BIP-ACE

Assembly Coding Environment

Eng. André Maiolini

22 de Abril de 2025

Sumário

1 Objetivo

Este relatório tem como objetivo apresentar uma introdução prática ao uso da ferramenta BIP-ACE (Assembly Coding Environment) e sua integração com o hardware BIP-FPGA - desenvolvidas sob a perspectiva de apoiar o ensino de programação em Assembly e de tópicos em Arquitetura e Organização de Computadores.

Dessa forma, o principal objetivo é permitir que estudantes traduzam conceitos teóricos em execuções reais em hardware, conectando software (código em linguagem de montagem) e resultados físicos.

Tendo isso em vista, foram priorizados os seguintes fatores:

1. **Ambiente integrado de desenvolvimento em Assembly**: oferecer um editor de código com sintaxe destacada e verificação de erros, assim como simplificar a geração de código de máquina;

2. **Promover a interação com hardware real**: habilitar o upload direto de programas para a memória do BIP-FPGA. Permitindo a execução em hardware real;

3. **Reduzir a complexidade do fluxo de trabalho**: eliminar a necessidade de ferramentas externas para geração de código e upload;

4. **Oferecer suporte pedagógico**: facilitar a criação de exercícios que conectem teoria à prática;

5. **Garantir acessibilidade e usabilidade**: fornecer uma interface intuitiva e prática para os usuários.

Tem-se em vista a diminuição da curva de aprendizagem de assuntos relacionados à disciplina de Arquitetura e Organização de Computadores, para cursos em Ciência da Computação ou Engenharia de Computação. Permitindo também o aumento do engajamento por meio da visualização imediata de resultados em hardware.

2 Instalação e Configuração

2.1 Pré-requisitos

A seguir são expostos os pré-requisitos para utilização do software BIP-ACE.

- **Sistema Operacional**: Windows 10 ou 11 (64-bit);
- **Hardware** (opcional): caso for realizar o upload do código no BIP-FPGA, verificar conexão via USB e se certificar da alimentação da placa.

2.2 Instalação

Para a instalação do software, acesse o repositório do projeto BIP-ACE e baixe a última release (arquivo **BIP-ACE.zip**). Salve em uma pasta de sua preferência. Extraia os arquivos da pasta compactada e, caso queira, crie um atalho na área de trabalho para o arquivo executável: **BIP-ACE.exe**.

O BIP-ACE não requer uma instalação tradicional - basta executar o arquivo **.exe**. Se o Windows bloquear a execução, clique em: **Mais informações** > **Executar mesmo assim**.

2.3 Verificação da Conexão com o Hardware

As etapas a seguir são opcionais, caso o usuário não tenha a intenção de utilizar o ambiente BIP-ACE integrado ao hardware BIP-FPGA.

A seguir é enumerado um passo-a-passo para verificação do BIP-FPGA.

- Conecte a placa BIP-FPGA ao computador via USB;
- Aguarde até que o Windows reconheça o dispositivo;
- Abra o **Gerenciador de Dispositivos** e verifique: **Portas (COM e LPT)**. Caso a porta de comunicação (COM) não tenha sido detectada, verifique os itens anteriores;

Na etapa de upload do código para o BIP-FPGA, haverá uma varredura das portas de comunicação (COMs) onde a porta correta deverá ser selecionada.

Caso nenhuma porta seja detectada, o ambiente acusará um erro - podendo ser o caso de retomar os passos anteriores.

2.4 Problemas Comuns e Soluções

Os principais problemas que podem ser enfrentados durante a execução do software são os seguintes.

1. **Arquivo .exe bloqueado**: desative temporariamente o antivírus ou use "Executar mesmo assim";

2. **Placa não detectada**: verifique o cabo USB ou reinicie o computador;

3. **Erro de permissão**: execute o BIP-ACE como administrador (botão direito > "Executar como administrador").

3 Ambiente de Desenvolvimento

Esta seção explica a interface do BIP-ACE (vide Figura 1) e como utilizá-la para escrever, montar e executar programas no BIP-FPGA.

