THE MINISTRY of EDUCATION and SCIENCE of RUSSIAN FEDERATION

SAMARA STATE AEROSPACE UNIVERSITY

Software development and debugging for NXP ARM7 MCUs

Learner's guide

SAMARA

2011

Compilers: Kudryavtsev Ilya Alexandrovich,
Kornilin Dmitry Vladimirovich

Software development and debugging for NXP ARM7 MCUs = Разработка и отладка программ для микроконтроллеров семейства ARM7 фирмы NXP [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 (0,44 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. 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

This lab is devoted to the studying of the programming features for 32 - bit ARM7TDMI microcontrollers in IAR EMBEDDED WORKBENCH IDE (Integrated Development Environment), developed by IAR SYSTEMS. These MCUs are high-performance devices, having Von Neumann's architecture with a rich set of peripherals.

Microcontrollers have sufficient capacity (up to 512 KB) of FLASH-on-chip memory, which encourage us to write programs in high level languages, and have quick access module, which increases the performance when running the program directly from the FLASH-memory.

Debugging programs with IAR Embedded Workbench in simulation mode, one can use a macro language, expanding the debugger's possibilities. This feature can compensate the lack of direct support for peripheral devices.

Code size and performance are equally important, when creating software for the real-time systems with high-level languages, like C or C++. To use IAR EMBEDDED WORKBENCH IDE in a full power developers have to know main compiler settings to optimize the program code size or speed. It is possible also to carry out an analysis of performance optimization with built-in tools, like Profiler (Profiling) and the code analyzer (Code Coverage). Studying this issue has three objectives: the getting the experience of working with the IAR, using C/C++ programming and to studying the characteristics of ARM7 microcontrollers.

This guide allows students to conceive the basics of C programming for ARM7 microcontrollers with the help of IAR Embedded Workbench. This guide either do not include full description of ARM7TDMI core's features or user's manual for IAR Embedded Workbench. Included are only brief comments, necessary for the understanding of code examples, the appendix contains some details of technical documentation for ARM7 microcontrollers.

1 Project creation

Software development will be performed, using environment IAR EMBEDDED WORKBENCH IDE, created by IAR SYSTEMS.

First of all you need to create a directory with a project, containing all necessary source files, auxiliary and output files. To create new project you can use menu «Project/Create New Project». After that (fig. 1) the program offers some types of the projects. Choose «Empty project» option, then in the following window you should enter the path to your directory and project's name. After this stage you can add or remove any sources at any time.

To save the time, we shall use prepared set of files, which you can find in the directory, named «Samples» (Instructor will tell you exact path). We shall copy necessary files from this directory and put them into our project, when needed.

ATTENTION ! DO NOT REMOVE, ADD OR MODIFY ANY FILES IN «Samples» DIRECTORY. YOU MAY PERFORM ALL MODIFICATIONS ONLY IN YOUR OWN DIRECTORY.

To begin your work with the first project, copy from «Samples» subdirectory Configurations and files Main.cpp, lcd.cpp.

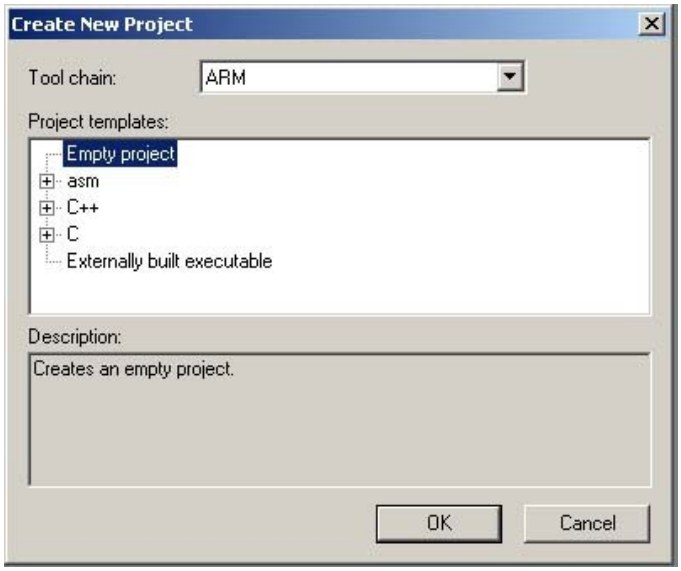

Figure 1 – Project creation

In the window Workspace in the left part of your workspace you can see two default project configurations: Debug and Release. We can also add our own configurations, for example, it is useful to have a configuration, loading code into the internal RAM. It is essential to use Debug configuration for debugging and Release configuration – for the final project with necessary optimizations.

