Proiect Structura si Organizarea Calculatoarelor.
Interactiunea C / Assembly.

Petrica Petru 1306B

Scopul proiectului:

Crearea unei aplicatii pentru a demonstra interactiunea limbajului C cu Assembly prin apelul functiilor scrise in Assembly in interiorul codului C.

In practica, in cadrul sistemelor de performanta inalta si in special in sistemele embedded, in anumite situatii limbajul C nu ofera suficient control asupra resurselor hardware. Alteori compilatorul de C nu ofera nivelul de optimizare de care e nevoie si care se poate atinge scriind codul in asamblare. De asemenea, in cadrul proiectelor embedded nu este atat de evident avantajul limbajui C de a fi portabil din motiv ca codul, in general este specializat pentru o anumita platforma.

In cadrul acestui proiect mi-am propus sa realizez o aplicatie care sa demonstreze apelul unei functii scrise in asamblare din limbajul C. In cadrul acestui proiect voi lucra cu placa Nano130KE3BN furnizata de compania Nuvoton. Aceasta placuta dispune de un LCD display. Mi-am propus sa afisez un sir de caractere pe acest display.

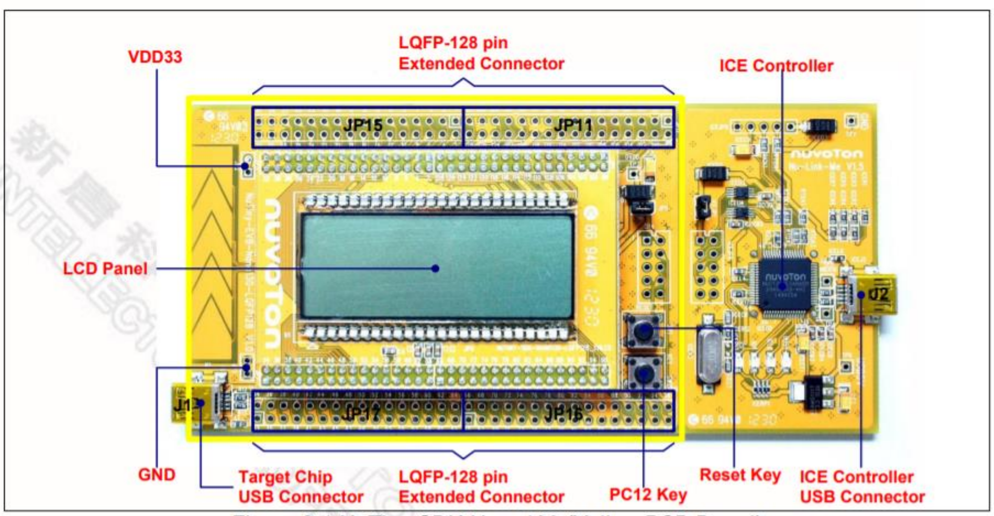
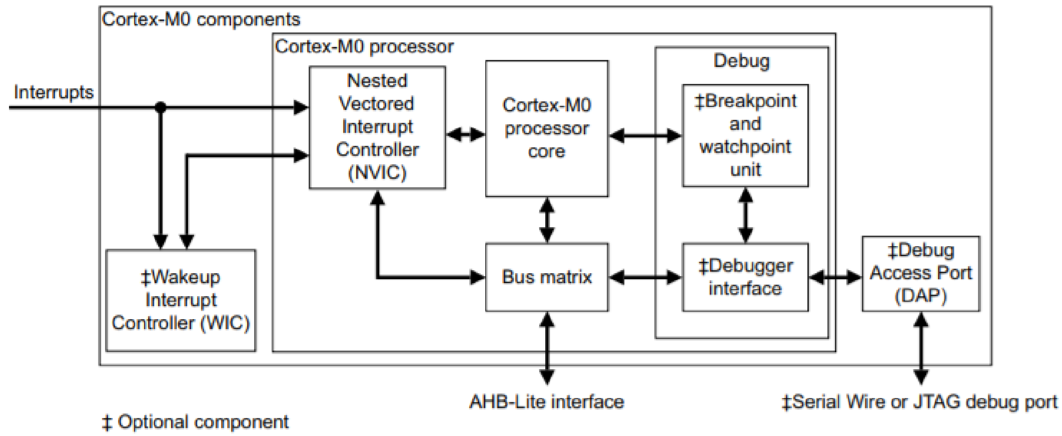


Figure 2-1 NuTiny-SDK-Nano130 (Yellow PCB Board)

Aceasta dispune de un procesor Cortex-M0, ce implementeaza specificatia ARMv6-m, arhitectura de tip Von-Neumann.

