

Copyright
by
Warren Clay Grant
2012

**The Report Committee for Warren Clay Grant
Certifies that this is the approved version of the following report:**

**Implementation of an Open Source JTAG Debugging Development
Chain for the BeagleBoard ARM[®] Cortex A-8**

**APPROVED BY
SUPERVISING COMMITTEE:**

Supervisor:

Jacob Abraham

Mark McDermott

**Implementation of an Open Source JTAG Debugging Development
Chain for the BeagleBoard ARM[®] Cortex A-8**

by

Warren Clay Grant, BSInfTech; MBA

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2012

Dedication

This report is dedicated to my wife Denise for her patience and dedicated support that made attaining this goal possible. Her understanding during my long hours for study while working full time is appreciated and will require a lot to make up.

Acknowledgements

I would like to acknowledge the professors, instructors, guest lecturers at the University of Texas. They all were truly experts in their respective fields and shared a volume and quality of knowledge that would otherwise have been out of reach.

May, 2012

Abstract

Implementation of an Open Source JTAG Debugging Development Chain for the BeagleBoard ARM[®] Cortex A-8

Warren Clay Grant, MSE

The University of Texas at Austin, 2012

Supervisor: Jacob Abraham

The BeagleBoard-xM, manufactured by Texas Instruments, is a small, low cost, open source development platform for the ARM[®] Cortex-A8 processor. This paper implements a hardware and software combination to connect to the ARM[®] processor via a JTAG connection for debugging. A FlySwatter interface board is utilized to connect the JTAG port to a host computer and a combination of software tools are implemented to demonstrate the capability for debugging the Linux kernel. The necessary files for booting the Linux 3.0 kernel were compiled and loaded on the BeagleBoard-xM and the host computer. Installation and selection of the components that make up the software tool chain are described. All the hardware and software used for this project are open source designs.

Table of Contents

List of Tables	viii
List of Figures	ix
List of Illustrations	x
<i>Introduction</i>	1
Chapter 1: <i>Hardware</i>	2
BeagleBoard.....	2
Processor	4
ARM [®] Basics.....	5
JTAG on the BeagleBoard	6
FlySwatter.....	10
Setup.....	12
Chapter 2: <i>Software</i>	14
GNU ARM Toolchain	14
BeagleBoard-xM	14
SD Card Setup	15
The Linux Kernel.....	16
OpenOCD.....	20
GNU Debugger and Insight	25
Chapter 3: <i>Problems and Forward Work</i>	29
Chapter 4: <i>Conclusion</i>	31
Appendix.....	32
References.....	33
Vita	34

List of Tables

Table 1, JTAG 14-pin Adapter	11
Table 2, Project Hardware	12
Table 3, OMAP Kernel Patches [11].....	18
Table 4, OpenOCD Installation	22

List of Figures

Figure 1. BeagleBoard-xM Block Diagram [1]	3
Figure 2, Cortex-A8 Block Diagram [2]	5
Figure 3, JTAG Scan Chain Example	8
Figure 4, JTAG TAP State Machine [2].....	10

List of Illustrations

Illustration 1, BeagleBoard-xM with Dimensions [1].....	3
Illustration 2, FlySwatter Adapter and JTAG Cable [5].....	7
Illustration 3, FlySwatter in Circuit Debugger	12
Illustration 4, BB-xM and Flyswatter Setup.....	13
Illustration 5, Overall Hardware Setup.....	13
Illustration 6 Port, Target Board Boot Screen at Serial Port	19
Illustration 7, Terminal Screen at Debian Prompt	20
Illustration 8, Scan Chain Results	23
Illustration 9, OpenOCD "halt" Window	26
Illustration 10, "start_kernel" in "main.c"	27
Illustration 11, Insight "head.s" Window	27
Illustration 12, Insight Debug Window	28
Illustration 13, Loading the Kernel	29

Introduction

The use of embedded systems is widespread and continues to increase. As more applications for embedded systems are identified, requirements for computing power and complexity are also rising. Writing and debugging software for these applications also presents a challenge. A hardware and software platform for their design can help gain a working knowledge and understanding for specific implementations of the embedded system.

This project attempts to establish a hardware platform with a JTAG debugging development chain using open source based products. The hardware consists of a Texas Instruments (TI) BeagleBoard-XM which contains an ARM[®] Cortex-A8 processor, a FlySwatter JTAG dongle from TinCanTools, and a PC running the Linux operating system that serves as the host computer. The BeagleBoard-XM and the JTAG dongle used are open source hardware designs. Open source software is used to implement and document the development tool chain and provides the ability actively debug code for the hardware target. The software includes OpenOCD and GDB implemented via Insight (a GDB GUI). The GNU ARM[®] toolchain was used to compile programs for the ARM[®] architecture.

Chapter 1: *Hardware*

BEAGLEBOARD

The BeagleBoard was designed by TI to provide a low cost way to explore open source hardware and software capabilities. It has the additional advantage of being a single board computer designed for low power and occupies a small foot print, approximately 85 X 86 mm. Though not intended as a full development platform, it was developed as a community supported platform to develop community software baselines and it provide the needed capabilities to implement a debugging toolchain for the purposes of this project.

Two versions of the BeagleBoard are available; BeagleBoard Rev C4 and the BeagleBoard-xM shown in Illustration 1. Bothe versions provide support for an SD card that can act as program storage space and provide the equivalent function of a hard drive that would be present on a on a personal computer. This project uses the BeagleBoard-xM which has several advantages such as a higher processor speed of 1 GHz, an RS-232 connector not provided in the C4 version and a four port USB hub with Ethernet. A more complete list of features is provided in Appendix A.

This board does not have NAND flash memory. The micro-SD card is accessed during boot-up. This is an advantage since mistakes can lead to a corruption of the NAND and render the board useless without a significant amount of effort to recover the NAND via the JTAG connection. Given that the board is used for experimenting and development, the elimination of this an issue provides a level of comfort and potentially a substantial time and cost savings.

Figure 1 shows a block diagram of the BeagleBoard. Overvoltage protection is another feature that prevents damage to the board when powering via the 5V DC

connection. The processor is a DM3730 System on Chip (SOC) design. The DM3730 is a Package on Package configuration where the 512MB is mounted on top of the processor.

