// Gustavo G. Plensack e Marcelo H. Gumiero - Projeto Final - #UARTSApp  
  
#include "derivative.h"  
  
#define TX\_MAX 100 // numero max de caracteres a serem transmitidos  
  
#define NMAX 1024 // numero max de elementos do buffer circular  
#define CICLOS\_TESTE 1000 // a cada mil ciclos do programa será enviado um char de verificação  
unsigned int cent\_global;  
char buffer[NMAX], buffer\_2[NMAX];  
int rp\_global=0, Rx\_Full, key1\_Full = 0, key2\_Full = 0, key3\_Full = 0 , RX\_full, RX\_full\_2, pare = 1, zere = 0;  
unsigned int po = 0, pi = 0, pi\_2=0, po\_2 = 0, n\_buffer = 0, n\_buffer\_2 =0; // respectivamente ponto de saida, ponto de entrada e numero de elementos do buffer  
char message\_2 [TX\_MAX+1];  
char RX\_data, RX\_data\_2; // variavel que armazena o dado da UART  
  
void init\_SysTick\_Handler(void);  
void SysTick\_Handler(void);  
void UART0\_IRQHandler(void); // trata a interrução do  
void delay10us( int t);  
void init\_GPIO\_LCD(); // inicializa as portas do GPIO para controlar LCD  
void init\_LCD(); // inicializa o LCD, configuração 2 colunas, limpeza do display, etc.  
void putchar\_LCD(char c); // imprime o código ASCII guardado em c no LCD  
// Esta função deve setar no início o sinal RS = 1, indicando que se está enviando dados ao LCD  
void sendcmd\_LCD(char c, int us); // envia código de comando para o LCD e espera us microssegundos  
// Esta função deve setar no início o sinal RS = 0, indicando que se está enviando comandos para o LCD  
void print\_LCD(char \*s); // envia mensagem ASCII armazenada no string s para o LCD  
void init\_UART0(); // configura UART 19200, 8 bits, sem paridade, 1stop 1 start bit  
void putchar\_UART0(char c); // imprime o código ASCII guardado em c no terminal serial  
char getchar\_UART0(); // leitura de caractere do teclado com retorno imediato<br />// caso o caractere não esteja pronto, retorna 0  
void print\_UART0( char \*s); // imprime string s codificada em ASCII no terminal serial  
void init\_GPIO(); // inicia a GPIO correspondente aos botoes  
int getkey();  
void preenche\_espaco (char \* string ,int i);  
void quebralinha();  
void init\_UART2();  
void putchar\_UART2(char c);  
void print\_UART2(char \*s);  
char getchar\_UART2();  
void UART2\_IRQHandler(void);  
void init\_LED();  
  
int main () {  
 // inicializacoes  
 init\_SysTick\_Handler();  
 init\_GPIO\_LCD();  
 init\_GPIO();  
 init\_LCD();  
 init\_UART0();  
 init\_UART2();  
 init\_LED();  
 int verifica\_conex = 1;  
 int teste\_conexao = 0;  
 int caractere\_especial = 0;  
 char c, d;  
 char message\_2 [TX\_MAX+1];  
 int i = 0, j = 0, autoriza = 0; // indice da sring message  
 int timer\_conex = 0;  
 while(1){  
  
  
  
  
 c = getchar\_UART0();  
 if (c){ // testa se algo foi recebido pela UART0  
 putchar\_UART2(c);  
 j++;  
 }else{  
 putchar\_UART2(0);  
 }  
  
 d = getchar\_UART2();  
  
 caractere\_especial = 0;  
  
 if(d){  
  
 if (d == 223){  
 sendcmd\_LCD(0xc0,4);  
 print\_LCD("OK!!");  
 verifica\_conex = 1;  
 caractere\_especial = 1;  
 timer\_conex=0;  
 }  
  
 if(d=='§'){  
 autoriza = 1;  
 //acende led  
 GPIOC\_PDOR = 0xFF;  
 GPIOC\_PSOR = (1<<10); // pulsa LE (Latch Enable) do 74573, em 1 (set)  
 GPIOC\_PCOR = (1<<10); // em 0 (clear)  
 caractere\_especial = 1;  
 }  
  
 if (d == 254) {  
 i = 0 ;  
 j = 0 ;  
 preenche\_espaco(message\_2,i); // destroi a mensagem  
 caractere\_especial = 1;  
 }  
  
 if (!caractere\_especial) {   
 message\_2[i] = d; // recebe o dado de mensagem  
 i++;  
 }  
 }  
  
