ARMY RESEARCH LABORATORY

An Evaluation and Development Environment for an ARM7-Based Autopilot Using the Atmel SAM7S256 Microcontroller

by Justin L. Shumaker and Ainsmar X. Brown

ARL-TN-367 August 2009

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TN-367 **August 2009**

An Evaluation and Development Environment for an ARM7-Based Autopilot Using the Atmel SAM7S256 Microcontroller

Justin L. Shumaker
Vehicle Technology Directorate, ARL

Ainsmar X. Brown
National Institute of Aerospace

REPORT DOCUMENTATION PAGE						*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* August 2009	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)** March 2009–April 2009
4. TITLE AND SUBTITLE An Evaluation and Development Environment for an ARM7-Based Autopilot Using the Atmel SAM7S256 Microcontroller		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Justin L. Shumaker and Ainsmar X. Brown[*]		**5d. PROJECT NUMBER** 9BU1C2
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-VTU Aberdeen Proving Ground, MD 21005-5066		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-367
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		
13. SUPPLEMENTARY NOTES [*]National Institute of Aerospace, 100 Exploration Way, Hampton, VA 23666		

14. ABSTRACT

This report provides a detailed procedure for interfacing an Atmel SAM7S256 board with a Linux-based PC. The environment and interface are related to a new autopilot project that is designed around the ARM7 architecture used within the U.S. Army Research Laboratory's Vehicle Technology Directorate. An example project is also discussed as a means to see how the environment operates.

15. SUBJECT TERMS

ARM7, Atmel, microcontroller, autopilot

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Justin L. Shumaker
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	18	**19b. TELEPHONE NUMBER** (*Include area code*) 410-278-2834

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction

The Advanced RISC Machine (ARM) architecture has become popular by its integration into many embedded applications. Platforms include mobile phones, handheld game systems, and industrial control systems since the 1980s. The ARM architecture has since found a niche in the embedded programming community, striking an often-needed balance between power and size. These microcontrollers provide more computational ability than smaller Microchip peripheral interface controllers or AVR microcontrollers, while at the same time requiring less power and consuming less volume than larger von Neuman-based computing systems.

The particular application being presented uses the Atmel SAM7S256 board (figure 1), which is readily available in the United States from several commercial online vendors such as Sparkfun. The AT91SAM7S256 chip is currently being used internally as the centerpiece of a new small-scaled autopilot for unmanned air and ground vehicles. Though many parts of the development environment stem from already available open source material, other components were prepared internally. The most daunting task was deciphering the more than 700-page-long manual on programming the ARM7. The results were distilled into several example files as well as the accompanying header files used to program the chip accordingly.

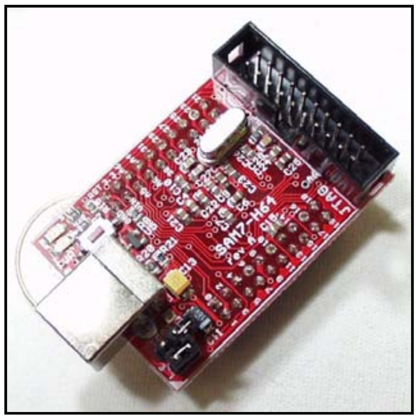


Figure 1. Atmel SAM7S256 development header board.

The following section includes step-by-step instructions on acquiring the needed hardware and software and setting up the development environment in Ubuntu Linux 8.04 for programming the ARM7 chipset. The appendix includes sample files for process verification. In order to make the material tractable to all users, including non-Linux users, additional process details are included to improve consistency in the results.

2. Procedure

The first step is to have a working installation of the Linux operating system available. It is recommended to start from a fresh Linux installation, preferably the latest version of Ubuntu. This setup has been tested with Ubuntu 8.04 long-term support but should be compatible with future iterations of the operating systems. Once Ubuntu has been properly installed, the update manager should update the system. Though it may be enough to only include the *Subversion* version repository control program and *libmpfr* from the *Synaptic Package Manager*, it is also recommended that the user installs the *build-essential*, *autoconf*, *automake*, and *libtool* packages as well. These files will update the compilation environment as discussed later on. The *apt-get* command in a terminal window is also an alternative to the *Synaptic Package Manager* method of downloading the updates, e.g., *apt-get install libtool*.

