Anexo 1

main.c

```
/*****************************************************************************
*****************/
/*
*/
/*
*/
/*        Identificador_de_CIs.ino                        Author(s): Bismark C.
& Rafael F.      */
/*
*/
/*                                                        Email(s) :
bismarkcotrim@hotmail.com   */
/*
goldcard99@hotmail.com      */
/*
*/
/*                                                        Address  : DF, Brasil,
72444-240       */
/*        Created: 2019/05/21 13:54:29 by rFeijo
*/
/*        Updated: 2019/05/27 13:16:45 by rFeijo
*/
/*                                                                  All Rights
Reserved         */
/*****************************************************************************
*****************/

// Includes
#include <msp430g2553.h>
#include "lcd.h"

#define DAT_PORT    P2
#define D4          BIT4
#define D5          BIT5
#define D6          BIT6
#define D7          BIT7
#define RS_PORT     P2
#define RS          BIT2
#define EN_PORT     P2
#define EN          BIT3

// Variables Declaration
unsigned int Pinos[14] = {BIT0, BIT1, BIT2, BIT7, 0, 0, 0, 0, 0, 0, BIT6, 0,
0, 0};                        // Vector of pins that will be part of the
communication with the IC

unsigned int Tabela[3][14] = {{1,0,1,0,1,0,4,0,1,0,1,0,1,3},
// This is the default table of NOT pins, since it only has one input and one
output
```

```
                              {0,1,2,0,1,2,4,1,2,0,1,2,0,3},
// This is the special NOR table that will be verified

                              {1,2,0,1,2,0,4,0,1,2,0,1,2,3}};

unsigned int Saidas[3][6] = {{0},{0},{0}};

unsigned int t, m, k, i, q, j;

// Delay Function
/*
void delay(volatile unsigned int a)
{
    TACCR0 = 1000 -1;               // 1MHz / 1000 = 1KHz(1 ms)
    TACTL |= TACLR;                 // Clear counter
    TACTL = TASSEL_2 + ID_0 + MC_1; // MCLK + 1MHz/1 + UpMode

    while(a--)
    {
        while((TACTL & TAIFG) == 0)
           ;
        TACTL &= ~TAIFG;
    }
    TACTL = MC_0;
}
*/

// NOT Function
void NOT()
{
    if (Saidas[0][0] == 1)
    {
      if (Saidas[1][0] == 0)
      {
        q = 4;
      }

      else
      {
        q = 10;
      }
    }

    else
    {
      q = 10;
    }
}

// NOR Function
void NOR()
{
    if (Saidas[0][0] == 1)
    {
      if (Saidas[1][0] == 0)
      {
        if(Saidas[2][0] == 0)
        {
          q = 2;
        }

        else
        {
          q = 10;
        }
      }

      else
      {
        q = 10;
      }
    }

    else
    {
      q = 10;
    }
}

// Analyzes for other CI's, other than NOT and NOR
void Analise()
{
  if (Saidas[0][0] == 1)
  {
    if(Saidas[1][0] == 0)
    {

      if(Saidas[2][0] == 1)
      {
        q = 0;
      }

      else
      {
        q = 10;
      }
    }

    else if(Saidas[1][0] == 1)
    {
      if(Saidas[2][0] == 0)
      {
        q = 266;
      }

      else
      {
        q = 10;
      }
    }

    else
    {
      q = 10;
    }
  }
  else if(Saidas[0][0] == 0)
  {
    if(Saidas[1][0] == 1)
    {
      if(Saidas[2][0] == 0)
      {
        q = 8;
      }

      else if(Saidas[2][0] == 1)
      {
        q = 32;
      }

      else
      {
        q = 10;
      }
    }

    else if(Saidas[1][0] == 0)
    {
      if(Saidas[2][0] == 1)
      {
        q = 86;
      }

      else
      {
        q = 10;
      }
    }

    else
    {
      q = 10;
    }
  }

  else
  {
    q = 10;
  }
}

// Function to test inputs low
void UMTeste()
{
  m = 0;
  int y;
  t = 0;

  for(y=0; y<14; y++)
  {
    if(((Tabela[k][y])== 1)||(Tabela[k][y])== 2)
    {
      P1OUT &= ~Pinos[y];
    }
  }

  delay(500);
  for(y=0; y<14; y++)
  {
    if((Tabela[k][y])== 0)
    {
      if ((P1IN & (Pinos[y])) == Pinos[y])
      {
          Saidas[m][t] = 1;
      }

      else
      {
          Saidas[m][t] = 0;
      }

      t++;
    }
  }
