

Sistema de Aquecimento para Beteiras

Rafael Feijó Leonardo
Graduação em Engenharia Eletrônica
Universidade de Brasília
Gama, DF, Brasil
goldcard99@hotmail.com

Resumo—Este documento apresenta a proposta de desenvolvimento de um sistema de controle de temperatura *PID* para aquários de pequeno porte, utilizando uma *FPGA* para a correção do erro da temperatura *setpoint*.

I. JUSTIFICATIVA

Com a evolução do aquarismo, eletrônicos diversos foram desenvolvidos buscando reproduzir o *habitat* natural dos peixes.

Dentre estes dispositivos, o aquecedor se tornou um dos mais populares. Isso devido ao fato de que a temperatura corporal do peixe é determinada pela água ao seu redor e, esta por sua vez, determina o metabolismo do animal. Ou seja, para baixas temperaturas o metabolismo desacelera, reduzindo o apetite, que por sua vez baixa a imunidade do peixe, dentre outros problemas característicos.

Entretanto, dentre os diversos modelos de aquecedores disponíveis no mercado, para aquários de até 25 Litros é difícil achar modelos de controle automático. Ou seja, a maioria dos disponíveis devem ser utilizados com um termômetro, em que o usuário deixa aquecer até a temperatura desejada e desliga o aparelho.

Nesse contexto, essa proposta visa desenvolver um sistema automático de controle, utilizando a correção por *PID*, para manutenção da temperatura da água de pequenos aquários, utilizando uma placa *FPGA Basys 3*, da *Xilinx* além de dispositivos sensor e atuador.

II. OBJETIVOS

Como objetivo, tem-se o desenvolvimento de um sistema de aquecimento, para aclimação de aquários de até 25 Litros, utilizando um algoritmo *PID*, em tempo real, embarcado em uma *FPGA*.

III. REQUISITOS FUNCIONAIS

- 1) Configuração da Temperatura *Setpoint*
O sistema deverá permitir a configuração da temperatura *setpoint* através de uma palavra de 32 bits, configurada via *IP VIO Core*.
- 2) Indicador *On/Off*
O sistema deverá possuir um *LED* indicativo do estado do sistema: ligado ou desligado.

- 3) Indicador *IDLE/WORKING*
O sistema deverá possuir um *LED* indicativo do estado de controle: *setpoint* atingido (*IDLE*) ou aquecendo (*WORKING*).
- 4) Cálculo *PID*
O sistema deverá ser capaz de realizar o cálculo do erro proporcional, integrativo e derivativo da temperatura recebida do microcontrolador.
- 5) Configuração do Ganho *Kp*, *Ki* e *Kd*
O sistema deverá permitir a configuração dos ganhos das parcelas proporcional, integrativa e derivativa do algoritmo.
- 6) Visualização da Temperatura Real
O sistema deverá apresentar a temperatura relativa do aquário, em tempo real, nos *displays 7 segmentos* da placa *FPGA*.
- 7) *Low Power Mode*
O sistema deverá possuir um botão que, se pressionado, desativará o algoritmo *PID* e colocará o microcontrolador em modo de baixo consumo.

IV. REQUISITOS NÃO FUNCIONAIS

- 1) Sensor de Temperatura
Sensor *DS18B20*, com interface *OneWire*, para aferição da temperatura relativa da água do aquário.
- 2) Microcontrolador
Microcontrolador *NodeMCU*, como ponte entre o sensor de temperatura e a placa *FPGA*, através de comunicação serial.
- 3) *FPGA*
Placa *Basys 3* responsável pelo cálculo do algoritmo *PID* e controle do *driver* do aquecedor.
- 4) Aquecedor
Pastilha *Peltier* como componente aquecedor.

V. DESCRIÇÃO DO MODELO

A modelagem do sistema em *hardware* é descrita pelas Figuras 1 e 2.

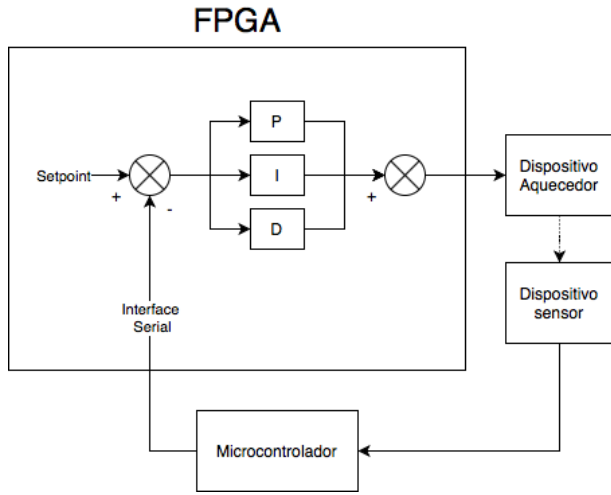


Figura 1. Diagrama modular do sistema de controle proposto.

