Part 1

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Table of Contents

		Page
	Table of Contents	1
1.	Introduction	2
2.	Hardware Requirement	3
3.	Software Requirement	3
4.	Prepare the IDE	4
	4.1 Installation of EWARM	
	4.2 Installation of SAM Boot Assistant (SAM-BA)	
	4.3 Installation of ARM-Jtag and H-JTAG driver	
5.	The first project (Let's blink the LEDs!)	11

Appendix A :When ARM JTAG failed with my Windows 2000 workstation29Appendix B :Why my first project didn't RESET?30

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

1. Introduction

Olimex AT91SAM7-P64-A development board provides a low cost alternative to evaluation of the Atmel's AT91SAM7S low pin-count 32-bit ARM RISC processor. Features of the AT91SAM7-P64-A development board as below.

- MCU: AT91SAM7S64 16/32 bit ARM7TDMI[™] with 64K Bytes Program Flash, 16K Bytes RAM, USB 2.0, RTT, 10 bit ADC, 2x UARTs, TWI (I²C), SPI, 3x 32bit TIMERS, 4x PWM, SSC (I²S), WDT, PDC (DMA) for all peripherals, up to 55MHz operation
- standard JTAG connector with ARM 2x10 pin layout for programming/debugging with ARM-JTAG
- USB connector
- Two channel RS232 interface and driver
- SD/MMC card connector
- two push-buttons
- trimpot connected to ADC
- thermistor connected to ADC
- two status LEDs
- single power supply: 6V 9V DC/AC required
- 18.432 MHz crystal
- Full Schematics available



Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

2. Hardware Requirement

Besides the AT91SAM7-P64 board, we need few additional hardware to get started:

- 1. A PC running Windows XP
- 2. Olimex ARM JTAG debugger
- 3. USB Cable (Type A <-> Type B)
- 4. Straight parallel port extender cable (optional)
- 5. 6V 9V DC transformer

3. Software Requirement

- Free 32KB Kickstart edition of IAR Embedded Workbench Kiskstart for ARM" V4.30A (EWARM) downloaded from <u>www.iar.se</u>
- H-JTAG Server downloaded from <u>www.olimex.com</u> or <u>www.TechToys.com.hk</u> under "*Tools and Software->ARM JTAG*" section. The latest version is v0.3.1.
- An example program downloaded from <u>www.TechToys.com.hk</u> under *"ARM Boards->Atmel SAM7-P64->Documents->*Getting Started with SAM7 (30 pages Guide) & <u>source code</u>".
- 4. SAM Boot Assistant (SAM-BA) obtained from www.at91.com

Part 1

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

4. Prepare the IDE

4.1 Installation of EWARM

The first thing to do is to download a copy of the "IAR Embedded Workbench Kiskstart for ARM" V4.30A (EWARM) from <u>http://supp.iar.com/Download/SW/?item=EWARM-KS32</u> and get it installed. After download, double click on the installation file (93MB). Complete the registration afterwards.



From Windows Start menu, there will be a program as IAR Embedded Workbench. My PC have IAR Embedded Workbench KickStart for MSP430 also, so there is another 'IAR Embedded...MSP430' above the ARM KickStart. This MSP430 workbench is not relevant to ARM development.

📷 IAR Systems	🚺 🚺 IAR Embedded Workbench KickStart for MSP430 V3	+	
🛗 H-JTAG VO.2	IAR Embedded Workbench for ARM Kickstart	۰Z	IAR Embedded Workbench
ATMEL Corporation	🕨 🛑 IAR Systems License Activation	()	IAR Product Information Log File
	🚾 🕜 IAR Systems License Manager	e)	Release notes
N		- 13 ^j	Uninstall IAR Embedded Workbench Kickstart for ARM $\mathtt{V4.30A}$

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

4.2 Installation of SAM Boot Assistant (SAM-BA) SAM-BA is a software provided by Atmel to in-circuit program AT91 devices via RS232 or USB. **NO PROGRAMMER IS REQUIRED!** From www.at91.com, download SAM-BA and get it installed. There is not much difficulty with it. Restart the PC after installation.



When SAM-BA starts, there is a dialog box asking for connection protocol as below. Our board is AT91SAM7S64-EK compatible. Select AT91SAM7S64-EK. We may use either serial connection or USB connection. Let's use USB connection because the board will be powered by the USB, no need to use external power for the SAM board in this case. Click on USB connection button.

SAM-BA 1.7 - Choose Protocol	
Select COM port : COM1	
Choose your board . [AT 31 3AM730	
DBGU connection USB c	onnection

I did that, and I have got an error message as below!



Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

The reason is that we need to perform SYSTEM RECOVERY PROCEDURE:

1. Shutdown the board (unplug USB cable in our case),

2. Before power up the board, the TST (pin 40), PA0/PGMEN0 (pin 48), PA1/PGMEN1 (pin 47), and PA2/PGMEN2 (pin 44) signals must be set (high). On Olimex AT91SAM7-P64-A evaluation board, short only the TEST jumper to set the TST signal (PA0, PA1, PA2 are all set by default via internal pull-up resistors),



3. Power up the board (re-connect the USB cable) and wait 10 sec,

4. Shut down the board (remove USB cable) and remove the TEST jumper,

5. Power up the board (re-connect USB cable): The board is now working with the *SAM-BA Boot* application from flash and waiting for connection through USB or DBGU port. Click on USB connection. The atm6124USB driver used by SAM-BA will be automatically installed.

