Universidade Federal de Santa Catarina
Centro Tecnológico – CTC
Departamento de Engenharia Elétrica
Laboratório de Comunicações e Sistemas Embarcados - LCS
Laboratório de Integração de Software e Hardware - LISHA
Grupo de Sistemas Embarcados - GSE

“Programação de Sistemas Embarcados”

Prof. Eduardo Augusto Bezerra

Eduardo.Bezerra@ufsc.br

Florianópolis, junho de 2014.

Características de programas em C para Firmware

Componentes básicos de um programa para uma arquitetura genérica

- Autoteste (boot)
- Configuração dos periféricos e CPU (boot)
- Inicialização dos periféricos (estado inicial)
- Execução da aplicação / laço infinito

Exemplo:

```
void main(){
    int i, c;
    inicia_display();     // mostra cursor, 2 linhas, blink
    limpa_display();      // posiciona 0, 0
    while (1) {
        escreve_string( "Teste... " );
    }
}
```

Características desse tipo de programa

Programador deve conhecer mapa de memória e I/O da arquitetura destino

Exemplo:
Para escrever em um LCD, a entrada RS (1 bit) do LCD deve estar em '1', a entrada EN (1 bit) do LCD deve estar em '1' e, após um delay deve ocorrer uma transição para '0' em EN após um byte estar disponível nos pinos de dados do LCD - link para simulador de LCD.

```
escreve_char:
setb P3.7          ; RS <- 1
setb P3.6          ; EN <- 1
mov P0, #42H       ; 'B'
lcall delay
clr P3.6           ; EN <- 0
```

Link para MOD51

```
void escreve_char( int ch ){
    delay();
    P3 |= 0x80;       // RS <- 1
    P3 |= 0x40;       // EN <- 1
    P0 = ch;          // caractere
    delay();
    P3 &= 0xBF;       // EN <- 0
}
```

Link para msc1210.h

Características desse tipo de programa

Exemplo de I/O - Interface com o LCD necessita:

- **8 bits para envio do byte a ser escrito**
- **1 bit para sinal de controle EN - habilita envio**
- **1 bit para sinal de controle RS**
  - **0 = escreve em IR (controle)**
  - **1 = escreve em ID (dado)**

Porquê são utilizadas as portas P0 e P3 do 8051 nesse exemplo?

Link para msc1210.h

```
void escreve_char( int ch ){
   delay();
   P3 |= 0x80;       // RS <- 1
   P3 |= 0x40;       // EN <- 1
   P0 = ch;          // caractere
   delay();
   P3 &= 0xBF;       // EN <- 0
}
```

Nível de abstração de programa em C

```
inicio: clr    P3.7        ; RS <- 0, configura display
        setb P3.6          ; EN <- 1
        mov P0, #38H       ; 8 bits, 2 linhas, 5x8
        lcall delay        ; chama rotina de delay
        clr P3.6           ; EN <- 0
        lcall delay        ; chama rotina de delay
        setb P3.6          ; EN <- 1
        mov P0, #0FH       ; display on, cursor on
        lcall  delay       ; chama rotina de delay
        clr P3.6           ; EN <- 0
        lcall delay        ; chama rotina de delay
        setb P3.7          ; RS <- 1, escreve dados
        setb P3.6          ; EN <- 1
        mov P0, #42H       ; 'B'
        clr P3.6           ; EN <- 0
        lcall  delay       ; chama rotina de delay
FIM: LJMP FIM
; Calculo da rotina de delay:
; T = 1 / 11,0592MHz = 90,4224537037ns
; 1 ciclo de maquina => 1c = 4*T = 361,689814815ns
; # ciclos em 1 seg = 1s/361,689814815ns = 2.764.800 ciclos
; # ciclos para execucao de delay() e' a soma do # ciclos
; para execucao de cada uma das instrucoes da rotina:
; # ciclos = 1c + { 1c + [ 1c + ( 2c ) * n + 2c ] * m + 2c } * q + 2c
; n, m, sao os limites dos lacos de repeticao
; q e' o numero de repeticoes dos lacos aninhados
; n = m = 255, e calculando q = 21,13 => 21
```

Link para a versão completa.

```
void inicia_display(){
   P3 |= 0x40;     P3 &= 0x3F;      P0 = 0x38;
   delay();
   P3 &= 0xBF;
   delay();
   P3 |= 0x40;     P3 &= 0x3F;      P0 = 0x0F;
   delay();
   P3 &= 0xBF;
}
void limpa_display(){
   P3 |= 0x40;     P3 &= 0x3F;      P0 = 0x01;
   delay();
   P3 &= 0xBF;
}
void escreve_char( int ch ){
   P3 |= 0x40;       P3 |= 0x80;      P0 = ch;
   delay();
   P3 &= 0xBF;
}
void main(){
   inicia_display();
   limpa_display();
   escreve_char('B');
}
// Calculo da rotina de delay? Melhor usar uma
// função de uma lib que use um timer.
```

Link para a versão completa.