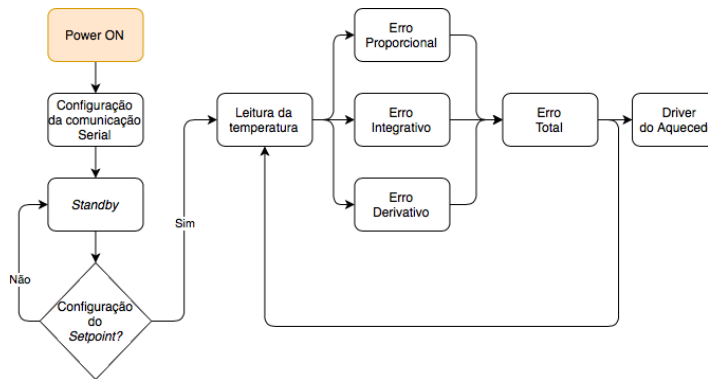


Figura 2. Fluxograma geral do sistema de controle proposto.

VI. METODOLOGIA

Como metodologia, o projeto foi dividido em 6 etapas, descritas abaixo.

- 1) Projeto de *Hardware* para conexão do microcontrolador, sensor, aquecedor e *FPGA*.
- 2) Aquisição de dados de temperatura e interface serial (*FPGA* - Microcontrolador).
- 3) Algoritmo PID: Bloco Proporcional.
- 4) Algoritmo PID: Bloco Integrador.
- 5) Algoritmo PID: Bloco derivativo.
- 6) Integração entre os blocos e *driver* do aquecedor.

Em cada etapa, serão realizadas simulações comportamentais que comprovem o funcionamento do bloco para posterior implementação em *hardware*.

VII. CRONOGRAMA

Tabela I
CRONOGRAMA PROPOSTO PARA O PROJETO.

| Semana | Data | PC | Meta |
|--------|---------------|----|------|
| 0 | 12/10 à 16/10 | X | 1 |
| 1 | 19/10 à 23/10 | | 2 |
| 2 | 26/10 à 30/10 | | 3 |
| 3 | 02/11 à 06/11 | | 3 |
| 4 | 09/11 à 13/11 | X | 4 |
| 5 | 16/11 à 20/11 | | 4 |
| 6 | 23/11 à 27/11 | | 5 |
| 7 | 30/11 à 04/12 | | 5 |
| 8 | 07/12 à 11/12 | | 6 |
| 9 | 14/12 à 18/12 | X | - |

VIII. DESENVOLVIMENTO

Para o desenvolvimento do projeto, foram criados *scripts* em *Octave* para simulação automática de cada um dos módulos projetados. Além disso, foram dispostos vídeos demonstrativos na pasta do projeto.

A. Projeto de Hardware

A Interface projetada, para comunicação entre o microcontrolador e a *Basys 3*, é mostrada na Figura 3.

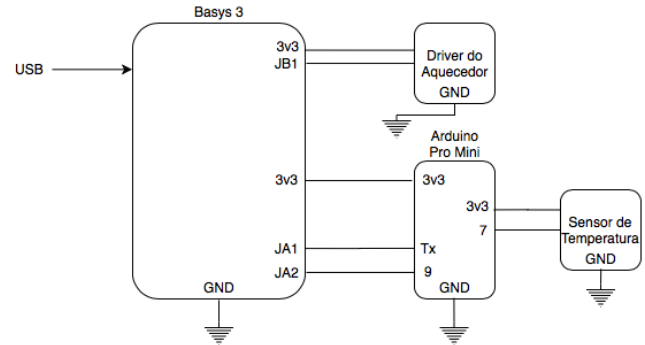


Figura 3. Interface Microcontrolador X FPGA.

O microcontrolador utilizado foi um *Arduino Pro Mini*, por questão de disponibilidade.

À seguir, são mostrados os esquemáticos do *driver* do aquecedor e o circuito do sensor de temperatura.

