THE MINISTRY of EDUCATION and SCIENCE of RUSSIAN FEDERATION

SAMARA STATE AEROSPACE UNIVERSITY

Working with the Embedded Artists LPC2148 evaluation boards

Learner's guide

SAMARA

2011

Compilers: Kudryavtsev Ilya Alexandrovich,
Kornilin Dmitry Vladimirovich

Working with the Embedded Artists LPC2148 evaluation boards = Работа с отладочными платами Embedded Artists LPC2148 [Electronic resource] : Learner’s guide / The Ministry of Education and Science of Russian Federation, Samara State Aerospace University; compilers I. A. Kudryavtsev, D. V. Kornilin. - Electronic text and graphic data (0,56 Mb). - Samara, 2011. - 1 CD-ROM.

Learner’s guide describes the problems of creating and debugging programs in C / C++ language for NXP ARM7 MCUs and using the evaluation boards LPC2148 Education Boards. Learner’s guide is developed in the Interuniversity Space Research Department. The learner’s guide is intended for the students, studying on the educational program 010900.68 “Applied mathematics and physics”, on the discipline “Radio complexes for flight monitoring and control of the micro/nanosatellites” in A semester.

CONTENTS

INTRODUCTION

During the development of devices based on microcontrollers, there are two interdependent objectives - the development of hardware and software creation. In the process of development may change the program, and circuit devices, which presents certain difficulties, and requires significant time and cost. In order to facilitate the development of widely used ready evaluation boards, which set a microcontroller and a set of standard peripherals. In this paper described the development board LPC2148 Education Boards (from Embedded Artists).

LPC2148 microcontrollers are 32 - bit RISC core with ARM7TDMI-S, has 512 KB of FLASH-memory, 32KB SRAM, as well as a rich set of peripherals. Microcontrollers are equipped with in-circuit programming system based on the JTAG interface and a special boot loader that allows to download the program via UART. Development tools include support for the C/C++, use of which is considered in these guidelines.

Methodical instructions allow students to learn the basics of debugging boards to develop and debug programs microcontroller core ARM. Guidance does not describe the features of the core ARM7TDMI, LPC2148 microcontrollers NXP company or a development environment IAR Embedded Workbench, provides only brief comments, necessary to understand the above code snippet, the appendix contains fragments of circuit debug board, a list of jobs for own work and sample programs.

1 Overview of the debug board

Evaluation Boards such as LPC2148 Education Boards (firm Embedded Artists), have in their composition, apart from a microcontroller, LCD display, buttons, joystick, serial adapter, speaker, stepper motor, LED matrix interface SPI, temperature sensor and potentiometer for working with ADC module, jack FLASH - card module and ZigBee. Also on the board (Figure 1) has all the necessary piping microcontroller (system power and clock) and connectors for interface with a computer and external breadboard.

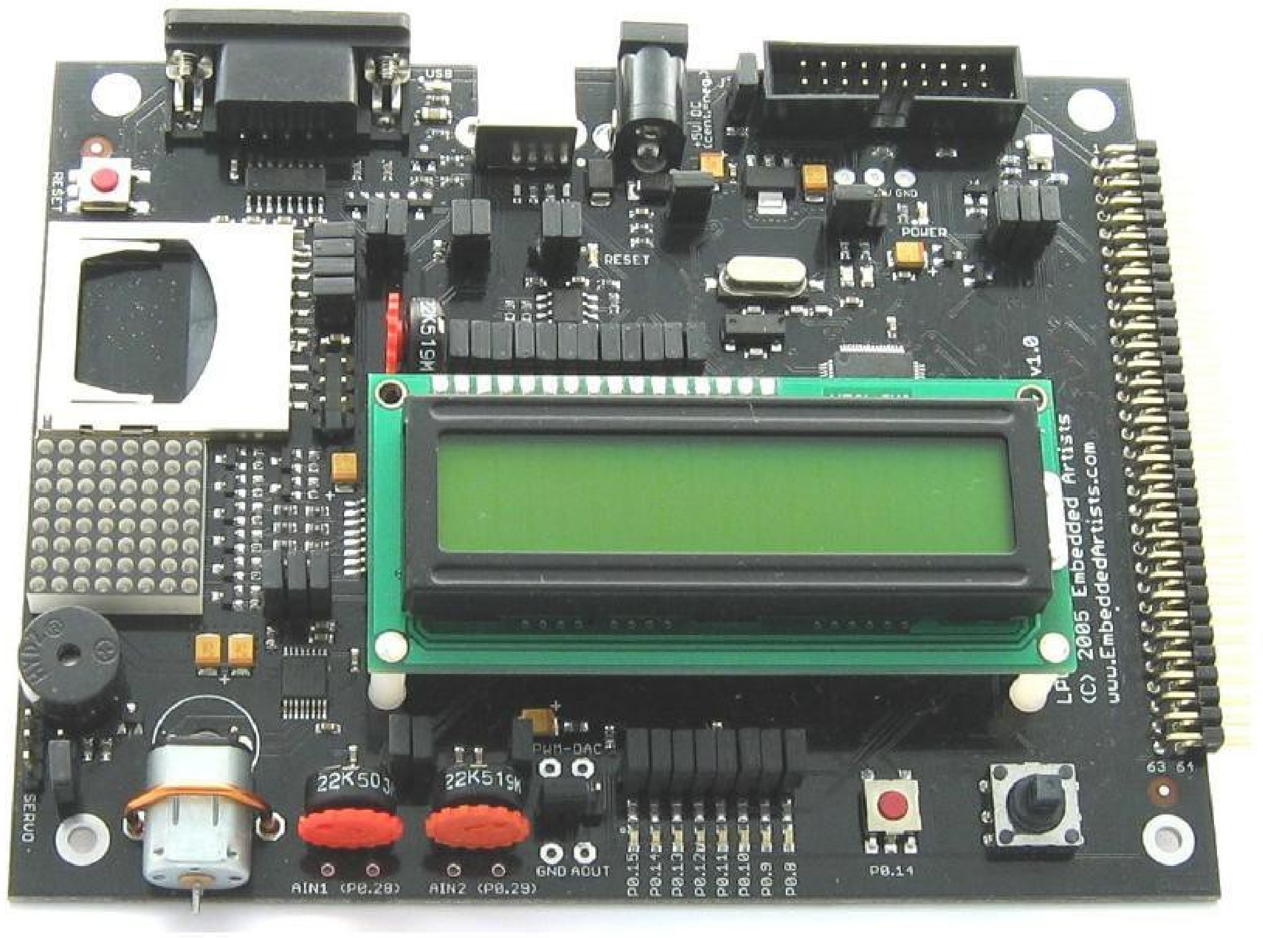


Figure 1 - Appearance of the debug board

2 Compile and debug programs using debugging tools

2.1 Compiling the project

Consider debugging created in the first part of the draft program with the debug board. To do this, make sure that part of the project files added «Main.cpp» and «lcd.cpp», as well as customized configuration of the project «Debug», «Release» and «RAM». Now the project is ready for compiling and debugging.

To assemble and build the object files you can use the menu PROJECT, select the tab where the MAKE or COMPILE. After completion of the compilation with errors or warnings will automatically open with the results of MESSAGES.

2.2 Hardware debugging

After simulation process is successfully completed, it is useful to test the program with the help of the development board. This step of the development process allows to test software/hardware interaction. Attach J-Link module to JTAG connector on the development board and insert the cable to the USB connector on your PC, then connect carefully with the cable USB minijack on the development board (close to power connector to the left of JTAG) with the second USB connector on your PC.

Power LED should confirm the presence of the power on the development board. Check LED blinking on J-Link module.

Set RAM configuration and start the debugger as mentioned earlier then start the program with the help of «Debug/Go» or F5. Check the project operation.

3 Investigation of MCU’s features

3.1 Basic I/O functions

After the project is started, you can see the words on LCD and LED blinking below LCD. Using LCD means setting options of its controller and storing codes of necessary symbols in appropriate location. The details of display’s controller operation you can see in the datasheet, which is placed in Samples\Doc subdirectory. C language offers set of functions for the operation with text strings, prototypes of such functions are described in STDIO.H header file. Detailed description you can find in the Samples\Doc subdirectory. You can also use built-in help system of IAR Embedded Workbench IDE.

This project demonstrates basic options of LPC2148 I/O system with the help of LED blinking and button polling. Development board has signatures, showing which pins of the MCU are connected to the certain LED or button and you can find schematic file in the Doc subdirectory.

3.2 MAM module

MAM accelerates MCU’s operation with FLASH memory with the help of preliminary instruction fetching. The details of MAM operation are described in the manual. The project demonstrates operation of MAM, which is activated/deactivated by the button, connected to P0.14 pin.

Select Release configuration, load the code into MCU’s memory and start the program. Pressing the button you can see performance changes, caused by the MAM operation.

3.3 PLL module

When PLL is not active, MCU’s clock frequency is equal to the frequency of quartz-crystal unit. To use maximum performance, you need to set and activate PLL module. This requires following procedure: Store M and P coefficients in PLLCFG, start PLL, wait until capture is performed, set PLL as a clock signal source for the MCU.

Remove «main.cpp» file from the project (click right mouse button on its name, then select «Remove» from context menu) and add «main_pll.cpp» file. Select RAM configuration, compile the project and start the program. Compare LED blinking period before and after P0.14 button pressing. Determine frequency value of MCU, when PLL is active.

3.4 Interrupt system

Remove «main_pll.cpp» and «lcd.cpp» files from the project and add «main_VIC.cpp» and «lpc2xxx_startup.s» instead. The latter is a standard file, including start code of LPC2148 MCU. This file is written in assembler language and is used for the initializing of interrupt vectors. Usually developers use the copy of this

file with necessary modifications. File «main_VIC.cpp» describes interrupt handlers for IRQ and FIQ interrupts. IRQ will be attached to timer 0, and FIQ will be attached to the external interrupt, activating by P0.14 button press. Details of VIC and timer operation are described in MCU datasheet, placed in the same directory.
Set RAM configuration, build and start program, then open message window «View\Terminal I/O». Look at the output messages, initiated by IRQ handler.

Press the button (P0.14) and look at the result in the terminal window.

Remove from the project file «main_VIC.cpp» and add file «main_VIC_UART.cpp». Carefully put the cable from USB jack to UART connector on the left. Set RAM configuration, build and start program. Start program RS-232, that is intended for RS-232 information interchange, set there baudrate 19200 and select appropriate COM port, set checkbox ASCII. Look at the information, transmitted by development board to the PC in the main window of the program. Change baudrate to 9600 and check the operability.

3.5 SPI module

Remove files «main_VIC.cpp» and «lpc2xxx_startup.s», and add files «main_SPI.cpp», «spi.cpp». Also copy file «spi.h» into the project directory. Set RAM configuration, build and start program. Using joystick (in the right lower corner of the development board) change position of active LED in the matrix. Correct the program to provide diagonal movement. Create your own effect of «LED movement». Schematic diagram of joystick connections is shown in the appendix.

3.6 ADC operation

Remove files «main_SPI.cpp», «spi.cpp», «spi.h» and add files «main_ADC.cpp», «adc.cpp», , «lcd.cpp». Also copy file «main_ADC.h». Instead «lpc2xxx_startup.s» insert file «lpc2xxx_startup_no_fiq.s», where fiq_handler is blocked. This allows to disable FIQ interrupt.

Set RAM configuration, build and start program. Using AIN1 potentiometer adjust motor rotation rate, simultaneously looking at the LCD. Try to determine rotation rate, analyzing the program and derive formula, connecting the number on LCD with actual rotation rate. Make the program to display rotational velocity.

4 Tasks for your own work

The complexity of tasks marked with asterisks.

1. Using existing on-board speaker and DAC module of the microcontroller, create audio-frequency generator with a frequency change of command, the computer is received via the UART (*).

