Abstract:

In the beginning of the modern era, the computer was invented to revolutionize society. However, the modern-day computer has many components and is very expensive, so it is not very effective use of resources for all systems to be run through a modern computer. To save on cost, most modern systems run on microprocessors such as the MCB1700 to eliminate the cost of extraneous features. This paper will explore the usage of the MCB1700 board as a multimedia media center. The multimedia center usage will be a photo gallery, an mp3 player though the pc cable and finally a game center.

Introduction

The MCB1700 board has many functionalities that allow it to accomplish anything a modern computer will be able to do. The only limitation of the board is on chip memory of 512kb flash and 64kb ram so if the program is very dense it would not be able to fit on the board. Also, the processor is a 100Mhz processor so running any processor heavy code will cause it to be extremely slow. For all intents and purposes, the MCB1700 board has the specification need to create a simple multi media center.

Past Work

The most relevant piece of past work is lab 1 of the embedded systems design. In this lab, the board was used to create an LCD screen display, to read values off the potential meter, and getting joystick input. A lot of code component and knowledge were used from this lab. Another project was the pong game though a video graphics array adapter on a separate board for Digital System Design which would help in the game design.

Methodology

The methodology to designing the project was to first get a working menu screen. Once, the working menu screen was created the project was complete a major component of the project would be complete. The menu screen only must call separate functions for photo, music player, and game. The photo gallery uses the same methodology as the menu screen as menu screen but rather than calling a method, it shows a different image. The music player was designed next and only the UI was needed as most of the core components were from the arm design example. The game was implemented last as it needed the most coding. The methodology was design from easiest to hardest.

Design

For the design to be modular, the media center was spilt into 4 files. The first file is the main method for the menu selects screen. The second file is for the photo gallery display screen. The

third file is the USB main file which connects and disconnects the USB. The final file was for the game.

Main File

In the main file, the first thing it need to do is initialize every needed for display such as the joystick and screen. Also, it sets where the selector of the menu and by default it starts at 1. It draws the menu by displaying strings towards screen and based on where the selector is it redraws that single line that the selector is on as there is no need to refresh the entire screen. The method takes input from the joystick of down, up, left. Down and up is for the selector and right is for entering the method. The joystick needs to keep track of its previous value, so it does not take the value twice because an interrupted happened twice while the user was holding joystick. It calls the method for whichever function the user specified such a photo gallery. After, it would refresh the screen and keep looping to make a responsive system.

Photo Gallery

In the photo gallery file, there is no initialization since the main file initializes the components needed. The photo gallery displays a header for the user and instruction on how to get back. The header has a selector which chooses which picture to display to the user. If the user goes behind the min or max index, the code wraps back around and sets the index to either max or min. The photo gallery displays the image towards the screen. To keep the size of image inside the screen, the images used were 150 x 150 so the photo gallery would look consistent. The image had to be flipped vertical in an image editing software as the board reads it in that direction. Once it detects the middle joystick button input, the file returns so it exits this function.

USB main audio

The USB must initialize the USB and enable the potentiometer interrupts. It creates the USB display screen once. It sets the USB connection to true. While the processor does not detect left joystick input, the processor would be stuck in a loop of getting potentiometer volume. Once it detects left joystick, it disables the interrupts, resets the memory location, sets the USB connection to false, and it returns which exits this function.

Game File

The game file first displays the instruction menu followed by a delay. It randomly generator a number between 0-3 and puts this into an array of answer sequence. Based upon which level the user is on, it generates n number of random numbers and puts it into the array of size 100. This means the max level is 100 as that is the max size of the array. Based on the answer array, the screen becomes that color followed by a delay and a white screen. The white screen was generated incase two of the same color is generated side by side so the user will be able to differentiate the difference. It displays which joystick value represents which color. It waits for the user to enter joystick values until the entire sequence is complete. If the user enters a sequence in the wrong order, the game over screen is displayed followed by a delay. Then, the function returns which exits the game. If the user enters the correct sequence, the screen displays

the next level screen. The sequence length is increased by one, so the user has more colors to remember each level. The increasing stops at level 100 in which case it will keep continue 100.

Experimental Results

The media center was successfully implemented. Some tricks were needed to make the screen responsive. The string display was response time was fast so for everything for but the game it was the display was fine. However, when I was trying to create more complex game, the screen refresh rate was very slow. To refresh the entire screen, the processor would have a notice able delay. A way around it was to localize the refreshing meaning to only refresh the images that needed to be refresh. The photo gallery only refreshes the picture and not the background nor the header which keeps the gallery smooth. The picture size was small to minimize delay. The menu screen header was only drawn once and only the selection line was redrawn and the new selection line to keep refresh rates high. This issue was also avoided during the game by making a game that does not need to refresh multiple times, but the slowness is still shown when I refresh to white screen to pad in between colors. Another issue was the USB, it needs the computer to allow it permission to take the speaker audio which is easily done shown figure 3. Also, some boards do not have working audio connection because of hardware issues. This was remedy by changing boards until the audio worked. Below is some sample figure from the implemented design.