Aplicações mais complexas levam a seleção de arquiteturas alvo com bom suporte de bibliotecas.

Exemplo: **mp3 player com chip 8051 (87C52) + chip decodificador mp3 (STA013) + FPGA**

- página do autor: http://www.pjrc.com/mp3/index.html
- código fonte disponível em: http://www.pjrc.com/mp3/firmware.html
- mapa de memória: http://www.pjrc.com/mp3/mem_map.html
- chip decodificador de mp3 STA013 com I2C: http://www.pjrc.com/mp3/sta013.html
- fontes em c e asm:

http://gse.ufsc.br/~bezerra/disciplinas/SistEmbarcados/mp3_player/mp3player0690/mp3/

Ambiente de desenvolvimento Renesas

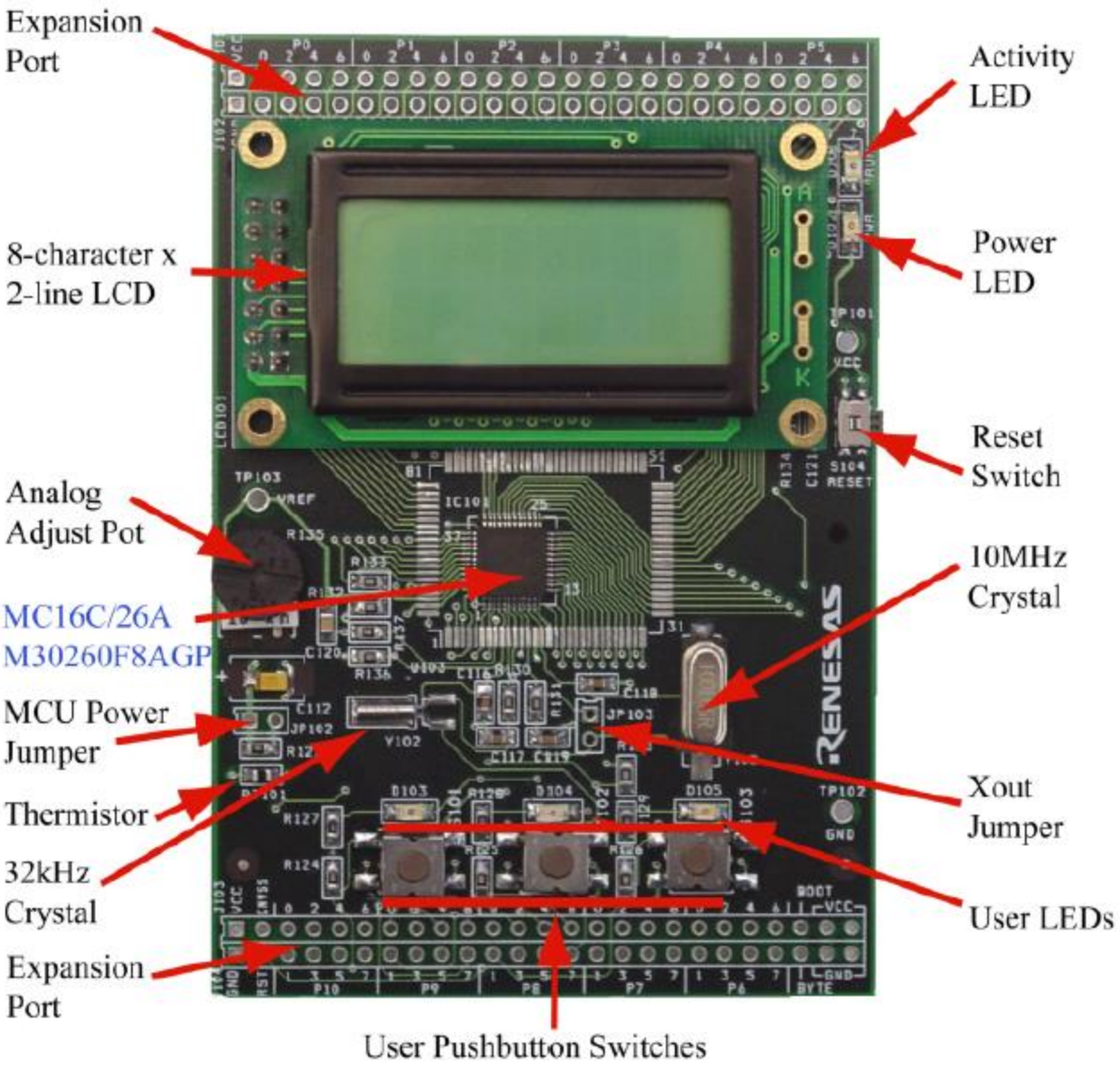
Documentação

- M16C_Software_Manual.pdf
- QSK26A User Manual.pdf
- QSK26AQuickStart.pdf
- QSK26A_Schematics.pdf
- QSK26A_Tutorial_2.ppt
- RTA-FoUSB-MON_Users_Manual.pdf
- rej09b0202_16c26ahm.pdf