2. Using existing on-board speaker and DAC module of the microcontroller, create audio-frequency oscillator with frequency-controlled potentiometer AIN2. (**)

3. Create a program that writes the data block of 100 bytes in the external EEPROM via I2C (**).

4. Create a program that depicts a snake crawling on dot-matrix display (in the direction of their choice, the length of 6 pixels). (***)

5. Create a program that displays the current temperature (using a sensor on the board) on the LCD display (*).

6. Create a program counter pressing P0.14 displaying decimal result on the LCD. To provide for counter reset (*)

7. Create a program unit that displays the decimal equivalent of the number received by the RS-232 dot matrix display. (**)

8. Create a program of random numbers in the range 0-99 to display on the LCD display in decimal format by pressing the P0.14. (**)

9. Create a program of light effects on matrix display that changes at random position of luminous points. (***)

10. Create a program for a device that displays on the LCD display the contents of the counter milliseconds when you click P0.14. (*)

5 Bibliography

1. Trevor Martin The insider's guide to the Philips ARM7 – BASED MICROCONTROLLERS. An engineer's introduction to the LPC 2000 Series.
2. UM10139 Volume 1: LPC214X User Manual Rev.01 – 15 August 2005.
3. ARM7 TDMI Rev.3 Technical Reference Manual

Appendix A

Source code of programs

«main.cpp» ---

```
#include <NXP/iolpc2148.h>
#include <stdio.h>

int main(void);
void InitLCD(void);
void SetBacklight (int Backlight);
void LCDTextOut (unsigned char const* top_line, unsigned char const* bottom_line);
int Sleep (int _slp);

int main()
{
  InitLCD ();
  SetBacklight (1);
  LCDTextOut ("NXP LPC2148", "Basic project");
  IO0DIR |= 0x0000FF00;        // Set P0.8 - P0.15 as outputs
  IO0SET |= 0x0000FF00;        // Clear P0.8 - P0.15
  IO0DIR &= 0xFFFFBFFF;        // P0.14 - input
  MAMCR=0;
  MAMTIM=3;
  int i=8;
  while(1)
  {
  IO0CLR |=(1<<i);             // shift LEDs
  Sleep(200);
  IO0SET |=(1<<i);
  Sleep(200);
  if(++i==14) i=8;
  if((IO0PIN&0x00004000)==0) MAMCR=2; else MAMCR=0;
  }
}
```

«lcd.cpp» ---

```
#include <NXP\iolpc2148.h>

#define bDISPLAY       16


void SetCommLCD(unsigned char  bT);
void SetDataLCD(unsigned char  bT);
void  WaitReadyLCD(void);
void ShowMsgLCD(unsigned char  bT, unsigned char  const *szT);


unsigned char  szHi[bDISPLAY + 8],szLo[bDISPLAY + 8];

unsigned char  const
      Symbols[0x100] =
      {
         0x20,0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x20,0x20,0x20,0x20,0x20,0x20,0x20,  // 0
         0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,  // 1
```

```
        0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2A,0x2B,0x2C,0x2D,0x2E,0x2F,  // 2
        0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3A,0x3B,0x3C,0x3D,0x3E,0x3F,  // 3
        0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4A,0x4B,0x4C,0x4D,0x4E,0x4F,  // 4
        0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5A,0x5B,0x5C,0x5D,0x5E,0x5F,  // 5
        0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6A,0x6B,0x6C,0x6D,0x6E,0x6F,  // 6
        0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7A,0x7B,0x7C,0x7D,0xC5,0x01,  // 7

        0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,  // 8
        0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,  // 9
        0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,  // A

        0xA2,0xB5,0x20,0x20,0x20,0x20,0x20,0x02,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,  // B
        0x41,0xA0,0x42,0xA1,0xE0,0x45,0xA3,0xA4,0xA5,0xA6,0x4B,0xA7,0x4D,0x48,0x4F,0xA8,  // C
        0x50,0x43,0x54,0xA9,0xAA,0x58,0xE1,0xAB,0xAC,0xE2,0xAD,0xAE,0xC4,0xAF,0xB0,0xB1,   //
D
        0x61,0xB2,0xB3,0xB4,0xE3,0x65,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0x6F,0xBE,   //
E
        0x70,0x63,0xBF,0x79,0xE4,0x78,0xE5,0xC0,0xC1,0xE6,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7   // F
    };


int Sleep (int _slp)
{
        int j;
        for (int i = 0; i < _slp * 1000; i++) j = i + 1;
        return j;
}

void SetCommLCD(unsigned char  bT)
{
        IO1DIR |= (0xFF << 16);         // Set data bus as outputs
        IO1DIR |= (1 << 24);        // Set RS as output
        IO1DIR |= (1 << 25);        // Set E as output
        IO0DIR |= (1 << 30);        // Set Backlight control as output
        IO0DIR |= (1 << 22);        // Set R/W as output

        IO1CLR = (1 << 25);             // E down
        IO1CLR = (1 << 24);             // RS = 0
        IO0CLR = (1 << 22);             // RW = 0
        IO1CLR = (0xFF) << 16;          // Clear data bus
        IO1SET = (bT) << 16;        // Write data to bus
        IO1SET = (1 << 25);             // E up
        Sleep (10);                     // Wait
        IO1CLR = (1 << 25);             // E down
}


void SetDataLCD(unsigned char  bT)
{
        IO1DIR |= (0xFF << 16);         // Set data bus as outputs
        IO1DIR |= (1 << 24);        // Set RS as output
        IO1DIR |= (1 << 25);        // Set E as output
        IO0DIR |= (1 << 30);        // Set Backlight control as output
        IO0DIR |= (1 << 22);        // Set R/W as output

        IO1CLR = (1 << 25);             // E down
        IO1SET = (1 << 24);             // RS = 1
        IO0CLR = (1 << 22);             // RW = 0
        IO1CLR = (0xFF) << 16;          // Clear data bus
        IO1SET = (bT) << 16;        // Write data to bus
        IO1SET = (1 << 25);             // E up
        Sleep (10);                     // Wait
        IO1CLR = (1 << 25);             // E down
}

void  WaitReadyLCD(void)
{
Sleep(1);
}


void ShowMsgLCD(unsigned char  bT, unsigned char  const *szT)
{
        WaitReadyLCD();
    SetCommLCD(bT);
        for (int i = 0; i < bDISPLAY; i++)
        {
        if ( !*szT ) break;
        WaitReadyLCD();
        SetDataLCD( Symbols[*szT++]);
        }
}



void InitLCD(void)
{
        Sleep (20);   SetCommLCD(0x30);
        Sleep (20);   SetCommLCD(0x30);
        Sleep (20);   SetCommLCD(0x30);

        WaitReadyLCD();   SetCommLCD(0x38);
        WaitReadyLCD();   SetCommLCD(0x08);
        WaitReadyLCD();   SetCommLCD(0x01);
        WaitReadyLCD();   SetCommLCD(0x06);
        WaitReadyLCD();   SetCommLCD(0x0C);
        WaitReadyLCD();   SetCommLCD(0x40);
        WaitReadyLCD();   SetCommLCD(0xC4);
}

void LCDTextOut (unsigned char const* top_line, unsigned char const* bottom_line)
{
        ShowMsgLCD(0x80, top_line);
        ShowMsgLCD(0xC0, bottom_line);
        WaitReadyLCD();
    SetCommLCD(0xC4);
}

void SetBacklight (int Backlight)
{
        if (Backlight) IO0SET |= 1 << 30;
        else         IO0CLR |= 1 << 30;
}
```

«main_pll.cpp» ---

```
#include <NXP/iolpc2148.h>
#include <stdio.h>

int main(void);
void InitLCD(void);
void SetBacklight (int Backlight);
void LCDTextOut (unsigned char const* top_line, unsigned char const* bottom_line);
int Sleep (int _slp);

int main()
{
 int LOW=1;

 InitLCD ();
 SetBacklight (1);
 LCDTextOut ("NXP LPC2148", "Basic project");
 IO0DIR |= 0x0000FF00;        // Set P0.8 - P0.15 as outputs
 IO0SET |= 0x0000FF00;         // Clear P0.8 - P0.15
 IO0DIR &= 0xFFFFBFFF;        // P0.14 - input

 int i=8;
 while(1)
 {
 IO0CLR |=(1<<i);             // shift LEDs
 Sleep(200);
 IO0SET |=(1<<i);
 Sleep(200);
 if(++i==14) i=8;
 if((IO0PIN&0x00004000)==0)
 {
  if(LOW==1)
    {
     PLLCFG  = 0x45;                        /* M = 5, P = 2 */
     PLLCON  = 0x01;                        /* PLL Enable */
     PLLFEED = 0xAA;                        /* Feed Sequence 1 */
     PLLFEED = 0x55;                        /* Feed Sequence 2 */
     while ((PLLSTAT & 0x0400) == 0);       /* Wait for PLL Lock */
     PLLCON  = 0x03;                        /* PLL Enable & Connect */
     PLLFEED = 0xAA;                        /* Feed Sequence 1 */
     PLLFEED = 0x55;                        /* Feed Sequence 2 */
     LOW=0;
    }
 }
 else
 {
     if (LOW==0)
     {
      PLLCFG  = 0x23;                       /* M = 3, P = 1 */
      PLLCON  = 0x01;                       /* PLL Enable */
      PLLFEED = 0xAA;                       /* Feed Sequence 1 */
      PLLFEED = 0x55;                       /* Feed Sequence 2 */
      while ((PLLSTAT & 0x0400) == 0);      /* Wait for PLL Lock */
      PLLCON  = 0x03;                       /* PLL Enable & Connect */
      PLLFEED = 0xAA;                       /* Feed Sequence 1 */
      PLLFEED = 0x55;                       /* Feed Sequence 2 */
      LOW=1;
     }
 }
 }
}
```

«main_VIC.cpp» --

```
#include <NXP/iolpc2148.h>
#include <stdio.h>
#include <intrinsics.h>

#define XTALFREQ 12000000         //XTAL frequency in Hz
#define PCLKFREQ (XTALFREQ/4)     //pclk must always be XTALFREQ/4?
#define TICKS_PER_SECOND 20       // TIMER0 interrupt is 100Hz

#define FALSE 0
#define TRUE !(FALSE)

int main(void);
void InitLCD(void);
void SetBacklight (int Backlight);
int Sleep (int _slp);
void LCDTextOut (unsigned char const* top_line, unsigned char const* bottom_line);
extern "C" __fiq __arm void fiq_handler (void);
__irq  __arm  void MM_TIMER0_ISR();

unsigned int Int_Count;
unsigned char buffer[16];
bool bl_TimerFlag;

int main(void)
{
 Int_Count=0;
 InitLCD();
 SetBacklight (1);
 bl_TimerFlag = FALSE;
 VPBDIV_bit.VPBDIV = 0;          // Init Peripherial divider Pckl = Clk/4

 T0IR=0xFF;                    // reset match and capture event interrupts
 T0TC=0;                       // Clear timer counter
 T0PR= 0;                      // No Prescalar
 T0MR0=PCLKFREQ/100;            // Count up to 36,864 for 100Hz interrupt, period = 10ms
 T0MCR = 3;                    // Reset Timer Counter & Interrupt on match
 T0TCR = 1;                    // Counting enable

                        //initialize P0.14 to EINT1 (active falling edge)
 EXTMODE  = 0x00000002;          //EINT1 is edge sensitive
 EXTPOLAR = 0x00000000;          //EINT1 is falling edge sensitive
 PINSEL0 &= ~0x30000000;
 PINSEL0 |=  0x20000000;
 EXTINT   = 0x00000002;         //reset EINT1 IRQ flag

 VICIntSelect  =  0;            // Set all VIC interrupts to IRQ for now
 VICIntEnClear = 0xFFFFFFFF;     // Diasable all interrupts
 VICProtection = 0;              // VIC registers can be accessed in User or
                        // privileged mode
 VICVectAddr = 0;               // Clear interrupt
 VICProtection = 0;              // Accesss VIC in USR | PROTECT
 VICIntSelect |= 0x00008000;                  //EINT1 interrupt is assigned to FIQ (not IRQ)
 VICIntSelect &= ~(1<<VIC_TIMER0);           // Timer 0 intrpt is an IRQ (VIC_TIMER0 = 4)
 VICVectAddr0 = (unsigned int)&MM_TIMER0_ISR; // Install ISR in VIC addr slot 0
 VICVectCntl0 = 0x20 | VIC_TIMER0;           // IRQ type, TIMER 0 int enabled
 VICIntEnable  = 0x00008000;                  //enable eint1 interrupt
 VICIntEnable |= (1<<VIC_TIMER0);            // enable Timer0 Interrupt

 __enable_interrupt();                       // Global interrupt enable


 while(TRUE)                  // Foreground "task"
 {
  if(bl_TimerFlag)
  {
   bl_TimerFlag = FALSE;      // Clear this flag if set by MM_TIMER0_ISR
   printf("IRQ interrupt!\n");
   sprintf((char*)buffer,"Interrupt %d",Int_Count++);
   LCDTextOut("IRQ TEST",buffer);
  }

 }                            // end foreground loop
}                             // end main()


/**************************************************************************
 * Function Name: fiq_handler
 * Parameters: void
 * Return: void
 *
 * Description: FIQ subroutine
* Note: This is ARM mode code - full 32 bit code
 *************************************************************************/
extern "C"__fiq __arm void fiq_handler (void)
{
 printf("FIQ interrupt!\n");
 EXTINT = 0x00000002;         //reset IRQ flag
 VICVectAddr = 0x00;          //dummy write to VIC to signal end of interrupt
}


__irq  __arm void MM_TIMER0_ISR()
{
 static unsigned int us_Ticks=0;
 us_Ticks++;
 if(us_Ticks == TICKS_PER_SECOND)
 {
  bl_TimerFlag = TRUE;        // The background "task"
  us_Ticks = 0;
 }
 T0IR = 1;                    // Clear timer interrupt
 VICVectAddr = 0;
}
```