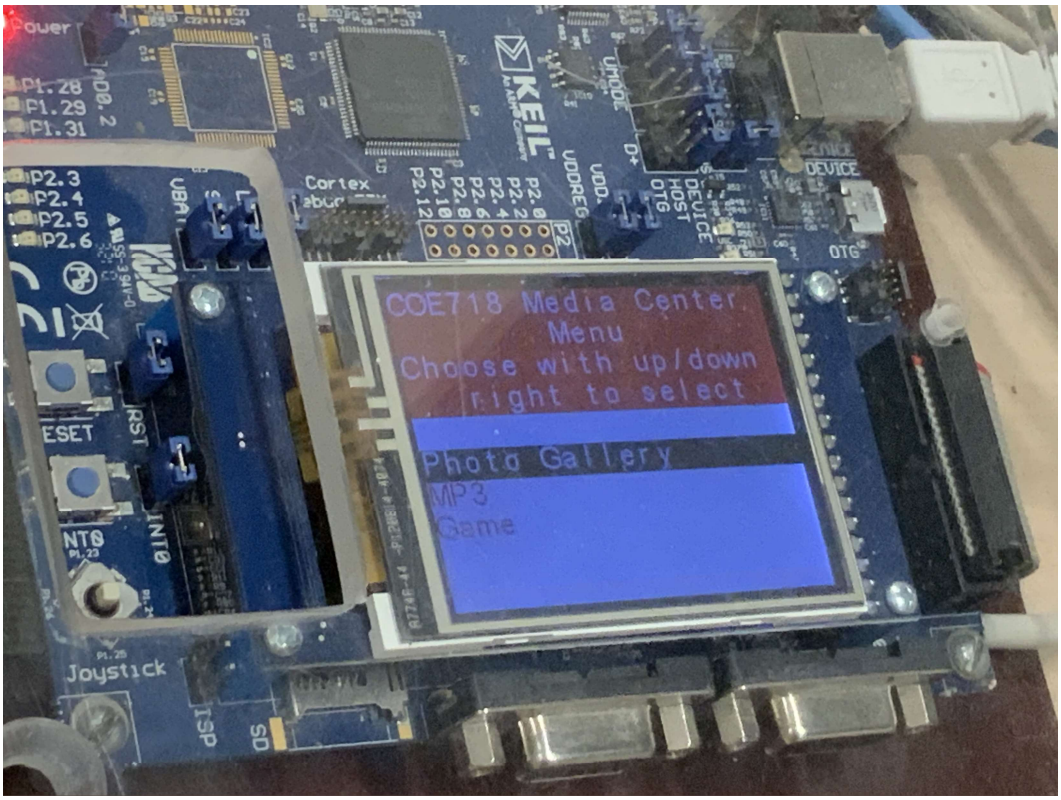

Figure 1: Main Menu Screen

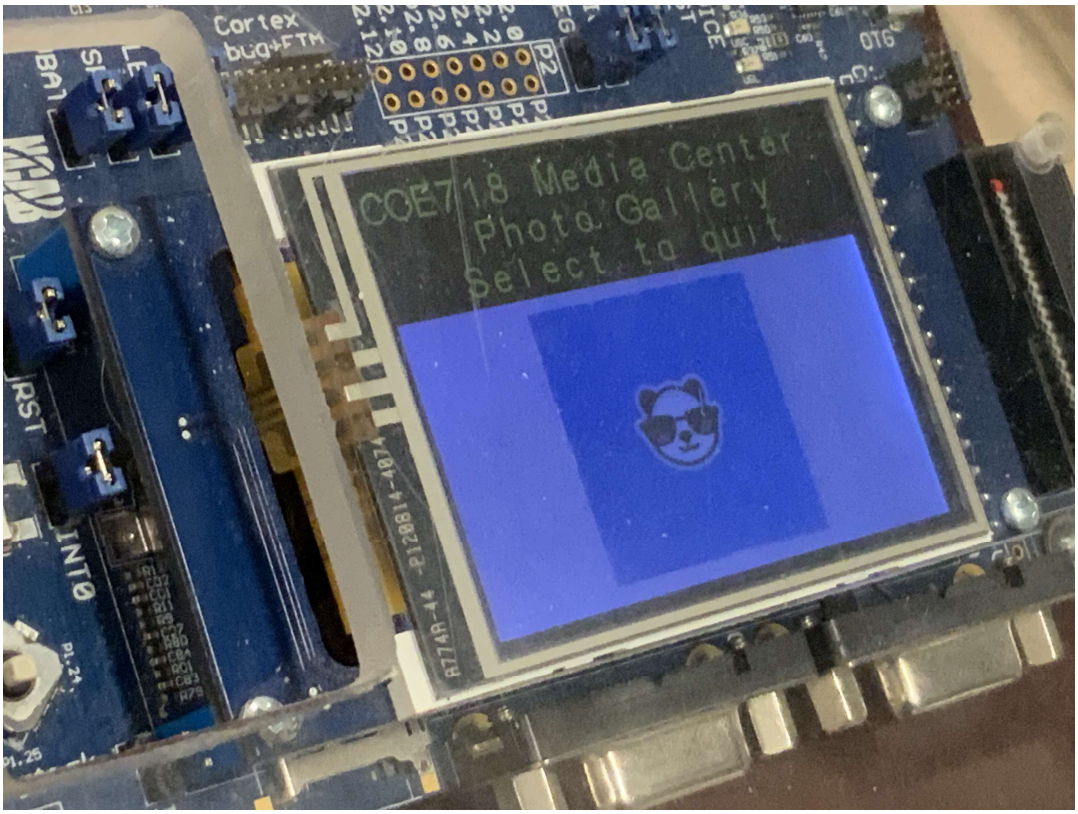

Figure 2: Photo Gallery Screen with index equal 3

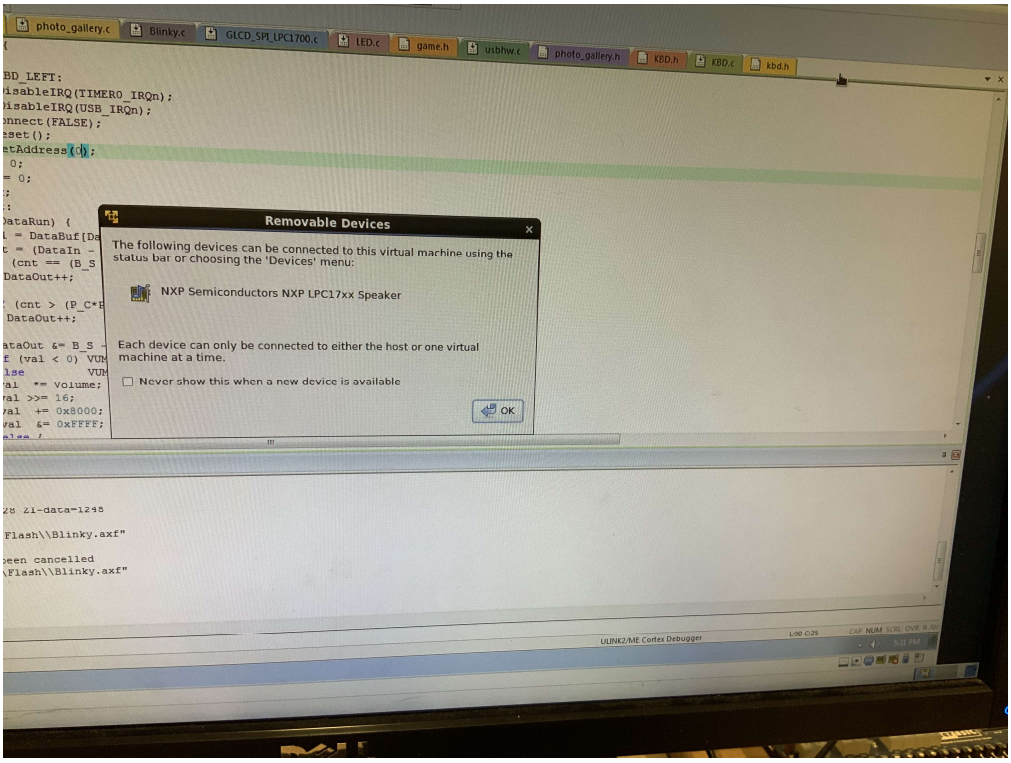

Figure 3: USB Device Adapter Connection

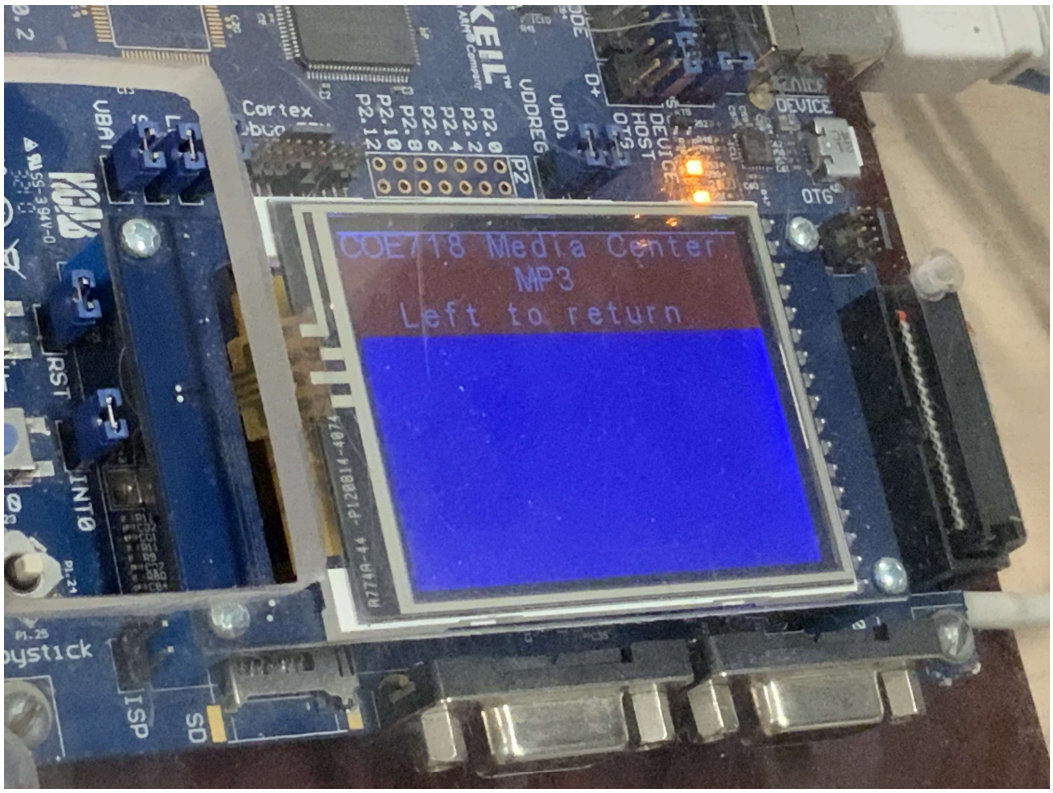

Figure 3 : USB Display Menu

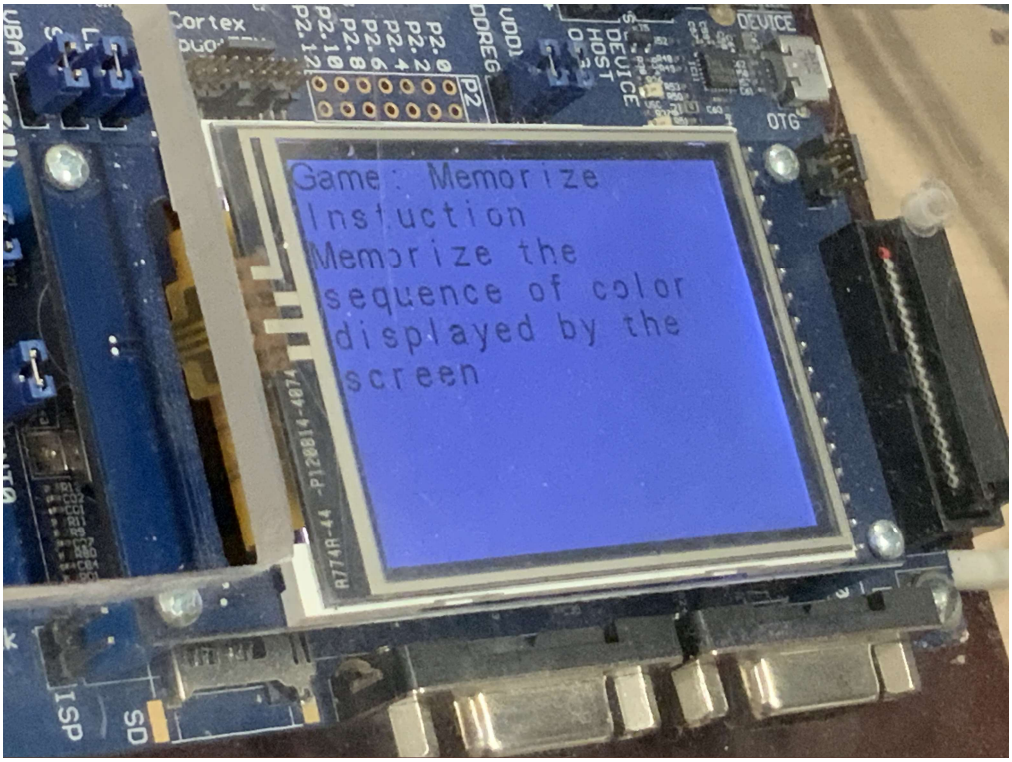

Figure 4: Game Menu Display

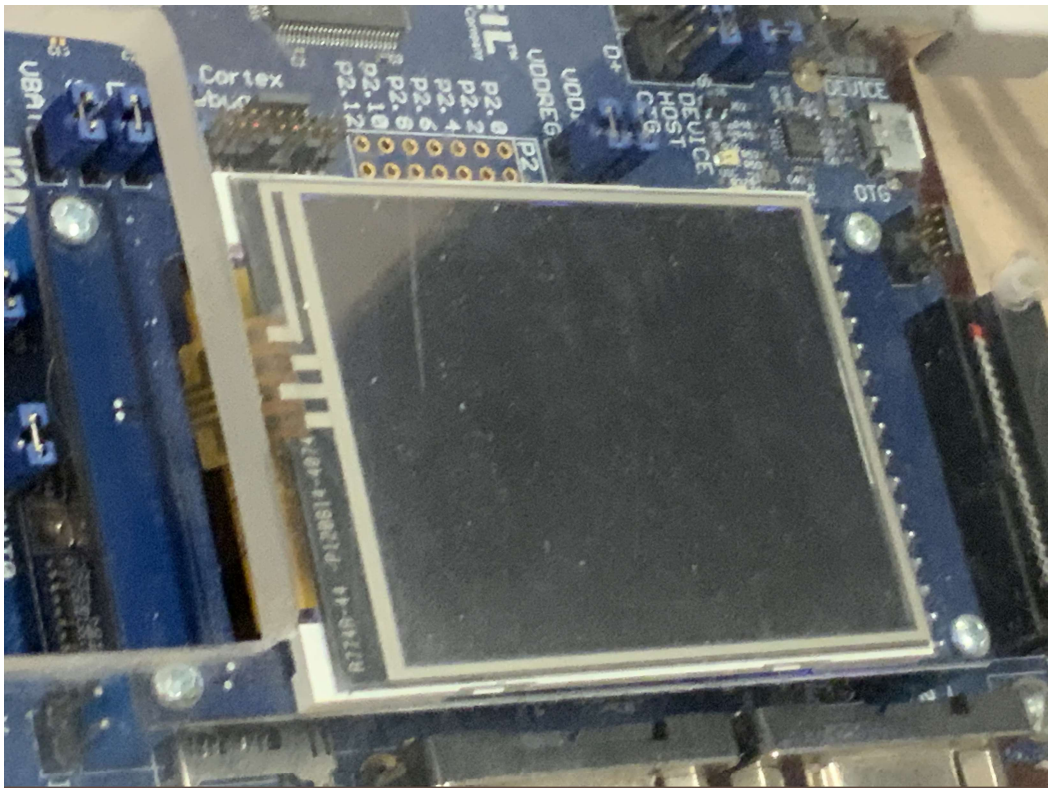

Figure 5: Game Color Display Color Equals Black

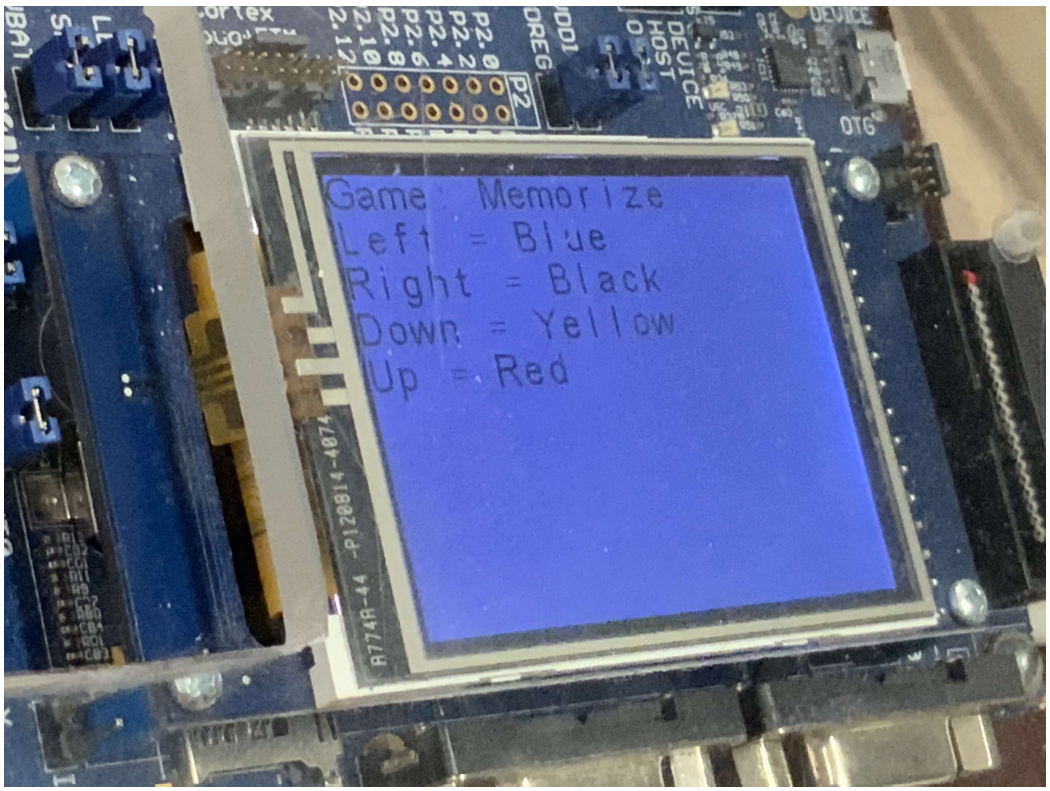

Figure 6 : User Input Selection Scree

Conclusion:

In conclusion, the MC1700 board can implement a multimedia center. The designer will face hardware bound issues such as screen refresh rate because of the slow processor. The designer can overcome these challenges by figure out creative solution such as refreshing part of the screen. Ultimately, the designer should figure out a solution to the problem though the usage and knowledge of software. Only as a last resort should the designer upgrade the hardware or the board in this case. As a good designer of embedded system would not only think about best outcome of the solution but also the cost of the solution. In the end, the systems goal is to be sold for profit and minizine production cost will increases profit. The correct balance between cost and functionality is what differentiate the skill level between decent designer and an excellent designer.

References:

Khan, G. (2019). *Lab Manual*. [online] Ee.ryerson.ca. Available at: https://www.ee.ryerson.ca/~courses/coe718/lab-project.html [Accessed 30 Nov. 2019].

