SENAI

ARM Cortex M0+

HAL

Prof° Fernando Simplicio

SENAI

ARM Cortex M0+

UART

Prof° Fernando Simplicio

UART0/1/n

- **Full Duplex**
- **Tanto Rx e Tx operam com o mesmo BaudRate.**
- **Buffer de recepção RX duplo.**
- **Suporta 4 diferentes fontes de entrada de clock (BUSCLK) através do registrador SIM->SOPT2, sendo:**
  - **MCGFLLCLK**
  - **MCGPLLCLK/2**
  - **OSCERCLK,**
  - **MCGIRCLK (suitable for low-power modes).**

UART0/1/n

- **BaudRate com prescaler de 13 bits (SBR [12:0]) onde UARTn[BDH] armazena 5 bits (High) e UARTn[BDL] armazena 8 bits (Low).**
- **Devemos configurar o BaudRate com a UART desabilitada (UARTn->C2=0;).**
- **Cálculo do BaudRate da UART:**

$$\text{UART0 baud rate} = \frac{\text{Source Clock Frequency}}{\text{SBR}*(\text{UART0\_}C4[\text{OSR}]+1)}$$

$$\text{All Other UART baud rate} = \frac{\text{Bus Clock Frequency}}{\text{SBR}*16}$$

Exemplo UART0

- **Exemplo: Cálculo do BaudRate da UART0 a 115200 bps:**
- **UART0_C4[OSR]= 0x03 (acquisition rate of 4–32 samples per bit time)**
- **Select the MCGFLLCLK as the source clock:**
  - **SIM_SOPT2[UART0SRC] = 1.**
  - **SIM_SOPT2[PLLFLLSEL] = 0.**

$$SBR = \frac{\text{UART0 source clock frequency}}{\text{baud rate}*(UART0\_C4[OSR]+1)} = \frac{48000000}{115200*(3+1)} = 104 \quad = 0x68$$

$$BDH = 0x00$$

$$BDL = 0x68$$

Exemplo UART1

- **Exemplo: BaudRate da UART1 configurada a 2400 bps.**
- **Assumindo:**
  - **System clock of 48 MHz.**
  - **Bus Clock divider of 2 (SIM_CLKDIV1[OUTDIV4] = 1), resuntando Bus Clock de 24MHz.**

$$SBR = \frac{\text{Bus clock frequency}}{\text{baud rate*16}} = \frac{24000000}{2400*16} = 625 = 0x271$$

$$BDH = 0x02$$

$$BDL = 0x71$$

Recursos Especiais UARTn

- **Idle-line wake up:** **Detecta automaticamente o caractere ‘n’ (idle character) quando UART0_C1[WAKE] = 0; Ao receber ‘n’ o registrador UART0_C2[RWU] é igual a zero.**
- **Address mark wake up:** **Quando UART0_C1[WAKE]=1 o bit UART0_C2[RWU] é zerado quando o bit MSB do caractere recebido for 1.**
- **Match address operation: Quando UART0_C4[MAEN1] e UART0_C4[MAEN2] assumirem 1, o caractere recebido com o bit MSB em 1 (UART0_C1[WAKE]=1) será comparado com UART0_MA1 e UART0_MA2. Quando comparado com sucesso, o bit UARTx_S1[RDRF] será 1. Todos os frames seguidos com bit MSB em zero serão recebidos e armazenados no buffer.**

Recursos Especiais UARTn

- **Modo de Transmissão em 8, 9 e 10 bits. (UARTx_C4[M10] =0; UARTx_C1[M] = N). Para dados maiores que 8 bits deve-se configurar UARTx_C3 [R8/T9 and R9/T8]. Esta configuração deverá ser feita antes de enviar o caractere em UARTx_D.**
- **Sinalizações:**
  - **UARTx_S1[TDRE] indica que o buffer de transmissão está vazio, portanto podemos enviar um caractere em UARTx_D.**
  - **UARTx_S1[TC] indica que a transmissão de um caractere foi completada.**
  - **UARTx_S1[RDRF] indica que o caractere recebido foi armazenado no buffer e está pronto para ser lido.**
  - **UARTx_S1[NE] indica e noise error.**
  - **UARTx_S1[FE] indica erro de frame.**
  - **UARTx_S1[PE] indica erro de paridade.**

Exemplo de Programa

```
void UART_init(void *UARTx, uint32_t Baud) {
    uint16_t tmp;

    if ((UARTLP_Type *) UARTx == UART0) {
        SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;          /* Enable GPIOA clock    */
        SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;          /* Enable UART0 clock    */

        PORTA->PCR[1] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2);
        PORTA->PCR[2] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2);

        ((UARTLP_Type *) UARTx)->C1 = 0x00;
        ((UARTLP_Type *) UARTx)->C2 = UARTLP_C2_RE_MASK | UARTLP_C2_TE_MASK;
        ((UARTLP_Type *) UARTx)->C3 = 0x00;
        ((UARTLP_Type *) UARTx)->C4 = 0x00;
        ((UARTLP_Type *) UARTx)->C5 = 0x00;
        SIM->SOPT2 &= ~(SIM_SOPT2_PLLFLLSEL_MASK);  /* MCGFLLCLK             */
        SIM->SOPT2 |= SIM_SOPT2_UART0SRC(1);         /* Using MCGFLLPLL       */

        tmp = SystemCoreClock / Baud;
        ((UARTLP_Type *) UARTx)->BDH &= ~UARTLP_BDH_SBR_MASK;
        ((UARTLP_Type *) UARTx)->BDH |= UARTLP_BDH_SBR(tmp);
        ((UARTLP_Type *) UARTx)->BDL &= ~UARTLP_BDL_SBR_MASK;
        ((UARTLP_Type *) UARTx)->BDL |= UARTLP_BDL_SBR(tmp);
    }
}
```