Found N	ew Hardware
÷	USB Device
Please	vait

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

SAM-BA will start. From here, we may download any executable file to our SAM7 ARM mcu (browse to *.bin file -> Send file -> press RESET button onboard). Visit our web page to find a simple LED blink binary file in *.bin format to simply blink the GREEN and YELLOW LEDs. Else, you may also visit <u>www.at91.com</u> for an executable version of Mass Storage application for SAM7S64.

Indeed, I was able to turn my SAM board to a USB thumb drive, though it was only a miserable 39KB FAT-format drive because it used the internal flash of the SAM7S64 for storage. Nevertheless, we may request the full source code from Atmel to learn implementation of a FAT file system in SAM. Because there is a mmc socket onboard exclusive with our SAM board, it will be possible for us to extend the file system to mmc cards and use the SAM board for any data logging project.

SAM-BA 1.7							- 🗆 :
- AT910AM7004 EK Memory	ect/Disconnect Help)					
A 1 31 SAM / S64-E.N. Memory Display							
Starting Address : 0x20200	0 Format : 32-bit	▼ Refresh					
Size : 128	byte(s)						
OxXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	_		-
OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX			
OxXXXXXXXX	OxXXXXXXXX	0xXXXXXXXX	0×XXXXXXXX	0×XXXXXXXX			_
OxXXXXXXX	OxXXXXXXXX	0xXXXXXXXX	OxXXXXXXXX	0×XXXXXXXX	_		
OxXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX			
0×XXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX			∎∎
SRAM FLASH							
Download / Upload File							
Send File Name : Browse Send File							
Receive File Name :	Browse	F	Receive File				
Address : 0x10	00000 Size (For R	eceive File) : 1024	byte(s)		Compare	sent file with memory	
Script(s) : Erase All Flash	active (Tcl8.4.9 / Tk	Execute					
SAM-BA 1.7) 1 % SAM-BA 1.7) 1 %				_			
				You are using a U	SB connection	Board : AT91SAM7S8	64-EK

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

4.3 Installation of ARM-Jtag and H-JTAG driver

Download H-JTAG V0.3.1 driver (under Tools & software-> ARM JTAG section) from our web site, unzip and install. After installation, there will be a short-cut on desktop and a non-plug&play device driver installed in the PC system's H-JTAG device manager.





Connect the ARM-JTAG via 20-pin IDC cable to the JTAG interface on SAM7 board as below. Connect ARM-JTAG to parallel port of your PC, use a 25-pin straight parallel port extension cable if necessary. Power-up the development board by connection of a 6V-9V DC transformer to PWR socket (or just leave the USB cable for power).



GETTING STARTED WITH ARM SAM7S64 Part 1 Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Double click H-JTAG short-cut to start the driver. The following screen program can be seen. This screen represents a successful connection between the PC and the SAM7S64 mcu via JTAG interface. From now on, we can minimize the H-JTAG Server and let it run in background.



If you don't see successful connection above, select **Settings** to customize the pin assignment as follows:

Wiggler (Predefined), that means we are using a wiggler clone.

```
TMS <-> Pin3 of parallel port (default)
```

ТСК	<-> Pin4 (default)
TDI	<-> Pin5 (default)
TDO	<-> Pin11(default)
nTRST	<-> Pin2
nSRST	<-> Pin6

Check nTRST output inverted

Make sure the parallel port setting of your PC matches 0x378 LPT1.

Jtag Settings				1
Jtag Selection	- Wiggler F	^o in Assignmen	t	
Wiggler (Predefined)	TMS	Pin3 D1	*	
C Sdt Jtag (Predefined)	TCK	Pin4 D2	*	
C User Defined	TDI	Pin5 D3	*	F
Reset Signal Output	TDO	Pin11 Busy		
I nTRST output inverted	nTRST	Pin2 D0	•	2
nSRST output inverted	nSRST	Pin6 D4	•	

GETTING STARTED WITH ARM SAM7S64 Part 1 Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Click on the magnifying glass icon to reconnect to target (SAM7S64 mcu). If you still cannot succeed in connection, please refer to Appendix A for remarks.

After the H-JTAG Server has made a successful connection with the target board, there will be an H-SERVER icon at the lower left corner of the desktop. This icon will change to a "PLAY" icon during debug.



Part 1

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

5. The first project (Let's blink the LEDs!)

Download an application from us at <u>www.TechToys.com.hk</u> and unzip the program to a convenient place. It is under ARM Boards -> Atmel SAM7-P64 -> Documents -> <u>Getting Started with SAM7</u> (30 pages Guide) & <u>source code</u>. Place the project folder anywhere you like. In my case, I have placed it under D:\SAM7S64.

Launch EWARM from Windows Start Menu, browse to D:\SAM7S64\AT91SAM7S-BasicTools\Compil\basic.eww



Click Open.