Appendix

Blinky.c

```
#include <LPC17xx.H>                    /* NXP LPC17xx definitions
*/
#include "string.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"
#include "usbdmain.h"
#include "photo_gallery.h"
#include "game.h"


extern int audio_main (void);


/*----------------------------------------------------------------------------
-
  Main Program
 *----------------------------------------------------------------------------
-*/
int main (void)
{       /* Main Program                         */
        int selector = 1; //selector to see which program is user the choosing

        //'1' for photo viewer, '2' for audio file, '3' for game
        int joystick_val = 0;   //track the current joystick value
        int joystick_prev = 0;  //track the previous value for the joystick

        KBD_Init();

  LED_Init ();
  GLCD_Init();

  GLCD_Clear  (White);
  SysTick_Config(SystemCoreClock/100);
        GLCD_SetBackColor(Red);
  GLCD_SetTextColor(White);
        GLCD_DisplayString (0, 0, 1, "COE718 Media Center ");
        GLCD_DisplayString (1, 0, 1, "                    ");
        GLCD_DisplayString (2, 0, 1, "Choose with up/down ");
        GLCD_DisplayString (3, 0, 1, "   right to select  ");

  for (;;)
        {
                        joystick_val = get_button();

                        if (joystick_val !=joystick_prev)
                          {
                                        if (joystick_val == KBD_DOWN)
                                                {
```

```
                                                selector +=1;
                                                if (selector >=4)
                                                        selector
= 1;
                                        }
                                else if (joystick_val == KBD_UP)
                                        {
                                                selector -=1;
                                                if (selector <=0)
                                                        selector
= 3;
                                        }
                                else if(joystick_val == KBD_RIGHT)
                                                {
                                                        if (selector ==
1)
                                                        {

        image_view();

        GLCD_Clear(White);

        GLCD_SetTextColor(Red);

        GLCD_DisplayString (0, 0, 1, "COE718 Media Center ");

        GLCD_DisplayString (1, 0, 1, "   Photo Gallery    ");
                                                        selector
= 1;
                                                        }
                                                        else if
(selector == 2)
                                                        {

        audio_main();

        GLCD_Clear(White);

        GLCD_SetTextColor(Red);

        GLCD_DisplayString (0, 0, 1, "COE718 Media Center ");

        GLCD_DisplayString (1, 0, 1, "        MP3         ");

                                                        selector
=1;
                                                        }
                                                        else if
(selector == 3)
                                                        {
                                                        game();

        GLCD_Clear(White);

        GLCD_SetTextColor(Red);

        GLCD_DisplayString (0, 0, 1, "COE718 Media Center ");
```

```
        GLCD_DisplayString (1, 0, 1, "       Game         ");
                                                                selector
= 1;
                                                              }
                                                          }
                                        joystick_prev = joystick_val;
                      }

                      if (selector == 1){
                                        GLCD_SetBackColor(Black);
                                        GLCD_SetTextColor(White);
                                        GLCD_DisplayString (5, 0, 1,
"Photo Gallery       ");
                                        GLCD_SetBackColor(White);
                                        GLCD_SetTextColor(Red);
                                        GLCD_DisplayString (6, 0, 1,
"MP3                 ");
                                        GLCD_DisplayString (7, 0, 1,
"Game                ");

                              } else if(selector == 2){
                                        GLCD_SetBackColor(White);
                                        GLCD_SetTextColor(Red);
                                        GLCD_DisplayString (5, 0, 1,
"Photo Gallery       ");
                                   GLCD_SetBackColor(Black);
                                        GLCD_SetTextColor(White);
                                        GLCD_DisplayString (6, 0, 1,
"MP3                 ");
                                        GLCD_SetBackColor(White);
                                        GLCD_SetTextColor(Red);
                                        GLCD_DisplayString (7, 0, 1,
"Game                ");
                              } else if(selector == 3){
                                        GLCD_SetBackColor(White);
                                        GLCD_SetTextColor(Red);
                                        GLCD_DisplayString (5, 0, 1,
"Photo Gallery       ");
                                        GLCD_DisplayString (6, 0, 1,
"MP3                 ");
                                   GLCD_SetBackColor(Black);
                                        GLCD_SetTextColor(White);
                                        GLCD_DisplayString (7, 0, 1,
"Game                ");
                              } else{
                                        GLCD_SetBackColor(White);
                                        GLCD_SetTextColor(Red);
                                        GLCD_DisplayString (5, 0, 1,
"Photo Gallery       ");
                                        GLCD_DisplayString (6, 0, 1,
"MP3                 ");
                                        GLCD_DisplayString (7, 0, 1,
"Game                ");
                              }
        GLCD_SetBackColor(Red);
  GLCD_SetTextColor(White);
```

```
        GLCD_DisplayString (0, 0, 1, "COE718 Media Center ");
        GLCD_DisplayString (1, 0, 1, "        Menu        ");
        GLCD_DisplayString (2, 0, 1, "Choose with up/down ");
        GLCD_DisplayString (3, 0, 1, "   right to select  ");
        }
}
```

Game.c

```
#include <LPC17xx.H>

#include "GLCD.h"
#include "LED.h"
#include "KBD.h"

#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void showcolor (int color);
int delay= 5000;

void delay_ms (uint16_t ms)
{
 uint16_t delay;
 volatile uint32_t i;
 for (delay = ms; delay >0 ; delay--)
  {
  for (i=3500; i >0;i--){};
  }
}

void game(void)
{
        int joystick_prev = 0;
        int joystick_val = 0;
        int color;
        int index;
        int ansnum;
        int ansgot=0;
        int  ans[100];
        int input;
        int level=1;
        GLCD_Clear(White);
        GLCD_SetBackColor(White);
  GLCD_SetTextColor(Black);
        GLCD_DisplayString (0, 0, 1, "Game: Memorize       ");
        GLCD_DisplayString (1, 0, 1, "Instuction           ");
        GLCD_DisplayString(2, 0, 1, "Memorize the  ");
        GLCD_DisplayString(3, 0, 1, "sequence of color  ");
        GLCD_DisplayString(4, 0, 1, "displayed by the ");
        GLCD_DisplayString(5, 0, 1, "screen ");

        delay =30000;
        delay_ms(delay);

        while (1){
```

```
		index = 0;
		delay =10000;

		while(index <level){
			color = rand()% 4;
			showcolor(color);
			ans[index] = color;
			GLCD_Clear(White);

			index= index+1;
		}
		GLCD_Clear(White);
		GLCD_SetBackColor(White);
		GLCD_SetTextColor(Black);
		GLCD_DisplayString (0, 0, 1, "Game: Memorize       ");
		GLCD_DisplayString (1, 0, 1, "Left = Blue           ");
		GLCD_DisplayString(2, 0, 1, "Right = Black ");
		GLCD_DisplayString(3, 0, 1, "Down = Yellow");
		GLCD_DisplayString(4, 0, 1, "Up = Red");
		ansnum = 0;

		while (ansnum < index){
			ansgot = 0;
			joystick_val = get_button();
			if (joystick_val !=joystick_prev)
			{
				if (joystick_val == KBD_RIGHT)
				{
					input =3;
					ansgot = 1;
				}
				else if (joystick_val== KBD_UP){
					input =0;
					ansgot = 1;
				}
				else if (joystick_val == KBD_DOWN){
					input =2;
					ansgot = 1;
				}
				else if (joystick_val == KBD_LEFT){
					input =1;
					ansgot = 1;
				}

				joystick_prev = joystick_val;
			}
			if (ansgot == 1){
				if (ans[ansnum] ==input){
					ansnum ++;
				}
				else{
						GLCD_Clear(White);
						GLCD_DisplayString (1, 0, 1,
"Game Over");
						GLCD_DisplayString(2, 0, 1, "You
lose");
				  if (ans[ansnum] == 0)
```

```
                                                        GLCD_DisplayString(3, 0,
1, "The color was Red");
                                                if (ans[ansnum] == 1)
                                                        GLCD_DisplayString(3, 0,
1, "The color was Blue");
                                                if (ans[ansnum] == 2)
                                                        GLCD_DisplayString(3, 0,
1, "The color was Yellow");
                                                if (ans[ansnum] == 3)
                                                        GLCD_DisplayString(3, 0,
1, "The color was Black");
                                                delay_ms(delay);
                                        return;
                                }
                        }

                }
                level = level+1;
                GLCD_Clear(White);
          GLCD_DisplayString (5, 0, 1, "Next Level");
                delay_ms(delay);
        }
}


void showcolor (int color){
        if (color == 0){
                GLCD_Clear(Red);
                delay_ms(delay);
        }
        else if (color == 1){
                GLCD_Clear(Blue);
                delay_ms(delay);
        }
        else if (color == 2){
                GLCD_Clear(Yellow);
                delay_ms(delay);
        }
        else {
                GLCD_Clear(Black);
                delay_ms(delay);
        }

}
```

