AT3 - Sistemas Embarcados

Luísa Machado, Leticia Aparecida Coelho
Engenharia de Telecomunicações, Instituto Federal de Santa Catarina
luisamachado@gmail.com, leticia.ac23@gmail.com

Setembro 2018

1 Introdução

Este relatório apresenta os resultados relativos a atividade 3 proposta na disciplina de Sistemas Embarcados (STE), ministrada pelo professor Roberto de Matos no curso de Engenharia de Telecomunicações. Os experimentos foram realizados no software Eclipse© e utilizando periféricos da placa Arduíno Mega a fim de proporcionar maior entendimento sobre a programação de microcontroladores e suas funcionalidades.

No primeiro experimento foi implementada a classe *UART* não bloqueante para enviar dados pela serial, em seguida foi adicionada a estrutura de dados *FIFO* para armazenamento dos dados enviados através da interface serial. O segundo experimento foi referente a classe *ExtInt* para tratamento de interrupções externas e a classe *PCINT* para tratamento de eventos não prioritários. Por último, as classes *Timer* e *Timeout* foram modificadas com o objetivo de obter interrupções de 16 bits.

2 Serial com interrupção

A classe *UART* tem o papel principal de realizar a transmissão e recepção de dados através de uma serial com interrupção. O construtor *UART* recebe como parâmetros a taxa de transmissão de dados (baud rate) e os tipos enumerados (enum) que definem características de transmissão, *DataBits_t*, *Parity_t* e *StopBit_t*. Os três enum's (enumeração) foram utilizados para simplificar três configurações habilitadas no registrador UCSR0C.

O enumerado *DataBits_t* cofigura o tamanho do quadro transmitido pela serial, utilizando esta *UART* pode ter de 6 a 8 bits. O enumerado *Parity_t* habilita e configura a geração de paridade, podendo ser sem paridade, com paridade par ou ímpar. A paridade é utilizada para verificar se houve erro no dado recebido. O enumerado *StopBit_t* habilita o número de bits usados para sinalizar o fim da comunicação para um único pacote, podendo ser 1 ou 2 bits.

Ainda no construtor, são habilitados a transmissão, a recepção e a interrupção por recebimento da *USART* no registrador *UCSR0B*.

Realiza-se um cálculo para encontrar o valor *UBRR0* a partir do valor de *baud rate*. A equação a seguir, onde a *F_CPU* é o valor da frequência da *CPU*, calcula a taxa de transmissão (*baud rate*) para o modo assíncrono.

$$MYUBRR = \frac{\frac{F\_CPU}{16}}{baud} - 1 \quad (1)$$

Essa classe tem funções para envio e recebimento de informação. A função *put*, tem como parâmetro um dado do tipo *uint8_t*, é utilizada para enviar até um byte de informação. A função *get*, retorna um dado do tipo *uint8_t*, recebe dados através da serial. Além dessas funções, a função *has_data* é usada para saber se há um novo dado para receber, além disso, as funções *tx_isr* e *rx_isr* são chamadas quando ocorre uma interrupção.

O funcionamento da transmissão acontece da seguinte forma: a função *put* recebe uma informação e guarda numa variável, em seguida coloca '1' no registrador *UCSR0B* na posição *UDRIE0*, essa posição do registrador setada em '1' habilita a interrupção de transmissão. A partir disso ocorre a interrupção que chama a função *tx_isr*, essa função escreve o dado recebido pela *put* no registrador *UDR0* e coloca '0' no registrador *UCSR0B* na posição *UDRIE0*, desabilitando a interrupção de transmissão.

O funcionamento da recepção decorre do seguinte modo: a função *rx_isr* é chamada pela interrupção de recepção, essa função lê o dado registrado no *UDR0* e seta para '*true*' a variável *_newdata* que informa a recepção de um dado. Com isso a função *get* quando chamada seta a *_newdata* em false e retorna o dado recebido.