}

// Function to test high inputs
void DOISTeste()
{
  m++;
  int x;
  t = 0;

  for(x=0; x<14; x++)
  {
    if(((Tabela[k][x])== 1)||(Tabela[k][x])== 2)
    {
      P1OUT |= Pinos[x];
    }
  }

  delay(500);

  for(x=0; x<14; x++)
  {
    if((Tabela[k][x])== 0)
    {
      if ((P1IN & (Pinos[x])) == Pinos[x])
      {
          Saidas[m][t] = 1;
      }

      else
      {
          Saidas[m][t] = 0;
      }

      t++;
    }
  }
}

// Function to test one input on high and the other on low
void TRESTeste()
{
  m++;
  int z;
  t = 0;

  for(z=0; z<14; z++)
  {
    if((Tabela[k][z])== 1)
    {
      P1OUT |= Pinos[z];
    }
    else if ((Tabela[k][z])== 2)
    {
      P1OUT &= ~(Pinos[z]);
    }
  }

  delay(500);

  for(z=0; z<14; z++)
  {
    if((Tabela[k][z])== 0)
    {
      if ((P1IN & (Pinos[z])) == Pinos[z])
      {
          Saidas[m][t] = 1;
      }

      else
      {
          Saidas[m][t] = 0;
      }

      t++;
    }
  }
}

// Result of the analyzes
void Resultado()
{
    for(k=0; k<3; k++)
    {
        lcd_setCursor(0,3);
        lcd_print("Analisando");
        lcd_setCursor(1,k);
        lcd_print(".");

      for(j=0; j<14; j++)
      {
        if((Tabela[k][j])== 4)
// If 4 appears in the Table[][], then it will be Low, GND
        {
          //pinMode(Pinos[j],OUTPUT);
          //digitalWrite(Pinos[j], LOW);
        }
        else if((Tabela[k][j])== 3)
// If 3 appears in the Table[][], then it will be High, VCC
        {
          //pinMode(Pinos[j],OUTPUT);
```

```
     //digitalWrite(Pinos[j], HIGH);
    }
    else if((Tabela[k][j])== 0)
// If 0 appears in the Table[][], then it will be IC Output
    {
     P1OUT &= ~(Pinos[j]);
     P1DIR &= ~(Pinos[j]);
    }
    else if(((Tabela[k][j])== 1)||((Tabela[k][j])== 2))
// If 1 or 2 appears in the Table[][], then it will be IC Input
    {
     P1DIR |=  (Pinos[j]);
     P1OUT &= ~(Pinos[j]);
    }
   }

   delay(50);

   UMTeste();

   delay(50);

   DOISTeste();

   delay(50);

   TRESTeste();

   delay(50);

   if (k == 0)
   {
    NOT();
   }

   else if(k == 1)
   {
    NOR();
   }

   else if (k == 2)
   {
    Analise();
   }

   if(q != 10)
   {
      if(q == 0)
      {
        lcd_clear();
        lcd_setCursor(0,6);
        lcd_print("NAND2");
        lcd_setCursor(1,5);
        lcd_print("7400");
        break;

      }
      else if(q == 2)
      {
        lcd_clear();
        lcd_setCursor(0,6);
        lcd_print("NOR2");
        lcd_setCursor(1,5);
        lcd_print("7402");
        break;
      }
      else if(q == 4)
      {
        lcd_clear();
        lcd_setCursor(0,6);
        lcd_print("NOT1");
        lcd_setCursor(1,5);
        lcd_print("7404");
        break;
      }
      else if(q == 8)
      {
        lcd_clear();
        lcd_setCursor(0,6);
        lcd_print("AND2");
        lcd_setCursor(1,5);
        lcd_print("7408");
        break;
      }
      else if(q == 32)
      {
        lcd_clear();
        lcd_setCursor(0,6);
        lcd_print("OR2");
        lcd_setCursor(1,5);
        lcd_print("7432");
        break;
      }
      else if(q == 86)
      {
        lcd_clear();
        lcd_setCursor(0,6);
        lcd_print("XOR2");
        lcd_setCursor(1,5);
        lcd_print("7486");
        break;
      }
      else if(q == 266)
      {
        lcd_clear();
        lcd_setCursor(0,6);
        lcd_print("XNOR2");
        lcd_setCursor(1,5);
        lcd_print("74233");
        break;
      }
      break;
   }

   else if((q == 10) && (k == 2))
   {
      lcd_clear();
      lcd_setCursor(0,6);
```

```
            lcd_print("ERRO");
            lcd_setCursor(1,1);
            lcd_print("CI Nao Achado");
            break;
      }
    }
}

// Setup Function
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;       // stop watchdog timer

    P1DIR &= ~BIT3;                 // P1.3 Input