Schiema procesorului Cortex-M0.



Instrumente puse la dispozitie de compania Nuvoton.

Pentru a incarca programul in memorie si a-l executa, voi folosi driverul pentru USB NuLink, acesta ofera si un set de instrumente pentru depanarea soft-ului.

NuTool PinView

NuTool_PinView ne ofera posibilitatea de a vizualiza configuratiile pinilor de pe placuta in timp real, astfel putem usor sa ne dam seama daca nu au fost activate functiile alternative ale portului, I/O pinii nu sunt configurati corect, etc.

Fisierele header puse la dispozitie de furnizor.

Nuvoton pune la dispozitia programatorilor un set de fisiere header si surse C ce contin structuri mapate in memorie peste adresele unde sunt mapate componentele hardware, de asemenea in aceste fisiere este continuta documentatia pentru fiecare structura / registru / pin sub forma unor comentarii. Voi folosi aceste fisiere header in scopul de a evita scrierea acestor structuri, care sunt doar transpuse din datasheet, fiecare registru aferent unei componente este definit in aceastra structura prin intermediul unui volatile unsigned int si se mapeaza direct peste adresa sa reala ca memory-mapped I/O.

```c
#define WDT_BASE            (APB1PERIPH_BASE + 0x04000)    ///< WDT register base address
#define WWDT_BASE           (APB1PERIPH_BASE + 0x04100)    ///< WWDT register base address
#define RTC_BASE            (APB1PERIPH_BASE + 0x08000)    ///< RTC register base address
#define TIMER0_BASE         (APB1PERIPH_BASE + 0x10000)    ///< TIMER0 register base address
#define TIMER1_BASE         (APB1PERIPH_BASE + 0x10100)    ///< TIMER1 register base address
#define I2C0_BASE           (APB1PERIPH_BASE + 0x20000)    ///< I2C0 register base address
#define SPI0_BASE           (APB1PERIPH_BASE + 0x30000)    ///< SPI0 register base address
#define PWM0_BASE           (APB1PERIPH_BASE + 0x40000)    ///< PWM0 register base address
#define UART0_BASE          (APB1PERIPH_BASE + 0x50000)    ///< UART0 register base address
#define DAC_BASE            (APB1PERIPH_BASE + 0xA0000)    ///< DAC register base address
#define LCD_BASE            (APB1PERIPH_BASE + 0xB0000)    ///< LCD register base address
#define SPI2_BASE           (APB1PERIPH_BASE + 0xD0000)    ///< SPI2 register base address
#define ADC_BASE            (APB1PERIPH_BASE + 0xE0000)    ///< ADC register base address
```

```c
#define WDT                 ((WDT_T *) WDT_BASE)           ///< Pointer to WDT register structure
#define WWDT                ((WWDT_T *) WWDT_BASE)         ///< Pointer to WWDT register structure
#define RTC                 ((RTC_T *) RTC_BASE)           ///< Pointer to RTC register structure
#define TIMER0              ((TIMER_T *) TIMER0_BASE)      ///< Pointer to TIMER0 register structure
#define TIMER1              ((TIMER_T *) TIMER1_BASE)      ///< Pointer to TIMER1 register structure
#define TIMER2              ((TIMER_T *) TIMER2_BASE)      ///< Pointer to TIMER2 register structure
#define TIMER3              ((TIMER_T *) TIMER3_BASE)      ///< Pointer to TIMER3 register structure
#define SHADOW              ((SHADOW_T *) SHADOW_BASE)     ///< Pointer to GPIO shadow register structure
#define I2C0                ((I2C_T *) I2C0_BASE)          ///< Pointer to I2C0 register structure
#define I2C1                ((I2C_T *) I2C1_BASE)          ///< Pointer to I2C1 register structure
#define SPI0                ((SPI_T *) SPI0_BASE)          ///< Pointer to SPI0 register structure
#define SPI1                ((SPI_T *) SPI1_BASE)          ///< Pointer to SPI1 register structure
#define SPI2                ((SPI_T *) SPI2_BASE)          ///< Pointer to SPI2 register structure
#define PWM0                ((PWM_T *) PWM0_BASE)          ///< Pointer to PWM0 register structure
#define PWM1                ((PWM_T *) PWM1_BASE)          ///< Pointer to PWM1 register structure
#define UART0               ((UART_T *) UART0_BASE)        ///< Pointer to UART0 register structure
#define UART1               ((UART_T *) UART1_BASE)        ///< Pointer to UART1 register structure
#define LCD                 ((LCD_T *) LCD_BASE)           ///< Pointer to LCD register structure
#define ADC                 ((ADC_T *) ADC_BASE)           ///< Pointer to ADC register structure
#define SC0                 ((SC_T *) SC0_BASE)            ///< Pointer to SC0 register structure
#define SC1                 ((SC_T *) SC1_BASE)            ///< Pointer to SC1 register structure
#define SC2                 ((SC_T *) SC2_BASE)            ///< Pointer to SC2 register structure
#define USBD                ((USBD_T *) USBD_BASE)         ///< Pointer to USBD register structure
#define I2S                 ((I2S_T *) I2S_BASE)           ///< Pointer to I2S register structure
#define DAC                 ((DAC_T *) DAC_BASE)           ///< Pointer to DAC register structure
```

```c
typedef struct
{


    /**
     * CTRL
     * ===================================================================================================
     * Offset: 0x00  I2S Control Register
     * ---------------------------------------------------------------------------------------------------
     * |Bits    |Field     |Descriptions
     * | :----: | :----:   | :---- |
     * |[0]     |I2SEN     |I2S Controller Enable
     * |        |          |0 = Disabled.
     * |        |          |1 = Enabled.
     * |[1]     |TXEN      |Transmit Enable
     * |        |          |0 = Data transmitting Disabled.
     * |        |          |1 = Data transmitting Enabled.
     * |[2]     |RXEN      |Receive Enable
     * |        |          |0 = Data receiving Disabled.
     * |        |          |1 = Data receiving Enabled.
     * |[3]     |MUTE      |Transmitting Mute Enable
     * |        |          |0 = Transmit data in buffer to channel.
     * |        |          |1 = Transmit '0' to channel.
     * |[5:4]   |WORDWIDTH |Word Width
```

```c

    /**
     * FCSTS
     * ===================================================================================================
     * Offset: 0x34  LCD frame counter status
     * ---------------------------------------------------------------------------------------------------
     * |Bits    |Field     |Descriptions
     * | :----: | :----:   | :---- |
     * |[0]     |FCSTS     |LCD Frame Counter Status
     * |        |          |0 = Frame counter value does not reach FCV (Frame Count TOP value).
     * |        |          |1 = Frame counter value reaches FCV (Frame Count TOP value).
     * |        |          |If the FCINTEN is s enabled, the frame counter overflow Interrupt is generated.
     * |[1]     |PDSTS     |Power-Down Interrupt Status
     * |        |          |0 = Inform system manager that LCD controller is not ready to enter power-down state
     * |        |          |1 = Inform system manager that LCD controller is ready to enter power-down state if
    */
    __IO uint32_t FCSTS;

} LCD_T;
```