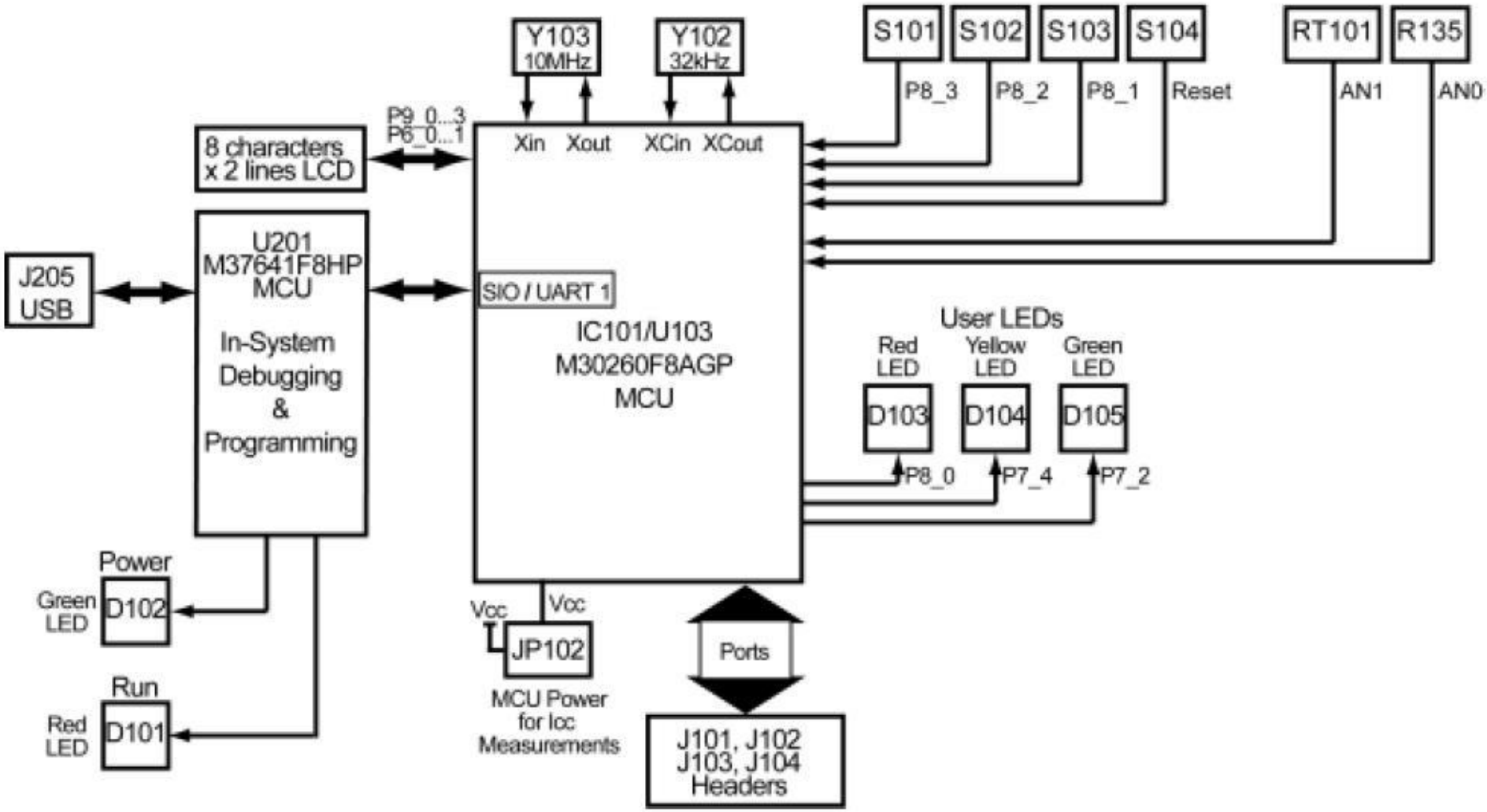
Kit QSK26A da Renesas

Programa prototipado na plataforma da Renesas

- **Renesas foi criada por divisões da Mitsubishi e Hitachi**
- **Utilizado microcontrolador da família M16C/26**
- **M16C/26 – MCU de 16 bits com CPU da série M16C/60**
- **Kit QSK26A conectado via USB (usado também como fonte)**
- **Manual de hardware M16C_Hardware_Manual_Rev0.9.pdf**