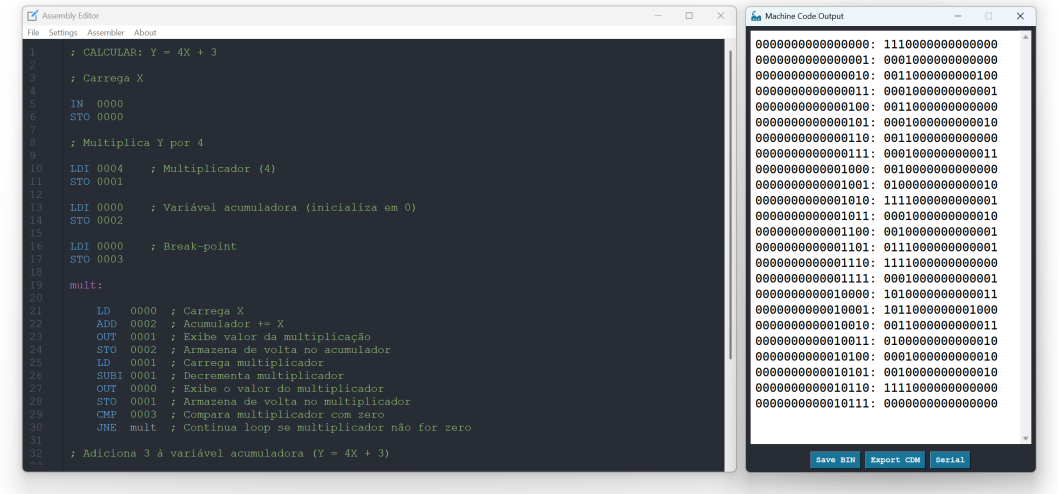

Figura 1: Interface do BIP-ACE em Dark Mode.

1. **Barra de Menu Superior**

- **File**: abrir ou salvar arquivo (.asm);
- **Settings**: permite a alternância entre temas: Dark e Light. Assim como o carregamento de diferentes conjuntos de instruções (no formato .json);
- **Assembler**: permite a montagem do código de máquina para posterior upload no BIP-FPGA ou exportação em arquivo .cdm (CEDAR Memory File) ou .bin;

2. **Editor de código**
    - Área principal para escrever código em Assembly;
    - Sintaxe destacada (instruções, labels, números e comentários);
    - Numeração de linhas.
3. **Saída de Código de Máquina**
    - Janela aberta ao realizar a montagem do código, através da barra de menu ou de seu atalho (CTRL+R);
    - Nessa janela o código gerado pode ser visualizado em binário, no formato: **<endereço>: <instrução>**;
    - Além disso, na barra inferior, estão as opções para exportação do código ou upload para a placa BIP-FPGA (via comunicação serial).

4 Montagem do Código

Com o entendimento do ambiente de desenvolvimento, pode-se enfatizar o fluxo básico de trabalho, que consiste em:

1. Abrir um arquivo .asm já salvo na máquina, ou escrever o código através do editor de código;
2. Caso seja um novo código, salvá-lo;
3. Executar o item **Assembler > Assemble** acessível através da barra de

menu superior;

4. Verificar o código de máquina gerado;

5. Realizar a exportação no formato desejado, ou o upload na placa BIP-FPGA através da comunicação serial.

O processo de upload do código de máquina gerado na memória da placa BIP-FPGA, será enfatizado mais adiante, na seção 6 (Integração com o Hardware).

5 Exemplos Práticos

Esta seção apresenta exemplos que ilustram a aplicação prática do BIP-ACE em diferentes cenários. Os códigos foram desenvolvidos para explorar as funcionalidades da ISA BIP-I e demonstrar a integração direta entre software e hardware, seguindo um fluxo crescente de complexidade.

5.1 Cálculo de Expressão Algébrica

Para introduzir as operações básicas definidas pela ISA do BIP-FPGA, considere o cálculo da seguinte expressão: $Y = 4X + 3$. O código utiliza instruções como LDI (carregar valor imediato no acumulador), ADDI (soma com constante), IN e OUT (para operações de entrada e saída - I/O) etc.

```
; CALCULAR: Y = 4X + 3
; Carrega X

IN  0000
STO 0000

; Multiplica Y por 4

LDI 0004    ; Multiplicador (4)
STO 0001
LDI 0000    ; Variável acumuladora (inicializa em 0)
STO 0002
LDI 0000    ; Break-point
STO 0003
```

```
mult:

    LD   0000  ; Carrega X
    ADD  0002  ; Acumulador += X
    OUT  0001  ; Exibe valor da multiplicação
    STO  0002  ; Armazena de volta no acumulador
    LD   0001  ; Carrega multiplicador
    SUBI 0001  ; Decrementa multiplicador
    OUT  0000  ; Exibe o valor do multiplicador
    STO  0001  ; Armazena de volta no multiplicador
    CMP  0003  ; Compara multiplicador com zero
    JNE  mult  ; Continua loop se multiplicador não for zero

; Adiciona 3 à variável acumuladora (Y = 4X + 3)

LDI 0003
ADD 0002
STO 0002

eop:

    LD   0002
    OUT  0000  ; Exibe o resultado
    HLT  0000  ; Para a execução
```