Setarea ceas-ului pentru LCD

Pentru a afisa ceva pe componenta de LCD, este nevoie in primul rand de configurat sursa de clock pentru display, pentru asta vom scrie in registrul CLKSEL1, bit-ul aferent cristalului de frecventa joasa. De asemenea trebuie de activa linia clock-ului pentru LCD din cadrul APB-ului (Advanced Peripheral Bus) de pe placuta.

```c
/* Clock source from external 12 MHz or 32 KHz crystal clock */
CLK->CLKSEL1 &= ~CLK_CLKSEL1_LCD_S_Msk;
CLK->CLKSEL1 |= (0x0 << CLK_CLKSEL1_LCD_S_LXT);

/* Enable clock on APB */
CLK->APBCLK |= CLK_APBCLK_LCD_EN;
```

Acesti registri sunt protejati, pentru scrierea in ei, este nevoie de a-i debloca, pentru aceasta este nevoie scrierea registrului SYS->RegLockAddr cu anumite constante.

```
/* Unlock protected registers */
while(SYS->RegLockAddr != SYS_RegLockAddr_RegUnLock_Msk) {
    SYS->RegLockAddr = 0x59;
    SYS->RegLockAddr = 0x16;
    SYS->RegLockAddr = 0x88;
}
```

Configurarea pinilor.

