Sistema de Controle de Reservatórios – Fase 4: Deploy em Hardware com Arduino UNO

Especificação de Projeto – Atividade de Programação 4

Sistema de Controle de Reservatórios – Fase 4: Deploy em Hardware com Arduino UNO

1. Introdução e Contexto

Esta especificação detalha os requisitos para a quarta e última fase do projeto de automação predial: a implementação do sistema de controle em um hardware real. Após o desenvolvimento e simulação das lógicas de controle em C++/Qt, o sistema completo será portado para a plataforma de prototipagem Arduino UNO.

O objetivo principal é substituir a simulação do processo físico e a Camada de Abstração de Hardware (HAL) simulada por uma nova HAL que interaja diretamente com os pinos de entrada e saída (I/O) do microcontrolador. Os sensores de nível serão emulados por chaves digitais, os atuadores (válvulas, bomba, resistência) serão representados por LEDs e o sensor de temperatura será simulado por um potenciômetro conectado a uma entrada analógica.

Esta fase final visa solidificar a compreensão sobre a portabilidade do software e a importância de uma arquitetura em camadas, que permite a transição de um ambiente de simulação para um alvo físico com alterações mínimas no código da lógica de aplicação (as máquinas de estado).

2. Objetivos de Aprendizagem

Ao final desta atividade, o estudante deverá ser capaz de:

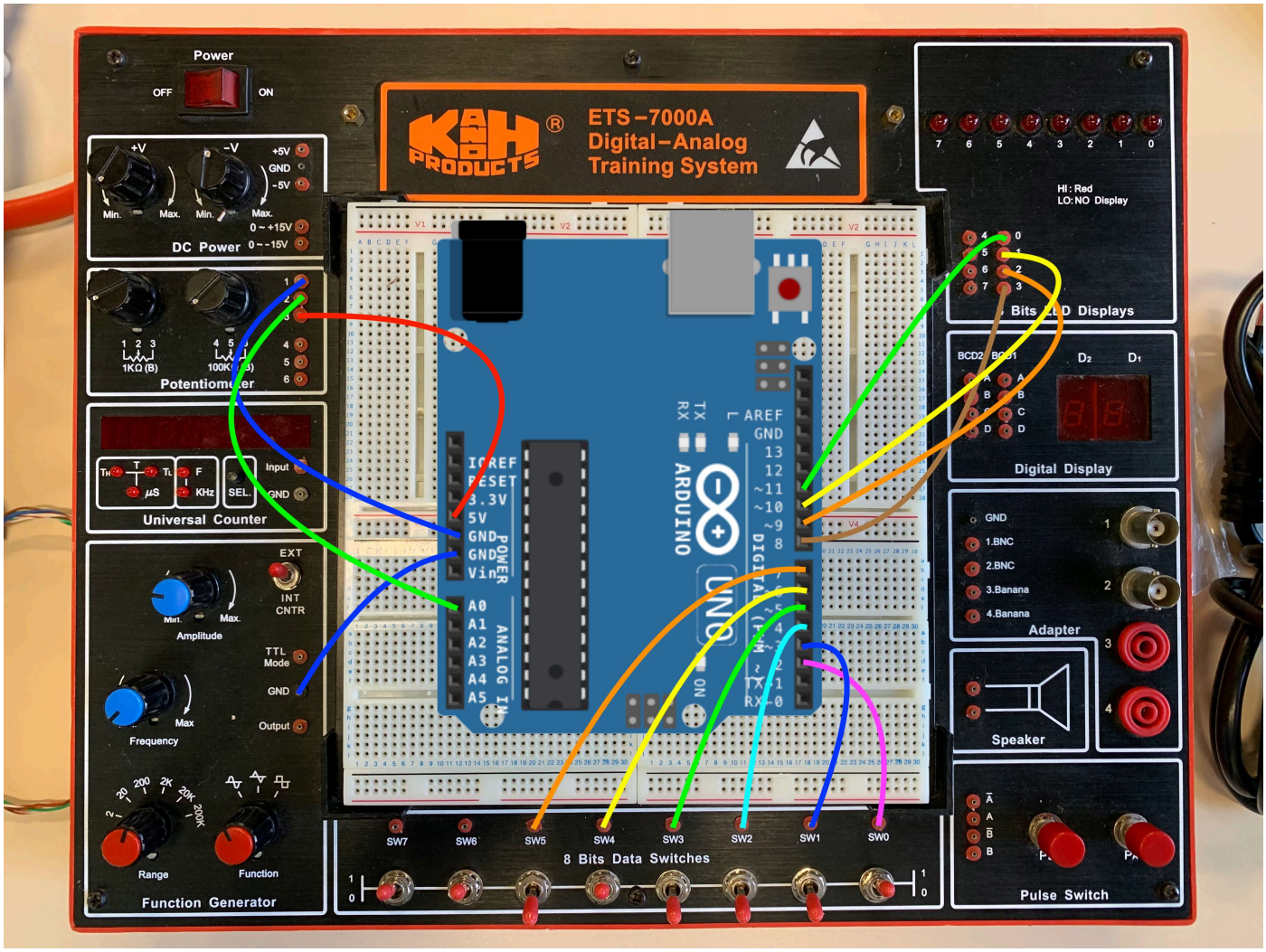
- Adaptar uma Camada de Abstração de Hardware (HAL) para um hardware específico (Arduino UNO).
- Mapear componentes lógicos (sensores e atuadores) para pinos físicos de um microcontrolador.
- Configurar, ler e escrever em portas de I/O digitais e ler portas analógicas.
- Implementar lógicas de temporização não-bloqueantes usando a função `millis()`, crucial para gerenciar atrasos de atuadores sem interromper o sistema.
- Converter valores de um sensor analógico (0-1023) para uma unidade de engenharia correspondente (graus Celsius).
- Validar a lógica de controle desenvolvida em simulação em um ambiente de hardware real.