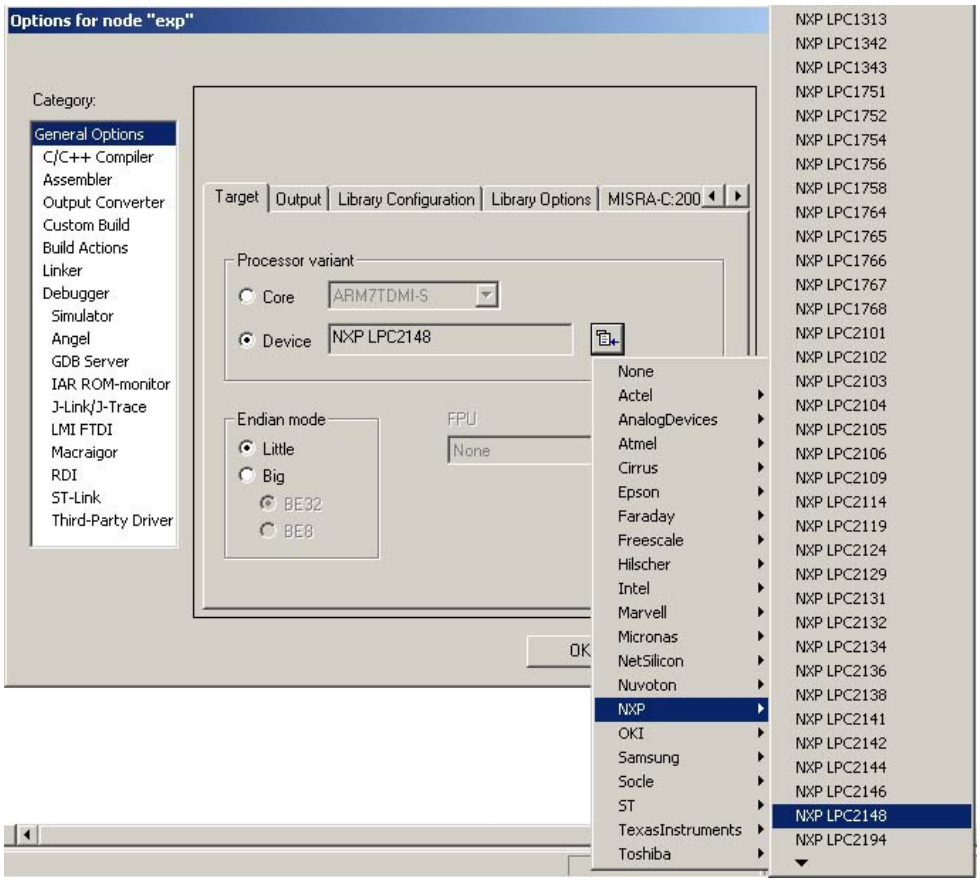

Figure 2 – MCU selection

We shall use Debug configuration for the simulation and Release configuration for the placement of the code in the Flash memory of microcontroller. RAM configuration will be used for the experiments with the code, placed in RAM.

Now we shall define our configurations. The first will be Debug configuration. Select this configuration, then click the name of the project with the right mouse button and select tab «Options…», then you will see the window, shown on fig. 2. Now select «General Options» tab and CPU type – LPC2148 of NXP company.

For using C++ syntax you need to select C/C++ compiler tab and mark «Automatic» or «Embedded C++» in «Language» group.

Select «Debugger» and set «Simulator» in the window «Driver». Other options we shall leave unchanged.

To define debug configuration with code placement in FLASH memory («Release» configuration) select the same MCU, then select «Linker», put the mark in the field «Override default» and enter the path to the file LPC2148_flash.icf, which is placed in the subdirectory Configurations. Select «Debugger» again and set the mark J-Link/J-Trace in «Driver» window.

To create configuration with the placement of the code in RAM select menu «Project\Edit Configurations», then «New» and create new configuration with the name RAM. Customize this configuration in the same way, as earlier, then select «Debugger» in the «Categories» window and select a debugger, as shown on the fig. 3. Mark «Use macro file(s)» and set the path to the file LPC2148_RAM.mac, which is placed in the subdirectory Configurations. Then select «Linker», mark «Override default» and set the path to the file LPC2148_RAM.icf in the same subdirectory.

