EEL7030 - Microprocessadores

Prof. Raimes Moraes

GpqCom – EEL

UFSC

Referências:

LPC23xx User Manual – NXP (http:// www.nxp.com/)

Material desenvolvido pelo Prof. Juliano Benfica - http://www.feng.pucrs.br/~jbenfica

The Insider's Guide To The NXP LPC2300/2400 Based Microcontrollers (http:// www.hitex.co.uk)

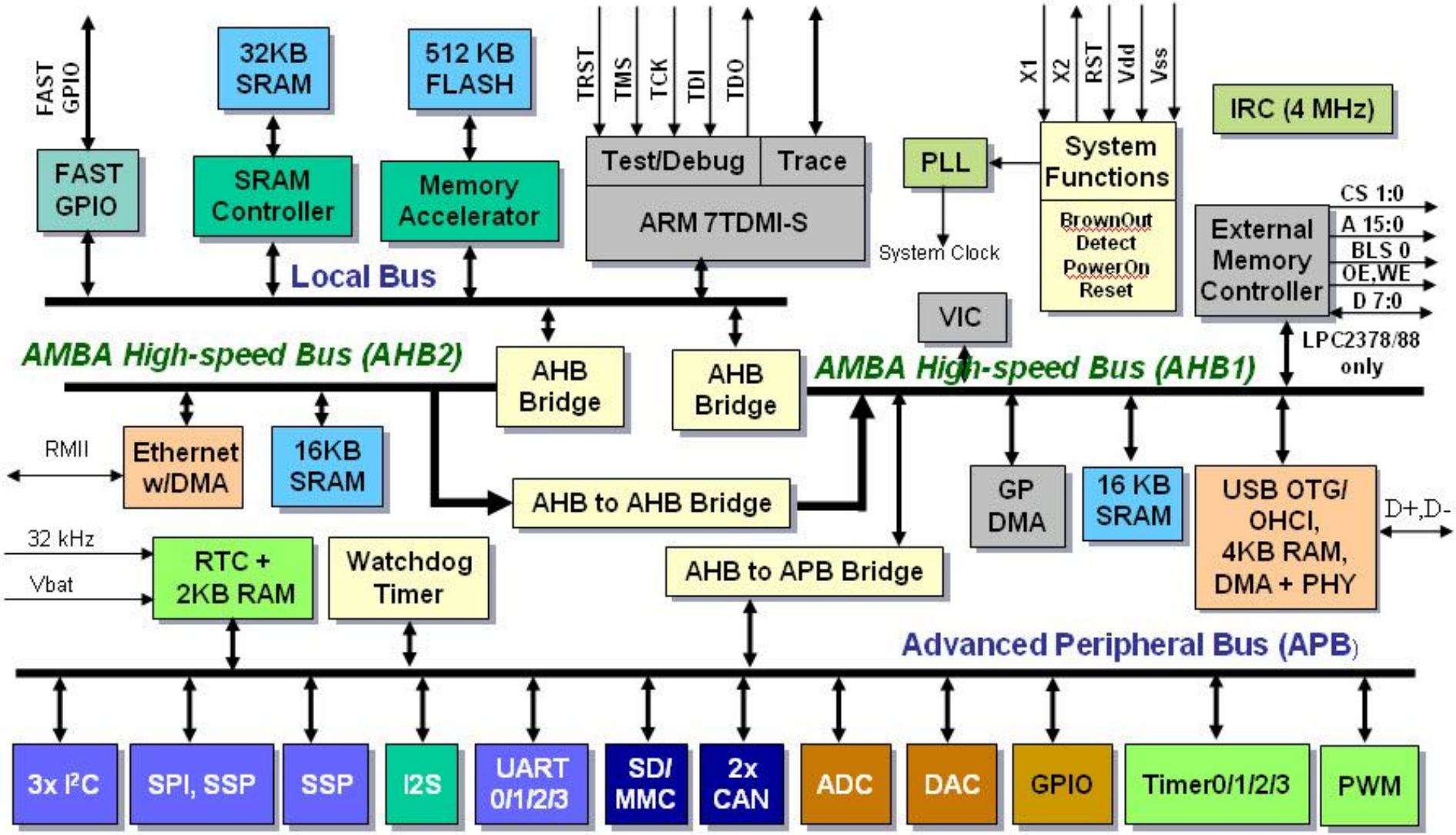
LPC2378

- **ARM7TDMI-S core - 72 MHz. Tensão única de alimentação: 3.3 V (3.0 V a 3.6 V).**
  - **T : Thumb (decodificador 16-bit);**
  - **D : Depuração JTAG;**
  - **M : Multiplicador rápido**
  - **I : depuração *In-circuit***
  - **S : Sintetizável (em HDL)**
- **Até 512 kB memória de programa flash**
- **32 kB SRAM de propósito geral (código ou dados)**
- **16 kB SRAM para Ethernet (código ou dados)**
- **8 kB SRAM para USB ou propósito geral**
- **2 kB SRAM de memória de dados alimentado por meio do Real-Time Clock (RTC) power.**

LPC2378

- **Interface para cartão de memória: *security digital card* (SD/MMC)**
- **104 pinos de I/O com configuração de resistores de pull-up/down**
- **10-bit ADC com entrada multiplexada entre 8 pinos.**
- **10-bit DAC**
- **4 Timers/counters e 1 PWM Timer**
- **Interfaces seriais: MAC Ethernet. USB 2.0, 4 UART, 2 SSP, SPI, 3 I2C, I2S e 2 CAN**
- **Real-Time Clock (RTC)**
- **WatchDog Timer**
- **GP DMA utilizado com SSP, I2S, SD/MMC e transferência memória-memória**

LPC23xx Block Diagram



LPC2378

Dois *Advanced High-performance Bus* (AHB) que possibilitam ativação simultânea do programa contido na flash, do DMA da Ethernet, do DMA ou da USB

DMA da Ethernet acessa os outros subsistemas através de *bus bridge*.

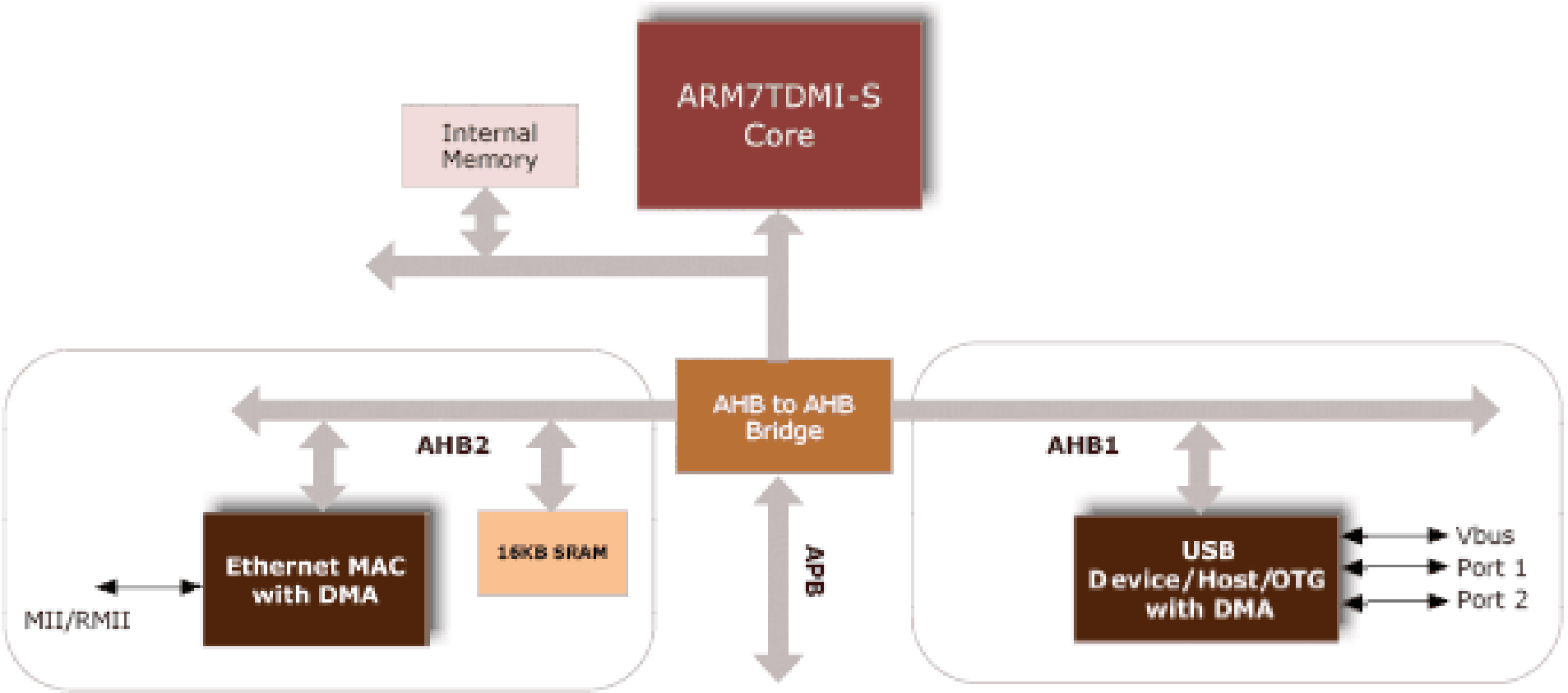
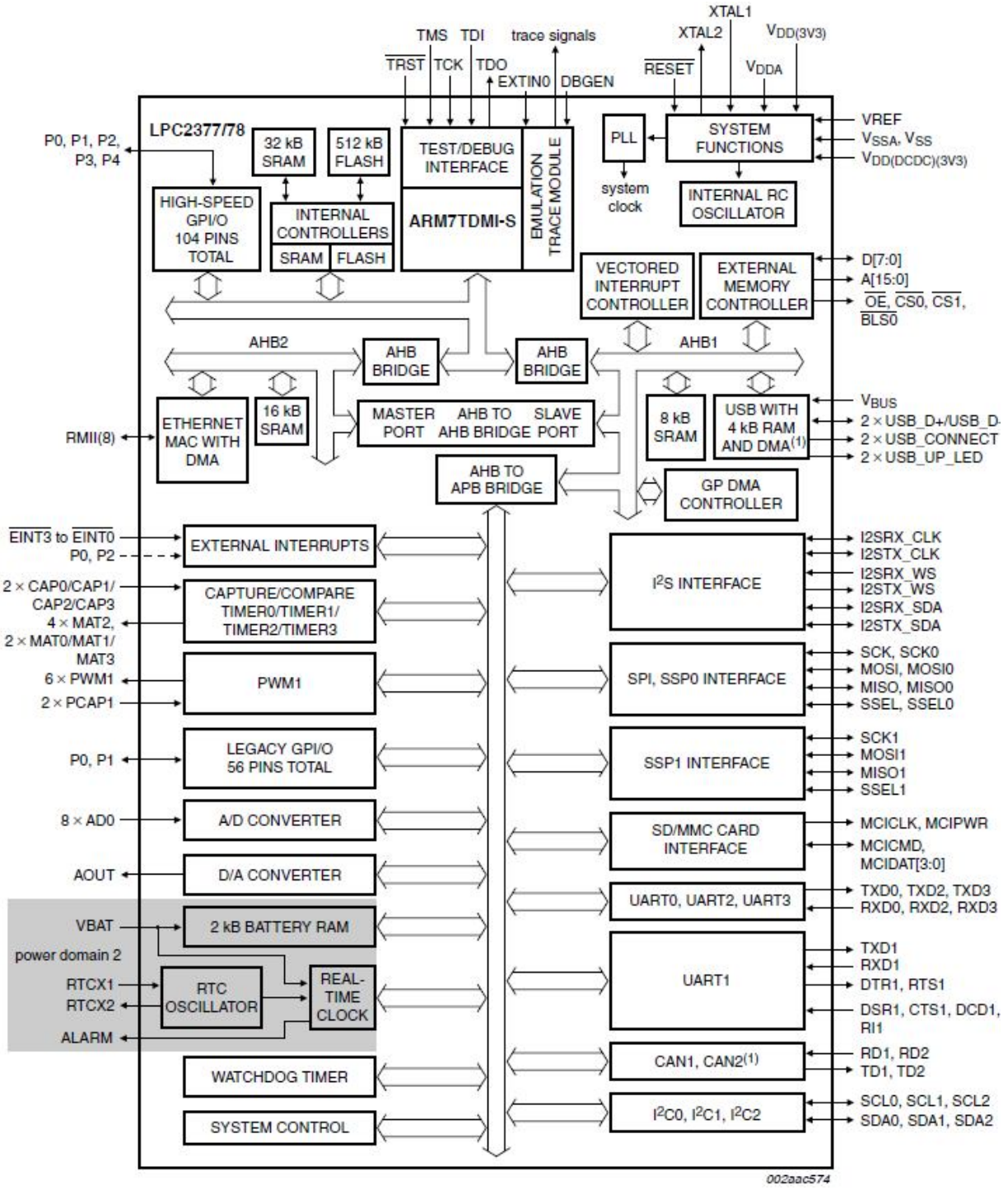
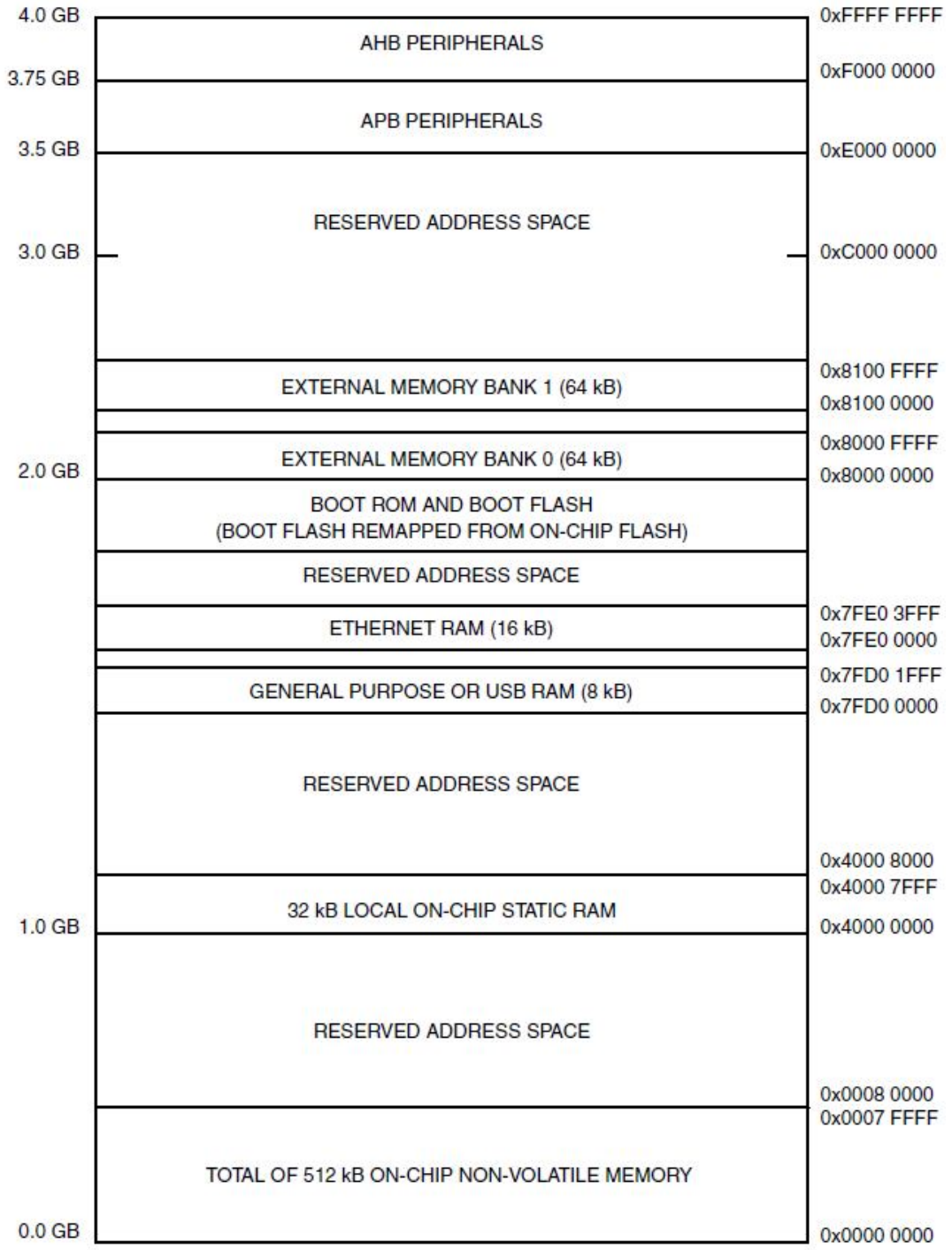


Diagrama de Blocos



Endereçamento da Memória



Energização dos Periféricos

- Para reduzir o consumo de energia, pode-se desativar periféricos sustando o *clock* destes através do registrador PCONP.

- Alguns módulos não podem ser desativados, por exemplo: Watchdog, GPIO, bloco de controle do sistema.

- Módulos analógicos têm consumo que independe do *clock*, podendo possuir alternativa específica para redução de consumo.

Table 56. Power Control for Peripherals register (PCONP - address 0xE01F C0C4) bit description

Bit	Symbol	Description	Reset value
0	-	Unused, always 0.	0
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	-	Unused, always 0.	1
6	PCPWM1	PWM1 power/clock control bit.	1
7	PCI2C0	The I^2C0 interface power/clock control bit.	1
8	PCSPI	The SPI interface power/clock control bit.	1
9	PCRTC	The RTC power/clock control bit.	1
10	PCSSP1	The SSP1 interface power/clock control bit.	1
11	PCEMC	External Memory Controller	1
12	PCAD	A/D converter (ADC) power/clock control bit. **Note:** Clear the PDN bit in the AD0CR (see Section 27–6.1) before clearing this bit, and set this bit before setting PDN.	0
13	PCAN1	CAN Controller 1 power/clock control bit.	0
14	PCAN2	CAN Controller 2 power/clock control bit.	0
18:15	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

PCONP (Continuação)

19	PCI2C1	The I^2C1 interface power/clock control bit.	1
20	-	Unused, always 0	0
21	PCSSP0	The SSP0 interface power/clock control bit.	1
22	PCTIM2	Timer 2 power/clock control bit.	0
23	PCTIM3	Timer 3 power/clock control bit.	0
24	PCUART2	UART 2 power/clock control bit.	0
25	PCUART3	UART 3 power/clock control bit.	0
26	PCI2C2	I^2C interface 2 power/clock control bit.	1
27	PCI2S	I^2S interface power/clock control bit.	0
28	PCSDC	SD card interface power/clock control bit.	0
29	PCGPDMA	GP DMA function power/clock control bit.	0
30	PCENET	Ethernet block power/clock control bit.	0
31	PCUSB	USB interface power/clock control bit.	0

Portas de Entrada e Saída

- **5 portas de entrada e saída de 32 pinos (104 pinos de I/O):**

P0.i, P1.i, P2.i, P3.i, P4.i; i: 0 a 31;

- **Alguns destes pinos têm função a ser definida (reservados)**

- **Necessário especificar direção e função dos pinos de cada porta. Por default, são todos configurados como pinos de entrada.**

- **Registradores de máscara permitem escrita em grupos de pinos, sem alterar os mascarados.**

- **P0 e P1 são acessíveis via registradores alocados *ARM local bus (Fast Port)* ou através de registradores alocados em endereços do *Advanced Peripheral Bus (*compatibilidade com versões anteriores*): Regular Port* (default).**

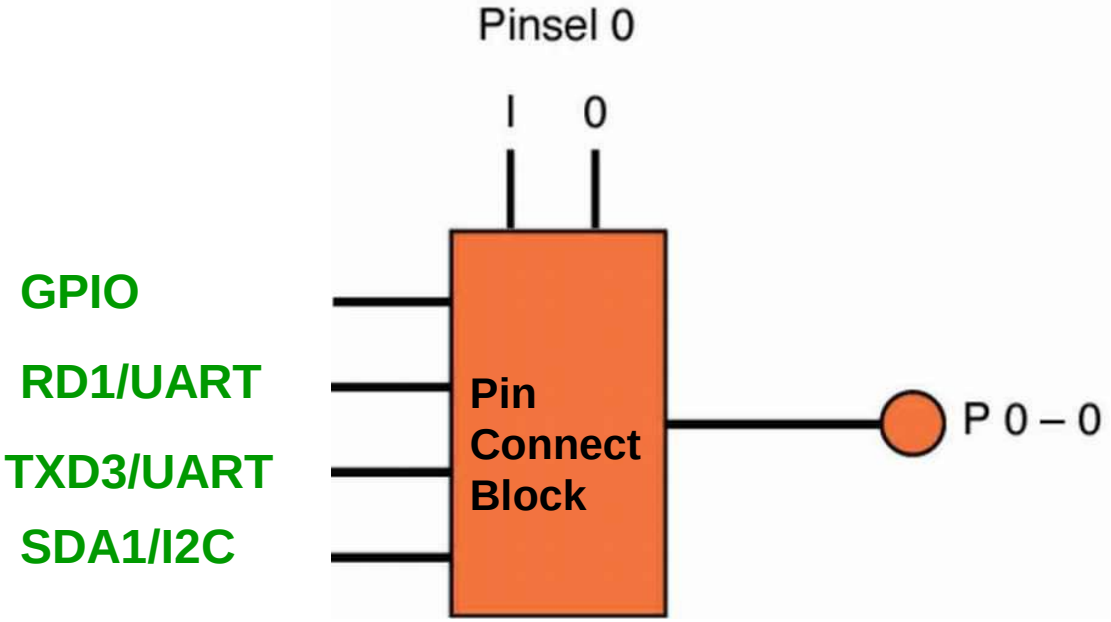
- **Modo *Fast* ou *Regular* especificado no bit 0 do registrador *System Controls and Status (SCS)***

Portas de Entrada e Saída

· **Configuração de portas de entrada e saída:**

- 10 registradores **PINSELj** (**j**: 0 a 10): Especificam função de cada pino. OBS: PINSEL5 não é utilizado na família LPC23XX.

