PSI3441 – Arquitetura de Sistemas Embarcados

Configuração do DMA no PE

Escola Politécnica da Universidade de São Paulo

Prof. Gustavo Rehder – grehder@lme.usp.br

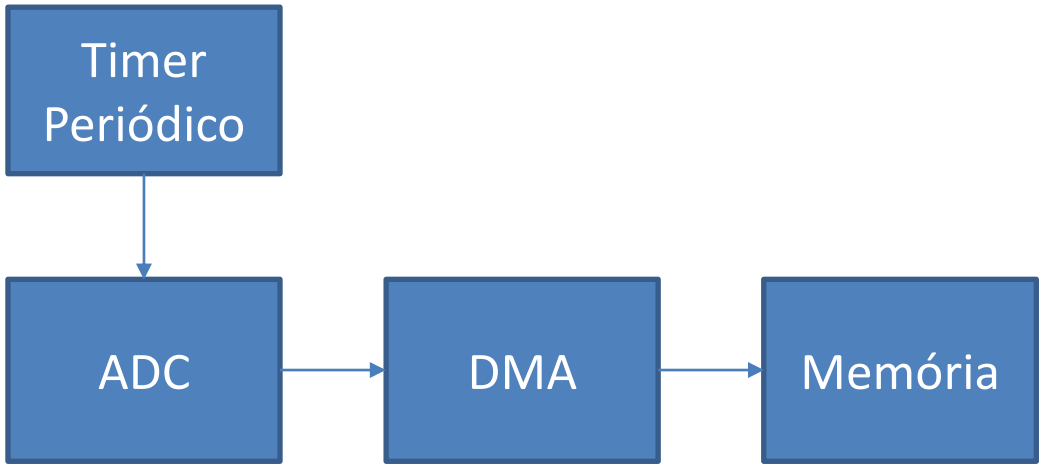
2018

Objetivo

- Configurar o DMA para transmitir os dados da aquisição analógica (ADC) para a memória.

Procedimento

- ADC realiza a conversão sincronizado (triggered) pelo timer periódico.
- O ADC ao terminar a conversão, requisita a transferência de dados ao DMA
- O DMA executa a transferência para a memória
- Ao terminar a transferência, os flags são limpos e o número de bits resetado



Procedimento

1. Configure um timer e certifique-se que ele está funcionando
2. Configure um canal de ADC e certifique-se que ele está sendo disparado pelo timer. Não esqueça de inicializar o trigger (AD1_EnableIntChanTrigger();)
3. Configure o DMA

Configuração do ADC

- Configurar canal
- Tempo de conversão
- Configurar Trigger (TPM0 overflow, por exemplo)
- Para habilitar a requisição de transferência ao controlador do DMA coloque a expressão no main (não em no loop infinito do main):

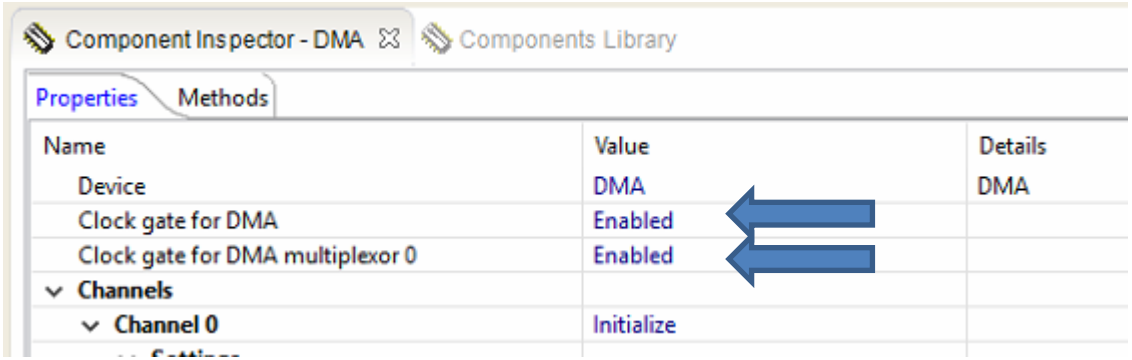
```
ADC0_SC2 |= ADC_SC2_DMAEN_MASK;    // DMA Enable
```

Obs: Habilitar o DMA pelo componente ADC resulta em erro!