    P1OUT |= BIT3;                  // P1.3 High
    P1REN |= BIT3;                  // P1.3 Pullup
    P1IE  |= BIT3;                  // P1.3 Interrupção habilitada
    P1IES |= BIT3;                  // P1.3 Pega a borda de High para Low
    P1IFG &= ~BIT3;                 // P1.3 Flag de interrupção limpa

    P1DIR |= BIT0;
    P1OUT |= BIT0;

// lcd_init(data_port, d4, d5, d6, d7, rs_port, rs, en_port, en)
    lcd_init(DAT_PORT, D4, D5, D6, D7, RS_PORT, RS, EN_PORT, EN);

    while(1)
    {
        lcd_clear();
        lcd_setCursor(0,0);
        lcd_print("Comecar o Teste");
        lcd_setCursor(1,0);
        lcd_print("Aperte o Botao");

        _BIS_SR(LPM4_bits + GIE);   // Sleep Mode Active

        lcd_clear();

        Resultado();

        P1OUT &= ~BIT0;
        P1OUT |= BIT6;

        P1DIR |= BIT0 + BIT6;
        P1DIR &= ~BIT3;

        delay(8000);

        P1OUT &= ~BIT6;
        P1OUT |= BIT0 + BIT3;

        P1DIR |= BIT0 + BIT6;
        P1DIR &= ~BIT3;
    }
}
```

Sleep_isr.asm

```
;;  Codigo Assembly da rotina de Sleep



    .cdecls C,NOLIST, "msp430.h"      ;; Processor specific definitions


    .global    Sleep_isr                  ;; Declare symbol to be exported
    .sect ".text:_isr"                ;; Code is relocatable
;-------------------------------------------------------------------------------
;===============================================================================
; Port1_isr      Port 1 Interrupt service
;===============================================================================

Sleep_isr

            mov.w    #0x00, P1IFG                  ; Limpa a flag de interrupção no P1.3
            bic.w    #LPM4, 0(SP)                  ; Sai do modo LPM4
            mov.w    #0x00, P1OUT
            mov.w    #BIT0 + BIT6, P1DIR
            reti                ; Retorna para a linha depois da habilitação do LPM4

  .if ($defined(__MSP430_HAS_MSP430XV2_CPU__) | $defined(__MSP430_HAS_MSP430X_CPU__))
      reta
  .else
      ret
  .endif
;===============================================================================
;            Interrupt Vectors
;===============================================================================
          .sect      PORT1_VECTOR    ; PORT1 Vector
          .word      Sleep_isr


      .end
```

lcd.c

```
#include <msp430.h>
#include "lcd.h"

// Global definitions for port & pin selection
const uint16_t ports[] = { (uint16_t) &P1OUT, (uint16_t) &P2OUT, (uint16_t) &P3OUT};
const uint16_t dirs[] = { (uint16_t) &P1DIR, (uint16_t) &P2DIR, (uint16_t) &P3DIR};
const uint16_t pins[] = {BIT0, BIT1, BIT2, BIT3, BIT4, BIT5, BIT6, BIT7};

uint16_t lcdPins[4], rsPin, enPin;
uint8_t lcdPort, rsPort, enPort;

#define pout(P)     ( (volatile uint8_t *)( ports[P] ) )
#define pdir(P)     ( (volatile uint8_t *)( dirs[P] ) )

// Delay function for producing delay in 0.1 ms increments
void delay(volatile unsigned int a)
{
    TACCR0 = 1000 -1;              // 1MHz / 1000 = 1KHz(1 ms)
    TACTL |= TACLR;                // Clear counter
    TACTL = TASSEL_2 + ID_0 + MC_1; // MCLK + 1MHz/1 + UpMode
    while(a--)
    {
        while((TACTL & TAIFG) == 0)
          ;
        TACTL &= ~TAIFG;
    }
    TACTL = MC_0;
}
// Function to pulse EN pin after data is written
void pulseEN(void)
{
    volatile uint8_t *enout;
    enout = pout(enPort);
    *enout |= enPin;
    delay(1);
    *enout &= ~enPin;
    delay(1);
}
// Fuction to write 4 bits of data to D4-D7 pins
void write4bits(uint8_t value)
{
    volatile uint8_t *datout;
    datout = pout(lcdPort);
    uint8_t i;
    for(i = 0; i < 4; i++)
    {
        if(value & 0x01)
            *datout |= lcdPins[i];
        else
            *datout &= ~lcdPins[i];
        value = value >> 1;
    }
}
//Function to write data/command to LCD
void lcd_write(uint8_t value, uint8_t mode)
{
    volatile uint8_t *rsout;
    rsout = pout(rsPort);