The next step is to download the development environment for the ARM7. The example files are downloaded by entering the following into the command terminal after changing the directory to the desired folder. One can create a directory *src* in the user's home directory, change to that directory, and then obtain the files as follows:

```
user@host:~$ sudo svn co --username=guest http://js.cx/svn/AT91SAM7 AT91SAM7
```

Once the files are retrieved, download the GNUARM (GNU is not Unix ARM) toolchain, which provides the modified C compiler for the ARM7 architecture. This software and additional mirrors can be found on the GNU ARM Web site.[1,2] Depending on whether the user is running a 32- or 64-bit system, a set of precompiled binaries may or may not be available. If the binaries are not available for the particular platform, then the source code must be downloaded and compiled using the *make* file (a script that sends instructions on how the program should be compiled), which will be identified as *Makefile* within the folder.

Because the current toolset is dependent on libraries from version 3.4.3 of the GNUARM toolchain, the *Makefile* in the latest version of the toolchain will have to be modified to point to the appropriate libraries. Inside the *Makefile* are hard-coded paths that may or may not point to the correct location. Look over each path and update any that appear to point in the wrong location. If it is unclear whether the paths are correct or not, check to see if the program or library exists in the specified path. Next, edit the *.bashrc* in the user's home directory through the terminal by typing:

```
user@host:~$ nano –w ~/.bashrc
```

[1] GNU ARM Home Page. http://www.gnuarm.com (accessed 3 March 2009).
[2] Amontec Home Page. http://amontec.com (accessed 13 August 2009).

Modify the *.bashrc* path file to reflect the addition of the GNUARM tool chain. Append the export path to the end of the file and save:

export PATH=$PATH:/usr/local/gnuarm-4.3.2/bin

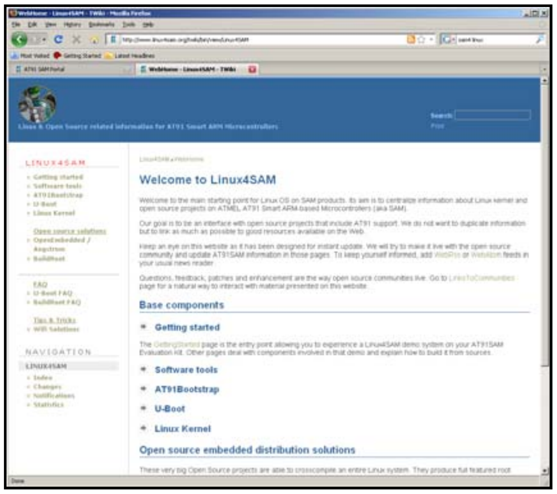
Compile the modules used by all the AT91SAM7 examples and projects by changing to the modules directory, updating the *Makefile* if necessary, and typing *make*. This action will create the *libarm.a* static library, which is essential for the examples and projects discussed herein. The *libarm.a* library, provides a number of utility functions to make serial, I^2C (Inter-Integrated Circuit), and SPI (Serial Peripheral Interface Bus) communications possible using only a couple lines of code. The library also provides additional utilities for configuring pins, setting the clock speed, managing power, performing analog-to-digital conversions, etc.

With the *libarm.a* now compiled, change the directory into the *timer* directory. This is located within the *examples* directory. Try to compile this example by typing the following two commands: *make clean* followed by *make*. If the previous steps were successful, then the file named *main.bin* will have been generated. This is the AT91SAM7S256-specific binary program that will be placed in the flash memory of the microcontroller.

Next, download the Linux SAM (Smart ARM-based Microcontroller) boot assistant from linux4sam.org[3]. Once again, the option is available to download either binaries or source code from the *software tools* menu. An explanation is also provided on the linux4sam.org Web site[3] for the procedure for mounting the USB (Univeral Serial Bus) to the serial device (figure 2) so that the computer will communicate with the ARM7 kit. It is recommended that the lines in figure 3 be inserted into a shell file for future use. If the *.sh* file is executed once, it should not be necessary to run it again on the same machine until it is rebooted again. Write the instructions in a preferred text editor such as Nano, VIM, or Emacs and save as *configure_samba.sh*. Then type *chmod +x configure_samba.sh* to make the file executable. Finally, type *./configure_samba.sh* to run the file.

At this time, the SAM7256 header board may be connected to the PC. To allow programming initially, clear the lock from the device by moving the TST (test) jumper across the two available pins and plugging in the USB outlet to a power source (figure 4). After 10 s, disconnect the USB cable from the header board and move the jumper to its original open position.

[3]Linux4SAM Home Page. http://www.linux4sam.org/twiki/bin/view/Linux4SAM/SoftwareTools (accessed 3 March 2009).



Linux & Open Source related information for AT91 Smart ARM Microcontrollers

Welcome to Linux4SAM

Welcome to the main starting point for Linux OS on SAM products. Its aim is to centralize information about Linux kernel and open source projects on ATMEL AT91 Smart ARM-based Microcontrollers (aka SAM).