 if (!verifica\_conex) {  
 timer\_conex++;  
 if (timer\_conex == CICLOS\_TESTE){  
 sendcmd\_LCD(0xc0,4);  
 print\_LCD("FAIL");  
 timer\_conex = 0;  
 }  
 }  
 teste\_conexao++ ; // atualiza a variavel pra enviar o teste de conexao  
  
 if (teste\_conexao == CICLOS\_TESTE ) {  
 putchar\_UART2 (223) ; // envia o ASCII 223  
 teste\_conexao = 0;  
 sendcmd\_LCD(0x80,4);  
 print\_LCD("Teste conexao ...");  
 verifica\_conex = 0 ;  
 }  
  
 switch(getkey()){  
  
 case 1: // autoriza leitura  
 if(autoriza){  
 preenche\_espaco(message\_2,i);  
  
 if(i){  
 print\_UART0(message\_2);  
 i = 0;  
 quebralinha();  
 j=0;  
 }  
 autoriza = 0;  
 //apaga leds  
 GPIOC\_PDOR = 0x00;  
 GPIOC\_PSOR = (1<<10); // pulsa LE (Latch Enable) do 74573, em 1 (set)  
 GPIOC\_PCOR = (1<<10); // em 0 (clear)  
 }  
  
 break;  
 case 2:  
 if(j){  
 putchar\_UART2('§'); // autoriza a leitura   
 }  
 break;  
 case 3:  
 putchar\_UART2(254); // autoriza a destruicao da mensagem  
 break;  
 }  
  
 if(j==TX\_MAX){  
 print\_UART0("Mensagem excedeu o tamanho máximo, por favor, envie!");  
 quebralinha();  
 j=0;  
 while(1){  
 if(getkey() == 1){  
 print\_UART0(message\_2);  
 i = 0;// clear msg  
 quebralinha();  
 break;  
 }  
 }  
 }  
 }  
 return 0;  
}  
  
void init\_UART2(){  
 SIM\_SCGC5 = SIM\_SCGC5 | (1<<13); //HABILITA port  
 SIM\_SCGC4 = SIM\_SCGC4 | (1<<12); //HABILITA UART2  
 //set mux(RX,TX)  
 PORTE\_PCR22 = 0x400;  
 PORTE\_PCR23 = 0x400;  
 //rate  
 UART2\_BDH = 0x00;  
 UART2\_BDL = 0x44;  
 //recebimento, envio de dados  
 UART2\_C2 = 0x0C;  
 UART2\_C2 = UART0\_C2 | (1<<5);  
 //habilita interrupção da UART0  
 NVIC\_ISER = NVIC\_ISER | (1<<14);  
  
  
 // habilita a interrupacao de TX  
 UART0\_C2 = UART0\_C2 | (1<<7);  
}  
  
  
void putchar\_UART2(char c){  
  
 while(n\_buffer\_2==NMAX){};  
 UART2\_C2 = 0x2C; // desabilita a interrupcao  
 buffer\_2[pi\_2]=c;   
 pi\_2 = (pi\_2 + 1)%NMAX;  
 n\_buffer\_2++;   
 UART2\_C2 = 0xAC; // reabilita a interrupcao  
}  
  
  
void print\_UART2(char \*s){  
 int i=0;   
 while (s[i]!='\0'){   
 putchar\_UART2 (s[i]);   
 i++;   
 }  
}  
  
char getchar\_UART2(){ // retorna caractere transmitido pela UART2, caso o caractere não esteja pronto, retorna 0  
 char c;  
 if (RX\_full\_2){ // verifica se existe um novo caractere na UART2 (Register Full)  
 c = RX\_data\_2; // lê o caractere  
 RX\_full\_2=0;  
 return c;  
 }  
 else return 0; // não existe caractere na UART  
}  
  
void UART2\_IRQHandler(void){  
 if(UART2\_S1 & UART\_S1\_RDRF\_MASK){ // testa se houve Recepção  
 RX\_data\_2 = UART2\_D;  
 RX\_full\_2 = 1;   
 }  
 else if ((UART2\_S1 & UART\_S1\_TDRE\_MASK)){ // testa se houve Transmissão  
 UART2\_D = buffer\_2[po\_2]; // envia o caractere pelo registrador de dados da UART2  
 po\_2=(po\_2+1)%NMAX;  
 n\_buffer\_2--;  
 if (n\_buffer\_2==0){  
 UART2\_C2 = 0x2C; // desabilita a interrupcao de transmissao   
 }  
 }  
  
}  
  
  
void quebralinha(){  
  
 putchar\_UART0(0x0A);  
 putchar\_UART0(0x0D);  
  