De asemeneaeste necesara configurarea pinilor aferenti pentru controlul LCD-ului, anume pinii vor reprezenta segmentele si com-urile, controlul segmentelor de pe display este prin intermediul I/O pinilor, care au functie alternativa SEG, acestia sunt multiplexati prin intermediul bitilor COM 3 – 0 pentru a permite controlul display-ului folosind cat mai putini pini.

```
/* Select LCD COMs, SEGs, V1 ~ V3, DH1, DH2 */
SYS->PA_L_MFP |= 0x77770000;    /* seg 36 ~ 39 */
SYS->PA_H_MFP |= 0x7777;        /* seg 20 ~ 23 */
SYS->PB_L_MFP = 0x77777777;     /* seg 10 ~ 13, 4 ~ 7 */
SYS->PB_H_MFP = 0x77777777;     /* LCD V1 ~ V3, seg 30 ~ 31, 24 ~ 26 */
SYS->PC_L_MFP |= 0x777777;      /* LCD COM3 ~ COM0, DH1/DH2 */
SYS->PC_H_MFP |= 0x77000000;    /* seg 32 ~ 33 */
SYS->PD_L_MFP |= 0x77770000;    /* seg 2 ~ 3, 34 ~ 35 */
SYS->PD_H_MFP = 0x77777777;     /* seg 0 ~ 1, 14 ~ 19 */
SYS->PE_L_MFP |= 0x70000000;    /* seg 8 */
SYS->PE_H_MFP |= 0x77700007;    /* seg 9, 27 ~ 29 */
```

Curatarea display-ului.

In continuare, in exemplele Demo, a fost recomandat sa resetam LCD-ul, dupa care sa scriem in toti bitii aferenti segmentelor valoarea 0 pentru a curata display-ul, dupa care asteptam un numar de cicli pentru ca LCD-ul sa se reseteze.

```
SYS->IPRST_CTL2 |= SYS_IPRST_CTL2_LCD_RST_Msk;
SYS->IPRST_CTL2 &= ~SYS_IPRST_CTL2_LCD_RST_Msk;

/* Enable LCD */
LCD->CTL &= ~LCD_CTL_EN_Msk;

/* Turn everything off */
LCD->MEM_0 = 0;
LCD->MEM_1 = 0;
LCD->MEM_2 = 0;
LCD->MEM_3 = 0;
LCD->MEM_4 = 0;
LCD->MEM_5 = 0;
LCD->MEM_6 = 0;
LCD->MEM_7 = 0;
LCD->MEM_8 = 0;
```

Introducerea unui delay pentru sincronizarea operatiilor.

Este recomandata asteptarea unui interval de timp scurt inainte de a continua operatiile cu display-ul. Pentru delay vom folosi SysTick, clock prezent in cadrul procesoarelor ce implementeaza ARM-v6m.

```
/* Wait a little bit for the values to sink */
SysTick->LOAD = 300 * CyclesPerUs;

SysTick->VAL  =  (0x00);
/* Set the clock source to internal and enable ARM SysTick */
SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk;

/* Waiting for down-count to zero */
while((SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk) == 0);
```

Configurare LCD.

In continuare urmeaza configurarea charge pump-ului pentru LCD, acesta permite aplicarea unui voltaj mai mare decat Vcc pentru o definire mai pronuntata a segmentelor pe LCD, dupa care activam LCD-ul setand bitul de enable din cadrul registrului CTL.

```
/* Waiting for down-count to zero */
while((SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk) == 0);

/* Configure LCD bias and enable charge pump */
// set internal source for charge pump
LCD->DISPCTL &= ~LCD_DISPCTL_BV_SEL_Msk;
// reset charge pump frequency to system clock
LCD->DISPCTL = LCD->DISPCTL & ~LCD_DISPCTL_CPUMP_FREQ_Msk;
// set churge pump voltage level to 3V
LCD->DISPCTL = LCD->DISPCTL & ~LCD_DISPCTL_CPUMP_VOL_SET_Msk | LCD_CPVOl_3V;
// disable bias reference ladder
LCD->DISPCTL &= ~LCD_DISPCTL_IBRL_EN_Msk;
// enable charge pump
LCD->DISPCTL |= LCD_DISPCTL_CPUMP_EN_Msk;

// Reset frame rate for LCD
LCD->CTL &= ~LCD_CTL_FREQ_Msk;
// Set the desired frame rate
LCD->CTL |= LCD_FREQ_DIV64;

// Set LCD mux according to the number of COMS we have
// In this case 4 COMS -> 1/4 duty
LCD->CTL = (LCD->CTL & ~LCD_CTL_MUX_Msk) | (3 << LCD_CTL_MUX_Pos);

// Set bias level to 1 / 3 bias
LCD->DISPCTL = LCD->DISPCTL & ~LCD_DISPCTL_BIAS_SEL_Msk | LCD_BIAS_THIRD;

/* Enable LCD */
LCD->CTL |= LCD_CTL_EN_Msk;
```