O programa de teste realizado para esta classe instancia a UART com os seguintes parâmetros: o *baud rate* igual a 9600, tamanho de quadro igual 8 bits, sem paridade e um *stop bit*. Posteriormente, foi enviado o carácter a usando o *put* e então o programa entra em um *loop* de verificação de recebimento de dados, a função get sinaliza quando algum dado é recebido, em seguida soma-se '1' ao dado e reenvia com a função *put*. Para verificar esse funcionamento foi utilizado o programa Cutecom, a partir dele pode-se enviar um dado via serial (USB) para a placa, e certificar que o dado foi enviado/recebido de forma correta recebendo o mesmo dado via serial.

3 Serial com FIFO

A serial com FIFO é a classe *UART* com a adição de uma implementação de FIFO, com o objetivo de receber vários dados sem que ocorra perda de informação. Também foi incluído o enumerado *DoubleSpeed_t*, que habilita ou desabilita o bit '1' do registrado UCSR0A para configurar a taxa de transferência da comunicação, quando habilitado a taxa de transferência da comunicação assíncrona é duplicada.

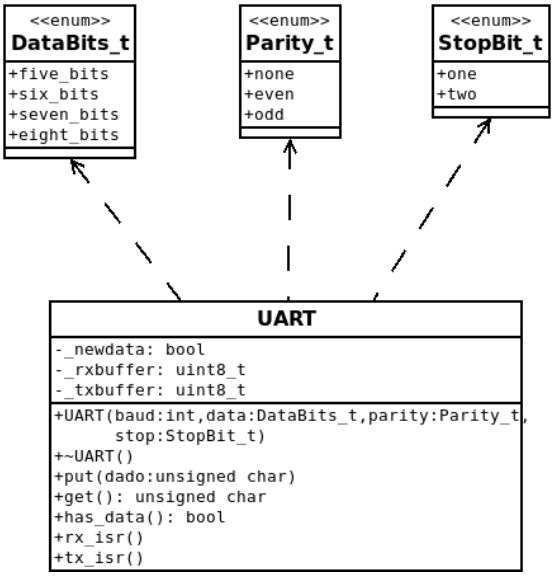

Figure 1: Diagrama de classe da UART. *Fonte: Próprio autor.*

O construtor dessa versão da classe UART, recebe além dos parâmetros da UART anterior, o *DoubleSpeed_t* e realiza o registro do mesmo no UCSR0A.

Para utilizar a FIFO na UART foi preciso fazer algumas modificações nas funções já existentes. Nesta versão, as variáveis que recebem e enviam os dados agora são do tipo FIFO, ou seja, foi criada uma lista que armazena os dados conforme recebimento e com retirada realizada de acordo com a ordem de chegada, sendo assim as funções da primeira versão da *UART* tiveram de ser adaptadas para utilizar as novas variáveis, além disso foi adicionada a função *puts* que envia uma *string*, vários caracteres seguidos.

A função put agora fica travada quando a FIFO está cheia e quando não está o dado é colocado na FIFO. A função get trava quando não tem dado e quando tem dado este será retornado.

O teste implementado para esta classe instancia a *UART* modificada com os mesmos parâmetros do teste da *UART* do primeiro experimento, porém com o acréscimo do *DoubleSpeed_t* desabilitado. Para testar o funcionamento, implementou-se um laço de repetição que envia uma *string* comprida, com o objetivo de testar a capacidade da FIFO e da mesma forma que o teste anterior foi utilizado o programa Cutecom, a partir do qual enviamos um dado junto com a string via serial (USB) para a placa, e certificamos que o dado foi enviado/recebido de forma correta recebendo o mesmo dado via serial.

