MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
CAMPUS SÃO JOSÉ
ENGENHARIA DE TELECOMUNICAÇÕES

RELATÓRIO
TRABALHO 2 STE

Aluno: Mário André Lehmkuhl de Abreu
Matéria: Sistemas Embarcados (STE)
Professor: Roberto de Matos

Outubro
2018

1 Introdução

O Trabalho 2 consiste em analisar 4 versões da Classe GPIO fornecida pelo professor, denominadas **GPIO_v1**, **GPIO_v1.2**, **GPIO_v1.3**, **GPIO_v2**. E relatar qual a diferença entre elas em relação ao seu método **construtor** e o método **set(val)**. Deve-se verificar quantos bytes cada versão usa para executar esses métodos, e mostrar o tamanho total dos recursos ocupados pelo atributo do objeto **"GPIO"**. Tambem deve-se completar a classe **GPIO** e **GPIO_port** para todos os pinos da plataforma Arduino na versão **GPIO_v2**. Nesse complemento os arrays devem ser forçados para a memória flash.

2 Análise

Aqui é descrito como ocorre a lógica nos métodos **contrutores** e **set(val)** em cada versão da classe **GPIO**. Ao final da descrição é mostrado o numeros de bytes utilizados para executar o metodo. O numero de bytes usado foi verificado no arquivo **.lss** de cada versão gerado após a compilação do código. Em todas as versoẽs o método construtor possue dois parâmetros a serem fornecidos. O primeiro é o numero do pino que se deseja se utilizar no AVR e o segundo é como o pino vai se comportar, isto é, como entrada ou saída. O método **set(val)** possue um parâmetro a ser fornecido que é como o pino na placa deve ser setado, isto é , em 0 ou 1.

2.1 GPIO_v1

- **GPIO(uint8_t id, PortDirection_t dir):** Fornecido o pino a ser usado e como ele vai se comportar, o metodo construtor pega o numero do pino e o copia para duas vaiaveis **_id** e **_bit**. A variavel **_id** é usada em uma estrurura de decisão de **Caso** pra determinar qual pino foi selecionado e determinar qual registrador **DDRn** deve-se configurar para configurar o comportamento do pino. Para saber se o pino vai ser configurado como saida (1) ou entrada (0) uma estrutura de decisão **if** é usada utilizando como comparação o segundo parametro do construtor. A variavel **_bit** é usado pra saber qual bit do registrador corresponde o pino escolhido.

  Tamanho utlizado: 810 bytes

- **void set(bool val = 1):** Usando as variavies **_id** e **_bit** geradas no método construtor, utilisa-se a variavel **_id** em uma estrutura de decisão de **Caso** para determinar qual registrador **PORTn** deve-se selecionar para setar o pino como 0 ou 1. Para selecionar se é 0 ou 1, após selecionar o caso, uma estrutura de decisão **if** é executada usando como variavel de comparação o parâmetro do **val**, se o parâmetro for 0 o pino é setado como 0, se for 1 é setado como 1. A variavel **_bit** é usada para determinar qual bit do registrador **DDRn** deve-se ser setado, correspondendo ao pino selecionado no metodo construtor.

  Tamanho utlizado: 498 bytes

- **void GPIO::clear():**

Tamanho utlizado: 526 bytes

- **void GPIO::toggle():**

  Tamanho utlizado: 240 bytes

- **Total:** 2074 bytes

2.2 GPIO_v1.2

- **GPIO(uint8_t id, PortDirection_t dir):** Passado o pino a ser usado e como ele vai se comportar, o metodo construtor pega o numero do pino e o copia para a vaiavel **_id**. A variavel **_id** é usada em uma estrurura de decisão de **Caso** pra determinar qual pino foi selecionado e determinar qual registrador deve-se utilizar. Nesse construtor três novas variaveis são usadas, que são **_pin**, **_ddr** e **_port**. Ao entrar no caso correspondente ao pino selecionado, essas três variavesis são usadas para receber o endereço de referencia dos registradores **PINn**, **DDRn** e **PORTn**. Alem delas a variavel **_bit** é criada e recebe o valor em 8 bits do valor 1 deslocado para a esquerda com base no valor da variavel **_id**. Ao sair do caso é utilizada uma estrutura de decisão **if** usando como comparação o segundo parametro do construtor **(dir)** para determinar o comportamento do pino, isto é , como entrada ou saída. Para realizar essa configuraçao é usado a variavel **_ddr**, que agora possue o endereço de referencia do registrador **DDRn**. Para selecionar o bit do registrador correspondente ao pino desejado é usado a variavel **_bit**.