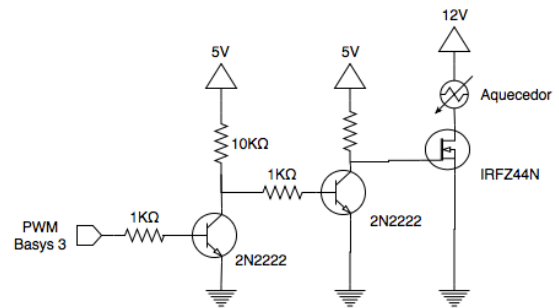


Figura 4. Esquemático do *Driver* de Potência do aquecedor.

Como componente aquecedor, foram utilizadas 2 Pastilhas Peltier 12706, em paralelo.

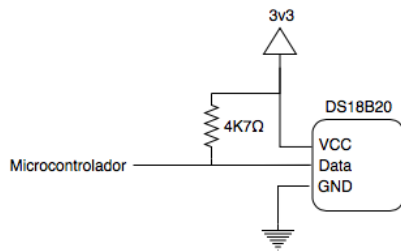


Figura 5. Esquemático de conexão entre o microcontrolador e o sensor de temperatura.

Para aferição da temperatura, foi utilizado o sensor DS18B20, à prova d'água.

B. Aquisição da Temperatura Relativa

Para aquisição da temperatura no Arduino, foi utilizada a biblioteca "DallasTemperature.h".

```
36 bus.requestTemperatures();
37
38 float t = bus.getTempCByIndex(0);
```

Figura 6. Printscreen do trecho de código em que se obtém a temperatura, em float.

Após armazenar a temperatura na variável 't', o valor é codificado em 1 byte, utilizando o formato Ponto Fixo (Figura 7), para envio via barramento UART.

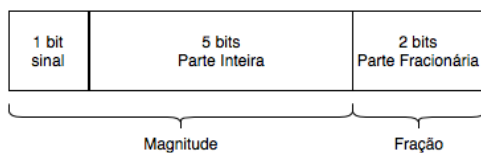


Figura 7. Modelo utilizado para transmissão da temperatura via interface Serial.

Dessa forma, é possível enviar valores entre -31.75 e +31.75. Entretanto, considerando a temperatura ideal para betoiras, a faixa utilizada foi de 0°C à 31.75°C.

```
50 void send_UART(float t)
51 {
52     // Convert temperature to unsigned fixed point
53     // (6 bits magnitude + 2 bits decimal)
54     int mag = (int)t;
55     int dec = (int)((t - mag) * 100);
56
57     if (dec == 0 || dec < 13)
58         dec = 0;
59     else if (dec > 12 && dec < 37)
60         dec = 1;
61     else if (dec > 36 && dec < 63)
62         dec = 2;
63     else if (dec > 62 && dec < 87)
64         dec = 3;
65     else
66     {
67         dec = 0;
68         mag += 1;
69     }
70 }
```

Figura 8. Printscreen do trecho de código em que o float é codificado em 1 byte, utilizando o formato Ponto Fixo.

```
71 // Check if magnitude is 5 bits length
72 if (mag < 0.0 || mag >= 32.0)
73     return;
74
75 // Mount fixed point byte
76 byte data;
77
78 data = (mag << 2) | dec;
79
80 // Send data via UART
81 Serial.write(data);
82
83 digitalWrite(LED_BUILTIN, HIGH);
84 delay(10);
85 digitalWrite(LED_BUILTIN, LOW);
86 }
```

Figura 9. Continuação do Printscreen do trecho de código em que o float é codificado em 1 byte, utilizando o formato Ponto Fixo.

Por fim, considerando que as variações de temperatura devem ser suaves para garantir o bem estar dos animais, foi adotado uma frequência de amostragem de 1Hz.

C. Interface Serial - Arduino X FPGA

Uma vez que o Arduino escreve um byte no barramento e gera um pulso no LED integrado (pino 9), a FPGA inicia o processamento.

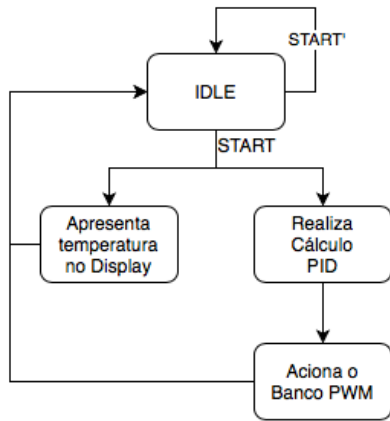


Figura 10. Esboço do fluxograma geral do sistema desenvolvido em hardware.