- 10 registradores **PINMODEk** (**k**: 0 a 9): Configura resistor (pull-up, pull-down, open)



Exemplo de seleção de função do pino 0 da Porta P0

Table 106. Pin function select register 0 (PINSEL0 - address 0xE002 C000) bit description (LPC2377/78 and LPC2388)

PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.0	GPIO Port 0.0	RD1[1]	TXD3	SDA1	00
3:2	P0.1	GPIO Port 0.1	TD1[1]	RXD3	SCL1	00
5:4	P0.2	GPIO Port 0.2	TXD0	Reserved	Reserved	00
7:6	P0.3	GPIO Port 0.3	RXD0	Reserved	Reserved	00
9:8	P0.4	GPIO Port 0.4	I2SRX_CLK	RD2[1]	CAP2.0	00
11:10	P0.5	GPIO Port 0.5	I2SRX_WS	TD2[1]	CAP2.1	00
13:12	P0.6	GPIO Port 0.6	I2SRX_SDA	SSEL1	MAT2.0	00
15:14	P0.7	GPIO Port 0.7	I2STX_CLK	SCK1	MAT2.1	00
17:16	P0.8	GPIO Port 0.8	I2STX_WS	MISO1	MAT2.2	00
19:18	P0.9	GPIO Port 0.9	I2STX_SDA	MOSI1	MAT2.3	00
21:20	P0.10	GPIO Port 0.10	TXD2	SDA2	MAT3.0	00
23:22	P0.11	GPIO Port 0.11	RXD2	SCL2	MAT3.1	00
25:24	P0.12	GPIO Port 0.12	Reserved	MISO1	AD0.6	00
27:26	P0.13	GPIO Port 0.13	USB_UP_LED2[1]	MOSI1	AD0.7	00
29:28	P0.14	GPIO Port 0.14	$\overline{\text{USB\_HSTEN2}}$[2]	USB_CONNECT2[1]	SSEL1	00
31:30	P0.15	GPIO Port 0.15	TXD1	SCK0	SCK	00

[1] LPC2378/88 only. These bits are reserved on LPC2377.

[2] LPC2388 only. These bits are reserved on LPC2377/78.

Table 108. Pin function select register 1 (PINSEL1 - address 0xE002 C004) bit description (LPC2377/78 and LPC2388)

PINSEL1	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.16	GPIO Port 0.16	RXD1	SSEL0	SSEL	00
3:2	P0.17	GPIO Port 0.17	CTS1	MISO0	MISO	00
5:4	P0.18	GPIO Port 0.18	DCD1	MOSI0	MOSI	00
7:6	P0.19	GPIO Port 0.19	DSR1	MCICLK	SDA1	00
9:8	P0.20	GPIO Port 0.20	DTR1	MCICMD	SCL1	00
11:10	P0.21	GPIO Port 0.21	RI1	MCIPWR	RD1[1]	00
13:12	P0.22	GPIO Port 0.22	RTS1	MCIDAT0	TD1[1]	00
15:14	P0.23	GPIO Port 0.23	AD0.0	I2SRX_CLK	CAP3.0	00
17:16	P0.24	GPIO Port 0.24	AD0.1	I2SRX_WS	CAP3.1	00
19:18	P0.25	GPIO Port 0.25	AD0.2	I2SRX_SDA	TXD3	00
21:20	P0.26	GPIO Port 0.26	AD0.3	AOUT	RXD3	00
27:26	P0.29	GPIO Port 0.29	USB_D+1	Reserved	Reserved	00
29:28	P0.30	GPIO Port 0.30	USB_D–1	Reserved	Reserved	00
31:30	P0.31	GPIO Port 0.30	USB_D+2	Reserved	Reserved	00

[1] LPC2378/88 only. These bits are reserved on LPC2377.

[2] Pins P027] and P0[28] are open-drain for I^2C0 and GPIO functionality for I^2C-bus compliance.

PINMODEk (k: 0 a 9)

PINMODE0 to PINMODE9 Values	Function	Value after Reset
00	Pin has an on-chip pull-up resistor enabled.	00
01	Reserved. This value should not be used.	
10	Pin has neither pull-up nor pull-down resistor enabled.	
11	Pin has an on-chip pull-down resistor enabled.	

Table 121. Pin Mode select register 0 (PINMODE0 - address 0xE002 C040) bit description

PINMODE0	Symbol	Value	Description	Reset value
1:0	P0.00MODE		PORT0 pin 0 on-chip pull-up/down resistor control.	00
		00	P0.00 pin has a pull-up resistor enabled.	
		01	Reserved. This value should not be used.	
		10	P0.00 pin has neither pull-up nor pull-down.	
		11	P0.00 has a pull-down resistor enabled.	
...				
31:30	P0.15MODE		PORT0 pin 15 on-chip pull-up/down resistor control.	00

Exemplo de configuração da Porta P0

```
void init_P0(void)        // inicializada como porta fast; saída
{
// Os valores abaixo correspondem ao default na inicialização
PINSEL0=0; //16 LSB pinos da P0 => entrada/saída
PINSEL1=0; //16 MSB pinos da P0 => entrada/saída

// Os valores abaixo correspondem ao default na inicialização
PINMODE0=0; //16 LSB pinos da P0 => PULL-UP
PINMODE1=0; //...10 MSB pinos da P0 => PULL-UP - ATÉ O PINO 26 -- Demais são
                              // ...reservados (TABELA 122)

// Modificando valores default
SCS|=0x01;      //bit 1 de SCS em '1' => P0 e P1 no modo FAST
}
```

Portas de Entrada e Saída

- **Registradores para escrita/leitura de portas de entrada e saída:**

- 05 registradores para definir função do pino

FIOnDIR (n: 0 a 4): ‘1’ => saída; ´0´ => entrada;

- 05 registradores para leitura de dados FIOnPIN (n: 0 a 4).

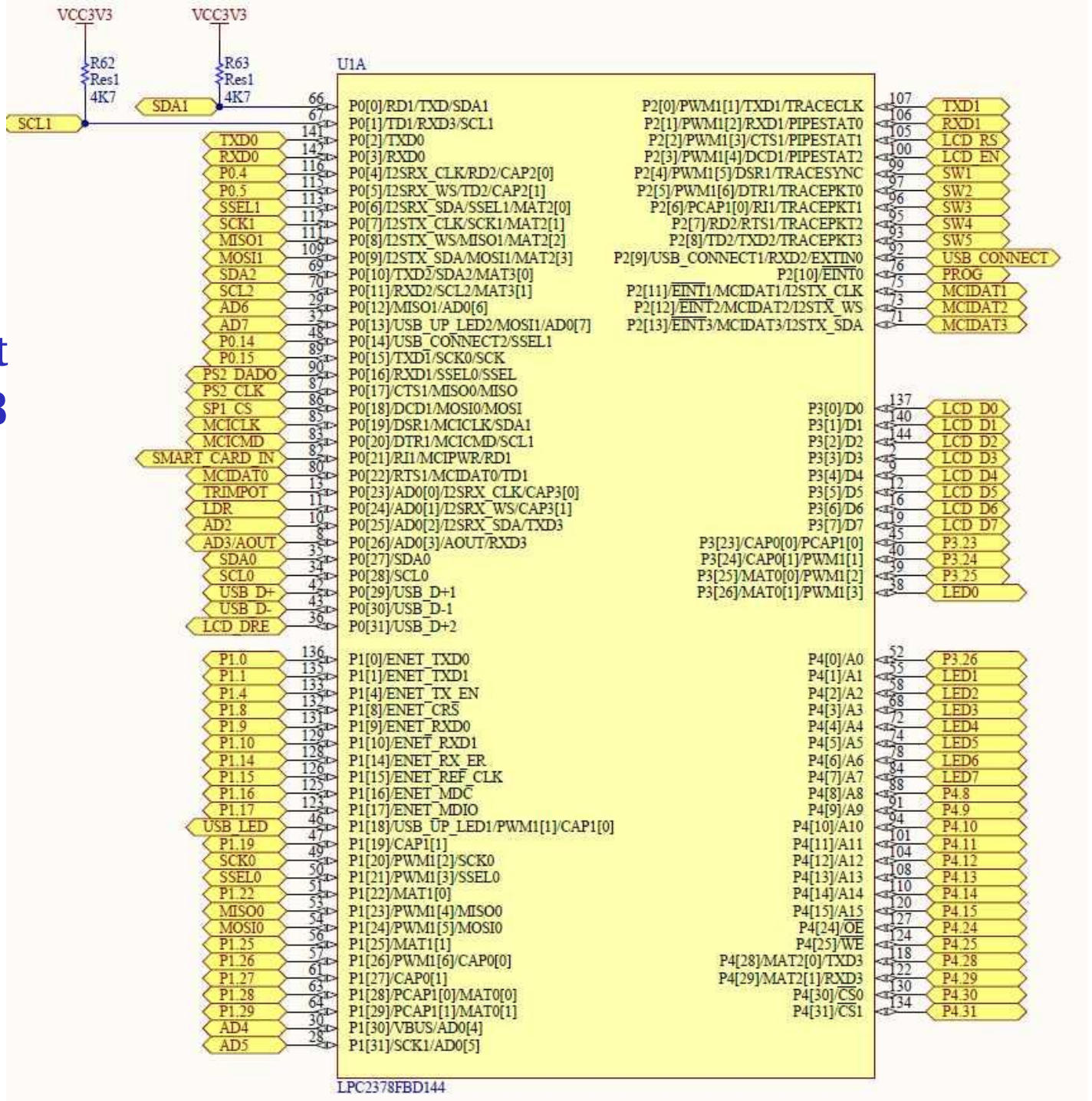
-15 registradores para escrita de dados FIOnPIN, FIOnSET, FIOnCLR (n: 0 a 4):

-- FIOnPIN: ‘0’ => escreve 0 - ´1´ => escreve 1;
-- FIOnCLR: ´1´ => escreve 0; - ‘0’ => não altera demais
-- FIOnSET; ´1´ => escreve 1; - ‘0’ => não altera demais

- 05 registradores FIOnMASK (n: 0 a 4): ‘0’ => habilita escrita; ´1´ => desabilita;

- **Estes registradodres de máscara permitem escrita em grupos de pinos, sem alterar os mascarados. Ex:FIO2MASK = 0xffffff00; FIO2PIN=0x32**

Conexões do kit com o LPC2378



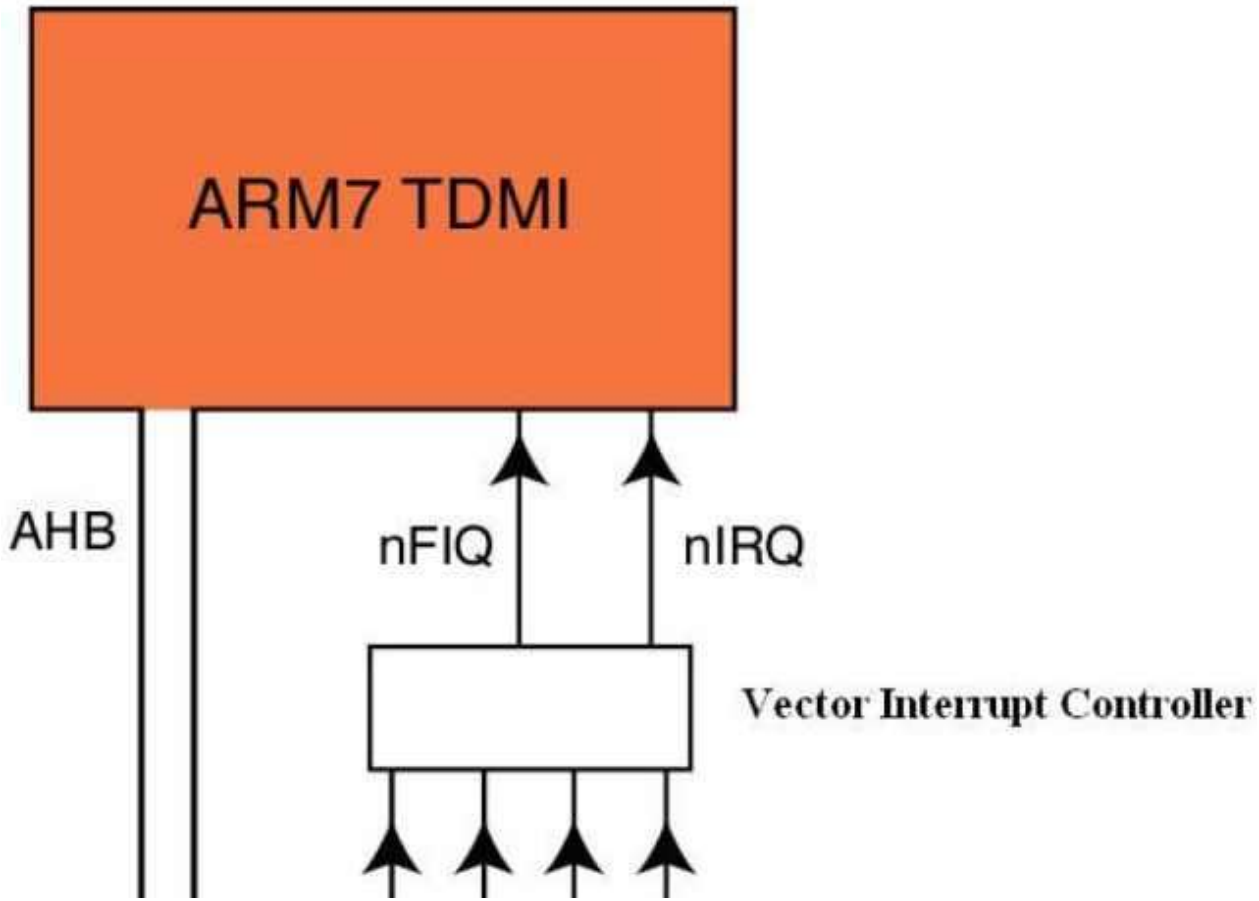
Exemplo para piscar led1 conectado a P4.1

```
#include <LPC23XX.H>

#define led1 (1<<1)                    //P4.1

void delay (int x)                     // rotina de atraso
{
        int i;      for(i=0;i<x;i++);
}

int main(void)
{

 FIO4DIR=0XFE;                         //Define P4.1 a P4.7 como saída
 FIO4SET=0xFE;          //Apaga leds na porta P4 (ativo em nível baixo)

 while (1) {

        FIO4CLR=led1;           delay(15000000);

        FIO4SET=led1;           delay(15000000);

                                }
return 0;
}
```

Interrupções

O ARM7 possui 2 linhas de solicitação de interrupção:

- FIQ (Fast Interrupt reQuest): Alta prioridade – End. Tratador: 0x0000001C
- vectored IRQ: Prioridade da interrupção programável – End. Tratador: **VICVectAddr**n



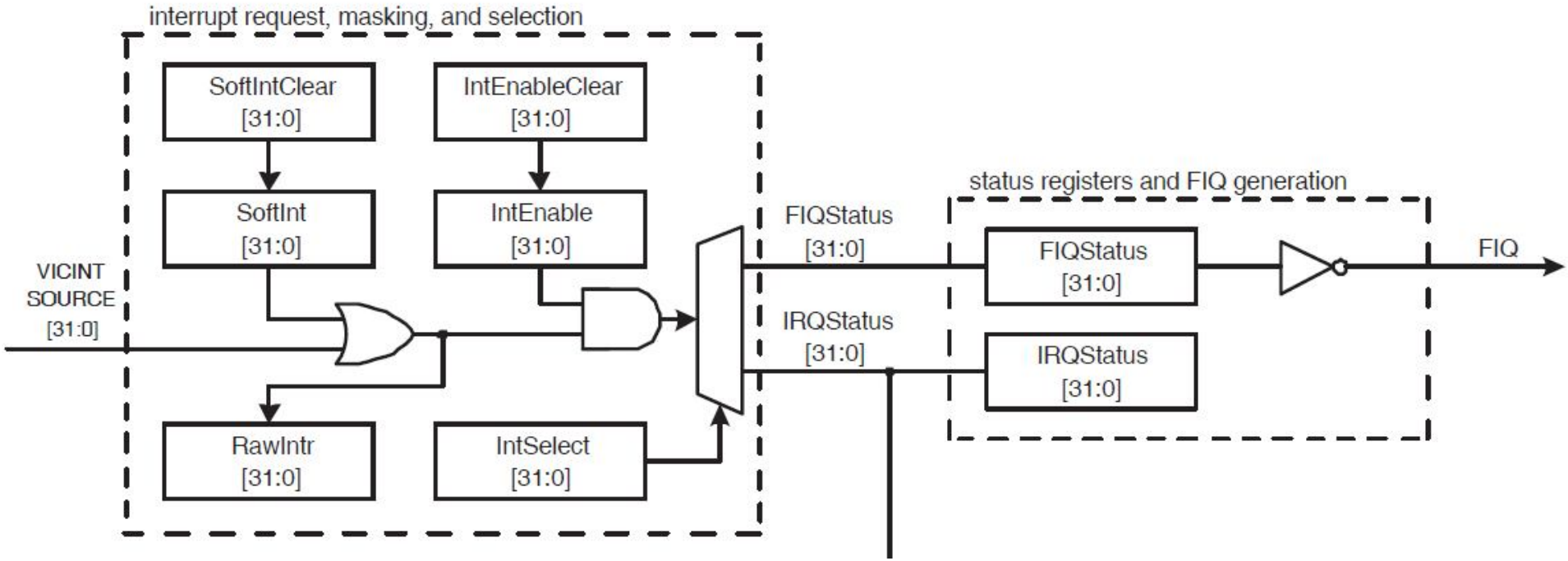
Para controlar acesso a estas linhas, há o *vectored interrupt controller*

Vectored Interrupt Controller (VIC)

O controlador de *Vectored Interrupts* aceita 32 solicitações de interrupção classificadas (de acordo com programação em **VICIntSelect**) como:

- FIQ (Fast Interrupt reQuest): Alta prioridade
- vectored IRQ (prioridade da interrupção programável: **VICVectPriority n (n: 0 a 31)**)

É possível solicitar a execução de qualquer canal de interrupção por software: **VICSoftInt**



Vectored Interrupt Controller (VIC)

VICIntSelect (*Interrupt Select Register*): Classifica as 32 fontes de interrupção como sendo FIQ ou IRQ, a partir da escrita de ‘1’ ou ‘0’, respectivamente, na posição
relativa à fonte de interrupção, conforme abaixo:

Table 87. Interrupt sources bit allocation table

Bit	31	30	29	28	27	26	25	24
Symbol	I2S	I2C2	UART3	UART2	TIMER3	TIMER2	GPDMA	SD/MMC
Bit	23	22	21	20	19	18	17	16
Symbol	CAN1&2	USB	Ethernet	BOD	I2C1	AD0	EINT3	EINT2
Bit	15	14	13	12	11	10	9	8
Symbol	EINT1	EINT0	RTC	PLL	SSP1	SPI/SSP0	I2C0	PWM1
Bit	7	6	5	4	3	2	1	0
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT

Interrupção por software é solicitada colocando-se nível lógico alto no registrador
VICSoftInt
na posição correspondente a interrupção desejada conforme a tabela acima.

Interrupção FIQ

Se mais de uma fonte de interrupção for classificada como FIQ, o registrador **VICIFIQStatus** deve ser lido para que se identifique a fonte de interrupção, tornando mais lento a execução do tratador.

Para que qualquer interrupção seja atendida, habilitar a mesma no registrador **VICIntEnable**

OBS: Verificar a necessidade de habilitar registradores do periférico para gerar a solicitação de interrupção

Escrever ‘1’ no *interrupt status flag* do periférico que solicitou a interrupção para resetar sua pendência.

Interrupção IRQ

Quando interrupção IRQ ocorre, o endereço do tratador a ser executado é aquele contido no registrador **VICVectAddrn (n: 0 a 31)** sendo que **n** corresponde ao periférico especificado na tabela abaixo:

Table 87. Interrupt sources bit allocation table

Bit	31	30	29	28	27	26	25	24
Symbol	I2S	I2C2	UART3	UART2	TIMER3	TIMER2	GPDMA	SD/MMC
Bit	23	22	21	20	19	18	17	16
Symbol	CAN1&2	USB	Ethernet	BOD	I2C1	AD0	EINT3	EINT2
Bit	15	14	13	12	11	10	9	8
Symbol	EINT1	EINT0	RTC	PLL	SSP1	SPI/SSP0	I2C0	PWM1
Bit	7	6	5	4	3	2	1	0
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT

OBS: Deve-se escrever neste registrador um valor qualquer ao final da execução do tratador de interrupção para sinalizar ao VIC que interrupção foi atendida. Escrever ‘1’ no interrupt status flag do periférico que solicitou a interrupção para resetar sua pendência.

VICVectPriorityn (n: 0 a 31) Especifica o nível de prioridade da interrupção **n.** O valor da prioridade vai de 0 a 15 (0: maior prioridade; 15:menor prioridade). Usar 4 LSB.

Registradores de Controle das Interrupções

VICIntSelect (*Interrupt Select Register*): Classifica as 32 fontes de interrupção como sendo FIQ (‘1’) ou IRQ (‘0’).

VICIntEnable (*Interrupt Enable Register*): Habilita interrupções escrevendo-se ‘1’ na posição correspondente ao slide anterior.

VICIntEnClr (*Interrupt Enable Clear Register*). Desabilita interrupções escrevendo-se ‘1’ na posição correspondente ao slide anterior.