«lpc2xxx_startup.s»---

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Part one of the system initialization code,
;; contains low-level
;; initialization.
;;
;; Copyright 2006 IAR Systems. All rights reserved.
;;
;; $Revision: 30870 $
;;

        MODULE  ?cstartup

        ;; Forward declaration of sections.
        SECTION IRQ_STACK:DATA:NOROOT(3)
       ;; SECTION FIQ_STACK:DATA:NOROOT(3)
        SECTION ABT_STACK:DATA:NOROOT(3)
        SECTION SVC_STACK:DATA:NOROOT(3)
        SECTION UND_STACK:DATA:NOROOT(3)
        SECTION CSTACK:DATA:NOROOT(3)

;
; The module in this file are included in the libraries, and may be
; replaced by any user-defined modules that define the PUBLIC symbol
; __iar_program_start or a user defined start symbol.
;
; To override the cstartup defined in the library, simply add your
; modified version to the workbench project.

        SECTION .intvec:CODE:NOROOT(2)

        PUBLIC  __vector
        PUBLIC  __vector_0x14
        PUBLIC  __iar_program_start
        EXTERN irq_handler ;;,fiq_handler

        ARM
__vector:
        ;;
        ldr   pc,[pc,#+24]           ;; Reset
        B   .                   ;; Undefined instructions
        B   .                   ;; Software interrupt (SWI/SVC)
        B   .                   ;; Prefetch abort
        B   .                   ;; Data abort
__vector_0x14:
        DC32  0                    ;; RESERVED
        ldr   pc,[pc,#+24]           ;; IRQ
        ldr   pc,[pc,#+24]           ;; FIQ

        DC32  __iar_program_start       ;; Reset
        DC32  0                    ;; Undefined instructions
        DC32  0                    ;; Software interrupt (SWI/SVC)
        DC32  0                    ;; Prefetch abort
        DC32  0                    ;; Data abort
        DC32  0                    ;; RESERVED
        DC32  irq_handler              ;; IRQ
       ;; DC32  fiq_handler              ;; FIQ

; ------------------------------------------------
; ?cstartup -- low-level system initialization code.
;
```

```
; After a reser execution starts here, the mode is ARM, supervisor
; with interrupts disabled.
;



        SECTION .text:CODE:NOROOT(2)

;        PUBLIC  ?cstartup
        EXTERN  ?main
        REQUIRE __vector

        ARM

__iar_program_start:
?cstartup:

;
; Add initialization needed before setup of stackpointers here.
;
; LPC2148 Errata
; Date: August 5, 2005
; Document Release: Version 1.0
; Device Affected: LPC2148
; Incorrect read of data from SRAM after Reset and MAM is not enabled or partially enabled MAM.1
; Init MAM before acsses to SRAM
MAMCR    DEFINE 0xE01FC000      ; MAM Control Register
MAMTIM   DEFINE 0xE01FC004      ; MAM Timing register

                ldr   r0,=MAMCR
                ldr   r1,=MAMTIM
                ldr   r2,=0
                str   r2,[r0]
                ldr   r2,=7
                str   r2,[r1]
                ldr   r2,=2
                str   r2,[r0]

;
; Initialize the stack pointers.
; The pattern below can be used for any of the exception stacks:
; FIQ, IRQ, SVC, ABT, UND, SYS.
; The USR mode uses the same stack as SYS.
; The stack segments must be defined in the linker command file,
; and be declared above.
;
; --------------------
; Mode, correspords to bits 0-5 in CPSR

MODE_MSK DEFINE 0x1F            ; Bit mask for mode bits in CPSR

USR_MODE DEFINE 0x10            ; User mode
;;FIQ_MODE DEFINE 0x11          ; Fast Interrupt Request mode
IRQ_MODE DEFINE 0x12            ; Interrupt Request mode
SVC_MODE DEFINE 0x13            ; Supervisor mode
ABT_MODE DEFINE 0x17            ; Abort mode
UND_MODE DEFINE 0x1B            ; Undefined Instruction mode
SYS_MODE DEFINE 0x1F            ; System mode
```

```
            mrs     r0,cpsr                       ; Original PSR value
            bic     r0,r0,#MODE_MSK                   ; Clear the mode bits
            orr     r0,r0,#SVC_MODE                   ; Set Supervisor mode bits
            msr     cpsr_c,r0                     ; Change the mode
            ldr     sp,=SFE(SVC_STACK)                ; End of SVC_STACK

            bic     r0,r0,#MODE_MSK                   ; Clear the mode bits
            orr     r0,r0,#ABT_MODE                   ; Set Abort mode bits
            msr     cpsr_c,r0                     ; Change the mode
            ldr     sp,=SFE(ABT_STACK)                ; End of ABT_STACK

            bic     r0,r0,#MODE_MSK                   ; Clear the mode bits
            orr     r0,r0,#UND_MODE                   ; Set Undefined mode bits
            msr     cpsr_c,r0                     ; Change the mode
            ldr     sp,=SFE(UND_STACK)                ; End of UND_STACK

;           bic     r0,r0,#MODE_MSK                   ; Clear the mode bits
 ;          orr     r0,r0,#FIQ_MODE                   ; Set FIR mode bits
  ;         msr     cpsr_c,r0                     ; Change the mode
   ;        ldr     sp,=SFE(FIQ_STACK)                ; End of FIR_STACK

            bic     r0,r0,#MODE_MSK                   ; Clear the mode bits
            orr     r0,r0,#IRQ_MODE                   ; Set IRQ mode bits
            msr     cpsr_c,r0                     ; Change the mode
            ldr     sp,=SFE(IRQ_STACK)                ; End of IRQ_STACK

            bic     r0,r0,#MODE_MSK                   ; Clear the mode bits
            orr     r0,r0,#SYS_MODE                   ; Set System mode bits
            msr     cpsr_c,r0                     ; Change the mode
            ldr     sp,=SFE(CSTACK)                   ; End of CSTACK


#ifdef __ARMVFP__
; Enable the VFP coprocessor.
            mov     r0, #0x40000000               ; Set EN bit in VFP
            fmxr    fpexc, r0                     ; FPEXC, clear others.

; Disable underflow exceptions by setting flush to zero mode.
; For full IEEE 754 underflow compliance this code should be removed
; and the appropriate exception handler installed.
            mov     r0, #0x01000000           ; Set FZ bit in VFP
            fmxr    fpscr, r0                 ; FPSCR, clear others.
#endif

; Add more initialization here


; Continue to ?main for more IAR specific system startup

            ldr     r0,=?main
            bx      r0

        END
```

«main_VIC_UART.cpp»---

```
/****************************************************************************
```

```
Minimal code for setting up Timer0 to interrupt on compare match on channel 0
to interrupt at 100Hz
XTALFREQ 12000000        //XTAL frequency in Hz
PCLKFREQ (XTALFREQ/4)    //pclk must always be XTALFREQ/4?

Open terminal I/O window in debugger using View/Terminal I/O in C-SPY to see
VICVectAddr value in exception handler. This is not routed to the serial port
because UARTx is not configured and no implementation of putchar()

***************************************************************************/
#include <NXP/iolpc2148.h>
#include <stdio.h>
#include <intrinsics.h>

#define TICKS_PER_SECOND 100      // TIMER0 interrupt is 100Hz
#define XTALFREQ 12000000         //XTAL frequency in Hz
#define PCLKFREQ (XTALFREQ/4)     //pclk must always be XTALFREQ/4
#define FALSE 0
#define TRUE !(FALSE)

int main(void);
extern "C" __fiq __arm void fiq_handler (void);
extern "C" __irq __arm void irq_handler (void);
void TransmitString(char* pStr);

bool bl_TimerFlag;

int main(void)
{
 bl_TimerFlag = FALSE;
 VPBDIV_bit.VPBDIV = 0;              // Init Peripherial divider Pckl = Clk/4
 U0FCR=0x07;

                          //Configure UART0 to 19200 baud, 8 bit, 1 stop, no parity
 U0LCR = 0x83;                       // Enable FIFOs whether used or not
                          // U0LCR = 0X80-enable access to divisor
                          // latch bit, necessary for setting baud rate
                          // Eight bits
                          // No parity
                          // One stop bit
 U0DLL = 0x0A;
 U0DLM = 0;
 U0LCR &=0x7F;                       // Disable access to divisor latch

 PINSEL0 = 0x05;

 T0IR=0xFF;                  // reset match and capture event interrupts
 T0TC=0;                     // Clear timer counter
 T0PR= 0;                    // No Prescalar
 T0MR0=PCLKFREQ/100;             // Count up to 36,864 for 100Hz interrupt, period = 10ms
 T0MCR = 3;                  // Reset Timer Counter & Interrupt on match
 T0TCR = 1;                  // Counting enable

                      //initialize P0.14 to EINT1 (active falling edge)
 EXTMODE  = 0x00000002;          //EINT1 is edge sensitive
 EXTPOLAR = 0x00000000;          //EINT1 is falling edge sensitive
 PINSEL0 &= ~0x30000000;
 PINSEL0 |=  0x20000000;
```

```
  EXTINT   = 0x00000002;          //reset EINT1 IRQ flag

  VICIntSelect  =  0;             // Set all VIC interrupts to IRQ for now
  VICIntEnClear = 0xFFFFFFFF;     // Diasable all interrupts
  VICProtection = 0;              // VIC registers can be accessed in User or
                        // privileged mode
  VICVectAddr = 0;                // Clear interrupt
  VICProtection = 0;              // Accesss VIC in USR | PROTECT


  VICIntSelect |= 0x00008000;                  //EINT1 interrupt is assigned to FIQ (not IRQ)
  VICIntSelect &= ~(1<<VIC_TIMER0);            // Timer 0 intrpt is an IRQ (VIC_TIMER0 = 4)
  VICVectAddr0 = (unsigned int)&irq_handler;   // Install ISR in VIC addr slot 0
  VICVectCntl0 = 0x20 | VIC_TIMER0;            // IRQ type, TIMER 0 int enabled
  VICIntEnable  = 0x00008000;                  //enable eint1 interrupt
  VICIntEnable |= (1<<VIC_TIMER0);             // enable Timer0 Interrupt