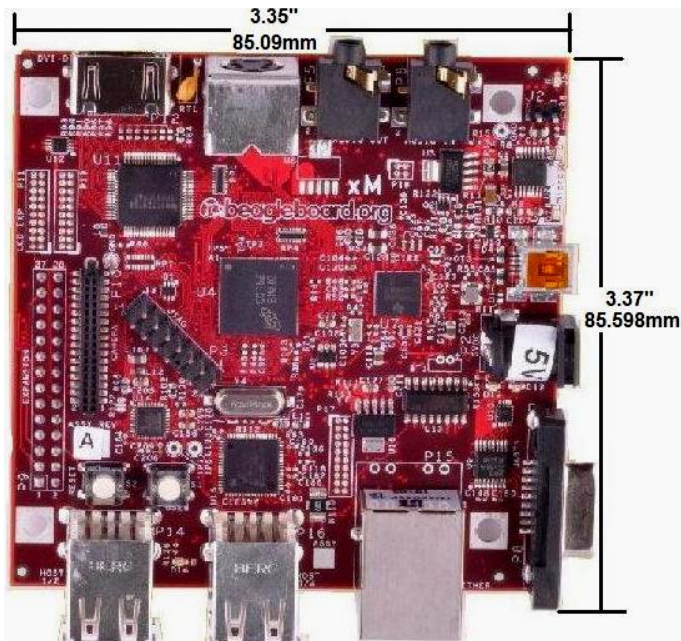


Illustration 1, BeagleBoard-xM with Dimensions [1]

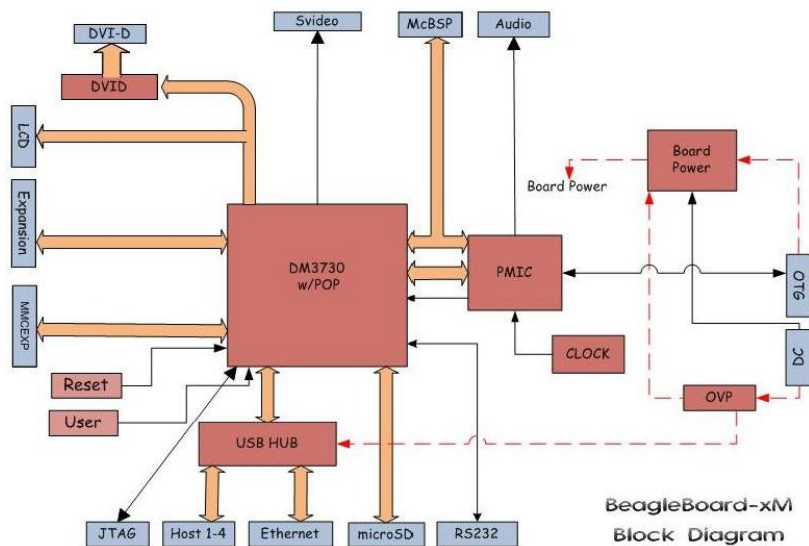


Figure 1. BeagleBoard-xM Block Diagram [1]

Processor

The processor in the BeagleBoard-xM has a DM3730 processor as opposed to the C4 version which contains an OMAP3530 processor [1]. The DM3730 is a System on Chip (SOC) design that includes a digital media processor, the ARM[®] Cortex-A8 Microprocessor, a graphics accelerator, and other features while maintaining compatibility with the OMAP 3 architecture.

A few of the key features of the Cortex-A8 processor include the following [2]:

- full implementation of the ARM[®] architecture v7-A instruction set
- Architecture (AMBA) with Advanced Extensible Interface (AXI) for main memory interface supporting multiple outstanding transactions
- a pipeline for executing ARM[®] integer instructions
- Memory Management Unit (MMU) and separate instruction and data Translation
- Look-aside Buffers (TLBs) of 32 entries each
- Embedded Trace Macrocell (ETM) support for non-invasive debug
- ARMv7 debug with watchpoint and breakpoint registers and a 32-bit Advanced Peripheral Bus (APB) slave interface to a CoreSight[™] debug system

The processor implements the ARMv7 Debug architecture that includes support for CoreSight[™]. The ARM[®] Debug Interface (ADIV5) [3] is designed to be compatible with the ARM[®] CoreSight[™] architecture [4]:

- a CoreSight[™] interface implementation is a valid implementation of ADIV5
- the ADIV5 specification does not require an ADI to be CoreSight[™] compliant

The Embedded Trace Macrocell (ETM) unit is a non-intrusive trace macrocell that filters and compresses an instruction and data trace for use in system debugging. The

Trace information such as executed instruction addresses, instruction condition codes, and exception information is generated via the ETM. ETM unit has an external interface outside of the processor called the Advanced Trace Bus (ATB) interface. There is limited access to this interface, however, and it is considered a rare case for accessing this interface using OpenOCD. OpenOCD, used in this project, accesses the processor via the JTAG interface and the APB interface as shown in Figure 2. The ETM can be accessed via the APB.

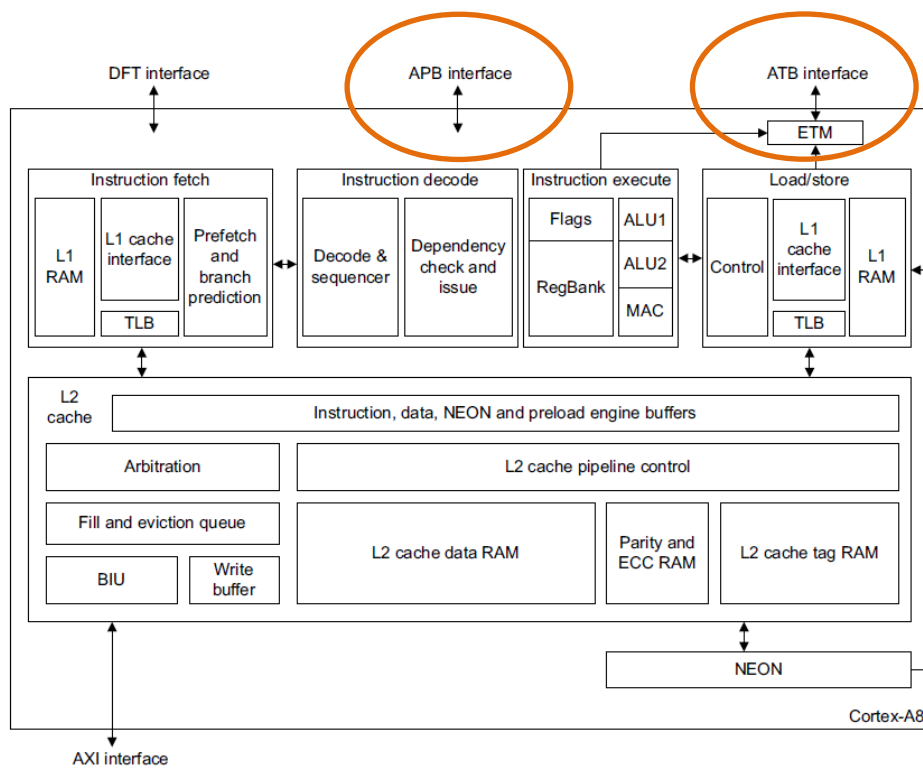


Figure 2, Cortex-A8 Block Diagram [2]

ARM® Basics

ARM® processors have thirty seven, thirty two bit long registers of which thirty are general purpose registers available according to which of up to 7 modes are selected. The processor modes are User where most tasks are executed, fast interrupt request (FIQ)

for high priority interrupt, interrupt request (IRQ) for lower priority interrupt, Supervisor for reset and software interrupt instruction, Abort for memory access violation handling, Undefined for undefined instruction handling, and System which is a privileged mode using the same registers as the user mode. Each mode can access a specific set of r0 – r12 registers, a program counter register R15, a stack register R13, and the current program status register (CPSR). The saved program status register (SPSR) can be accessed in privileged modes.