VICVectPriority**n (n: 0 a 31)** Especifica o nível de prioridade da interrupção **n.** O valor da prioridade vai de 0 a 15 (0: maior prioridade; 15:menor prioridade). Usar 4 LSB.

VICVectAddr**n (n: 0 a 31)** Contém endereço do tratador da interrupção **n.**

VICSoftInt (*Software Interrupt Register*). Solicita, por software, a execução de algum dos 32 tratadores.

VICSoftIntClear (*Software Interrupt Clear Register*). Limpa pendência de interrupção solicitada através do **VICSoftInt.**

VICRawIntr (*Raw Interrupt Status Register*). Contém pendência de todo os tipos de interrupções (software, hardware, FIQ, IRQ, habilitadas, desabilitadas)

Interrupções Externas

EINT0, EINT1, EINT2, EINT3

Além destas fontes, todo pino da porta P0 e P2 gera interrupção pelo VIC slot da EINT3

Configuração:

IO0_INT_ EN_R e IO2_INT_ EN_R (GPIO Interrupt Enable for Rising edge register) :

‘1’ => interrupção por borda de subida no pino setado; IO0_INT_EN_R=0x08 (P0.3).

IO0_INT_ EN_F e IO2_INT_ EN_F (GPIO Interrupt Enable for Falling edge register)

‘1’ => interrupção por borda de descida no pino setado; IO0_INT_EN_F=0x08 (P0.3).

IO0_INT_ STAT_R e IO2_ STAT_ R (GPIO Interrupt Status for Rising register)

‘1’ => ocorreu interrupção por borda de subida no pino setado.

IO0_INT_ CLR e IO2_INT_ CLR (GPIO Interrupt Clear register)

‘1’ => limpa interrupção. Deve ser realizado no atendimento da interrupção.

Exemplo de alocação de tratador e configuração de interrupção externa

```
__irq void IO_IRQHandler(void)
{                                                   //#define SW1 (1<<4)
if((IO2_INT_STAT_F & SW1)==SW1)          //Checa se o pino P2.4 foi para zero
{                                                   // Adequado qdo  ha mais de uma int.
                                                    // ...habilitada                                   if(testa_int)
testa_int=0;                                                                                  else testa_int=1;
          IO2_INT_CLR = SW1;//limpa a flag de interrupção
}

                    VICVectAddr = 0;                    //sinaliza interrupção atendida
}


void configura_io_int(void)
{
IO2_INT_EN_F = SW1;          //configura para gerar interrupção na borda de descida

VICVectAddr17 = (unsigned long)IO_IRQHandler; //função de atendimento interrupção
VICVectPriority17 = 15;                                    //prioridade da interrupção = 15

VICIntEnable = (1 << 17);                                  //habilita interrupção
}
```

Programa que complementa led1 quando do acionamento de SW1

```
#include <LPC23XX.H>
#include <stdio.h>

#define led1 (1<<1) //P4.1
#define SW1 (1<<4)

char testa_int=0;

// Inserir funções do slide anterior


int main(void)
{
FIO4DIR=0XFE;                    //Define P4.1 a P4.7 como saída
FIO4SET=0xFE;          //Apaga leds na porta P4 (ativo em nível baixo)

 configura_io_int();

 while (1) {                 if (testa_int==0)  FIO4CLR=led1;
                             else       FIO4SET=led1;
            }
return 0;
}
```

Timers

- 4 Timer/Counters de 32 bits com *prescaler* programável de 32 bits.

- Todos são programados de forma idêntica.

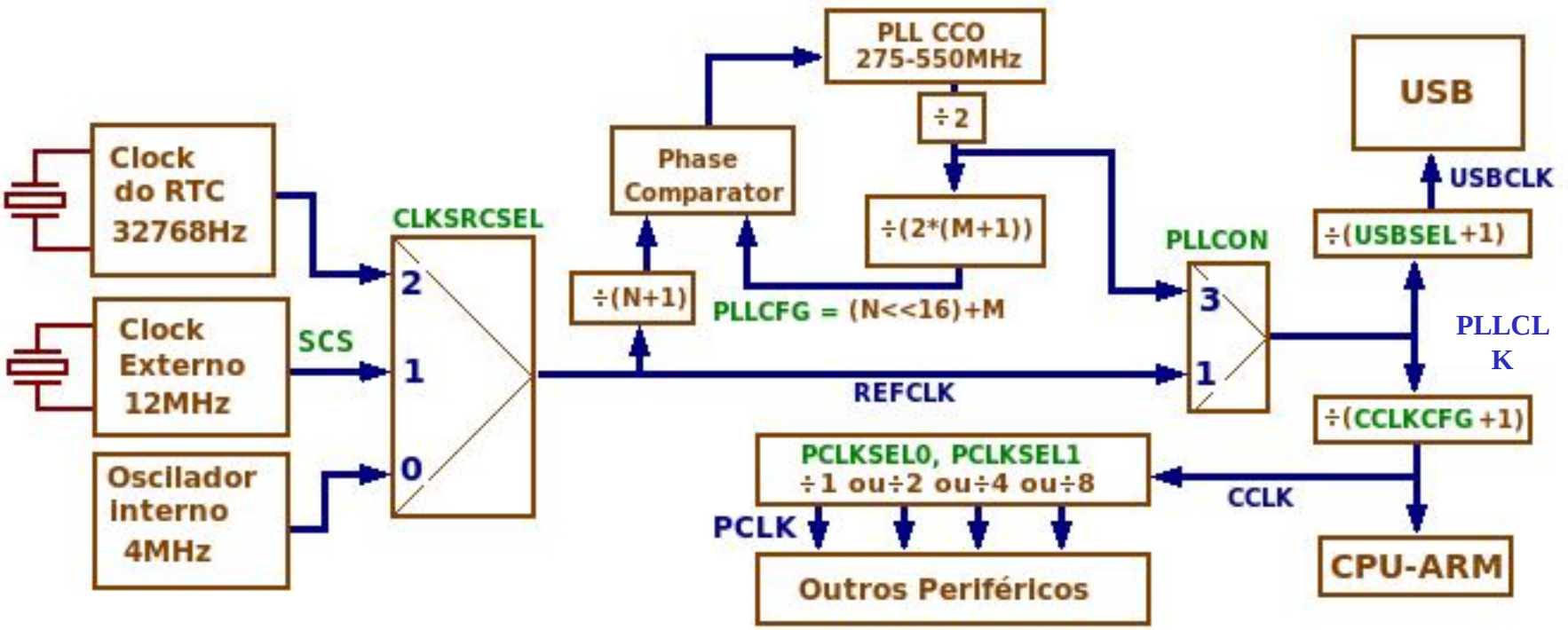
- *Default* clock: PCLK (*peripheral clock*)

- 3 diferentes modos de configuração:

  - Propósito Geral
  - Captura
  - *Match*

- Cada Timer/Counter pode receber sinais de 2 pinos de entrada e sinalizar eventos em até 4 pinos de saída

Clocks do LPC2378



Reproduzido de:
http://www.feng.pucrs.br/~stemmer/labproc/apostila2011/lpc-06.html

- 2 fontes internas de clock:
  - CCLK: CPU e periféricos AHB (USB, Ethernet, DMA)
  - PCLK: periféricos APB

Timers

Configuração Básica:

1. **Power**: No registrador PCONP (Tabela 4–56), setar bits PCTIM0/1/2/3.

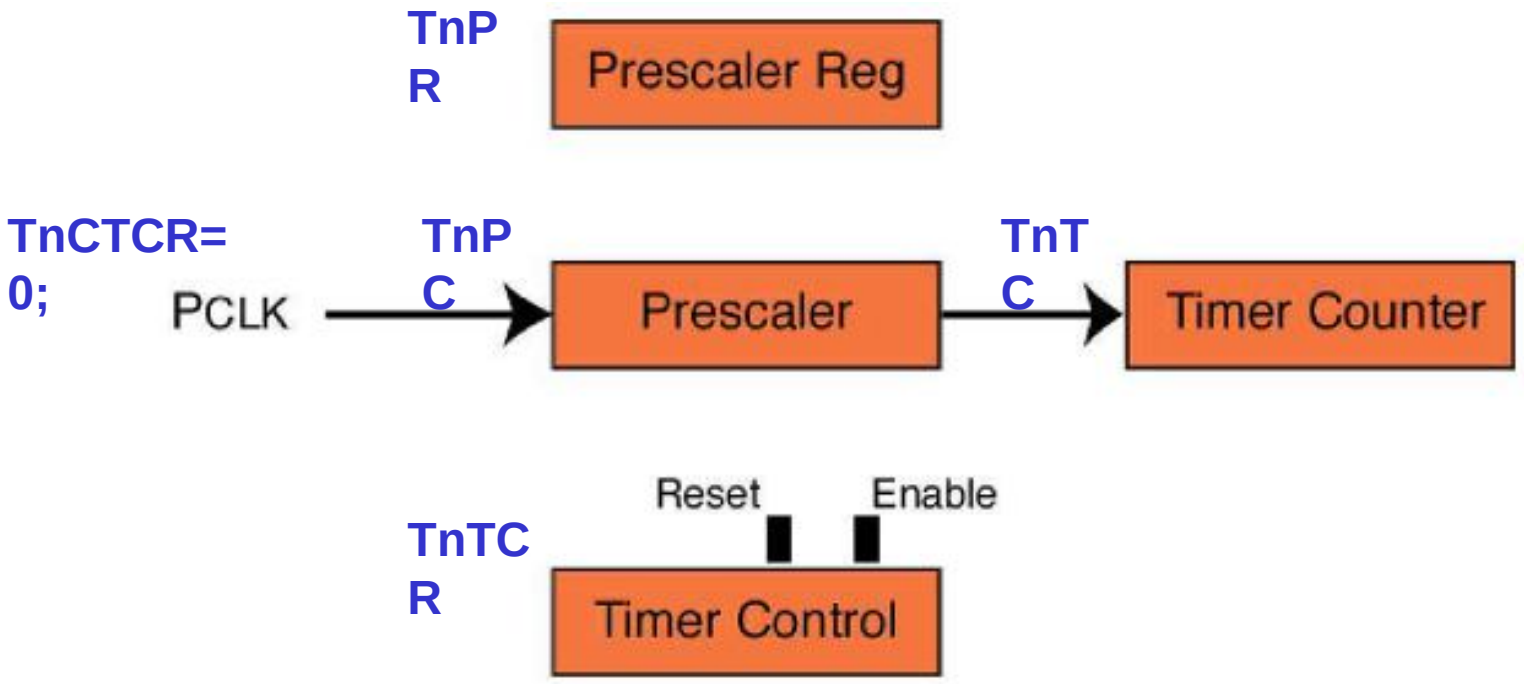
Após reset, Timer0/1 estão energizados (PCTIM0/1 = 1), Timer2/3 não (PCTIM2/3 = 0).

2. **Clock**: No registrador PCLK_SEL0 (Tabela 4–49), setar PCLK_TIMER0/1; em PCLK_SEL1 (Tabela 4–50), setar PCLK_TIMER2/3.

3. **Pinos**: Especificar o modo dos pinos (PINSELn and PINMODEn)

4. **Interrupções**: ver T0/1/2/3MCR (Table 23–476) e T0/1/2/3CCR (Tabela 23–477) para eventos de *match* e *capture*. Habilitá-las no registrador VICIntEnable (Tabela 6–76).

Timers – Tn (n: 0 a 3)



· **TnTCR (8 bits): pára (0x00) , habilita (0x01) e reseta (0x02) a contagem.**

·**TnCTCR(8 bits) : = 0x0 => modo *timer* incrementado em borda de subida de PCLK ≠ 0x0 => modo *Counter* e pino de origem *do clock*** (CAPn.0 ou CAPn.1) **.**

· **TnPC é um contador que é incrementado até o valor contido em TnPR qdo então:**

```
TnTC = TnTC+1;
TnPC = 0;
```

· **TnPC pode ser observado e controlado através do barramento da interface.**

TnTCR (8 bits)

Habilita e reseta a contagem

Table 474: Timer Control Register (TCR, TIMERn: TnTCR - addresses 0xE000 4004, 0xE000 8004, 0xE007 0004, 0xE007 4004) bit description

Bit	Symbol	Description	Reset Value
0	Counter Enable	When one, the Timer Counter and Prescale Counter are enabled for counting. When zero, the counters are disabled.	0
1	Counter Reset	When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of PCLK. The counters remain reset until TCR[1] is returned to zero.	0
7:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

TnCTCR (8 bits): Especifica modo de funcionamento do *Timer*/Counter

Table 475: Count Control Register (T[0/1/2/3]CTCR - addresses 0xE000 4070, 0xE000 8070, 0xE007 0070, 0xE007 4070) bit description

Bit	Symbol	Value	Description	Reset Value
1:0	Counter/ Timer Mode		This field selects which rising PCLK edges can increment Timer's Prescale Counter (PC), or clear PC and increment Timer Counter (TC). Timer Mode: the TC is incremented when the Prescale Counter matches the Prescale Register.	00
		00	Timer Mode: every rising PCLK edge	
		01	Counter Mode: TC is incremented on rising edges on the CAP input selected by bits 3:2.	
		10	Counter Mode: TC is incremented on falling edges on the CAP input selected by bits 3:2.	
		11	Counter Mode: TC is incremented on both edges on the CAP input selected by bits 3:2.	
3:2	Count Input Select		When bits 1:0 in this register are not 00, these bits select which CAP pin is sampled for clocking:	00
		00	CAPn.0 for TIMERn	
		01	CAPn.1 for TIMERn **Note:** If Counter mode is selected for a particular CAPn input in the TnCTCR, the 3 bits for that input in the Capture Control Register (TnCCR) must be programmed as 000. However, capture and/or interrupt can be selected for the other CAPn input in the same timer.	
		10	reserved	
		11	reserved	
7:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

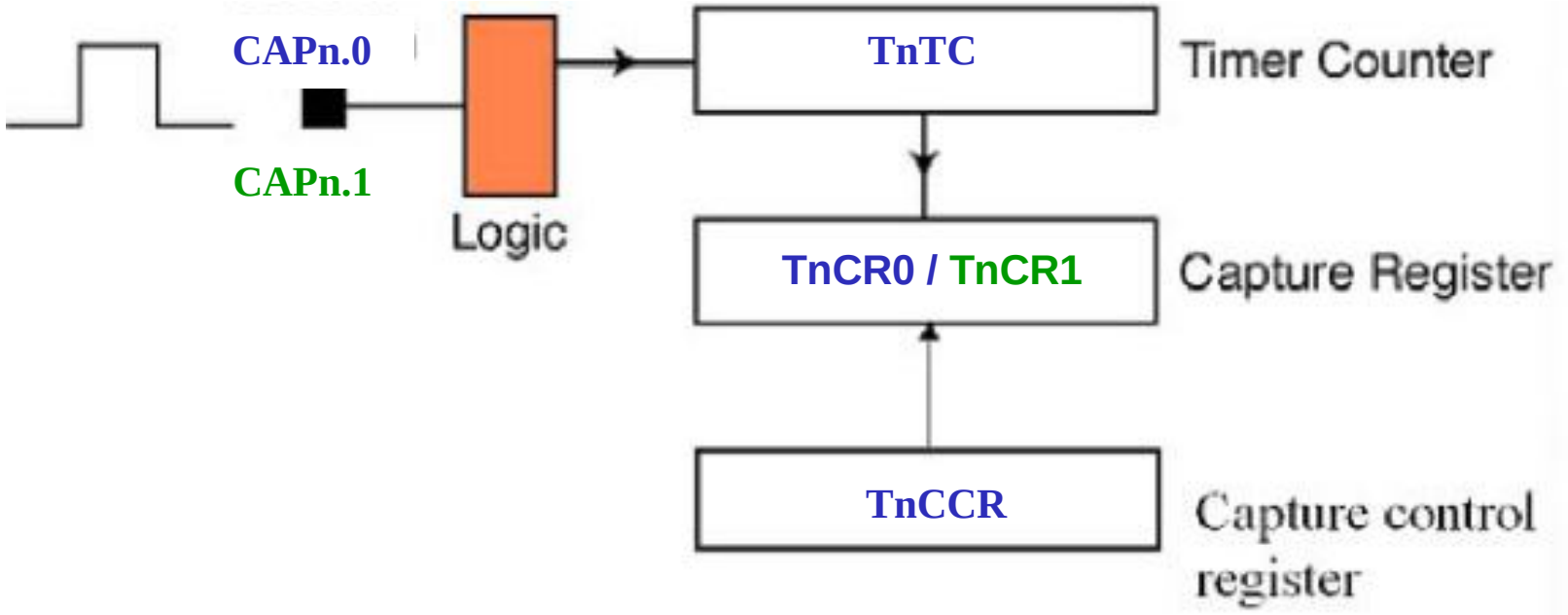
```
void init_timer0(void)  //funcao para inicializar o timer 0.
{
T0TCR = 0;                  // Pára contagem
T0PR = 12000000/1000 - 1; // Prescaler para incrementar T0TC a cada milisegundo
T0TCR = 2;                  // Reseta T0
T0TCR = 1;                  // Inicia contagem
}
```

```
    void delay_1ms(int t)                   //função  para espera de 1ms.
    {
    unsigned int tint;

    tint = T0TC + t;                                // tint = Valor futuro do T0TC

    while(tint!= T0TC);                     // espera ate que tint==T0TC
    }
```

Exemplo para piscar led1 conectado a P4.1

```
#include <LPC23XX.H>
#include <stdio.h>

#define led1 (1<<1)                    //P4.1

// Inserir funções do slide anterior

int main(void)
{

 FIO4DIR=0XFE;                         //Define P4.1 a P4.7 como saída
 FIO4SET=0xFE;              //Apaga leds na porta P4 (ativo em nível baixo)

init_timer0();

 while (1) {

            FIO4CLR=led1;              delay_1ms (1000);

            FIO4SET=led1;              delay_1ms (1000);

                                  }
return 0;
}
```

Modo Captura (n: 0 a 3)



·**TnCR0** (**TnCR1**) é carregado com o valor de TnTC quando borda de descida ou subida é aplicado ao pino **CAPn.0** (**CAPn.1**).

· TnCCR habilita a função de captura, especifica evento (borda, borda de subida, borda de descida) que realiza a captura e habilita interrupção associada a este modo.

TnCCR (16 bits)

Habilita a função de captura, especifica evento (borda, borda de subida, borda de descida) que realiza a captura e habilita interrupção associada a este modo.

Table 477: Capture Control Register (T[0/1/2/3]CCR - addresses 0xE000 4028, 0xE000 8020, 0xE007 0028, 0xE007 4028) bit description

Bit	Symbol	Value	Description	Reset Value
0	CAP0RE	1	Capture on CAPn.0 rising edge: a sequence of 0 then 1 on CAPn.0 will cause CR0 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
1	CAP0FE	1	Capture on CAPn.0 falling edge: a sequence of 1 then 0 on CAPn.0 will cause CR0 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
2	CAP0I	1	Interrupt on CAPn.0 event: a CR0 load due to a CAPn.0 event will generate an interrupt.	0
		0	This feature is disabled.	
3	CAP1RE	1	Capture on CAPn.1 rising edge: a sequence of 0 then 1 on CAPn.1 will cause CR1 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
4	CAP1FE	1	Capture on CAPn.1 falling edge: a sequence of 1 then 0 on CAPn.1 will cause CR1 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
5	CAP1I	1	Interrupt on CAPn.1 event: a CR1 load due to a CAPn.1 event will generate an interrupt.	0
		0	This feature is disabled.	
15:6	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

TnIR - Interrupt Register

Limpa interrupções (escrita) e identifica interrupções pendentes (leitura) devido aos modos CAPTURA E MATCH dos timers.