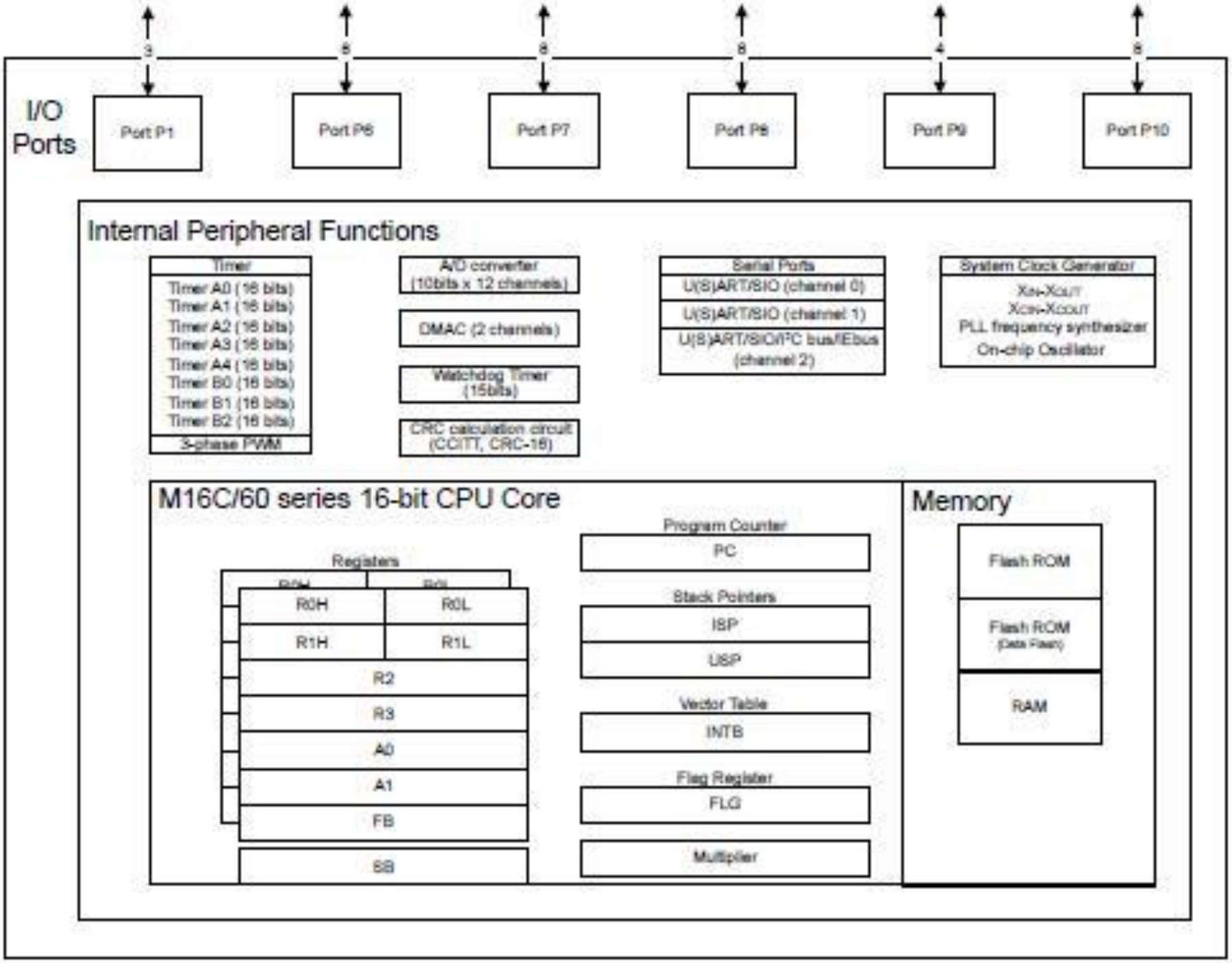
Kit QSK26A da Renesas



Kit QSK26A da Renesas

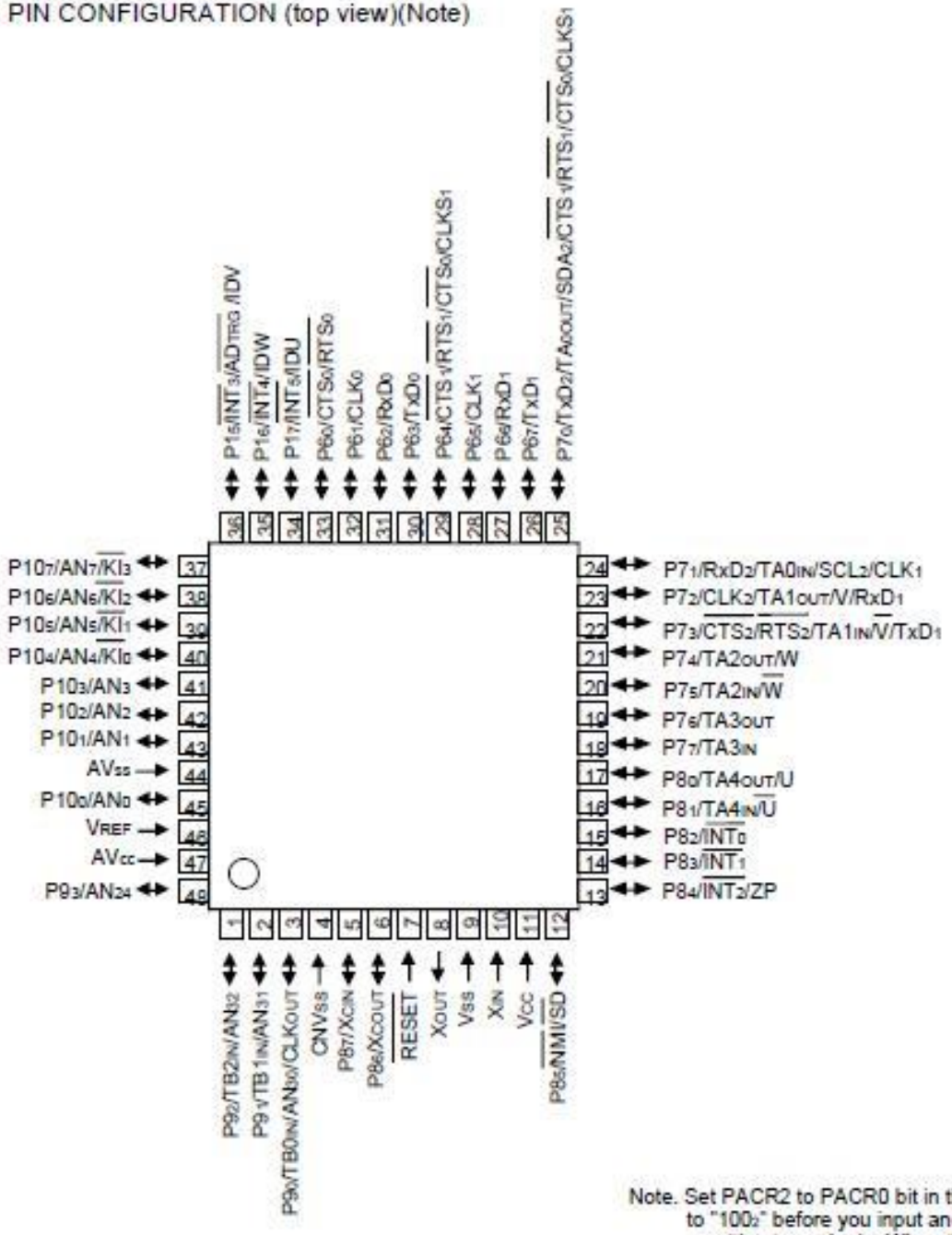
Item	Specification
MCU	M30260F8AGP
Clocks	Main Clock: crystal 10 MHz, PLL, or ring oscillator Sub Clock: 32.768 kHz crystal
Memory	RAM: 2kB (1920 Bytes user available due to kernel) High E/W Data Block: 2kB × 2 Flash ROM: 62kB (63,744 Bytes)
Connectors	[J101-J104]: Four 25-pin, single row, measurement test points connected to the MCU pins. Can also be used to connect your own expansion boards via 2×25 headers. [J205]: Mini-USB connector, used for in-circuit debugging and programming
Jumpers	[JP102]: MCU Power for Icc Measurements [JP103]: Oscillator Stop detection
Switches	[S101]: Pushbutton (connected to P8_3) [S102]: Pushbutton (connected to P8_2) [S103]: Pushbutton (connected to P8_1) [S104]: Pushbutton (connected to Reset)
LEDs	[D101] (Red): Run LED (in-circuit debugging/programming activity) [D102] (Green): Power [D103] (Red): User output (connected to P8_0) [D104] (Yellow): User output (connected to P7_4) [D105] (Green): User output (connected to P7_2)
LCD	2-line × 8-character LCD with KS0066 controller IC