Open Workspac	e	? ×
Look in:	Compil 💌 🔶 🛅 🖷	
History Desktop My Documents My Computer	 bin Flash_debug Flash_Workspace_files RAM_Debug resource settings SrcIAR tools basic.eww 	
Mu Natwork P	File name: basic.eww	Open
My Network P	Files of type: Workspace Files (*.eww)	Cancel

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

You will see the workspace opened. The left panel is the workspace panel. Make sure the <u>drop-down menu</u> have RAM_Debug selected.

			🔽 🍾 🎠 🖳 🔍 🧔 🎼 🛤 🐘 🛤 💥 🚫				
Vorkspace 🔺	5	×	main Cstartup.579 Cstartup 5AM7				
RAM_Debug		-					
Files	22	12:					
🗆 🖻 Basic - RAM Debug	*		#include "AT915AM7564_inc.h"				
Ha Basic							
🗍 🖵 🔀 main.c		*	- Area Definition				
			·				
E Cstartup_SAM7.c		*	;				
			; ?RESET				
at91SAM7S64_16KRAM.xcl			; Reset Vector.				
			; Normally, segment INTVEC is linked at address 0.				
BAM7.mac			; For debugging purposes, INTVEC may be placed at other				
SAM7_RAM.mac			; addresses.				
🖵 📮 🤖 Output			; A debugger that honors the entry point will start the				
🖵 📓 Basic.d79	1		; program in a normal way even if INTVEC is not at address 0.				
			BDOCDAN ODESET				
			RSEC INTRANSTART REMAR				
			RSEG INTRAMEND REMAP				
			RSEG ICODE: CODE: ROOT(2)				
			CODE32 ; Always ARM mode after reset				
			org 0				
			reset				

This project is not much different from an example available under the installation directory of EWARM. I just modified the board.h file to suit our SAM board. Besides, I have also modified some of the functions provided by the standard library to see how the program run. Example is :

// AT91F_PMC_EnablePeriphClock (AT91C_BASE_PMC, 1 << AT91C_ID_PIOA) ;
*AT91C_PMC_SCER = AT91C_CKGR_MOSCEN; // main oscillator enable
*AT91C_PMC_PCER = 1<<AT91C_ID_PIOA; // peripheral clock enable</pre>

Please see comment in main.c and board.h for modifications made.

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Let's debug our application as follows:

Under **Project**->**Options**, select category **Debugger**. There are several

options built-in the EWARM. H-JTAG Server is RDI compatible, so we select RDI as the C-SPY Driver. Check **Run to** check-box, and put in **main** in the text box. This means we jump to the c-function main () in main.c when debug. If this option is not checked, we will jump to the first line of code of Cstartup.s79 when debug.



Click on another instance of **RDI** underneath **Debugger**. Browse to the H-JTAG RDI driver (H-JTAG.dll) from the H-JTAG installation directory. Normally, it would be under C:\Program Files\H-JTAG V0.2\H-JTAG.dll. Finally press **OK** to save all settings.

Category:		Factory Settings
General Options	RDI	
Assembler Custom Build	Manufacturer RDI driver	
Build Actions	C:\Program Files\H-JTAG V0.2\	H-JTAG.dll
Linker Debugger Simulator Angel IAR ROM-monitor J-Link Macraigor RDI Third-Party Driver	☐ Allow hardware reset ☐ ETM trace ☐ Log RDI communication	Note Use the RDI menu to specify additional driver settings. (This menu is available after the RDI driver has been located) Catch exceptions East Data FIQ Undef Prefetch SWI IRQ
	\$TOOLKIT_DIR\$*csp.com	nlog

Part 1

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Just to make sure we have all code complied and linked, we may right click on the Basic – RAM_Debug workspace and select Rebuild All.

X IAR Embedded Workbench IDE	
File Edit View Project RDI Tool	s Window Help
🗅 🛩 🖬 🕼 🕼 🖓 🖻 🖻	n a
Workspace	× main.c
RAM_Debug	//* Object
Files	8. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
🗆 🖪 Basic - RAM_Debug	Options
📙 📮 🧰 Basic	wait (
📙 🖵 🗟 main.c	Make Begin
📙 🛱 🚞 Cstartup	Compile nsigned
Cstartup.s79	Rebuild All hange_s
📋 🖵 🖪 Cstartup_SAM7.c	Dr (wait
	Stop Build End
□	
at91SAM7S64_NoR	Add • •
SAM7.mac	Remove
SAM7_RAM.mac	
	Source Code Control
Hasic.d/9	File Properties
	Set as Active ain()
	Begin
	int i;
	// First
	// Perip.
	// Che P

Build results show no error and warning.

×	Messages
	Rebuilding configuration: Basic - RAM_Debug Updating build tree
	5 file(s) deleted. Updating build tree Cstartup.s79 Cstartup_SAM7.c main.c Linking
П	Total number of errors: 0 Total number of warnings: 0
Build	Build Debug Log

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Expand each of the categories under the workspace. We can see all output files and their dependency.