Table 473: Interrupt Register (T[0/1/2/3]IR - addresses 0xE000 4000, 0xE000 8000, 0xE007 0000, 0xE007 4000) bit description

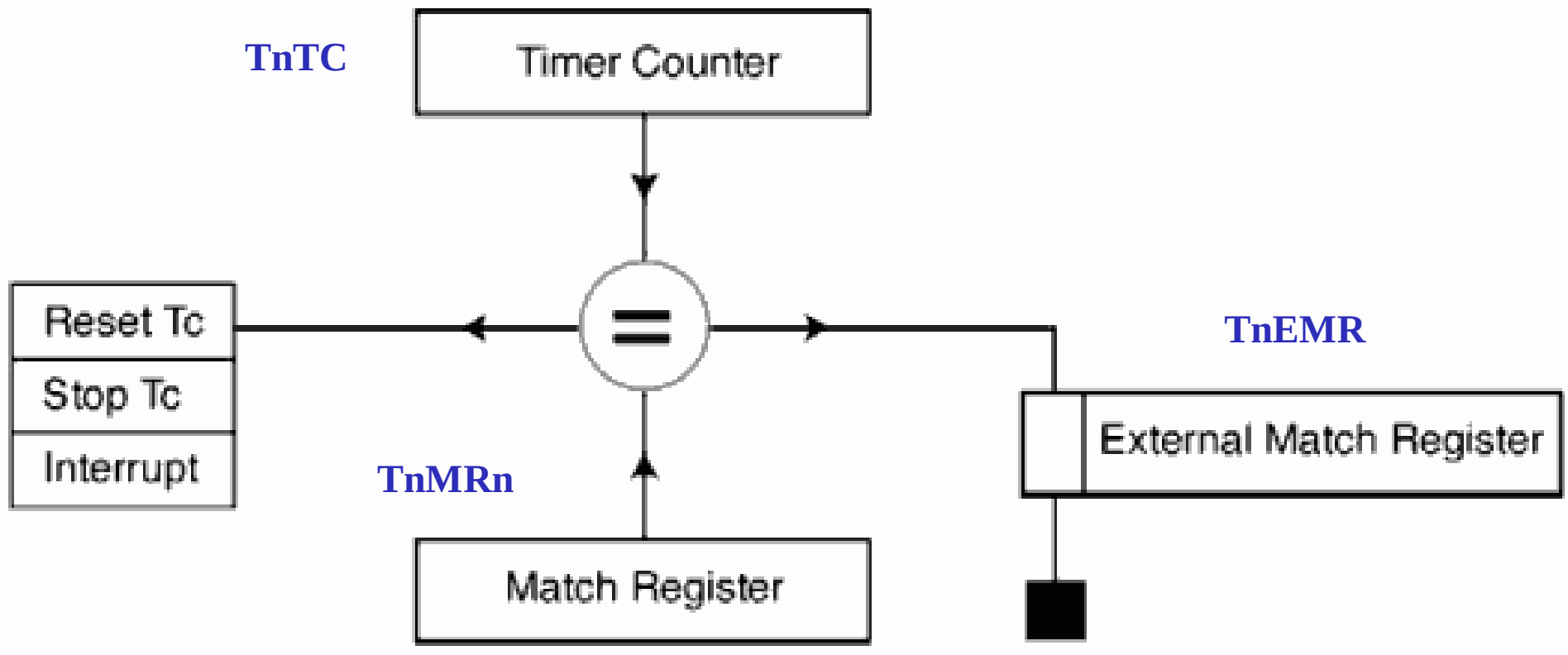
Bit	Symbol	Description	Reset Value
0	MR0 Interrupt	Interrupt flag for match channel 0.	0
1	MR1 Interrupt	Interrupt flag for match channel 1.	0
2	MR2 Interrupt	Interrupt flag for match channel 2.	0
3	MR3 Interrupt	Interrupt flag for match channel 3.	0
4	CR0 Interrupt	Interrupt flag for capture channel 0 event.	0
5	CR1 Interrupt	Interrupt flag for capture channel 1 event.	0
6	-	Reserved	0
7	-	Reserved	0

Quando uma interrupção ocorre, o tratador executado é aquele cujo endereço está contido no registrador VICVectAddr**n** (**n**: 4, 5, 26 ou 27)

Modo Match

Há 4 *match registeres* (T**n**MR**n**) para cada *timer* **(n: 0 a 3)**. Quando a contagem coincide com o valor carregado no MR**n**, evento pode ser solicitado (ver **TnMCR**):

- Atuar sobre o timer (reset, stop, interrupção ou combinação destes);
- Atuar sobre pino de saída (set, clear, complementa).



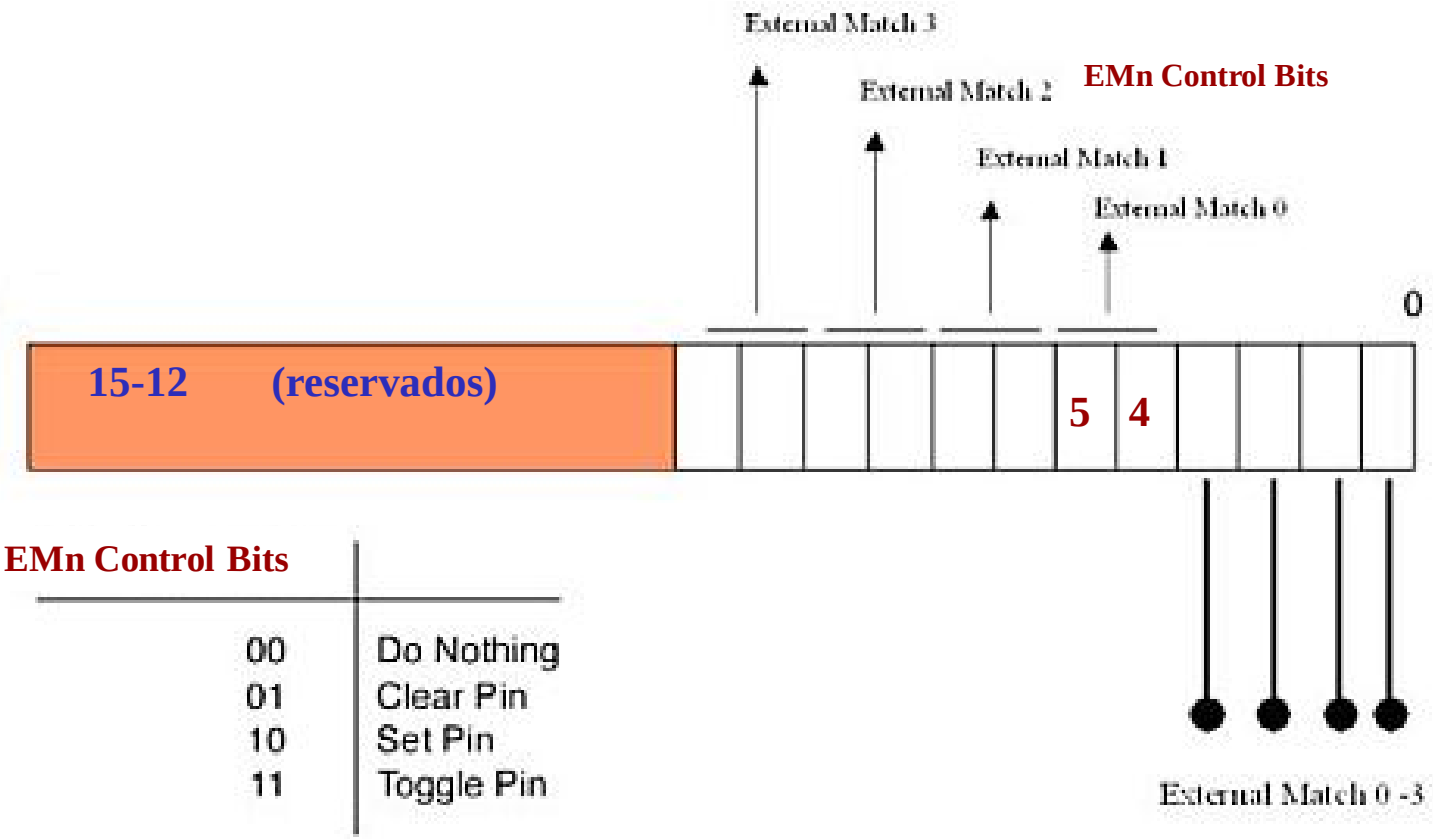
TnMCR (16 bits) – Match Control Register

Quando a contagem é atingida, especifica se interrupção é gerada, se o contador (TnTC) e *prescaler* (TnPC) são resetados/interrompidos.

Table 476: Match Control Register (T[0/1/2/3]MCR - addresses 0xE000 4014, 0xE000 8014, 0xE007 0014, 0xE007 4014) bit description

Bit	Symbol	Value	Description	Reset Value
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.	0
		0	This interrupt is disabled	
1	MR0R	1	Reset on MR0: the TC will be reset if MR0 matches it.	0
		0	Feature disabled.	
2	MR0S	1	Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.	0
		0	Feature disabled.	
			•••	
11	MR3S	1	Stop on MR3: the TC and PC will be stopped and TCR[0] will be set to 0 if MR3 matches the TC.	0
		0	Feature disabled.	
15:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

TnEMR External Match Register



EMn Control Bits

00	Do Nothing
01	Clear Pin
10	Set Pin
11	Toggle Pin

EM0. Bit que pode ser alterado quando T0TC = MR0, dependendo do valor em T0EMR[5:4] conforme tabela acima. Este valor pode ser atribuído ao pino externo MAT0.**m** (**m**: 0 a 1); para tal, alterar função nos registradores PINSELn.

EMn (**n**: 1 a 3). Mesmo que acima sendo que outros bits definem valor atribuído ao pino externo MAT**n**.**m** (**m**: 0 a 1 ou 0 a 3 – depende do timer).

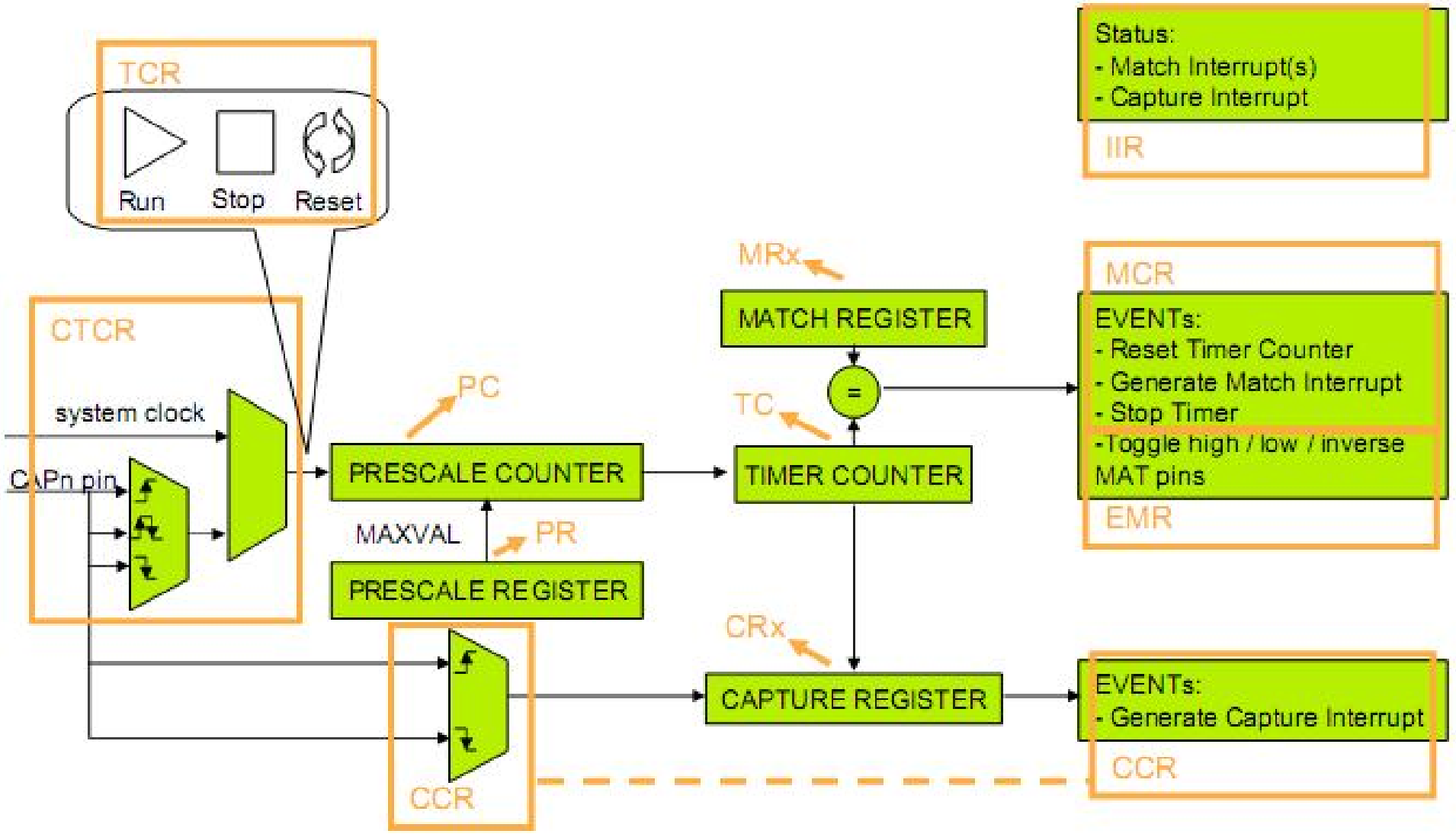
Exercícios

Sabendo que tem-se: leds vermelhos em:P3.26, P4.2, P4.5
leds verdes em: P4.1, P4.4, P4.7
leds amarelos em: P4.3, P4.6

chave SW1 conectado ao P2.4

1) Implementar um semáforo para carros que esteja geralmente aberto. Quando interrupção for gerada pela chave SW1, fechar o semáforo para os carros (vermelho e verde – atrasos de 2 segundos entre passagem) e abrir semáforo pedestre. Depois de 2 segundos, piscar o vermelho, fechá-lo e abrir o semáforo de pedestre.

1) Acrescentar ao exemplo acima um terceiro semáforo para automóveis. Reduzir tempo para fechamento do primeiro semáforo quando a chave SW1 for acionada.


TCR
Run
Stop
Reset
Status:
- Match Interrupt(s)
- Capture Interrupt
IIR
MRx
MCR
MATCH REGISTER
CTCR
system clock
CAPn pin
PC
PRESCALE COUNTER
TC
=
TIMER COUNTER
EVENTs:
- Reset Timer Counter
- Generate Match Interrupt
- Stop Timer
-Toggle high / low / inverse MAT pins
EMR
MAXVAL
PR
PRESCALE REGISTER
CRx
CAPTURE REGISTER
EVENTs:
- Generate Capture Interrupt
CCR
CCR

Funções para inicialização e escrita de dados no LCD

```
#include <lpc23xx.h>
#include <stdio.h>

#define LCD_RS (1<<2)      //RS em P2.2
#define LCD_EN (1<<3)      //RS em P2.3

// Inserir Funções init_timer0 e delay_1ms

void lcd_write(int valor)
{
FIO3CLR = 0xFF;                                  // 8 LSB = 0
FIO3SET = valor;                                 // seta bits
FIO2SET = LCD_EN;        delay_1ms(1);                  //EN = 1
FIO2CLR = LCD_EN;        delay_1ms(1);                  //EN = 0
}

void lcd_comando(int comando)     {        FIO2CLR = LCD_RS;          //RS=0
                                           lcd_write(comando);          }


void lcd_dado(int dado)           {        FIO2SET = LCD_RS;          //RS=1
                                           lcd_write(dado);       }
```

```
void lcd_init(void)       {
int i=0, init[]={0x38,0x0e,0x06,0x01,'$'}; // caracteres de inicialização do display
                                          // -- function set - entry mode - display on/off
          while (init[i]!='$')                    {          lcd_comando(init[i]);    i++;}

                    }

void lcd_string(char buffer[]) { int i=0;  //enviar string terminada com $
                                                                                    while
(buffer[i]!='$')  {          lcd_dado(buffer[i]);      i++;}
                                  }


int main(void)  // Programa para mostrar mensagem no LCD
{
char cadeia[]="Hello World$";              /* array contendo mensagem */

                    FIO3DIR=0xFF;
                    FIO2DIR=LCD_RS+LCD_EN;

                    init_timer0();
                    lcd_init();
                    lcd_string(cadeia);
                    while(1);
}
```

RTC (Real Time Clock)

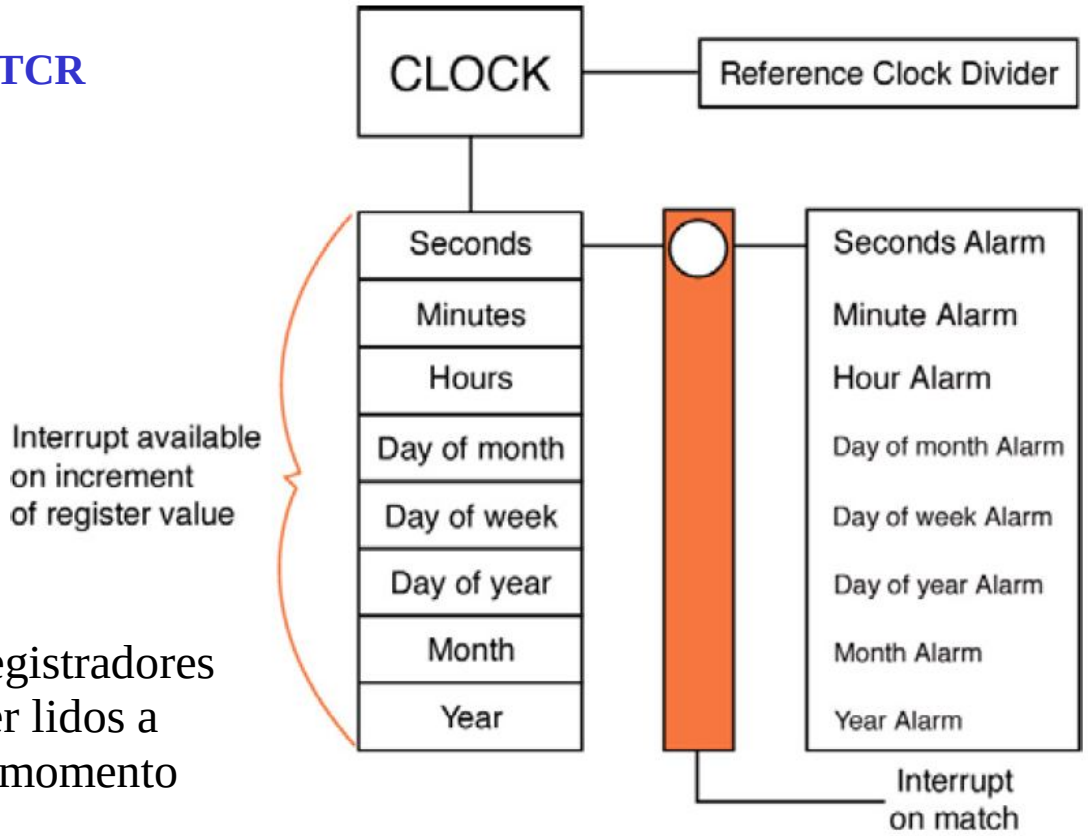
- **Conjunto de contadores para horário e data. Funciona para o sistema ligado e, opcionalmente, quando desligado.**

- **Possui 2 kB de SRAM sendo ambos energizados pelo pino VBAT conectado a bateria ou outra fonte.**

- **Clock para o RTC pode ser originado por cristal de 32.768 Hz ou por PCLK/prescaler**

- **Possui pino de saída de alarme para auxiliar saída de modos *Power-down***

RTC_CTCR



RTC_CTCR
Clock Tick Counter Register:

Conta 32.768 para incrementar contador de segundos

Estes 8 registradores Podem ser lidos a qualquer momento

RTC (Real Time Clock)

Configuração Básica

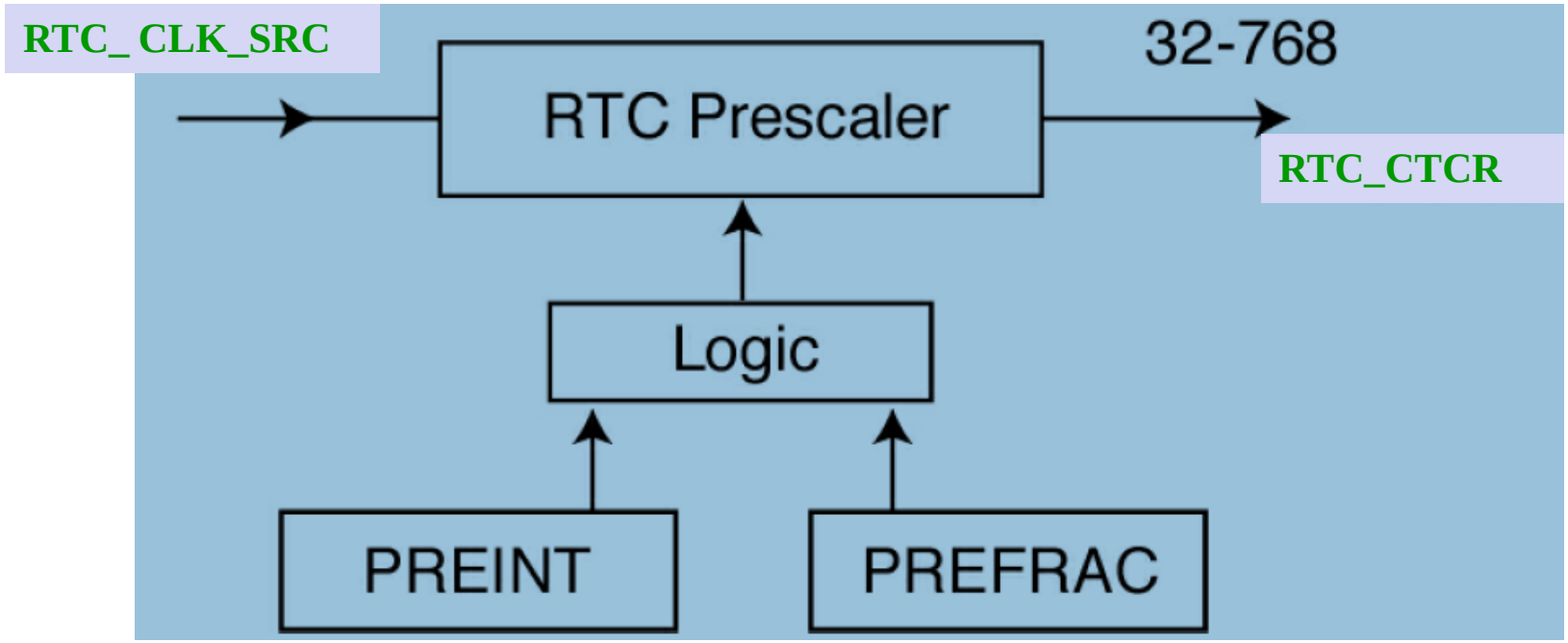
1. **Power**: No registrador PCONP (Tabela 4–56), setar bits PCRTC.

Após reset, RTC está habilitada.

2. **Clock**: Selecionar fonte de clock (Tabela 26–502). Para *peripheral clock (*PCLK), selecione PCLK_RTC no registrador PCLK_SEL0 (Tabela 4–49). O PCLK tem de ser dividido.

3. **Interrupts**: Habilitar no registrador VICIntEnable (Tabela 6–76).