Na FPGA, o byte é recebido via *IP Core "serialcom"* e, para apresentação nos displays 7 segmentos, segue o fluxo:

- 1) Inverte o byte recebido;
- 2) Separa a parte inteira da parte fracionária;
- 3) Codifica a parte inteira para 7 segmentos;
- 4) Codifica a parte fracionária para 7 segmentos;
- 5) Multiplexa os 4 dígitos do display;

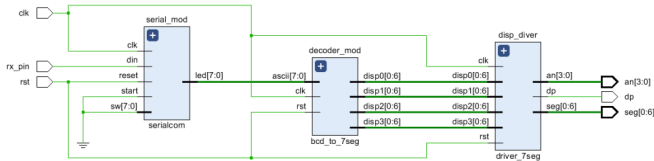


Figura 11. *Printscreen* do esquemático RTL gerado para a Interface Arduino X Basys3.

D. Algoritmo PID

O controle Proporcional/Integral/Derivativo (PID) é uma técnica que, à partir de um sinal de entrada e um *setpoint*, faz com que o erro seja minimizado pela ação proporcional, zerado pela ação integral e obtido com uma velocidade antecipativa pela ação derivativa.

No caso deste projeto, o microcontrolador envia um pulso de *start* sempre que disponibilizar novos dados de temperatura. Assim sendo, a FPGA poderá operar com seu clock em máxima velocidade, 100MHz.

Inicialmente, através do *VIO CORE* são configurados os parâmetros: *setpoint*, *kP*, *kI* e *kD*, todos em Ponto Flutuante simples (32 bits), inicializados em 25.0, 1.0, 1.0 e 1.0, respectivamente.

Ademais, ao receber a temperatura em Ponto Fixo, o módulo irá converter essa medida para Ponto Flutuante simples (32 bits), através do *IP Core "Floating Point"*.

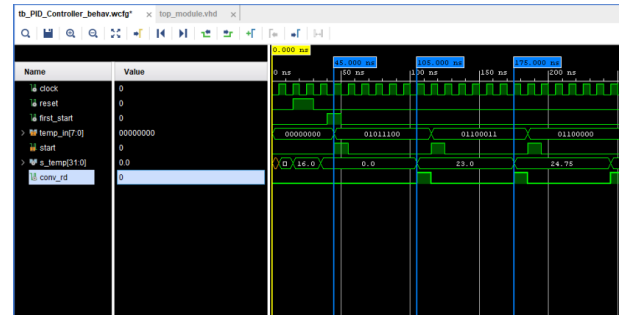


Figura 12. *Printscreen* da simulação comportamental da conversão de ponto fixo para ponto flutuante simples. Note latência de 60 ns e *throughput* de 14 MFLOPS.

Na sequência, é calculado o erro:

$$error = setpoint - realTemperature$$

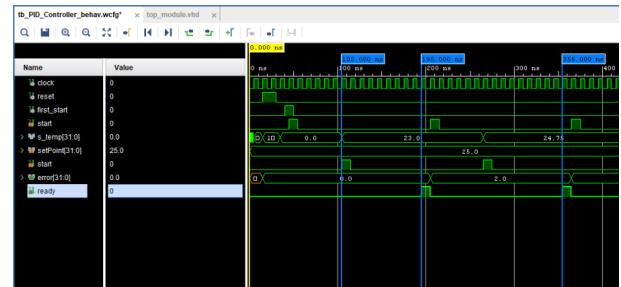


Figura 13. *Printscreen* da simulação comportamental do cálculo do erro. Note latência de 90 ns e *throughput* de 6 MFLOPS.

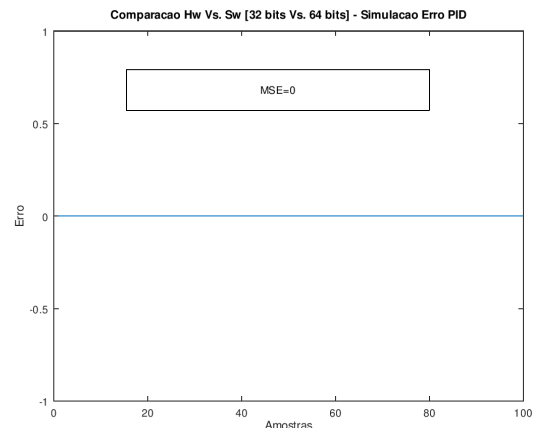


Figura 14. Gráfico comparativo entre *Hardware* (32 bits) e *Software* (64 bits) para o cálculo do erro. Note erro nulo.