The application binary interface (ABI) is a calling convention that specifies the interface between applications and the operating system. The Embedded ABI (EABI) specifies standards for register usage, data types, file formats, et. al. for an embedded system [5]. According to the convention argument passing is performed using registers r0–r3, r4–r11 are used for local variables, r12 is used as a function call scratch register, r13 as the stack pointer, r14 as the link register, and r15 as the program counter [6].

JTAG on the BeagleBoard

The JTAG connection accesses the Debug Port supported by ARM[®] for the CoreSight[™] architecture specification [4]. The JTAG port is IEEE 1149.1 compliant and uses the standard nTRST, TCK, TMS, TDI, and TDO signals. Two instrumentation pins, EMU0 and EMU1 are also used and determine the initial scan chain configuration. As noted in the DM37x technical reference manual, the EMU power must have pull up resistor equal to 10k Ω installed before starting the debugger [4]. The JATG cable adapter for the FlySwatter shown in Illustration 2 has jumpers available for EMU1 and 2. The jumpers must be placed correctly between 1 and 2 in lieu of 0 and 1 to enable initialization of the scan chain.

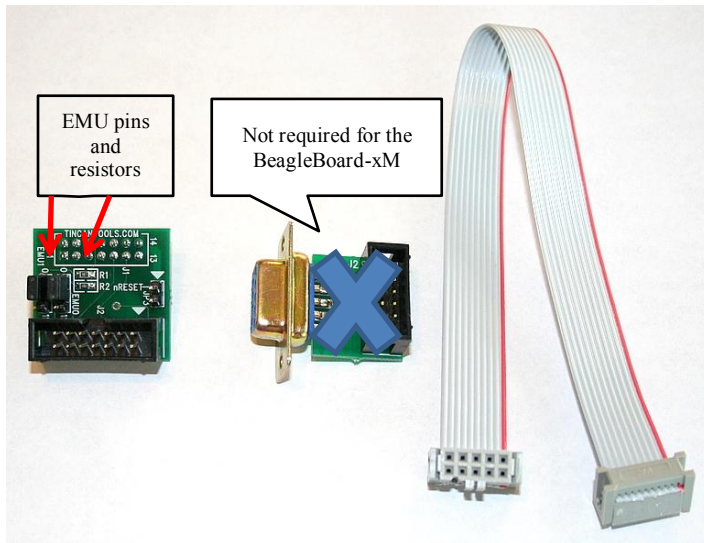


Illustration 2, FlySwatter Adapter and JTAG Cable [5]

By pulling EMU0 and EMU1 high, the board is placed in the TAP router-only mode. The TAP router exists to control up to 16 TAP controllers. In the router-only mode no secondary TAPs are selected and the TAP router is the only TAP between TDI and TDO (data in and data out) [4].

TAPs are added to the scan chain by programming the tap router. Once this is accomplished then the debug access port (DAP) is added to the scan chain. The DAP is interfaced to the APB shown in Figure 2 and discussed previously. The debugger then has access to the entire memory space without the need for sending the processor into a debug state [4]. Modules on the DM3730 are mapped to the DAP address. OpenOCD executes the IDCODE instruction to return information about devices in the chain. The terminal output showing the ROM table from OpenOCD:

```

> dap info 1
AP ID register 0x04770002
    Type is MEM-AP APB
AP BASE 0x80000000
ROM table in legacy format
MEMTYPE System memory not present. Dedicated debug bus.
ROMTABLE[0x0] = 0xd4010003
    Component base address 0x54010000, start address 0x54010000
    Component class is 0x9, CoreSight component
    Type is 0x13, Trace Source, Processor
    Peripheral ID[4..0] = hex 04 20 6b b9 21
    Part is Cortex-A8 ETM (Embedded Trace)
ROMTABLE[0x4] = 0xd4011003
    Component base address 0x54011000, start address 0x54011000
    Component class is 0x9, CoreSight component
    Type is 0x15, Debug Logic, Processor
    Peripheral ID[4..0] = hex 04 20 6b bc 08
    Part is Cortex-A8 Debug (Debug Unit)
ROMTABLE[0x8] = 0xd4012003
    Component base address 0x54012000, start address 0x54012000
    Component class is 0x9, CoreSight component
    Type is 0x64, Debug Control, Reserved
    Peripheral ID[4..0] = hex 00 00 09 71 13
    Part is *- unrecognized *-
ROMTABLE[0xc] = 0xd4013002
    Component not present
ROMTABLE[0x10] = 0xd4019003
    Component base address 0x54019000, start address 0x54019000
    Component class is 0x9, CoreSight component
    Type is 0x11, Trace Sink, Port
    Peripheral ID[4..0] = hex 04 00 1b b9 12
    Part is Coresight TPIU (Trace Port Interface Unit)
ROMTABLE[0x14] = 0xd401b003
    Component base address 0x5401b000, start address 0x5401b000
    Component class is 0x9, CoreSight component
    Type is 0x21, Trace Sink, Buffer
    Peripheral ID[4..0] = hex 04 00 0b b9 07
    Part is Coresight ETB (Trace Buffer)
ROMTABLE[0x18] = 0xd401d003
    Component base address 0x5401d000, start address 0x5401d000
    Component class is 0xf, PrimeCell or System component
    Peripheral ID[4..0] = hex 00 00 09 73 43
    Part is TI DAPCTL
ROMTABLE[0x1c] = 0xd4500003
    Component base address 0x54500000, start address 0x54500000
    Component class is 0x9, CoreSight component
    Type is 0x63, Trace Source, Reserved
    Peripheral ID[4..0] = hex 00 00 19 71 20
    Part is TI SDTI (System Debug Trace Interface)
ROMTABLE[0x20] = 0x0
    End of ROM table

```

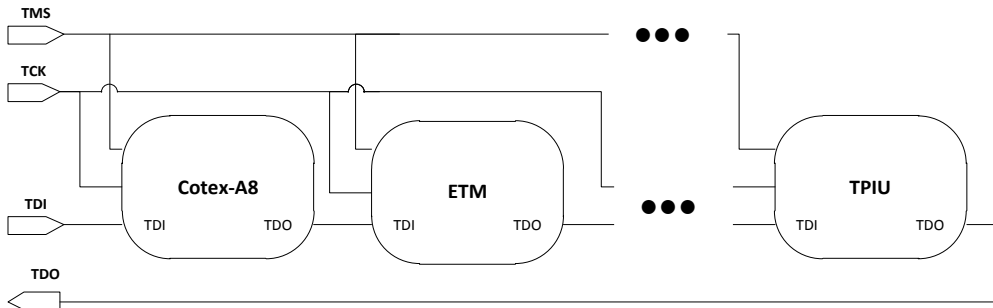


Figure 3, JTAG Scan Chain Example

The technical reference manual [4] lists the modules as the ETM, the Cortex-A8, the trace port interface unit (TPIU), and the embedded trace buffer (ETB) module. Note from the listing above that 2 additional modules are identified along with one that is “unrecognized” because it is “reserved” by the architecture and one that is listed as not present. The DAPCTL sounds like it may be a DAP control module, but the documentation does not go into detail about its function. The SDTI (System Debug Trace Interface) module is described as implementing system trace during debug emulation and details of its configuration, protocol, data format and function are provided in manual [4].