Geração do Clock do RTC



Clock do RTC pode ser obtido de cristal externo (RTCX1 e RTCX2) ou a partir do PCLK

ClK_SRC	PREINT	PREFRAC
PCLK	int(PCLK/32768) -1	PCLK - ((RTC_PREINT + 1)*32768)
32.768	1	1

Seleção da Fonte de Clock para o RTC

Table 502. Clock Control Register (CCR - address 0xE002 4008) bit description

Bit	Symbol	Description	Reset value
0	CLKEN	Clock Enable. When this bit is a one the time counters are enabled. When it is a zero, they are disabled so that they may be initialized.	NA
1	CTCRST	CTC Reset. When one, the elements in the Clock Tick Counter are reset. The elements remain reset until CCR[1] is changed to zero.	NA
3:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
4	CLKSRC	If this bit is 0, the Clock Tick Counter takes its clock from the Prescaler, as on earlier devices in the NXP Embedded ARM family. See Section 4–7.4 for selection of the peripheral clock for the RTC. If this bit is 1, the CTC takes its clock from the 32 kHz oscillator that's connected to the RTCX1 and RTCX2 pins (see Section 26–10 "RTC external 32 kHz oscillator component selection" for hardware details).	NA
7:5	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Geração do Interrupções a partir do RTC: Incremento

VICVectAddr13: Deve ser carregado com endereço do tratador de interrupção.

VICIntEnable: Deve ser setado na posição correspondente para habilitar(0x2000).

Table 503. Counter Increment Interrupt Register (CIIR - address 0xE002 400C) bit description

Bit	Symbol	Description	Reset value
0	IMSEC	When 1, an increment of the Second value generates an interrupt.	NA
1	IMMIN	When 1, an increment of the Minute value generates an interrupt.	NA
2	IMHOUR	When 1, an increment of the Hour value generates an interrupt.	NA
3	IMDOM	When 1, an increment of the Day of Month value generates an interrupt.	NA
4	IMDOW	When 1, an increment of the Day of Week value generates an interrupt.	NA
5	IMDOY	When 1, an increment of the Day of Year value generates an interrupt.	NA
6	IMMON	When 1, an increment of the Month value generates an interrupt.	NA
7	IMYEAR	When 1, an increment of the Year value generates an interrupt.	NA

Geração do Interrupções a partir do RTC: Alarme

Table 505. Alarm Mask Register (AMR - address 0xE002 4010) bit description

Bit	Symbol	Description	Reset value
0	AMRSEC	When 1, the Second value is not compared for the alarm.	NA
1	AMRMIN	When 1, the Minutes value is not compared for the alarm.	NA
2	AMRHOUR	When 1, the Hour value is not compared for the alarm.	NA
3	AMRDOM	When 1, the Day of Month value is not compared for the alarm.	NA
4	AMRDOW	When 1, the Day of Week value is not compared for the alarm.	NA
5	AMRDOY	When 1, the Day of Year value is not compared for the alarm.	NA
6	AMRMON	When 1, the Month value is not compared for the alarm.	NA
7	AMRYEAR	When 1, the Year value is not compared for the alarm.	NA

Identificação da fonte de Interrupção do RTC no Tratador

Após seu uso para identificar a origem da interrupção (caso necessário), este registrador deve ser setado na posição correspondente à fonte de interrupção do RTC no tratador para resetar a solicitação de execução do tratador.

Table 500. Interrupt Location Register (ILR - address 0xE002 4000) bit description

Bit	Symbol	Description	Reset value
0	RTCCIF	When one, the Counter Increment Interrupt block generated an interrupt. Writing a one to this bit location clears the counter increment interrupt.	NC
1	RTCALF	When one, the alarm registers generated an interrupt. Writing a one to this bit location clears the alarm interrupt.	NC
2	RTSSF	When one, the Counter Increment Sub-Seconds interrupt is generated. The interrupt rate is determined by the CISS register.	NC
7:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Funcão para inicializar o RTC:

```
void RTCinit(void)
{
RTC_AMR = 0;
RTC_CIIR = 0;
RTC_CCR = 0;
RTC_PREINT = 1;                 //UTILIZANDO O CRYSTAL DE 32.768 HZ
RTC_PREFRAC = 1;                //UTILIZANDO O CRYSTAL DE 32.768 HZ
//RTC_PREINT = 365;             //UTILIZANDO PCLK = 12MHz
//RTC_PREFRAC = 6912;                   //UTILIZANDO PCLK = 12MHz
}
```

Funcão para iniciar o RTC:

```
void RTCstart(void)
{
RTC_CCR = 0x11;       //UTILIZANDO O CRYSTAL DE 32.768 HZ – 10001 binário
//RTC_CCR = 0x01;     //UTILIZANDO O CRYSTAL DE 12 MHZ
}
```

Funcão para interromper o RTC:

```
void RTCstop(void)
{
RTC_CCR= 0x10;        //UTILIZANDO O CRYSTAL DE 32.768KHZ – 10000 binário
//RTC_CCR = 0x00;     //UTILIZANDO O CRYSTAL DE 12MHZ
}
```

Funcão para configurar hora, minuto e segundo do RTC:

```
void RTCconfig(int hora, int minuto, int segundo)
{
RTCstop();                //Parar o RTC para ajuste
RTC_HOUR = hora;
RTC_MIN = minuto;
RTC_SEC = segundo;
RTCstart();               //Reinicia o RTC
}
```

```
#include <lpc23xx.h>              // Programa para mostrar hora no lcd
#include <stdio.h>
#include <string.h>

#define LCD_RS (1<<2)
#define LCD_EN (1<<3)

// Inserir funções do RTC (acima), LCD e timer (atraso de 1 ms)

int main(void)
{
char vetor[16];

FIO3DIR=0xFF;                          //  barramento de dados do LCD
FIO2DIR=LCD_RS+LCD_EN;                 //  pinos de controle do LCD

init_timer0();       lcd_init();    RTCinit();       RTCconfig(21,55,0);

 while (1) {    lcd_comando(0x80);
                sprintf(vetor, "Hora: %2d:%2d:%2d$",RTC_HOUR,RTC_MIN,RTC_SEC);
                lcd_string(vetor);                          }

return 0;
}
```

UARTs

- **4 UARTs integradas a gerador de baud rate**
- **FIFOs de 16 bytes para recepção e transmissão**
- **UART1 oferece controle de interface com modem**
- **UART3 possui suporte para comunicação com infravermelho IrDA**
- **Após reset, UART0 e UART1 estão energizadas (PCUART0/1 = 1) demais, não.**

UARTs

Configuração Básica

1. **Power**: No registrador PCONP (Tabela 4–56), setar bits PCUART0/2/3. Após reset, UART0 está habilitada (PCUART0 = 1), UART2/3 não (PCUART2/3 = 0).

2. **Peripheral clock**: No registrador PCLK_SEL0 (Tabela 4–49), setar PCLK_UART0; no registrador PCLK_SEL1 (Tabela 4–50), setar PCLK_UART2/3.

3. **Baud rate**: No registrador U0/2/3LCR (Tabela 16–356), setar bit DLAB =1 para acessar registradores DLL e DLM (Tabelas 16–351 e 351) e estabelecer baud rate. Para maior exatidão, setar a baud rate fracional (Tabela 16–362).

4. **UART FIFO**: Para habilitar FIFO, usar bit 0 do registrador U0FCR (Tabela 16–355).

5. **Pinos**: Selecionar modo UART nos registradores PINSELn e PINMODEn . **Pinos de recepção UART não devem ter *pull-down resistors* habilitados.**

6. **Interrupções**: Habilitadas em VICIntEnable . Para ter acesso a U0/2/3IER(Tabela 16–353), fazer bit DLAB =0 no registrador U0/2/3LCR (Tabela 16–356)

UARTn Line Control – 1

Especifica formato do dado a ser transmitido ou recebido.

Table 356. UARTn Line Control Register (U0LCR - address 0xE000 C00C, U2LCR - 0xE007 800C, U3LCR - 0xE007 C00C) bit description

Bit	Symbol	Value	Description	Reset Value
1:0	Word Length Select	00	5 bit character length	0
		01	6 bit character length	
		10	7 bit character length	
		11	8 bit character length	
2	Stop Bit Select	0	1 stop bit.	0
		1	2 stop bits (1.5 if UnLCR[1:0]=00).	
3	Parity Enable	0	Disable parity generation and checking.	0
		1	Enable parity generation and checking.	

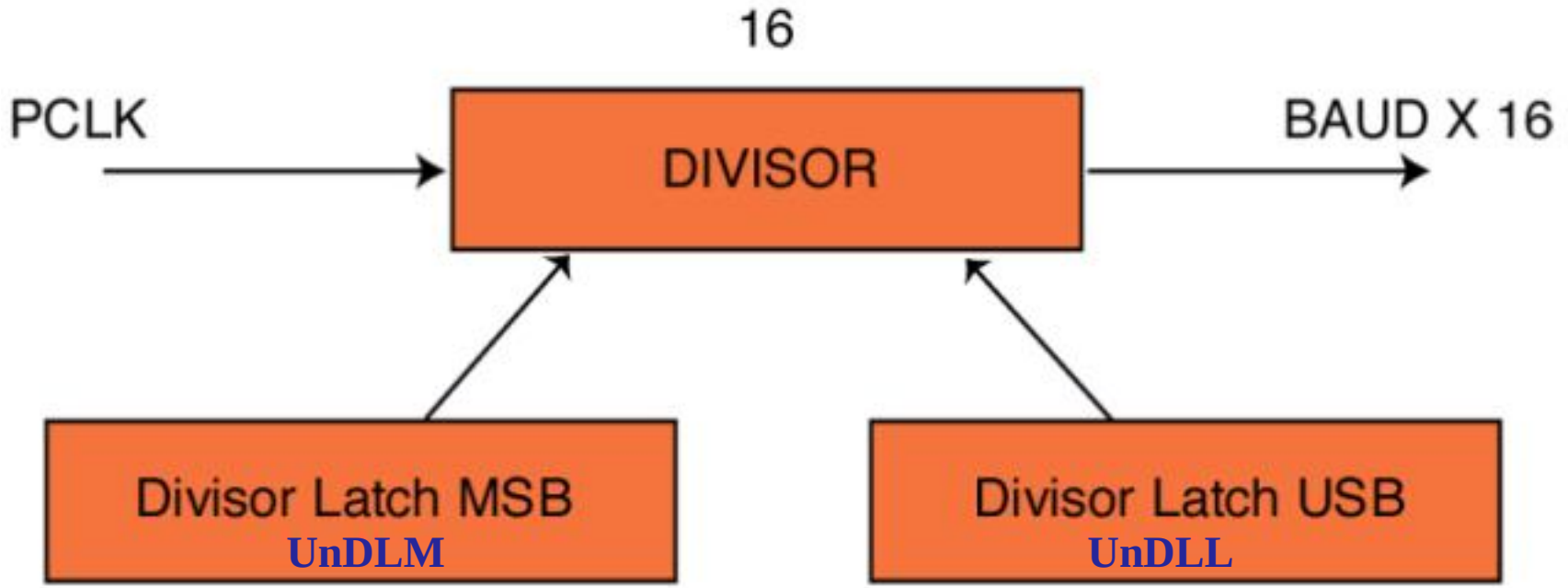
UARTn Line Control – 2

Especifica formato do dado a ser transmitido ou recebido.

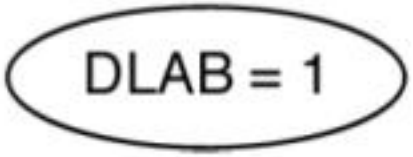
Table 356. UARTn Line Control Register (U0LCR - address 0xE000 C00C, U2LCR - 0xE007 800C, U3LCR - 0xE007 C00C) bit description

Bit	Symbol	Value	Description	Reset Value
5:4	Parity Select	00	Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd.	0
		01	Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even.	
		10	Forced "1" stick parity.	
		11	Forced "0" stick parity.	
6	Break Control	0	Disable break transmission.	0
		1	Enable break transmission. Output pin UART0 TXD is forced to logic 0 when UnLCR[6] is active high.	
7	Divisor Latch Access Bit (DLAB)	0	Disable access to Divisor Latches.	0
		1	Enable access to Divisor Latches.	

Baud Rate Generator



Divisor = PCLK/(16xBaud_rate)



UARTs toleram cerca de 5% de erro em relação à taxa desejada. Para reduzir o erro, usar **UnFDR** =>

UART0/2/3 Fractional Divider Register

Table 362. UARTn Fractional Divider Register (U0FDR - address 0xE000 C028, U2FDR - 0xE007 8028, U3FDR - 0xE007 C028) bit description

Bit	Function	Value	Description	Reset value
3:0	DIVADDVAL	0	Baud-rate generation pre-scaler divisor value. If this field is 0, fractional baud-rate generator will not impact the UARTn baudrate.	0
7:4	MULVAL	1	Baud-rate pre-scaler multiplier value. This field must be greater or equal 1 for UARTn to operate properly, regardless of whether the fractional baud-rate generator is used or not.	1
31:8	-	NA	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

$$UARTnbaudrate = PCLK/(16 \times (256 \times UnDLM + UnDLL) \times (1 + DivAddVal/MulVal))$$

UART0/2/3 FIFO Control Register

Table 355. UARTn FIFO Control Register (U0FCR - address 0xE000 C008, U2FCR - 0xE007 8008, U3FCR - 0xE007 C008, Write Only) bit description

Bit	Symbol	Value	Description	Res
0	FIFO Enable	0	UARTn FIFOs are disabled. Must not be used in the application.	0
		1	Active high enable for both UARTn Rx and TX FIFOs and UnFCR[7:1] access. This bit must be set for proper UARTn operation. Any transition on this bit will automatically clear the UARTn FIFOs.	
1	RX FIFO Reset	0	No impact on either of UARTn FIFOs.	0
		1	Writing a logic 1 to UnFCR[1] will clear all bytes in UARTn Rx FIFO and reset the pointer logic. This bit is self-clearing.	
2	TX FIFO Reset	0	No impact on either of UARTn FIFOs.	0
		1	Writing a logic 1 to UnFCR[2] will clear all bytes in UARTn TX FIFO and reset the pointer logic. This bit is self-clearing.	
5:3	-	0	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	RX Trigger Level		These two bits determine how many receiver UARTn FIFO characters must be written before an interrupt is activated.	0
		00	Trigger level 0 (1 character or 0x01)	
		01	Trigger level 1 (4 characters or 0x04)	
		10	Trigger level 2 (8 characters or 0x08)	
		11	Trigger level 3 (14 characters or 0x0E)	

```
void init_serial (void)
{
PINSEL0 |= 0x00000050;      /* Define função dos pinos da P0: TxD0 e RxD0 */
U0FCR = 7;                  /* Habilita e reseta FIFO de TX e RX */
U0FDR = 0x80;               /* Fractional divider não utilizada */
U0LCR = 0x83;               /* 8 bits, sem paridade, 1 Stop bit, DLAB = 1 */
U0DLL = 78;                 /* 9600 Baud Rate @ 12.0 MHZ PCLK */
U0DLM = 0;                  /* High divisor latch = 0 */
U0LCR = 0x03;               /* DLAB = 0 */
}
```

UnRBR - *UARTn Receiver Buffer Register*: Registrador correspondente ao menos recente dado recebido serialmente na FIFO .

```
int le_serial(void) //funcao para ler a serial 0.
{
while (!(U0LSR & 0x01));        //Verifica o BIT1 do U0LSR
                                //'0' => não recebeu dado; '1' =>  recebeu dado.
return (U0RBR);                 //Retorna o dado recebido
}
```

UnTHR - *UARTn Transmit Holding Register*: Registrador correspondente ao último dado da FIFO a ser transmistido pela serial.

```
void envia_serial (int dado) //funcao para enviar um dado para serial 0.
{
while (!(U0LSR & 0x20));                //Verifica o BIT6 do U0LSR
                                        //'0' => não enviou dado; '1' =>  enviou dado.
U0THR = dado;                           //envia novo dado
}
```

UARTn Line Status - 1

Table 357. UARTn Line Status Register (U0LSR - address 0xE000 C014, U2LSR - 0xE007 8014, U3LSR - 0xE007 C014, Read Only) bit description

Bit	Symbol	Value	Description	Reset Value
0	Receiver Data Ready (RDR)		UnLSR0 is set when the UnRBR holds an unread character and is cleared when the UARTn RBR FIFO is empty.	0
		0	UnRBR is empty.	
		1	UnRBR contains valid data.	
1	Overrun Error (OE)		The overrun error condition is set as soon as it occurs. An UnLSR read clears UnLSR1. UnLSR1 is set when UARTn RSR has a new character assembled and the UARTn RBR FIFO is full. In this case, the UARTn RBR FIFO will not be overwritten and the character in the UARTn RSR will be lost.	0
		0	Overrun error status is inactive.	
		1	Overrun error status is active.	
2	Parity Error (PE)		When the parity bit of a received character is in the wrong state, a parity error occurs. An UnLSR read clears UnLSR[2]. Time of parity error detection is dependent on UnFCR[0]. **Note:** A parity error is associated with the character at the top of the UARTn RBR FIFO.	0
		0	Parity error status is inactive.	
		1	Parity error status is active.	

UARTn Line Status - 2

Table 357. UARTn Line Status Register (U0LSR - address 0xE000 C014, U2LSR - 0xE007 8014, U3LSR - 0xE007 C014, Read Only) bit description

Bit	Symbol	Value	Description	Reset Value
3	Framing Error (FE)		When the stop bit of a received character is a logic 0, a framing error occurs. An UnLSR read clears UnLSR[3]. The time of the framing error detection is dependent on UnFCR0. Upon detection of a framing error, the Rx will attempt to resynchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error. **Note:** A framing error is associated with the character at the top of the UARTn RBR FIFO.	0
		0	Framing error status is inactive.	
		1	Framing error status is active.	
4	Break Interrupt (BI)		When RXDn is held in the spacing state (all 0's) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RXDn goes to marking state (all 1's). An UnLSR read clears this status bit. The time of break detection is dependent on UnFCR[0]. **Note:** The break interrupt is associated with the character at the top of the UARTn RBR FIFO.	0
		0	Break interrupt status is inactive.	
		1	Break interrupt status is active.	
5	Transmitter Holding Register Empty (THRE))		THRE is set immediately upon detection of an empty UARTn THR and is cleared on a UnTHR write.	1
		0	UnTHR contains valid data.	
		1	UnTHR is empty.	
6	Transmitter Empty (TEMT)		TEMT is set when both UnTHR and UnTSR are empty; TEMT is cleared when either the UnTSR or the UnTHR contain valid data.	1
		0	UnTHR and/or the UnTSR contains valid data.	
		1	UnTHR and the UnTSR are empty.	
7	Error in RX FIFO (RXFE)		UnLSR[7] is set when a character with a Rx error such as framing error, parity error or break interrupt, is loaded into the UnRBR. This bit is cleared when the UnLSR register is read and there are no subsequent errors in the UARTn FIFO.	0
		0	UnRBR contains no UARTn RX errors or UnFCR[0]=0.	
		1	UARTn RBR contains at least one UARTn RX error.	

Programa que recebe e transmite dados pela serial e os mostra no lcd

```
int main(void)
{
char dado_serial;

        FIO3DIR=0xFF;                   //  barramento de dados do LCD
        FIO2DIR=LCD_RS+LCD_EN;          //  pinos de controle do LCD

        init_timer0();
        lcd_init();
        init_serial();

        lcd_comando(0x80);

while (1) {
                        dado_serial = le_serial();
                        lcd_dado(dado_serial);
                        dado_serial+=1;
                        envia_serial(dado_serial);

                        }

return 0;
}
```

Programa que recebe dados pela serial e os mostra no lcd

```
#include <lpc23xx.h>
#include <stdio.h>

#define LCD_RS (1<<2)
#define LCD_EN (1<<3)

// Inserir funções da serial, timer e lcd aqui

int main(void)  {

int i=0,temp;

        init_serial();
        init_timer0();

        FIO3DIR|=0xFF;
        FIO2DIR|=LCD_RS+LCD_EN;

        lcd_init();

        while(1)                    {       i++;
                                    if (i==16) { i=0; lcd_comando(0x80);}
                                    temp = le_serial();
                                    lcd_dado(temp);
                        }
        return 0;
                }
```

Conversor Analógico Digital

- **8 canais de entrada multiplexados;**

- **Conversor de 10 bits por aproximações sucessivas em 410kSPS;**

- **Faixa de tensão de entrada: 0 a 3 V;**

- **Tempo de conversão $\geq$ 2.44 µs;**

- **Modo de conversão Burst para uma ou múltiplas entradas;**

- **Trigger de conversão por transição de clock em pino de entrada ou timer (timer Match)**

- **Registradores individuais para resultado de conversão de cada canal para reduzir tempo de execução do tratador de interrupção**

Conversor Analógico Digital

Configuração Básica:

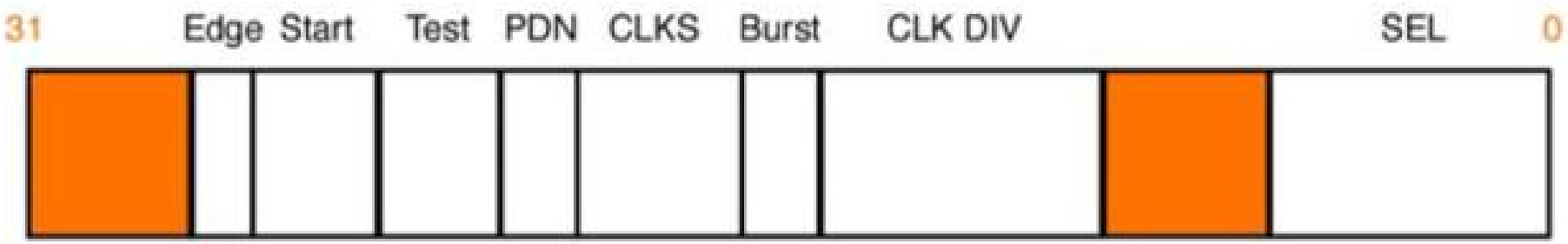
1. **Energização**: No registrador PCONP (Tabela 4–56), setar bits **PCADCn**
   Habilitar o ADC no registrador AD0CR (bit PDN) - Tabela 27–519.