3. Mapeamento de Hardware

O sistema será implementado no Arduino UNO utilizando uma mesa digital de prototipagem. A tabela abaixo detalha a conexão entre os componentes do sistema e os periféricos físicos.

Componente Lógico	Tipo	Pino Arduino	Dispositivo Físico
Sensor `s11` (Nível Baixo t1)	Entrada Digital	2	Chave SW0
Sensor `s12` (Nível Alto t1)	Entrada Digital	3	Chave SW1
Sensor `s21` (Nível Baixo t2)	Entrada Digital	4	Chave SW2
Sensor `s22` (Nível Alto t2)	Entrada Digital	5	Chave SW3
Sensor `s31` (Nível Baixo t3)	Entrada Digital	6	Chave SW4
Sensor `s32` (Nível Alto t3)	Entrada Digital	7	Chave SW5
Válvula `v1`	Saída Digital	8	LED3
Válvula `v2`	Saída Digital	9	LED2
Bomba `b1`	Saída Digital	10	LED1
Resistência `r1`	Saída Digital	11	LED0
Sensor `st1` (Temperatura t3)	Entrada Analógica	A0	Potenciômetro

Nota: Se forem necessários LEDs adicionais (por exemplo, para indicar estados de erro ou transição), o estudante é responsável por selecionar os pinos, atualizar a HAL e realizar a fiação correspondente.



4. Requisitos de Implementação na HAL

Toda a interação com o hardware deve ser encapsulada na Camada de Abstração de Hardware (HAL). As máquinas de estado desenvolvidas nas fases anteriores **não devem ser alteradas**; elas continuarão a chamar funções da HAL, como `s11()` ou `v1(true)`, mas a implementação interna dessas funções será diferente.

