

Problema 1: Rega Automatizada!

Cláudio Daniel Figueiredo Peruna * Paulo Gabriel da Rocha Costa Silva †
Paulo Henrique Dantas Barreto ‡

2024

Resumo

Este relatório visa apresentar a metodologia e o desenvolvimento de um sistema automatizado de irrigação. O sistema foi projetado para otimizar o uso da água em plantações, buscando aprimorar a eficiência econômica em comparação aos métodos tradicionais de irrigação. Utilizando a linguagem de programação *Verilog estrutural* e o *software Quartus*, o sistema foi implementado em uma placa *CPLD MAX II*. Os resultados alcançados confirmaram o sucesso do projeto, demonstrando a viabilidade técnica e a eficácia operacional do sistema proposto. Além disso, o projeto contribuiu para a consolidação de conhecimentos teóricos relacionados a Sistemas Digitais, corroborando com a literatura existente e indicando caminhos para futuras pesquisas e aplicações práticas no campo da automação agrícola.

Palavras-chaves: irrigação automatizada. circuitos digitais. verilog. quartus. CPLD MAX II. eficiência hídrica. sistemas digitais. tecnologia agrícola.

Introdução

A eficiência agrícola é um tema de suma importância para a sociedade e estudado há anos. Um exemplo é a Revolução Verde, que foi uma proposta iniciada na década de 30 buscando revolucionar o sistema agrícola internacional ao aprimorar sua produção, segundo (ZAMBENEDETTI et al., 2021). Para atingir seus objetivos, essa proposta visava o uso de técnicas modernas como o uso de organismos geneticamente melhorados, uso de mecanização, como também o uso de irrigação automatizada.

Pensando nestes benefícios, este projeto tem como objetivo de desenvolver um protótipo de sistema automático de irrigação. Para tal, foi utilizado o *Kit CPLD-LEDS*, que utiliza a placa *CPLD MAX II* para então simular o ambiente e interface de interação

*danielperuna2012@gmail.com

†paulogrcsilva@gmail.com

‡rickdeuxvult@gmail.com

com o usuário. Os interruptores foram usados para simular os sensores de níveis da caixa d'água e de ambiente, assim como a escolha da opção do menu pelo usuário, o barramento de LEDs para simular a abertura das válvulas, o LED RGB para o alarme, e o *display* de sete segmentos, para o painel de informações.

1 Metodologia

1.1 Ferramentas Utilizadas

O *Kit de Desenvolvimento LEDS-CPLD* fora utilizada para a realização do projeto, a placa utilizada é a *EPM240T100C5N* da família *MAX II*; a linguagem de descrição de hardware utilizada fora o *Verilog HDL*, e seu compilador, o *Quartus II* da *Intel* nas versões 18 e 20; Já como editor de código, fora utilizado o próprio *Quartus II* e, em certos momentos, o *Visual Studio Code* da *Microsoft*. *Python 3.10* foi utilizado como ferramenta auxiliar para gerar os arquivos de configuração e limpar o projeto após compilado, assim como o *Git* fora usado para versionamento do código, e o *Github* como repositório remoto.

1.1.1 Visão Geral

Para facilitar a compreensão da arquitetura do sistema proposto, recomenda-se a análise do fluxograma geral do circuito, apresentado na Figura 1.

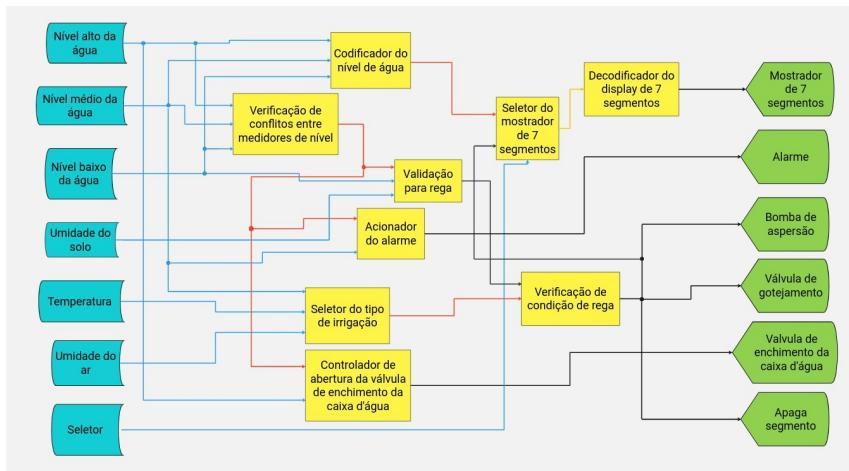


Figura 1 – Visão Geral do projeto

Através deste, é possível identificar a estratégia de modularização adotada, que consiste na verificação sistemática dos níveis da caixa d'água (alto, médio e baixo). Esta verificação é realizada por meio de um algoritmo de detecção de conflitos entre os sensores, assegurando que as combinações de sinais sejam coerentes e livres de erros, antes de prosseguir para a etapa de codificação dos dados.

A saída do processo de verificação é direcionada para um controlador responsável pela operação da válvula de enchimento da caixa d'água, que também é influenciado pelo sensor de nível alto. Paralelamente, o seletor de irrigação processa entradas provenientes do sensor de nível médio da água, além dos sensores de temperatura e umidade do ar, para determinar as condições adequadas de irrigação.

O sistema de alarme é acionado com base no nível médio da água e nos resultados da verificação de conflitos, alertando sobre quaisquer discrepâncias ou condições críticas. Finalmente, o decodificador do display de 7 segmentos interpreta os sinais do seletor para exibir informações pertinentes ao tipo de irrigação empregado ou ao nível atual da caixa d'água.