One other key aspect of the CoreSight™ architecture should be mentioned. The DAP is an implementation of the ADiv5 by way of CoreSight™ DAP-Lite [5]. The components that make up the DAP-Lite interface to the board/processor are the debug ports (DP) and access ports (AP). Note that the AP and DP together are referred to as the DAP. The DAP implements the JTAG-DP which sets up the JTAG connection to the debugger and host computer.

The JTAG-DP operation is controlled by an IEEE 1149.1 compliant state machine. The TMS signal that queries the controller is shown as an example in Figure 4.

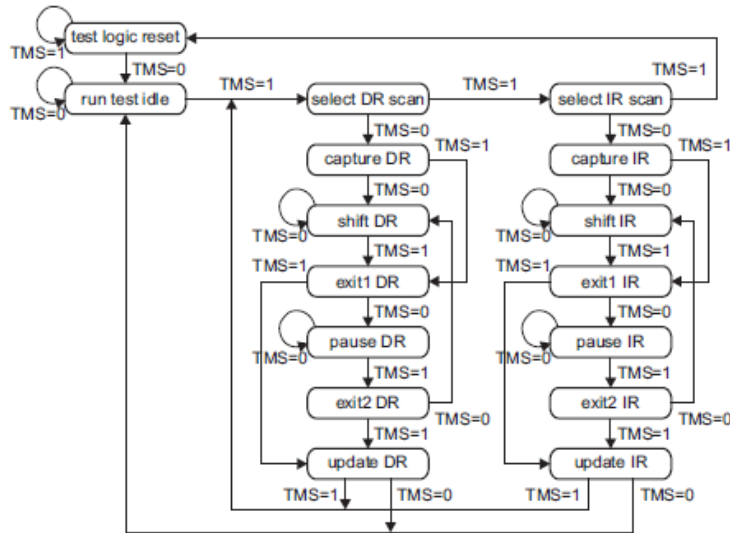


Figure 4, JTAG TAP State Machine [2]

The data is sent and received serially. The IR and DR in Figure 4 are the instruction and data registers respectively. The state machine can access data, instructions, and perform a reset. The clock signal, TCK, provides the means to step through the state machine. To load IR values, for example, the controller must be in the Shift IR state. Data is transferred on each clock pulse via TDI and TDO.

One last point on the JTAG controller needs to be made concerning the ICEPick module. The ICEPick module allows the controller to select which subsystem the TAPs are accessible to in multiple processor systems. If a subsystem is powered down for any reason, such as for power savings, the scan chain would be interrupted and the JTAG connection would fail. The ICEPick allows the powered down system to be ignored so the other subsystems can be accessed. The ICEPick also manages the power, clock and reset for each TAP. Since the DM3730 has a Digital Signal Processor, and other subsystems, the ICEPick module is included. All accesses to the JTAG signals are accomplished via the ICEPick module.

FlySwatter

The FlySwatter is the in-circuit debugger. It provides the interface between the USB port on the host computer and the JTAG connection on the target computer, in this case the BeagleBoard-xM. A standard 14 pin JTAG connector is provided and matches up with the 14 pin connector on the BeagleBoard-xM via an adapter made for that purpose. The adapter is shown in Illustration 2. The adapter converts from the standard 14 pin JTAG layout to the ARM[®] specific layout on the board shown in Table 1.

ARM-14-JTAG FlySwatter			TI-14-JTAG BeagleBoard-xM		
VREF	1 - - 2	GND	JTAG_TMS	1 - - 2	JTAG_nTRST
JTAG_nTRST	3 - - 4	GND	JTAG_TDI	3 - - 4	GND
JTAG_TDI	5 - - 6	GND	VREF	5 - x 6	KEY (empty)
JTAG_TMS	7 - - 8	GND	JTAG_TDO	7 - - 8	GND
JTAG_TCK	9 - - 10	GND	JTAG_RTCK	9 - - 10	GND
JTAG_TDO	11 - - 12	JTAG_SRST_N	JTAG_TCK	11 - - 12	GND
VREF	13 - - 14	GND	JTAG_EMU0	13 - - 14	JTAG_EMU1

Table 1, JTAG 14-pin Adapter

An RS232 interface is also provided and supports modem protocols. Both the RS-232 and USB to JTAG interface utilizes Future Technology Devices International Ltd. (FTDI's) FT232 Dual USB UART/FIFO¹. The FT232 has two ports, A and B. Port A is used for the USB to JTAG interface and Port B is for the serial UART. The FlySwatter provides a standard 14-pin JTAG interface as well as a standard RS232 port with support for full modem signals. The USB 2.0 standard is supported and it supports a number of different target system voltages. Though 3.3V appears to be the most popular, the board for this project requires a JTAG connector voltage of 1.8V. The FlySwatter board is also supported in OpenOCD which provides an interface for it.

The FlySwatter provides the interface for OpenOCD to find the DAP, check for power and timing information, get the ROM table and compare the peripheral IDs [4].

Illustration 3 shows the Flyswatter as used for this project with the adapter, RS-232 extender, and JTAG cable installed.

¹ FTDI manufactures the FT232 dual chip and provides drivers on its website www.ftdichip.com. The drivers are also included in most of the recent Linux versions.



Illustration 3, FlySwatter in Circuit Debugger

Setup

The hardware setup consists of the components listed in Table 2.

Project Hardware
Host Computer (HP 2133 Netbook)
USB Cable
FlySwatter in Circuit Debugger
JTAG 14 Pin Adapter
RS-232 extender
BeagleBoard-xM Target Computer
5V power cord for BeagleBoard-xM

Table 2, Project Hardware

The BeagleBoard-xM has the added advantage of having an RS-232 connector on the board. Previous versions did not have that available, though the adapter kit supplied one with the JTAG 14 pin adapter. A protective case was added to prevent damage by

physical, or electrostatic, means. The BeagleBoard-xM is shown connected to the FlySwatter with associated cables in the illustration below.

The host computer is an HP 2133 netbook with Ubuntu Linux 11.10 installed at the beginning of the project. The overall hardware setup is shown in Illustration 5.