  __enable_interrupt();                        // Global interrupt enable


  while(TRUE)                                  // Foreground "task"
  {
   if(bl_TimerFlag)
   {
    bl_TimerFlag = FALSE;
    TransmitString("IRQ Interrupt processed");
   }
  }                                            // end foreground loop
}                                              // end main()

/*****************************************************************************
 * Function Name: irq_handler
 * Parameters: void
 * Return: void
 *
 * Description: IRQ exception handler, this will call appropriate isr after
 * reading value out of VICVectAddr
* Note: This is ARM mode code - full 32 bit code
 *****************************************************************************/
extern "C" __irq __arm void irq_handler (void)
{
  static unsigned int us_count=0;
  //us_count++;
  if(us_count++ == TICKS_PER_SECOND)
  {
   us_count = 0;
   bl_TimerFlag = TRUE;
  }
  VICVectAddr = 0;                   // Clear interrupt in VIC
  T0IR = 1;
}

/*************************************************************************
 * Function Name: fiq_handler
 * Parameters: void
 * Return: void
 *
 * Description: FIQ subroutine
* Note: This is ARM mode code - full 32 bit code
 *********************************************************************/
extern "C"__fiq __arm void fiq_handler (void)
{
  TransmitString("FIQ interrupt!");
  EXTINT = 0x00000002;
  VICVectAddr = 0x00;         //dummy write to VIC to signal end of interrupt
}


void TransmitString(char* pStr)
{
  while (*pStr != 0)
  {
   U0THR = *pStr;
   while(!(U0LSR & 0x40));
   pStr++;
  }
}
```

«main_SPI.cpp»---

```
#include <NXP/iolpc2148.h>
#include "spi.h"

int main(void);
int Sleep (int _slp);

int main (void)
{
        int X = 4, Y = 4;
        InitSPI ();

        while (1)
        {
         switch ((IO0PIN & (0x1F << 16)) >> 16)            // Read value from joystick
         {
              case 0x1D:   if (Y < 8) Y++; break;// Up
              case 0x0F:   if (Y > 1) Y--; break;  // Down
              case 0x17:   if (X > 1) X--; break;  // Left
              case 0x1B:     if (X < 8) X++; break;      // Right
          case (0x1D&0x17): if ((Y < 8)&(X > 1)){ Y++; X--;} break;  // UpLeft
          case (0x1D&0x1B): if ((Y < 8)&(X < 8)){ Y++; X++;} break;      // UpRight
          case (0x0F&0x17): if ((Y > 1)&(X > 1)){ Y--; X--;} break;    //DownLeft
          case (0x0F&0x1B): if ((Y > 1)&(X < 8)){ Y--; X++;} break;  // DownRight

              default: break;
         }

         SPIPutDot (X, Y);
         Sleep (300);
        }
}

int Sleep (int _slp)
{
        int j;
        for (int i = 0; i < _slp * 1000; i++) j = i + 1;
        return j;
}
```

«spi.cpp»---

```
#include <NXP/iolpc2148.h>
#include "spi.h"


void InitSPI (void)
{
        IO0DIR |= (1 << 15);                    // Chip select for shift registers
        IO0SET = (1 << 15);
        PINSEL0 |= (1 << 8) | (1 << 10) | (1 << 12);
        S0SPCR =            (0 << 2) |          // set up bits per transfer (0 - 8bit, 1 - defined)
                (1 << 3) |          // CPHA mode
                (1 << 4) |          // SCK is active high
                (1 << 5) |          // Set SPI as master
                (0 << 8);      // 16 bits per transfer
        S0SPCCR = 64;
}


void SPIPutDot (int x, int y)
{
        IO0SET = (1 << 15);                         // Pull up P0.15
        S0SPDR = ~(1 << (y - 1));
        while ((S0SPSR & (1 << 7)) == 0);               // Wait until data is sent
        S0SPDR = ~(1 << (8 - x));
        while ((S0SPSR & (1 << 7)) == 0);               // Wait until data is sent
        IO0CLR = (1 << 15);                         // Pull down P0.15 ("apply new settings")
}
```

«spi.h»---

```
#ifndef SPI_H
#define SPI_H

void InitSPI (void);
void SPIPutDot (int x, int y);

#endif
```

«main_ADC.cpp»---

```
/*****************************************************************************
Minimal code for setting up Timer0 to interrupt on compare match on channel 0
to interrupt at 100Hz
XTALFREQ 12000000       //XTAL frequency in Hz
PCLKFREQ (XTALFREQ/4)    //pclk must always be XTALFREQ/4?

Open terminal I/O window in debugger using View/Terminal I/O in C-SPY to see
VICVectAddr value in exception handler. This is not routed to the serial port
because UARTx is not configured and no implementation of putchar()

*****************************************************************************/
#include <NXP/iolpc2148.h>
#include <stdio.h>
```

```
#include <intrinsics.h>
#include "main_ADC.h"

unsigned int Speed;

int main(void)
{
  unsigned char buffer[32];
  VPBDIV_bit.VPBDIV = 0;          // Init Peripherial divider Pclk = Clk/4

  T0IR=0xFF;                    // reset match and capture event interrupts
  T0TC=0;                       // Clear timer counter
  T0PR= 0;                      // No Prescaler
  T0MR0=1000;                     // Count up to 36,864 for 1000Hz interrupt, period = 1ms
  T0MCR = 3;                    // Reset Timer Counter & Interrupt on match
  T0TCR = 1;                    // Counting enable

  VICIntSelect  =  0;           // Set all VIC interrupts to IRQ for now
  VICIntEnClear = 0xFFFFFFFF;     // Diasable all interrupts
  VICProtection = 0;              // VIC registers can be accessed in User or
                           // privileged mode
  VICVectAddr = 0;                // Clear interrupt
  VICProtection = 0;              // Accesss VIC in USR | PROTECT

  VICIntSelect &= ~(1<<VIC_TIMER0);           // Timer 0 intrpt is an IRQ (VIC_TIMER0 = 4)
  VICVectAddr0 = (unsigned int)&irq_handler;   // Install ISR in VIC addr slot 0
  VICVectCntl0 = 0x20 | VIC_TIMER0;           // IRQ type, TIMER 0 int enabled
  VICIntEnable |= (1<<VIC_TIMER0);            // enable Timer0 Interrupt

  InitADC();
  InitLCD ();
  SetBacklight (1);
  LCDTextOut ("NXP LPC2148", "ADC project");
  IO0DIR |= 0x0020FF00;        // Set P0.8 - P0.15,P0.21 as outputs
  IO0SET |= 0x0020FF00;         // Clear P0.8 - P0.15,P0.21
  __enable_interrupt();                  // Global interrupt enable


  while(TRUE)
  {
    Speed=1023-ADCReadValue();
    if(Speed<23) Speed=23;
    sprintf((char*)buffer,"Speed=%d  ",1023-Speed);
    LCDTextOut ("NXP LPC2148", buffer);
    Sleep(100);
  }
}

/*************************************************************************
 * Function Name: irq_handler
 * Parameters: void
 * Return: void
 *
 * Description: IRQ exception handler, this will call appropriate isr after
 * reading value out of VICVectAddr
* Note: This is ARM mode code - full 32 bit code
 *************************************************************************/
extern "C" __irq __arm void irq_handler (void)
```

```
{
 const int A[12]={1,1,0,0,1,1,0,0,1,1,0,0};
 const int B[12]={0,1,1,0,0,1,1,0,0,1,1,0};
 static unsigned int us_Ticks,n;
 us_Ticks++;
 if(us_Ticks >= Speed)
 {
  us_Ticks = 0;
  if(A[n]==0) IO0CLR|=(1<<21); else IO0SET|=(1<<21);
  if(B[n]==0) IO0CLR|=(1<<12); else IO0SET|=(1<<12);
  if(++n>11) n=0;
 }
 T0IR = 1;                          // Clear timer interrupt
 VICVectAddr = 0;                       // Clear interrupt in VIC
}
```

«adc.cpp»---

```
#include <NXP/iolpc2148.h>

void InitADC (void)
{
        /*********************************************************/
        /*          Connect PIN connect block to ADC0.1          */
        /*********************************************************/
        PINSEL1 |= (1 << 24);

        /*********************************************************/
        /*                 Configure ADC                  */
        /*********************************************************/
        AD0CR =      (1 << 1) |                 // Select ADC0.1 channel is active
               (4 << 8) |                 // Set Clock divider
               (1 << 16) |                // Set BURST
               (0 << 17) |                // Set ADC resolution
               (1 << 21) |                // Power on ADC
               (1 << 24);                 // Start conversion now
}

int ADCReadValue (void)
{
        int ADCValue;

        /*********************************************************/
        /*               Read value from ADC                */
        /*********************************************************/
        while (((ADCValue = AD0DR) & 0x80000000) == 0);

        /*********************************************************/
        /*      Separate ADC value from other information       */
        /*********************************************************/
        return (ADCValue >> 6) & (0x3FF);
}
```

«main_ADC.h»--

```
#define XTALFREQ 12000000        //XTAL frequency in Hz
#define PCLKFREQ (XTALFREQ/4)     //pclk must always be XTALFREQ/4?
```

```
#define FALSE 0
#define TRUE !(FALSE)