Our goal is to be an interface with open source projects that include AT91 support. We do not want to duplicate information but to link as much as possible to good resources available on the Web.

Keep an eye on this website as it has been designed for instant update. We will try to make it live with the open source community and update AT91SAM information in those pages. To keep yourself informed, add WebRss or WebAtom feeds in your usual news reader.

Questions, feedback, patches and enhancement are the way open source communities live. Go to LinksToCommunities page for a natural way to interact with material presented on this website.

Base components

- Getting started

The GettingStarted page is the entry point allowing you to experience a Linux4SAM demo system on your AT91SAM Evaluation Kit. Other pages deal with components involved in that demo and explain how to build it from sources.

- Software tools
- AT91Bootstrap
- U-Boot
- Linux Kernel

Open source embedded distribution solutions

These very big Open Source projects are able to crosscompile an entire Linux system. They produce full featured root

Figure 2. Linux4SAM Web site screenshot.

```
sudo rmmod usbserial
sudo modprobe usbserial vender=0x03eb product=0x6124
sudo lsusb –d 03eb:6124
```

Figure 3. Example .sh file.

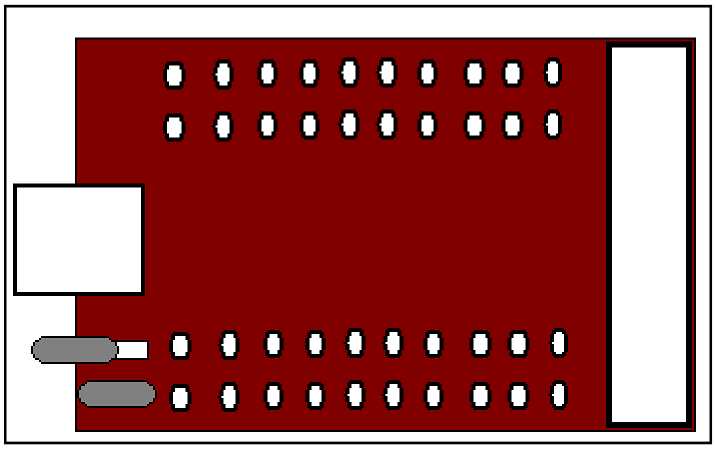


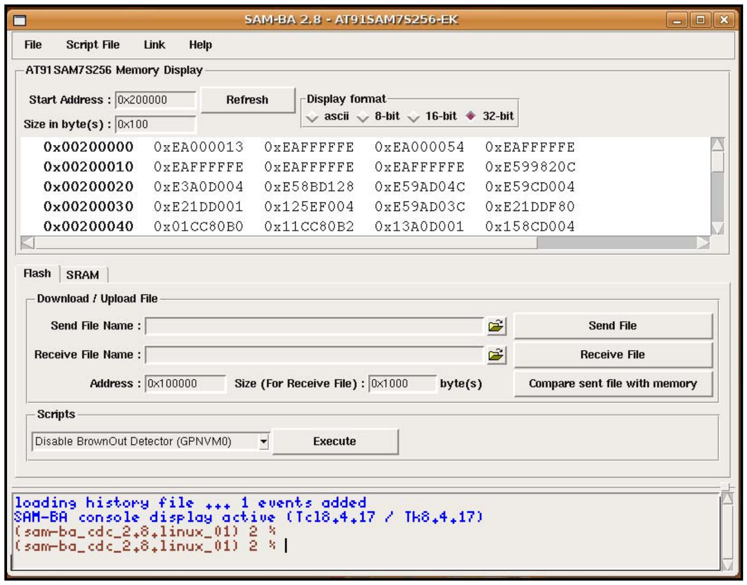
Figure 4. Diagram of header board with TST jumper removed.

The device should now be ready for programming. A useful way to determine if the communications are successful is to display a live system log in the terminal window to see if the computer recognizes the device when it is plugged in. This can be accomplished by typing the following line in the command window:

user@host:~$ tail –f /var/log/syslog

The *tail –f* command appends the most recent lines from the file *syslog* in */var/log/* to the command window display. From *syslog*, the user is able to view the current system actions including the response to any external USB devices recently added. The display to the window should resemble *new full speed USB device using ehci_hcd and address X*, where *X* may be any number applicable to a particular machine.