Exemplo de Programa

```
void UART_sendchar(void *UARTx, char c) {
    if ((UARTLP_Type *) UARTx == UART0) {
        /* Check if transmit data register is empty */
        while (((UARTLP_Type *) UARTx)->S1 & UARTLP_S1_TDRE_MASK)
            ;
        /* Send the data */
        ((UARTLP_Type *) UARTx)->D = c;
    }
    else {
        /* Check if transmit data register is empty */
        while (((UART_Type *) UARTx)->S1 & UARTLP_S1_TDRE_MASK)
            ;
        /* Send the data */
        ((UART_Type *) UARTx)->D = c;
    }
}
```

Exemplo de Programa

```
char UART_receivechar(void *UARTx) {
    char c;

    if ((UARTLP_Type *) UARTx == UART0) {
        /* Wait while receive data register is empty */
        while ((((UARTLP_Type *) UARTx)->S1 & UARTLP_S1_RDRF_MASK) == 0)
            ;
        /* receive the data */
        c = ((UARTLP_Type *) UARTx)->D;
    }
    else {
        /* Wait while receive data register is empty */
        while ((((UART_Type *) UARTx)->S1 & UARTLP_S1_RDRF_MASK) == 0)
            ;
        /* receive the data */
        c = ((UART_Type *) UARTx)->D;
    }

    return c;
}
```

Exemplo de Programa

```
void UART_PutStr(void *UARTx, uint8* str)
{
    uint16 1=0;
    while(str[i] != 0)
    {
        while( !(((UARTLP_Type *) UARTx)->S1&UART_S1_TDRE_MASK));
        ((UARTLP_Type *) UARTx)->D = str[i];
        i++;
    }
}
```