2. **Clock**: No registrador PCLK_SEL0 (Tabela 4–49), setar PCLK_ADC. Necessário dividir o clock => bits CLKDIV (Tabela 27–519)

3. **Pinos**: Especificar função de entrada AD nos pinos da P0 e P1 (PINSELn e PINMODEn).

4. **Interrupções**: Habilitar de acordo com Tabela 27–522.

AD0CR: A/D Control Register



7:0 - **SEL**: Seleciona quais canais AD serão amostrados e convertidos Bit 0 em '1' seleciona AD0.0 e assim por diante

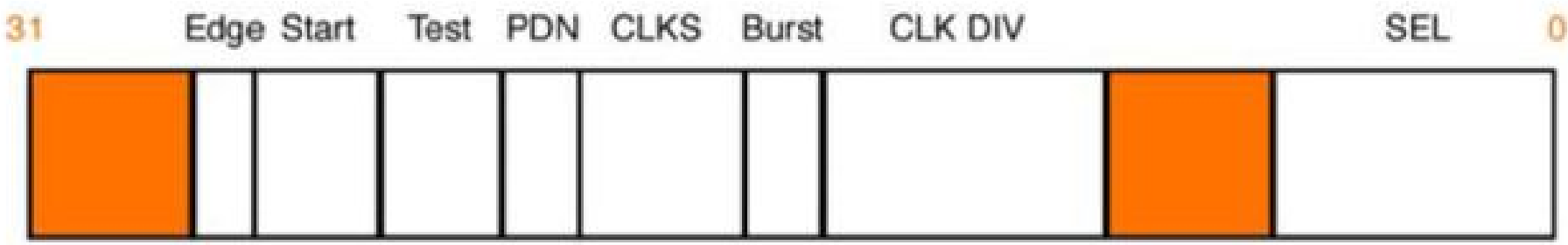
15:8 - **CLKDIV**: PCLK/(CLKDIV+1) < 4,5 MHz

16 - **BURST**: 0 => Conversão controlada por software.
1 => Conversão controlada por hardware.

19:17 - **CLKS**: Nro de clocks para conversão (Nclk) = 11 – CLKS; Resol.: Nclk -1.

21 - **PDN**: 0 => A/D em modo *power-down* (desligado).
1 => A/D energizado.

AD0CR: A/D Control Register

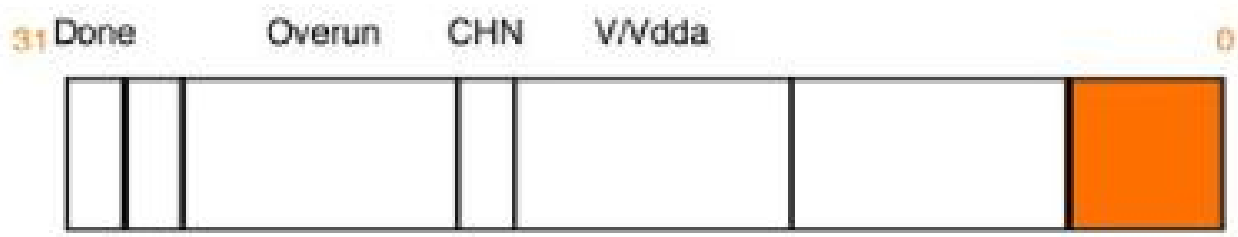


26:24 - **START**: Quando Burst=0, estes bits controlam o início da conversão

Start Bits	Conversion Trigger
0 0 0	Halted
0 0 1	Start Now
0 1 0	PO - 16
0 1 1	PO - 22
1 0 0	MAT 0-1
1 0 1	MAT 0-3
1 1 0	MATT 1-0
1 1 1	MAT 1-1

27 - **EDGE**: 0 => Trigger de Conversão por borda de subida.
1 => Trigger de Conversão por borda de descida

AD0GDR: A/D Global Data Register



31 - **DONE**: 0 => Conversão não concluída.
1 => Conversão concluída.

30 - **OVERUN**: 1 em Burst Mode => Conversão anterior foi perdida.

26:24 - **CHN**: Informa canal da última conversão realizada.

15:6 - **V/VREF**: Qdo DONE é 1, este campo contém razão entre tensão aplicada no campo de entrada sobre tensão de referência.

Funcão que configura A/D:

```
void init_ad (void)        //funcao para configurar o A/D canal 0.
{
AD0CR = 1;                          // ADC0 desligado
PCONP |= 0x00001000;                      // Energiza o ADC
PINSEL1 |= 0x4000;                  // Define pino como ADC0
AD0CR = 0x00200201;                       // Configura ADC0
}
```

Funcão para leitura do A/D:

```
int le_ad (void)                          //funcão para configurar o A/D canal 0.
{
int k=0;
AD0CR |= 0x01000000;                      //Inicia a conversao (Liga bit 24:START)
while(!( AD0GDR & 0x80000000));     // Espera terminar a conversao
k = ((AD0DR0 > > 6) & 0x3ff);             // desloca 6 bits
                                    // pois os primeiros 6 bits não são usados.
return k;
}
```

Programa que recebe dado do AD0 e os mostra no lcd

```
#include <lpc23xx.h>
#include <stdio.h>

#define LCD_RS (1<<2)
#define LCD_EN (1<<3)

// Inserir funções do  AD, timer e lcd aqui

int main(void)
{
char vetor[16];
int i
float temp;

init_timer0();

FIO3DIR|=0xFF;
FIO2DIR|=LCD_RS+LCD_EN;

lcd_init();
init_ad();

 while (1) {                    i=le_ad();
                                temp=((i*3.3)/1023);
                                lcd_comando(0x80);
                                sprintf(vetor,"Valor AD:%1.3f$",temp);
                                lcd_string(vetor);
                        }
 return 0;
}
```

Interfaces I2C: *Inter-Integrated Circuit*

Características

• Três I2C padrão que podem ser configuradas como Mestre, Escravo ou Mestre/Escravo.

• Arbitragem entre mestres transmitindo simultaneamente sem perda de dados no barramento.

• Clock programável para ajustar taxa de transferência dof I2C. Até 400 kibi/s

• Transferência bidirecional de dados entre mestres e escravos.

Clique para editar o estilo do subtítulo mestre

• Sincronização do clock serial para permitir dispositivos transmitindo a diferentes taxas se comunicarem.

• Sincronização do clock serial pode ser utilizado como mecanismo de *handshake* para interromper e retomar transferência serial.

• O barramento I2C pode ser utilizado para teste e diagnóstico.

Configuração Básica:

1. **Energização**: No registrador PCONP (Tabela 4–56), setar bits **PCI2Cn**
   Após reset, todas estão habilitadas.

2. **Clock**: No registrador PCLK_SEL0 (Tabela 4–49), setar PCLK_I2C0; em PCLK_SEL1 selecionar PCLK_I2C1/2 (Seção 4–7.4)

3. **Pinos**: Especificar função I2C nos pinos da P0 e P1 (PINSEL0 a 4 e PINMODE0 a 4 – Seção 9.5). OBS: pinos SDA0 e SCL0 são *open-drain* (Section 9–5.13).

Clique para editar o estilo do subtítulo mestre

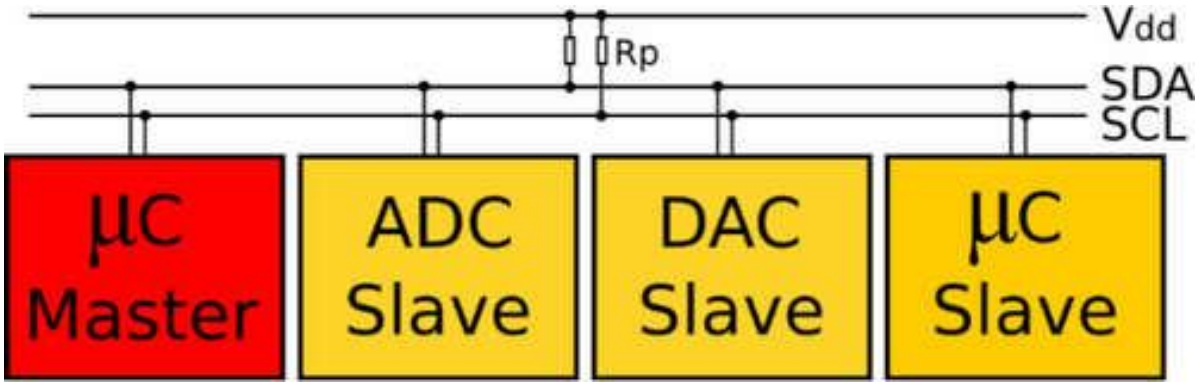
4. **Interrupções**: Habilitar no registrador VICIntEnable (Tabela 6–76).

5. I**nitialização**: Seção 21–9.12.1 e Seção 21–10.1.

Transmissão Serial

I2C: *Inter-Integrated Circuit*

Mestre Único



SDA - **S**erial **DA**ta line
SCL - **S**erial **CL**ock line

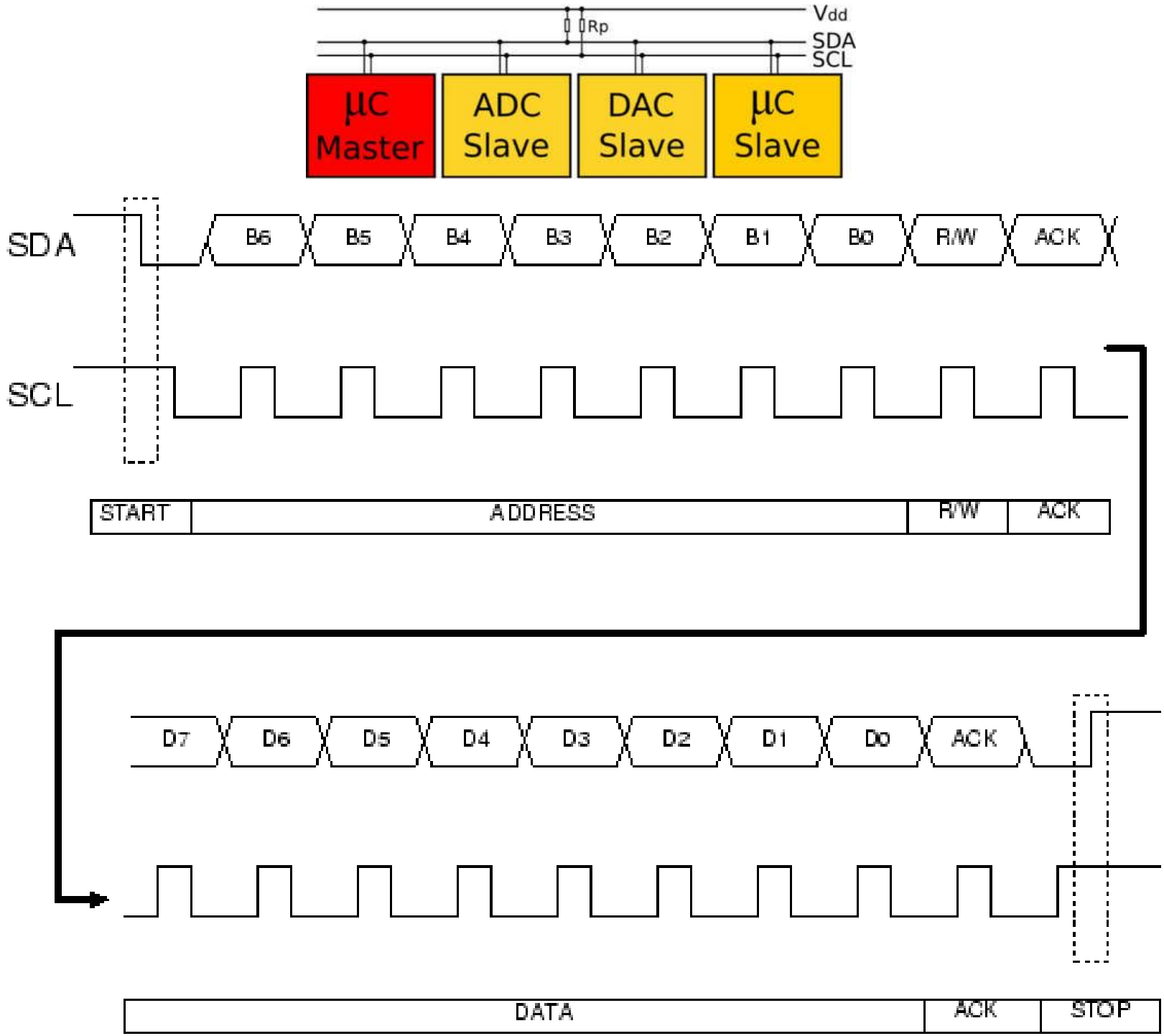
- **Mestre envia clock e endereço do escravo.**

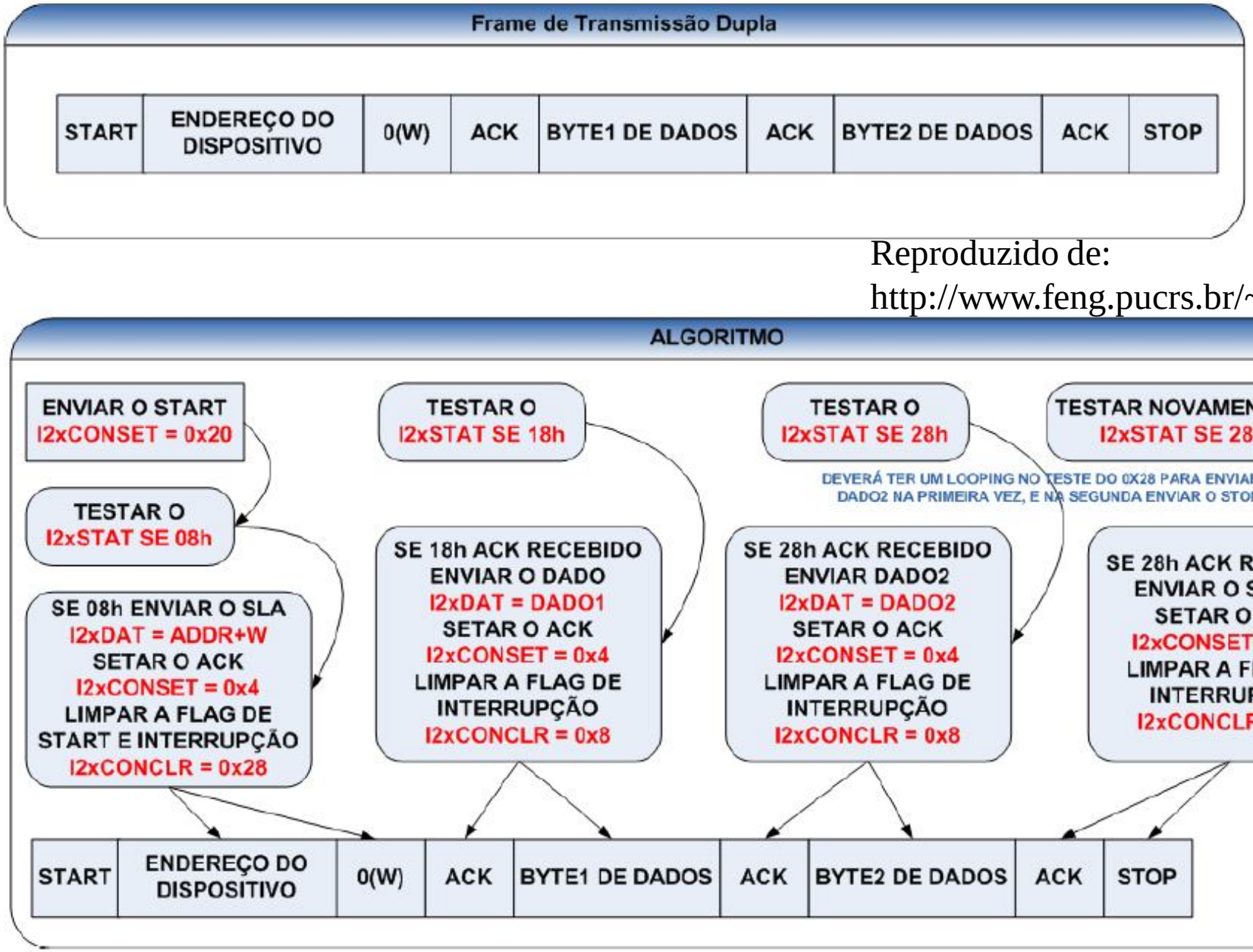
- **7 bits são utilizados para endereçamento; há 16 endereços reservados => máximo de 112 escravos**

- **CIs tem endereço fixo ou os bits menos significativos do endereço do dispositivo são especificados por nível lógico atribuído a pinos**

- **Velocidade de transmissão de dados => Até 3,4 Mbi/s no modo *High Speed***

- **Mestre e escravo podem trocar de papel após bit de stop.**

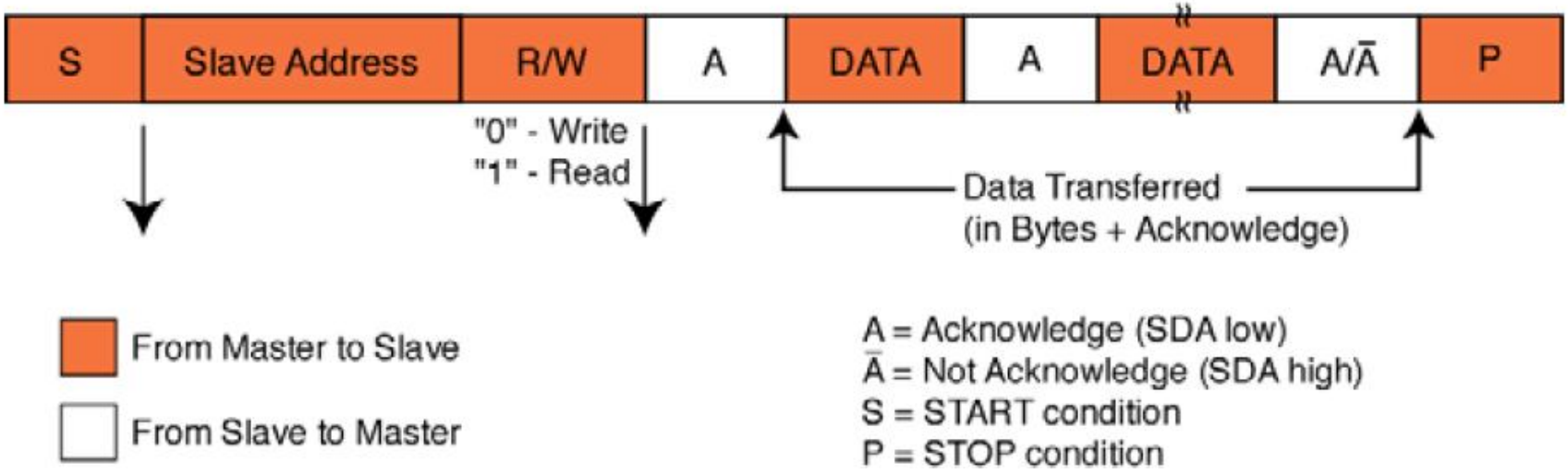

Vdd
Rp
SDA
SCL
µC Master
ADC Slave
DAC Slave
µC Slave
SDA
B6
B5
B4
B3
B2
B1
B0
R/W
ACK
SCL
START
ADDRESS
R/W
ACK
D7
D6
D5
D4
D3
D2
D1
D0
ACK
DATA
ACK
STOP



Reproduzido de:
http://www.feng.pucrs.br/~jbenfica/

Figura 2: Diagrama para Transmissão Dupla

Etapa	Escrita	Leitura	Nro Bits	Campo
1	START bit	START bit	1	START
2	Slave address	Slave address	7	ADDRESS
3	bit 0	bit 1	1	W/R
4	Aguarda ACK (‘0’) => “recebido”	Aguarda ACK (‘0’) => “recebido”	1	ACK
5	Envia dados	Recebe dados	8	DATA
6	Aguarda ACK (‘0’) Volta a 5 ou vai a 7	Envia ACK (‘0’) e volta a 5; ou NACK (‘1’) e vai a 7	1	ACK
7	STOP bit	STOP bit	1	STOP



Registradores de Controle das Interface I2C - 1

I2CONSENT (*I2C Control Set Register*):

BIT	CAMPO	FUNÇÃO
0 e 1 e 7	Reservados	
2	AA - assert ack. flag	1= > ACK foi recebido
3	SI - interrupt flag	1=> Status do I2C mudou
4	STO -- stop flag	1=> Transmite STOP ; faz I2C mestre
5	STA – start flag	1=> Transmite START; faz I2C mestre
6	I2EN - I2C enable	1=> Interface I2C habilitada.

OBS: SI deve ser resetado por software, escrevendo 1 ao bit 3 do I2CONCLR.

I2CONCLR (*I2C Control Clear Register*): 1 => Reseta bit correspondente do I2CONSENT; 0 => sem efeito.

I2ADR (*I2C Slave Address Register*): Contém endereço de 7 bits do escravo. Bit 0 em1 => *general call*.

I2DAT (*I2C Data Register*): Contém dado a ser enviado ou recebido.

I2STAT (*I2C Status Register*): Contém código que informa resultado de operação.