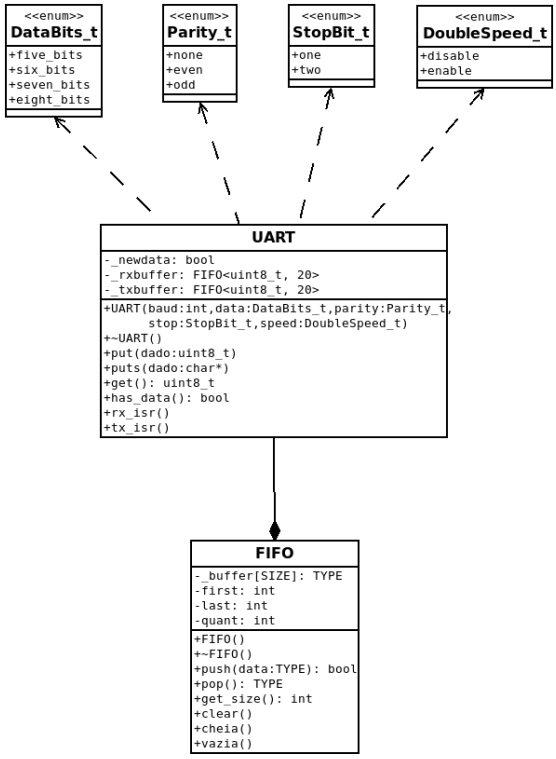

Figure 2: Diagrama de classe da UART. *Fonte: Próprio autor.*

4 Interrupção externa

No microprocessador ATMEGA 2560, as interrupções externas realizam a modificação de status do pino *INTn* para realização de funções estabelecidas. As interrupções são acionadas pelo pelos pinos *INTn*, onde n varia de 0 à 7.

Nos registradores EICRA e EICRB, a configuração de leitura de borda são realizadas podendo ser configuradas como "*rising*", "*falling*", "*low_level*" e "*any_edge*". Onde é importante observar que interrupções de baixo nível e de borda são detectadas de forma assíncrona.

A função *enable* seta '1' na posição da *INTn*, enquanto o *disable* seta '0' na posição da *INTn*. A função *callback* chama o ponteiro que guarda o endereço da função de *callback* a ser utilizada quando ocorre a interrupção.

O programa de teste para a classe ExtInt foi implementada um objeto para instanciar quatro INTn, a 0, 1, 4 e 5, cada uma foi instanciada utilizando uma configuração de detecção de borda diferente. Posteriormente, cada objeto foi habilitado usando o enable. Para analisar o funcionamento foi utilizado o Cutecom para enviar receber pela serial, além de montar o circuito a seguir, onde está *ToMicrocontroller* é onde sai os fios que vão para os pinos 20, 21, 2 e

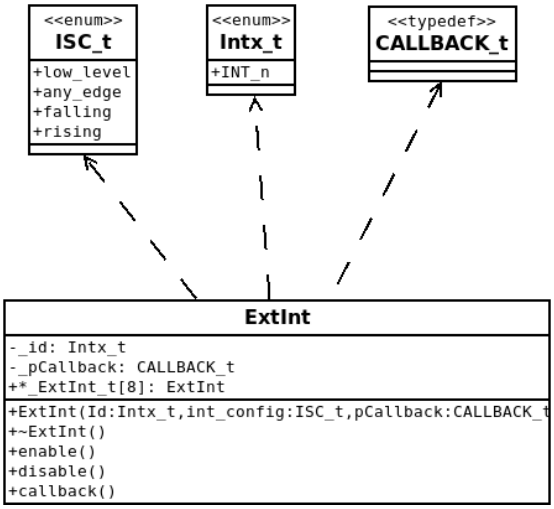

Figure 3: Diagrama de classe da ExtInt. *Fonte: Próprio autor.*

3.

5 PCINT

As interrupções por troca de pino *PCINT* são assíncronas, realizam a verificação dos pinos *PCINTn*, onde *n* varia de 0 à 23, para realização de funções estabelecidas. A classe *PCINT* possui como característica um tipo enumerado (*PCINTx_t*) que mapeia todos os registradores disponíveis para estas interrupções.