Configuração do DMA

Utilize o componente Init_DMA

- Habilite o clock (clock gate for DMA e clock gate for DMA for Multiplexor)



Códigos adicionados no DMA.c, habilitando os clocks

SIM_SCGC7 |= SIM_SCGC7_DMA_MASK; /* SIM_SCGC7: DMA=1 */

SIM_SCGC6 |= SIM_SCGC6_DMAMUX_MASK; /* SIM_SCGC6: DMAMUX=1 */

- Inicialize o Canal 0

  DMAMUX0_CHCFG0 = DMAMUX_CHCFG_ENBL_MASK

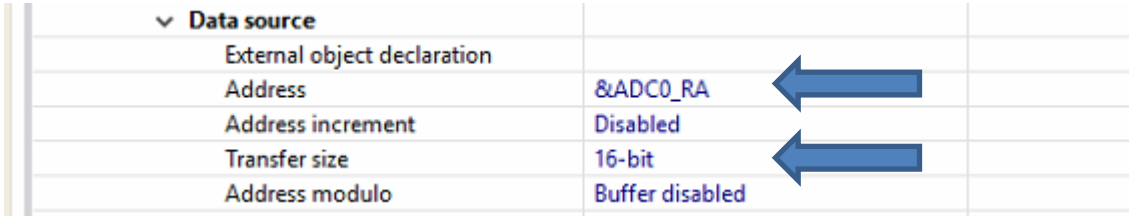
  /* #define DMAMUX_CHCFG_ENBL_MASK 0x80u */

  - Transfer mode: **cycle steal**

  DMA_DCR0 = DMA_DCR_CS_MASK

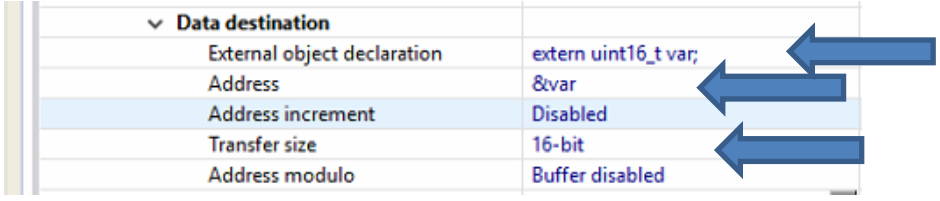
  /* #define DMA_DCR_CS_MASK 0x20000000u */

- Data Source: /* endereço dos dados a serem transferidos */
  - Address: **&ADC0_RA** DMA_SAR0 = (uint32_t)(&ADC0_RA);
  - Transfer Size: **16 bits** DMA_DCR0 = DMA_DCR_SSIZE(0x02)

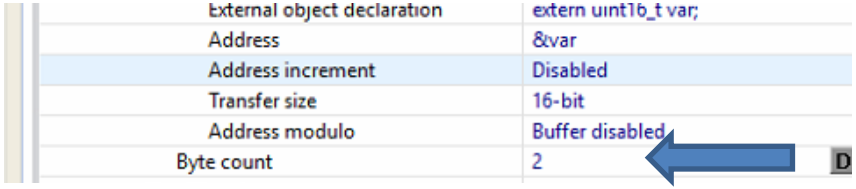


/* adiciona **extern uint16_t var;** no DMA.c para criar a variável global */

- Data Destination: /* endereço dos dados a serem recebidos*/
  - External Object Declaration: **extern uint16_t var;**
  - Address: **&var** DMA_DAR0 = (uint32_t)(&var);
  - Transfer Size: **16 bits** DMA_DCR0 = DMA_DCR_DSIZE(0x02)