Object files = *.r79 Linker command files = *.xcl Macro files for debugger = *.mac Output file for RAM-Debug = *.d79

RAM_Debug Image: Constraint of the second seco
Files # # □ Basic - RAM_Debug ✓ □ □ Basic ✓ □ □ Basic ✓ □ □ □ Basic ✓ □ □ □ □ □ ✓ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
□ □ ■
Image: Constraint of the second se
Imain.c Imain.c Imain.pli Imain.pbi Imain.r79 Imain.r79 Imain.pli
Imain.pbi Imain.r79
Imain.r79 Imain.r79 <td< td=""></td<>
AT91SAM7S64.h
Board.h Board.h Bib_AT91SAM7S64.h Cstartup
└── 📓 lib_AT91SAM7S64.h
🗕 🛱 🧰 Cstartup
ー
📘 🗕 Output
🗌 🔰 🖵 📓 Östartup.r79
🛛 🛛 🛏 📓 AT91 SAM7S64_inc.h
🛛 🖵 🛱 Cstartup_SAM7.c
🛛 🚽 📓 Cstartup_SAM7.pbi
🛛 🖳 🖾 Cstartup_SAM7.r79
🛛 🗕 🛗 AT91SAM7S64.h
🛛 🛏 📓 Board.h
🗌 🖵 📓 lib_AT91SAM7S64.h
🛛 🛏 📓 at91SAM7S64_16KRAM.xcl
🛛 🛏 📓 at91 SAM7S64_NoRemap.xcl
📙 🗕 📓 SAM7.mac
📙 🖵 📓 SAM7_RAM.mac
🖵 📮 🚞 Output
🗕 🖽 📓 Basic.d79
🖵 📓 Basic.map
Basic

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

It is possible for us to customerize what file to output under the C complier option. Right click on the project workspace, **Basic-RAM_Debug** -> **Options**

Workspace RAM_Debug	× main.c
Files	82 ⊡; # # I.
	Options
□ └─── 🛱 main.c └─── 🛱 Output □ └─── 🛱 main.lst □ └─── 🛱 main.pbi	Make Compile Rebuild All Clean

Select **C/C++ Complier** under Category, select **List** tab. Check Output list file option. Rebuild All.

Options for node "Basi	c" X
Category: General Options C/C++ Compiler Assembler Custom Build Build Actions Linker Debugger Simulator Angel IAR ROM-monitor J-Link Macraigor RDI Third-Party Driver	Factory Settings Language Optimization Output List Preprocessor Diagnostics Output list file Assembler mnemonics Output assembler file Include source Include compiler cell frame information
	OK Cancel

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Now we see a list file with an extension *.lst. Open main.lst and browse to the bottom, we can see the CODE segment data bytes and CODE memory usage.

Workspace ×	main.c main.lst
RAM_Debug	change speed 68
Files 🐉 🕰 🔺	wait 24
Resic - RAM Dobug*	nain 88
	??DataTablel 4
	<pre>?<initializer for="" ledspeed=""> 4</initializer></pre>
	Others 36
main.lst	
📗 🔚 main.pbi	208 bytes in segment CODE
📗 🖳 🔚 main.r79	8 bytes in segment DATA_C
📕 🗕 🛗 AT91SAM7S64.h	4 bytes in segment DATA_I
🛛 🗕 🔚 Board.h	4 bytes in segment DATA_ID
📙 🔚 lib_AT91SAM7S64.h	12 bytes in segment INITTAB
🗕 🖵 🔁 Cstartup	
🛛 🛏 📓 Cstartup.s79	184 bytes of CODE memory (+ 36 bytes shared)
U U U U U U U U U U U U U U U U U U U	12 bytes of CONST memory
📄 🖳 🛄 Ċstartup.r79	4 bytes of DATA memory
🛛 🖾 AT91SAM7S64_inc.h	Expert years
📙 🖵 🛱 Cstartup_SAM7.c	Vernings, none
🛛 🗕 🖵 🗀 Output	warnings: none
🛛 🚽 🛗 Östartup_SAM7.Ist	
🗌 🗕 🔚 Cstartup_SAM7.pbi	
🗌 🖵 📓 Cstartup_SAM7.r79	
AT91SAM7S64 h	

Go to workspace Options again. Under C\C++ complier -> Optimization -> select None(Best debug support).

Rebuild All.



Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Compare the build results with maximum optimization & No optimization, we can see the difference in both CODE and memory usage!

Workspace	×	main.c main.lst
RAM_Debug	▼	main 108
Files	2% 📴	??DataTable8 4
Regio - DAM Dobug *		??DataTable9 4
	•	<pre>?<initializer for="" ledspeed=""> 4</initializer></pre>
		Others 44
III		
📕 📙 🔁 Output		
📕 🔚 main.lst		268 bytes in segment CODE
📕 🔚 🔚 main.pbi		8 bytes in segment DATA C
📕 🔚 📠 main.r79		4 bytes in segment DATA I
🚽 🚽 📓 AT91SAM7S64.h		4 bytes in segment DATA_ID
📕 🗕 📓 Board.h		12 bytes in segment INITTAB
📄 🖳 🔚 lib_AT91SAM7S64.h		
🛛 🗕 🔁 Cstartup		232 bytes of CODE memory (+ 48 bytes shared)
📕 🖃 📓 Cstartup.s79		12 bytes of CONST memory
📙 🖵 🛱 Cstartup_SAM7.c		4 bytes of DATA memory
📕 🗕 🗀 Output		
📕 📄 🧰 Čstartup SAM7.lst		Errors: none
Cstartup SAM7.pbi		Warnings: none
Cstartup_SAM7.r79		

Part 1

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Under **Project** -> **Debug**, select **Debug**. Before that, make sure the H-JTAG server has started and successfully connected to the board as described in previous section.