SENAI

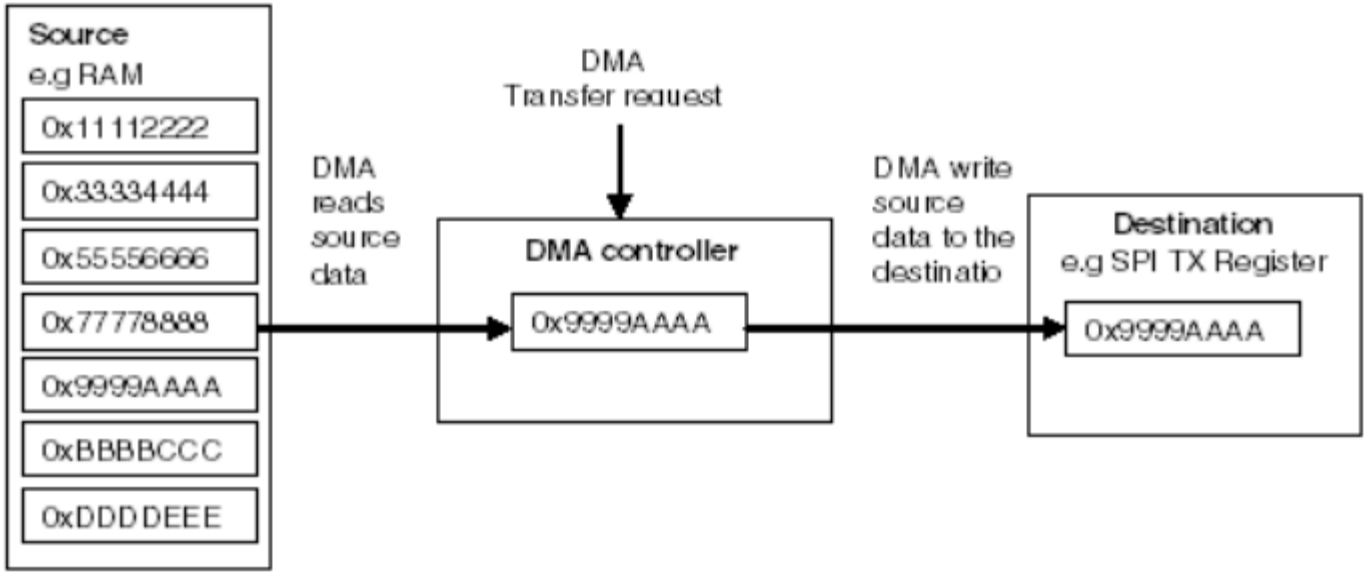
ARM Cortex M0+

Direct Memory Access (DMA) Controller

Prof° Fernando Simplicio

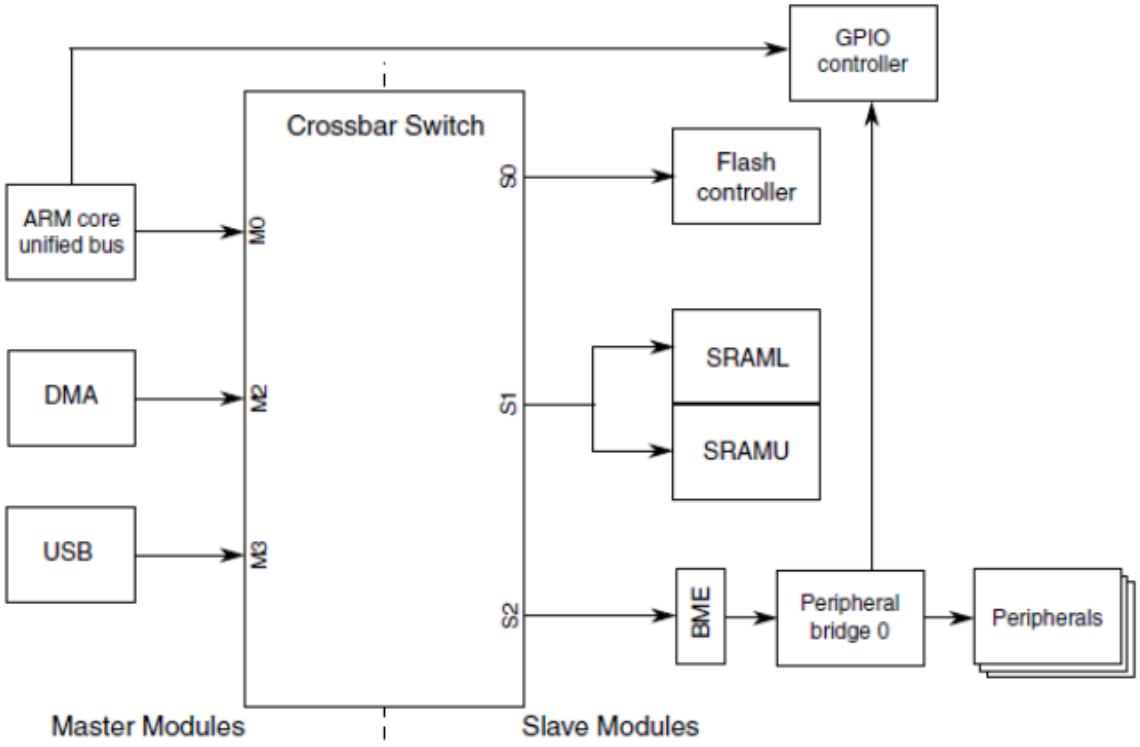
DMA

- **Dispositivo utilizado para trocar/movimentar dados entre diferentes posições de memórias e periféricos, sem a intervenção da CPU.**
- **Trabalha em paralelo com a CPU. Aumenta o desempenho do projeto pois a CPU fica menos sobrecarregada.**



DMA

- **Contém um buffer de 32 bits para a passagem temporária dos dados entre memórias.**
- **Baseia-se em um sistema multi-master onde a CPU e o DMA podem acessar simultaneamente um slave.**



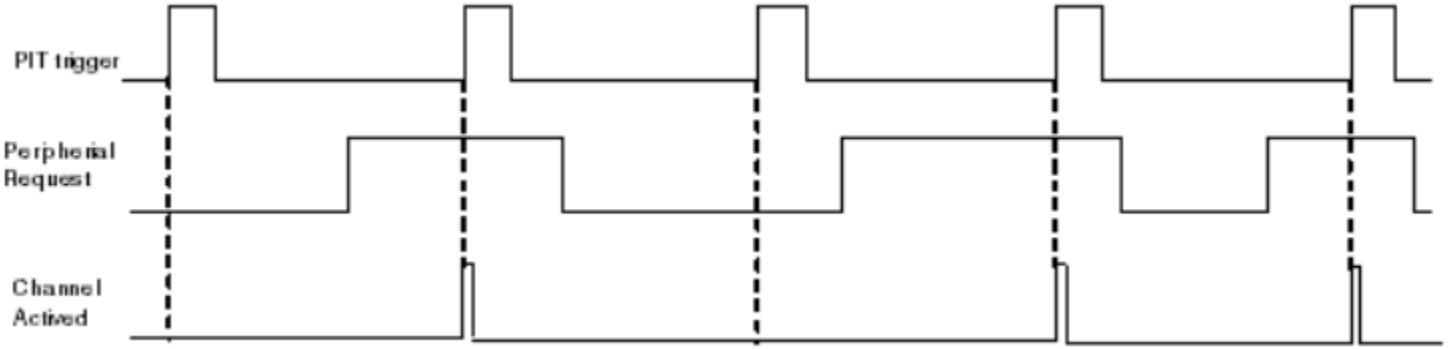
Acessos simultâneos a um slave são tratados de acordo com a prioridade de cada master.

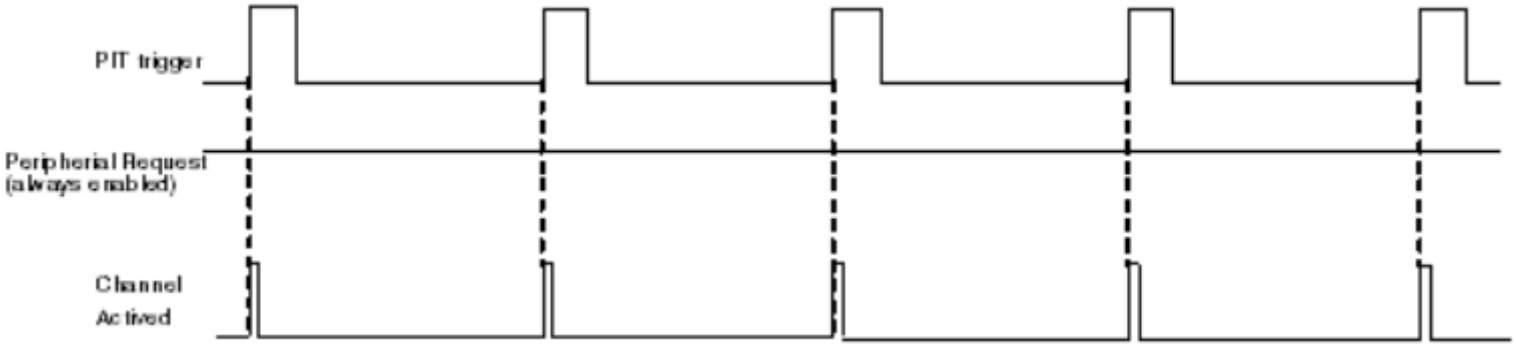
DMA

- **Inicialmente o DMA deverá ser habilitado por DMA_DCRn[START].**
- **Os modos do DMA são:**
  - **Disabled mod;**
  - **Normal mode**
  - **Periodic Trigger mod.**