Código 1: Cálculo de expressão algébrica em Assembly

Para executar o código, basta montar o código - através do atalho **<CTRL+R>** ou no menu: **Assembler > Assemble** - e exportar o código ou realizar o upload no BIP-FPGA. O arquivo de extensão .cdm (CEDAR Memory File) pode ser usado para carregar na memória de código no software Cedar Logic. Outra opção é utilizar o emulador BIPy, também desenvolvido para a disciplina de Arquitetura e Organização de Computadores.

Caso o aluno opte por utilizar o BIPy, é necessário que se substitua a instrução IN por um LDI (load imediato) e retirar a instrução OUT. Pois, a ISA (Instruction Set Architecture) original do BIP não oferece essas instruções.

5.2 Sequência de Fibonacci

O código inicia definindo os valores iniciais da sequência ($F(0) = 0$ e $F(1) = 1$) nos endereços 20 e 21. Um loop infinito calcula cada novo termo somando os dois anteriores, armazenando o resultado em 22 e atualizando os valores anteriores. A instrução OUT 00 exibe cada termo no display (porte de saída), criando uma sequência visível (1, 1, 2, 3, 5, 8...).

```
; Gera Fibonacci até 255

LDI  0      ; ACC = 0 (F(0))
STO  20
LDI  1      ; ACC = 1 (F(1))
STO  21
OUT  00     ; Exibe F(1)

loop:

    LD   20     ; Carrega F(n-2)
    ADD  21     ; Soma com F(n-1) → F(n)
    STO  22
    OUT  00     ; Exibe F(n)
    LD   21     ; Atualiza F(n-2) e F(n-1)
    STO  20
    LD   22
    STO  21
    JUMP loop   ; Itera infinitamente
```

Código 2: Cálculo da sequência de Fibonacci em Assembly

Para fins pedagógicos, sugere-se desafiar os alunos a modificar o código para limitar a sequência ao ultrapassar 100, introduzindo comparações (CMP) e saltos condicionais (JL ou JG).

6 Integração com o Hardware

A comunicação serial entre o BIP-ACE e o BIP-FPGA é estabelecida via protocolo UART (Universal Asynchronous Receiver/Transmitter). Esse protocolo é uma versão generalizada dos protocolos EIA, RS-232, RS-422 e RS-485, que visa fornecer um periférico customizável que possa ser utilizado em qualquer um desses protocolos.

Para conseguir realizar a transmissão utilizando UART, é necessário que o sinal do terminal de transmissão (COM do computador) esteja conectado no terminal de recepção do BIP-FPGA (via USB). Cada transmissão segue um formato fixo, conforme definido na Figura 2.

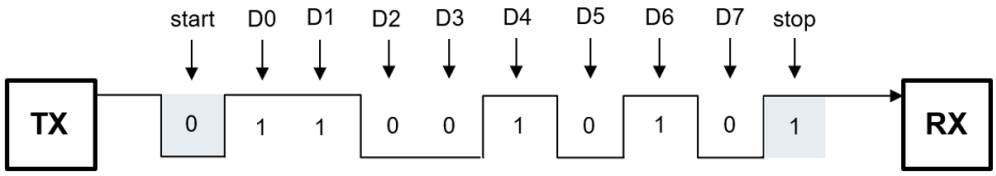

Figura 2: Esquema do funcionamento da transmissão via UART.

Inicialmente é fornecido um bit de start (início), sempre em nível lógico baixo - permitindo ao receptor sincronizar o clock interno. Então, prossegue-se com os bits de dados (D0 a D7), representando um byte transmitido serialmente. A ordem pode variar (geralmente LSB-first, ou seja, D0 é o bit menos significativo). Por último, chega um bit de stop (parada), sempre em nível lógico alto - indicando o fim do frame.