Afisarea caracterelor pe LCD.

Pentru afisarea mai usoara a caracterelor pe display, dispunem de un sir de definitii a com-urilor si seg-urilor ce trebuie selectate pentru a afisa un anumit caracter.

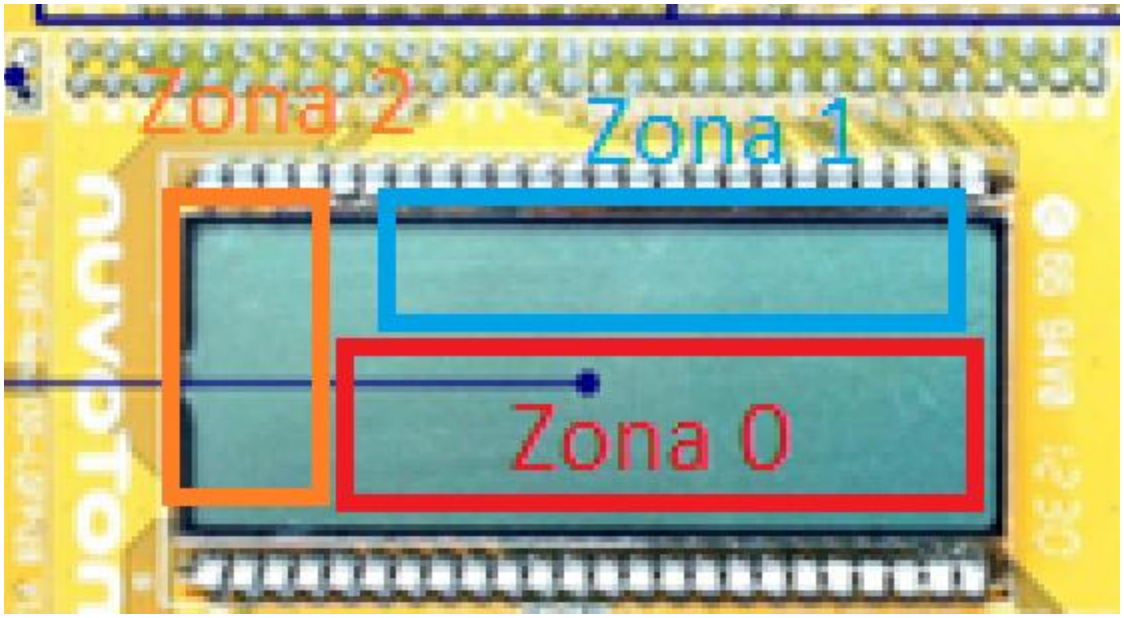
```
/****************************************************************************//**
 *
 * Defines each text's segment (alphabet+numeric) in terms of COM and BIT numbers,
 * Using this way that text segment can be consisted of each bit in the
 * following bit pattern:
 * @illustration
 *                A
 *          ----------
 *          |\    |J  /|
 *         F| H   |   K |B
 *          |   \ | /   |
 *          --G-- --M--
 *          |    /| \   |
 *        E |   Q |   N |C
 *          | /   |P   \|
 *          -----------
 *              D
 *
 *          -------0------
 *
 *         |    \7  |8  /9 |
 *         |5    \  |   /    |1
 *
 *          --6---   ---10--
 *
 *         |     /  |   \11 |
 *         |4   /13 |12 \    |2
 *
 *          -------3------
 *
 *
 *
 ****************************************************************************/
```

```
const char Zone0[sub_Zone0][Zone0_Digit_SegNum][2] =
{
    {
        // 1
        //{com, seg}
        // A      // B      // C      // D
        {3,  0}, {2,  0}, {1,  0}, {0,  0},
        // E      // F      // G      // H
        {1, 38}, {2, 38}, {2, 39}, {3, 39},
        // J      // K      // M      // N
        {3,  1}, {2,  1}, {1,  1}, {0,  1},
        // P      // Q
        {0, 39}, {1, 39},
    },
    {
        // 2
        //{com, seg}
        // A      // B      // C      // D
        {3,  4}, {2,  4}, {1,  4}, {0,  4},
        // E      // F      // G      // H
        {1,  2}, {2,  2}, {2,  3}, {3,  3},
        // J      // K      // M      // N
        {3,  5}, {2,  5}, {1,  5}, {0, 5},
        // P      // Q
        {0,  3}, {1,  3},
    },
    {
        // 3
        //{com, seg}
```

```
const uint16_t Zone0_TextDisplay[] =
{
    0x0000, /* space */
    0x1100, /* ! */
    0x0280, /* " */
    0x0000, /* # */
    0x0000, /* $ */
    0x0000, /* % */
    0x0000, /* & */
    0x0000, /* £  */
    0x0039, /* ( */
    0x000f, /* ) */
    0x3fc0, /* * */
    0x1540, /* + */
    0x0000, /* , */
    0x0440, /* - */
    0x8000, /* . */
    0x2200, /* / */

    0x003f, /* 0 */
    0x0006, /* 1 */
    0x045b, /* 2 */
    0x044f, /* 3 */
    0x0466, /* 4 */
    0x046d, /* 5 */
    0x047d, /* 6 */
    0x0007, /* 7 */
    0x047f, /* 8 */
    0x046f, /* 9 */
```