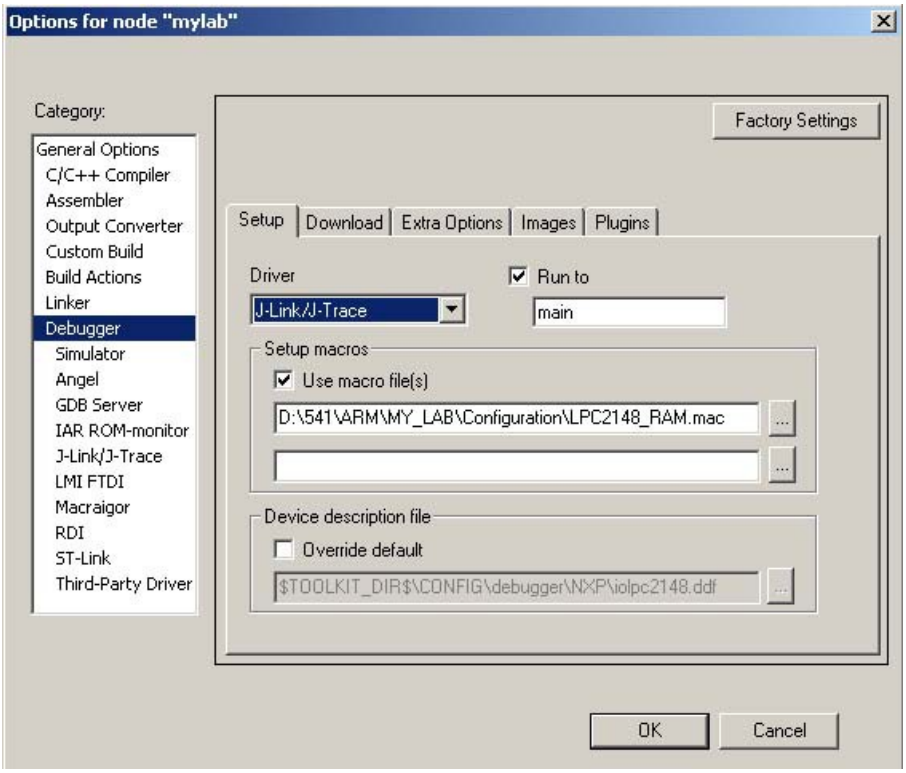

Figure 3 – Customizing the debugger

Now click right mouse button over «Add Files» and add the files «Main.cpp» and «lcd.cpp». The project is ready for the compilation and debugging.

2 Compilation and debugging

2.1 Project compilation

To translate and link the object file you can use PROJECT menu and activate MAKE or COMPILE tab. After the completion of the process, the program opens MESSAGES window, if errors or warnings are present.

2.2 Debug modes

Select Debug configuration, which is intended for the simulation purposes. To start debugger select «Download and Debug» in PROJECT menu, then debugger will start automatically, and a cursor will be placed on the «main» function.

Debugger allows executing the program step-by-step, using F10/F11 or a pictogram in the toolbox panel. Pressing F10 you make the debugger to execute one instruction in one step. If it is a subroutine call, it would be executed as one instruction. Using F11 you can merge in the subroutine and execute it as a separate program. You can also start the program in automatic mode with preliminary setting of breakpoints (see details below).

In addition, you can execute the program with a delay (AUTOSTEP), in which commands are executed with adjustable delay. Also, you can use mode with a stop at the cursor position, performing before the release of the subroutine.

2.3 Viewing and modifying registers and memory variables

To modify the register you should open the window «Register» (Register tab menu View). To modify the registry you can type the value directly in the box and press ENTER.

It is useful to view and modify variables window «Watch». Using a right-click in this box, you can add (ADD), remove (REMOVE) or change the display format of the variable. In the «Quick watch» window you can also see the value of variables whose values are not necessarily inspected all the time. If necessary, you can transfer temporary variable in a permanent window WATCH. You should add symbol "#" to the first character of the name of registers, for example, to display the content of the IO0DIR, enter #IO0DIR.

To view and modify memory locations you can use MEMORY tab. This opens a dialog box where you can view and modify the desired cell. For your convenience, you can display a separate area of memory related to the RAM, FLASH, etc. With this window you can also set the breakpoint on access to individual memory cells or the entire range.

2.4 Using breakpoints

Used programming environment suggests the possibility of using three types of breakpoints: in particular instruction code, to access a memory location and the point of interruption instead of stopping the execution of the program (Immediate).

To edit a breakpoint, you can use the tab «Edit breakpoints», which opens a special dialog box.

To set a breakpoint user manual can use the context menu in the corresponding position of the cursor. To set a breakpoint on memory access it is convenient to use the window «Memory».

Debugging the interrupt's routine it is convenient to use a macro function, connected with the breakpoint, which would automatically modify memory or display any information. To do this edit box «Action», which have to specify the name of the macro functions.

2.5 The use of macro functions

Macro functions are to be created in separate files with «mac» extension. To use macro function, you should use menu «Debug/Macros», enter the name of the file, containing macros, then add it to the list of available macro functions («Add») and register it («Register»).