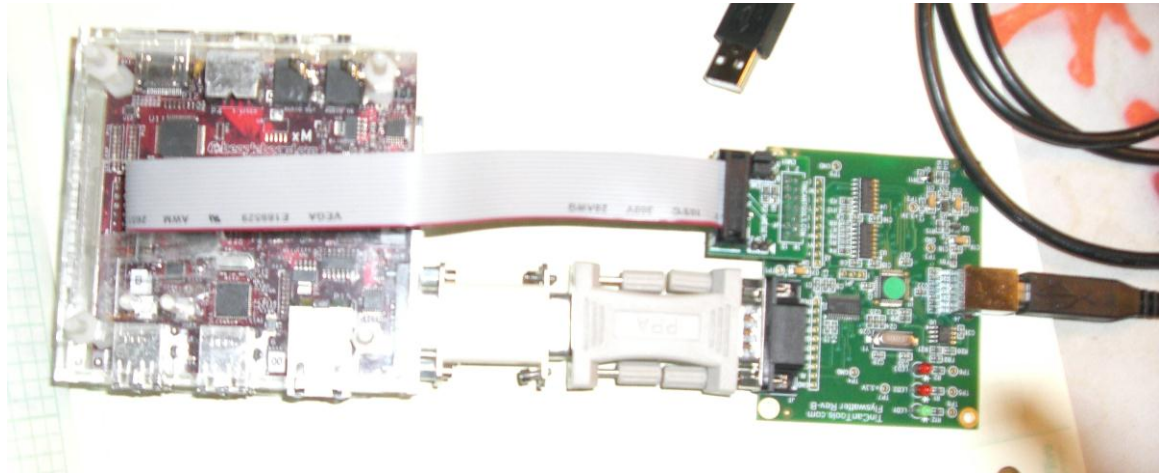


Illustration 4, BB-xM and Flyswatter Setup

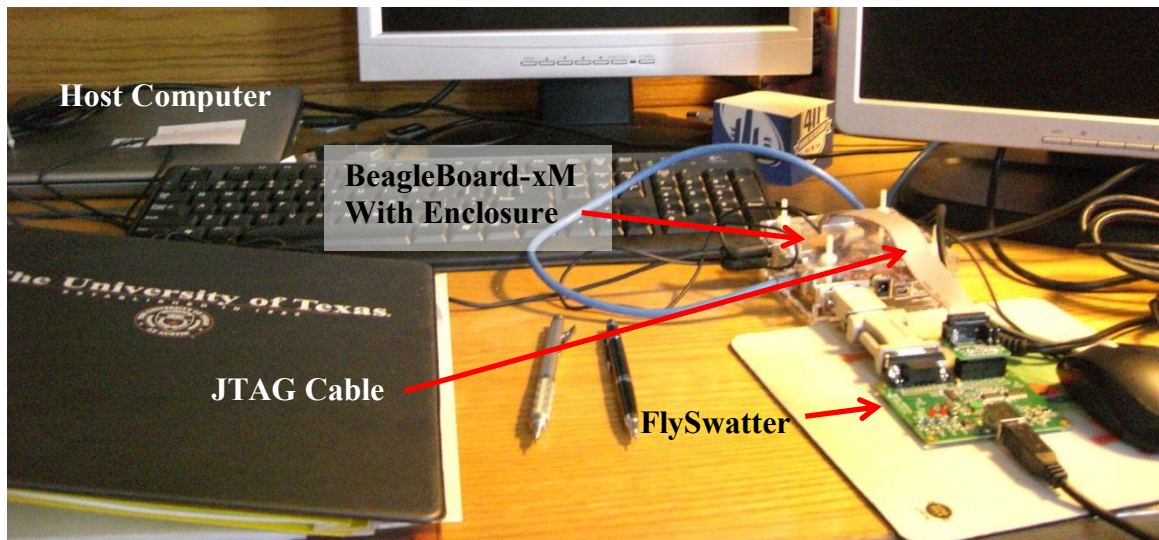


Illustration 5, Overall Hardware Setup

Chapter 2: *Software*

GNU ARM TOOLCHAIN

Compiling code on a host computer to run on a target computer with a different hardware architecture requires a cross compiler. Particularly for embedded systems, programming on a host computer can be faster and more convenient. It may also be necessary in the case where a boot loader and operating system need to be developed for the embedded system before it can run. The host computer for this project is Linux based (UBUNTU 10.10). The Linaro ARM cross tool chain (version 4.6.2-14) is an open source tool chain that was installed with little effort² using “`apt-get install gcc-arm-linux-gnueabi.`” Setting the PATH to the “`usr/arm-linux-gnueabi/bin`” folder then enables use of the cross compiler in other directories. The tool chain includes the GNU binutils, the GNU C compiler (gcc), and the glibc library. The tools required for compiling the uboot file and libncurses (provides capability for an ASCII based GUI) are also needed. Each of these can be downloaded separately and compiled on the host machine, but unlike other tool chains such the CodeSourcery G++ Lite toolchain, all of the Linaro toolchain [8] source is covered by the GNU General Public License (GPL) and is freely available.

BEAGLEBOARD-XM

The BeagleBoard-xM has no flash and boots directly from a microSD card. The SD card must be setup properly in order for the system to boot and load the Linux operating system. A boot partition (FAT32 file system format for this project) on the SD card is required with MLO, uEnv.txt, and u-boot.img files. The ROM code is designed to detect file allocation table (FAT) format types, so it will not boot from a Linux formatted partition. The Linux file system is located on a second ext3 partition. The card acts like a

² In order to get the Linaro toolchain an additional repository must be added (“`sudo add-apt-repository ppa:linaro-maintainers/toolchain`”).

hard drive since the master boot record (MBR) was created in the first partition during format. A floppy drive-like configuration is also supported [2].

The Multimedia Card Loader (MMC Loader or MLO) is the image read by the system read only memory (ROM) for the boot procedure. The ROM code checks for a valid MBR signature of 0xAA55 at offset 01FEh. Once found and other requisite conditions are checked successfully, the ROM code performs a translation of each FAT entry corresponding to the MLO file and places the results in a buffer. The booting procedure then refers to the buffer for accessing the file [2]. The general purpose memory controller (GPMC) is used to access NAND such as that on earlier versions of the BeagleBoard. Instructions for accessing the NAND via the GPMC are contained in the MLO file. The MLO file then directs boot up via the boot loader contained in the file u-boot.img. The boot loader then passes system information to, and executes, the kernel.

The MLO file can be downloaded directly from a number of sources available via a quick Google search or the one supplied with the board can be used. Since an understanding and hands-on application of the software implementation is desired, the MLO file was compiled from source [8]. The source code and patches were downloaded and compiled using the Linaro Arm tool chain. The u-boot.img file is included in the source and is also produced during compilation.

SD Card Setup

As noted previously the ARM[®] Cortex-A8 on the BeagleBoard-xM requires two partitions, a FAT and an ext3. GParted is an open source disk partitioning tool that has a graphical user interface (GUI). Gparted was used to set up the boot and ext3 partitions on a 4MB SD card. The boot partition size selected is 65MB and the rest was allocated to

ext3 partition, about 3.7GB, for the file system. The MLO and u-boot.img were then copied to boot partition along with a uEnv.txt file.

A boot.scr file was supplied on the SD card that came with board when purchased. It contains for the boot parameters to be used. More recent versions of u-boot reference the uEnv.txt file instead of the boot.scr. To change parameters using boot.scr a new image would have to be made using a boot.cmd text file for each set of modifications. The uEnv.txt file is plain text and can be changed easily. This saves time and eliminates several additional steps. This is particularly helpful when trying out different parameters to observe the effects on the system, even when they are as simple as changing the display resolution which was modified for this project. The compressed kernel image file and address to begin loading the kernel are also specified here.

The last file to go onto the boot partition is the compressed Linux kernel image, zImage which is produced by compiling the kernel. The second partition receives the root file system and additional Linux modules. The modules were installed in a separate directory and compressed during compilation (a script file was used from reference [10] with minor changes). The file system and module were then uncompressed to the rootfs partition of the SD card for use on the target board.