Item		Performance
CPU	Number of Basic Instructions	91 instructions
	Minimum Instruction Execution Time	50 ns (f(BCLK)= 20MHz, Vcc= 3.0V to 5.5V) (M16C/26A, M16C/26T(T-ver.)) 100 ns (f(BCLK)= 10MHz, Vcc= 2.7V to 5.5V) (M16C/26A) 50 ns (f(BCLK)= 20MHz, Vcc= 4.2V to 5.5V -40 to 105°C) (M16C/26T(V-ver.)) 62.5 ns (f(BCLK)= 16MHz, Vcc= 4.2V to 5.5V -40 to 125°C) (M16C/26T(V-ver.))
	Operation Mode	Single chip mode
	Address Space	1M byte
	Memory Capacity	ROM/RAM : See the product list
Peripheral function	Port	Input/Output : 39 lines
	Multifunction Timer	TimerA:16 bits x 5 channels, TimerB:16 bits x 3 channels Three-phase Motor Control Timer
	Serial I/O	2 channels (UART, clock synchronous serial I/O) 1 channel (UART, clock synchronous, I²C bus[1] , or IEBus[2])
	A/D Converter	10 bit A/D Converter : 1 circuit, 12 channels
	DMAC	2 channels
	CRC Calcuration Circuit	2 polynomial (CRC-CCITT and CRC-16) with MSB/LSB selectable
	Watchdog Timer	15 bits x 1 channel (with prescaler)
	Interrupt	20 internal and 8 external sources, 4 software sources, 7 levels
	Clock Generation Circuit	4 circuits Main clock(*), Sub-clock(*) On-chip oscillator, PLL frequency synthesizer (*)These circuit contain a built-in feedback resister.
	Oscillation Stop Detection	Main clock oscillation stop, re-oscillation detection function
	Voltage Detection Circuit	Available(M16C/26A, Option[4]), Absent(M16C/26T)
Electrical Characteristics	Power Supply Voltage	Vcc=3.0V to 5.5V (f(BCLK)=20MHz) (M16C/26A) Vcc=2.7V to 5.5V (f(BCLK)=10MHz)
		Vcc=3.0V to 5.5V (M16C/26T(T-ver.))
		Vcc=4.2V to 5.5V (M16C/26T(V-ver.))
	Power Consumption	16mA (Vcc=5V, f(BCLK)=20MHz) 25 µA (Vcc=3V, f(BCLK)=f(XCIN)=32KHz on RAM) 1.8 µA (Vcc=3V, f(BCLK)=f(XCIN)=32KHz, In wait mode) 0.7 µA (Vcc=3V, In stop mode)
Flash memory Version	Program/Erase Supply Voltage	2.7V to 5.5V (M16C/26A) 3.0V to 5.5V (M16C/26T(T-ver.)) 4.2V to 5.5V (M16C/26T(V-ver.))
	Program and Erase Endurance	100 times (all area) or 1,000 times (block 0 to 3) / 10,000 times (block A, block B)[3]
Operating Ambient Temperature		-20 to 85°C / -40 to 85°C [5] (M16C/26A)
		-40 to 85°C (M16C/26T(T-ver.))
		-40 to 105°C / -40 to 125°C (M16C/26T(V-ver.))
Package		48-pin plastic molded QFP


I/O Ports
8
8
8
8
4
8
Port P1
Port P6
Port P7
Port P8
Port P9
Port P10
Internal Peripheral Functions
Timer
Timer A0 (16 bits)
Timer A1 (16 bits)
Timer A2 (16 bits)
Timer A3 (16 bits)
Timer A4 (16 bits)
Timer B0 (16 bits)
Timer B1 (16 bits)
Timer B2 (16 bits)
3-phase PWM
A/D converter (10bits x 12 channels)
DMAC (2 channels)
Watchdog Timer (15bits)
CRC calculation circuit (CCITT, CRC-16)
Serial Ports
U(S)ART/SIO (channel 0)
U(S)ART/SIO (channel 1)
U(S)ART/SIO/I²C bus/IEbus (channel 2)
System Clock Generator
XIN-XOUT
XCIN-XCOUT
PLL frequency synthesizer
On-chip Oscillator
M16C/60 series 16-bit CPU Core
Registers
R0H
R0L
R1H
R1L
R2
R3
A0
A1
FB
SB
Program Counter
PC
Stack Pointers
ISP
USP
Vector Table
INTB
Flag Register
FLG
Multiplier
Memory
Flash ROM
Flash ROM
(Data Flash)
RAM