    if(mode == CMD)
        *rsout &= ~rsPin;              // Set RS -> LOW for Command mode
    else
        *rsout |= rsPin;               // Set RS -> HIGH for Data mode

    write4bits(value>>4);              // Write high nibble first
    pulseEN();
    delay(1);

    write4bits(value&0x0F);            // Write low nibble next
    pulseEN();
    delay(1);
}
// Function to print a string on LCD
void lcd_print(char *s)
{
    while(*s)
    {
        lcd_write(*s, DATA);
        s++;
    }
}
// Function to move cursor to desired position on LCD
void lcd_setCursor(uint8_t row, uint8_t col)
{
    const uint8_t row_offsets[] = { 0x00, 0x40, 0x14, 0x54 };
    lcd_write(LCD_SETDDRAMADDR | (col + row_offsets[row]), CMD);
    delay(1);
}

// Initialize LCD - Specify Port Number, Pin Number of D4, D5, D6, D7, RS and EN
void lcd_init(uint8_t dat_port, uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7, uint8_t rs_port, uint8_t rs, uint8_t
en_port, uint8_t en)
{
#if defined(EASY_MODE)
    lcdPins[0] = pins[d4];
    lcdPins[1] = pins[d5];
    lcdPins[2] = pins[d6];
    lcdPins[3] = pins[d7];

    rsPin = pins[rs];
    enPin = pins[en];
#else
    lcdPins[0] = d4;
    lcdPins[1] = d5;
    lcdPins[2] = d6;
    lcdPins[3] = d7;
    rsPin = rs;
    enPin = en;
#endif

    // Set SEL bits to GPIO mode for P2.6 & P2.7
    if(dat_port == 2)
        P2SEL &= ~(lcdPins[0] + lcdPins[1] + lcdPins[2] + lcdPins[3]);
    if(rs_port == 2)
        P2SEL &= ~rsPin;
    if(en_port == 2)
        P2SEL &= ~enPin;

    lcdPort = dat_port-1;
    rsPort = rs_port-1;
    enPort = en_port-1;

    volatile uint8_t *datdir;
    volatile uint8_t *rsdir;
    volatile uint8_t *endir;
    volatile uint8_t *datout;
    volatile uint8_t *rsout;
    volatile uint8_t *enout;

    datdir = pdir(lcdPort);
    rsdir = pdir(rsPort);
    endir = pdir(enPort);

    datout = pout(lcdPort);
    rsout = pout(rsPort);
    enout = pout(enPort);

    *datdir |= (lcdPins[0] + lcdPins[1] + lcdPins[2] + lcdPins[3]);
    *rsdir |= rsPin;
    *endir |= enPin;
    *datout &= ~(d4+d5+d6+d7);
    *rsout |= ~rsPin;
    *enout |= ~enPin;

    const char lcdMode = LCD_4BITMODE + LCD_2LINE + LCD_5x8DOTS;
    const char dispMode = LCD_DISPLAYON + LCD_CURSORON + LCD_BLINKON;

    delay(150);                              // Wait for power up ( 15ms )
    lcd_write(0x33, CMD);                    // Initialization Sequence 1
    delay(50);                               // Wait ( 4.1 ms )
    lcd_write(0x32, CMD);                    // Initialization Sequence 2
    delay(1);                                // Wait ( 100 us )

    // All subsequent commands take 40 us to execute, except clear & cursor return (1.64 ms)

    lcd_write(LCD_FUNCTIONSET | lcdMode, CMD);        // Set LCD mode
    delay(1);

    lcd_write(LCD_DISPLAYCONTROL | dispMode, CMD);    // Display on Cursor on
    delay(1);

    lcd_write(LCD_CLEARDISPLAY, CMD);                 // Clear screen
    delay(20);

    lcd_write(LCD_ENTRYMODESET | LCD_ENTRYLEFT, CMD);   // Auto Increment Cursor
    delay(1);

    lcd_setCursor(0,0);                       // Goto Row 1 Column 1
}

void lcd_clear(void)
{
    lcd_write(LCD_CLEARDISPLAY, CMD);                 // Clear screen
    delay(20);
    lcd_setCursor(0,0);
}
```

lcd.h

```
#ifndef LCD_H_
#define LCD_H_

#include <inttypes.h>

// commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00
```

```
// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

#define P1      1
#define P2      2
#define P3      3

#define CMD     0
#define DATA    1

void lcd_init(uint8_t, uint8_t, uint8_t, uint8_t, uint8_t, uint8_t, uint8_t, uint8_t, uint8_t);
void lcd_setCursor(uint8_t, uint8_t);
void lcd_print(char *);
void lcd_write(uint8_t, uint8_t);
void write4bits(uint8_t);
void pulseEN(void);
void delay(volatile unsigned int a);
void lcd_clear(void);

#endif /* LCD_H_ */
```