No caso do BIP-FPGA a taxa de transmissão (baud rate) é configurada para 9600 bps.

Como cada instrução no BIP-FPGA requer 28 bits (12 bits para endereçamento e 16 bits para o código de operação), o sistema emprega um mecanismo de empacotamento em 4 bytes (32 bits). O processo ocorre em duas etapas:

1. Recepção Byte a Byte: Um circuito receptor UART (Figura 3) captura cada byte transmitido e o armazena em um registrador de desloca-

mento de 32 bits.

2. Montagem do Pacote: Após receber 4 bytes, o registrador gera um sinal data_rdy (dado pronto), indicando que a instrução completa está disponível para escrita na memória.

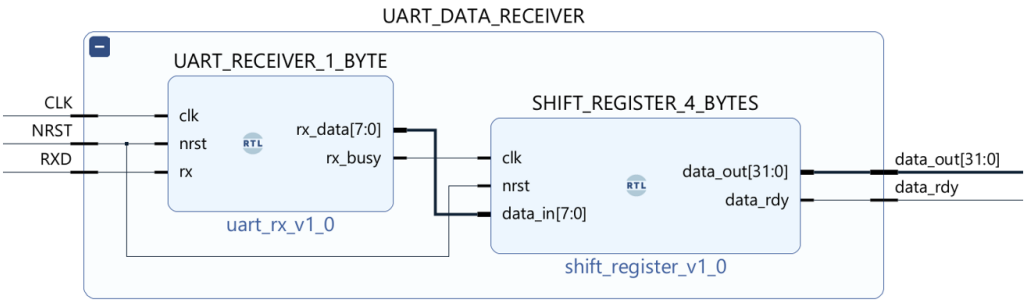

Figura 3: Diagrama de blocos para recepção de instruções via UART.

A atualização dinâmica do código na FPGA é controlada por uma Máquina de Estados Finitos (FSM), conforme detalhado na Figura 4. Os estados são:

1. IDLE (Ocioso):
    - Estado inicial, aguardando a recepção de dados.
2. WR (Escrita):
    - Ativado quando data_rdy é sinalizado. A instrução é gravada na memória de programa (code memory).
3. EX (Execução):
    - Libera o circuito para executar o código carregado.

Ao finalizar o upload (sinalizado pelo pacote 0xF000 - EOT), a FSM retorna ao estado EX. O contador de programa do BIP-FPGA deve ser reiniciado manualmente, prevenindo a execução de código incompleto ou corrompido. Esse mecanismo permite a atualização em tempo real (live memory update), onde o código pode ser modificado sem reinicializar o hardware.

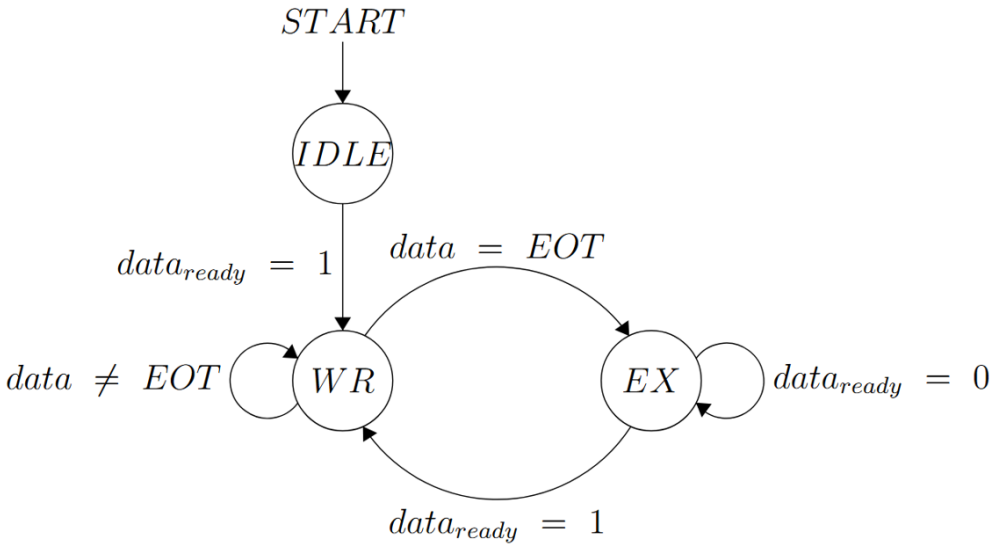

Figura 4: Máquina de estados finitos para gerenciamento da memória de código.