4.1. Funções Essenciais de I/O Digital e Inicialização

A HAL deve prover uma função `init_hal()`, que será responsável por toda a configuração inicial do hardware. Essa função deve ser chamada dentro da rotina `setup()` principal do Arduino. Dentro de `init_hal()`, as seguintes funções do Arduino serão utilizadas:

- `pinMode(pino, MODO)`: Configura um pino específico para operar como entrada (`INPUT`) ou saída (`OUTPUT`). Toda a configuração de pinos deve ser feita em `init_hal()`.
  Exemplo: `pinMode(8, OUTPUT); // Configura o pino 8 (LED3) como saída.`
- `digitalRead(pino)`: Lê o valor de um pino digital, que pode ser `HIGH` (alto) ou `LOW` (baixo). Será usada para verificar o estado das chaves que simulam os sensores.
  Exemplo: `bool status_s11 = digitalRead(2); // Lê o estado da chave SW0.`
- `digitalWrite(pino, VALOR)`: Escreve um valor digital (`HIGH` ou `LOW`) em um pino. Será usada para ligar ou desligar os LEDs que representam os atuadores.

Exemplo: `digitalWrite(8, HIGH); // Liga o LED3 (ativa a válvula v1).`

4.2. Aquisição do Sinal de Temperatura (Analógico)

O potenciômetro simula o sensor de temperatura `st1`. Sua leitura será feita pela porta analógica A0 usando a função:

- `analogRead(pino)`: Lê a tensão em um pino de entrada analógica e a converte em um valor numérico entre 0 (para 0 volts) e 1023 (para 5 volts).
  Exemplo: `int valorPot = analogRead(A0);`

Conforme o requisito, este valor (0 a 1023) deve ser mapeado para a faixa de temperatura de 40.0 a 60.0 °C.

```
float lerTemperatura() {
    int valorLido = analogRead(A0);
    // Mapeia a faixa de 0-1023 para 40.0-60.0
    float temperatura = 40.0 + (valorLido / 1023.0) * (60.0 - 40.0);
    return temperatura;
}
```

4.3. Obtenção do Tempo do Sistema (now() e millis())

A camada de abstração de hardware deve prover a função `now()`, que retorna o tempo de sistema em milissegundos. Esta função é essencial para que as máquinas de estado controlem temporizações de forma não-bloqueante. A implementação de `now()` na HAL para Arduino deve, por sua vez, utilizar a função nativa `millis()`.

- `millis()`: Retorna o número de milissegundos desde que a placa Arduino começou a executar o programa atual.

5. Procedimento de Validação

1. **Setup Inicial:** Clone o repositório do template base para iniciar o projeto. Este template já possui a estrutura de diretórios e arquivos necessária.
   `git clone https://github.com/afmiguel/template-arduino-microchip-studio.git`
2. **Montagem Física:** Realize todas as conexões elétricas entre o Arduino UNO e a mesa digital, conforme a tabela de mapeamento.
3. **Implementação da HAL:** O arquivo de interface `hal.h` já existe no diretório comum (`common`). O estudante deve criar o arquivo `hal.cpp` específico para a plataforma Arduino, implementando todas as funções declaradas no `.h`.
4. **Integração:** Adapte o código principal para instanciar e utilizar a nova HAL para Arduino. As máquinas de estado devem permanecer inalteradas.
5. **Compilação e Deploy:** Utilize o **Microchip Studio** como toolchain para compilar o projeto e transferir o firmware para a placa Arduino.
6. **Teste Funcional:**
   - Manipule as chaves (SW0-SW5) para simular as mudanças nos níveis dos reservatórios.

- Verifique se os LEDs correspondentes aos atuadores (v1, v2, b1, r1) acendem e apagam de acordo com a lógica implementada nas máquinas de estado.
- Gire o potenciômetro e, através do Serial Monitor, verifique se a leitura de temperatura está correta.
- Teste especificamente as lógicas de proteção, como o desligamento da bomba `b1` quando `s11` desativa, e o desligamento da resistência `r1` quando `s31` desativa.
- **Atenção ao Estado de Erro:** Lembre-se que as máquinas de estado possuem uma lógica de falha segura. Se uma condição inválida for detectada (ex: sensor `s12` ativo com `s11` inativo), o sistema entrará em estado de erro e todos os atuadores serão desligados. Para retornar à operação normal, será necessário **reiniciar a placa Arduino**.