If everything goes smooth, we would be able to debug our first application under the debug window. The green pointer stops at main().

💥 IAR Embedded Workbench IDE		
File Edit View Project Debug Disassembly RDI	Tools Wir	ndow Help
D 🛎 🖬 🕼 🎒 🔏 🚶 🖬 🖻 🗠 🗠 🗌		
≤╘╘┶┟╝╝		_
Workspace	×	main.c
RAM_Debug	•	change_speed
Files	82 B	for (waiting_t
🗉 🖻 Basic - RAM_Debuq *	✓	}//* End
🛛 🛏 🚞 Basic		//*
		//* Function Name
		//* Object
🛛 🖵 🔲 Output		//* Input Paramet
ll		//* Output Parame
		. //*
		⇒int main()
		{//* Begin
		int i;

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

The following short-keys allow us to perform stepping.

Go	F5
Step Over	F10
Step Into	F11
Step Out	Shift+F11

Click on **View** menu->**Register**. One more time, click on **View** menu->**Disassembly**.

Now we see the source file main.c, Register map, and the Disassembly listing on the same Window.



-										
	nain.c	-**	Register	<u> </u>	D	Disassembly				
		unsigned int waiting_time ;	CPU Registers	-		Goto		Memory	_	
		change_speed () ;	RO	= 0x000	ΙГ	Next label	is a T	humb label		
		<pre>for(waiting_time = 0; waiting_time < Led</pre>	R1	= 0x000		main:				
	>//	* End	R2	$= 0 \times 000$	111	UUUUU26C * 1T0	8510 10 DWC	PUSH	{R4,	LR}
			R3	$= 0 \times 000$		00000265	10 PMC 4814	<u> </u>	CKGR MOE	<u>CEN;</u> // ma [PC #0∞0501
	11+		R4	$= 0 \times 000$		00000270	2101	MOV	R1	#1
	11*	Function Name : Main	R5	$= 0 \times 000$		00000272	6001	STR	R1,	[R0, #0]
	11+	Object : Software entry poi	R6	$= 0 \times 000$		<u>*AT9</u>	1C PMC	PCER = 1< <at< th=""><th>91C ID PI</th><th><u>:OA;</u> // pe</th></at<>	91C ID PI	<u>:OA;</u> // pe
	11+	Input Parameters : none.	R7	$= 0 \times 000$		00000274	4813	LDR	RO,	[PC,#0x04C]
	11+	Output Parameters : none.	R8	$= 0 \times 000$		00000276	6001	STR	R1, P1	#4 [DO #0]
	11+		R9	$= 0 \times 000$		*AT9	1C PIO	A PER = LED M	ASK:	[10, #0]
	int.	main()	R10	$= 0 \times 000$		0000027A	4817	LDR	R0,	[PC,#0x05C]
	11	* Begin	R11	= 0x000		0000027C	21C0	MON	R1,	#192
		int i.	R12	= UxUUU		0000027E	02C9	LSL	R1,	R1, #11
		(/ First enable the clock of the DIO he	R13 (SP	$) = 0 \times 0 0 0$		UUUUU280	1C PTO	A OFR - TED M	ACV.	[KU, #U]
		// Perinberal Clock remained for SWI & S	KI4 (LK	$= 0 \times 0 = 0$		00000282	4811	LDR	RO.	[PC.#0x044]
		(/ the Bower Management Controller (DMC)		= 0x600		00000284	21C0	MOV	R1,	#192
		(/ Writing any of the registers of the 1		- 0xF00		00000286	02C9	LSL	R1,	R1, #11
		(/ enabled This means that the configur	R8 fig	= 0x000		00000288	6001	STR .	R1,	[KN` #O]
		(/ clock to be enabled Thus output INI	R9 fig	= 0x000		00000284	4810	A SODR = LED	RO RO	FPC #0v0401
		((Harmon when the clock is disabled	$R10 fi\sigma$	= 0x000		0000028C	21C0	MOV	R1	#192
		((Note that the Input Change Intermut	R11 fig	= 0x000		0000028E	02C9	LSL	R1,	R1, #11
		// woce chac che input change interrupt	R12 fig	= 0x000		00000290	6001	STR	R1,	[RO, #O]

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Click on **View** menu->**Memory**.



From the memory map, we see the machine code 4814, 2101, 6001 at the addresses 0x026E, 0x0270, and 0x0272 respectively.

L

_									
×	Go to			/lemory		·			
	000001e0	bc70	Ъ000	4770	46c0	02dc	0000	02e8	0000
	000001f0	6bc0	4770	Ъ500	4838	f7ff	fffa	0300	d40a
	00000200	4819	6800	490d	4288	d305	4817	4916	6809
	00000210	4a0b	1889	6001	4830	f7ff	ffea	02c0	d40a
	00000220	4811	6800	4907	4288	d205	480f	490e	6809
	00000230	4a05	1889	6001	bc01	4700	46c0	12Ъ9	0000
	00000240	ed48	ffff	4000	02db	12Ь8	0000	Ь510	f7ff
	00000250	ffd1	2000	1 ± 04	4804	6800	4284	d201	1c64
	00000260	e7f9	bc10	bc01	4700	02f4	0000	Ь510	4814
	00000270	2101	6001	4813	2104	6001	4817	21c0	02c9
Ĕ	00000280	6001	4811	21c0	02c9	6001	4810	21c0	02c9
Re	00000290	6001	2000	1 ± 04	2002	dafb	480c	2104	4361



Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Use F11 to step into each line of codes to see what happen.