1.1.2 Imagens da Placa

A interface do usuário do sistema é projetada para exibir uma variedade de informações através de um *display* de 7 segmentos, um LED RGB e uma barra de LEDs. Esses componentes fornecem uma representação visual das condições operacionais do sistema, conforme descrito na Seção 1.3 (Pinagem).

Quando o *display* de 7 segmentos é configurado para indicar o nível da água na caixa, quatro estados distintos podem ser representados: nível crítico (Figura 2), nível baixo (Figura 3), nível médio (Figura 4) e nível alto (Figura 5). Cada estado é associado a uma imagem específica no *display*, permitindo uma identificação rápida e intuitiva do status atual da caixa d'água.

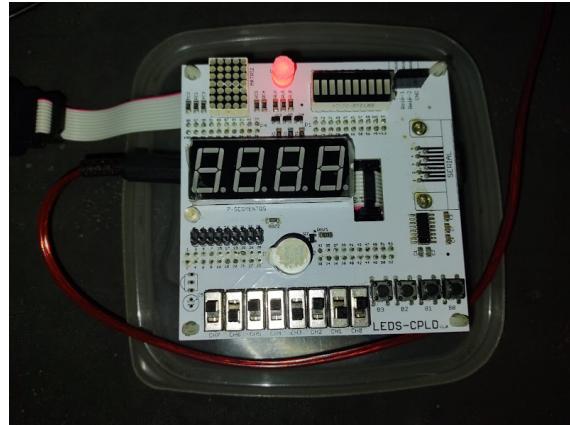


Figura 2 – Nível crítico da caixa d'água

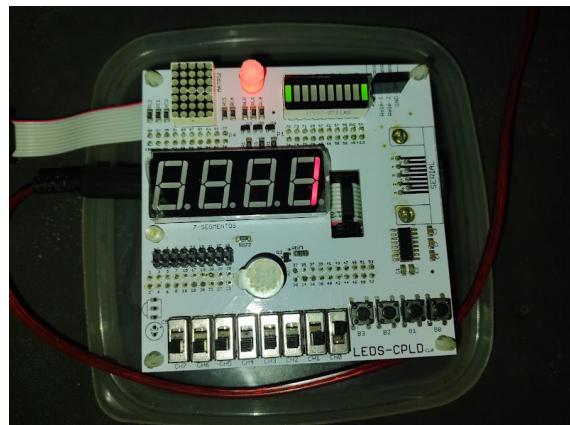


Figura 3 – Nível baixo da caixa d'água

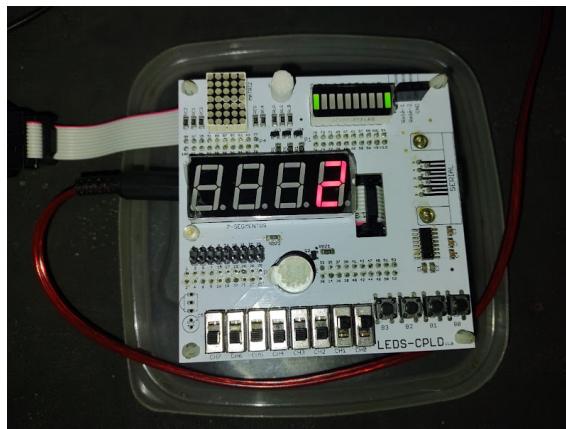


Figura 4 – Nível médio da caixa d’água

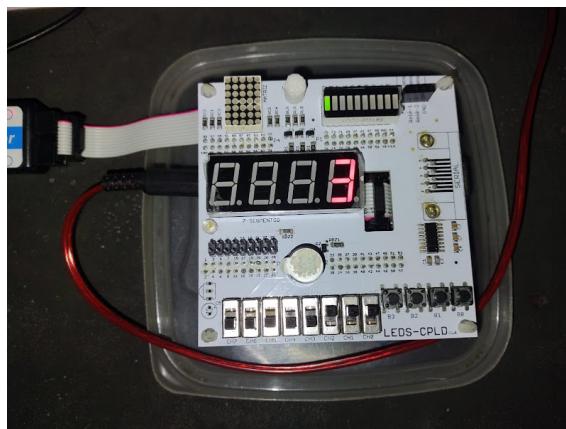


Figura 5 – Nível alto da caixa d’água

Além disso, o seletor do *display* pode ser ajustado para mostrar o modo de irrigação em operação. Neste caso, três imagens são possíveis: a representação da irrigação por aspersão (Figura 6), a irrigação por gotejamento, e uma indicação do estado do alarme (ativo, na Figura 7 ou inativo, na Figura 8), que é determinado pela análise dos sinais dos sensores. É importante notar que os modos de irrigação são visualizados não apenas no *display* de 7 segmentos, mas também na barra de LEDs, proporcionando uma dupla confirmação do tipo de irrigação selecionado.