  Tamanho utlizado: 624 bytes

- **void set(bool val = 1):** Para setar o pino em 0 ou 1 uma estrutura de decisão **if** utilizando como comparação o parâmetro **val** é usada. Se o parâmetro for 0 o pino é setado em 0. Se for 1 é setado em 1. Para setar o pino é usado as variaveis **_port** e **_bit** criadas no construtor. A variavel **_port** possue o endereço de referência do registrador **PORTn**, e a variavel **_bit** o numero do bit correspondente do registrador.

  Tamanho utlizado: 118 bytes

- **void GPIO::clear():**

  Tamanho utlizado: 144 bytes

- **void GPIO::toggle():**

  Tamanho utlizado: 50 bytes

- **Total:** 936 bytes

2.3 GPIO_v1.3

- **GPIO(uint8_t id, PortDirection_t dir):** Passado o pino a ser usado e como ele vai se comportar, o metodo construtor pega o numero do pino e o copia para a variavel **_id**. A variavel **_id** é usada em uma estrurura de

decisão de **Caso** pra determinar qual pino foi selecionado e determinar qual registrador deve-se utilizar. Ao entrar no caso correspondente duas variaveis são usadas **_bit** que recebe o valor em 8 bits do numero 1 deslocado para a esquerda com base no valor da variavel **_id**, e determina qual é o bit do registrador que representa o pino selecionado. E a variavel **_Px** que é um ponteiro do tipo **GPIO_PORT** que recebe o endereço do registrador selecionado. Após configurar essas variaveis o caso termina e variavel **_Px** chama o metodo **dir(_bit, dir)** para selecionar o comportamento do pino. No metodo **dir(_bit, dir)** uma estrutura de decisão **if** é usada, e utilizando como comparação a variavel **dir**, se determina o comportamneto do pino. Para configurar o comportamento é usado a variavel **_ddr** que foi criada pela classe **GPIO_PORT** ao ser criado a variavel **_Px** no construtor. Ao criar a variavel **_Px** do tipo **GPIO_PORT** ela cria três variaveis **_pin**, **_ddr** e **_port**, que recebem os endereços dos registradores **PIN**, **DDR** e **PORT** respectivamente do pino correspondente selecionado. Para selecionar o bit correspondene do registrador se usa a variavel **_bit**.

Tamanho utlizado: 520 bytes

- **void set(bool val = 1):** Para setar o pino em 0 ou 1 é usado a variavel **_Px** criada no construtor. Usa-se ela pra chamar o metodo **_Px->set(_bit, val)**. Nesse método é usado uma estrutura de decisão **if** utilizando como comparação o parametro **val**. Se o parametro for 0 o pino é setado em 0. Se for 1 é setado em 1. Para setar o pino é usado as variaveis **_port** e **_bit** criadas no construtor **GPIO_PORT** e **GPIO** respectivamente. A variavel **_port** possue o endereço de referrencia do registrador **PORTn**, e a variavel **_bit** o numero do bit correspondente do registrador.

  Tamanho utlizado: 146 bytes

- **void GPIO::clear():**

  Tamanho utlizado: 182 bytes

- **void GPIO::toggle():**

  Tamanho utlizado: 90 bytes

- **Total:** 938 bytes

2.4 GPIO_v2

- **GPIO(uint8_t id, PortDirection_t dir):** Passado o pino a ser usado e como ele vai se comportar, o metodo construtor inicia e pega a variavel **_bit** e a faz receber o valor do array **id_to_bit** da classe **GPIO_PORT**. A posição do array que a variavel **_bit** recebe é feita com base no valor do parâmetro **id**. O valor retornado do array é o numero 1 em 8 bits deslocado para a esquerda em x posições ja determinado pelo pino selecionado. Depois pega-se a variavel **_port** e a faz receber o array **AllPorts** da classe **GPIO_PORT**. O valor da posição desse array e feita pelo uso do array **id_to_port** tambem da