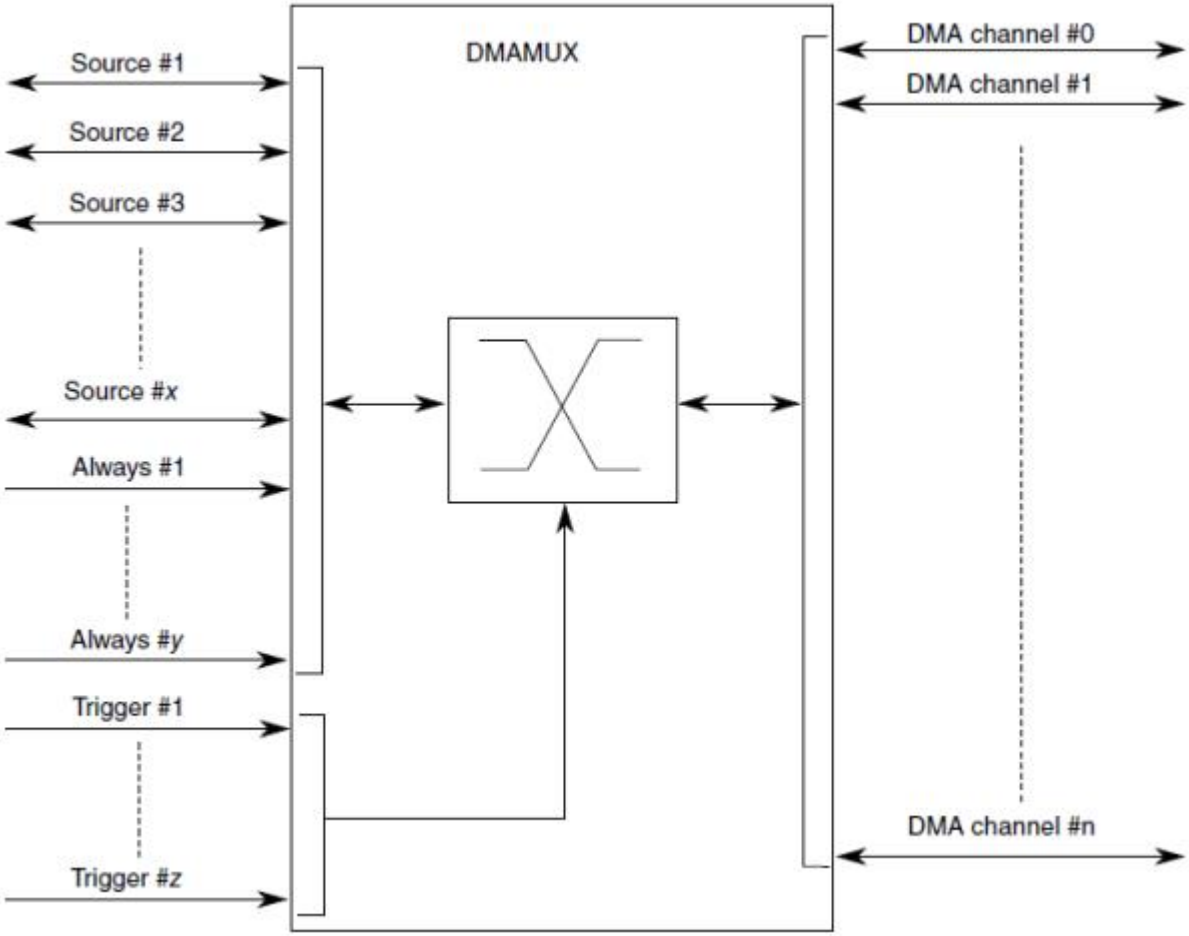
- **No modo Periodico quem determina o tempo de amostragem do DMA é o temporizador PIT. Ideal para monitoramento de status de registradores ou para transferência de dados periódicos.**

DMA


PIT trigger
Peripherial Request
Channel Actived

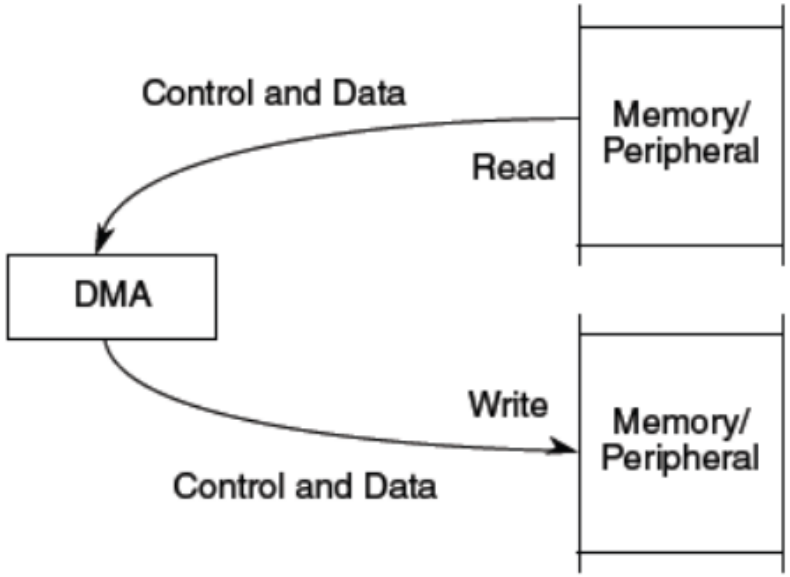

PIT trigger
Peripherial Request (always enabled)
Channel Actived

DMA


DMAMUX
Source #1
Source #2
Source #3
Source #x
Always #1
Always #y
Trigger #1
Trigger #z
DMA channel #0
DMA channel #1
DMA channel #n