Once the system log file indicates that a new device (i.e., */dev/ttyUSB0*) has been created, run the SAM-BA (Smart ARM-based Microcontrollers Boot Assistant) application to begin programming the chip (figure 5). (Change the directory to where SAM-BA was installed, which most likely will be */home/user_name/src/sam-ba_cdc_2.8.linux_01/*. Run SAM-BA by typing *./sam-ba_cdc_2.8.linux_01* (one may have to input *chmod +x sam-ba_cdc_2.8.linux_01* to mark the file as executable). The current device should already be selected as the current connection */dev/ttyUSBX*, where *X* can be any number depending on the number of other serial-to-USB devices connected to the machine. Also select *AT91SAM7S256-EK* as the board to program. In the window that opens up, select the folder to the left of the *send file* button and locate the *main.bin* from the timer directory. Once the *main.bin* is selected, click the *send file* button and answer yes to any prompts regarding locking and unlocking the module.



Start Address : 0x200000
Size in byte(s) : 0x100
Display format: ascii, 8-bit, 16-bit, [✓ 32-bit]

```
0x00200000   0xEA000013   0xEAFFFFFE   0xEA000054   0xEAFFFFFE
0x00200010   0xEAFFFFFE   0xEAFFFFFE   0xEAFFFFFE   0xE599820C
0x00200020   0xE3A0D004   0xE58BD128   0xE59AD04C   0xE59CD004
0x00200030   0xE21DD001   0x125EF004   0xE59AD03C   0xE21DDF80
0x00200040   0x01CC80B0   0x11CC80B2   0x13A0D001   0x158CD004
```

Send File Name :
Receive File Name :
Address : 0x100000   Size (For Receive File) : 0x1000 byte(s)
Scripts: Disable BrownOut Detector (GPNVM0)

```
loading history file ... 1 events added
SAM-BA console display active (Tcl8.4.17 / Tk8.4.17)
(sam-ba_cdc_2.8.linux_01) 2 %
(sam-ba_cdc_2.8.linux_01) 2 %
```

Figure 5. SAM-BA graphical user interface screenshot.

At this point, the header boards should be programmed. Disconnect and then reconnect the USB cable or power source from the development board. The status light-emitting diode (LED) on the development board should now be blinking. The user may return to the *main.c* file for the timer and change the frequency at which the LED on the header board will blink by changing the DIV128 expression to DIV32. This same general procedure may be followed to run any of the other examples provided or any original code that may be generated by the user for future use. It may be useful to copy a working *Makefile* from another example directory instead of modifying it each time.

3. Future Applications

The development environment is arranged to allow easy transition when programming the ARM7 autopilot. In the future, filtering and estimation methods will be applied to the autopilot, which will then be used as a controller for small- and microscale air and ground vehicles within the scope of the Vehicle Technology Directorate's unmanned systems research.

Appendix. Sample Code

This appendix appear in its original form, without editorial change.

```
#include "board.h"
#include "pio.h"
#include "tc.h"
#include "lowlevel.h"

unsigned int FiqCount = 0;

static void
TimerIrqHandler (void)
{
  AT91C_BASE_TC2->TC_SR; /* read TC Status Register to clear interrupt */

  if  ((AT91C_BASE_PIOA->PIO_ODSR & LEDSTATUS) == LEDSTATUS)
  {
    AT91C_BASE_PIOA->PIO_CODR = LEDSTATUS; /* turn status on */
  }
  else
  {
    AT91C_BASE_PIOA->PIO_SODR = LEDSTATUS; /* turn status led off */
  }
}


int
main (void)
{
  low_level_init (EXT_OSC, PLL_DIV, PLL_MUL, PRESCALE);

  /* PIO Enable, AB Select (0=A), Output Enable, Default Output State, Pull-Up Enable, Filter
Enable, PCINT Enable, Multi-Drive Enable */
  pio_init (LED_MASK, 0xFFFFFFFF, 0, LED_MASK, LED_MASK, 0, 0, 0, 0);

  /* Timed Interrupt Example */
  tc_init (TC2, TC_DIV32, TC_COMPARE, 37500, TimerIrqHandler, 4);

  while (1) {}
}
```

