Desenvolvimento de um Controle Sem Fio para Jogo Pong via Bluetooth Low Energy

Bruno Vinicius Machado Castanho, Jamerson Muniz, e Victor Hugo de Oliveira Carvalho

***Resumo*—Neste projeto foi realizado o desenvolvimento de um sistema embarcado, composto por uma placa FPGA com saída VGA e um microcontrolador que se comunica com um aplicativo web através de uma comunicação *bluetooth*. A lógica do jogo, incluindo a geração de sinal de vídeo, foi implementada em VHDL numa placa DE10-Lite. O sistema de controle, utiliza um microcontrolador para realizar a comunicação entre um aplicativo *Web* e a placa FPGA através de uma interface de cliente para se comunicar diretamente com o hardware através da API Web Bluetooth. Este projeto demonstra a viabilidade de se desacoplar a lógica de um jogo (executada em hardware dedicado como uma FPGA) da sua interface de controle, construindo interfaces homem-máquina (IHM) responsivas e acessíveis. O sistema final permite que um jogador controle a raquete do jogo (movimentos para cima e para baixo) ao pressionar botões em uma página da web, com os comandos sendo transmitidos via Bluetooth Low Energy (BLE) e traduzidos em sinais digitais de nível lógico baixo nas portas GPIO do microcontrolador, que por sua vez servem de entrada para a FPGA.**

***Index Terms*—Sistemas Embarcados, FPGA, ESP32, Bluetooth Low Energy (BLE), Web Bluetooth, VHDL, GATT.**

I. Introdução

A ARQUITETURA completa do sistema, foi dividida em duas frentes principais e independentes: a implementação do jogo na plataforma FPGA, juntamente com a saída de vídeo, e o desenvolvimento do sistema de controle sem fio, que consiste no *firmware* do dispositivo embarcado (servidor BLE) e na aplicação cliente (interface web).

II. Materiais

O desenvolvimento do projeto foi realizado com uma combinação de hardwares embarcados e de lógica reconfigurável, juntamente com ambientes de desenvolvimento e linguagens de programação.

A. Componentes de Hardware

A plataforma física do projeto é composta por três elementos principais.

Placa de Desenvolvimento FPGA DE10-Lite, como mostrado na Figura 1 Este componente, equipado com um FPGA Intel MAX 10, foi o núcleo do sistema de jogo. Toda a lógica do Pong, incluindo a geração de sinal de vídeo no padrão VGA (800x600), a movimentação dos elementos (raquetes e bola) e a contagem de placar, foi implementada diretamente no hardware da placa utilizando VHDL.

Os autores são estudantes de Engenharia Eletrônica na Universidade Tecnológica Federal do Paraná (UTFPR), Câmpus Toledo, PR, Brasil.

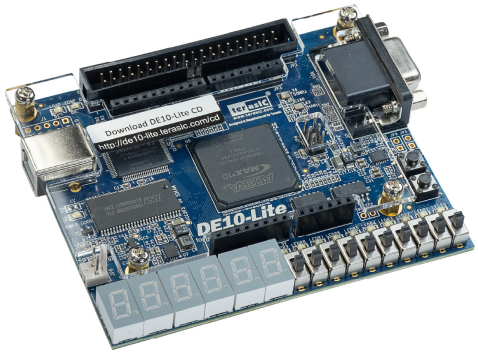

Figura 1. Placa de Desenvolvimento FPGA DE10-Lite

Módulo Microcontrolador ESP32-DevKitC, como mostrado na Figura 2. Atuando como a ponte de comunicação sem fio, o ESP32 foi responsável por receber os comandos do jogador. Suas principais funcionalidades no projeto foram atuar como um servidor BLE (Bluetooth Low Energy), expondo um serviço GATT customizado, e traduzir os comandos recebidos em sinais digitais de nível lógico nas suas portas GPIO.

Figura 2. Placa de desenvolvimento Esp32 DevKitC

Componentes de Interface e Proteção. Foram utilizados resistores de 330 Ω para a limitação de corrente e proteção na interface entre os pinos de saída do ESP32 (3.3V) e os pinos de entrada da FPGA. A montagem do circuito foi realizada em uma protoboard padrão, com jumpers para as conexões.

B. Software e Ferramentas de Desenvolvimento

O desenvolvimento do software e da lógica de hardware foi realizado utilizando as seguintes ferramentas.

Intel Quartus Prime Lite Edition. Ambiente de desenvolvimento integrado (IDE) utilizado para a escrita do código em VHDL, sua síntese, simulação funcional e programação do arquivo de configuração no dispositivo FPGA.