int main(void);
void InitLCD(void);
void SetBacklight (int Backlight);
void LCDTextOut (unsigned char const* top_line, unsigned char const* bottom_line);
int Sleep (int _slp);
void InitADC (void);
int ADCReadValue (void);
extern "C" __irq __arm void irq_handler (void);
```

«lpc2xxx_startup_no_fiq.s»--

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Part one of the system initialization code,
;; contains low-level
;; initialization.
;;
;; Copyright 2006 IAR Systems. All rights reserved.
;;
;; $Revision: 30870 $
;;

        MODULE  ?cstartup

        ;; Forward declaration of sections.
        SECTION IRQ_STACK:DATA:NOROOT(3)
        SECTION FIQ_STACK:DATA:NOROOT(3)
        SECTION ABT_STACK:DATA:NOROOT(3)
        SECTION SVC_STACK:DATA:NOROOT(3)
        SECTION UND_STACK:DATA:NOROOT(3)
        SECTION CSTACK:DATA:NOROOT(3)

;
; The module in this file are included in the libraries, and may be
; replaced by any user-defined modules that define the PUBLIC symbol
; __iar_program_start or a user defined start symbol.
;
; To override the cstartup defined in the library, simply add your
; modified version to the workbench project.

        SECTION .intvec:CODE:NOROOT(2)

        PUBLIC  __vector
        PUBLIC  __vector_0x14
        PUBLIC  __iar_program_start
        EXTERN irq_handler

        ARM
__vector:
        ;;
        ldr   pc,[pc,#+24]            ;; Reset
        B   .                    ;; Undefined instructions
        B   .                    ;; Software interrupt (SWI/SVC)
        B   .                    ;; Prefetch abort
        B   .                    ;; Data abort
```

```
__vector_0x14:
      DC32  0                 ;; RESERVED
      ldr   pc,[pc,#-0xFF0]         ;; IRQ
      DC32  0                 ;; FIQ

      DC32  __iar_program_start      ;; Reset
      DC32  0                 ;; Undefined instructions
      DC32  0                 ;; Software interrupt (SWI/SVC)
      DC32  0                 ;; Prefetch abort
      DC32  0                 ;; Data abort
      DC32  0                 ;; RESERVED
      DC32  0                 ;; IRQ
      DC32  0                 ;; FIQ

; ------------------------------------------------
; ?cstartup -- low-level system initialization code.
;
; After a reser execution starts here, the mode is ARM, supervisor
; with interrupts disabled.
;



      SECTION .text:CODE:NOROOT(2)

;       PUBLIC  ?cstartup
      EXTERN  ?main
      REQUIRE __vector

      ARM

__iar_program_start:
?cstartup:

;
; Add initialization needed before setup of stackpointers here.
;
; LPC2148 Errata
; Date: August 5, 2005
; Document Release: Version 1.0
; Device Affected: LPC2148
; Incorrect read of data from SRAM after Reset and MAM is not enabled or partially enabled MAM.1
; Init MAM before acsses to SRAM
MAMCR    DEFINE 0xE01FC000      ; MAM Control Register
MAMTIM   DEFINE 0xE01FC004      ; MAM Timing register

            ldr   r0,=MAMCR
            ldr   r1,=MAMTIM
            ldr   r2,=0
            str   r2,[r0]
            ldr   r2,=7
            str   r2,[r1]
            ldr   r2,=2
            str   r2,[r0]

;
; Initialize the stack pointers.
; The pattern below can be used for any of the exception stacks:
```

```
; FIQ, IRQ, SVC, ABT, UND, SYS.
; The USR mode uses the same stack as SYS.
; The stack segments must be defined in the linker command file,
; and be declared above.
;
; --------------------
; Mode, correspords to bits 0-5 in CPSR

MODE_MSK DEFINE 0x1F            ; Bit mask for mode bits in CPSR

USR_MODE DEFINE 0x10            ; User mode
FIQ_MODE DEFINE 0x11            ; Fast Interrupt Request mode
IRQ_MODE DEFINE 0x12            ; Interrupt Request mode
SVC_MODE DEFINE 0x13            ; Supervisor mode
ABT_MODE DEFINE 0x17            ; Abort mode
UND_MODE DEFINE 0x1B            ; Undefined Instruction mode
SYS_MODE DEFINE 0x1F            ; System mode

        mrs     r0,cpsr                 ; Original PSR value
        bic     r0,r0,#MODE_MSK         ; Clear the mode bits
        orr     r0,r0,#SVC_MODE         ; Set Supervisor mode bits
        msr     cpsr_c,r0               ; Change the mode
        ldr     sp,=SFE(SVC_STACK)      ; End of SVC_STACK

        bic     r0,r0,#MODE_MSK         ; Clear the mode bits
        orr     r0,r0,#ABT_MODE         ; Set Abort mode bits
        msr     cpsr_c,r0               ; Change the mode
        ldr     sp,=SFE(ABT_STACK)      ; End of ABT_STACK

        bic     r0,r0,#MODE_MSK         ; Clear the mode bits
        orr     r0,r0,#UND_MODE         ; Set Undefined mode bits
        msr     cpsr_c,r0               ; Change the mode
        ldr     sp,=SFE(UND_STACK)      ; End of UND_STACK

        bic     r0,r0,#MODE_MSK         ; Clear the mode bits
        orr     r0,r0,#FIQ_MODE         ; Set FIR mode bits
        msr     cpsr_c,r0               ; Change the mode
        ldr     sp,=SFE(FIQ_STACK)      ; End of FIR_STACK

        bic     r0,r0,#MODE_MSK         ; Clear the mode bits
        orr     r0,r0,#IRQ_MODE         ; Set IRQ mode bits
        msr     cpsr_c,r0               ; Change the mode
        ldr     sp,=SFE(IRQ_STACK)      ; End of IRQ_STACK

        bic     r0,r0,#MODE_MSK         ; Clear the mode bits
        orr     r0,r0,#SYS_MODE         ; Set System mode bits
        msr     cpsr_c,r0               ; Change the mode
        ldr     sp,=SFE(CSTACK)         ; End of CSTACK


#ifdef __ARMVFP__
; Enable the VFP coprocessor.
        mov     r0, #0x40000000         ; Set EN bit in VFP
        fmxr    fpexc, r0               ; FPEXC, clear others.

; Disable underflow exceptions by setting flush to zero mode.
; For full IEEE 754 underflow compliance this code should be removed
; and the appropriate exception handler installed.
```

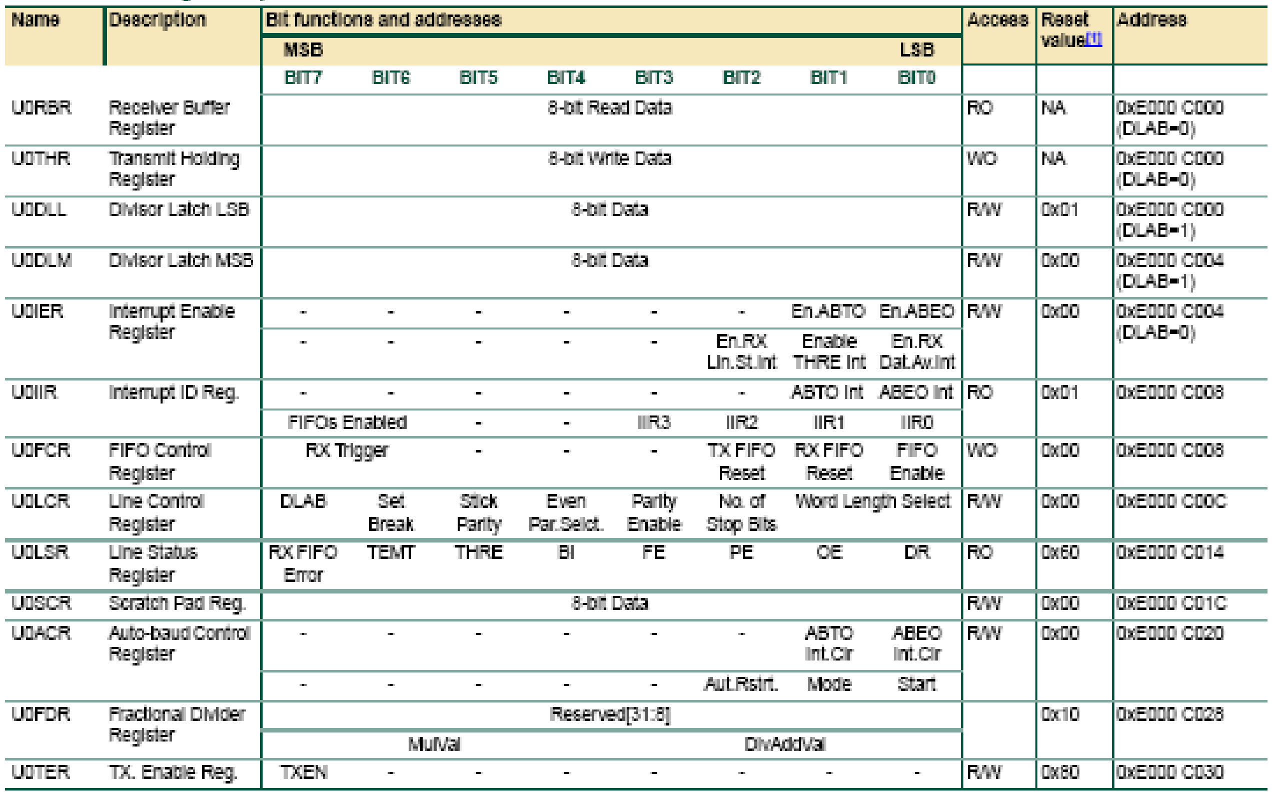
```
        mov    r0, #0x01000000      ; Set FZ bit in VFP
        fmxr   fpscr, r0            ; FPSCR, clear others.
#endif

; Add more initialization here


; Continue to ?main for more IAR specific system startup

        ldr    r0,=?main
        bx     r0

    END
```

Appendix B

Table B.1 IO ports control registers

Name	description	access	State after reset
IOXPIN	Designed to read the state of pin	R/W	-
IOXSET	Recording 1 set high logic level at pin	R/W	0x00000000
IOXCLR	Recording 1 set low logic level at pin	R/W	0x00000000
IOXDIR	The direction of transmission. Recording 1 configures the output mode to output	R/W	0x00000000

Table C.1 UART0 Registers



Name	Description	Bit functions and addresses								Access	Reset value[1]	Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
U0RBR	Receiver Buffer Register	8-bit Read Data								RO	NA	0xE000 C000 (DLAB=0)
U0THR	Transmit Holding Register	8-bit Write Data								WO	NA	0xE000 C000 (DLAB=0)
U0DLL	Divisor Latch LSB	8-bit Data								R/W	0x01	0xE000 C000 (DLAB=1)
U0DLM	Divisor Latch MSB	8-bit Data								R/W	0x00	0xE000 C004 (DLAB=1)
U0IER	Interrupt Enable Register	-	-	-	-	-	-	En.ABTO	En.ABEO	R/W	0x00	0xE000 C004 (DLAB=0)
		-	-	-	-	-	En.RX Lin.St.Int	Enable THRE Int	En.RX Dat.Av.Int			
U0IIR	Interrupt ID Reg.	-	-	-	-	-	-	ABTO Int	ABEO Int	RO	0x01	0xE000 C008
		FIFOs Enabled		-	-	IIR3	IIR2	IIR1	IIR0			
U0FCR	FIFO Control Register	RX Trigger		-	-	-	TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE000 C008
U0LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Par.Selct.	Parity Enable	No. of Stop Bits	Word Length Select		R/W	0x00	0xE000 C00C
U0LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE000 C014
U0SCR	Scratch Pad Reg.	8-bit Data								R/W	0x00	0xE000 C01C
U0ACR	Auto-baud Control Register	-	-	-	-	-	-	ABTO Int.Clr	ABEO Int.Clr	R/W	0x00	0xE000 C020
		-	-	-	-	-	Aut.Rstrt.	Mode	Start			
U0FDR	Fractional Divider Register	Reserved[31:8]									0x10	0xE000 C028
		MulVal				DivAddVal						
U0TER	TX. Enable Reg.	TXEN	-	-	-	-	-	-	-	R/W	0x80	0xE000 C030

The formula for calculating the UART transmission rate

$$UART0_{baudrate} = \frac{PCLK}{16 \times (16 \times U0DLM + U0DLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

The main registers:
U0RSR - register the received data;
U0THR - data register for transmission;
U0IER - Interrupt Enable UART:
Bit 0 - 1 - enable interrupt if the received data;
Bit 1 - 1 - enable interrupt when buffer under the program;
Bit 2 - 1 - enable interrupt when a particular state line RX
U0LSR - line control status register: Configures the format make [3];
U0LSR - line status register: Current status of the port (error)