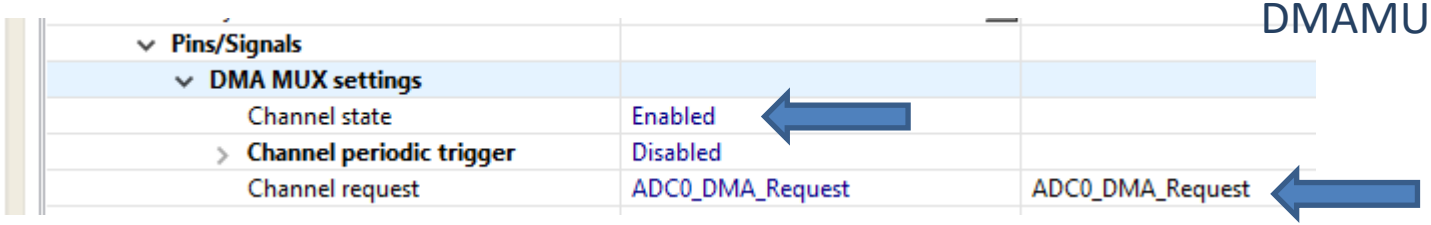
- Byte Count: **2** /* número de bytes a serem transferidos */



DMA_DSR_BCR0 = DMA_DSR_BCR_BCR(0x02);

Configuração do DMA

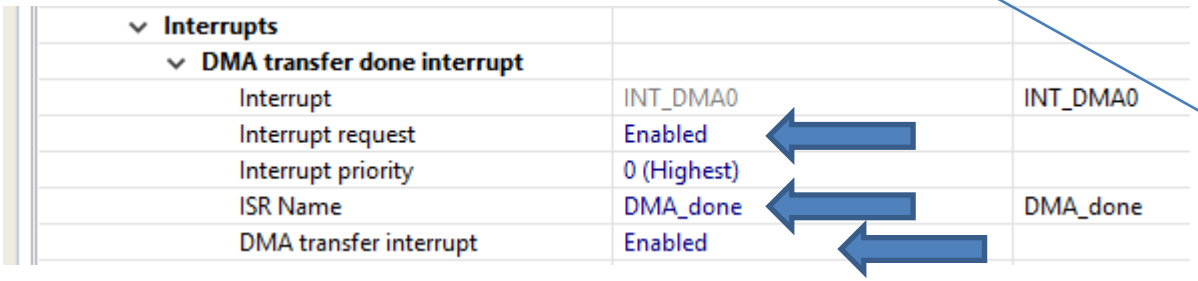
- Pin/Signal – DMA Mux settings: /* configura o mux do DMA */
  - Channel State: **Enable** DMAMUX0_CHCFG0 = DMAMUX_CHCFG_ENBL_MASK
  - Channel Request: **ADC0_DMA_Request** DMAMUX0_CHCFG0 = DMAMUX_CHCFG_SOURCE(0x28)



- Interrupt – DMA transfer done interrupt /*configura interrupção após transferência*/
  - Interrupt Request: Enable NVIC_ISER |= 0x0000 0001 (habilita a interrupção do IRQ0)
  - ISR Name: Dê um nome da interrupção Ex. DMA_done
  - DMA transfer interrupt: Enable

Define em vectors.c o *Handler* da interrupção do canal 0 do DMA (IRQ 0) como **DMA_done**.

DMA_DCR0 = DMA_DCR_EINT_MASK



- Initialization – External Request : Enable /*habilita requisições de periféricos */

DMA_DCR0 = DMA_DCR_ERQ_MASK /*#define DMA_DCR_ERQ_MASK 0x40000000u */

Configuração do DMA

- Declare no main.c a variável var como global (i.e. fora da função do main): uint16_t var;
- No Events.c, crie a função da interrupção (*Handler*):

```
PE_ISR(DMA_done)
{
DMA_DSR0 |= DMA_DSR_BCR_DONE_MASK;      // Clear Done Flag
DMA_DSR_BCR0 |= DMA_DSR_BCR_BCR(2);     // Set byte count register
}
```