O construtor *PCINT* é apenas declarado, não possui implementação de manipulação de parâmetros. O objeto com o tipo enumerado é criado na função *enable*. A implementação das funções de gerenciamento (*manager*) e desabilitação (*disable* fazem o controle de utilização do recurso de interrupção. Além disso, um vetor trata do acontecimento de eventos (*events*) e outro vetor armazena o histórico de eventos recentes (*hist*). O retorno é dado através do vetor *_pCallbacks* que possui o registro de cada *PCINT*.

Na função de gerenciamento (*manager*) os registradores *PCMSK2*, *PCMSK1* e *PCMSK0* controlam quais pinos contribuem para as interrupções, é realizada a verificação do vetor de eventos e de cada PCMSK_n com o objetivo de certificar que a interrupção realmente aconteceu, enviando uma sinalização para o callback.

A função de habilitação da interrupção (*enable*) tem como parâmetro o id da interrupção que será configurada e o *callback*. Configura o registrador *PCICR* para interrupções, para *PCINTn* com n menor que 7 é utilizado o bit *PCIE0*, com n entre 7 e 15 habilitou-se o bit *PCIE1* e para n maior que 15 habilitou-se o bit *PCIE0*. Assim, a configuração e habilitação de todas as interrupções de pino foi realizada. Para desabilitar a interrupção a mesma lógica é utilizada, porém

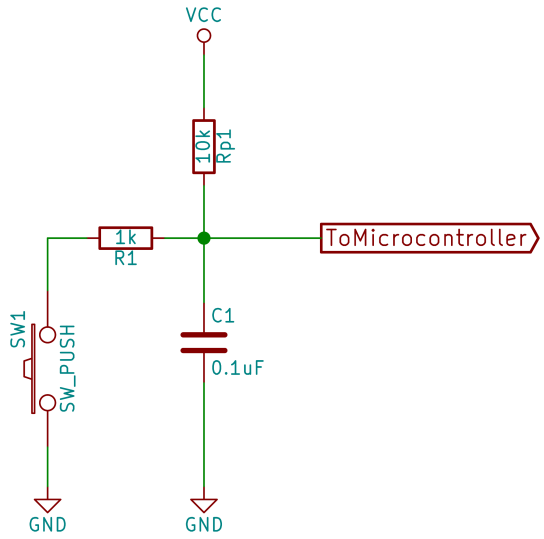

Figure 4: Circuito teste.

os registradores *PCMSK2*, *PCMSK1* e *PCMSK0* recebem zero para desabilitar a operação.

As interrupções são verificadas através do *PCINT0_vect*, *PCINT1_vect* e *PCINT2_vect*. Cada interrupção verifica o pino atual e configura no vetor de eventos um novo evento que depende do anterior e da habilitação do registrador *PCMSKn*, assim valida a interrupção e repassa as informações para a classe PCINT.

Para realização de testes para a classe PCINT foi implementada a instanciação de um objeto desta classe, em seguida foram habilitadas com a função *enable* as $PCINT_n$ com n igual a 4, 9, 10 e 16. Uma verificação recursiva foi produzido para analisar se algum dado é recebido na *UART*, depois tem um delay de 2 segundos e a função *manager* verifica se aconteceu alguma interrupção. Para executar esse programa teste foi montado o circuito a seguir. Na Figura 6, a flag (ToMicrocontroller) indica onde estão os fios que serão conectados nos pinos 10, 14, 15 e A8.

6 Timer e Timeout

O objetivo dessa atividade foi a implementação de dois *Timers* de interrupção, com 8 bits e 16 bits, sendo possível a escolha da configuração utilizada no construtor.