  
}  
  
void preenche\_espaco (char \* string ,int i){  
  
 while (i < TX\_MAX){  
 string [i] = ' ';  
 i++;  
 }  
 // coloca espacos em branco no fim da mensagem pra manter a formatacao e apagar lixos de mensagens anteriores  
}  
  
  
void delay10us( int t) {  
 t = t\*19;  
 while (t) {  
 //t--;  
 \_\_asm volatile(  
 "mov r1,r1 \n"  
 "ldr r3,[r7,#4] \n"  
 "sub r3,#1 \n"  
 "str r3,[r7,#4] \n");  
 }  
} // função de delay desenvolvida no último LAB  
  
  
void init\_GPIO(){  
  
 SIM\_SCGC5 = SIM\_SCGC5 | (1<<9); // Habilita PORTA  
 PORTA\_PCR4 = 0x00000100; // Configura MUX para PORTA bit 4  
 PORTA\_PCR5 = 0x00000100; // Configura MUX para PORTA bit 5  
 PORTA\_PCR12 = 0x00000100; // Configura MUX para PORTA bit 12  
 PORTA\_PCR12 = 0x00000100; // Configura MUX para PORTA bit 12  
  
}  
  
void init\_GPIO\_LCD() {  
  
 SIM\_SCGC5 = SIM\_SCGC5 | (1<<11);// setar o clock de PORTC  
 PORTC\_PCR0 = 0x00000100; // seta o MUX  
 PORTC\_PCR1 = 0x00000100; // seta o MUX  
 PORTC\_PCR2 = 0x00000100; // seta o MUX  
 PORTC\_PCR3 = 0x00000100; // seta o MUX  
 PORTC\_PCR4 = 0x00000100; // seta o MUX  
 PORTC\_PCR5 = 0x00000100; // seta o MUX  
 PORTC\_PCR6 = 0x00000100; // seta o MUX  
 PORTC\_PCR7 = 0x00000100; // seta o MUX  
 PORTC\_PCR8 = 0x00000100; // seta o MUX  
 PORTC\_PCR9 = 0x00000100; // seta o MUX  
 GPIOC\_PDDR = 0x03FF; // Define os 8 LSBs como saídas  
  
}  
  
void init\_LCD(){  
 // ligar o LCD  
  
 sendcmd\_LCD( 0x3F , 3005);  
  
 sendcmd\_LCD( 0x0E, 4); // Correção feita pelo professor para tirar o blink do cursor  
  
 delay10us(4); // espera 40 us  
  
 sendcmd\_LCD( 0x01, 4); // Limpa o display  
  
 sendcmd\_LCD( 0x06, 159); // Limpa o display  
}  
  
void sendcmd\_LCD( char c, int us){  
 GPIOC\_PCOR = (1<<8); // ativa comandos  
 delay10us(us); // espera n us  
 GPIOC\_PSOR = (1<<9); //liga o E  
 GPIOC\_PDOR = GPIOC\_PDOR & 0xFFFFFF00;  
 GPIOC\_PDOR = GPIOC\_PDOR | c; // Seleciona a instrução  
 GPIOC\_PCOR = (1<<9); //desliga o E  
  
}  
  
void putchar\_LCD( char c){  
 GPIOC\_PSOR = (1<<9); // E = 1  
 GPIOC\_PSOR = (1<<8); // ativa o RS para dado  
 GPIOC\_PDOR = (GPIOC\_PDOR & 0xffffff00) | c ;  
 delay10us(4);  
 GPIOC\_PCOR = (1<<9); // E = 0  
}  
  
void print\_LCD (char \*string){  
 int n = 0;  
 while (string[n] != '\0'){  
 putchar\_LCD(string[n]);  
 n++;  
 }  
}  
  
void init\_UART0(){  
 SIM\_SCGC5 = SIM\_SCGC5 | (1<<9); //HABILITA portA  
 SIM\_SCGC4 = SIM\_SCGC4 | (1<<10); //HABILITA UART0  
 //set mux(RX,TX)  
 PORTA\_PCR1 = 0x00000200;  
 PORTA\_PCR2 = 0x00000200;  
 //ativando clock da UART0  
 SIM\_SOPT2 = SIM\_SOPT2 | (1<<26);  
 //rate  
 UART0\_BDH = 0x00;  
 UART0\_BDL = 0x44;  
 //recebimento, envio de dados  
 UART0\_C2 = 0x0C;  
 UART0\_C2 = UART0\_C2 | (1<<5);  
 //habilita interrupção da UART0  
 NVIC\_ISER = NVIC\_ISER | (1<<12);  
 // habilita a interrupacao de TX  
 UART0\_C2 = UART0\_C2 | (1<<7);  
} // configura UART 19200, 8 bits, sem paridade, 1 stop 1 start bit e autoriza interrupções  
  