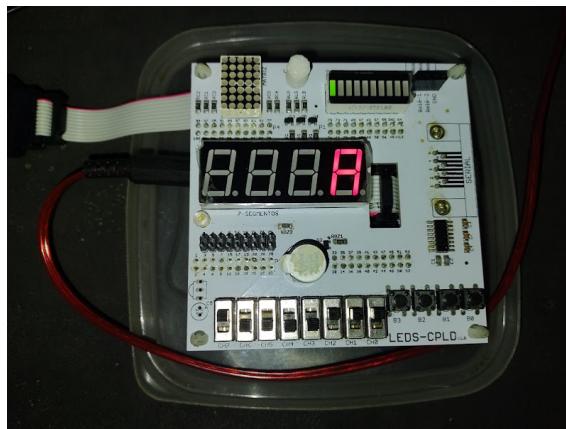


Figura 6 – Bomba de Aspersão ativada

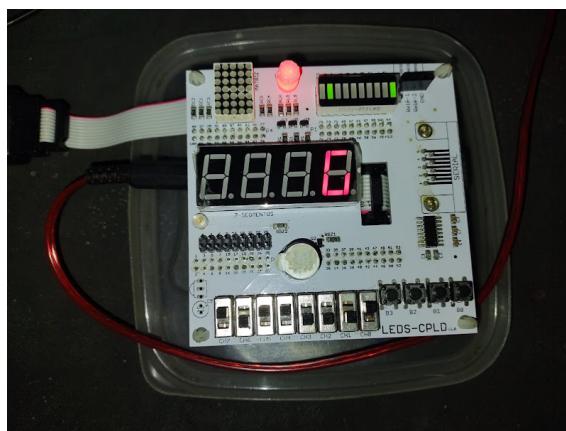


Figura 7 – Válvula de Gotejamento ativada, com Alarme ligado

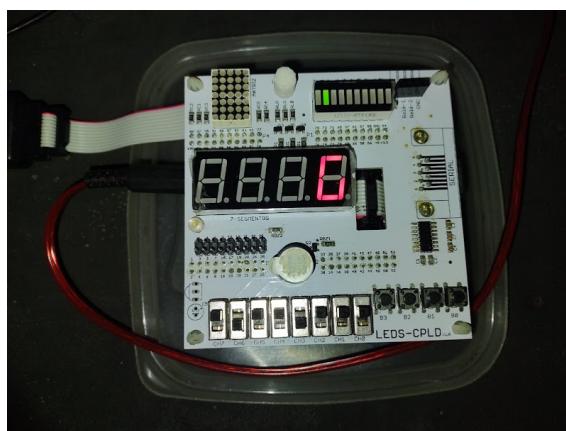


Figura 8 – Válvula de Gotejamento ativada, com Alarme desligado

1.2 Escrita do código e Documentação

Na implementação do código-fonte, adotou-se uma metodologia que prioriza a legibilidade e a manutenção do software. Utilizaram-se variáveis semânticas para garantir clareza e autoexplicabilidade, minimizando assim a necessidade de comentários adicionais. A redução de ruído foi alcançada por meio da eliminação de elementos supérfluos e da utilização de nomes para instâncias apenas quando estritamente necessário, seguindo as melhores práticas de codificação.

Em relação à documentação, cada módulo foi meticulosamente documentado, descrevendo suas funcionalidades e o papel que desempenham dentro da arquitetura do sistema. Esta documentação detalhada é fundamental para a compreensão do projeto como um todo, facilitando a integração de novos colaboradores e a expansão futura do sistema.

1.3 Módulos

Optou-se pela modularização do problema em questão, visando uma série de benefícios intrínsecos a essa abordagem. A divisão em módulos distintos permite não apenas uma manutenção mais eficiente, mas também facilita a visualização e otimização do projeto como um todo. Os módulos propostos são delineados a seguir, cada um com suas respectivas funções e interfaces claramente definidas, permitindo assim uma integração coesa e uma colaboração interdisciplinar efetiva.

1.3.1 Alarme

O módulo de alarme, *alarm.v*, é responsável por indicar baixo nível de água ou conflitos nos sensores da caixa d'água. Ele possui duas entradas: uma que indica se o nível de água está abaixo do nível médio, *mid_water_level*, e outra que indica se há valores conflitantes dos sensores da caixa d'água, *conflicting_values*. A saída do módulo, representado por *alarm_on*, é um sinal que controla o estado do alarme.

Sua representação em tabela verdade pode ser visto na Tabela 1, assim como sua simplificação usando produto das somas pode ser visto na Equação (1) e seu esquema pode ser visto na Figura 9.

<i>mid_water_level</i>	<i>conflicting_values</i>	<i>alarm_on</i>
0	0	1
0	1	1
1	0	0
1	1	1

Tabela 1 – Tabela verdade do módulo *alarm.v*

$$\text{alarm_on} = \text{mid_water_level}' + \text{conflicting_values} \quad (1)$$

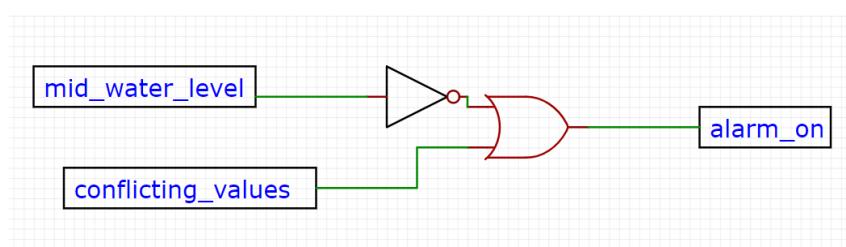


Figura 9 – Esquema do alarme

1.3.2 Verificadores de conflitos nos sensores da caixa d'água

O módulo *water_sensors_checker.v* tem como finalidade verificar se existem valores conflitantes entre os sensores de nível de água do sistema. Esta verificação é crucial para garantir a precisão das leituras dos níveis de água e evitar operações inconsistentes do sistema de irrigação.

O módulo possui três entradas correspondentes aos diferentes sensores de nível de água: baixo (*low_level*), médio (*mid_level*) e alto (*high_level*). Sua saída é o sinal *conflict*, que indica se há conflito entre os valores dos sensores.

Logo abaixo estão a tabela verdade em Tabela 2 do módulo, assim como sua simplificação usando soma de produtos na Equação (2). Para fins didáticos, a equação utiliza outros nomes mais simples para se referir às entradas. Por tanto, A é equivalente a *high_level*, B a *mid_level*, C a *low_level* e Z a *conflict*. Seu esquema pode ser visto na Figura 10.

