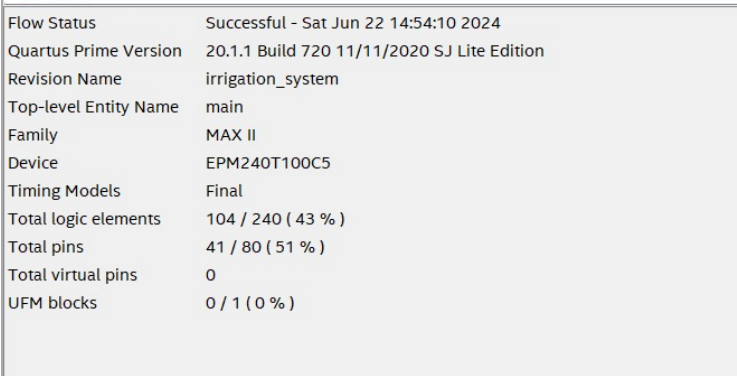
1.6 Ferramentas Auxiliares

Para simplificar o processo de desenvolvimento do projeto, foram empregados dois arquivos Python: o *clean.py* e o *update_config.py*. Ambos desempenharam papéis importantes no projeto anterior e foram novamente utilizados nesta iteração.

2 Resultados

No desenvolvimento do projeto, visando a utilização otimizada dos recursos da CPLD, foram adotadas diversas estratégias, como o espelhamento de colunas na matriz de LEDs, o que resultou em uma significativa redução no consumo de Elementos Lógicos (LEs).

No total, foram utilizados 104 dos 240 Elementos Lógicos disponíveis e 41 dos 80 pinos da CPLD. A utilização detalhada dos elementos lógicos é apresentada nas Figuras 30 e 31.



Flow Status	Successful - Sat Jun 22 14:54:10 2024
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	irrigation_system
Top-level Entity Name	main
Family	MAX II
Device	EPM240T100C5
Timing Models	Final
Total logic elements	104 / 240 (43 %)
Total pins	41 / 80 (51 %)
Total virtual pins	0
UFM blocks	0 / 1 (0 %)

Figura 30 – Utilização dos Elementos Lógicos

	Resource	Usage
1	▼ Total logic elements	108
1	-- Combinational with no register	66
2	-- Register only	4
3	-- Combinational with a register	38
2		
3	▼ Logic element usage by number of LUT inputs	
1	-- 4 input functions	49
2	-- 3 input functions	16
3	-- 2 input functions	6
4	-- 1 input functions	33
5	-- 0 input functions	0
4		
5	▼ Logic elements by mode	
1	-- normal mode	108
2	-- arithmetic mode	0
3	-- qfbk mode	0
4	-- register cascade mode	0
5	-- synchronous clear/load mode	0
6	-- asynchronous clear/load mode	16
6		
7	Total registers	42
8	I/O pins	41
9	Maximum fan-out node	mid_water_level
10	Maximum fan-out	16
11	Total fan-out	382
12	Average fan-out	2.56

Figura 31 – Uso dos recursos da CPLD

2.1 Imagens da Placa

Conforme mencionado na Seção 1.5 (Pinagem), foram adotadas diversas formas de representar as entradas para ilustrar de maneira clara cada situação possível do ambiente.

Primeiramente, a matriz de LEDs é utilizada para representar tanto o nível da caixa d'água (Figuras 32, 33, 34 e 35), quanto o tipo de irrigação (Figuras 36 e 37). Essas duas informações são alternadas periodicamente na mesma matriz, permitindo que sejam visualizadas integralmente.