DMA

- **Cada *channel* do DMA é capaz de transferir dados de 8, 16 ou 32 bits de um endereço de memória para outro.**
- **Existe um grupo de registradores responsáveis pelos endereçamentos *source*, destino e de controle.**



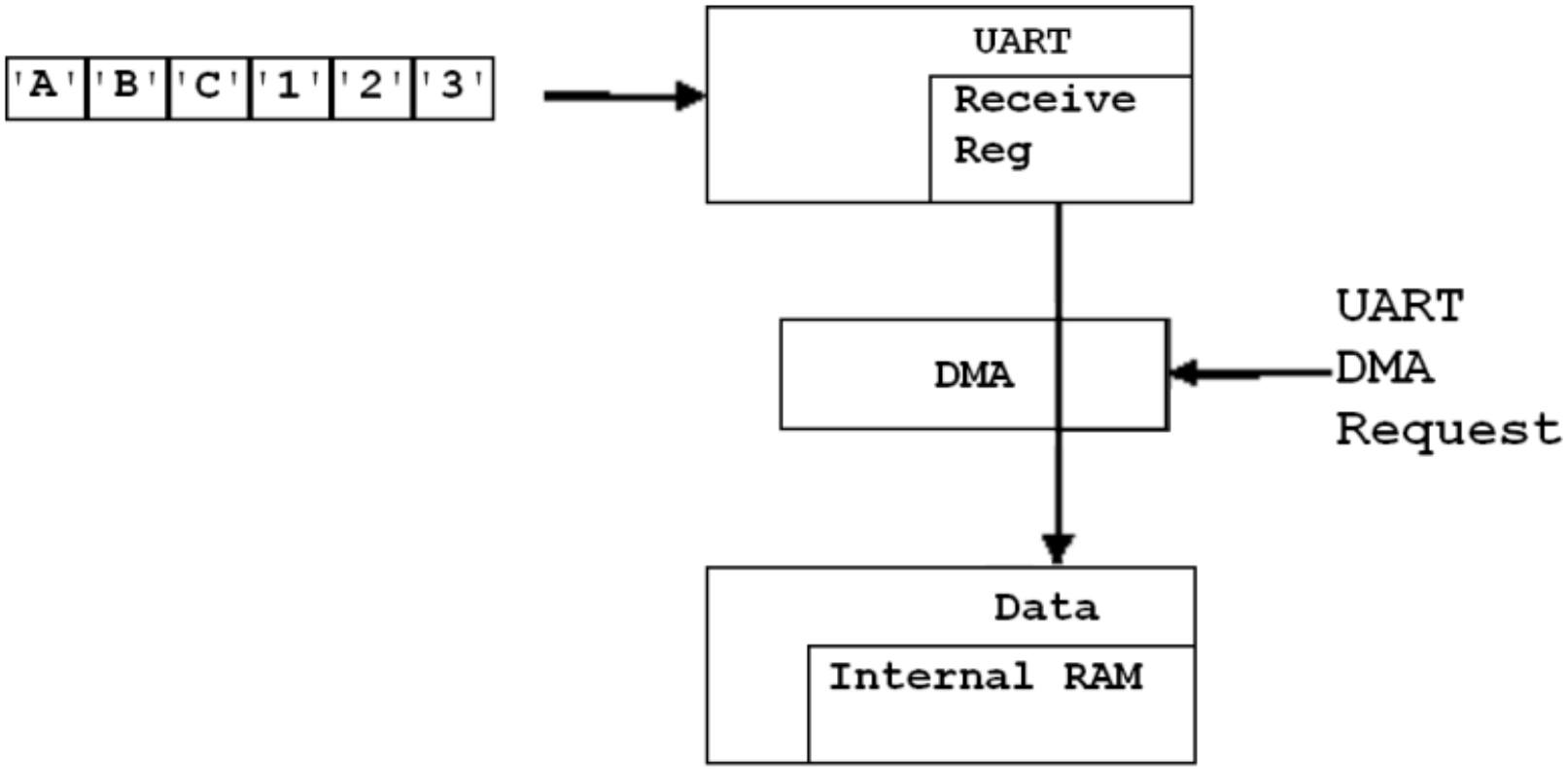
DMA

- **Para inicializar o canal é necessário configurar o endereço de apontamento, contador de bytes e o registrador de controle do DMA.**
- **Transferencia dos dados por meio do endereço de apontamento.**
- **Encerramento da transferencia: O status da transferencia de dados permanece em DSRn e a contagem de bytes em BCRn.**
- **O canal de menor índice possui maior prioridade que os demais canais.**

Configuração do DMA

1. **Habilitar o clock do DMA e o DMA MUX.**
2. **Se necessário desabilite inicialmente todos os canais do DMA através DMAMUX_CHCFGn = 0;**
3. **Apagar os flags de erros e acknowledge e sucesso de transferencias.**
4. **Escrever o endereço do source, endereço de destino, número de caracteres/status e configurar o registrador de controle de transferencia do DMA.**
5. **Configure o DMA MUX para rotear o sinal pelo canal escolhido.**

Exemplo UART - DMA



Exemplo UART - DMA

- **Cada caractere recebido é transferido pelo DMA para um endereço da memória RAM.**
- **Após cada confirmação de transferência o endereço de apontamento é incrementado.**
- **Cada transferência é feita quando o flag indicador de buffer “cheio” for sinalizado pela UART.**