Photo gallery

```
#include <LPC17xx.H>
#include "photo_gallery.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"

extern unsigned char sim[];
extern unsigned char simile[];
extern unsigned char asd[];
```

```
void image_view(void)
{
        int joystick_prev = 0;
        int joystick_val = 0;
        int seq = 0;
        int exit = 0;
        GLCD_Clear(White);
        GLCD_SetBackColor(Black);
  GLCD_SetTextColor(Yellow);
        GLCD_DisplayString (0, 0, 1, "COE718 Media Center ");
        GLCD_DisplayString (1, 0, 1, "    Photo Gallery   ");
        GLCD_DisplayString (2, 0, 1, "   Select to quit   ");
        while(!exit){
                joystick_val = get_button();
                if (joystick_val !=joystick_prev)
                {
                        if (joystick_val == KBD_RIGHT)
                        {
                                seq +=1;
                                if (seq == 3)
                          seq = 0;
                        }
                        else if (joystick_val == KBD_LEFT)
                        {
                                seq -=1;
                                if (seq < 0)
                                        seq = 2;
                        }
                        else if (joystick_val == KBD_SELECT)
                        {
                                exit =1;
                        }
                        joystick_prev = joystick_val;
                }
        if (seq == 0){
                GLCD_Bitmap(80, 80, 150, 150, sim);
        } else if(seq == 1){
                GLCD_Bitmap(80, 80, 150, 150, simile);
        } else if(seq == 2){
                GLCD_Bitmap(80, 80, 150, 150, asd);
        }
        }
}
```

Usbmain.c

```
/*----------------------------------------------------------------------------
-
 * Name:    usbmain.c
 * Purpose: USB Audio Class Demo
 * Version: V1.20
 *----------------------------------------------------------------------------
-
 *      This software is supplied "AS IS" without any warranties, express,
 *      implied or statutory, including but not limited to the implied
 *      warranties of fitness for purpose, satisfactory quality and
 *      noninfringement. Keil extends you a royalty-free right to reproduce
```

```
 *      and distribute executable files created using this software for use
 *      on NXP Semiconductors LPC microcontroller devices only. Nothing else
 *      gives you the right to use this software.
 *
 * Copyright (c) 2009 Keil - An ARM Company. All rights reserved.
 *----------------------------------------------------------------------------
*/

#include "LPC17xx.h"                        /* LPC17xx definitions */
#include "type.h"

#include "usb.h"
#include "usbcfg.h"
#include "usbhw.h"
#include "usbcore.h"
#include "usbaudio.h"

extern  void SystemClockUpdate(void);
extern uint32_t SystemFrequency;
uint8_t  Mute;                              /* Mute State */
uint32_t Volume;                            /* Volume Level */

#if USB_DMA
uint32_t *InfoBuf = (uint32_t *)(DMA_BUF_ADR);
short *DataBuf = (short *)(DMA_BUF_ADR + 4*P_C);
#else
uint32_t InfoBuf[P_C];
short DataBuf[B_S];                         /* Data Buffer */
#endif

uint16_t  DataOut;                           /* Data Out Index */
uint16_t  DataIn;                            /* Data In Index */

uint8_t   DataRun;                           /* Data Stream Run State */
uint16_t  PotVal;                            /* Potenciometer Value */
uint32_t  VUM;                               /* VU Meter */
uint32_t  Tick;                              /* Time Tick */


/*
 * Get Potenciometer Value
 */

void get_potval (void) {
  uint32_t val;

  LPC_ADC->CR |= 0x01000000;              /* Start A/D Conversion */
  do {
    val = LPC_ADC->GDR;                   /* Read A/D Data Register */
  } while ((val & 0x80000000) == 0);      /* Wait for end of A/D Conversion
*/
  LPC_ADC->CR &= ~0x01000000;             /* Stop A/D Conversion */
  PotVal = ((val >> 8) & 0xF8) +          /* Extract Potenciometer Value */
           ((val >> 7) & 0x08);
}
```

```
/*
 * Timer Counter 0 Interrupt Service Routine
 *   executed each 31.25us (32kHz frequency)
 */

void TIMER0_IRQHandler(void)
{
  long  val;
  uint32_t cnt;

  if (DataRun) {                                /* Data Stream is running */
    val = DataBuf[DataOut];                     /* Get Audio Sample */
    cnt = (DataIn - DataOut) & (B_S - 1);       /* Buffer Data Count */
    if (cnt == (B_S - P_C*P_S)) {               /* Too much Data in Buffer */
      DataOut++;                                /* Skip one Sample */
    }
    if (cnt > (P_C*P_S)) {                      /* Still enough Data in Buffer */
      DataOut++;                                /* Update Data Out Index */
    }
    DataOut &= B_S - 1;                         /* Adjust Buffer Out Index */
    if (val < 0) VUM -= val;                    /* Accumulate Neg Value */
    else         VUM += val;                    /* Accumulate Pos Value */
    val  *= Volume;                             /* Apply Volume Level */
    val >>= 16;                                 /* Adjust Value */
    val  += 0x8000;                             /* Add Bias */
    val  &= 0xFFFF;                             /* Mask Value */
  } else {
    val = 0x8000;                               /* DAC Middle Point */
  }

  if (Mute) {
    val = 0x8000;                               /* DAC Middle Point */
  }

  LPC_DAC->CR = val & 0xFFC0;                 /* Set Speaker Output */

  if ((Tick++ & 0x03FF) == 0) {                 /* On every 1024th Tick */
    get_potval();                               /* Get Potenciometer Value */
    if (VolCur == 0x8000) {                     /* Check for Minimum Level */
      Volume = 0;                               /* No Sound */
    } else {
      Volume = VolCur * PotVal;                 /* Chained Volume Level */
    }
    val = VUM >> 20;                            /* Scale Accumulated Value */
    VUM = 0;                                    /* Clear VUM */
    if (val > 7) val = 7;                       /* Limit Value */
  }

  LPC_TIM0->IR = 1;                             /* Clear Interrupt Flag */
}



/*****************************************************************************
*
**   Main Function  main()
```

```
*********************************************************************************
*/
int main (void)
{
  volatile uint32_t pclkdiv, pclk;

  /* SystemClockUpdate() updates the SystemFrequency variable */
  SystemClockUpdate();

  LPC_PINCON->PINSEL1 &=~((0x03<<18)|(0x03<<20));
  /* P0.25, A0.0, function 01, P0.26 AOUT, function 10 */
  LPC_PINCON->PINSEL1 |= ((0x01<<18)|(0x02<<20));

  /* Enable CLOCK into ADC controller */
  LPC_SC->PCONP |= (1 << 12);

  LPC_ADC->CR = 0x00200E04;           /* ADC: 10-bit AIN2 @ 4MHz */
  LPC_DAC->CR = 0x00008000;           /* DAC Output set to Middle Point */

  /* By default, the PCLKSELx value is zero, thus, the PCLK for
  all the peripherals is 1/4 of the SystemFrequency. */
  /* Bit 2~3 is for TIMER0 */
  pclkdiv = (LPC_SC->PCLKSEL0 >> 2) & 0x03;
  switch ( pclkdiv )
  {
        case 0x00:
        default:
          pclk = SystemFrequency/4;
        break;
        case 0x01:
          pclk = SystemFrequency;
        break;
        case 0x02:
          pclk = SystemFrequency/2;
        break;
        case 0x03:
          pclk = SystemFrequency/8;
        break;
  }

  LPC_TIM0->MR0 = pclk/DATA_FREQ - 1; /* TC0 Match Value 0 */
  LPC_TIM0->MCR = 3;                                  /* TCO Interrupt and
Reset on MR0 */
  LPC_TIM0->TCR = 1;                                  /* TC0 Enable */
  NVIC_EnableIRQ(TIMER0_IRQn);

  USB_Init();                           /* USB Initialization */
  USB_Connect(TRUE);           /* USB Connect */

}

/*********************************************************************************
**
**                            End Of File
*********************************************************************************
*/
```