<i>high_level</i>	<i>mid_level</i>	<i>low_level</i>	<i>conflict</i>
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Tabela 2 – Tabela verdade do módulo water sensors checker.v

$$\begin{aligned}
 Z &= A'BC' + AB'C' + AB'C + ABC' \\
 Z &= A'BC' + A(B'C' + B'C) + ABC' \\
 Z &= A'BC' + AB' + ABC' \\
 Z &= C'(A'B + AB) + AB' \\
 Z &= C'B + AB' \equiv A'B + B'C
 \end{aligned} \tag{2}$$

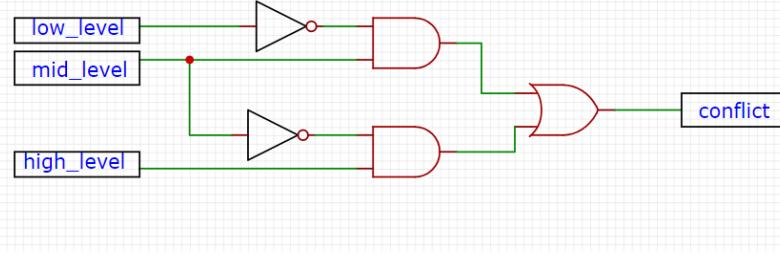


Figura 10 – Esquema da verificação de conflitos entre os sensores da caixa d’água

1.3.3 Controlador do suprimento da caixa d’água

O módulo *water_supply_controller* é responsável por controlar a válvula de fornecimento de água em um sistema. Sua função é garantir que a válvula permaneça fechada em determinadas condições específicas, conforme as seguintes regras:

A válvula deve permanecer fechada se houver valores conflitantes provenientes dos sensores de nível de água. Isso é necessário para evitar operações inconsistentes do sistema de irrigação que podem resultar de leituras imprecisas dos sensores.

A válvula deve permanecer fechada quando o reservatório de água estiver cheio, ou seja, quando o nível de água estiver alto. Isso é essencial para evitar o desperdício de água e possíveis danos causados por transbordamento.

Sua representação na tabela verdade pode ser vista em *Tabela 3*, e sua simplificação por meio de soma de produtos pode ser encontrada na *Equação (3)*, no código em *Verilog*, porém, a lógica é implementada por meio de uma porta *NOR*, isso se deve à simplificação por meio do Teorema de DeMorgan como demonstrado por ([FLOYD, 2007b](#)), e seu esquema na *Figura 11*.

<i>water_sensors_conflicting</i> ,	<i>high_water_level</i>	<i>valvule</i>
0	0	1
0	1	0
1	0	0
1	1	0

Tabela 3 – Tabela verdade do módulo *water_supply_controller.v*

$$valvule = water_sensors_conflicting' \times high_water_level'$$

Aplicando Teorema de DeMorgan (3)

$$valvule = (water_sensors_conflicting + high_water_level)'$$

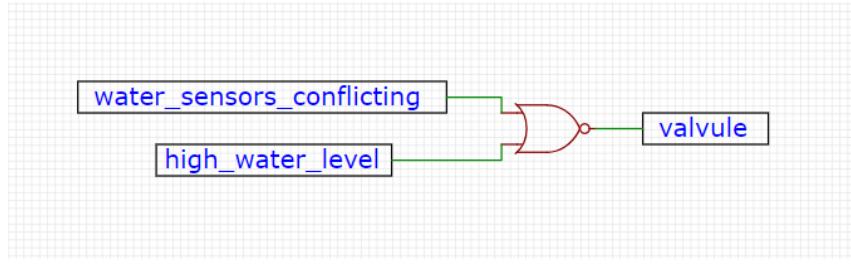


Figura 11 – Esquema da abertura entrada de água

1.3.4 Codificador do nível da caixa d'água

O módulo *Verilog* denominado *water_encoder* tem como função codificar o nível de água em um sistema, atribuindo bits a diferentes estados de nível de água. Essa codificação é essencial para o funcionamento do decodificador, possibilitando a representação adequada do nível de água em dispositivos de exibição, como demonstrado por (FLOYD, 2007a).

O módulo possui três entradas correspondentes aos diferentes estados de nível de água: alto (*high*), médio (*mid*) e baixo (*low*). As saídas do módulo consistem em dois bits codificados (*encoded_water_Bit0* e *encoded_water_Bit1*), que representam os diferentes níveis de água de acordo com a codificação especificada.

O representação da tabela verdade do módulo pode ser encontrada na Tabela 4, e sua simplificação na Equação (4) e (5), usando o método de soma de produtos.

high	mid	low	bit0	bit1
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Tabela 4 – Tabela verdade do módulo *water_encoder.v*

$$bit0 = high' \times mid' \times low + high \times mid \times low \quad (4)$$

$$\begin{aligned}
 bit1 &= high' \times mid \times low + high \times mid \times low \\
 bit1 &= (mid \times low)(high' + high) \\
 bit1 &= mid \times low
 \end{aligned} \quad (5)$$

1.3.5 Controlador de irrigação

O módulo *irrigation_controller* é responsável por verificar os pré-requisitos necessários para o funcionamento do sistema de irrigação. Esses pré-requisitos devem ser

atendidos para permitir a ativação dos dispositivos de irrigação, como o gotejador e o aspersor. As condições que o sistema de irrigação deve obedecer são as seguintes: a) Todos os sensores devem estar funcionando corretamente. b) O nível de água não deve estar crítico. c) O solo deve estar seco.