There are some additional tools required for successful setup of the SD card to go into the target computer. “u-boot-tools”

The Linux Kernel

Cross compiling the Linux kernel can be a daunting task. The latest kernel provides support for arm cores, but the patches and modules for the OMAP architecture must also be installed. Using the “git” clone, checkout and commit functions, the latest kernel was obtained via “git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git”

and development source from reference [10]. The version at the time of this report is 3.2.15-x8. The patches were then collected and merged prior to cross compiling the kernel. An example of the kernel patches for the OMAP3 is shown in Table 3.

Parameters in the make file must be verified to include the architecture for the target of the kernel, ARM[®], and the direction to perform a cross compilation along with a location of the cross compiler tools, in this case the GNU ARM compiler “gcc-arm-linux-gnueabi.” The kernel configuration is specified up front and can be edited one of several ways. The “menuconfig” option provides a graphical interface to set the compiler options and is useful when trying to avoid errors from mistakes made when typing on the keyboard. The default selections are adequate with the exception of kernel debugging. This is normally not selected, but needed for this project. It adds some time to the compilation process, but produces a “vmlinux” file in addition to the compressed kernel file “zImage.” The vmlinux file contains the linux kernel and the kernel symbol table. The symbol table is required on the host computer to debug the target computer.

Once the kernel is compiled, the zImage file from “/KERNEL/arch/arm/boot” is copied to the boot partition of the SD card. Getting all the right files to the correct locations on each respective partition for the target board SD card is important. The MLO file must be the first file copied to the boot partition, otherwise the target will not boot.

The next step is removing the SD card from the host and placing into the target computer. As already noted, the FlySwatter interface has a second port for the serial connection on the BeagleBoard-xM. Using Minicom on the host computer with the serial port set to 115200, 8N1 allows all the boot messages to be seen from the target computer. For the project it was beneficial to use a serial-to-usb converter and a Microsoft Windows based machine running Tera Term VT. This reduced the number of open windows on the host computer and allowed better access to the host computer functions.

#	Description	Module Name	Owner
1	omap: serial: fix non-empty uart fifo read abort	Platform	Vikram Pandita
2	OMAP3 : Enable TWL4030 Keypad for Zoom2 and Zoom3 boards	Gaia	Manjunath GK
3	Zoom2/3:Update hsmmc board config params	HSMMC	Madhu
4	omap3: zoom2/3: make MMC slot work again	HSMMC	Anand G
5	Correcting GPMC_CONFIG1_DEVICETYPE_NAND	NAND	Vimal
6	Add NAND Lock/Unlock feature	NAND	Vimal Singh
7	OMAP: ZOOM2: Correcting key mapping for few keys	Keypad	Vimal
8	omap3: pm: Add T2 Keypad as a wakeup source	Keypad	Teerth
9	omap: serial: fix coding style indentation	Platform	Vikram Pandita
10	omap: zoom3: enable ehci support	Platform	Vikram Pandita
11	OMAP3 : Fix I2C lockup during timeout/error cases	I2C	Manjunath
12	ARM: OMAP3: PM: T2 keypad wakeup for Zoom2	Power	Lesly A M
13	OMAP3: add support for 192Mhz DPLL4M2 output	Platform	Vishwa
14	OMAP3: introduce DPLL4 Jtype	Platform	Vishwa
15	OMAP3: Correct width for CLKSEL Fields	Platform	Vishwa
16	OMAP3: Introduce 3630 DPLL4 HSDivider changes	Platform	Mike T
17	OMAP3630: Clock: Workaround for DPLL HS divider limitation	Power	Vijay
18	3630 DVFS	Power	Romit
19	Introducing gpmc nand.c for GPMC specific NAND ini	NAND	Vimal
20	OMAP SDP Introducing board sdp flash.c for flash	NAND	Vimal
21	OMAP3: Add support for flash on 3430SDP board	NAND	Vimal
22	Zoom3: Defconfig update	Platform	Manjunath
23	PM debug: Fix warning when no CONFIG_DEBUG_FS	Power	Sergio
24	OMAP2/3 PM: Adding power domain APIs for reading the next logic and mem state	Power	Thara
25	OMAP3 PM: Defining .pwrsts_logic_ret field for core power domain structurePower	Power	Thara
26	OMAP: HWMOD: Add support for early device register into omap device layer	Power	Thara
27	FIX OMAP3:McBSP poll read and write for OMAP3	McBSP	Rafiuddin Syed

Table 3, OMAP Kernel Patches [11]

The output from the serial target computer serial is shown in Illustration 6 below. The serial port, board name, and the uEnv.txt file can be seen with the “Loaded environment from uEnv.txt” message that follows. The zImage file is then read and the message “Starting kernel...” appears. The rest of the messages are associated with booting the kernel and show up on the serial port just as they would on a connected computer monitor for this, or any other Linux machine.

```
COM7:115200baud - Tera Term VT
File Edit Setup Control Window Help
U-Boot SPL 2012.04-rc1-dirty (Apr 14 2012 - 14:57:19)
Texas Instruments Revision detection unimplemented
OMAP SD/MMC: 0
timed out in wait_for_bb: I2C_STAT=1000
reading u-boot.img
reading u-boot.img

U-Boot 2012.04-rc1-dirty (Apr 14 2012 - 14:57:19)

OMAP3630/3730-GP ES1.1, CPU-OPP2, L3-165MHz, Max CPU Clock 1 Ghz
OMAP3 Beagle board + LPDDR/NAND
I2C: ready
DRAM: 512 MiB
NAND: 0 MiB
MMC: OMAP SD/MMC: 0
*** Warning - readenv() failed, using default environment

In: serial
Out: serial
Err: serial
Beagle xM Rev A
No EEPROM on expansion board
No EEPROM on expansion board
Die ID #0c180001fff00000015739eb02a01f
Net: Net Initialization Skipped
No ethernet found.
Hit any key to stop autoboot: 0
The user button is currently NOT pressed.
SD/MMC found on device 0
reading uEnv.txt

1050 bytes read
Loaded environment from uEnv.txt
Importing environment from mmc ...
reading zImage

3002344 bytes read
Booting from mmc ...

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[ 0.000000] Booting Linux on physical CPU 0
[ 0.000000] Initializing cgroup subsys cpuset
[ 0.000000] Initializing cgroup subsys cpu
[ 0.000000] Linux version 3.2.15-x8 (jim@jim-laptop) (gcc versio
tu/Linaro 4.6.2-14ubuntu2~ppa1) > #1 SMP Sun Apr 15 02:10:05 CDT 20
[ 0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7),
[ 0.000000] CPU: PIPT / UIPT nonaliasing data cache, UIPT aliasi
n cache
[ 0.000000] Machine: OMAP3 Beagle Board
[ 0.000000] Beagle expansionboard: none
[ 0.000000] Beagle second expansionboard: none
[ 0.000000] Reserving 12582912 bytes SDRAM for URAM
[ 0.000000] Memory policy: ECC disabled, Data cache writeback
[ 0.000000] OMAP3630 ES1.1 (l2cache iva sgx neon isp 192mhz_clk
[ 0.000000] Clocking rate (Crystal/Core/MPU): 26.0/400/600 MHz
[ 0.000000] PERCPU: Embedded 8 pages/cpu @c0d02000 s10784 r8192
```