ESP-IDF (Espressif IoT Development Framework). Framework oficial da Espressif, utilizado para o desenvolvimento do firmware do ESP32 em linguagem C. O ambiente de desenvolvimento foi configurado no Visual Studio Code com a extensão oficial da Espressif.

Tecnologias Web. A aplicação cliente foi desenvolvida com HTML5, CSS3 (utilizando o framework Tailwind CSS para estilização rápida) e JavaScript (ES6+). A comunicação com o hardware foi possibilitada pela **Web Bluetooth API**, que permite que scripts executados no navegador interajam com dispositivos BLE próximos.

III. Aplicação Web

A aplicação web atua como o cliente GATT, sendo responsável por iniciar a comunicação e enviar os comandos para o periférico. A sua interface foi desenvolvida com HTML para a estrutura e CSS para a estilização. Toda a lógica de controle e comunicação é implementada em *JavaScript*, utilizando a *Web Bluetooth* API, que serve como interface de programação para que o navegador acesse o hardware *Bluetooth* do dispositivo.

O fluxo de conexão é assíncrono e iniciado pelo usuário ao clicar no botão ”Conectar”, o que dispara uma função de requerimento. Para máxima compatibilidade, a busca é configurada com `acceptAllDevices: true`, exibindo todos os periféricos BLE ao alcance, enquanto a permissão para aceder ao serviço específico é solicitada via `optionalServices`. Após a seleção do dispositivo pelo utilizador, o script executa uma cadeia de promessas (*Promise chain*) com `async/await` para conectar-se ao servidor GATT e obter as referências ao serviço e à característica de escrita.

A lógica de controle é implementada anexando múltiplos *event listeners* aos botões da interface. Os eventos `mousedown` (clique de rato) e `touchstart` (toque em ecrã) são utilizados para enviar os comandos de ativação (’1’ e ’2’). Para garantir que o comando de desativação seja enviado de forma fiável, são monitorados três eventos distintos: `mouseup` (soltar o rato), `touchend` (remover o dedo do ecrã) e `mouseleave` (arrastar o rato para fora do botão). Essa abordagem evita que a saída do hardware fique ”presa”no estado ativo.

A comunicação efetiva é realizada através do método `characteristic.writeValueWithoutResponse()`. Este método está alinhado com a configuração do ESP32, enviando os dados sem aguardar uma confirmação, o que é crucial para a baixa latência exigida em uma aplicação de controle. O dado a ser enviado (o caractere de comando) é primeiro codificado num `ArrayBuffer` utilizando um `TextEncoder` antes de ser transmitido.

A. Fluxo de envio de dados

O processo detalhado de envio de um único comando, desde a interação inicial do usuário no navegador até a efetiva transmissão do dado, é ilustrado no fluxograma da Figura 3. Conforme demonstrado, a ação física do usuário, como o evento `mousedown`, dispara um *event listener* que invoca a função JavaScript correspondente. Esta função prepara o caractere de comando (e.g., ’1’), que é então codificado para um formato binário (`ArrayBuffer`) através da interface `TextEncoder`. Finalmente, o dado formatado é enviado ao hardware pelo método `characteristic.writeValueWithoutResponse()`. Esta sequência de operações do lado do cliente foi projetada para ser executada com a máxima eficiência, minimizando a sobrecarga de processamento e contribuindo diretamente para a baixa latência do sistema de controle.

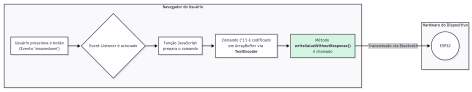

Figura 3. Diagrama de Blocos da Aplicação Web

IV. Firmware do ESP32

O software embarcado no ESP32 atua como um periférico BLE (*Bluetooth Low Energy*), implementando um servidor GATT (*Generic Attribute Profile*). A arquitetura é orientada a eventos, de modo que o fluxo principal do programa é reativo, não sequencial. Funções de *callback* são registradas para eventos específicos da pilha *Bluetooth*, e a CPU do microcontrolador permanece em estado de baixo consumo até que um evento precise ser processado.