Figura 32 – Nível crítico na matriz de LEDs



Figura 33 – Nível baixo na matriz de LEDs



Figura 34 – Nível médio na matriz de LEDs



Figura 35 – Nível alto na matriz de LEDs

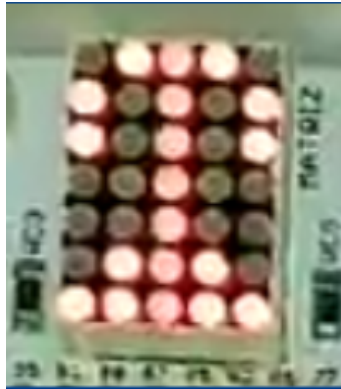


Figura 36 – Aspersão representada na matriz de LEDs



Figura 37 – Gotejamento representada na matriz de LEDs

Além disso, o estado da válvula de enchimento da caixa d'água é exibido no primeiro dígito (da esquerda para a direita) de um display de 7 segmentos. Neste contexto, a letra "A" indica que a válvula está "Aberta" e a letra "F" que está "Fechada". As figuras pertinentes mostram essas indicações. Adicionalmente, o display de 7 segmentos também exibe o temporizador, que é configurado para iniciar com 30 minutos no modo de Gotejamento e 15 minutos no modo de Aspersão, conforme ilustrado nas Figuras 38 e 39.



Figura 38 – Válvula aberta e temporizador



Figura 39 – Válvula fechada e temporizador

Em caso de conflito nos sensores, o display de 7 segmentos é capaz de mostrar uma mensagem de erro, alertando sobre a anomalia detectada. Esta funcionalidade é útil para a identificação e resolução rápida de problemas operacionais. A Figura 40 demonstra a visualização dessa mensagem de erro.



Figura 40 – Mensagem de erro

Considerações finais

Durante o desenvolvimento do projeto, foram enfrentadas diversas dificuldades técnicas significativas. Algumas das principais envolveram a implementação dos *flip-flops*, componentes críticos para o controle de estados e temporização dentro do circuito. Outra complexidade técnica foi a implementação da funcionalidade de parada do temporizador ao atingir o valor zero, garantindo que ele cessasse corretamente suas operações.

O temporizador em si apresentou desafios adicionais, exigindo uma abordagem dedicada para assegurar seu funcionamento conforme especificado. A resolução desses problemas demandou um estudo aprofundado da literatura especializada, além de consultas contínuas com monitores e professores, cujas orientações foram fundamentais para superar as dificuldades encontradas.

Cada módulo do projeto, desde aqueles responsáveis pela matriz de LEDs até os relacionados ao temporizador, foi desenvolvido com o objetivo de atender aos requisitos especificados para o circuito. A matriz de LEDs, por exemplo, foi projetada para alternar periodicamente entre a indicação do nível da caixa d'água e o tipo de irrigação, garantindo uma representação clara das informações.

Os resultados obtidos com esses módulos indicam um desempenho satisfatório em todas as métricas avaliadas, comprovando o êxito do design e da implementação. A integração harmoniosa dos diferentes componentes demonstra a robustez do circuito desenvolvido e sua capacidade de operar dentro dos parâmetros projetados, atendendo às necessidades específicas do sistema de irrigação automatizado.

Problem 2: ... Automated Irrigation (Improvements)!

Cláudio Daniel Figueredo Peruna ^{*} Paulo Gabriel da Rocha Costa Silva [†]
Paulo Henrique Dantas Barreto [‡]

2024

Abstract

This study presents the evolution of an automated irrigation system, initially proposed, with the implementation of an enhanced circuit. The new design incorporates advanced functionalities, including a timer and an improved interface for displaying system information. The development tools Verilog and Quartus were employed for the implementation on the CPLD MAX II board. The results demonstrate the effectiveness of using resources such as synchronous and asynchronous counters, flip-flops, latches, and clock manipulation, highlighting the system's potential to optimize irrigation processes.

Key-words: automated irrigation. digital circuits. Verilog. Quartus. CPLD MAX II. digital systems. synchronous counters. asynchronous counters. flip-flops. latches. clock manipulation. agricultural automation. process optimization.

Referências

FLOYD, T. Álgebra booleana e simplificação lógica. In: _____. [S.l.]: Bookman, 2007. (Sistemas Digitais: Fundamentos E Aplicações). Citado na página [6](#).

GATES, B. *The Road Ahead*. [S.l.]: Viking Penguin, 1995. Citado na página [1](#).

^{*}danielperuna2012@gmail.com

[†]paulogrcsilva@gmail.com

[‡]rickdeuxvult@gmail.com