Illustration 6 Port, Target Board Boot Screen at Serial Port

The DVI port is enabled on the board and the display settings were adjusted to accommodate an HD monitor that was available for the project. The kernel boots successfully and commands are able to be entered via a connected keyboard or the serial interface. The “uname -a” and “lsb_release -a” commands were run to demonstrate the results and are shown in the terminal screen in Illustration 7.

```
l 10.338745] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
l 10.549652] EXT4-fs (mmcblk0p2): re-mounted. Opts: errors=remount-r
o
l 13.736083] smsc95xx 1-2.1:1.0: eth0: link up, 100Mbps, full-duplex
l lpa 0x45E1
l 25.399719] sshd (872): /proc/872/oom_adj is deprecated, please use
/proc/872/oom_score_adj instead.
Debian GNU/Linux 6.0 devel tty02
devel login: root
Password:
Last login: Tue Apr 17 16:24:45 CDT 2012 on tty1
Linux devel 3.2.15-x8 #1 SMP Sun Apr 15 02:10:05 CDT 2012 armv7l
The programs included with the Debian GNU/Linux system are free softwa
re;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@devel:~# uname -a
Linux devel 3.2.15-x8 #1 SMP Sun Apr 15 02:10:05 CDT 2012 armv7l GNU/L
inux
root@devel:~# lsb_release -a
No LSB modules are available.
Distributor ID: Debian
Description: Debian GNU/Linux 6.0.4 (squeeze)
Release: 6.0.4
Codename: squeeze
root@devel:~# █
```

Illustration 7, Terminal Screen at Debian Prompt

A few “tweaks” needed to be made once the kernel was running. In order to get an Ethernet connection working, modifications were made to the “/etc/network/interfaces” file. “auto eth0” and “eth0 inet dhcp” appended to the file enabled the internet connection.

OPENOCD

OpenOCD³ is an abbreviation for open on-chip debugger. It provides the capability to program, debug, and boundary scan test remote target embedded platforms [7]. OpenOCD communicates with the target via a hardware adapter such as the FlySwatter used here. Compilation and installation is on the host computer and the drivers for the debug adapter must be included during compilation. The FTDI drivers mentioned previously support this requirement for the FlySwatter and provide the interface to the JTAG connector on the target board.

The latest version of the OpenOCD software at the time of this writing is 0.5.0 and is the version used for this project. In order to get the latest version the source was

³ The OpenOCD software is covered under the GNU General Public License and is available from the Sourceforge website: <http://sourceforge.net/projects/openocd/files/openocd/>

downloaded and built on the host computer “git clone git://openocd.git.sourceforge.net/gitroot/openocd/openocd.” Some additional drivers and tools were required for the OpenOCD configuration.

1. pkg-config
 - a. Obtained from “pkgconfig.freedesktop.org/releases/” [13]
 - b. This is tool allows compiler options to be entered on the command line thus preventing them from being hard coded. It is helpful since there are a multitude of options available and updates occur often.
2. libusb library
 - a. Obtained using “apt-get install libusb-dev”
 - b. This is an open source C library needed for the FTDI open source driver
3. libftdi
 - a. Obtained using “apt-get install libftdi-dev”
 - b. The open source FTDI FT232 driver
4. libtool
 - a. Needed for compiling OpenOCD since it used in the open source provided scripts
 - b. Provides an interface for using shared libraries for consistency
5. Texinfo
 - a. Installed using “apt-get install texinfo”
 - b. Provides format used for the documentation for OpenOCD

In order to install OpenOCD the following commands were run to get the final install as shown in Table 4:

#	Command	Description
1	<code>./bootstrap</code>	configure the autoconf
2	<code>./configure --enable-maintainer-mode --enable-ft2232_libftdi</code>	to configure the ftdi driver (the open source version)
3	<code>Make</code>	Compile OpenOCD
4	<code>Make install</code>	Installation in applicable directories

Table 4, OpenOCD Installation

To run the software two configuration files are needed; one for the FlySwatter and one for the BeagleBoard-xM. Though the -xM is OMAP compliant, the processor architecture is different, so it will need to reference a different target chip type. The “ti_beagleboard_xm.cfg” file references the configuration file for the dm37x, “amdm37x.cfg.” This configuration file contains the specific information for the -xM chip set as well as information for the am35x. The TAPs for JTAG are set up in this file and follow the convention specified in the DM37x technical reference manual [2]. The TAPs must be added to the scan chain in order such that the TAP closest to TDO comes first. The dm37x processor SRAM begins at address 0x4020 0000. The configuration file reserves the first 16K starting at that address for use by the OpenOCD software.

OpenOCD processes the configuration files input on the command line when it starts: “openocd -f interface/flyswatter.cfg -f ti_beagleboard_xm.cfg.”

The JTAG setup is accomplished via the target configuration file “amdm37x.cfg.” The ICEPick module is referenced as “icepick.cfg” and will be the last in the JTAG chain since it is closest to TDI. The ICEPick configuration file selects the JTAG router and sets up control of the data and instruction register scans in the configuration file.

The target configuration file finds the chip type (dm37x) and establishes the expected IDCODE for the JRC. In this case it is 0x1b89102f. This is validated by the “scan_chain” command in OpenOCD. As shown in Illustration 8, dm37x.jrc is enabled and the IDCODE matches the expected IDCODE. The adapter frequency is set and then the chain is set up. OpenOCD requires all devices to be declared using the “jtag newtap” command and, as mentioned earlier, they must be declared in order.

```

Open On-Chip Debugger
> scan_chain
  TapName          Enabled  IdCode      Expected    IrLen IrCap  IrMask
-----
0 dm37x.dap        Y        0x00000000  0x00000000   4 0x01  0x0f
1 dm37x.arm2       n        0x00000000  0x00000000   4 0x01  0x0f
2 dm37x.dsp        n        0x00000000  0x00000000  38 0x25  0x3f
3 dm37x.d2d        n        0x00000000  0x00000000   4 0x01  0x0f
4 dm37x.jrc        Y        0x1b89102f  0x1b89102f   6 0x01  0x3f
                   0x0b89102f
>

```

Illustration 8, Scan Chain Results

The code snippet below illustrates the TAP declaration on line 1 followed by the last declaration shown as line 2 for the ICEPick.

```

1 jtag newtap dm37x arm2 -irlen 4 -ircapture 0x1 -irmask 0x0f -disable
2 jtag newtap dm37x jrc -irlen 6 -ircapture 0x1 -irmask 0x3f 0x1b89102f

```

The irlen number represents the length of the instruction register in bits. The disable parameter is used to flag a TAP that is not part of the scan chain after a reset via TRST or by entering the RESET state on the state machine. The ircapture is the bit pattern loaded into the JTAG SR. This is for entering the capture IR state as shown in Figure 4. Per the JTAG specification the two least significant bits of this value should be one. The irmask value is used with the ircapture to as a check to verify the scans are

working. If the TAPs are not configured, OpenOCD has an auto TAP discovery feature. This was attempted by the author without success.