1) Algoritmo PID: Bloco Proporcional

O Bloco Proporcional é modelado conforme a equação:

$$P = error * kP$$

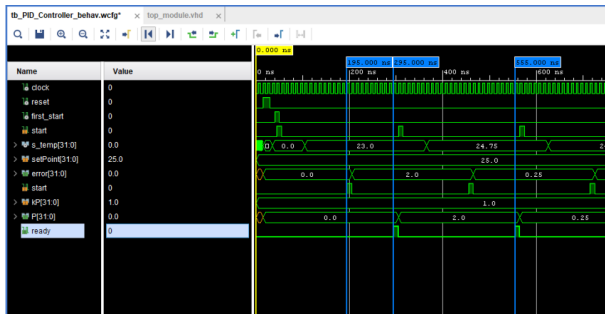


Figura 15. *Printscreen* da simulação comportamental do cálculo Proporcional. Note latência de 100 ns e *throughput* de 4 MFLOPS.

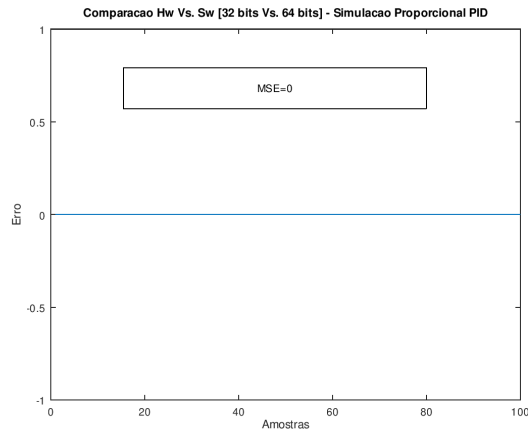


Figura 16. Gráfico comparativo entre *Hardware* (32 bits) e *Software* (64 bits) para o cálculo do bloco proporcional. Note erro nulo.

2) Algoritmo PID: Bloco Integrativo

O Bloco Integrativo é modelado conforme a equação:

$$I = I + error * kI * dT$$

Para a frequência de amostragem de 1Hz, $dT = 1$ e, portanto:

$$I = I + error * kI$$

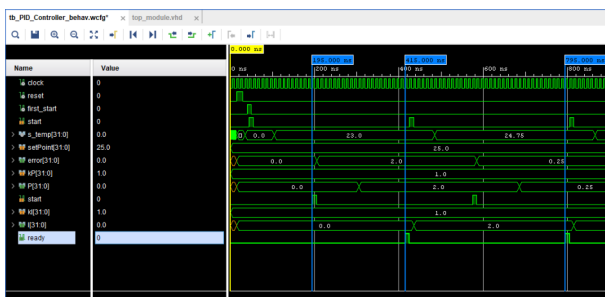


Figura 17. *Printscreen* da simulação comportamental do cálculo Integrativo. Note latência de 220 ns e *throughput* de 3 MFLOPS.

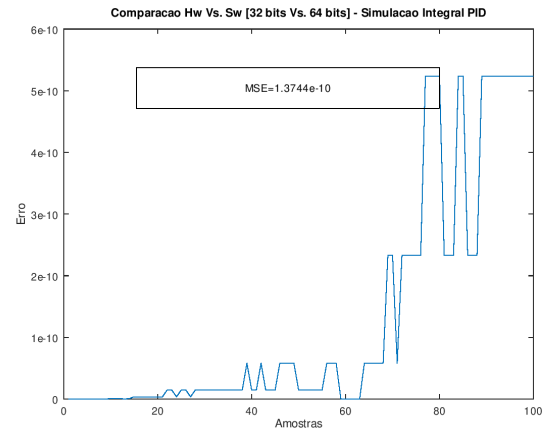


Figura 18. Gráfico comparativo entre *Hardware* (32 bits) e *Software* (64 bits) para o cálculo do bloco integrativo. Note erro máximo de 1.3744×10^{-10} .

3) Algoritmo PID: Bloco Derivativo

O Bloco Derivativo é modelado conforme a equação:

$$D = (D - error) * kI * dT$$

Para a frequência de amostragem de 1Hz, $dT = 1$ e, portanto:

$$D = (D - error) * kI$$

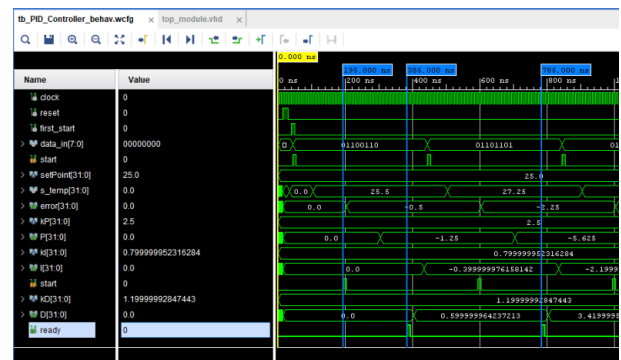


Figura 19. *Printscreen* da simulação comportamental do cálculo Derivativo. Note latência de 190 ns e *throughput* de 2.5 MFLOPS.