Exemplo UART - DMA

```
// Disable DMA MUX channel first
DMAMUX0_CHCFG0 = 0x00;
// Clear pending errors and/or the done bit
if (((DMA_DSR_BCR0 & DMA_DSR_BCR_DONE_MASK) == DMA_DSR_BCR_DONE_MASK)
    | ((DMA_DSR_BCR0 & DMA_DSR_BCR_BES_MASK) == DMA_DSR_BCR_BES_MASK)
    | ((DMA_DSR_BCR0 & DMA_DSR_BCR_BED_MASK) == DMA_DSR_BCR_BED_MASK)
    | ((DMA_DSR_BCR0 & DMA_DSR_BCR_CE_MASK) == DMA_DSR_BCR_CE_MASK))
DMA_DSR_BCR0 |= DMA_DSR_BCR_DONE_MASK;
// Set Source Address (this is the UART0_D register
DMA_SAR0 = SOURCE_ADDRESS;
// Set BCR to know how many bytes to transfer
DMA_DSR_BCR0 = DMA_DSR_BCR_BCR(32);
// Clear Source size and Destination size fields.
DMA_DCR0 &= ~(DMA_DCR_SSIZE_MASK
    | DMA_DCR_DSIZE_MASK
);
```

Exemplo UART - DMA

```
// Set DMA as follows:
// Source size is byte size
// Destination size is byte size
// D_REQ cleared automatically by hardware
// Destination address will be incremented after each transfer
// Cycle Steal mode
// External Requests are enabled
// Asynchronous DMA requests are enabled.
DMA_DCR0 |= (DMA_DCR_SSIZE(1)
    | DMA_DCR_DSIZE(1)
    | DMA_DCR_D_REQ_MASK
    | DMA_DCR_DINC_MASK
    | DMA_DCR_CS_MASK
    | DMA_DCR_ERQ_MASK
    | DMA_DCR_EADREQ_MASK
    | DMA_DCR_EINT_MASK
);
// Set destination address
DMA_DAR0 = DESTINATION_ADDRESS;
```

SENAI

ARM Cortex M0+

Direct Memory Access (DMA) Controller

Exemplo Prático

Prof° Fernando Simplicio

TIMER (PIT) + ADC + DMA

- **A cada estouro de 1s do TIMER (PIT) é feito uma amostragem pelo conversor AD configurado em 16bits. O valor de cada conversão é enviado pelo DMA para uma variável buffer (RAM).**
- **O valor do buffer pode ser apresentado na janela de depuração do KDS.**

Programação do DMA

```
void dma_init(void)
{
    ready = 0;
    //Habilita o clocks do DMA
    SIM_SCGC6 |= SIM_SCGC6_DMAMUX_MASK;
    SIM_SCGC7 |= SIM_SCGC7_DMA_MASK;

    //Desabilita primeiro todos os DMA Mux channel
    DMAMUX0_CHCFG0 = 0x00;

    // Configure DMA
    DMA_SAR0 = (uint32_t)&ADC0_RA;      //Carrega o endereco (SOURCE) do buffer do conversor ADC no DMA
    DMA_DAR0 = (uint32_t)&value;        //Carrega o endereco (Destino) da variável Buffer;
    DMA_DSR_BCR0 = DMA_DSR_BCR_BCR(2);  //Informa quantos bytes será transferido por vez.
                                        //Neste caso, 2 bytes (16 bits) por vez.
    DMA_DCR0 |= (DMA_DCR_EINT_MASK|     // Habilita a interrupt do DMA (acionada a cada transferencia realizada)
                 DMA_DCR_ERQ_MASK |     // Habilita peripheral request, ou seja, quem vai disparar a transferencia
                                        // será um periférico. Podemo também usar o bit de START.
                 DMA_DCR_CS_MASK  |     // Forces a single read/write transfer per request.
                 DMA_DCR_SSIZE(2) |     // Set source size to 16 bits
                 DMA_DCR_DINC_MASK|     // Set increments to destination address
                 DMA_DCR_DMOD(1)  |     // Destination address modulo of 16 bytes
                 DMA_DCR_DSIZE(2));     // Set destination size of 16 bits


    // Enable DMA channel and source
    DMAMUX0_CHCFG0 |= DMAMUX_CHCFG_ENBL_MASK | DMAMUX_CHCFG_SOURCE(40); // Enable DMA channel and set ADC0 as source
    // Enable interrupt
    NVIC_EnableIRQ(DMA0_IRQn);
}
```

Programação do DMA

23.3.4 DMA Control Register (DMA_DCR*n*)

Address: 4000_8000h base + 10Ch offset + (16d × i), where i=0d to 3d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EINT	ERQ	CS	AA	0			Reserved	EADREQ	SINC	SSIZE		DINC	DSIZE		0
W																START
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SMOD				DMOD				D_REQ	0	LINKCC		LCH1		LCH2	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Programação do DMA

Field	Description
31 EINT	Enable interrupt on completion of transfer Determines whether an interrupt is generated by completing a transfer or by the occurrence of an error condition. 0 No interrupt is generated. 1 Interrupt signal is enabled.
30 ERQ	Enable peripheral request **CAUTION:** Be careful: a collision can occur between the START bit and D_REQ when the ERQ bit is 1. 0 Peripheral request is ignored. 1 Enables peripheral request to initiate transfer. A software-initiated request (setting the START bit) is always enabled.
29 CS	Cycle steal 0 DMA continuously makes read/write transfers until the BCR decrements to 0. 1 Forces a single read/write transfer per request.
22 SINC	Source increment Controls whether the source address increments after each successful transfer. 0 No change to SAR after a successful transfer. 1 The SAR increments by 1, 2, 4 as determined by the transfer size.