A tabela verdade do mesmo pode ser encontrada na Tabela 5, sua simplificação na Equação (6), usando soma de produtos e seu esquema encontrado na Figura 12.

<i>water_sensor_conflicting</i>	<i>earth_humidity</i>	<i>low_water_level</i>	<i>irrigation_off</i>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Tabela 5 – Tabela verdade do módulo *irrigation_controller.v*

$$\text{irrigation_off} = \text{water_sensor_conflicting}' \times \text{earth_humidity}' \times \text{low_water_level} \quad (6)$$

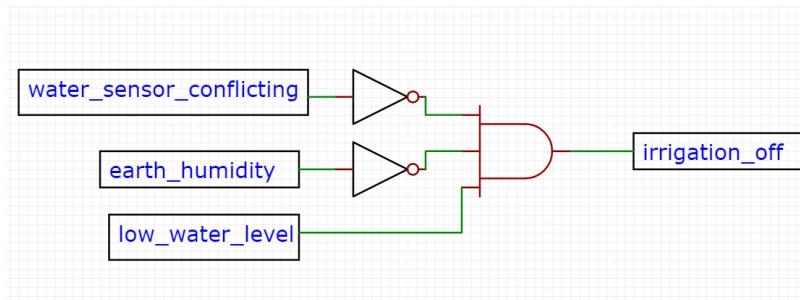


Figura 12 – Esquema do controlador de irrigação.

1.3.6 Seletor de irrigação

O módulo *Verilog irrigation_selector* é responsável por determinar qual dos modos de irrigação deverá ser ativo, se aspersor ou gotejador. Para isso, deve ser escolhido com base em diferentes condições ambientais e de fornecimento de água. Quando sua saída é verdadeira, o modo de aspersão será ativo, do contrário, o modo de gotejamento, essa saída é representada pelo sinal *splinker_mode_on*. Existem dois cenários em que o aspersor deve ser ativado: a) Quando a umidade do ar estiver baixa. b) Quando a umidade do ar estiver alta, o clima estiver frio e o nível de água estiver acima do nível médio.

Sua tabela verdade está representada pela Tabela 6 e sua simplificação por soma de produtos, pela Equação (7), sendo que *A* representa *mid_water_level*, *B*, *low_temperature*, *C*, *air_humidity* e *Z*, a saída *splinker_mode_on*. Sua representação esquemática pode ser vista na Figura 13.

<i>mid_water_level</i>	<i>low_temperature</i>	<i>air_humidity</i>	<i>splinker_mode_on</i>
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Tabela 6 – Tabela verdade do módulo *irrigation_selector.v*

$$\begin{aligned}
 Z &= A'B'C + A'BC' + A'B'C' + AB'C + ABC' \\
 Z &= A'C'(B' + B') + A'B'(C' + C) + ABC' \\
 Z &= A'C' + AB' + ABC' \\
 Z &= A'C' + A(B' + BC') \\
 Z &= A'C'' + AB' + AC' \\
 Z &= C'(A' + A) + AB' \\
 Z &= AB' + C'
 \end{aligned} \tag{7}$$

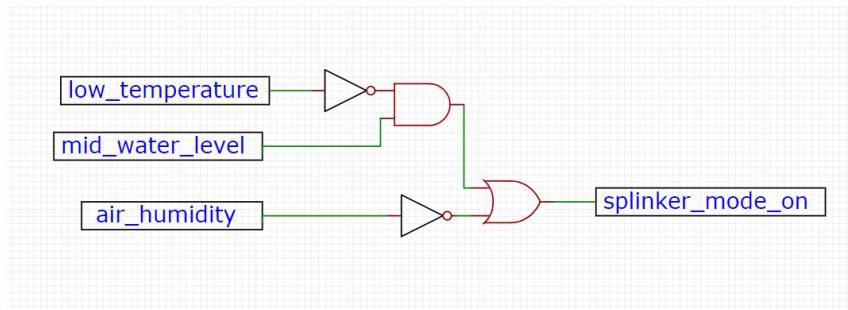


Figura 13 – Esquema do seletor de irrigação

1.3.7 Multiplexador 2x1

O módulo *Verilog multiplexer_2x1.v* é um componente que seleciona entre dois sinais de entrada (*l0* e *l1*) com base no valor de um sinal de seleção *s*, assim como descrito por (FLOYD, 2007a). O sinal de saída *out* é definido de acordo com o valor de *s*: se *s* for 0, o sinal de saída será igual a *l0*; se *s* for 1, o sinal de saída será igual a *l1*. O processo de seleção é realizado internamente no módulo, garantindo que apenas um dos sinais de entrada seja propagado para o sinal de saída, dependendo do valor de *s*. Seu esquema em diagrama pode ser encontrado na Figura 14.

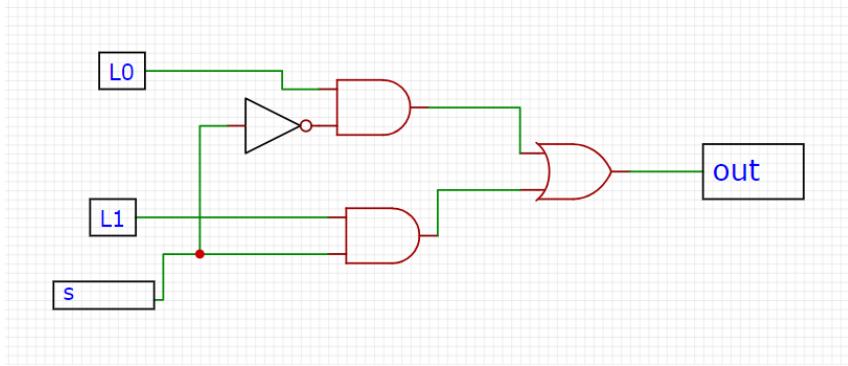


Figura 14 – Esquema do Multiplexador 2x1

1.3.8 Desabilitador de *display*

O módulo *disable_display* é responsável por desabilitar cada grupo do *display* de sete segmentos. Para isso, fora utilizada uma propriedade de simplificação booleana que diz que $A'A = 1$, como pode ser visto na Tabela 7, portanto o resultado sempre será nível lógico alto, independentemente do valor de A . Seu esquema pode ser encontrado na Figura 15.