Main.c	Register View	Disassembly Listing
*AT91C_PMC_SCER = AT91C_CKGR_MOSCEN;	PC = 0x0000026E	➡ 0000026E 4814 LDR R0, [PC,#0x050]
		;[0x2C0]=PMC_SCER (0xFFFF FC00)

PC = Program Counter @ the address 0x0000 026E

LDR R0, [PC, #0x050]; assembler instruction for moving the value at the address [PC+0x050] to register R0 A comment generated from the assembly listing states that [0x2C0]=PMC_SCER (0xFFFF FC00).

0x2C0 = 0x270 + #0x050.

0x270 is the PC for the next line of code Browse a little bit downwards on the Memory map, we have a value of 0xFFFF FC00, which is the address of the register PMC_SCER defined under AT91SAM7S64.h. It is now located at the program address of 0x2C0 after linking the program.

	000002a0	4a0D	2021	
	000002Ъ0	4a07	5851	6
	000002c0	fc00	ffff	ł
	000002d0	02ec	0000	1
	000002e0	02f4	0000	(
	000002f0	0000	0004	đ
	00000300	1d9c	ad28	1
	00000310	3d5e	b17d	¢
Ĕ	00000320	2e58	4258	ł
Me	00000330	a850	1884	6

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Press F11 to single step. R0 and R1 registers assigned the value of 0xFFFF FC00 and 0x0000 0001, respectively.

Main.c	Register View	Disassembly Listing
*AT91C_PMC_SCER = AT91C_CKGR_MOSCEN;	R0 = 0xFFFFFC00 R1 = 0x000002E8	0000026E 4814 LDR R0, [P → 00000270 2101 MOV R1, #1
*AT91C_PMC_SCER = AT91C_CKGR_MOSCEN;	$\begin{array}{c} \mathbf{R0} &= 0 \times \mathrm{FFFFFC00} \\ \mathbf{R1} &= 0 \times 00000001 \end{array}$	00000270 2101 MOV R1, #1 ➡ 00000272 6001 STR R1, [R0

By single stepping 3 times, we ran the following code:

100	10 500	0070 17010				
<u>*AT9</u>	<u>IC PMC</u>	SCER = AT91C	CKGR MOS	<u>SCEN;</u>	may	
0000026E	4814	LDR	RO,	[PC,	#0x050]	
00000270	2101	MOV	R1,	#1 🔫		
00000272	6001	STR	R1,	[R0,	#0]	
						-
						1

Load the value 0xFFFF FC00, which is the address of the PMC register PMC_SCER (system clock enable register) located at the program address 0x2C0, to the working register R0 of the CPU.

Load the value 0x0000 0001 (AT91C_CKGR_MOSCEN) to R1

Load the value holded in R1 to the address holded at [R0+0], that is 0xFFFF FC00 in this case. The result is writing a value of 0x0000 0001 to the address 0xFFFF FC00 (PMC_SCER).

23 By John Leung, updated 19 July 06

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

In conclusion, the result of the first c-code

***AT91C_PMC_SCER = AT91C_CKGR_MOSCEN;**

is to write 0x0000 0001 to the PMC_SCER register, thus enabling the Processor clock by setting PCK=1.

Reading the manual of SAM7S64, I have learned more about PMC_SCER. A screen shot of page 189 as shown on right.

Continue stepping the codes until

***AT91C_PMC_PCER = 1<<AT91C_ID_PIOA;** finished.

Take a look at the Memory map down to the address 0xFFFF FFC0, we see the value of 0xFFFF FC08 (System Clock Status Register, PMC_SCSR) hold a value of 0x0000 0001, and 0xFFFF FC18 (Peripheral Clock Status Register, PMC_PCSR) hold a value of 0x0000 0004. It seems that because PMC_SCER and PMC_PCER are write-only registers, we cannot get viewed of the value directly from the Memory map. Instead, we need to rely on the PMC_SCSR and PMC_PCSR registers. Please correct me if I am wrong.

26.0.1 DMC System Clock Enable Degister								
20.9.1 PMC System Clock Enable Register								
Register Name	e: PMC_S	CER						
Access Type:	Access Type: Write-only							
31	30	29	28	27	26	25	24	
-	-	-	-	-	-	-	-	
23	22	21	20	19	18	17	16	
-	-	-	-	-	-	-	-	
15	14	13	12	11	10	9	8	
-	-	-	-	-	PCK2	PCK1	PCK0	
7	6	5	4	3	2	1	0	
9	-	-	-	-	-	-	PCK	
• PCK: Proces 0 = No effect.	PCK: Processor Clock Enable 0 = No effect.							
1 = Enables the	e Processor cio	ck. 						
0 No offect	evice Fort ord							
0 = NO effect.								
1 = Enables the 48 MHz clock of the USB Device Port.								
(Does not pertain to AT91SAM7S32.)								
PCKx: Programmable Clock x Output Enable								
0 = No effect.								
1 = Enables the corresponding Programmable Clock output.								