A Figura 5 exibe o layout dos componentes de I/O configurados na FPGA. Existem dois conjuntos de displays de 7 segmentos utilizados para os portes de saída. Dois conjuntos de chaves seletoras para formar os portes de entrada. Conjunto de botões para controle da execução.

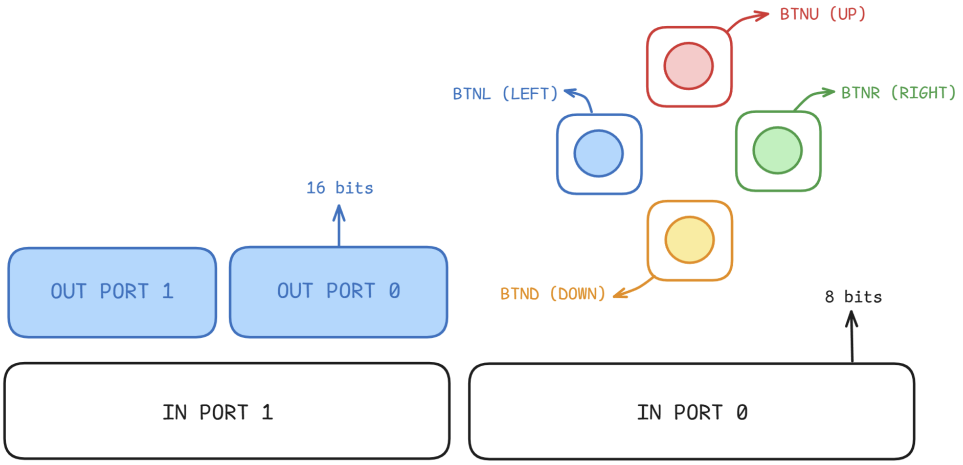

Figura 5: Esquema do layout do BIP-FPGA.

O botão superior (BTNU) permite a multiplexação dos portes de saída (out ports), de forma a alternar para a visualização do valor atual do contador de programa (PC) e registrador de instrução (IR).

O botão inferior (BTND) realiza o reset da execução do BIP-FPGA. Ou seja, em outras palavras, reinicia o contador de programa para zero - apontando para a primeira instrução na memória.

Os botões BTNL (esquerdo) e BTNR (direito) do BIP-FPGA são recursos essenciais para depuração e análise detalhada de programas em Assembly. Quando utilizados em conjunto, permitem pausar a execução do código e avançar manualmente por cada instrução, oferecendo um controle preciso sobre o fluxo do programa.

O botão BTNL atua como um interruptor de pausa. Ao ser pressionado, interrompe imediatamente a execução do código em qualquer estágio, congelando o estado atual do sistema. Enquanto o sistema está pausado, um LED sinaliza visualmente o modo de pausa, facilitando a identificação do estado atual pelo usuário.

No modo pausado, o botão BTNR assume a função de execução passo a passo (step mode). Cada pressionamento deste botão avança uma única instrução do programa. O display do BIP-FPGA, quando utilizando também o

botão BTNU, pode exibir informações em tempo real, como a instrução a ser executada, tornando o processo transparente e didático.

Para retomar a execução contínua, basta pressionar BTNL novamente. O sistema reinicia a operação automática a partir do ponto onde foi pausado, mantendo a integridade do contador de programa e dos dados armazenados.

Um exemplo prático dessa integração pode ser a depuração de um controle de fluxo de execução dinâmico - como o caso de saltos condicionais. Permite verificar de modo facilitado o código a ser executado pela tomada de decisão.

A Figura 6 exibe a placa FPGA com o BIP-FPGA carregado, executando o código para cálculo da sequência de Fibonacci. O código foi configurado para cessar a execução ao atingir o valor decimal 987 e imprimir o valor 7 para indicar a finalização da execução com sucesso.