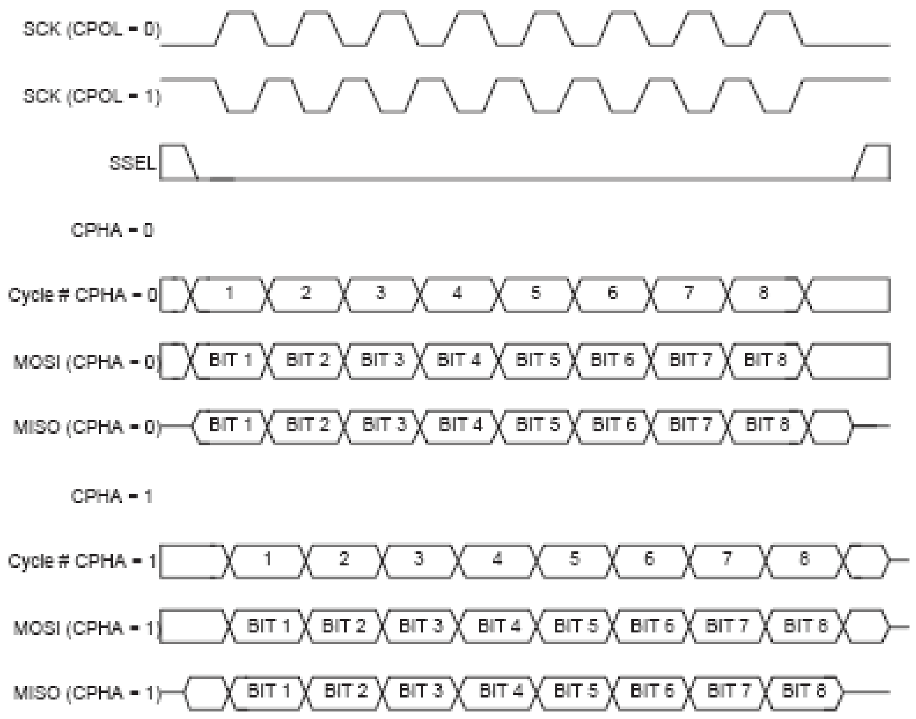


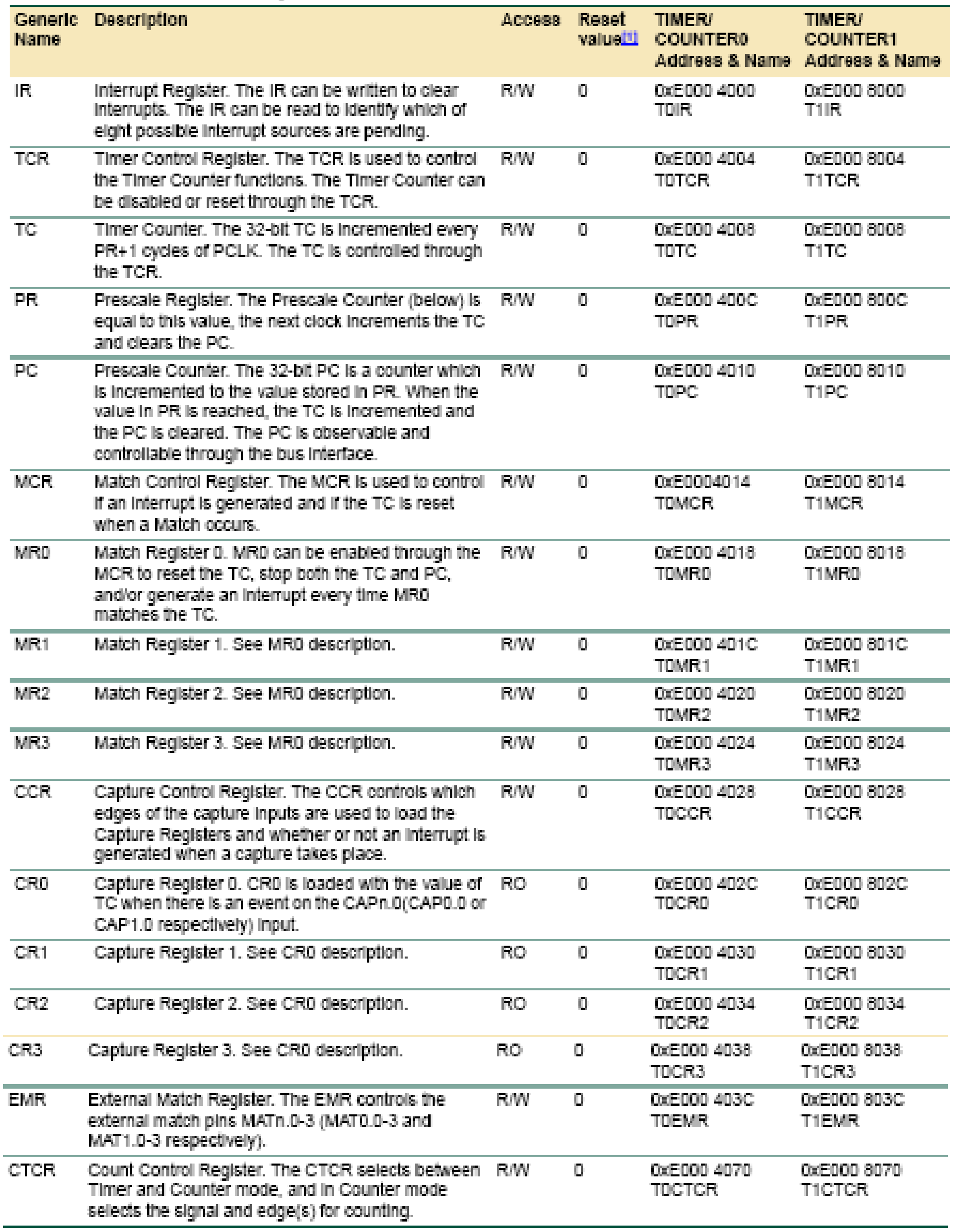
Figure C.1 - Formats the transfer module in SPI

Table D.1 SPI module registers

Name	Description	Access	Reset value[1]	Address
S0SPCR	SPI Control Register. This register controls the operation of the SPI.	R/W	0x00	0xE002 0000
S0SPSR	SPI Status Register. This register shows the status of the SPI.	RO	0x00	0xE002 0004
S0SPDR	SPI Data Register. This bi-directional register provides the transmit and receive data for the SPI. Transmit data is provided to the SPI0 by writing to this register. Data received by the SPI0 can be read from this register.	R/W	0x00	0xE002 0008
S0SPCCR	SPI Clock Counter Register. This register controls the frequency of a master's SCK0.	R/W	0x00	0xE002 000C
S0SPINT	SPI Interrupt Flag. This register contains the interrupt flag for the SPI interface.	R/W	0x00	0xE002 001C

The most important registers are:
S0SPCR - control register, the format detailed in [3];
S0SPSR - status register reflects the current state (error);
S0SPDR - register containing the transmitted and received data;
S0SPCCR-register control the frequency transmit mode MASTER.

Table E.1 The timer registers



Generic Name	Description	Access	Reset value[1]	TIMER/ COUNTER0 Address & Name	TIMER/ COUNTER1 Address & Name
IR	Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.	R/W	0	0xE000 4000 T0IR	0xE000 8000 T1IR
TCR	Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	R/W	0	0xE000 4004 T0TCR	0xE000 8004 T1TCR
TC	Timer Counter. The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	R/W	0	0xE000 4008 T0TC	0xE000 8008 T1TC
PR	Prescale Register. The Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	R/W	0	0xE000 400C T0PR	0xE000 800C T1PR
PC	Prescale Counter. The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface.	R/W	0	0xE000 4010 T0PC	0xE000 8010 T1PC
MCR	Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	R/W	0	0xE0004014 T0MCR	0xE000 8014 T1MCR
MR0	Match Register 0. MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC.	R/W	0	0xE000 4018 T0MR0	0xE000 8018 T1MR0
MR1	Match Register 1. See MR0 description.	R/W	0	0xE000 401C T0MR1	0xE000 801C T1MR1
MR2	Match Register 2. See MR0 description.	R/W	0	0xE000 4020 T0MR2	0xE000 8020 T1MR2
MR3	Match Register 3. See MR0 description.	R/W	0	0xE000 4024 T0MR3	0xE000 8024 T1MR3
CCR	Capture Control Register. The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated when a capture takes place.	R/W	0	0xE000 4028 T0CCR	0xE000 8028 T1CCR
CR0	Capture Register 0. CR0 is loaded with the value of TC when there is an event on the CAPn.0(CAP0.0 or CAP1.0 respectively) input.	RO	0	0xE000 402C T0CR0	0xE000 802C T1CR0
CR1	Capture Register 1. See CR0 description.	RO	0	0xE000 4030 T0CR1	0xE000 8030 T1CR1
CR2	Capture Register 2. See CR0 description.	RO	0	0xE000 4034 T0CR2	0xE000 8034 T1CR2
CR3	Capture Register 3. See CR0 description.	RO	0	0xE000 4038 T0CR3	0xE000 8038 T1CR3
EMR	External Match Register. The EMR controls the external match pins MATn.0-3 (MAT0.0-3 and MAT1.0-3 respectively).	R/W	0	0xE000 403C T0EMR	0xE000 803C T1EMR
CTCR	Count Control Register. The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	R/W	0	0xE000 4070 T0CTCR	0xE000 8070 T1CTCR

The main registers: (X - 0 or 1, depending on the timer)
ThTCR - control register;
TxTC - count register;
TxPR - register prescaler;
TxMR0 - register containing the value at which the match interrupt is generated and the counter is reset;
TxMCR - control register mode matching;
TxIR - timer interrupt control register.

Table F.1 ADC registers

Generic Name	Description	Access	Reset value[1]	AD0 Address & Name	AD1 Address & Name
ADCR	A/D Control Register. The ADCR register must be written to select the operating mode before A/D conversion can occur.	R/W	0x0000 0001	0xE003 4000 ADOCR	0xE006 0000 AD1CR
ADGDR	A/D Global Data Register. This register contains the ADC's DONE bit and the result of the most recent A/D conversion.	R/W	NA	0xE003 4004 AD0GDR	0xE006 0004 AD1GDR
ADSTAT	A/D Status Register. This register contains DONE and OVERRUN flags for all of the A/D channels, as well as the A/D interrupt flag.	RO	0x0000 0000	0xE003 4030 AD0STAT	0xE006 0030 AD1STAT
ADGSR	A/D Global Start Register. This address can be written (in the AD0 address range) to start conversions in both A/D converters simultaneously.	WO	0x00	0xE003 4008 ADGSR	
ADINTEN	A/D Interrupt Enable Register. This register contains enable bits that allow the DONE flag of each A/D channel to be included or excluded from contributing to the generation of an A/D interrupt.	R/W	0x0000 0100	0xE003 400C AD0INTEN	0xE006 000C AD1INTEN
ADDR0	A/D Channel 0 Data Register. This register contains the result of the most recent conversion completed on channel 0.	RO	NA	0xE003 4010 AD0DR0	0xE006 0010 AD1DR0
ADDR1	A/D Channel 1 Data Register. This register contains the result of the most recent conversion completed on channel 1.	RO	NA	0xE003 4014 AD0DR1	0xE006 0014 AD1DR1
ADDR2	A/D Channel 2 Data Register. This register contains the result of the most recent conversion completed on channel 2.	RO	NA	0xE003 4018 AD0DR2	0xE006 0018 AD1DR2
ADDR3	A/D Channel 3 Data Register. This register contains the result of the most recent conversion completed on channel 3.	RO	NA	0xE003 401C AD0DR3	0xE006 001C AD1DR3
ADDR4	A/D Channel 4 Data Register. This register contains the result of the most recent conversion completed on channel 4.	RO	NA	0xE003 4020 AD0DR4	0xE006 0020 AD1DR4
ADDR5	A/D Channel 5 Data Register. This register contains the result of the most recent conversion completed on channel 5.	RO	NA	0xE003 4024 AD0DR5	0xE006 0024 AD1DR5
ADDR6	A/D Channel 6 Data Register. This register contains the result of the most recent conversion completed on channel 6.	RO	NA	0xE003 4028 AD0DR6	0xE006 0028 AD1DR6
ADDR7	A/D Channel 7 Data Register. This register contains the result of the most recent conversion completed on channel 7.	RO	NA	0xE003 402C AD0DR7	0xE006 002C AD1DR7