Programação do DMA

22 SINC	Source increment Controls whether the source address increments after each successful transfer. 0 No change to SAR after a successful transfer. 1 The SAR increments by 1, 2, 4 as determined by the transfer size.
21–20 SSIZE	Source size Determines the data size of the source bus cycle for the DMA controller. 00 32-bit 01 8-bit 10 16-bit 11 Reserved (generates a configuration error (DSRn[CE]) if incorrectly specified at time of channel activation)
19 DINC	Destination increment Controls whether the destination address increments after each successful transfer. 0 No change to the DAR after a successful transfer. 1 The DAR increments by 1, 2, 4 depending upon the size of the transfer.
18–17 DSIZE	Destination size Determines the data size of the destination bus cycle for the DMA controller. 00 32-bit 01 8-bit
	10 16-bit 11 Reserved (generates a configuration error (DSRn[CE]) if incorrectly specified at time of channel activation)
16 START	Start transfer 0 DMA inactive 1 The DMA begins the transfer in accordance to the values in the TCDn. START is cleared automatically after one module clock and always reads as logic 0.

Programação do DMA

```
/*
 * Handles DMA0 interrupt
 * Resets the BCR register and clears the DONE flag
 * */
void DMA0_IRQHandler(void)
{
    /* Enable DMA0*/
    DMA_DSR_BCR0 |= DMA_DSR_BCR_DONE_MASK;  // Clear Done Flag
    DMA_DSR_BCR0 |= DMA_DSR_BCR_BCR(2);     // Set byte count register
    ready += 1;
}
```

24 DONE	Transactions done Set when all DMA controller transactions complete as determined by transfer count, or based on error conditions. When BCR reaches zero, DONE is set when the final transfer completes successfully. DONE can also be used to abort a transfer by resetting the status bits. When a transfer completes, software must clear DONE before reprogramming the DMA. 0 DMA transfer is not yet complete. Writing a 0 has no effect. 1 DMA transfer completed. Writing a 1 to this bit clears all DMA status bits and should be used in an interrupt service routine to clear the DMA interrupt and error bits.
23–0 BCR	This field contains the number of bytes yet to be transferred for a given block. **Restriction:** BCR must be written with a value equal to or less than 0F_FFFFh. After being written with a value in this range, bits 23-20 of BCR read back as 1110b. A write to BCR of a value

Table continues on the next page...

Programação do TIMER (PIT)

- **A cada estouro de 1s do TIMER (PIT) é feito uma amostragem pelo conversor AD configurado em 16bits.**

```
void pit_init(void)
{
    // Habilita PIT clock
    SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;

    // Enable Green LED clock and MUX
    SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;
    PORTB_PCR19 = PORT_PCR_MUX(1);
    GPIOB_PDDR |= (1 << LED_GREEN);
    GPIOB_PSOR |= (1 << LED_GREEN);

    // Turn on PIT
    PIT_MCR = 0;

    // Inicializa o TIMER PIT para gerar uma interrupção a cada 1 segundo
    PIT_LDVAL0 = 0x1312CFF; // 1/20Mhz = 50ns   (1s/50ns)-1= 19,999,999 cycles or 0x1312CFF
    PIT_TCTRL0 |= PIT_TCTRL_TIE_MASK | PIT_TCTRL_TEN_MASK; // Enable interrupt and enable timer

    // Enable interrupt registers ISER and ICPR
    NVIC_EnableIRQ(PIT_IRQn);
}
```

Programação do TIMER (PIT)

- **Interrupção PIT + Start de Conversão do AD.**

```
/*  Handles PIT interrupt if enabled
 *
 *  Starts conversion in ADC0 with single ended channel 8 (PTB0) as input
 *
 * */
void PIT_IRQHandler(void)
{
    // Apaga o flag de interrupt do TIMER PIT
    PIT_TFLG0 = PIT_TFLG_TIF_MASK;

    // Write to SC1A to start conversion with channel 8 PTB0
    ADC0_SC1A = (ADC_SC1_ADCH(ADC_CHANNEL) |
                 (ADC0_SC1A & (ADC_SC1_AIEN_MASK | ADC_SC1_DIFF_MASK)));

    // Toggle Green LED a cada 1 segundo
    GPIOB_PTOR = (1 << LED_GREEN);
}
```

Programação do TIMER (PIT)

- **PIT_MCR**

PIT_MCR field descriptions

Field	Description
0–28 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
29 Reserved	This field is reserved.
30 MDIS	Module Disable - (PIT section) Disables the standard timers. The RTI timer is not affected by this field. This field must be enabled before any other setup is done. 0 Clock for standard PIT timers is enabled. 1 Clock for standard PIT timers is disabled.
31 FRZ	Freeze Allows the timers to be stopped when the device enters the Debug mode. 0 Timers continue to run in Debug mode. 1 Timers are stopped in Debug mode.

Programação do TIMER (PIT)

- **PIT (Contador decrescente)**

PIT_TCTRL*n* field descriptions

Field	Description
0–28 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
29 CHN	Chain Mode When activated, Timer n-1 needs to expire before timer n can decrement by 1. Timer 0 can not be changed. 0 Timer is not chained. 1 Timer is chained to previous timer. For example, for Channel 2, if this field is set, Timer 2 is chained to Timer 1.
30 TIE	Timer Interrupt Enable When an interrupt is pending, or, TFLGn[TIF] is set, enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TFLGn[TIF] must be cleared first. 0 Interrupt requests from Timer n are disabled. 1 Interrupt will be requested whenever TIF is set.
31 TEN	Timer Enable Enables or disables the timer. 0 Timer n is disabled. 1 Timer n is enabled.