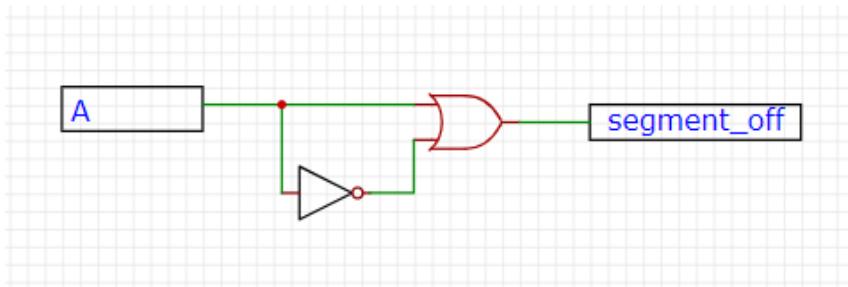


Figura 15 – Esquema do desabilitador de display

A	segment_off
0	1
1	1

Tabela 7 – Tabela verdade do módulo *disable_display.v*

1.3.9 Seletor do modo de *display*

O módulo *display_selector* é um seletor de *display*, implementado como um multiplexador (MUX) em Verilog. Ele é responsável por selecionar os dados que serão exibidos em um *display* de sete segmentos com dois bits de entrada. O sinal *selector* determina qual conjunto de dados será enviado para a exibição. Quando *selector* está em estado alto, os bits de entrada *encoded_irrigation_Bit1* e *encoded_irrigation_Bit0* são selecionados e enviados para os segmentos do display correspondentes. Quando *selector* está em estado baixo, os bits de entrada *encoded_water_Bit1* e *encoded_water_Bit0* são selecionados e enviados para o *display*. Esse processo permite alternar entre a exibição dos dados de

irrigação e de água no *display*, fornecendo ao usuário as informações necessárias sobre o sistema de irrigação em tempo real.

1.3.10 Decodificador do *display*

O módulo *display_decoder* é responsável por converter os bits de entrada em sinais individuais para cada segmento de um *display* de sete segmentos. Utilizando a lógica booleana, ele interpreta os sinais de entrada *entry_Bit0* e *entry_Bit1*, juntamente com o sinal de controle *selector*, para determinar quais segmentos devem ser ativados para representar o valor de entrada. Cada segmento (A, B, C, D, E, F, G) é calculado individualmente com base nos valores dos bits de entrada e no seletor. As operações lógicas de *NOT*, *AND* e *OR* são utilizadas para calcular a ativação de cada segmento de acordo com a combinação de entrada, garantindo que a representação no *display* seja precisa e corresponda ao valor desejado.

Sua tabela verdade pode ser vista na Tabela 8, onde *A* é o bit menos significativo e *B* é o mais significativo, ao lado, tem-se as saídas para cada segmento do *display*, enumerados de *a* a *g*. Já a simplificação de suas expressões podem ser vistas da Equação (8) a Equação (14), vale ressaltar que fora utilizado soma de produtos como método principal, exceto pela função *e*, que usou produto de somas. Já a visão esquemática geral pode ser vista na Figura 16.

A	B	C	a	b	c	d	e	f	g
0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	0	1	1	1	1
0	1	0	0	0	1	0	0	1	0
0	1	1	0	0	0	0	1	1	0
1	0	0	1	1	1	1	1	1	1
1	0	1	0	0	0	1	0	0	0
1	1	0	0	1	0	0	0	0	1
1	1	1	1	1	1	1	1	1	1

Tabela 8 – Tabela verdade do módulo *display_decoder.v*

$$\begin{aligned}
 a &= A'B'C' + A'B'C + AB'C' + A'B'C' + ABC \\
 a &= A'B'(C' + C) + A(B'C' + BC) + ABC \\
 a &= A'B' + AB'C' + ABC \\
 a &= B'(A' + AC') + ABC \\
 a &= B'(A' + C') + ABC \\
 a &= B'A' + BC' + ABC \equiv ABC + B'C' + A'B'
 \end{aligned} \tag{8}$$

$$\begin{aligned}
 b &= A'B'C' + AB'C' + ABC' + ABC \\
 b &= A'B'C' + AB'C' + AB(C' + C) \\
 b &= A'B'C' + AB'C' + AB \\
 b &= B'C'(A' + A) + AB \\
 b &= B'C' + AB
 \end{aligned} \tag{9}$$

$$\begin{aligned}
c &= A'B'C' + A'B'C' + AB'C' + ABC \\
c &= A'C'(B' + B') + AB'C' + ABC \\
c &= A'C' + AB'C' + ABC \\
c &= C'(A' + AB') + ABC \\
c &= C'(A' + B') + ABC \\
c &= A'C' + B'C' + ABC
\end{aligned} \tag{10}$$

$$\begin{aligned}
d &= A'B'C' + A'B'C + AB'C + ABC \\
d &= A'B'(C' + C) + AB'(C' + C) + ABC \\
d &= A'B' + AB' + ABC \\
d &= B'(A + A') + ABC \\
d &= B' + ABC \\
d &= B' + AC
\end{aligned} \tag{11}$$