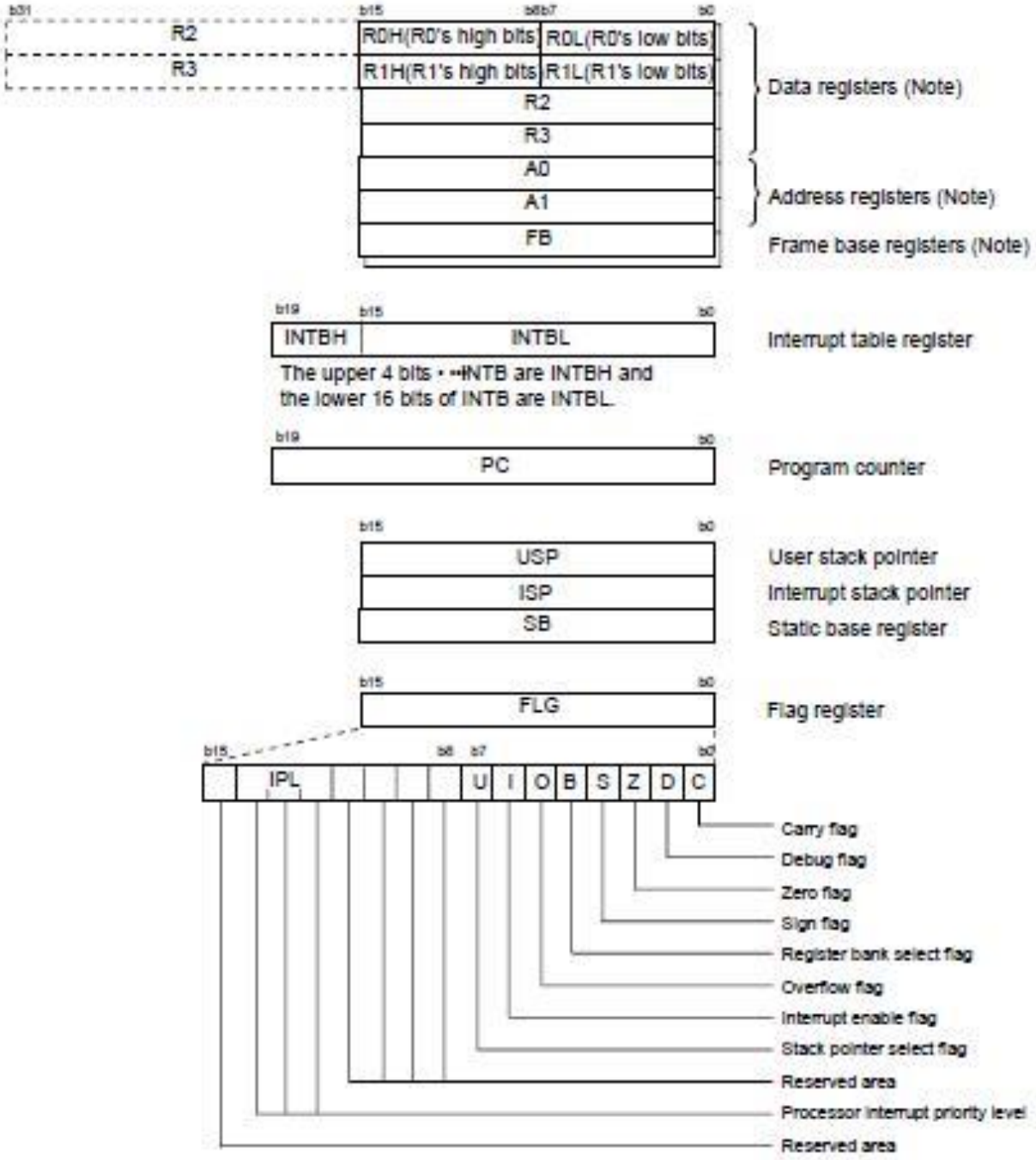
Kit QSK26A da Renesas

PIN CONFIGURATION (top view)(Note)



Note. Set PACR2 to PACR0 bit in the PACR register to "100$_2$" before you input and output it after resetting to each pin. When the PACR register isn't set up, the input and output function of some of the pins are disabled.

Kit QSK26A da Renesas



Note: These registers comprise a register bank. There are two register banks.

Ambiente de desenvolvimento da Renesas

- Tutorial para criação de projetos no HEW:

http://gse.ufsc.br/~bezerra/disciplinas/SistEmbarcados/renesas/public/QSK26A_Tutorial_2.ppt

- Análise dos arquivos gerados: ver slide 10 do tutorial

Exercício

Exercício - I/O básico

- Configurar portas de I/O
  - Entrada: botões
  - Saída: LEDs

- Ao pressionar botão, acende o LED correspondente

Em um PC seria o equivalente a conectar LEDs e botões em uma porta paralela, e realizar leitura/escrita nessa porta.