O fluxo de inicialização começa com a ativação da NVS (*Non-Volatile Storage*) através de uma função de inicialização, pré-requisito necessário para o *Bluetooth* armazenar as chaves de pareamento. Em seguida, as portas GPIO 4 e 5 são configuradas como saídas digitais em estado inicial de nível lógico alto, para a lógica *active-low* utilizada. A inicialização do *Bluetooth Low Energy* (BLE) é uma sequência crítica de chamadas da API do ESP-IDF: a partir de uma função de inicialização é alocado os recursos básicos, `esp_bt_controller_enable()` ativa o modo dual (BLE e Clássico), `esp_bluedroid_init()` e `esp_bluedroid_enable()` preparam e ativam a pilha de software *Bluedroid*. Imediatamente após, duas funções de *callback* são registradas: `esp_ble_gap_register_event_handler()` para um *handler* que gerencia eventos de anúncio, e `esp_ble_gatts_register_event_handler()` para um *handler* dedicado aos eventos do servidor GATT, que é o núcleo da interação com o cliente.

A estrutura GATT do servidor é composta por um Serviço customizado com o UUID `0x00FF`, que por sua vez contém uma única Característica com o UUID `0xFF01`. No código, isto é definido através de uma estrutura de dados, tipicamente um array (`gatts_service_inst_t`), que descreve a ”tabela GATT”completa. Cada entrada define um serviço ou uma característica, seus UUIDs, propriedades e permissões. A propriedade da nossa característica é definida como `ESP_GATT_CHAR_PROP_BIT_WRITE_NR` (Write No Response), e as permissões como `ESP_GATT_PERM_WRITE`. Após a definição, a função

`esp_ble_gatts_create_service()` é chamada para registrar esta estrutura no BLE. Para garantir a descoberta pela API *Web Bluetooth*, foi adotada uma estratégia de *Scan Response*: o pacote de *advertising* principal anuncia o nome do dispositivo, enquanto o pacote de resposta (*scan response*) anuncia o UUID do serviço. Isso contorna o limite de 31 bytes do pacote de anúncio e robustece a descoberta.

A característica foi configurada com a propriedade *Write Without Response*. Essa decisão técnica significa que o ESP32 não envia uma confirmação (ACK) de volta para o cliente após receber um comando, o que é fundamental para minimizar a latência da comunicação. Do ponto de vista do cliente, isso permite enviar múltiplos comandos em rápida sucessão sem aguardar a conclusão de cada um, uma característica essencial para a responsividade exigida por um controle de jogo.

O núcleo da funcionalidade reside no tratamento do evento `ESP_GATTS_WRITE_EVT` dentro do *callback* do GATTS. A função de *callback* registrada recebe como parâmetros o tipo de evento e um ponteiro para uma `union` (`esp_gatts_cb_param_t`). Para um evento de escrita, o código acessa o membro `write` desta `union` (`param->write`), que contém informações como o handle da conexão e, crucialmente, um ponteiro para o valor recebido (`param->write.value`) e seu tamanho (`param->write.len`). O firmware então extrai o primeiro byte deste *payload* e utiliza uma estrutura `switch-case` para mapeá-lo a uma chamada de função `gpio_set_level()`. Os caracteres '1' e '0' ativam e desativam a saída na GPIO 5, enquanto os caracteres '2' e '3' controlam a ativação e desativação da saída na GPIO 4.

A. Fluxo de envio de dados do ESP32

O fluxo de envio de dados do Esp32, desde a função que envia os dados, até o recebimento dos dados pelo FPGA esta detalhado no fluxograma da Figura 4. O diagrama ilustra o caminho que o dado percorre até ser lido pelo programa do FPGA.

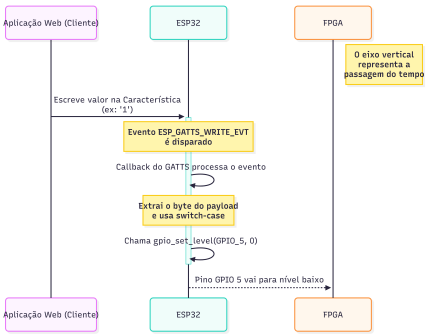

Figura 4. Diagrama de Blocos do Código de envio de dados ao ESP

V. Descrição do Hardware implementado em FPGA

A estrutura do hardware é organizada em blocos funcionais, conforme ilustrado pelo diagrama na Figura 5. Essa arquitetura, dividida em seções principais, contempla desde a temporização do sinal VGA até a lógica de movimentação dos elementos do jogo. O primeiro bloco define os parâmetros da temporização VGA, incluindo valores de sincronização horizontal e vertical, estabelecendo uma resolução efetiva de 800x600 pixels.