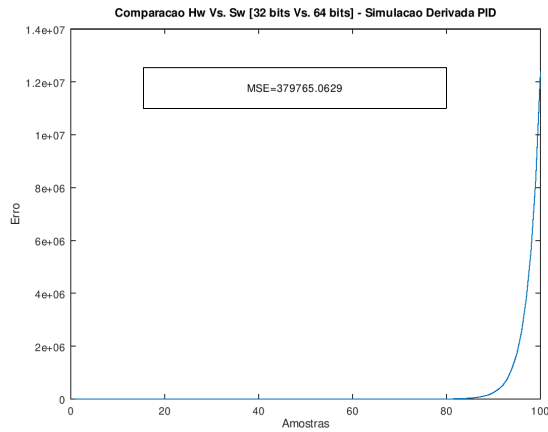


Figura 20. Gráfico comparativo entre *Hardware* (32 bits) e *Software* (64 bits) para o cálculo do bloco derivativo. Note erro máximo de 3.797×10^5 .

4) Algoritmo PID: Bloco Somador

O Bloco Somador é modelado conforme a equação:

$$PID = P + I + D$$

onde P, I e D são variáveis obtidas à partir dos blocos anteriormente citados.

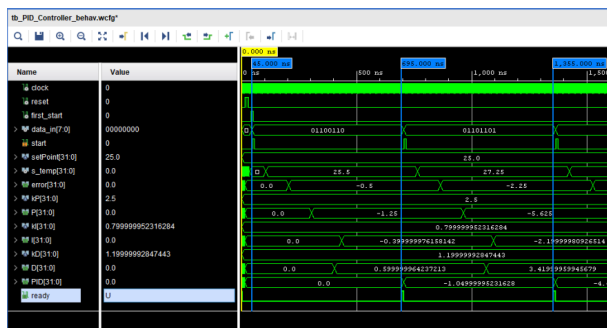


Figura 21. Printscreen da simulação comportamental do cálculo PID. Note latência de 650 ns e *throughput* de 1.5 MFLOPS.

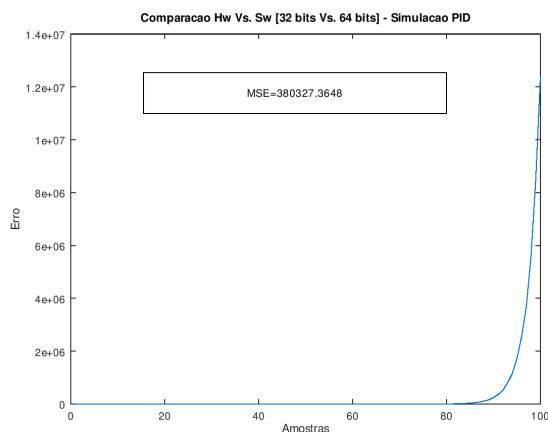


Figura 22. Gráfico comparativo entre *Hardware* (32 bits) e *Software* (64 bits) para o cálculo PID. Note erro máximo de 3.803×10^5 .

5) Algoritmo PID: Bloco Conversor

Por fim, o "Bloco Conversor" é responsável por converter a saída PID para um *duty cycle* válido, conforme mostrado na Tabela II.

Tabela II
RELAÇÃO DE CONVERSÃO PID X *Duty Cycle*.

| PID | Código | <i>Duty Cycle</i> |
|----------------------|--------|-------------------|
| < 0.5 | 000 | 0% |
| $0.5 \leq PID < 1.5$ | 001 | 10% |
| $1.5 \leq PID < 2.5$ | 010 | 25% |
| $2.5 \leq PID < 3.5$ | 011 | 40% |
| $3.5 \leq PID < 4.5$ | 100 | 50% |
| $4.5 \leq PID < 5.5$ | 101 | 60% |
| $5.5 \leq PID < 6.5$ | 110 | 75% |
| ≥ 6.5 | 111 | 100% |