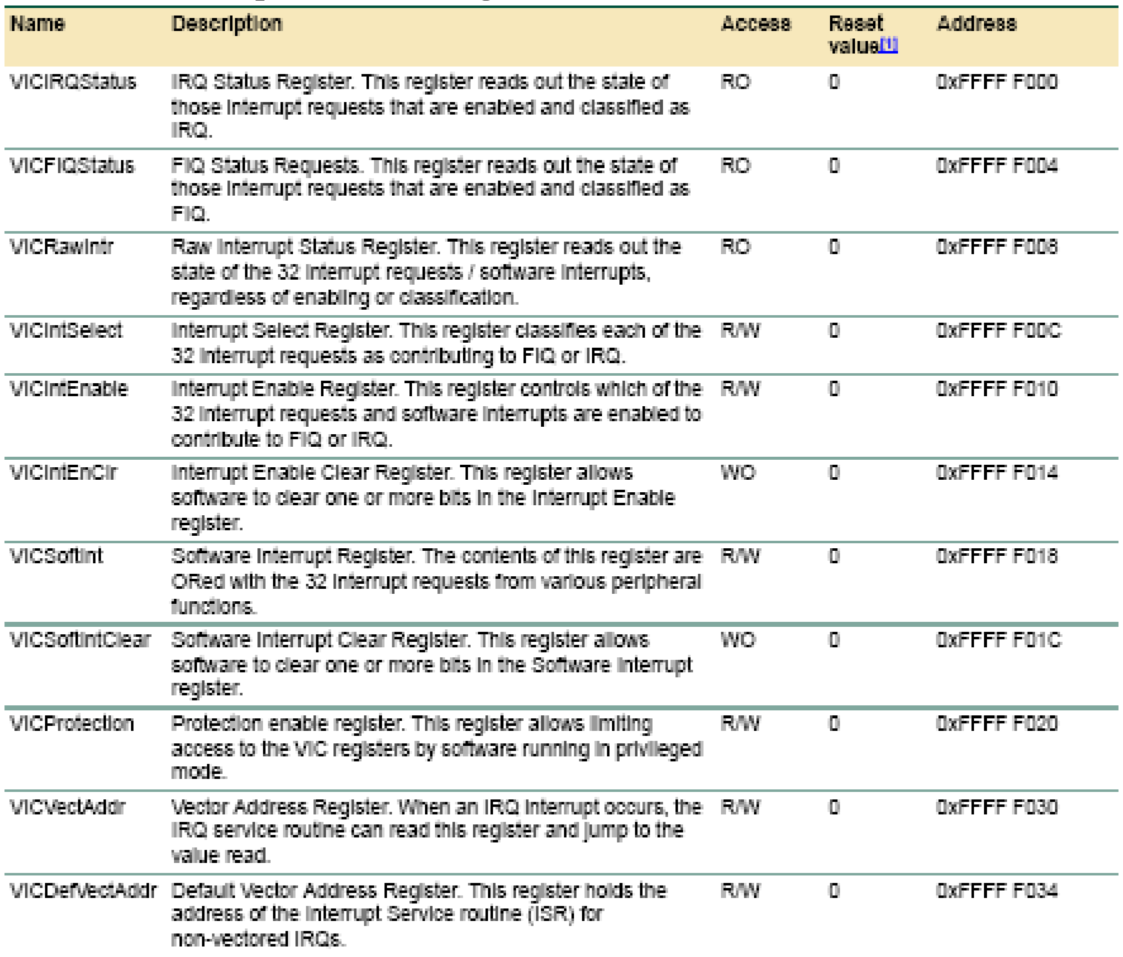
The main registers:
ADCR - control register;
ADGDR - a register containing the result and the last bit of preparedness;
ADSTAT - ADC status register (all channels);
ADDRX - the result register of the channel.

Table E.2 Format Registry DACR (0xE006C000) DAC control

Bit	Symbol	Value	Description	Reset value
5:0	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:6	VALUE		After the selected settling time after this field is written with a new VALUE, the voltage on the A_{OUT} pin (with respect to V_{SSA}) is VALUE/1024 * V_{REF}.	0
16	BIAS	0	The settling time of the DAC is 1 μs max, and the maximum current is 700 uA.	0
		1	The settling time of the DAC is 2.5 μs and the maximum current is 350 μA.	
31:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

To activate the DAC to set bits 19:18 in the state register PINSEL1 "10"

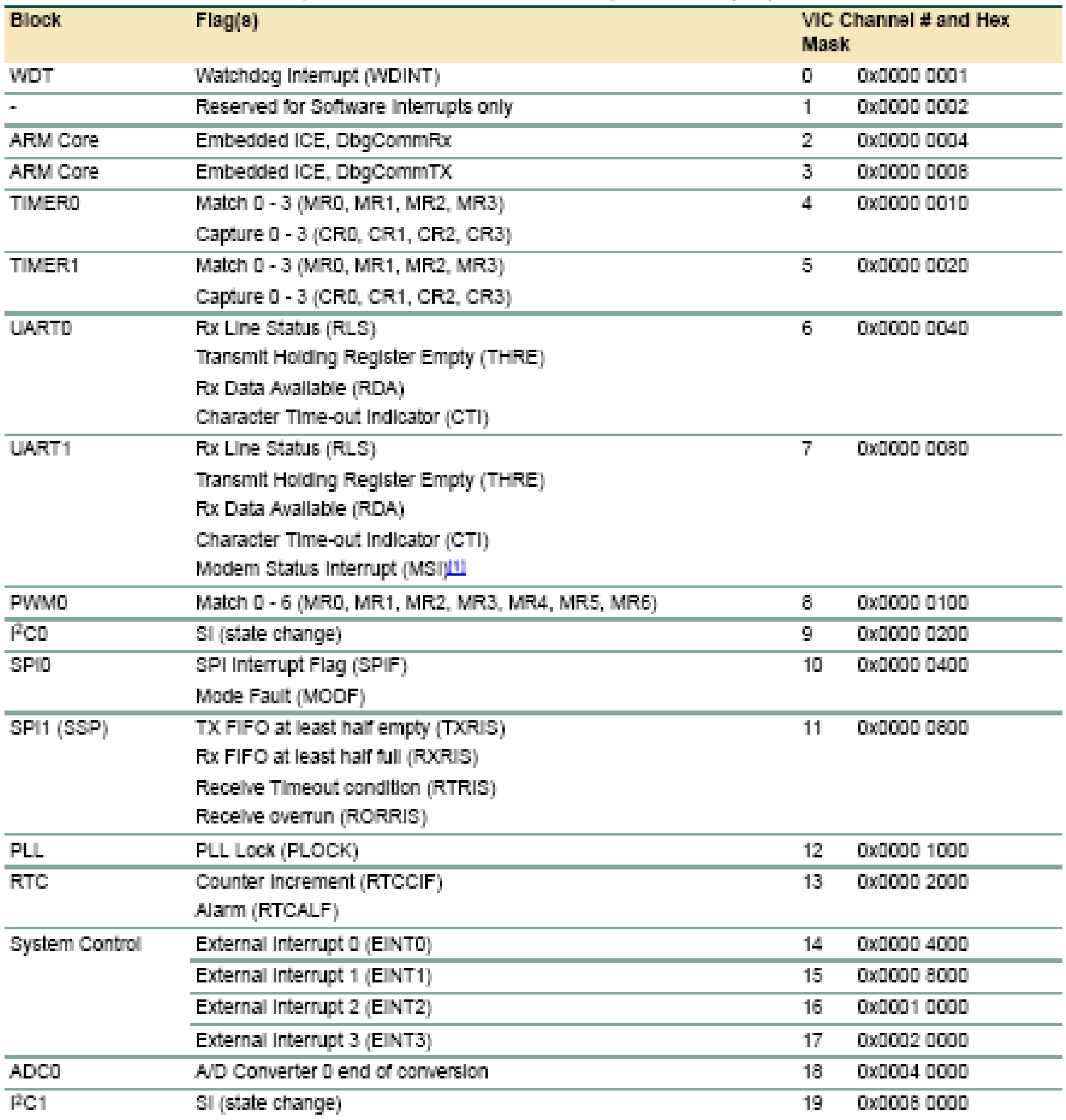
Table G.1 interrupt controller registers



Name	Description	Access	Reset value[1]	Address
VICIRQStatus	IRQ Status Register. This register reads out the state of those interrupt requests that are enabled and classified as IRQ.	RO	0	0xFFFF F000
VICFIQStatus	FIQ Status Requests. This register reads out the state of those interrupt requests that are enabled and classified as FIQ.	RO	0	0xFFFF F004
VICRawIntr	Raw Interrupt Status Register. This register reads out the state of the 32 interrupt requests / software interrupts, regardless of enabling or classification.	RO	0	0xFFFF F008
VICIntSelect	Interrupt Select Register. This register classifies each of the 32 interrupt requests as contributing to FIQ or IRQ.	R/W	0	0xFFFF F00C
VICIntEnable	Interrupt Enable Register. This register controls which of the 32 interrupt requests and software interrupts are enabled to contribute to FIQ or IRQ.	R/W	0	0xFFFF F010
VICIntEnClr	Interrupt Enable Clear Register. This register allows software to clear one or more bits in the Interrupt Enable register.	WO	0	0xFFFF F014
VICSoftInt	Software Interrupt Register. The contents of this register are ORed with the 32 interrupt requests from various peripheral functions.	R/W	0	0xFFFF F018
VICSoftIntClear	Software Interrupt Clear Register. This register allows software to clear one or more bits in the Software Interrupt register.	WO	0	0xFFFF F01C
VICProtection	Protection enable register. This register allows limiting access to the VIC registers by software running in privileged mode.	R/W	0	0xFFFF F020
VICVectAddr	Vector Address Register. When an IRQ interrupt occurs, the IRQ service routine can read this register and jump to the value read.	R/W	0	0xFFFF F030
VICDefVectAddr	Default Vector Address Register. This register holds the address of the Interrupt Service routine (ISR) for non-vectored IRQs.	R/W	0	0xFFFF F034

The main registers:
VicSoftInt - register bits which correspond to the existing demand at the moment interrupts;
VicSoftIntClear - register reset VicSoftInt;
VicIntEnable - register an individual permit / prohibit interrupt;
VicIntEnClear - register reset VicIntEnable;
VicIntSelect - register selection anchor (IRQ or FIQ) for each interrupt;
VicVectCntl0-15 - slots Registers IRQ (bit resolution and contain a number of slots for each of the 16 vector interrupt IRQ);
VicVectAddr0-15 - Address registers vectors IRQ;
VicVectDefAddr - address register handler nevektornogo IRQ;
VicVectAddr - register containing the address of the treated IRQ;
VicProtection - register to enable access to the registers of VIC.

Table G.2 Interrupt Sources



Block	Flag(s)	VIC Channel # and Hex Mask	
WDT	Watchdog Interrupt (WDINT)	0	0x0000 0001
-	Reserved for Software Interrupts only	1	0x0000 0002
ARM Core	Embedded ICE, DbgCommRx	2	0x0000 0004
ARM Core	Embedded ICE, DbgCommTX	3	0x0000 0008
TIMER0	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	4	0x0000 0010
TIMER1	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	5	0x0000 0020
UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI)	6	0x0000 0040
UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Status Interrupt (MSI)[1]	7	0x0000 0080
PWM0	Match 0 - 6 (MR0, MR1, MR2, MR3, MR4, MR5, MR6)	8	0x0000 0100
I^2C0	SI (state change)	9	0x0000 0200
SPI0	SPI Interrupt Flag (SPIF) Mode Fault (MODF)	10	0x0000 0400
SPI1 (SSP)	TX FIFO at least half empty (TXRIS) Rx FIFO at least half full (RXRIS) Receive Timeout condition (RTRIS) Receive overrun (RORRIS)	11	0x0000 0800
PLL	PLL Lock (PLOCK)	12	0x0000 1000
RTC	Counter Increment (RTCCIF) Alarm (RTCALF)	13	0x0000 2000
System Control	External Interrupt 0 (EINT0)	14	0x0000 4000
	External Interrupt 1 (EINT1)	15	0x0000 8000
	External Interrupt 2 (EINT2)	16	0x0001 0000
	External Interrupt 3 (EINT3)	17	0x0002 0000
ADC0	A/D Converter 0 end of conversion	18	0x0004 0000
I^2C1	SI (state change)	19	0x0008 0000