classe **GPIO_PORT** que usa a varial **id** para determinar a posição que vai ser selecionada. Fazendo essa seleção da posição dos arrays o valor retornado é o endereço do registrador correspondente ao pino selecionado. Por ultimo a variavel **_port** chama o metodo **dir(_bit, dir)** da classe **GPIO_PORT** para configurar o comportamento do pino. No método **dir(_bit, dir)** é usado uma estrutura de decisão **if** para selecionar se o pino vai ser saída (1) ou entrada (0). Para configurar o comportamento é usado a varivel **_ddr** criada quando se definiu a variavel **_port** no construtor. Essa variavel possui o endereço do registrador **DDRn** do pino selecionado. Para selecionar o bit do registrador correspondente ao pino usa-se a variavel **_bit** passada como parametro no metodo **dir(_bit, dir)**.

Tamanho utlizado: 226 bytes

- **void set(bool val = 1):** Para setar o pino em 0 ou 1 é usado a variavel **_port** criada no construtor. Usa-se ela pra chamar o metodo **set(_bit, val)** da clsse **GPIO_PORT**. Nesse método é usado uma estrutura de decisão **if** utilizando como comparação o parametro **val**. Se o parametro for 0 o pino é setado em 0. Se for 1 é setado em 1. Para setar o pino é usado as variaveis **_port** e **_bit** criadas no construtor **GPIO_PORT** e **GPIO** respectivamente. A variavel **_port** possue o endereço do registrador **PORTn**, e a variavel **_bit** o numero do bit correspondente ao pino selecionado no registrador.

  Tamanho utlizado: 146 bytes

- **void GPIO::clear():**

  Tamanho utlizado: 182 bytes

- **void GPIO::toggle():**

  Tamanho utlizado: 90 bytes

- **Total:** 644 bytes

2.5 Análise entre as versões

Olhando o numero de byte utilizado nos métodos em cada versão chegou-se as seguintes conclusões:

- A versão subsequente a anterior é mais eficiente em relação ao número de bytes usados.

- A primeira versão **(GPIO_v1)** é a menos eficiente pois foi a que utilizou mais bytes nos dois metodos. E a ultima versão **(GPIO_v2)** a mais eficiente pois utilizou menos bytes em relação as outras.

- Percebeu-se que a redução no número de bytes em cada versão esta relacionado em grande parte com o uso da estrutura de decisão de **Caso** e **if**. Pois na versão **v1** tanto o construtor e o set usam **Caso** e **if**. Nessa versão há um **if** para cada **Caso**. Na versão **v1.2** o **if** foi tirado de dentro do **Caso**, sendo executado só

no final, depois de ja selecionado o registrador desejado. O que ja refletiu na diminuição no numero de bytes. No metodo set a diminuição tambem ocorreu pois nao se usou mais o **Caso**, ficou só um **if**. Na versão **v1.3** a estrutura de caso ainda continua, mas pra determinar o registrador utlizado é usado a classe **GPIO_PORT**, que repassa só o endereço do registrador que vai se utilizar, nao precisando deixar declarado como na versão **v1.2**. No método set é usado uma estrutura de decisão **if** como na versão **v1.2**, mas no caso esse **if** é executado na classe **GPIO_PORT**. Por ultimo na versão **v2** ocorreu mais redução de bytes pois a estrutura **Caso** naõ foi utilizada, sendo só usado o uso de arrays da classe **GPIO_PORT** para selecionar os registradores desejados. No método set se usou a estrutura **if** da classe **GPIO_PORT**.

- Nas quatro versões verificadas as versões **v1.2**, **v1.3** e **v2** usam a variavel **_bit** para calcular o deslocamento de bit a esquerda para determinar o bit usado no registrador. Já na versão **v1** a variavel não é usada, o calculo é feito diretamente no registrador quando necessario. Verificando essa diferença de implementação notou-se que o mais interesante de se usar é o com o uso da versão que utiliza a variavel **_bit**, pois desse jeito o calculo é feito só uma vez, sendo só necessario reutilzar a variavel quando preciso. Já no caso da versão **v1** sempre é calculado o deslocamento de bit o que resulta em mais tempo gasto e uma maior numero de bytes usado.