Abcuser.c

```
/*----------------------------------------------------------------------------
-
 *      U S B  -  K e r n e l
 *----------------------------------------------------------------------------
-
 *      Name:    ADCUSER.C
 *      Purpose: Audio Device Class Custom User Module
 *      Version: V1.10
 *----------------------------------------------------------------------------
-
 *      This software is supplied "AS IS" without any warranties, express,
 *      implied or statutory, including but not limited to the implied
 *      warranties of fitness for purpose, satisfactory quality and
 *      noninfringement. Keil extends you a royalty-free right to reproduce
 *      and distribute executable files created using this software for use
 *      on NXP Semiconductors LPC family microcontroller devices only.
Nothing
 *      else gives you the right to use this software.
 *
 * Copyright (c) 2009 Keil - An ARM Company. All rights reserved.
 *----------------------------------------------------------------------------
*/

#include "type.h"

#include "usb.h"
#include "audio.h"
#include "usbcfg.h"
#include "usbcore.h"
#include "adcuser.h"

#include "usbaudio.h"

      uint16_t VolCur = 0x0100;     /* Volume Current Value */
const uint16_t VolMin = 0x0000;     /* Volume Minimum Value */
const uint16_t VolMax = 0x0100;     /* Volume Maximum Value */
const uint16_t VolRes = 0x0004;     /* Volume Resolution */

/*
 *  Audio Device Class Interface Get Request Callback
 *   Called automatically on ADC Interface Get Request
 *    Parameters:      None (global SetupPacket and EP0Buf)
 *    Return Value:    TRUE - Success, FALSE - Error
 */

uint32_t ADC_IF_GetRequest (void) {

/*
  Interface = SetupPacket.wIndex.WB.L;
  EntityID  = SetupPacket.wIndex.WB.H;
  Request   = SetupPacket.bRequest;
  Value     = SetupPacket.wValue.W;
  ...
*/
```

```
  if (SetupPacket.wIndex.W == 0x0200) {
    /* Feature Unit: Interface = 0, ID = 2 */
    if (SetupPacket.wValue.WB.L == 0) {
      /* Master Channel */
      switch (SetupPacket.wValue.WB.H) {
        case AUDIO_MUTE_CONTROL:
          switch (SetupPacket.bRequest) {
            case AUDIO_REQUEST_GET_CUR:
              EP0Buf[0] = Mute;
              return (TRUE);
          }
          break;
        case AUDIO_VOLUME_CONTROL:
          switch (SetupPacket.bRequest) {
            case AUDIO_REQUEST_GET_CUR:
              *((__packed uint16_t *)EP0Buf) = VolCur;
              return (TRUE);
            case AUDIO_REQUEST_GET_MIN:
              *((__packed uint16_t *)EP0Buf) = VolMin;
              return (TRUE);
            case AUDIO_REQUEST_GET_MAX:
              *((__packed uint16_t *)EP0Buf) = VolMax;
              return (TRUE);
            case AUDIO_REQUEST_GET_RES:
              *((__packed uint16_t *)EP0Buf) = VolRes;
              return (TRUE);
          }
          break;
      }
    }
  }
  return (FALSE);  /* Not Supported */
}


/*
 *  Audio Device Class Interface Set Request Callback
 *   Called automatically on ADC Interface Set Request
 *    Parameters:      None (global SetupPacket and EP0Buf)
 *    Return Value:    TRUE - Success, FALSE - Error
 */

uint32_t ADC_IF_SetRequest (void) {

/*
  Interface = SetupPacket.wIndex.WB.L;
  EntityID  = SetupPacket.wIndex.WB.H;
  Request   = SetupPacket.bRequest;
  Value     = SetupPacket.wValue.W;
  ...
*/

  if (SetupPacket.wIndex.W == 0x0200) {
    /* Feature Unit: Interface = 0, ID = 2 */
    if (SetupPacket.wValue.WB.L == 0) {
      /* Master Channel */
      switch (SetupPacket.wValue.WB.H) {
```

```
        case AUDIO_MUTE_CONTROL:
          switch (SetupPacket.bRequest) {
            case AUDIO_REQUEST_SET_CUR:
              Mute = EP0Buf[0];
              return (TRUE);
          }
          break;
        case AUDIO_VOLUME_CONTROL:
          switch (SetupPacket.bRequest) {
            case AUDIO_REQUEST_SET_CUR:
              VolCur = *((__packed uint16_t *)EP0Buf);
              return (TRUE);
          }
          break;
      }
    }
  }
  return (FALSE);  /* Not Supported */
}


/*
 *  Audio Device Class EndPoint Get Request Callback
 *   Called automatically on ADC EndPoint Get Request
 *    Parameters:      None (global SetupPacket and EP0Buf)
 *    Return Value:    TRUE - Success, FALSE - Error
 */

uint32_t ADC_EP_GetRequest (void) {

/*
  EndPoint = SetupPacket.wIndex.WB.L;
  Request  = SetupPacket.bRequest;
  Value    = SetupPacket.wValue.W;
  ...
*/
  return (FALSE);  /* Not Supported */
}


/*
 *  Audio Device Class EndPoint Set Request Callback
 *   Called automatically on ADC EndPoint Set Request
 *    Parameters:      None (global SetupPacket and EP0Buf)
 *    Return Value:    TRUE - Success, FALSE - Error
 */

uint32_t ADC_EP_SetRequest (void) {

/*
  EndPoint = SetupPacket.wIndex.WB.L;
  Request  = SetupPacket.bRequest;
  Value    = SetupPacket.wValue.W;
  ...
*/
  return (FALSE);  /* Not Supported */
}
```