	fffffc00	0000	0000	0000	0000	0001
	fffffc10	0000	0000	0000	0000	0004
	fffffc20	0601	0000	257c	0001	3f00
	fffffc30	0007	0000	0000	0000	0000
	fffffc40	0000	0000	0000	0000	0000
	fffffc50	0000	0000	0000	0000	0000
	fffffc60	0000	0000	0000	0000	000d
	fffffc70	0000	0000	0000	0000	0000
	fffffc80	0000	0000	0000	0000	0000
	fffffc90	0000	0000	0000	0000	0000
\geq	fffffcaO	0000	0000	0000	0000	0000
ğ	fffffcb0	0000	0000	0000	0000	0000
희	fffffcc0	0000	0000	0000	0000	0000
21	CCCCC 10	0000	0000	0000	0000	0000

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Finally, press (Go) 😕 to run the program.

Green and yellow LEDs blink.

Press B1 to increase the rate, B2 to decrease rate.



Press (**Break**) **b** to stop running, and step over the main loop by repeating F10 to watch LED blink.

```
for (;;)
{
    // Once a Shot on each led
    for ( i=0 ; i < NB_LEB ; i++ )
    {
        *AT91C_PIOA_CODR = led_mask[i]; //clear output data register, turn ON LED
        Vait();
        *AT91C_PIOA_SODR = led_mask[i]; //set output data register, turn OFF LED
        wait();
    }// End for
}// End for</pre>
```

Part 1

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

After debugging, it is time to produce an executable file for flash download.

Click on Stop Debugging icon to quit.

Go back to workspace and select **bin** category under the drop-down menu.

There is a little bit different with the output option for *.bin output. Under Linker category, Output becomes Basic.bin, instead of Basic.d79 for RAM_Debug.

🔏 IAR Embedded Workbench I	DE
File Edit View Project RDI T	ools Windo
🗅 📂 🗐 🞒 🕹 🕮 🖿	e 0 0
Workspace	×
RAM_Debug	•
RAM_Debug Flash_debug	
bin	
🗕 🖵 🖻 Basic	
📙 🖵 🛱 main.c	
📔 🛏 🧰 Output	
📔 🔶 🔝 AT91SAM7S6	
📔 🛏 🔝 Board.h	
📋 🖳 🔚 lib_AT91SAM	

Options for node "Basi	с"	×
Category: General Options C/C++ Compiler Assembler Custom Build Build Actions Linker Debugger Simulator Angel IAR ROM-monitor J-Link Macraigor RDI Third-Party Driver	Output Extra Output #define Diagr Output file Image: Construction of the construction of the construction of the construction modules Image: Construction of the construction modules Format O bebug information for C-SPY Image: Construction modules Image: Construction of the construction modules Image: Construction modules Image: Construction of the construction modules Image: Construction modules Image: Construction of the construction modules Image: Construction modules Image: Construction of the construction modules Image: Construction modules Image: Construction of the construction modules Image: Construction modules Image: Construction of the construction modules Image: Construction modules Image: Construction of the construction modules Image: Construction modules Image: Construction of the construction of the construction modules Image: Construction of the construction of	Factory Settings

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Another difference is the linker command file option. For *.bin output, we use **at91SAMS64_NoRemap.xcl**.

Options for node "Basic"						
Category: General Options C/C++ Compiler Assembler Custom Build Build Actions Linker Debugger Simulator Angel IAR ROM-monitor J-Link Macraigor	Factory Settings Output Extra Output #define Diagnostics List Config Proce Linker command file ✓ Override default \$PROJ_DIR\$\resource\at91SAM7S64_NoRemap.xcl ✓ Command file configuration tool ✓ Override default program entry ○ Entry label _program_start Defined by application Search paths: (one per line)					
RDI Third-Party Driver	\$TOOLKIT_DIR\$\LIB\ Raw binary image File: Symbol: Segment: Align: OK Cancel					

Compile and link again by **Build All** for this bin project.

Repeat the SAM-BA SYSTEM RECOVERY PROCEDURE under section 4.2 of this manual. Browse to Basic.bin in

D:\SAM7S64\AT91SAM7S-BasicTools\Compil\bin\Basic.bin

Open			? ×
Look in:	🔁 Exe 💌	- 🗈 💣 🎟	
History History Desktop My Documents My Computer My Computer My Network P	 History Desktop My Documents My Computer 3.5 Floppy (A:) Local Disk (C:) Local Disk (D) SAM7564 AT91SAM75-BasicTools Compil Din Exe Protel93se_pack6 (E:) Compact Disc (F:) My Network Places DataSheets IR_Rehab Temp 		pen Incel