Zonele de display

LCD-ul este impartit in 3 zone, in fiecare zona pot fi afisate anumite caractere / segmente, eu voi afisa text in zona 0.



Functia pentru afisarea sirului de caractere pe LCD.

Urmatoarea functie primeste un sir de caractere si citeste pentru fiecare caracter din structurile o masca aferenta segmentelor care trebuie activate si seteaza com-ul si seg-ul respectiv. Aceasta functie apeleaza functia LCD_SetPixel, pe care o voi implementa in Assembly.

```c
void LCD_PrintString(uint32_t  u32Zone, char *string)
{
    /*
        All the LCD segments are turned on by setting the corresponding COM
        and the corresponding segment.

        The segment data is divided into 3 zones, each one of them having
        multiple subzones.

        The subzones contain multiple segment data structs which contain
        the corresponding com number and segment number
    */
    int      data, length, index;
    uint16_t bitfield;
    uint32_t com, bit;
    const char *segment_offset;

    int      i;

    length = strlen(string);
    index  = 0;

    /* LCD_ZoneInfo is an array containing pointers to the starts of the 3 zones */

    /* Sub_Zone_Num is the number of characters which can be displayed in that zone */
    for (index = 0; index < LCD_ZoneInfo[u32Zone].Sub_Zone_Num; index++)
    {
        if (index < length)
        {
            data = (int) *string;
        }
        else
        {
            /* If the string's length is less than the number of the characters
               which can be displayed, we pad it with ' ' */
            data = 0x20; /* SPACE */
        }
        /* defined letters currently start at "SPACE" - 0x20; */
        data = data - 0x20;

        /* Zone_TextDisplay contains bitmasks which represent which bits must
           be turned on in order to display a particular character.
           To get the specific bitfield we jump to the beginning + data */
        bitfield = *(Zone_TextDisplay[u32Zone] + data);

        /* We go through all the seg data and set only turn on the segments
           specified by bitfield */
        for (i = 0; i < LCD_ZoneInfo[u32Zone].Zone_Digit_SegNum; i++)
        {
            /* Zone[u32Zone] -> the pointer to the start of the zone
               index * LCD_ZoneInfo[u32Zone].Zone_Digit_SegNum * 2-> offset to the start
               of the data for the specific segnum
               i * 2 -> offset to the current seg / com pair */
            segment_offset = Zone[u32Zone]
                    + index * LCD_ZoneInfo[u32Zone].Zone_Digit_SegNum * 2
                    + i * 2;
            bit = *(segment_offset + 1);

            com = *segment_offset;

            /* Check if this segment should be turned on */
            if (bitfield & (1 << i))
            {
                LCD_SetPixel(com, bit, 1);
            } else {
                LCD_SetPixel(com, bit, 0);
            }
        }
        string++;
    }
}
```

Declararea functie LCD_SetPixel in fisierul sursa C.