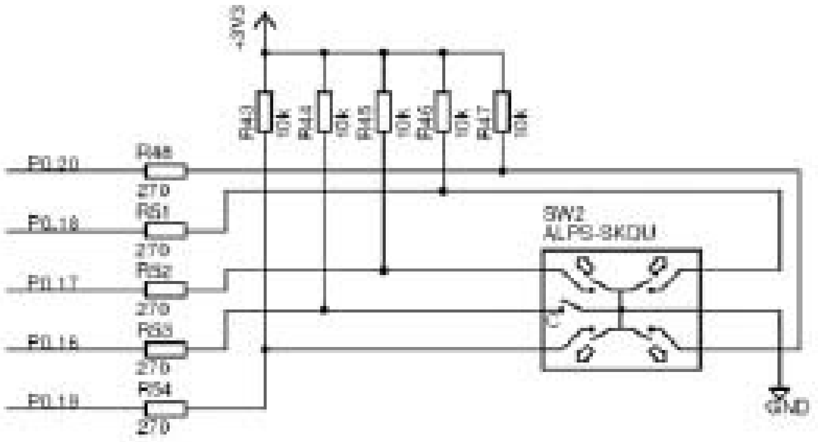


Figure H.1 - Joystick schematics

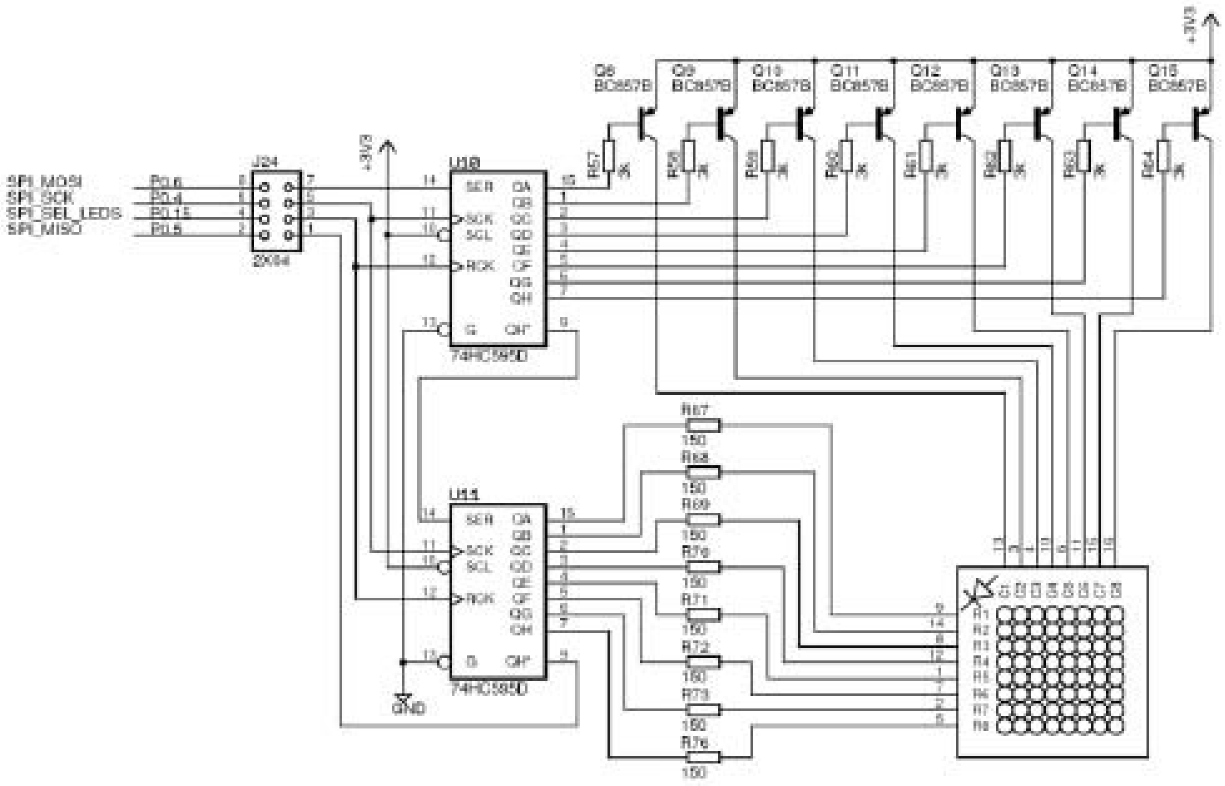


Figure H.2 - Matrix indicator schematics

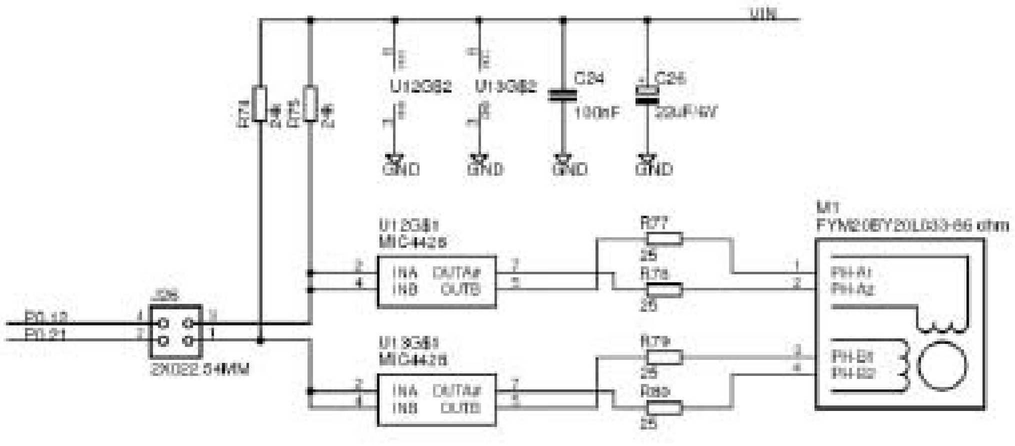


Figure H.3 - Stepper motor schematics

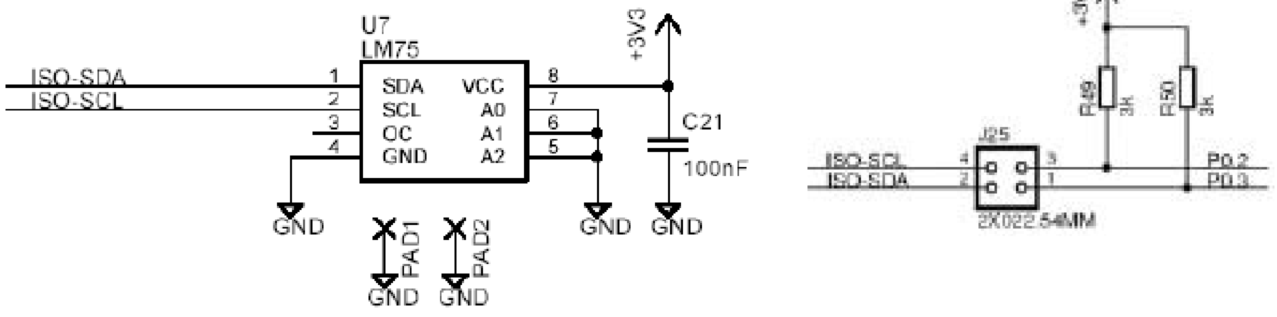


Figure H.4 - Temperature sensor schematics

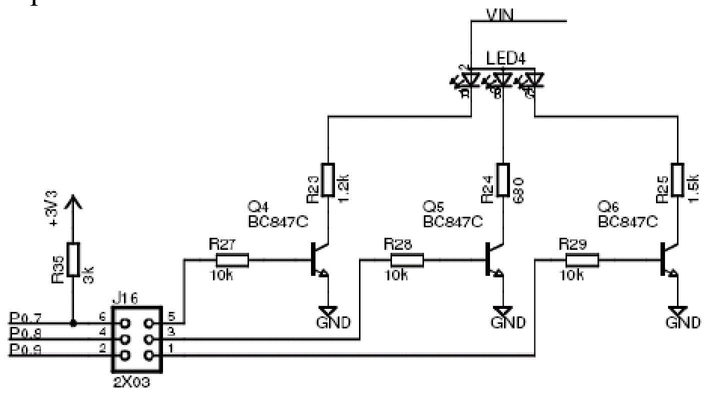


Figure H.5 - RGB indicator

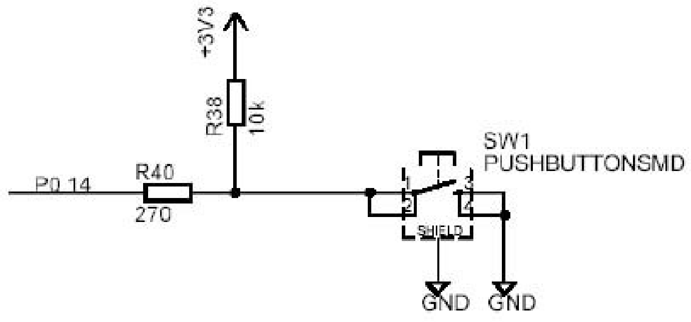


Figure H.6 - P0.14 button

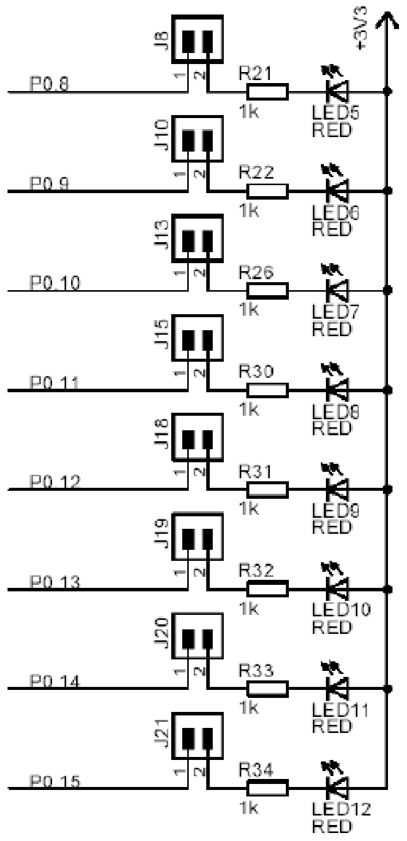


Figure H.7 - LEDs

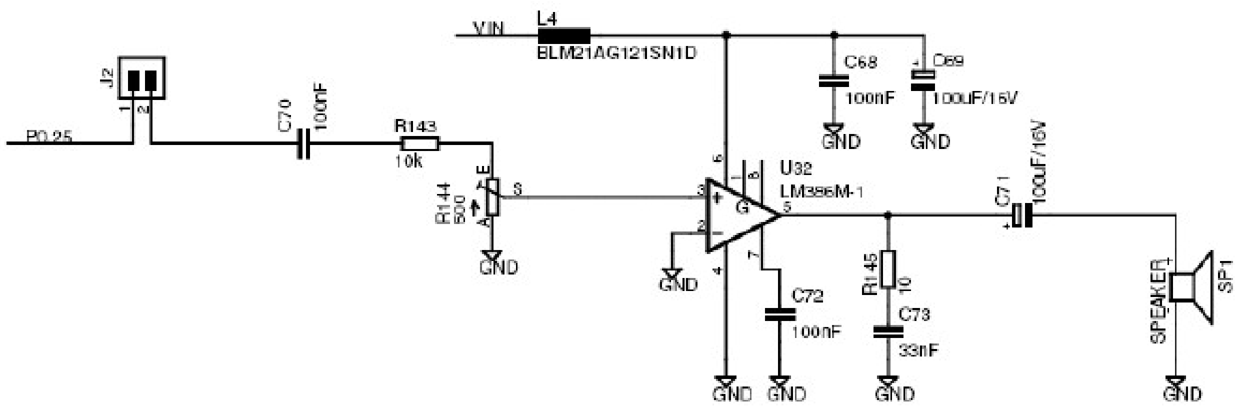


Figure H.8 - Speaker schematics

Educational edition

Working with the Embedded Artists LPC2148 evaluation boards

Learner's guide.

Compilers: Ilya Kudryavtsev, Dmitry Kornilin

Samara State Aerospace University (SSAU)