void putchar\_UART0(char c){  
 while(n\_buffer == NMAX) ;  
 UART0\_C2 = 0x02c; //desabilita interrupçao  
 buffer[pi] = c;  
 pi=(pi+1)%NMAX;  
 n\_buffer++;  
 UART0\_C2 = 0x0ac; // habilita a interrupção  
} // imprime o código ASCII guardado no terminal serial  
  
char getchar\_UART0( ) {  
  
 if (RX\_full == 1){  
 RX\_full = 0;  
 return RX\_data;  
 }  
 else {  
 return 0;  
 }  
  
}  
  
void print\_UART0(char \*c){  
 int n = 0;  
 while (c[n] != '\0'){  
 putchar\_UART0(c[n]);  
 n++;  
 }  
  
}  
  
void init\_SysTick\_Handler(void){  
 SYST\_RVR = 0x00032000; ;  
 SYST\_CSR = SYST\_CSR | 0x00000007;  
}  
  
void SysTick\_Handler(void){  
  
 static int key1 = 0, key2 = 0, key3 = 0,cent\_local = 0, a; // mesmo código do anterior só que testa os valores da leitura dos botoes  
 int pta4, pta5, pta12;  
 a = GPIOA\_PDIR; //faz a verificação dos botões  
  
 pta4 = a & (1<<4);  
 pta5 = a & (1<<5);  
 pta12 = a & (1<<12);  
  
 if(rp\_global==0){  
  
 if(zere){//zera  
 cent\_local = 0;  
 zere = 0;  
 }  
  
 if(!pare){  
 cent\_local++;  
 }  
 cent\_global = cent\_local;  
 rp\_global=1;  
 }  
  
 if(!key1 && pta4){  
 key1\_Full = 1;  
 }  
  
 if(!key2 && pta5){  
 key2\_Full = 1;  
 }  
  
 if(!key3 && pta12){  
 key3\_Full = 1;  
 }  
 key1 = pta4; // atualiza o estado das keys  
 key2 = pta5;  
 key3 = pta12;  
  
}  
  
void UART0\_IRQHandler(void){  
  
 if(UART0\_S1 & UART\_S1\_RDRF\_MASK){   
 RX\_data=UART0\_D;   
 RX\_full=1;  
 }  
  
 else { // transmissão pronta para ocorrer  
 UART0\_D = buffer[po]; // limpa o registrador TDRE   
 po=(po+1)%NMAX;   
 n\_buffer--; // reduz o numero de caracteres no buffer  
 if((n\_buffer == 0) || (po - pi == 0)){ // condição de buffer vazio   
 UART0\_C2 = 0x02c; // desabilita a interrupção  
 }  
 }  
  
}  
  
int getkey(){  
  
 if (key1\_Full == 1 ){  
  
 key1\_Full = 0;  
  
 return 1;  
 }  
  
  
 if (key2\_Full == 1 ){  
  
 key2\_Full = 0;  
  
 return 2;  
 }  
  
  
 if (key3\_Full == 1 ){  
  
 key3\_Full = 0;  
  
 return 3;  
 }  
  
 return 0;  
  
}  
  
void init\_LED(){  
 SIM\_SCGC5 |= 0x00000A00; // Habilita PORTA e PORTC  
  
 PORTC\_PCR0 |= 0x00000100; // Configura pin MUX control para PORTC aparecer nos pinos: (001)  
 PORTC\_PCR1 |= 0x00000100;  
 PORTC\_PCR2 |= 0x00000100;  
 PORTC\_PCR3 |= 0x00000100;  
 PORTC\_PCR4 |= 0x00000100;  
 PORTC\_PCR5 |= 0x00000100;  
 PORTC\_PCR6 |= 0x00000100;  
 PORTC\_PCR7 |= 0x00000100;  
 PORTC\_PCR10 |= 0x00000100;  
  
 PORTA\_PCR4 |= 0x00000100; // Configura MUX para PORTA bit 4 aparecer nos pinos (incrementar)  
 PORTA\_PCR5 |= 0x00000100; // Configura MUX para PORTA bit 5 aparecer nos pinos (decrementar)  
  
 GPIOC\_PDDR |= 0x000004FF; // Configura bits 0-10 do PORTC como saídas  
 // LEDs apagados (bits 0-7) em zero  
 // LE (Latch Enable) do registrador 74573 desabilitado (0)  
}