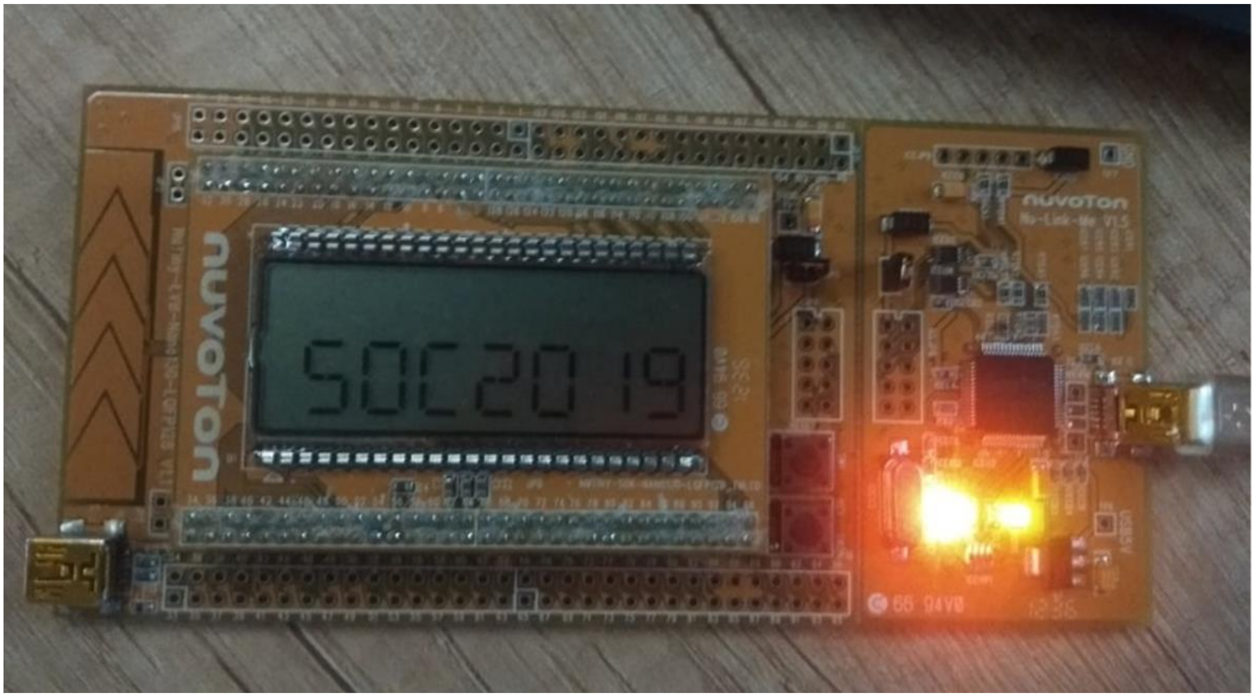
Pentru a putea apela in limbajul C o functie, aceasta trebuie sa fie declarata si marcata cu extern, functiile sunt implicit declarate extern in cazul cand nu sunt marcate cu **static.** In fisierul .asm va trebui declarata o eticheta de tip **PUBLIC** cu aceeasi denumire ca si functia declarata in C.

```
void LCD_SetPixel(uint32_t u32Com, uint32_t u32Seg, uint32_t u32OnFlag);
```

Implementarea functiei LCD_SetPixel in limbaj de asamblare.