Core cm3.c

```
/**************************************************************************
**
 * @file:    core_cm3.c
 * @purpose: CMSIS Cortex-M3 Core Peripheral Access Layer Source File
 * @version: V1.20
 * @date:    22. May 2009
 *------------------------------------------------------------------------
-
 *
 * Copyright (C) 2009 ARM Limited. All rights reserved.
 *
 * ARM Limited (ARM) is supplying this software for use with Cortex-Mx
 * processor based microcontrollers.  This file can be freely distributed
 * within development tools that are supporting such ARM based processors.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
IMPLIED
 * OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 * ARM SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *

**************************************************************************
*/



#include <stdint.h>


/* define compiler specific symbols */
#if defined   ( __CC_ARM   )
  #define __ASM            __asm          /*!< asm keyword for armcc
*/
  #define __INLINE         __inline       /*!< inline keyword for armcc
*/

#elif defined ( __ICCARM__ )
  #define __ASM           __asm           /*!< asm keyword for iarcc
*/
  #define __INLINE        inline          /*!< inline keyword for iarcc.
Only avaiable in High optimization mode! */

#elif defined (  __GNUC__  )
  #define __ASM             __asm         /*!< asm keyword for gcc
*/
  #define __INLINE          inline        /*!< inline keyword for gcc
*/

#elif defined   (  __TASKING__  )
```

```
  #define __ASM            __asm           /*!< asm keyword for TASKING
Compiler          */
  #define __INLINE         inline          /*!< inline keyword for TASKING
Compiler        */

#endif



#if defined ( __CC_ARM   ) /*------------------RealView Compiler ------------
-----*/

/**
 * @brief  Return the Process Stack Pointer
 *
 * @param  none
 * @return uint32_t ProcessStackPointer
 *
 * Return the actual process stack pointer
 */
__ASM uint32_t __get_PSP(void)
{
  mrs r0, psp
  bx lr
}

/**
 * @brief  Set the Process Stack Pointer
 *
 * @param  uint32_t Process Stack Pointer
 * @return none
 *
 * Assign the value ProcessStackPointer to the MSP
 * (process stack pointer) Cortex processor register
 */
__ASM void __set_PSP(uint32_t topOfProcStack)
{
  msr psp, r0
  bx lr
}

/**
 * @brief  Return the Main Stack Pointer
 *
 * @param  none
 * @return uint32_t Main Stack Pointer
 *
 * Return the current value of the MSP (main stack pointer)
 * Cortex processor register
 */
__ASM uint32_t __get_MSP(void)
{
  mrs r0, msp
  bx lr
}

/**
```

```
 * @brief  Set the Main Stack Pointer
 *
 * @param  uint32_t Main Stack Pointer
 * @return none
 *
 * Assign the value mainStackPointer to the MSP
 * (main stack pointer) Cortex processor register
 */
__ASM void __set_MSP(uint32_t mainStackPointer)
{
  msr msp, r0
  bx lr
}

/**
 * @brief  Reverse byte order in unsigned short value
 *
 * @param  uint16_t value to reverse
 * @return uint32_t reversed value
 *
 * Reverse byte order in unsigned short value
 */
__ASM uint32_t __REV16(uint16_t value)
{
  rev16 r0, r0
  bx lr
}

/**
 * @brief  Reverse byte order in signed short value with sign extension to
integer
 *
 * @param  int16_t value to reverse
 * @return int32_t reversed value
 *
 * Reverse byte order in signed short value with sign extension to integer
 */
__ASM int32_t __REVSH(int16_t value)
{
  revsh r0, r0
  bx lr
}


#if (__ARMCC_VERSION < 400000)

/**
 * @brief  Remove the exclusive lock created by ldrex
 *
 * @param  none
 * @return none
 *
 * Removes the exclusive lock which is created by ldrex.
 */
__ASM void __CLREX(void)
{
  clrex
```

```
}

/**
 * @brief  Return the Base Priority value
 *
 * @param  none
 * @return uint32_t BasePriority
 *
 * Return the content of the base priority register
 */
__ASM uint32_t  __get_BASEPRI(void)
{
  mrs r0, basepri
  bx lr
}

/**
 * @brief  Set the Base Priority value
 *
 * @param  uint32_t BasePriority
 * @return none
 *
 * Set the base priority register
 */
__ASM void __set_BASEPRI(uint32_t basePri)
{
  msr basepri, r0
  bx lr
}

/**
 * @brief  Return the Priority Mask value
 *
 * @param  none
 * @return uint32_t PriMask
 *
 * Return the state of the priority mask bit from the priority mask
 * register
 */
__ASM uint32_t __get_PRIMASK(void)
{
  mrs r0, primask
  bx lr
}

/**
 * @brief  Set the Priority Mask value
 *
 * @param  uint32_t PriMask
 * @return none
 *
 * Set the priority mask bit in the priority mask register
 */
__ASM void __set_PRIMASK(uint32_t priMask)
{
  msr primask, r0
  bx lr
```

```
}

/**
 * @brief  Return the Fault Mask value
 *
 * @param  none
 * @return uint32_t FaultMask
 *
 * Return the content of the fault mask register
 */
__ASM uint32_t  __get_FAULTMASK(void)
{
  mrs r0, faultmask
  bx lr
}

/**
 * @brief  Set the Fault Mask value
 *
 * @param  uint32_t faultMask value
 * @return none
 *
 * Set the fault mask register
 */
__ASM void __set_FAULTMASK(uint32_t faultMask)
{
  msr faultmask, r0
  bx lr
}

/**
 * @brief  Return the Control Register value
 *
 * @param  none
 * @return uint32_t Control value
 *
 * Return the content of the control register
 */
__ASM uint32_t  __get_CONTROL(void)
{
  mrs r0, control
  bx lr
}

/**
 * @brief  Set the Control Register value
 *
 * @param  uint32_t Control value
 * @return none
 *
 * Set the control register
 */
__ASM void __set_CONTROL(uint32_t control)
{
  msr control, r0
  bx lr
}
```

```
#endif /* __ARMCC_VERSION  */


#elif (defined (__ICCARM__)) /*------------------ ICC Compiler --------------
-----*/
#pragma diag_suppress=Pe940

/**
 * @brief  Return the Process Stack Pointer
 *
 * @param  none
 * @return uint32_t ProcessStackPointer
 *
 * Return the actual process stack pointer
 */
uint32_t __get_PSP(void)
{
  __ASM("mrs r0, psp");
  __ASM("bx lr");
}

/**
 * @brief  Set the Process Stack Pointer
 *
 * @param  uint32_t Process Stack Pointer
 * @return none
 *
 * Assign the value ProcessStackPointer to the MSP
 * (process stack pointer) Cortex processor register
 */
void __set_PSP(uint32_t topOfProcStack)
{
  __ASM("msr psp, r0");
  __ASM("bx lr");
}

/**
 * @brief  Return the Main Stack Pointer
 *
 * @param  none
 * @return uint32_t Main Stack Pointer
 *
 * Return the current value of the MSP (main stack pointer)
 * Cortex processor register
 */
uint32_t __get_MSP(void)
{
  __ASM("mrs r0, msp");
  __ASM("bx lr");
}

/**
 * @brief  Set the Main Stack Pointer
 *
 * @param  uint32_t Main Stack Pointer
 * @return none
```

```
 *
 * Assign the value mainStackPointer to the MSP
 * (main stack pointer) Cortex processor register
 */
void __set_MSP(uint32_t topOfMainStack)
{
  __ASM("msr msp, r0");
  __ASM("bx lr");
}

/**
 * @brief  Reverse byte order in unsigned short value
 *
 * @param  uint16_t value to reverse
 * @return uint32_t reversed value
 *
 * Reverse byte order in unsigned short value
 */
uint32_t __REV16(uint16_t value)
{
  __ASM("rev16 r0, r0");
  __ASM("bx lr");
}

/**
 * @brief  Reverse bit order of value
 *
 * @param  uint32_t value to reverse
 * @return uint32_t reversed value
 *
 * Reverse bit order of value
 */
uint32_t __RBIT(uint32_t value)
{
  __ASM("rbit r0, r0");
  __ASM("bx lr");
}

/**
 * @brief  LDR Exclusive
 *
 * @param  uint8_t* address
 * @return uint8_t value of (*address)
 *
 * Exclusive LDR command
 */
uint8_t __LDREXB(uint8_t *addr)
{
  __ASM("ldrexb r0, [r0]");
  __ASM("bx lr");
}

/**
 * @brief  LDR Exclusive
 *
 * @param  uint16_t* address
 * @return uint16_t value of (*address)
```

```
 *
 * Exclusive LDR command
 */
uint16_t __LDREXH(uint16_t *addr)
{
  __ASM("ldrexh r0, [r0]");
  __ASM("bx lr");
}

/**
 * @brief  LDR Exclusive
 *
 * @param  uint32_t* address
 * @return uint32_t value of (*address)
 *
 * Exclusive LDR command
 */
uint32_t __LDREXW(uint32_t *addr)
{
  __ASM("ldrex r0, [r0]");
  __ASM("bx lr");
}

/**
 * @brief  STR Exclusive
 *
 * @param  uint8_t *address
 * @param  uint8_t value to store
 * @return uint32_t successful / failed
 *
 * Exclusive STR command
 */
uint32_t __STREXB(uint8_t value, uint8_t *addr)
{
  __ASM("strexb r0, r0, [r1]");
  __ASM("bx lr");
}