- O método **set** da versão **v1.2** teve a melhor eficiência no numero de uso de bytes. Verificou-se que isso ocorreu pelo fato de só se usar a estrutura **if** e essa verão nao possuir a classe **GPIO_PORT** para acessar o método **set**, que faria o código ter que saltar para outra classe pra concluir a execução.

- Os códigos apresentados mostram um esbouço inicial de como utilizar lógicas diferentes para se utilizar as portas do AVR. Como visto, dependendo a lógica utilizada o uso de bytes necessaario para executar o metodo desejado pode ter uma grande redução. Desse modo nota-se que se for completar o codigo para se utilizar todos os pinos do AVR a versão mais indicada a se usar é a versão **v2**, pois utiliza os arrays para determinar os registradores a ser usado, o que deixa o código mais enxuto e com o uso de menos bytes em relação as outras versões. Ja se for usar as outras versões o codigo vai ficar mais grande, pelo fato de usar a estrutura **Caso**, pois vai ser necessario usar muito **Caso** para os 70 pinos do AVR. Dessas outras versões, a versão **v1** tem mai destaque, pois em cada **Caso** ainda tem um **if** dentro, o que para 70 pinos vai ficar grande o código.

- O método **clear()** em todas as versões é maior e mais lento que o método **set()**. Isso ocorre porque o método **clear()** chama o método **set()** para colocar o valor do pino selecionado em 0. Devido a essa chamada ele precisa carregar toda a execução do método **set()**. E como o **set()** é chamado dentro do **clear()** o tamanho do **clear()** é maior pois deve-se somar o valor do **clear()** mais o **set()**. Uma alternativa para resolver isso é não chamar o método **set()**, e sim colocar o endereço do registrador **DDRn** corresposndente ao pino selecionado

direto no método. Assim não precisa chamar outro método, ganhando tempo de execução.

3 Implementação da versão v2 para todos os pinos do AVR

Para implementar todos os 70 pinos do AVR com base no código inicial fornecido pelo professor, foi necessario seguir o esquema ja definido pelo código. Desse modo foi feita as seguintes modificações:

- Na classe GPIO_PORT o array **AllPorts** que estava com 4 posições foi modificado para 11 para conter todos os endereços dos registradores dos pinos do AVR.

- Na classe **GPIO_PORT** os arrays **id_to_port** e **id_to_bit** foram modificados de 14 posições para 70 posições para conter todos os pinos do AVR. No array **id_to_port** para preencher as novas posições criadas se seguiu a sequência dos pinos da coluna **Port** da tabela de **Mapeamento de Pinos Arduino Mega** fornecida pelo professor. No array **id_to_port** se seguiu o mesmo esquema só que se utilizou a coluna **Pin** da tabela.

- No metodo **enum Ports_index** da classe **GPIO_PORT** se adicionou o restantes das letras dos pinos que faltavam. Esses pinos são usados pelo array **id_to_port** para definir qual a posição no array **AllPorts** deve ser chamado para determinar qual registrador usar.

- Como especificado, deve-se fazer os array que são **const** serem armazenados na memoria de programa (flash). Os arrays que possuem essa caracteristica são o **id_to_port** e o **id_to_bit**. Para fazer isso nos array se adicionou a palavra chave **PROGMEM** logo após a declaração do tamanho da posição.

  const uint8_t id_to_port[70] PROGMEM = {

  Para poder acessar o conteudo dos array nas memoria de programa deve-se usar a palavra chave **pgm_read_byte(&)**

  pgm_read_byte(&(GPIO_PORT::id_to_bit[id])).

  O uso desse recurso foi usado pelo fato de que em alguns casos o tamanho da memoria RAM da placa é limitado e pode nao ser o suficiente para realizar o programa desejado. No entanto a memoria flash das placas pode posssuir mais espaço disponivel. Desse modo o uso desse espaço é desejado. O uso de dados constantes é interessante colocar na memoria flash, pelo fato deles não serem alterados e com isso é mais desejavel quarda-los ali, não ocupando um espaço que pode ser necessario na memoria RAM. Como os arrays **id_to_port** e o **id_to_bit** possuem um numero de posições consideravel eles se encaixam muito bem nesse cenário.