2.6 Emulation interrupts

To emulate interrupts you should use the tab «Interrupts» from menu «Simulator». Activate checkbox «Enable», select the desired interrupt vector, then specify the number of cycles until first occurrence («First Activation»), repetition period of «Repeat interval». You can also define the duration of the interrupt request «Hold time» (specify «Infinite»), the probability of interruption in the period in percents and its deviation, if needed.

There is also the possibility of manual activation of the interrupt at any time by using the menu «Simulator/Forced Interrupts». In the list you should select the desired vector, and then click «Trigger».

2.7 Performance analysis and program optimization

To analyze the efficiency of the program, there are two tools: analyzer and profiler code execution. The profiler can be started from the menu «View/Profiling». Profiler provides numerical data (in cycles) and relative (in percents) of pure run-time of selected functions and the overall run-time, including library functions.

Analyzing these data, we can make conclusions about the bottlenecks in the program and find the ways of code optimization.

Performance analyzer («View/Code Coverage») is intended to search the parts of the program, which are not executed for some reasons. Such functions are marked by the red color, green color designates functions, performed entirely, red and green - partially executed. Inside the "red" functions, you can see the individual "yellow" transitions to non-executed code.

Return to the Source Editor and open the settings dialog compiler («Project / Options / C + + Compiler»). By selecting the tab «Code», you can do some experiment with the compiler settings, observing the results in the debugger.

After debugging code, you ought to perform the procedure of the optimization and check the program for its functionality. This is necessary because during the optimization process, compiler can discard the execution of certain functions which ap-

pears to be unnecessary (from compiler's point of view), for example, repeated switching the direction of the port from input to output.

Make some experiments with the profiler and analyzer, changing the way of code optimization and the frequency of the generated interrupts.

3 Bibliography

1. Trevor Martin The insider's guide to the Philips ARM7 – BASED MICRO-CONTROLLERS. 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

Table A.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 B.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)

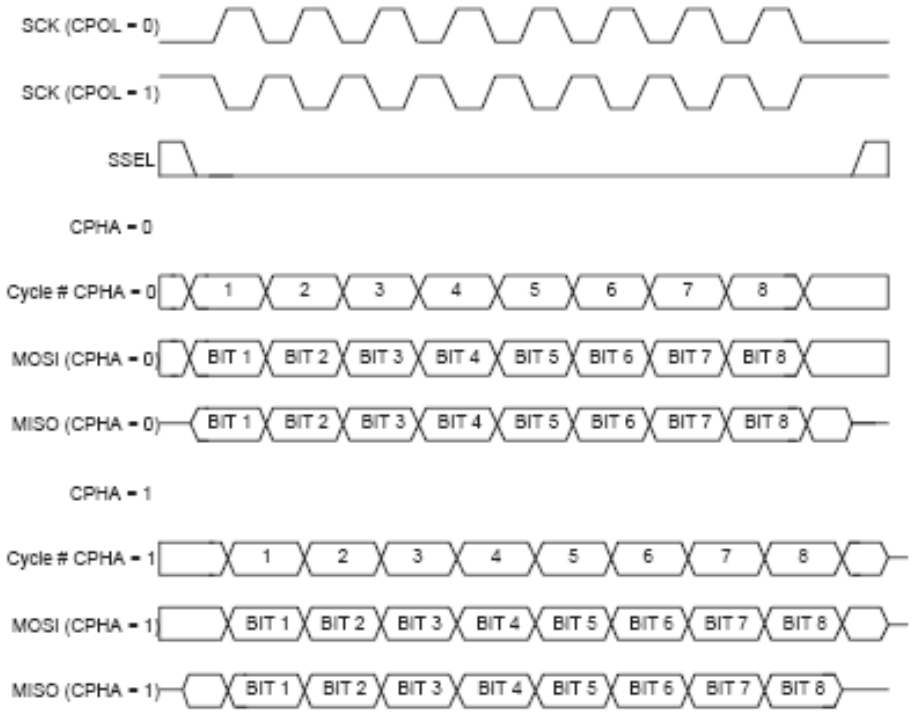

Figure C.1 - Formats the transfer module in SPI

Table C.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 D.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 E.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 AD0CR	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 F.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 of non-vectored IRQ;
VicVectAddr - register containing the address of the treated IRQ;
VicProtection - register to enable access to the registers of VIC.

Table F.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

Educational edition

Software development and debugging for NXP ARM7 MCUs

Learner's guide.

Compilers: Ilya Kudryavtsev, Dmitry Kornilin

Samara State Aerospace University (SSAU)