/**
 * @brief  STR Exclusive
 *
 * @param  uint16_t *address
 * @param  uint16_t value to store
 * @return uint32_t successful / failed
 *
 * Exclusive STR command
 */
uint32_t __STREXH(uint16_t value, uint16_t *addr)
{
  __ASM("strexh r0, r0, [r1]");
  __ASM("bx lr");
}

/**
 * @brief  STR Exclusive
 *
 * @param  uint32_t *address
```

```
 * @param  uint32_t value to store
 * @return uint32_t successful / failed
 *
 * Exclusive STR command
 */
uint32_t __STREXW(uint32_t value, uint32_t *addr)
{
  __ASM("strex r0, r0, [r1]");
  __ASM("bx lr");
}

#pragma diag_default=Pe940


#elif (defined (__GNUC__)) /*------------------ GNU Compiler ----------------
-----*/

/**
 * @brief  Return the Process Stack Pointer
 *
 * @param  none
 * @return uint32_t ProcessStackPointer
 *
 * Return the actual process stack pointer
 */
uint32_t __get_PSP(void) __attribute__( ( naked ) );
uint32_t __get_PSP(void)
{
  uint32_t result=0;

  __ASM volatile ("MRS %0, psp\n\t"
                  "MOV r0, %0 \n\t"
                  "BX  lr     \n\t"  : "=r" (result) );
  return(result);
}


/**
 * @brief  Set the Process Stack Pointer
 *
 * @param  uint32_t Process Stack Pointer
 * @return none
 *
 * Assign the value ProcessStackPointer to the MSP
 * (process stack pointer) Cortex processor register
 */
void __set_PSP(uint32_t topOfProcStack) __attribute__( ( naked ) );
void __set_PSP(uint32_t topOfProcStack)
{
  __ASM volatile ("MSR psp, %0\n\t"
                  "BX  lr     \n\t" : : "r" (topOfProcStack) );
}

/**
 * @brief  Return the Main Stack Pointer
 *
 * @param  none
```

```
 * @return uint32_t Main Stack Pointer
 *
 * Return the current value of the MSP (main stack pointer)
 * Cortex processor register
 */
uint32_t __get_MSP(void) __attribute__( ( naked ) );
uint32_t __get_MSP(void)
{
  uint32_t result=0;

  __ASM volatile ("MRS %0, msp\n\t"
                  "MOV r0, %0 \n\t"
                  "BX  lr     \n\t"  : "=r" (result) );
  return(result);
}

/**
 * @brief  Set the Main Stack Pointer
 *
 * @param  uint32_t Main Stack Pointer
 * @return none
 *
 * Assign the value mainStackPointer to the MSP
 * (main stack pointer) Cortex processor register
 */
void __set_MSP(uint32_t topOfMainStack) __attribute__( ( naked ) );
void __set_MSP(uint32_t topOfMainStack)
{
  __ASM volatile ("MSR msp, %0\n\t"
                  "BX  lr     \n\t" : : "r" (topOfMainStack) );
}

/**
 * @brief  Return the Base Priority value
 *
 * @param  none
 * @return uint32_t BasePriority
 *
 * Return the content of the base priority register
 */
uint32_t __get_BASEPRI(void)
{
  uint32_t result=0;

  __ASM volatile ("MRS %0, basepri_max" : "=r" (result) );
  return(result);
}

/**
 * @brief  Set the Base Priority value
 *
 * @param  uint32_t BasePriority
 * @return none
 *
 * Set the base priority register
 */
void __set_BASEPRI(uint32_t value)
```

```
{
  __ASM volatile ("MSR basepri, %0" : : "r" (value) );
}

/**
 * @brief  Return the Priority Mask value
 *
 * @param  none
 * @return uint32_t PriMask
 *
 * Return the state of the priority mask bit from the priority mask
 * register
 */
uint32_t __get_PRIMASK(void)
{
  uint32_t result=0;

  __ASM volatile ("MRS %0, primask" : "=r" (result) );
  return(result);
}

/**
 * @brief  Set the Priority Mask value
 *
 * @param  uint32_t PriMask
 * @return none
 *
 * Set the priority mask bit in the priority mask register
 */
void __set_PRIMASK(uint32_t priMask)
{
  __ASM volatile ("MSR primask, %0" : : "r" (priMask) );
}

/**
 * @brief  Return the Fault Mask value
 *
 * @param  none
 * @return uint32_t FaultMask
 *
 * Return the content of the fault mask register
 */
uint32_t __get_FAULTMASK(void)
{
  uint32_t result=0;

  __ASM volatile ("MRS %0, faultmask" : "=r" (result) );
  return(result);
}

/**
 * @brief  Set the Fault Mask value
 *
 * @param  uint32_t faultMask value
 * @return none
 *
 * Set the fault mask register
```

```
 */
void __set_FAULTMASK(uint32_t faultMask)
{
  __ASM volatile ("MSR faultmask, %0" : : "r" (faultMask) );
}

/**
 * @brief  Reverse byte order in integer value
 *
 * @param  uint32_t value to reverse
 * @return uint32_t reversed value
 *
 * Reverse byte order in integer value
 */
uint32_t __REV(uint32_t value)
{
  uint32_t result=0;

  __ASM volatile ("rev %0, %1" : "=r" (result) : "r" (value) );
  return(result);
}

/**
 * @brief  Reverse byte order in unsigned short value
 *
 * @param  uint16_t value to reverse
 * @return uint32_t reversed value
 *
 * Reverse byte order in unsigned short value
 */
uint32_t __REV16(uint16_t value)
{
  uint32_t result=0;

  __ASM volatile ("rev16 %0, %1" : "=r" (result) : "r" (value) );
  return(result);
}

/**
 * @brief  Reverse byte order in signed short value with sign extension to
integer
 *
 * @param  int32_t value to reverse
 * @return int32_t reversed value
 *
 * Reverse byte order in signed short value with sign extension to integer
 */
int32_t __REVSH(int16_t value)
{
  uint32_t result=0;

  __ASM volatile ("revsh %0, %1" : "=r" (result) : "r" (value) );
  return(result);
}

/**
 * @brief  Reverse bit order of value
```

```
 *
 * @param  uint32_t value to reverse
 * @return uint32_t reversed value
 *
 * Reverse bit order of value
 */
uint32_t __RBIT(uint32_t value)
{
  uint32_t result=0;

   __ASM volatile ("rbit %0, %1" : "=r" (result) : "r" (value) );
   return(result);
}

/**
 * @brief  LDR Exclusive
 *
 * @param  uint8_t* address
 * @return uint8_t value of (*address)
 *
 * Exclusive LDR command
 */
uint8_t __LDREXB(uint8_t *addr)
{
    uint8_t result=0;

   __ASM volatile ("ldrexb %0, [%1]" : "=r" (result) : "r" (addr) );
   return(result);
}

/**
 * @brief  LDR Exclusive
 *
 * @param  uint16_t* address
 * @return uint16_t value of (*address)
 *
 * Exclusive LDR command
 */
uint16_t __LDREXH(uint16_t *addr)
{
    uint16_t result=0;

   __ASM volatile ("ldrexh %0, [%1]" : "=r" (result) : "r" (addr) );
   return(result);
}

/**
 * @brief  LDR Exclusive
 *
 * @param  uint32_t* address
 * @return uint32_t value of (*address)
 *
 * Exclusive LDR command
 */
uint32_t __LDREXW(uint32_t *addr)
{
    uint32_t result=0;
```

```
   __ASM volatile ("ldrex %0, [%1]" : "=r" (result) : "r" (addr) );
   return(result);
}

/**
 * @brief  STR Exclusive
 *
 * @param  uint8_t *address
 * @param  uint8_t value to store
 * @return uint32_t successful / failed
 *
 * Exclusive STR command
 */
uint32_t __STREXB(uint8_t value, uint8_t *addr)
{
   uint32_t result=0;

   __ASM volatile ("strexb %0, %2, [%1]" : "=r" (result) : "r" (addr), "r"
(value) );
   return(result);
}

/**
 * @brief  STR Exclusive
 *
 * @param  uint16_t *address
 * @param  uint16_t value to store
 * @return uint32_t successful / failed
 *
 * Exclusive STR command
 */
uint32_t __STREXH(uint16_t value, uint16_t *addr)
{
   uint32_t result=0;

   __ASM volatile ("strexh %0, %2, [%1]" : "=r" (result) : "r" (addr), "r"
(value) );
   return(result);
}

/**
 * @brief  STR Exclusive
 *
 * @param  uint32_t *address
 * @param  uint32_t value to store
 * @return uint32_t successful / failed
 *
 * Exclusive STR command
 */
uint32_t __STREXW(uint32_t value, uint32_t *addr)
{
   uint32_t result=0;

   __ASM volatile ("strex %0, %2, [%1]" : "=r" (result) : "r" (addr), "r"
(value) );
   return(result);
```

```
}

/**
 * @brief  Return the Control Register value
 *
 * @param  none
 * @return uint32_t Control value
 *
 * Return the content of the control register
 */
uint32_t __get_CONTROL(void)
{
  uint32_t result=0;

  __ASM volatile ("MRS %0, control" : "=r" (result) );
  return(result);
}

/**
 * @brief  Set the Control Register value
 *
 * @param  uint32_t Control value
 * @return none
 *
 * Set the control register
 */
void __set_CONTROL(uint32_t control)
{
  __ASM volatile ("MSR control, %0" : : "r" (control) );
}

#elif (defined (__TASKING__)) /*------------------ TASKING Compiler ---------
------------*/
/* TASKING carm specific functions */

/*
 * The CMSIS functions have been implemented as intrinsics in the compiler.
 * Please use "carm -?i" to get an up to date list of all instrinsics,
 * Including the CMSIS ones.
 */

#endif
```