O construtor da classe *Timer* recebe como parâmetros a variável enumerada que identifica o número de bits do *timer* ($_t$) e a variável que possibilita o cálculo de frequência máxima de operação (freq), diferenciando através do tipo

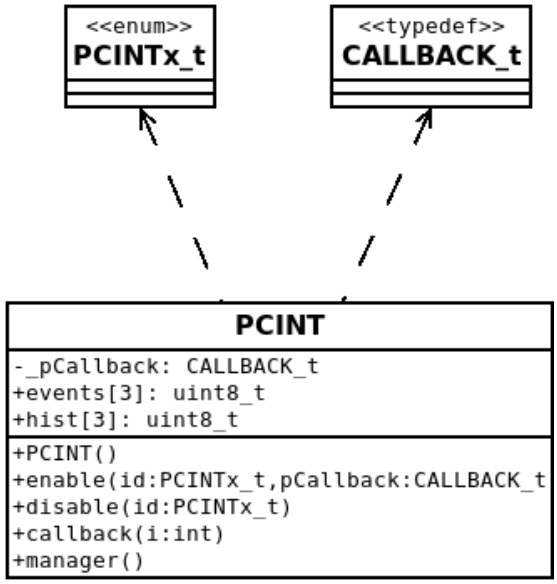

Figure 5: Diagrama de classe da PCINT. *Fonte: Próprio autor*

de operação. Para a utilização do *Timer* de 8 bits habilita-se o registrador *TCCT0A*, e configura o registrador *TCCR0B* para configurar parâmetros que afetam a frequência de operação. Além disso, o cálculo relativo a frequência de operação é realizado, o registrador *TCNT0* é habilitado com o tempo de utilização e o registrador de máscara de temporizador *TIMSK0* recebe valor '0x001' o que habilita a interrupção Timer/Counter0 Overflow por ter o bit *TOIE0* gravado em um.

As interrupções de *TIMER0* e *TIMER1* são executadas com funções que trabalham com a diferença de bits do *timer*. A função $ISR8\_handler()$ é executada quando a interrupção de *TIMER0* acontece, para isso o registrador *TCNT0* é habilitado, a contagem de *ticks* para interrupção sofre um acréscimo e o registrador de controle de *timeout* é verificado, para certificar-se de que o tempo está ou não finalizado. A função de interrupção do TIMER1 é semelhante, porém habilita a interrupção com *timer* de 16 bits, para isso o registrador *TCNT1* é habilitado.

Ainda nesta atividade, a classe *Timeout* foi desenvolvida com objetivo de ter o correto funcionamento do *Timer*. Possui atributos que tratam dos eventos que ocorrem no *timer*, e funções que fazem a sua manipulação. A função construtora não possui parâmetros de entrada, apenas inicializa as variáveis da classe. A função *config*, pode ser considerada a mais importante, pois configura o intervalo e *callback* atual para habilitar o *timeout*. A função *checkTimeout()* verifica o contador de *timeout*, ou seja, gerencia o tempo da interrupção.

Para o teste dessa classe não é necessário ter um circuito montado, pois apenas utilizou-se a classe UART para enviar caracteres de teste. Instanciou-se

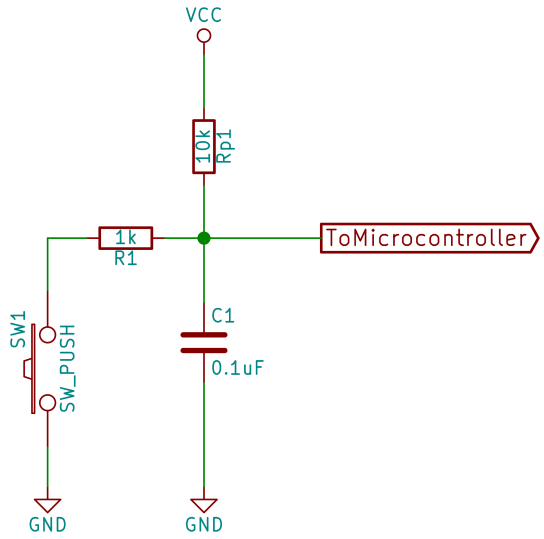

Figure 6: Circuito teste.

Figure 7: Diagrama de classe do Timer e Timeout.

a classe Timer com timer 8 e 16 separadamente, depois adicionou a função que ocorrerá na interrupção com o tempo do timeout de 5 segundos. Análogo a PCINT, há uma função chamada timeoutManager que verifica se a contagem do timeout acabou.