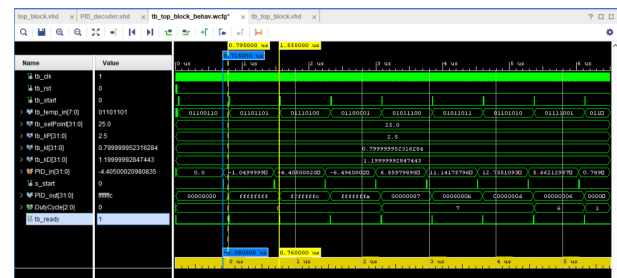


Figura 23. Printscreen da simulação comportamental do conversor PID X *Duty Cycle*. Note latência de 80 ns e *throughput* de 1.3 MFLOPS.

6) Algoritmo PID: Integração

Nesta etapa, foram adicionados os *IPs VIO Core* e o *ILA* para validação do sistema desenvolvido.

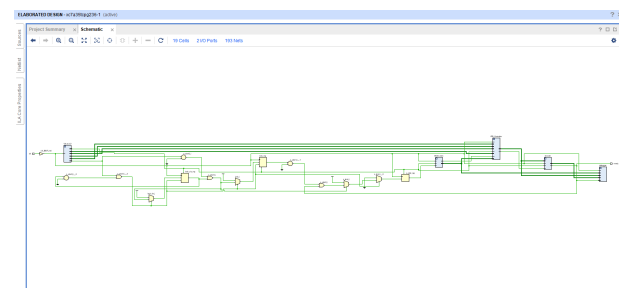


Figura 24. Printscreen do esquema RTL gerado após inclusão dos *VIO Core* e *ILA*.

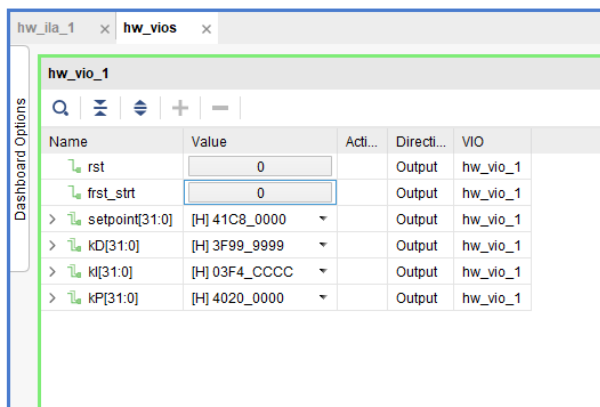


Figura 25. Printscren das configurações do VIO Core.

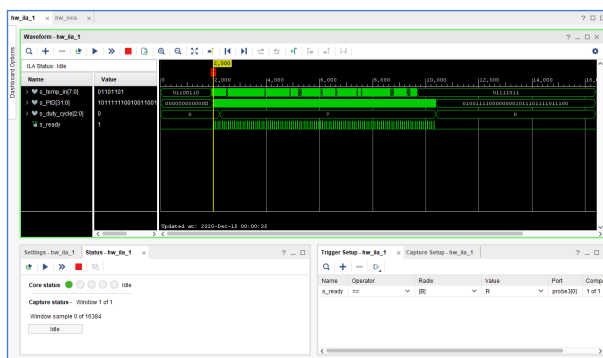


Figura 26. Printscren da saída obtida pelo ILA.

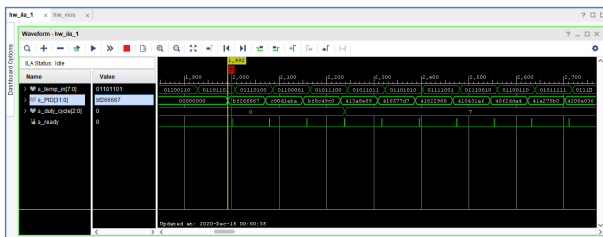


Figura 27. Zoom do printscren da saída obtida pelo ILA. Note a transição dos valores na saída PID.

Por fim, o IP ILA Core foi retirado e o sistema foi implementado.

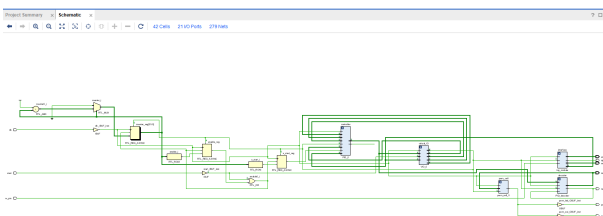


Figura 28. Printscren do esquemático RTL gerado.