Registradores de Controle das Interface I2C - 2

I2CnSCLH (I2C SCL High Duty Cycle Register): Registrador de16 bits que especifica tempo de nível lógico alto do clock do I2C em termos de nro. de ciclos de clock do PCLK

I2CnSCLL (I2C SCL Low Duty Cycle Register): Registrador de16 bits que especifica tempo de nível lógico baixo do clock do I2C em termos de nro. de ciclos de clock do PCLK

$$I2CCLK = PCLK/(I2CxSCLH+I2CxSCLL)$$

I2CSTAT : Master Transmitter mode

Valor	Significado
0x08	Condição de START foi transmitida
0x10	Condição de START foi re-transmitida
0x18	SLA+W foi transmitido; ACK foi recebido
0x20	SLA+W foi transmitido; NACK foi recebido
0x28	I2DAT foi transmitido; ACK foi recebido
0x30	I2DAT foi transmitido; NACK foi recebido

Funcão que configura a I2C_0:

```
void init_i2c0(void)
{
PCONP |= (1<<7);//0x04000080;        // Energiza do I2C0 e I2C2
PINSEL1 |= 0X1400000;
I20CONCLR = 0xff;                    //Limpa flags
I20CONSET = 0x40;                    /* Habilita o I2C-2 em modo mestre */
I20SCLH = 100;                               /* Tempo alto do SCL */
I20SCLL = 100;                               /* Tempo baixo do SCL */
}
```

LM75A: Digital Temperature Sensor

Faixa de medição: -55 C a +125 C

11-bit ADC com resolução de 0.125 C

I2C Address:1001xxx; onde xxx=000 no kit

Table 9. Temp register

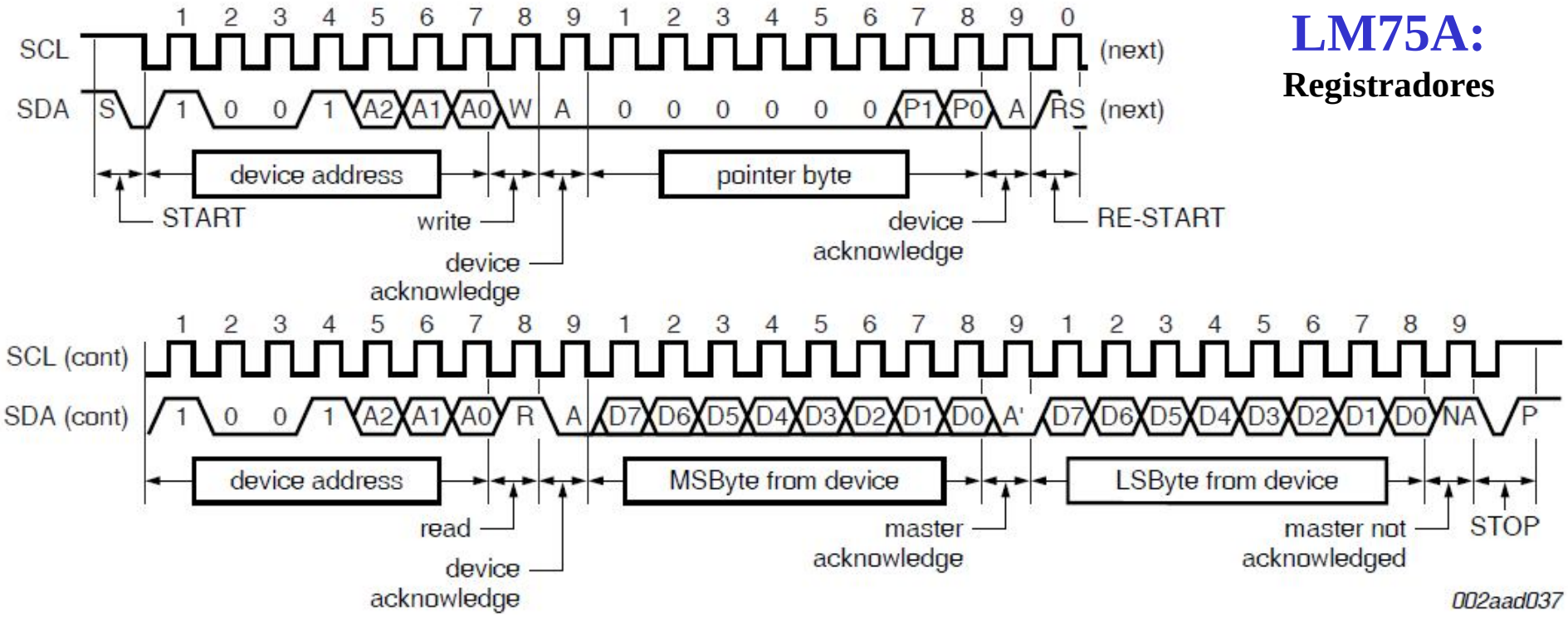
MSByte								LSByte							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	X	X	X	X	X

Se D10 = 0 => Temp value (°C) = +(Temp data) x 0.125 °C.

Se D10 = 1 => Temp value (°C) = = -(2's complement of Temp data) x 0.125 °C.

LM75A:

Registradores



1.11 POINTER REGISTER (Selects which registers will be read from or written to):

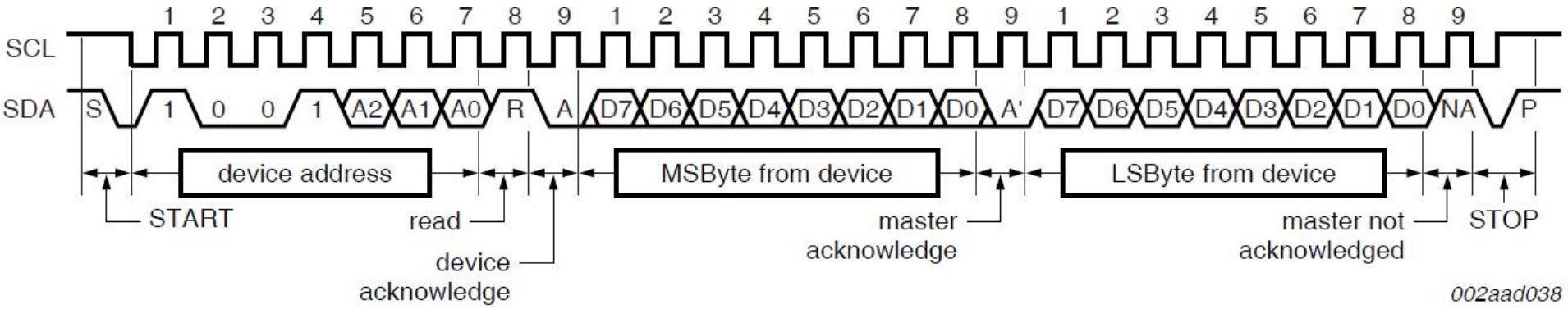
P7	P6	P5	P4	P3	P2	P1	P0
0	0	0	0	0	Register Select		

P0-P1: Register Select:

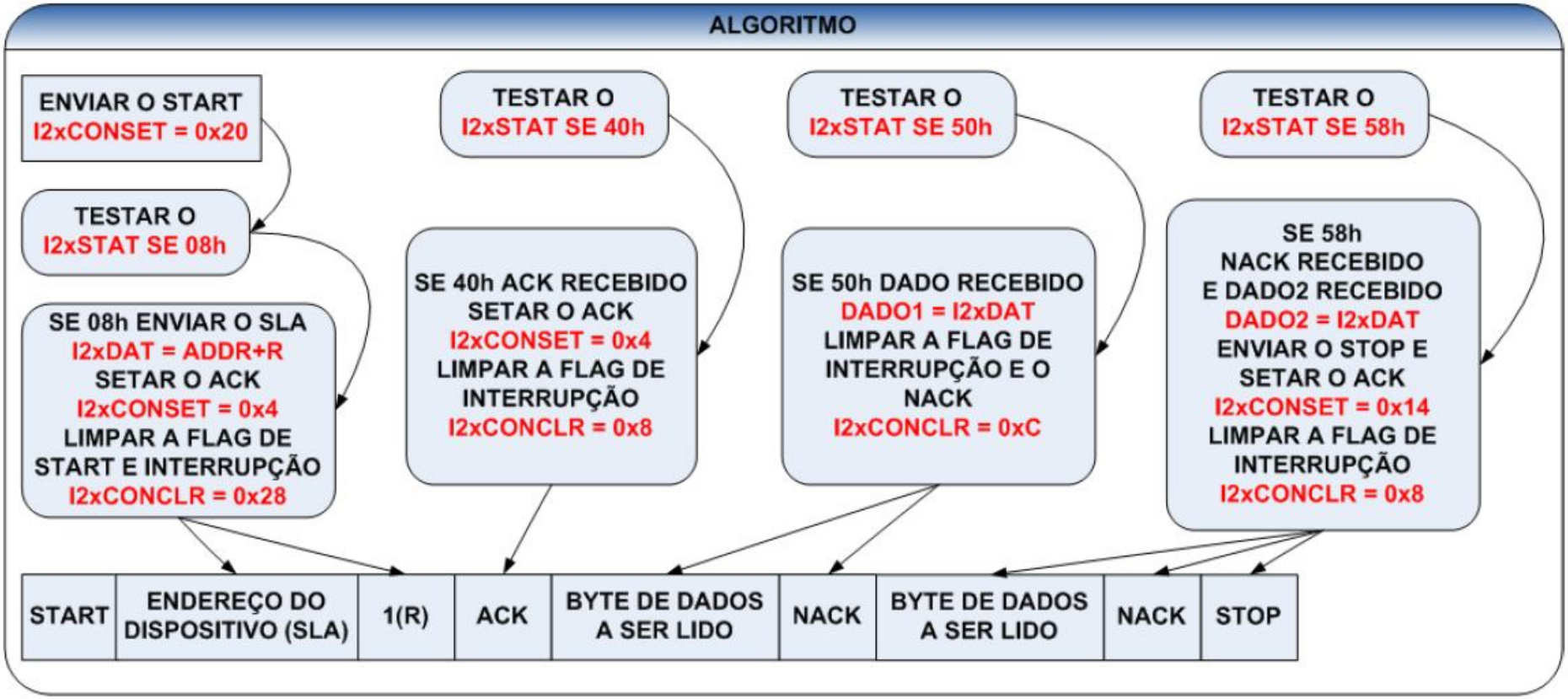
P2	P1	P0	Register
0	0	0	Temperature (Read only) (Power-up default)
0	0	1	Configuration (Read/Write)
0	1	0	T_{HYST} (Read/Write)
0	1	1	T_{OS} (Read/Write)
1	1	1	Product ID Register

LM75A: Digital Temperature Sensor

Após ligado, é possível apenas ler os *temp registers*, pois valor *default* do *pointer register* permite acesso aos mesmos.



Reproduzido de:
http://www.feng.pucrs.br/~jbenfica/



I2CSTAT : Master Receiver mode

Valor	Significado
0x08	Condição de START foi transmitida
0x10	Condição de START foi re-transmitida
0x40	SLA+R foi transmitido; ACK foi recebido
0x48	SLA+R foi transmitido; NACK foi recebido
0x50	Dado foi recebido; ACK foi enviado
0x58	Dado foi recebido; NACK foi enviado

```
fixa_while=1                    // Trecho de código para leitura I2C de sensor de temperatura
I20CONSET=0x20;                         // ENVIO DE BIT DE START.
 while (fixa_while==1)
{
        switch (I20STAT) {

            case 0x08:              // START recebido
                        I20DAT=0x91;        //          ENVIA ADDR+R
                        I20CONCLR=0x28;           //LIMPA FLAG DE START e DE INT
                        break;

            case 0x40:                      // SLA+R foi transmitido;
                        I20CONSET=0x04;             //          ACK
            I20CONCLR=0x08;             //LIMPA FLAG DE INT
                        break;

             case 0x48:                     // START não recebido
                        I20CONSET=0x20;           // envia novo start
            I20CONCLR=0x08;           //LIMPA FLAG DE INT
                        break;

            case 0x50:                      // Dado foi recebido; ACK foi enviado
                        dado=I20DAT;
                        I20CONCLR=0x0C;           //LIMPA FLAG DE  INT e ENVIA ACK
                        break;

            case 0x58:                      //Dado foi recebido; NACK foi enviado
                        fixa_while=2;
                        I20CONSET=0x14;             //   ENVIA           NACK e STOP BIT
                        I20CONCLR=0x08;             //  LIMPA FLAG DE INT
                                }           // end switch
} // end while
```

```
I20CONSET=0x20;

    while (fixa_while==0)
    {
          switch (I20STAT)  {

                case 0x08:
                      I20DAT=0x90;       //          ADDR+w
                      I20CONSET=0x04;            //          ACK
        I20CONCLR=0x28;           //  LIMPA FLAG DE START E DE INT
                      break;

                case 0x18:
                      I20DAT=0x0;        //          ADDR+w
                      I20CONSET=0x04;            //          ACK
                      I20CONCLR=0x08;            //          LIMPA FLAG DE START E DE INT
                      break;

                case 0x28:
                      if (!cont) {            cont=1;
                            I20DAT=0x0;       // ADDR+w
                            I20CONSET=0x04;           // ACK
                            I20CONCLR=0x08;     // LIMPA FLAG DE START E DE INT
                          }
                      else {      cont=0;
        fixa_while=1;
                            I20CONSET=0x14;           // ACK
                            I20CONCLR=0x08;     // LIMPA FLAG DE START E DE INT
                        }
                      } //end switch
        }           //end while
```

Conversor Digital Analógico

Característica:

- **Conversor de 10 bits**
- **Saída *"Buffered"***
- **Modo *power down***
- **Trade off entre velocidade e potência**

Configuração Básica:

1. **Energização**: DAC sempre energizado.

2. **Clock:** No registrador PCLK_SEL0 (Tabala 4–49), setar PCLK_DAC.

3. **Pinos**: Selecionar saída DAC (PINSEL1 e PINMODE1 – P1.20 e 21 => PINSEL1 | = 0x00200000; para selecionar P0.26 em AOUT)

D/A Converter Register (DACR)

5:0 - Reservados

15:6 – **VALUE**: Preencher com valor de 0 a 1024. Vout = VALUE/1024 × VREF.

16 - **BIAS**: Tempo de estabilizacão de Vout

0 =>1us e corrente de saída de 700uA.
1 => 2,5us e corrente de saída de 350uA.

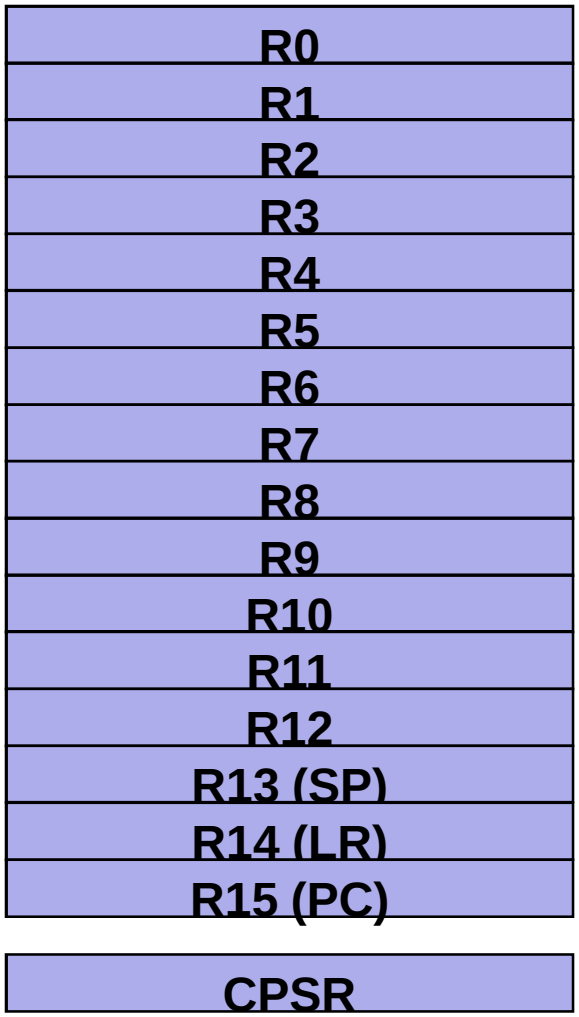
31:17 - Reservados

Funcão que configura D/A:

```
void escreva_DA (int dado)
{
int DAC_BIAS=0;                  //escolher 0 ou 1 conforme configuracão
PINSEL1 |= 0x00200000;
DACR = (dado<<6) | DAC_BIAS;
}
```

ARM7: Advanced RISC Machine

- **RISC**: Reduced Instruction Set Computer



- **16 registradores de 32 bits (R0-R15):**
  - **Propósito Geral: R0-R12**
  - **R13: *Stack Pointer* (SP)**
  - **R14: Guarda end. retorno de rotina (*Link Reg*.)**
  - **R15: *Program Counter* (PC)**

ARM7 - Pipeline

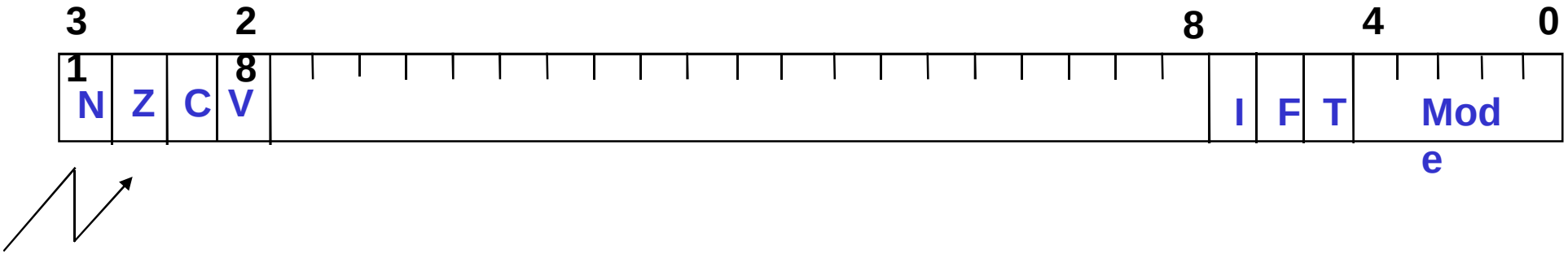
- Pipeline de 3 estágios: busca, decodifica, executa. Estágios estão deslocado de 4 bytes (32bits)

Endereço	Operação
PC	Busca
PC-4	Decodifica
PC-8	Executa

- 0x4000 LDR PC,[PC,#4] => PC receberá o valor 0x400C

- Tipos de dados de 8, 16 e 32 bits.

CPSR e SPSR: Program Status Registers



Copies of the ALU status flags (latched if the instruction has the "S" bit set).

- **Flag setado qdo:**

N = resultado negativo
Z = resultado nulo
C = vai ou vem 1 em operações aritméticas
V = overflow em operações aritméticas

.Bits setados desabilitam:

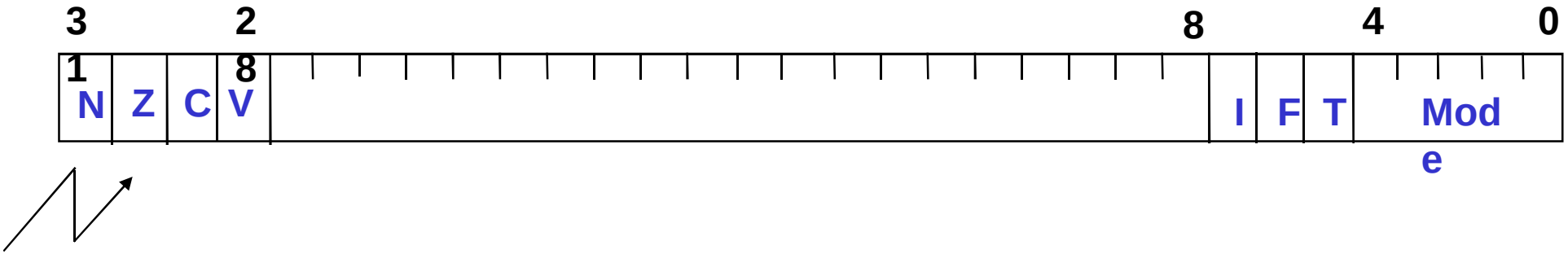
I = IRQ.
F = FIQ.

.T Bit define estado (acima de v4T):

T = 0, ARM
T = 1, Thumb

*** Modo [4:0]: Alterado por mudança de fluxo de execução do programa (exceção)**

CPSR e SPSR: Program Status Registers



Copies of the ALU status flags (latched if the instruction has the "S" bit set).

Para manipulação, são divididos em 4 campos de 8 bits:

Control (**c**): bits 0-7: 5 bits de modo, I & F bits de habilitação de interrupção, e o bit T do ARMv4T.

Extension (**x**): bits 8-15: Reservado para uso futuro (não usado em versões 3, 4 & 4T)

Status (**s**): bits 16-23: Reservado para uso futuro (não usado em versões 3, 4 & 4T)

Flags (**f**): bits 24-31: NZCV flags (28-31) e 4 bits (24-27) Reservado para uso futuro

CPSR e SPSR: Alterando Valor

- **MRS** e **MSR**: instruções que transferem conteúdo de CPSR ou SPSR **para**/**do** registrador.

- Sintaxe:
  - **MRS{<cond>} Rd,<psr> ; Rd = <psr>**
  - **MSR{<cond>} <psr[_fields]>,Rm ; <psr[_fields]> = Rm**

onde

  - **<psr> = CPSR or SPSR**
  - **[_fields] = campo qualquer de <psr>**

- Aceita dados imediatos:

  - **MSR{<cond>} <psr_fields>,#Immediate;**

  - **Exemplo: MSR CPSR_c, #0x1F ; go to System mode, IRQ & FIQ enabled**
  - In User Mode, all bits can be read but only the condition flags (_f) can be written.