Comparação com o acesso I/O no PC

```
#include <stdio.h>
#include <sys/io.h>
int main(){
   if(ioperm(0x378, 3, 1)) {
     // acesso 0x378, 0x379, 0x37A
     printf("Erro! Precisa ser root.\n");
   } else {
     printf("Abriu a paralela\n");
     outb(0x55, 0x378);
     // envia 01010101 para paralela
   }
   return 0;
}
```

Comparação com o acesso I/O no PC

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main() {
  int fd, n;   char dado[1];    printf("Abrindo a porta paralela...\n");
  fd = open("/dev/port", O_RDWR | O_NOCTTY | O_NDELAY);
  if (fd != -1) {      // Sucesso!
    fcntl(fd, F_SETFL, 0);       dado[0] = 0xAA;
    printf("Escreve: %x hexa\n", dado[0]);
    lseek (fd, 0x378, SEEK_SET);
    n = write(fd, dado, 1);  // envia 1 byte para a porta paralela
    sleep(1);          dado[0] = 0x55;
    printf("Escreve: %x hexa\n", dado[0]);
    lseek (fd, -1, SEEK_CUR);
    n = write(fd, dado, 1);  // envia 1 byte para a porta paralela
    if (n < 0)
      printf("Erro! write() falhou.\n");
    else {
      fcntl(fd, F_SETFL, FNDELAY);
      n = read(fd, dado, 1); // leitura de 1 byte da paralela
      printf("Leu: %x hexa\n", dado[0]);
      fcntl(fd, F_SETFL, 0);
    }
  }
  else
    printf("Erro!! Nao conseguiu abrir a porta paralela!\n");
  return 0;
}
```

Exercício - Análise do Tempo de Resposta

- Configurar portas de I/O
  - Entrada: botões
  - Saída: LEDs
- Ao pressionar botão, altera o estado dos LEDs
- Análise do tempo de resposta de uma pessoa
  - Tempo de resposta ao estímulo do LED
- Plano de desenvolvimento
  - Utilizar um esqueleto de programa como base (tutorial)
  - Configurar e testar os LEDs
  - Configurar e testar os botões
  - Configurar e testar o LCD
  - Criar e testar a rotina de delay para acender LED

Exercício - Análise do Tempo de Resposta

```
#include "stdio.h"
#include "sfr262.h"
#include "SKP_LCD.h"
#define RED_LED (p8_0)
#define YEL_LED (p7_4)
#define GRN_LED (p7_2)
#define LED_ON  (0)
#define LED_OFF (1)
#define DIR_IN  (0)
#define DIR_OUT (1)
#define SW1 (p8_3)
#define SW2 (p8_2)
#define SW3 (p8_1)

void init_switches() {
  pd8_1 = pd8_2 = pd8_3 = DIR_IN;
}

void init_LEDs() {
  pd8_0 = pd7_4 = pd7_2 = DIR_OUT;
  RED_LED = YEL_LED = GRN_LED = LED_ON;
  RED_LED = YEL_LED = GRN_LED = LED_OFF;
}

void test_switches(void) {
  while (1) {
    RED_LED = (!SW1)? LED_ON : LED_OFF;
    YEL_LED = (!SW2)? LED_ON : LED_OFF;
    GRN_LED = (!SW3)? LED_ON : LED_OFF;
  }
}
```

link para projeto completo

```
void main () {
  char buf[9];
  long int i, r=12345;
  init_switches();
  init_LEDs();
  InitDisplay();
#if (1)
  test_switches();
#endif
  DisplayString(LCD_LINE1, "Response");
  DisplayString(LCD_LINE2, " Timer  ");
  while(1) {
    for (i=0; i<200000+(r%50000); i++)
      ;
    i=0;
    RED_LED = YEL_LED = GRN_LED = LED_ON;
    while (SW1)
      i++;

#if (1)
    sprintf(buf, "%8ld", i);
    DisplayString(LCD_LINE1, buf);
    DisplayString(LCD_LINE2, "iters.  ");
#else
    sprintf(buf, "%8.3f", i*39.1/287674);
    DisplayString(LCD_LINE1, buf);
    DisplayString(LCD_LINE2, "millisec");
#endif
    RED_LED = YEL_LED = GRN_LED = LED_OFF;
    r=0;
    while (!SW1) /* wait for switch to come up */
      r++;
  }
}
```

Projetos

- Requisitos da aplicação, quais recursos precisamos ter disponiveis?
  - LCD, timer, portas seriais, i2c, interrupcao, ethernet, usb, ad/da, wifi, can, bluetooth, rfid, ...
- Projetos básicos:
  - vending machine
  - acionamento de carga (lâmpadas) via pinos de I/O
  - gerar clock variável (seleção pelos botões) e medir no osciloscópio
- Projetos intermediários:
  - comunicação entre duas placas via UART;
  - leitura de código de barras via UART;
  - sensor de temperatura (conversor AD);
- Projetos avançados:
  - leitura/escrita de smart card (usando i2c nos pinos ou modo i2c da UART)

Desenvolvimento de aplicação com smart-card I²C e código de barras na plataforma ARM7