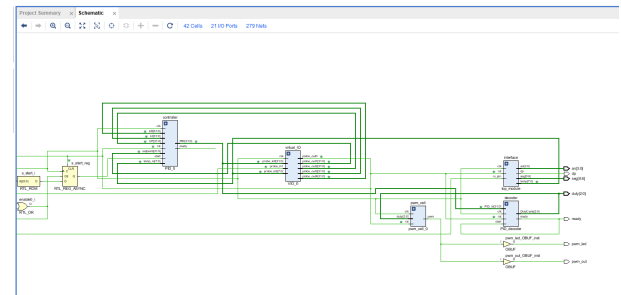


Figura 29. Zoom do printscren do esquemático RTL gerado.

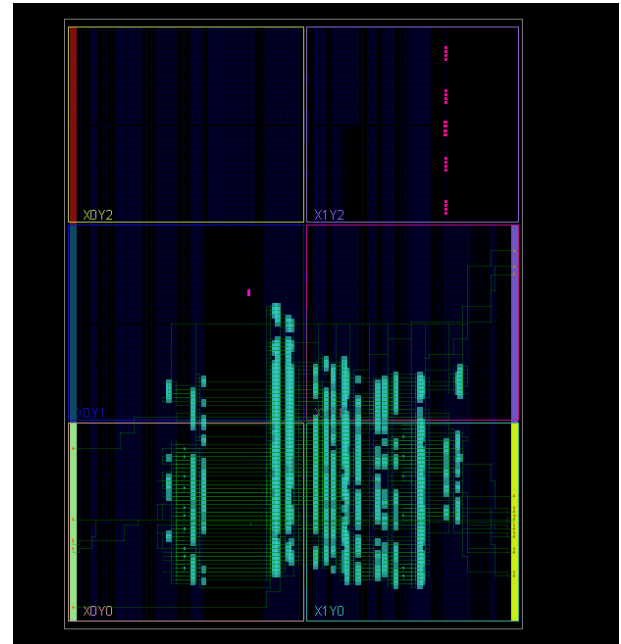


Figura 30. Printscren do layout gerado, pós place and route.

| Setup | Hold | Pulse Width |
|--------------------------------------|----------------------------------|---|
| Worst Negative Slack (WNS): 4.088 ns | Worst Hold Slack (WHS): 0.013 ns | Worst Pulse Width Slack (WPWS): 3.750 ns |
| Total Negative Slack (TNS): 0.000 ns | Total Hold Slack (THS): 0.000 ns | Total Pulse Width Negative Slack (TPWS): 0.000 ns |
| Number of Failing Endpoints: 0 | Number of Failing Endpoints: 0 | Number of Failing Endpoints: 0 |
| Total Number of Endpoints: 9488 | Total Number of Endpoints: 9472 | Total Number of Endpoints: 5301 |

All user specified timing constraints are met.

Figura 31. Printscren do report de timing do sistema implementado.

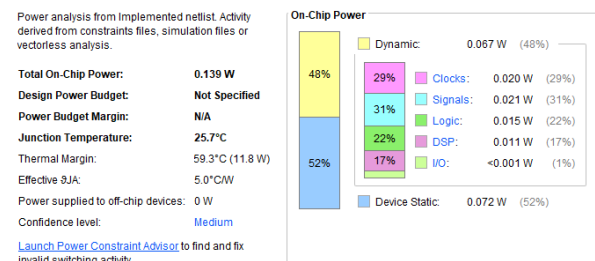


Figura 32. Printscren do report de consumo energético. Note consumo total de, 139mW.

Tabela III
DESCRITIVO DE CONSUMO DE RECURSOS.

| Bloco | LUTs | FFs | IOs | Blocos DSP |
|-----------------|------------|------------|----------|------------|
| Top Module | 2637 (13%) | 5170 (12%) | 21 (20%) | 16 (18%) |
| Vio Core | 231 (1%) | 730 (2%) | 0 (0%) | 0 (0%) |
| PID | 1732 (8%) | 3365 (8%) | 0 (0%) | 16 (18%) |
| Decoder PID/PWM | 200 (1%) | 303 (1%) | 0 (0%) | 0 (0%) |
| PWM Cell | 36 (1%) | 33 (0%) | 0 (0%) | 0 (0%) |

Não foram utilizados Blocos BRAM.

Por fim, um vídeo foi gravado para demonstração prática do sistema e pode ser acessado em: <https://www.youtube.com/watch?v=dOSDXR83vMM>.