$$\begin{aligned}
e &= A'B'C' + A'B'C + A'BC + AB'C' + ABC \\
e &= A'B'(C' + C) + BC(A' + A) + AB'C' \\
e &= A'B' + BC + AB'C' \\
e &= B'(A' + AC') + BC \\
e &= B'(A' + C') + BC \\
e &= B'A' + B'C' + BC \equiv B'C' + A'C + BC
\end{aligned} \tag{12}$$

$$\begin{aligned}
f &= (A' + B + C')(A' + B' + C) \\
f &= (A'A + A'B' + A'C) + (A'B + B'B + BC) + (A'C' + B'C' + CC'') \\
f &= (0 + A'B' + A'C) + (A'B + 0 + BC) + (A'C' + B'C' + 0) \\
f &= A'B' + A'C + A'B + BC + A'C' + B'C' \\
f &= A'(B' + B) + A'(C + C') + BC + B'C' \\
f &= A' + A' + BC + B'C' \\
f &= A' + BC + B'C'
\end{aligned} \tag{13}$$

$$\begin{aligned}
g &= A'B'C' + A'B'C + AB'C' + ABC' + ABC \\
g &= A'B'(C' + C) + AB'C' + AB(C' + C) \\
g &= A'B' + AB'C' + AB \\
g &= B'(A' + AC') + AB \\
g &= B'(A' + C') + AB \\
g &= A'B' + B'C' + AB
\end{aligned} \tag{14}$$

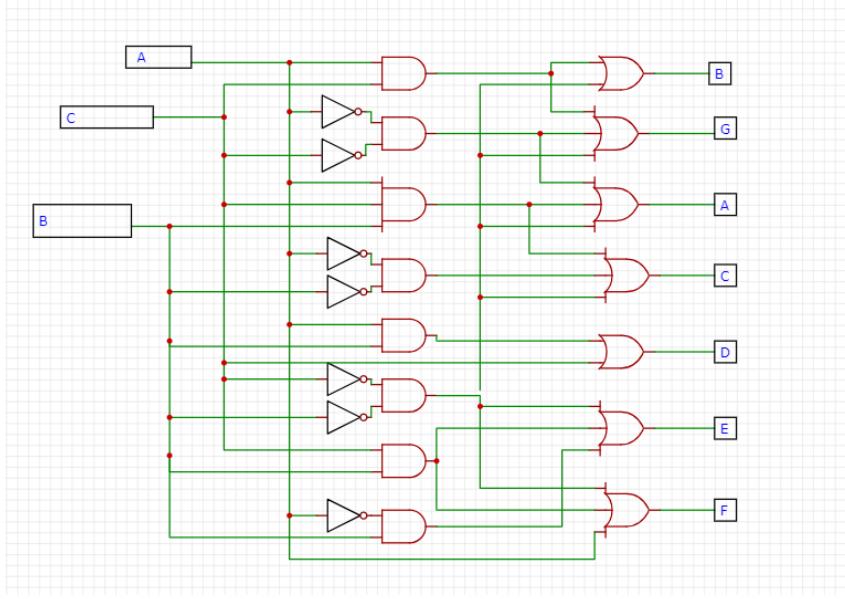


Figura 16 – Esquema geral do *Display* de 7 Segmentos

1.4 Pinagem

Para representar as diferentes situações do circuito, como a abertura das válvulas e os níveis de água na caixa, foram cuidadosamente selecionados os pinos mais apropriados na placa. A seguir, descrevemos os pinos selecionados para representar cada uma dessas situações.

Válvula de Entrada: A abertura da válvula de entrada (*Ve*) é representada visualmente pelo *LED0*, localizado no pino 54 da placa.

Para representar tanto os níveis da caixa d'água quanto os dados dos sensores de umidade do ar, do solo e de temperatura, optou-se por utilizar as *Chaves HH* da placa. Desta forma, para garantir o funcionamento completo do circuito, foram atribuídas sete chaves no total.

Os níveis baixo, médio e alto da caixa d'água são representados respectivamente pelas chaves: *CH0*, *CH1* e *CH2*, localizadas nos pinos 42, 38 e 30. Já os sensores de ambiente, como é o caso dos sensores de umidade do solo, umidade do ar e temperatura foram representados respectivamente pelas chaves *CH3*, *CH4* e *CH5*, localizadas nos pinos 36, 34 e 30.

Além disso, o último pino foi dedicado para alternar o modo de exibição do display de 7 segmentos (pino 35), permitindo mostrar o nível da caixa d'água ou o tipo de irrigação (podendo ser aspersão ou gotejamento), conforme selecionado pelo usuário.

As regas de aspersão e Gotejamento foram visualmente representadas em dois locais distintos da placa. Inicialmente, optou-se por indicar a ativação dessas regas nos *LEDs LED9* (pino 76) para aspersão e *LED8* (pino 75) para gotejamento, proporcionando uma visualização direta e acessível. Esta escolha permitiu a rápida identificação da ativação das regas, antes mesmo da codificação do *display* de 7 segmentos, facilitando o desenvolvimento do projeto como um todo.

Já o alarme foi integrado ao circuito utilizando o *LED RGB*, configurado para

emitir luz vermelha, localizado no *pino 86*.

Além disso, o *display* de segmentos é empregado para representar os tipos de rega, conforme mencionado anteriormente, e o nível da caixa d'água. Os segmentos do dígito 3 (último dígito) são utilizados para essa finalidade, com os seguintes pinos associados: *39, 41, 70, 90, 91, 98 e 100*. Não obstante, fora necessário desabilitar partes do *display*, como é o caso do ponto e de dos *displays* à esqueda, para isso, ativando os pinos *66, 68, 88 e 96*.