Figure A-1. Time example.

```
# ***************************************************************
# Makefile for flash execution
# Use "make 64" for AT91SAM7S64 or "Make 256" for AT91SAM7S256.
# Default is "make 64"
# ***************************************************************

# variables
CC      = arm-elf-gcc
LD      = arm-elf-ld -v
AR      = arm-elf-ar
AS      = arm-elf-as
CP      = arm-elf-objcopy
OD      = arm-elf-objdump

CFLAGS            = -I./ -Imodules/ -c -fno-common -Wall -Os
CASMFLAGS   = -Imodules/ -Os -c -g -Wa,-a,-ad
# AFLAGS          = -ahls -mapcs-32 -o crt.o
# LFLAGS          =  -Map main.map -T$(LINKER_SCRIPT)
CPFLAGS           = --output-target=binary
ODFLAGS           = -x --syms
GNUARM            = /usr/local/gnuarm-3.4.3
OUT         = libarm7.a
#GNUARM           = /c/Program\ Files/GNUARM

OBJECTS = tc.o isrsupport.o lowlevel.o status_led.o usart.o adc.o pio.o pmc.o
usb.o pwm.o spi.o vreg.o wdt.o aic.o twi.o pdc.o
OBJECTS_LST = tc.lst isrsupport.lst lowlevel.lst status_led.lst usart.lst
adc.lst pio.lst pmc.lst usb.lst pwm.lst pwm.lst spi.lst vreg.lst wdt.lst
aic.lst twi.lst pdc.lst

64: CFLAGS += -DMCU=64
64: LINKER_SCRIPT = AT91SAM7S64.ld
64: $(OUT)

256: CFLAGS += -DMCU=256
256: LINKER_SCRIPT = AT91SAM7S256.ld
256: $(OUT)

clean:
      -rm -f $(OBJECTS) $(OBJECTS_LST) $(OUT)

$(OUT): $(OBJECTS) $(LINKER_SCRIPT)
      @ echo "..linking"
#     $(AR) rcs $(OUT) $(OBJECTS)
      $(AR) rcs $(OUT) $(OBJECTS) $(GNUARM)/arm-elf/lib/libc.a $(GNUARM)/arm-
elf/lib/libm.a $(GNUARM)/lib/gcc/arm-elf/3.4.3/libgcc.a $(GNUARM)/arm-
elf/lib/libg.a

lowlevel.o: lowlevel.c
      $(CC) $(CASMFLAGS) lowlevel.c > lowlevel.lst
      $(CC) $(CFLAGS) lowlevel.c

tc.o: tc.c
      $(CC) $(CASMFLAGS) tc.c > tc.lst
      $(CC) $(CFLAGS) tc.c

isrsupport.o: isrsupport.c
```

Figure A-2. Makefile for building modules.

```
      $(CC) $(CASMFLAGS) isrsupport.c > isrsupport.lst
      $(CC) $(CFLAGS) isrsupport.c

status_led.o: status_led.c
      $(CC) $(CASMFLAGS) status_led.c > status_led.lst
      $(CC) $(CFLAGS) status_led.c

usart.o: usart.c
      $(CC) $(CASMFLAGS) usart.c > usart.lst
      $(CC) $(CFLAGS) usart.c

adc.o: adc.c
      $(CC) $(CASMFLAGS) adc.c > adc.lst
      $(CC) $(CFLAGS) adc.c

pio.o: pio.c
      $(CC) $(CASMFLAGS) pio.c > pio.lst
      $(CC) $(CFLAGS) pio.c

pmc.o: pmc.c
      $(CC) $(CASMFLAGS) pmc.c > pmc.lst
      $(CC) $(CFLAGS) pmc.c

usb.o: usb.c
      $(CC) $(CASMFLAGS) usb.c > usb.lst
      $(CC) $(CFLAGS) usb.c

pwm.o: pwm.c
      $(CC) $(CASMFLAGS) pwm.c > pwm.lst
      $(CC) $(CFLAGS) pwm.c

spi.o: spi.c
      $(CC) $(CASMFLAGS) spi.c > spi.lst
      $(CC) $(CFLAGS) spi.c

vreg.o: vreg.c
      $(CC) $(CASMFLAGS) vreg.c > vreg.lst
      $(CC) $(CFLAGS) vreg.c

wdt.o: wdt.c
      $(CC) $(CASMFLAGS) wdt.c > wdt.lst
      $(CC) $(CFLAGS) wdt.c

aic.o: aic.c
      $(CC) $(CASMFLAGS) aic.c > aic.lst
      $(CC) $(CFLAGS) aic.c

twi.o: twi.c
      $(CC) $(CASMFLAGS) twi.c > twi.lst
      $(CC) $(CFLAGS) twi.c

pdc.o: pdc.c
      $(CC) $(CASMFLAGS) pdc.c > pdc.lst
      $(CC) $(CFLAGS) pdc.c
```

Figure A-2. Makefile for building modules (continued).

NO. OF COPIES	ORGANIZATION
1 (PDF only)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FORT BELVOIR VA 22060-6218
1	DIRECTOR US ARMY RESEARCH LAB IMNE ALC HRR 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB RDRL CIM L 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB RDRL CIM P 2800 POWDER MILL RD ADELPHI MD 20783-1197
	ABERDEEN PROVING GROUND
1	DIR USARL RDRL CIM G (BLDG 4600)

INTENTIONALLY LEFT BLANK.