Programação do ADC

```
/*  adc_init()
 * Calibrates and initializes adc to perform single conversions and generate
 * DMA requests at the end of the conversion
 *
 * */
void adc_init(void)
{
    // Enable clocks
    SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;   // ADC 0 clock
    SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;  // PTB0 clock

    // Calibrate ADC
    adc_cal();

    // Configure ADC
    ADC0_CFG1 = 0; // Reset register
    ADC0_CFG1 |= (ADC_CFG1_MODE(3)  |   // 16 bits mode
                  ADC_CFG1_ADICLK(0)|   // Input Bus Clock (20-25 MHz out of reset (FEI mode))
                  ADC_CFG1_ADIV(1)) ;   // Clock divide by 2 (10-12.5 MHz)

    ADC0_SC2 |= ADC_SC2_DMAEN_MASK;    // DMA Enable

    ADC0_SC3 = 0; // Reset SC3

    ADC0_SC1A |= ADC_SC1_ADCH(31); // Disable module
}
```

Programação do ADC

```
/*unsigned short    adc_read(unsigned char ch)
 *  Reads the specified adc channel and returns the 16 bits read value
 *
 *  ch -> Number of the channel in which the reading will be performed
 *  Returns the -> Result of the conversion performed by the adc
 *
 * */
unsigned short adc_read(unsigned char ch)
{
    ADC0_SC1A = (ch & ADC_SC1_ADCH_MASK) |
                (ADC0_SC1A & (ADC_SC1_AIEN_MASK | ADC_SC1_DIFF_MASK));     // Write to SC1A to start conversion
    while(ADC0_SC2 & ADC_SC2_ADACT_MASK);    // Conversion in progress
    while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); // Run until the conversion is complete
    return ADC0_RA;
}
```

ADC0_CFG1

6–5 ADIV	Clock Divide Select ADIV selects the divide ratio used by the ADC to generate the internal clock ADCK. 00 The divide ratio is 1 and the clock rate is input clock. 01 The divide ratio is 2 and the clock rate is (input clock)/2. 10 The divide ratio is 4 and the clock rate is (input clock)/4. 11 The divide ratio is 8 and the clock rate is (input clock)/8.
3–2 MODE	Conversion mode selection Selects the ADC resolution mode. 00 When DIFF=0:It is single-ended 8-bit conversion; when DIFF=1, it is differential 9-bit conversion with 2's complement output. 01 When DIFF=0:It is single-ended 12-bit conversion ; when DIFF=1, it is differential 13-bit conversion with 2's complement output. 10 When DIFF=0:It is single-ended 10-bit conversion ; when DIFF=1, it is differential 11-bit conversion with 2's complement output. 11 When DIFF=0:It is single-ended 16-bit conversion; when DIFF=1, it is differential 16-bit conversion with 2's complement output.
1–0 ADICLK	Input Clock Select Selects the input clock source to generate the internal clock, ADCK. Note that when the ADACK clock source is selected, it is not required to be active prior to conversion start. When it is selected and it is not active prior to a conversion start, when CFG2[ADACKEN]=0, the asynchronous clock is activated at the start of a conversion and deactivated when conversions are terminated. In this case, there is an associated clock startup delay each time the clock source is re-activated. 00 Bus clock 01 (Bus clock)/2 10 Alternate clock (ALTCLK) 11 Asynchronous clock (ADACK)

ADC0_SC2

2 DMAEN	DMA Enable 0 DMA is disabled. 1 DMA is enabled and will assert the ADC DMA request during an ADC conversion complete event noted when any of the SC1n[COCO] flags is asserted.

Main.c

```c
#include "pit.h"
#include "dma.h"

uint16_t  value[8];
extern char ready;

int main(void)
{
    int i, avg;
    avg = 0;
    ready = 0;

    dma_init();
    adc_init();
    pit_init();

    for(;;)
    {
        if(ready > ADC_READS)
        {
            for(i = 0; i < ADC_READS; i++)
                avg += value[i];
            avg /= ADC_READS;

            ready = 0;
        }
    }

    return 0;
}
```

SENAI

ARM Cortex M0+

Direct Memory Access (DMA) Controller

Exemplo Prático (UART)

Prof° Fernando Simplicio

UART0 + DMA

```
// Setup UART0 DMA
// Enable UART0 DMA functionality on receipt of a character
UART0_C5 |= UART0_C5_RDMAE_MASK;

 // Set Source Address (this is the UART0_D register
 DMA_SAR0 = (uint32_t)&UART0_D;


// Enables the DMA channel and select the DMA Channel Source
DMAMUX0_CHCFG0 |= DMAMUX_CHCFG_ENBL_MASK | DMAMUX_CHCFG_SOURCE(2);
//Enable channel 0, Request source = UART 0 Receive
```

DMAMUX.h - Bloco de notas

Arquivo Editar Formatar Exibir Ajuda

```
  #define DMAMUX_CHCFG_SOURCE_DISABLED      0                    // 0x00
  #define DMAMUX_CHCFG_SOURCE_UART0_RX      2                    // 0x02 UART0 RX -
0 and on DMA MUX 1 (when available)
  #define DMAMUX_CHCFG_SOURCE_UART0_TX      3                    // 0x03 UART0 TX
  #define DMAMUX_CHCFG_SOURCE_UART1_RX      4                    // 0x04 UART1 RX
  #define DMAMUX_CHCFG_SOURCE_UART1_TX      5                    // 0x05 UART1 TX
  #define DMAMUX_CHCFG_SOURCE_UART2_RX      6                    // 0x06 UART2 RX
  #define DMAMUX_CHCFG_SOURCE_UART2_TX      7                    // 0x07 UART2 TX
  #define DMAMUX_CHCFG_SOURCE_UART3_RX      8                    // 0x08 UART3 RX
  #define DMAMUX_CHCFG_SOURCE_UART3_TX      9                    // 0x09 UART3 TX
  #define DMAMUX_CHCFG_SOURCE_UART4_RX      10                   // 0x0a UART4 RX
```

UART + DMA

- **Desafio**

Receber bytes pela Serial e transferi-los pelo DMA a um buffer. O conteúdo deste buffer deverá ser apresentado em um LCD.