1.5 Ferramentas Auxiliares

No desenvolvimento do projeto, foram implementadas duas ferramentas auxiliares utilizando a linguagem de programação *Python*, versão 3.10. A primeira ferramenta, denominada *clean.py*, tem como função principal a remoção de todos os arquivos temporários gerados durante o uso do software *Quartus II*. Esta operação é essencial para manter a integridade do ambiente de desenvolvimento e evitar a propagação de inconsistências durante o processo de compilação.

A segunda ferramenta, *update_config.py*, é projetada para automatizar a atualização do arquivo de configuração do *Quartus*. Ela opera lendo um arquivo no formato *TOML* e extrai as variáveis necessárias para configurar adequadamente o ambiente de compilação. As variáveis extraídas incluem a pinagem, o diretório de origem dos módulos e as especificações do dispositivo alvo.

O *script* gerado por esta ferramenta é capaz de configurar a compilação paralela do compilador, definir o dispositivo de destino, adicionar os módulos necessários conforme especificado no diretório de origem, além de configurar a pinagem. Com isso, elimina-se a necessidade de interação manual com o *Pin Planner* do *Quartus*, otimizando o fluxo de trabalho e reduzindo a margem de erro humano.

2 Resultados

O circuito sintetizado é relativamente simples, consistindo principalmente de operações lógicas combinacionais e utilizando uma quantidade moderada de recursos do CPLD. Não há armazenamento de estados ou operações sequenciais no circuito, o que pode ser observado pelo fato de não haver nenhum registro (*Register only*) no projeto.

Assim, ao analisar os resultados da síntese do circuito no *Quartus* é revelado que o projeto utiliza um total de 14 elementos lógicos (LEs) em um *Complex Programmable Logic Device* (CPLD). Todos esses LEs são empregados em operações combinacionais, sem a presença de *flip-flops* ou outros elementos de armazenamento. A distribuição dos LEs por número de entradas nas Tabelas de Verdade mostra uma predominância de operações com múltiplas entradas. Em relação aos pinos de entrada/saída (*I/O pins*), o circuito faz uso de 22 deles. Além disso, o circuito possui um total de 60 *fan-outs*, com uma média de 1.67 *fan-outs* por nó.

Em relação aos resultados esperados, o projeto alcançou sucesso em todas as métricas estabelecidas, satisfazendo integralmente os requisitos técnicos para o circuito. Este êxito não apenas demonstra a eficácia do sistema automatizado de irrigação desenvolvido, mas também reforça a importância da adequada preparação e capacitação técnica na execução de projetos complexos em Sistemas Digitais.

Considerações finais

No desenvolvimento do projeto, os desafios mais significativos estiveram relacionados ao processo de aprendizagem das tecnologias e metodologias aplicadas. Embora estes desafios sejam intrínsecos ao processo de pesquisa e desenvolvimento, eles não são o foco principal deste relatório.

A concepção do circuito abordou diversos aspectos cruciais para seu funcionamento eficiente. Cada módulo, desde o alarme até o controlador de irrigação, foi cuidadosamente projetado para atender aos requisitos específicos do sistema. A análise dos resultados da síntese do circuito no *Quartus* demonstrou que o projeto alcançou sucesso em todas as métricas estabelecidas, utilizando um número moderado de recursos do CPLD. Essa eficiência confirma a importância da preparação técnica adequada na execução de projetos complexos em Sistemas Digitais. A integração harmoniosa dos diferentes módulos resultou em um sistema robusto e confiável, capaz de atender às necessidades de irrigação residencial de forma sustentável e eficaz.

Problema 1: Rega Automatizada!

Cláudio Daniel Figueiredo Peruna * Paulo Gabriel da Rocha Costa Silva †
Paulo Henrique Dantas Barreto ‡

2024

Abstract

This report aims to present the methodology and development of an automated irrigation system. The system was designed to optimize water usage in crops, aiming to enhance economic efficiency compared to traditional irrigation methods. Using the Verilog structural programming language and *Quartus software*, the system was implemented on a *CPLD MAX II* board. The achieved results confirmed the project's success, demonstrating the technical feasibility and operational effectiveness of the proposed system. Additionally, the project contributed to the consolidation of theoretical knowledge related to Digital Systems, aligning with existing literature and suggesting paths for future research and practical applications in the field of agricultural automation.

Key-words: automated irrigation. digital circuits. Verilog. Quartus. CPLD MAX II. water efficiency. digital systems. agricultural technology.

Referências

FLOYD, T. Funções da lógica combinacional. In: _____. [S.l.]: Bookman, 2007. (Sistemas Digitais: Fundamentos e Aplicações), p. 312–364. Citado 2 vezes nas páginas [9](#) e [11](#).

FLOYD, T. Álgebra booleana e simplificação lógica. In: _____. [S.l.]: Bookman, 2007. (Sistemas Digitais: Fundamentos E Aplicações), p. 207–209. Citado na página [8](#).

*danielperuna2012@gmail.com

†paulogrcsilva@gmail.com

‡rickdeuxvult@gmail.com

ZAMBENEDETTI, L. et al. Revolução verde: História e impactos no desenvolvimento agrícola. In: AGRICULTURA E AGROINDÚSTRIA NO CONTEXTO DO DESENVOLVIMENTO RURAL SUSTENTÁVEL. Editora Científica, 2021. (AGRICULTURA E AGROINDÚSTRIA NO CONTEXTO DO DESENVOLVIMENTO RURAL SUSTENTÁVEL), p. 370–377. Disponível em: <<https://downloads.editoracientifica.com.br/articles/210705219.pdf>>. Citado na página 1.