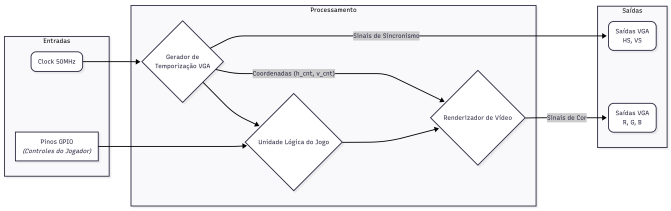

Figura 5. Diagrama de blocos da arquitetura VHDL implementada na FPGA.

A segunda seção do código define os parâmetros do jogo. São especificadas as dimensões dos elementos gráficos, como largura e altura das raquetes (barras verticais), tamanho da bola, velocidade de movimentação da bola e das raquetes, e as posições fixas das raquetes esquerda e direita na tela. Também são definidas as cores utilizadas no jogo em formato RGB 4:4:4, codificadas com 12 bits.

Na sequência, a terceira seção declara os sinais internos utilizados na geração dos sinais VGA, incluindo os contadores horizontais (h_cnt) e verticais (v_cnt) que percorrem as posições de cada pixel dentro de um quadro. Além disso, são definidos os sinais de sincronismo (hs_reg e vs_reg) e o sinal frame_tick, que indica o início de um novo quadro. A partir deste sinal de temporização é atualizado a lógica do jogo uma vez por quadro.

O estado do jogo é armazenado na quarta seção. Nela, são declaradas variáveis que representam a posição vertical das raquetes ($p1\_y$ e $p2\_y$), a posição e direção da bola ($ball\_x$, $ball\_y$, $ball\_dx$, $ball\_dy$), além do placar de cada jogador ($score\_l$ e $score\_r$). A movimentação da bola e o controle do placar dependem desses valores, que são atualizados a cada pulso de quadro.

A quinta seção do código implementa um procedimento chamado draw_rect, responsável por desenhar retângulos na tela. Este procedimento é utilizado para gerar os elementos gráficos do jogo, como as raquetes, a bola e o placar. Ele verifica se o pixel atual está dentro da área de um retângulo definido e, se sim, aplica a cor correspondente ao pixel.

A sexta parte do código realiza a geração dos sinais de sincronismo VGA. Através de um processo sensível à borda de subida do clock de 50 MHz, os contadores horizontal e vertical são atualizados, gerando os sinais de sincronismo horizontal (HS) e vertical (VS). Quando o pixel atual está dentro da região visível, o sinal $active\_xy$ é ativado. O pulso frame_tick é gerado no início de cada novo quadro, sendo este o gatilho principal para a lógica de movimentação do jogo.

A lógica central do jogo é implementada na sétima seção da descrição em VHDL. Dentro de um processo sensível à borda de subida do clock, as movimentações das raquetes e da bola são processadas apenas quando o sinal frame_tick está ativo, ou seja, uma vez por quadro. Os controles dos jogadores foram implementados utilizando pinos do vetor ARDUINO_IO, com lógica ativa em nível baixo. O jogador 1 utiliza os pinos 2 e 3

para subir e descer sua raquete, enquanto o jogador 2 utiliza os pinos 6 e 7. Esses pinos são acionados por comandos enviados a partir do ESP32, no qual uma de suas GPIOs, configurada como saída com nível lógico alto por padrão, é chaveada para nível lógico baixo a fim de sinalizar uma ação ao FPGA. Entre o ESP32 e o FPGA é inserido um resistor de proteção, com a finalidade de limitar a corrente e preservar a integridade dos dois sistemas.

A bola se move automaticamente de acordo com os valores de $ball\_dx$ e $ball\_dy$, refletindo nas bordas superior e inferior, e também ao colidir com as raquetes dos jogadores. A colisão com uma raquete altera a direção da bola e inverte sua velocidade vertical, gerando o efeito de rebote. Caso a bola ultrapasse os limites horizontais da tela, é considerado ponto do jogador adversário, atualizando o placar e reposicionando a bola ao centro.

Por fim, a oitava seção é responsável pela geração do vídeo propriamente dito. Um processo combina a posição atual dos contadores (h_cnt, v_cnt) com o estado do jogo para gerar os sinais de cor VGA_R, VGA_G e VGA_B. O processo desenha as raquetes utilizando draw_rect, centralizadas nas posições atuais dos jogadores. A bola também é desenhada como um quadrado. O placar é representado por até nove quadrados coloridos no topo da tela, à esquerda e à direita, de acordo com a pontuação de cada jogador. Os valores de cor para cada pixel são separados em componentes RGB e enviados às saídas correspondentes.