As CPUs ARM7 (versão 4T da arquitetura) podem executar dois conjuntos de instruções:

ARM (34 instruções de 32 bits):

– Acesso ao conjunto de 16 registradores da CPU (uso geral)

– Execução condicional de todas as instruções

– Suporte a instruções para co-processadores

Thumb (30 instruções de 16 bits)

– Aumento da densidade de código (comparado ao modo ARM).

– Código Thumb ocupa espaço cerca de 30% menor que a mesma tarefa codificada com instruções ARM. 40% mais lento que ARM.

– Em virtude da largura reduzida:

» Acesso direto a apenas 8 registradores (uso geral)
» Inexistência de instruções condicionais (a não ser os desvios condicionais)
» Inexistência de instruções para co-processadores
» Impossibilidade de alterar o modo de processamento.

Estado Thumb (conjunto de 16 bits)

• Disponíveis 30 instruções (cada uma com largura de 16 bits)

• Características

– Trata-se de um subconjunto de instruções da ARM comprimidas de 32 para 16 bits

– Estas instruções são decomprimidas em tempo real e convertidas nas instruções ARM equivalentes, antes de serem executadas.

– Em virtude da largura reduzida:

» Acesso direto a apenas 8 registradores (uso geral)

» Inexistência de instruções condicionais (a não ser os desvios condicionais)

» Inexistência de instruções para co-processadores

» Impossibilidade de alterar o modo de processamento.

» Não podem se valer de instruções encontradas no modo ARM como: deslocamento de um dado, MAC, etc.

- O ARM tem 7 modos básicos de funcionamento:

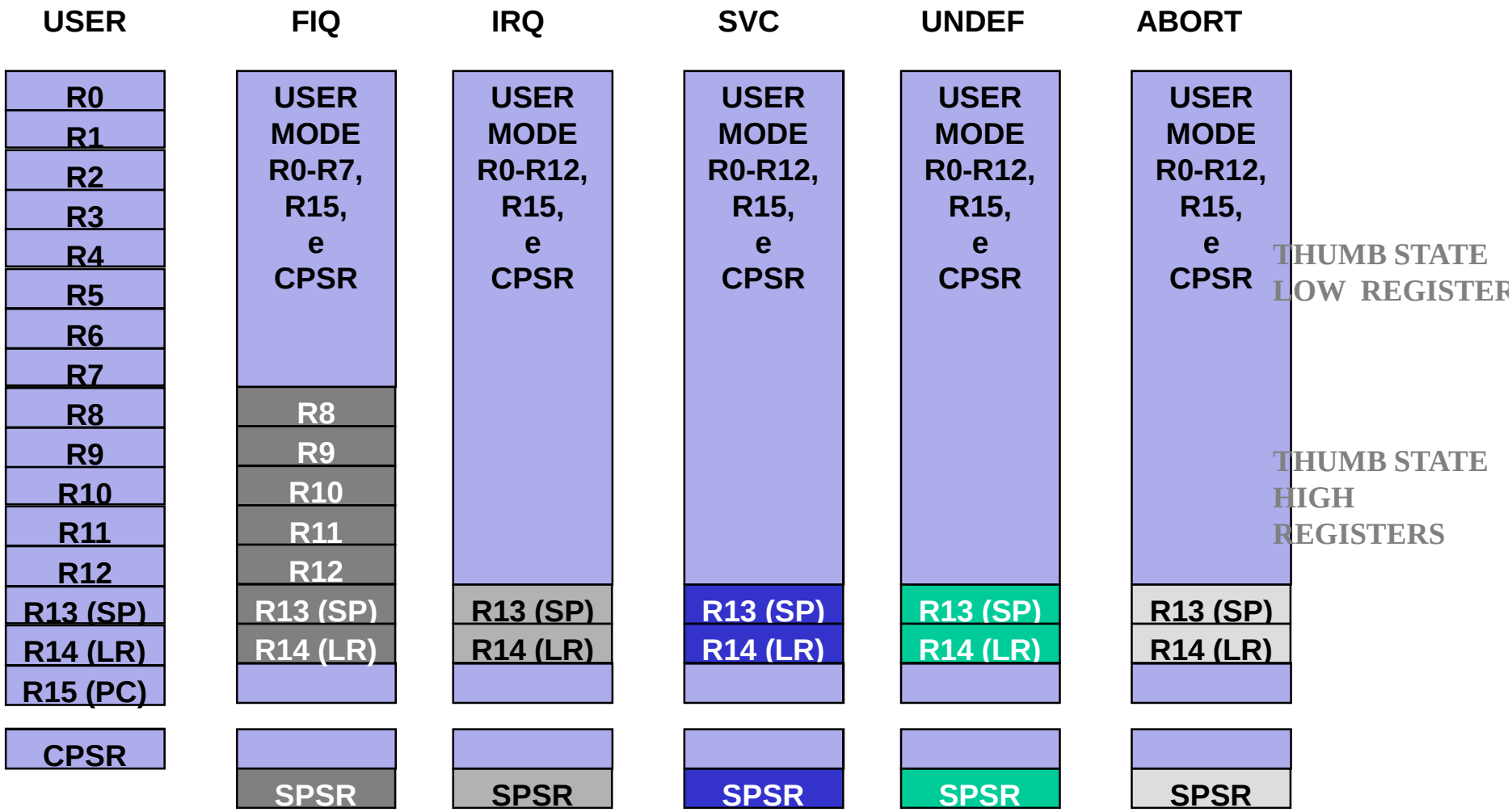
  - **User : Modo não privilegiado no qual a maioria das tarefas são executadas.**
  - **FIQ : Tratamento de interrupção de alta prioridade (fast).**
  - **IRQ : Tratamento de interrupção de baixa prioridade (normal).**
  - **Supervisor : Tratamento de reset e de interrupção de software.**
  - **Abort : Tratamento de violações de acesso à memória.**
  - **Undef : Tratamento de instruções indefinidas.**
  - **System : Modo privilegiado que usa os mesmos registradores do user mode e acessa rotinas de S.O.**

Program Status Register - CPSR

CPSR[4:0]	Mode	Use	Registers
10000	User	Normal user code	user
10001	FIQ	Processing fast interrupts	_fiq
10010	IRQ	Processing standard interrupts	_irq
10011	SVC	Processing software interrupts (SWIs)	_svc
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined instruction traps	_und
11111	System	Running privileged operating system tasks	user

Modos de Exceção: Interrupção e Erro

Cada modo possui R13, R14 e SPSR próprios. Modo Fiq possui ainda seus R8 a R12. Total de 37 registradores internos



OBS: SYSTEM MODE usa os mesmos registradores do user mode e acessa rotinas de S.O

Tabela de Vetores de Exceção

Exceção	Modo	Endereço
Reset	Supervisor	0x00000000
Instrução indefinida	Undefined	0x00000004
Software Interrupt	Supervisor	0x00000008
Aborta Instrução	Abort	0x0000000C
Aborta Dado	Abort	0x00000010
VERSOES ANTIGAS	RESERVADO	0x00000014
Interrupção	IRQ	0x00000018
Interrupção Rápida	FIQ	0x0000001C

Tratamento de Exceção e Tabela de Vetores

- **Quando exceção ocorre:**

1) **Copia CPSR em SPSR_<modo de exceção>**
2) **Seta bits do CPSR (Mode field)**
   - **R13 e R14 substituídos pelos do modo**
   - **Entra em estado ARM (se em Thumb)**
   - **Desabilita flag de interrupção (I=0;se for o caso)**

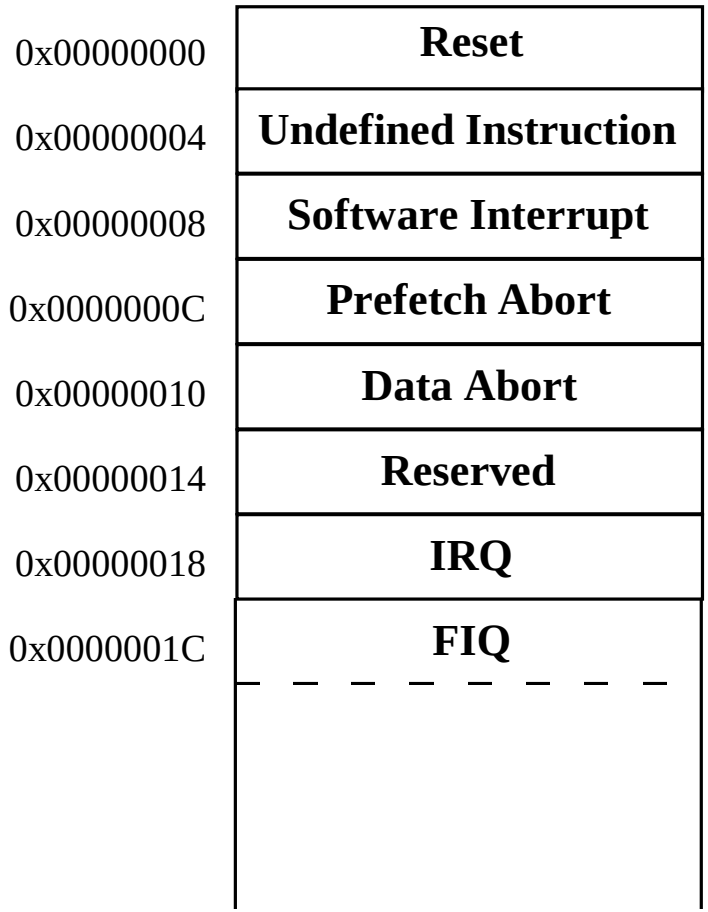
3) Salva endereço de retorno (PC+4) em R14

4) Carrega PC com o endereço do vetor do modo de exceção

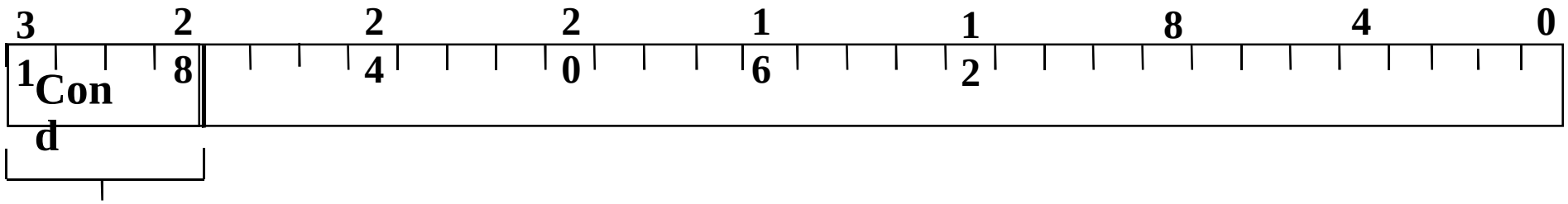
- **Ao retornar da exceção:**

1) Recupera CPSR de SPSR_<mode>

2) Recupera PC de R14_<mode>



Condition Field

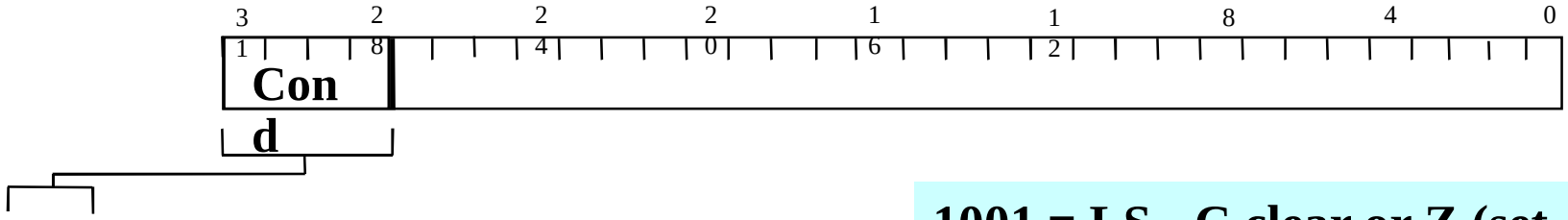


Instruções contém campo de condição, sendo executada se seus 4 MSB bits são iguais aos 4 MSB bits de CSPR. Caso contrário, NOP.

Desnecessário o uso de saltos condicionais que degradam pipeline (3 cycles to preencher).

- Existem 16 condições.

Condition Field



0000 = EQ - Z set (equal)

0001 = NE - Z clear (not equal)

0010 = HS / CS - C set (unsigned higher or same)

0011 = LO / CC - C clear (unsigned lower)

0100 = MI -N set (negative)

0101 = PL - N clear (positive or zero)

0110 = VS - V set (overflow)

0111 = VC - V clear (no overflow)

1000 = HI - C set and Z clear (unsigned higher)

1001 = LS - C clear or Z (set unsigned lower or same)

1010 = GE - N set and V set, or N clear and V clear (>or =)

1011 = LT - N set and V clear, or N clear and V set (>)

1100 = GT - Z clear, and either N set and V set, or N clear and V set (>)

1101 = LE - Z set, or N set and V clear,or N clear and V set (<, or =)

1110 = AL - always

1111 = NV - reserved.

Condition Field

Seja a instrução:

ADD r0,r1,r2 ; r0 = r1 + r2 (ADDAL)

Para executá-la somente se flag de zero estiver setado, usar sufixo EQ:

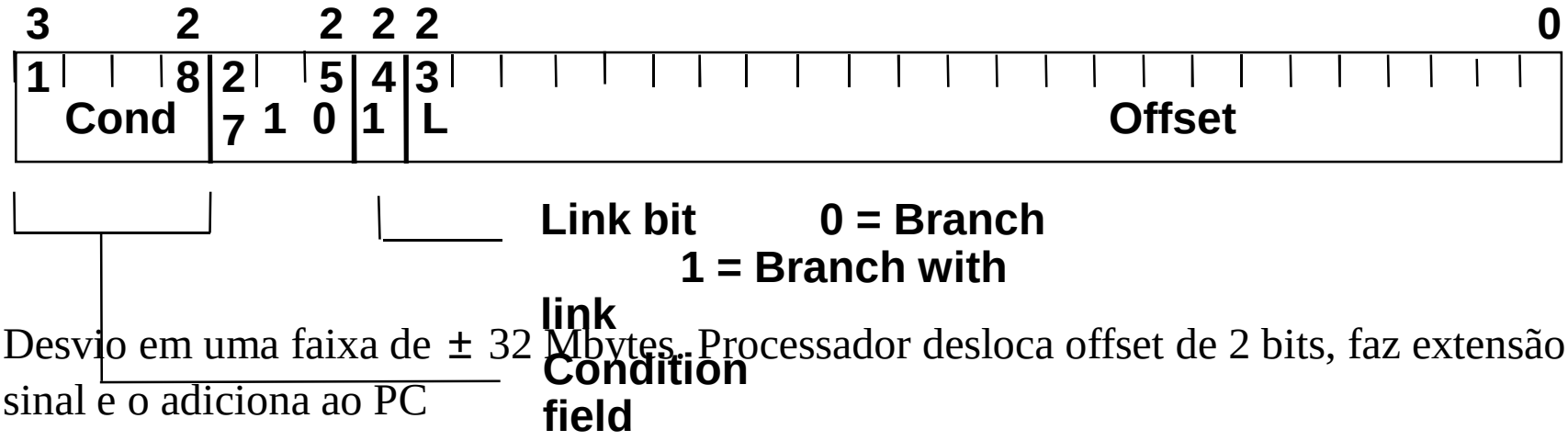
ADDEQ r0,r1,r2 ; Se flag de zero estiver setado …
; ... r0 = r1 + r2

Para forçar alteração de flags (à exceção de instruções de comparação), usar sufixo "S":

ADDS r0,r1,r2 ; r0 = r1 + r2 ; ... e altera flags

Instruções de Desvio (Branch)

- **Branch :** **B{<cond>} label**
- **Branch with Link:** **BL{<cond>}sub_routine_label**



* Desvio em uma faixa de ± 32 Mbytes. Processador desloca offset de 2 bits, faz extensão de sinal e o adiciona ao PC

* BL salva endereço de retorno (PC+4) no registrador R14

* BX ou BLX cumprem a mesma tarefa, modificando o estado de ARM para Thumb (ou vice versa)

* **0x400 B 0x8000 => PC = 0x8000**
* **0x400 BL 0x8000 => PC = 0x8000 ; R14 = 0x404**

Instruções de Processamento de Dados

Aritimético:	**ADD ADC SUB SBC RSB RSC**
Lógico:	**AND ORR EOR BIC**
Comparação:	**CMP CMN TST TEQ**
Transporte de dados:	**MOV MVN**

- Estas instruções funcionam apenas em registradores (não em memória)
- Sintaxe: **<Operação>{<cond>} {S} Rd, Rn, Operando**
  reg reg (reg ou imediato)
  resultado, operando1, operando2

  - Comparação altera flags - não requer Rd
  - Transporte de dados não requer Rn

- Segundo operando vai para a ALU via barrel shifter.

Operações Arithméticas

- **ADD operando1 + operando2**
- **ADC operando1 + operando2 + carry**
- **SUB operando1 - operando2**
- **SBC operando1 - operando2 + carry -1**
- **RSB operando2 - operando1**
- **RSC operando2 - operando1 + carry – 1**

- **Sintaxe:**
  - **<Operação>{<cond>}{S} Rd, Rn, Operando2**

- **Exemplos**
  - **ADD r0, r1, r2**
  - **SUBGT r3, r3, #1**
  - **RSBLES r4, r5, #5**

Comparações

- **Apenas modificam os flags, sem modificar registrador. Sufixo “S” desnecessário**

- **Operações:**
  - **CMP operando1 – operando2**
  - **CMN operando1 + operando2**
  - **TST operando1 AND operando2**
  - **TEQ operando1 EOR operando2**

- **Sintaxe:**
  - **<Operação>{<cond>} Rn, Operando2**

- **Exemplos:**
  - **CMP r0, r1**
  - **TSTEQ r2, #5**

Transferência de Dados

- **MOV operand2**
- **MVN NOT operand2**

- **Sintaxe:**
  - **<Operação>{<cond>}{S} Rd, Operando2**

- **Exemplos:**
  - **MOV r0, r1**
  - **MOVS r2, #10**
  - **MVNEQ r1,#0**

Transferência de Dados

LDR	**STR**	Word
LDRB	**STRB**	Byte
LDRH	**STRH**	Halfword
LDRSB		Signed byte load
LDRSH		Signed halfword load

- OBS: Memória deve permitir o acesso de tais formatos

- Sintaxe:
  - **LDR**{<cond>}{<size>} Rd, <address>
  - **STR**{<cond>}{<size>} Rd, <address>

e.g. **LDREQB**

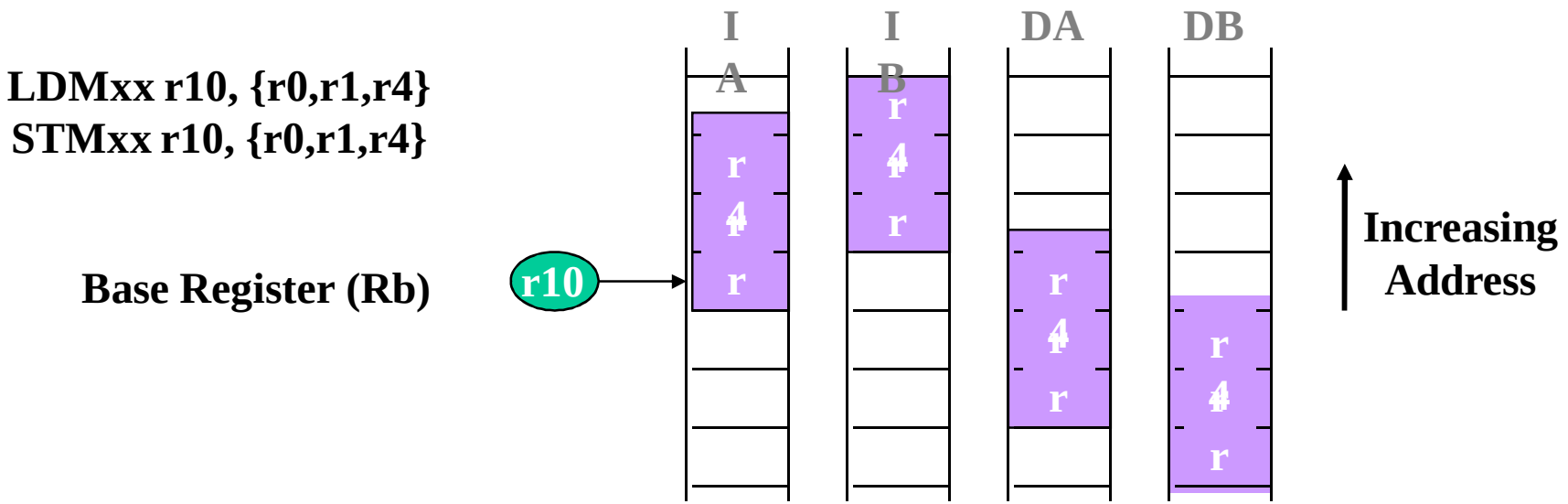
Transferência Multipla de Dados

- Sintaxe:

<LDM|STM>{<cond>}<addressing_mode> Rb{!}, <register list>

- 4 modos de endereçamento:

LDMIA / STMIA	incrementa depois
LDMIB / STMIB	incrementa antes
LDMDA / STMDA	decrementa depois
LDMDB / STMDB	decrementa antes



Operações Lógicas

- **AND operando1 AND operando2**
- **EOR operando1 EOR operando2**
- **ORR operando1 OR operando2**
- **BIC operando1 AND NOT operando2 [ie bit clear]**

- **Sintaxe:**
  - **<Operação>{<cond>}{S} Rd, Rn, Operando2**

- **Exemplos:**
  - **AND r0, r1, r2**
  - **BICEQ r2, r3, #7**
  - **EORS r1,r3,r0**