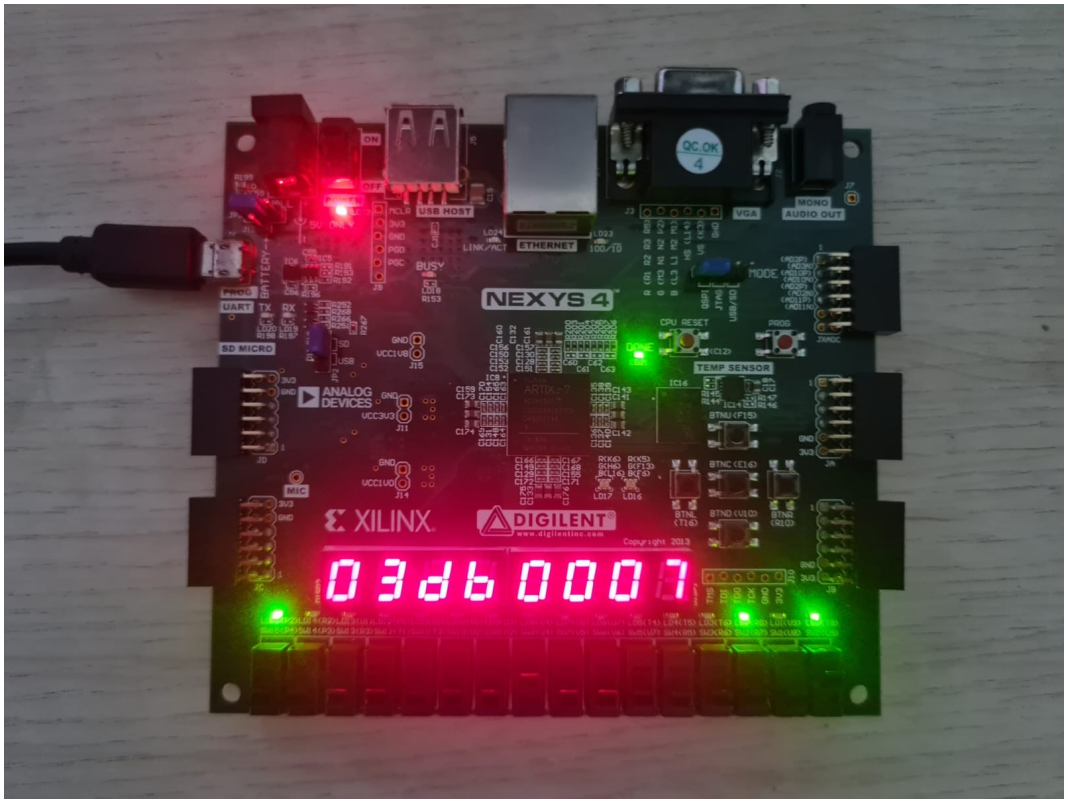

Figura 6: Foto do BIP-FPGA executando código.

7 Atividades Sugeridas

Esta seção foi redigida visando consolidar o aprendizado prático em Assembly e Arquitetura e Organização de Computadores, incentivando a experimentação direta com o ambiente BIP-ACE e o hardware BIP-FPGA. As propostas seguem uma progressão, partindo de ajustes simples em exemplos existentes, até projetos mais criativos.

7.1 Modificação de Exemplos

A estratégia central aqui consiste em adaptar os exemplos fornecido nesse manual, alterando parâmetros, lógicas ou objetivos. Por exemplo, o código de cálculo da sequência de Fibonacci (fib.asm), originalmente infinito, pode ser modificado para interromper a sequência ao atingir um valor pré-definido (ex: 144), introduzindo instruções de comparação e desvios condicionais.

7.2 Desenvolvimento de Projetos

Para ir além da modificação pontual, propõe-se a criação de algum projeto que integre múltiplos conceitos.

Um exemplo pode eser o cálculo de fatorial em assembly:

- Receber um número inteiro *n* (via porte de entrada);
- Calcular *n*! (fatorial de *n*);
- Exibir o resultado no display da placa BIP-FPGA;

O projeto desafia o aluno a gerenciar entradas dinâmicas providas por hardware e a superar limitações da ISA BIP-I, como a ausência de instruções complexas (ex: multiplicação), exigindo a implementação manual de operações via loops. Além disso, requer domínio de saltos condicionais para controle de fluxo e alocação eficiente de memória para variáveis e sub-rotinas.

Essa abordagem prática desenvolve habilidades críticas em sistemas embarcados: adaptação a restrições de hardware, otimização de código Assembly e integração entre lógica e componentes físicos.