VI. Resultados e Discussão

A implementação do sistema proposto foi concluída com sucesso, resultando em um controle sem fio funcional e responsivo para o jogo Pong executado na FPGA. A integração entre os três subsistemas principais — a aplicação web, o firmware do ESP32 e a lógica em VHDL — operou conforme o planejado, validando a arquitetura de comunicação e controle. O fluxograma completo do fluxo de dados do sistema é apresentado na Figura 6.

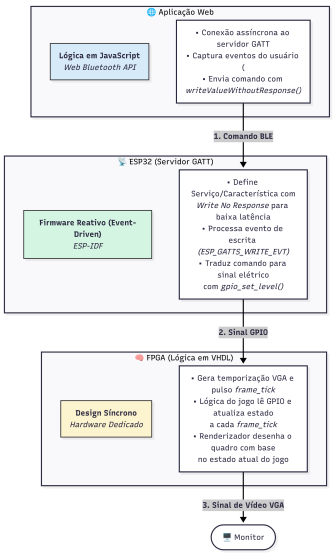

Figura 6. Diagrama de Blocos Geral do Projeto

O processo, conforme ilustrado, inicia-se na interface da Aplicação Web como mostrado na Figura 7. Os testes práticos demonstraram que a interface desenvolvida com HTML e JavaScript foi capaz de se conectar de forma estável ao ESP32 através da API Web Bluetooth. A captura dos eventos de toque e clique do usuário e o envio imediato do **Comando BLE (1)** com o método `writeValueWithoutResponse` provaram ser uma estratégia eficaz, resultando em uma latência de comando imperceptível do ponto de vista do jogador.

Figura 7. Aplicação Web do Controle do Jogo Pong

O ESP32, atuando como servidor GATT, recebeu os comandos de forma confiável. A arquitetura de firmware reativo (event-driven) baseada no ESP-IDF mostrou-se altamente eficiente. O evento `ESP_GATTS_WRITE_EVT` foi processado instantaneamente ao receber o comando BLE. A tradução deste comando em um sinal elétrico, através da chamada da função `gpio_set_level()`, ocorre sem atrasos significativos, gerando o **Sinal GPIO (2)** que serviu de entrada para a FPGA.

O FPGA, com seu design síncrono, consegue processar os sinais de entrada do ESP32 de forma precisa. A lógica do jogo, sincronizada pelo pulso `frame_tick`, lia o estado dos pinos GPIO a cada quadro, garantindo que a movimentação da raquete fosse suave e sem interrupções. A renderização do quadro, baseada no estado atualizado do jogo, resultou na geração de um **Sinal de Vídeo VGA (3)** estável e correto, exibindo o jogo no monitor de forma totalmente jogável.

Os resultados validam a arquitetura modular do projeto. A separação clara entre a interface do usuário (web), a camada de comunicação sem fio (ESP32) e o processamento de hardware de alta velocidade (FPGA) não só funcionou como se mostrou uma abordagem robusta e flexível.

VII. Conclusão

O projeto demonstrou com sucesso a integração eficaz entre diferentes plataformas de hardware (FPGA e microcontrolador) e tecnologias web modernas para criar um sistema de jogo completo, com um controle sem fio funcional, responsivo e de baixa latência. A utilização do ESP32 como servidor BLE e de uma aplicação web como cliente provou ser uma arquitetura poderosa e flexível para a interface do usuário, eliminando a necessidade de desenvolvimento de aplicativos nativos. Os desafios relacionados à descoberta de dispositivos e à estabilidade da conexão foram superados através de uma configuração cuidadosa dos pacotes de advertising do BLE e da implementação de uma lógica de busca robusta na aplicação web. O resultado final é um sistema que não apenas cumpre os seus requisitos funcionais para o controle de um jogo Pong,

mas também serve como um excelente modelo para futuros projetos que envolvam a interação entre sistemas embarcados, FPGAs e interfaces de usuário baseadas na web.

REFERÊNCIAS

[1] Ashenden, P. J. (2007). *The Designer's Guide to VHDL* (3rd ed.). Morgan Kaufmann.
[2] Espressif Systems. (2023). *ESP-IDF Programming Guide*. Recuperado de https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
[3] Gomez, C., Oller, J., & Paradells, J. (2012). Overview of the Bluetooth Low Energy Protocol. *IEEE Communications Magazine, 50*(12), 101-107.
[4] Web Bluetooth Community Group. (2023). *Web Bluetooth API Specification*. W3C. Recuperado de https://webbluetoothcg.github.io/web-bluetooth/
[5] Wolf, M. (2012). *Computers as Components: Principles of Embedded Computing System Design* (3rd ed.). Morgan Kaufmann.