Part 1

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

SAM DA 1 7									
SAMPDA 1.7									
File Script File Connect/Disconnect Help									
AT91SAM7S64-EK Memo	AT91SAM7S64-EK Memory Display								
Starting Address : 0x20200	0 Format : 32-bit	 Refresh 							
Size : 128	byte(s)								
OxXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	A				
OxXXXXXXX	OxXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXX					
OxXXXXXXX	OxXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXXX					
OxXXXXXXX	OxXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXX					
OxXXXXXXX	OxXXXXXXX	OxXXXXXXXX	OxXXXXXXX	OxXXXXXXXX					
OxXXXXXXXX	OxXXXXXXX	OxXXXXXXXX	OxXXXXXXXX	OxXXXXXXX					
SRAM FLASH									
- Download / Upload File-									
Send File Name : D:/	SAM7S64/AT91SAM7SJ	BasicTools/Compil/bit	n/Eve/Basic	Browse	Send File				
Beceive File Name :		Susier Solar Complex Di		Browse	Beceive File				
Address : Ox1	00000 Size (For B	eceive File) : 1024			Compare sent file with memory				
			5,10(0)						
Script(s) :									
Erase All Flash	-	Execute							
Eldoornarridon									
landing bistory file events added									
SAM-BA console display active (Tcl8.4.9 / Tk8.4.9)									
(SAM-BA 1.7) 1 %									
(SAM-BA 1.7) 1 % send_file FLASH:: "D:/SAM7S64/AT91SAM7S-BasicTools/Compil/bin/Exe/Basic.bin" "0x100000" 1 Ja Send File D:/SAM7S64/AT01SAM7S-BasicTools/Compil/bin/Exe/Basic bin at address 0x100000									
-I- File size = 1024 byte(s)									
(SAM-BA 1.7) 1 %									
				You are using a U	SB connection Board : AT91SAM7S64-EK 🚽				

Click **Send File** to program flash. Press RESET button onboard to see the result. Try pressing B1 and B2 to increase/decrease the blink rate.

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Appendix A : When ARM JTAG failed with my Windows 2000 workstation

To be honest, I didn't debug my first application that smooth as stated in this document. There are several PCs in the office. Some of them are Windows XP Chinese, while my main workstation for programming is a PIII 500 running Windows 2000 SP4.

When I used a Windows 2000 workstation, I just couldn't connect the ARM-JTAG with the SAM board. The following H-JTAG Server message came

out no matter how I changed the **Settings**.

Finally, I just dig out an old **SDT JTAG** and a DIY JTAG that I have made some months ago. By using the same H-JTAG software driver, both of them worked; though I needed to change **Settings** to **sdt Jtag** for the yellow one as shown below.

H-JTAG Server

File Operations Settings Help

H-JTAG Server

H-JTAG Server

Unable to find target. Please make sure that the hardware

is properly connected and powered up.

OK

Bready

For the **DIY JTAG**, I just use the same settings as Olimex JTAG.



Remarks: I still don't know why Olimex ARM-JTAG not willing to work on my Windows 2000 workstation. Anyway, it is OK with Windows XP

Part 1

Keywords: Get started, Microcontroller, ARM, SAM7S64, ARM-JTAG, IAR EWARM

Appendix **B** : Why my first project didn't RESET?

There was an interesting finding in my first project. Right after I have programmed the chip with Basic.bin by SAM-BA, press on RESET button did reset the chip. This was within expectation from my experience with PIC / 8051. No matter what, a low on the RESET pin of a mcu will reset the chip, right?

However, after I have removed the power from the board and reconnect, the same RESET button no longer worked! Why?

A hint on page 70 of SAM7S64 data sheet : there is a Reset Controller handling all resets of the system.

I have to program the RSTC_MR register, setting the URSTEN bit to enable detection of a low level on the pin NRST for external reset! This URSTEN bit is 0 by default! In the main.c program, I have made the following line:

```
Reset Controller Mode Register
14.4.3
Register Name:
                        RSTC_MR
Access Type:
                        Read/Write
      31
                                       29
                                                       28
                                                                       27
                       30
                                                                                       26
                                                                                                       25
                                                                                                                        24
                                                               KE)
       23
                       22
                                       21
                                                       20
                                                                       19
                                                                                       18
                                                                                                       17
                                                                                                                    BODIEN
      15
                       14
                                       13
                                                       12
                                                                       11
                                                                                       10
                                                                                                                        8
                                                                                                       9
                                                                                             ERST
                                                    URSTIEN
                                                                                                                    URSTEN

    UBSTEN: User Reset Enable

0 = The detection of a low level on the pin NRST does not generate a User Rese
1 = The detection of a low level on the pin NRST triggers a User Reset.
• URSTIEN: User Reset Interrupt Enable
0 = USRTS bit in RSTC_SR at 1 has no effect on rstc_irq.
1 = USRTS bit in RSTC_SR at 1 asserts rstc_irq if URSTEN = 0.

    BODIEN: Brownout Detection Interrupt Enable

0 = BODSTS bit in RSTC_SR at 1 has no effect on rstc_irq.
1 = BODSTS bit in RSTC_SR at 1 asserts rstc_irq.

    ERSTL: External Reset Length

This field defines the external reset length. The external reset is asserted during a time of 2<sup>(ERSTL+1)</sup> Slow Clock cycles. This 
allows assertion duration to be programmed between 60 µs and 2 seconds.

    KEY: Password
```

Should be written at value 0xA5. Writing any other value in this field aborts the write operation

```
int main()
{//* Begin
    int i;
    ....
    //*AT91C_RSTC_RMR = (0xA5000000|AT91C_RSTC_URSTEN);
    ....
```

}

Uncomment this code and reprogram the chip, I have got a working NRST pin back! Up-to-now, I know there is much to learn with this little SAM!

...to be continued