Pentru scrierea codului in asamblare este necesar sa stim ca toate procesoarele ARM-v6m cu un set de instructiuni RISC, suporta doar setul de instructiune THUMB pe 16 biti, din acest motiv este necesar sa declaram alinierea sectiunii de cod la 2 bytes (**NOROOT(2)**) si activarea setului de instructiuni **THUMB**. Parametrii functiei sunt transmisi prin registrii **R1**, **R2**, **R3**, valoarea de return este scrisa in **R1**, iar la iesirea din functie este necesara scrierea de pe stiva a valorii de return in registrul **PC**. De asemenea, orice sectiune de date trebuie aliniata la 4 bytes, din aceasta cauza, la sfarsitul codului este inserata o instructione **NOP** pentru a avea un numar de 38 de instructiuni, care va fi echivalentul a 38 * 2 = 76 bytes.

```
        PUBLIC LCD_SetPixel

;; Nuvoton130 has a Cortex-m0, ARM v6-m, Supports only THUMB instructions
        SECTION `.text`:CODE:NOROOT(2)
        THUMB

;; The symbol name has to match the C function name
LCD_SetPixel:
        ;; Save modified registers
        PUSH    {R4 - R7, LR}
        ;; Second argument (u32Seg) is passed in R1 register
        ;; To calculate it divided by 4, shift by 2 to the left, store in R3
        LSRS     R3, R1, #+2
        ;; Multiply memnum (stored in R3) by 4 store in R4
        LSLS     R4, R3, #+2
        ;; u32Seg - 4 * memnum (stored in R4)
        SUBS     R4, R1, R4
        ;; Multiply the rezult by 8 (shift left)
        LSLS     R4, R4, #+3
        ;; Load in R5 the address of the memory mapped LCD segment data
        LDR      R5, LCD_MEM_BASE ;; (0x400b0008)

        ;; Calculate mask
        MOVS     R6, #+4
        MULS     R6, R3, R6
        LDR      R7, [R5, R6]
        MOVS     R6, #+1
        LSLS     R6, R6, R0
        LSLS     R6, R6, R4

        ;; Check if we want to turn it on or off
        CMP      R2, #+0
        BEQ      ResetPixel

SetPixel:
        ;; Set the required pixel
        ORRS     R6, R6, R7
        MOVS     R7, #+4
        MULS     R7, R3, R7
        STR      R6, [R5, R7]
        B        Ready

ResetPixel:
        ;; Reset the required pixel
        BICS     R7, R7, R6
        MOVS     R6, #+4
        MULS     R6, R3, R6
        STR      R7, [R5, R6]

Ready:
        ;; SysTick->LOAD = 300 * CyclesPerUs (225);
        MOVS     R6, #+225
        LSLS     R6, R6, #+4
        LDR      R7, SYS_TICK_LOAD   ;; 0xe000e014
        STR      R6, [R7, #+0]

        ;; SysTick->VAL = 0;
        MOVS     R6, #+0
        LDR      R7, SYS_TICK_VALUE  ;; 0xe000e018
        STR      R6, [R7, #+0]

        ;; SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk;
        LDR      R6, SYS_TICK_CTRL   ;; 0xe000e010
        MOVS     R7, #+5
        STR      R7, [R6, #+0]

        ;; while((SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk) == 0);
WaitTimer:
        LDR      R7, [R6, #+0]
        LSLS     R7, R7, #+15
        BPL      WaitTimer

        POP      {R4 - R7, PC}       ;; return
        ;; In THUMB mode, data sections have to be 4 byte aligned, since we have
        ;; 37 instructions, we need one more to make it aligned
        NOP

        DATA
LCD_MEM_BASE:
        DC32     0x400b0008
SYS_TICK_LOAD:
        DC32     0xe000e014
SYS_TICK_VALUE:
        DC32     0xe000e018
SYS_TICK_CTRL:
        DC32     0xe000e010

        END
```



Concluzii:

In cazul aplicatiilor embedded este vitala intelegerea detailata a platformei de lucru si a resurselor aferente acesteia. Limbajul C ne expune o interfata simplificata pentru a genera codul pentru sistemul pe care lucram, insa fara o cunoastere buna a codului masina generat este dificil de depanat codul, in special atunci cand o componenta hardware nu produce rezultatele dorite. Este uneori necesar sa scriem unele parti ale proiectului in limbaj de asamblare pentru a obtine acces la resursele low-level ale sistemului. Pentru a putea folosi aceste componente, este nevoie de o intelegere buna a modului in care compilatorul de C genereaza codul in asamblare si cum putem interschimba cu usurinta aceste 2 limbaje.

Referinte:

ARMv6-M Architecture Reference Manual

NuMicro Family Nano 100 Series Datasheet

https://github.com/PetricaP/ProiectSOC/references

https://en.wikipedia.org/wiki/Liquid-crystal_display

https://en.wikipedia.org/wiki/Charge_pump

https://www.pacificdisplay.com/lcd_multiplex_drive.htm