AT1 - Sistemas Embarcados

Leticia Coelho

Setembro 2018

1 Introdução

Este relatório apresenta os resultados da comparação de 4 versões da classe GPIO, esta atividade foi proposta na disciplina de Sistemas Embarcados (STE), ministrada pelo professor Roberto de Matos no curso de Engenharia de Telecomunicações do Instituto Federal de Ciência, Educação e Tecnologia de Santa Catarina. Os experimentos foram realizados no software Eclipse® e fazendo a verificação das características utilizando o programa *objdump*, a fim de proporcionar maior entendimento sobre a diferença na programação de microcontroladores, sendo este estudo dirigido ao ATMega2560 utilizando os periféricos da placa Arduíno Mega.

2 Comparativo - *GPIO*

2.1 Versão 1

Esta versão utiliza laços de case para a implementação da classe GPIO, tendo uma implementação extensa e com maior tempo de execução. Assim, utiliza a mascara do bit e não o próprio. Isso torna as funções de *construtor* e *set* extensas pois é necessário analisar cada caso isoladamente, sem utilização de macros a implementação se torna manual e vagarosa, além de possui um baixo desempenho temporal devido ao aumento no número de linhas.

3 Versão 1.2

Utiliza os ponteiros *_pin*, *_ddr* e *_port* minimiza a implementação das funções *set(bool val)* e *get( )* não sendo necessário utilizar laços de verificação pois a definição das características da porta é realizada na função construtora.

4 Versão 1.3

Realiza a classe *GPIO_port* que possui as características da porta, ou seja o *pin*, *ddr* e *port*. Além disso, essa classe possui também os métodos de manipulação

das características do registrador.

Declara os ponteiros *PB,PE,PG* e *PH* que são mapeados em memória possuindo melhoria na utilização de bytes, diminuindo em até 170 bytes na utilização da classe. Nesse caso não possui um ponteiro para cada característica da porta como na versão 2, mas sim um ponteiro que relaciona a porta e suas características como um todo. A desvantagem desta implementação é a necessidade de mais uma chamada de função nas funções *clear()*, *set(bool val)* e *get( )*, gerando mais inserções e remoções na pilha de processos.

5 Versão 2

Ainda trabalhando na classe *GPIO_port* criou as variáveis *id_to_port* e *id_to_bit* que realizam uma indexação do id para a porta e o bit. O acesso a registrador é feito através da posição de cada um dos vetores que mapeiam as características do vetor. Utilizando a macro *PROGMEM* envia os dados das constantes para a memória Flash, otimizando a utilização e evitando problemas relacionados a manipulação de dados na memória RAM.

6 Análise

As diferentes implementações demonstraram que ao realizar a operação *_bit* como (1 « BIT_PORTA) repetidas vezes gera maior consumo de recursos do AVR, guardando a mascara inserindo o valor na variável **__bit** torna possível verificar a diminuição do tamanho do código e consequentemente o tempo de execução.

A inserção das constantes na memória Flash é utilizado para evitar as limitações de espaço em memória RAM, assim utiliza-se o espaço disponível na memória Flash para armazenar valores necessários. Utilizando esta técnica o acesso as constantes deve ser realizado através de funções específicas, pois as constantes estão armazenadas em outra memória, neste caso utilizamos a biblioteca *PGMSpace* que oferece o acesso ao atributo especial *progmem*, acessa com a utilização do AVR-Libc pelo fornecimento de macros simples como *PROGMEM* e utilizando as demais macros fornecidas é possível verificar e modificar as constantes na memória.

Tabela de análise das funções

Função	**V1**	**V 1.2**	**V 1.3**	**V 2**
Construtor	505	364	247	94
Set	319	65	30	30
Clear	21	21	21	21