Once the description is completed in the target configuration file, the TAPs are enabled. Several TCK cycles are then sent to ensure the things are running followed by a “tapenable” command. The next section establishes the work area for OpenOCD as discussed earlier at address 0x420 0000 with 16K reserved for the program.

```
1 $_TARGETNAME configure -work-area-phys 0x40200000 -work-area-size
   0x4000
```

The JTAG clock is then slowed down to ensure that it will function with slowest processor core clock and then a software restart is completed by writing a 0b10 to address 0x4830 7250 and is shown in table 3-452 of reference [7]. The target is then reinitialized with the “amdm37x_dbginit” function and the adapter speed is set to 100 kHz. “interface.c” contains the information and instructions for specifying JTAG state transitions (see Figure 4). A case structure is provided for each state, Reset, Idle, DRSHIFT, DRPAUSE, IRSHIFT, and IRPAUSE.

The “cortex_a.c” file sets up target polling, breakpoints, read and writes, the processor mmu, and the virtual to physical memory address structure. Virtual address space is separated between user and kernel space with addresses from 0x0000 0000 to 0xbfff ffff as user space and from 0xc000 0000 to 0xffff ffff for supervisor address mode.

The ADIV5 interface mentioned previously is implemented in “arm_adi_v5.c.” The DAP, comprised of the DP and the AP is setup with the JTAG-DP and the MEM-AP for accessing memory registers.

Once OPenOCD has started a separate terminal window is opened in order to connect to the OpenOCD session via “telnet localhost 4444.” From the prompt the “scan_chain,” “dap info,” register information and other commands can be issued.

GNU DEBUGGER AND INSIGHT

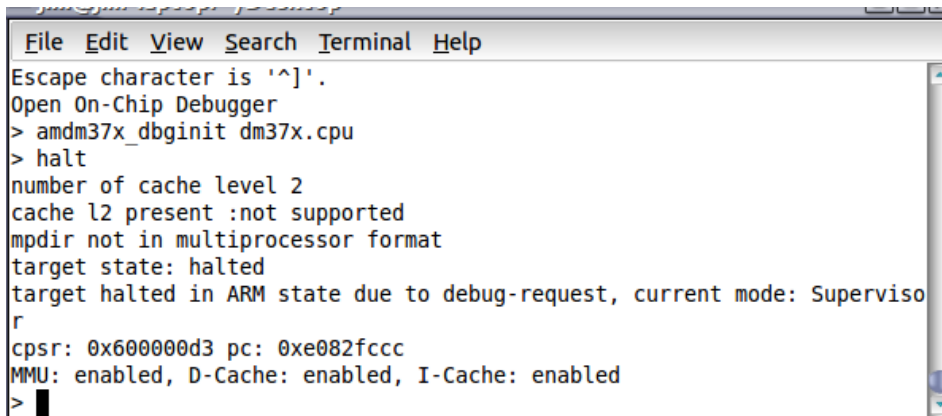
OpenOCD provides the capability to use a GDB server to monitor and control program execution. Insight [16] is a graphical environment integrated with GDB and is released under the terms of the GNU GPL. Insight has a program start window where the source code can be viewed, and watch, stack, register, memory browser and GDB terminal windows. It provides the ability to have multiple windows open at the same time, which is huge advantage when comparing registers, memory and program instructions. Though other alternatives are available this choice provided the most hassle free option for the debug environment and required only a small amount of work to set everything up.

The Insight software was downloaded and compiled for use with ARM[®] architecture. Once OpenOCD is running Insight is started with “arm-linux-gnueabi-insight” and then a connection made to the target; “target remote localhost :3333.” This can also be done with the “Run” menu in the source window.

With all the hardware connections made, the host and target computers running, and the connection established via OpenOCD, initialization is accomplished with the “amdm37x_dbginit dm37x.cpu” command and quick check of the scan chain and registers verifies everything is working properly. Insight is then started and connected to the target. From Insight the symbol information from the vmlinux file is loaded and the registers and memory can be viewed. The assembly code with the symbol information appears in the source window.

The target must be halted to obtain register and memory values. When the target is running, commands can be processed via the serial connection to the BeagleBoard or a keyboard if connected. After the processor is halted via the JTAG connection, a quick check of the target demonstrates that commands are no longer accepted. The “monitor

resume” command restores the running target machine again meaning that commands are passed through the JTAG chain to the processor as expected. Cross checking the register values between the OpenOCD window and the Insight register window also validates that the correct values are being passed to the debugger. The initial insight window shows the beginning of the Linux kernel in “head.s” (the start of the kernel). “ENTRY(stext)” forces the SVC processor mode. The lookup for the processor and architecture type are performed prior to the “__create_page_tables” function. The MMU is then setup and enabled. The OpenOCD connection window shows the MMU as “enabled” if connected to the target with a running kernel as shown in Illustration 9.



```
File Edit View Search Terminal Help
Escape character is '^]'.
Open On-Chip Debugger
> amdm37x_dbginit dm37x.cpu
> halt
number of cache level 2
cache l2 present :not supported
mpdir not in multiprocessor format
target state: halted
target halted in ARM state due to debug-request, current mode: Supervisor
r
cpsr: 0x600000d3 pc: 0xe082fccc
MMU: enabled, D-Cache: enabled, I-Cache: enabled
> █
```

Illustration 9, OpenOCD "halt" Window

The “__mmap_switched,” which holds the address of the “start_kernel” function is then run (see Illustration 10 and Illustration 11 below).

```

main.c | start_kernel | SRC+ASM
285 #ifndef CONFIG_DEBUG_PAGEALLOC
286 int __read_mostly debug_pagealloc_enabled = 0;
287 #endif
288
289 static int __init init_setup(char *str)
290 {
291     unsigned int i;
292
293     execute_command = str;
294     /*
295      * In case LILO is going to boot us with default command line,
296      * it prepends "auto" before the whole cmdline which makes
297      * the shell think it should execute a script with such name.
298      * So we ignore all arguments entered before _init=... [MJ]
299      */
300     for (i = 1; i < MAX_INIT_ARGS; i++)
301         argv_init[i] = NULL;
302     return 1;
303 }
304 __setup("init=", init_setup);
305
306 static int __init rdinit_setup(char *str)
307 {
308
309 }
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Illustration 10, "start_kernel" in "main.c"

```

head.S | stext | SRC+ASM
82
83 THUMB( adr    r9, BSYM(1f) ) @ Kernel is always entered in ARM.
84 THUMB( bx    r9            ) @ If this is a Thumb-2 kernel,
85 THUMB( .thumb                ) @ switch to Thumb now.
86 THUMB(1:
87
88     setmode PSR_F_BIT | PSR_I_BIT | SVC_MODE, r9 @ ensure svc mode
89                                     @ and irq's disabled
90     mrc    p15, 0, r9, c0, c0 @ get processor id
91     bl    __lookup_processor_type @ r5=procinfo r9=cpuid
92     movs  r10, r5 @ invalid processor (r5=0)?
93 THUMB( it eq ) @ force fixup-able long branch encoding
94     beq  __error_p @ yes, error 'p'
95
96 #ifndef CONFIG_XIP_KERNEL
97     adr  r3, 2f
98     ldmia r3, {r4, r8}
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Illustration 11, Insight "head.s" Window

Register and memory values update and are viewed in the debug window (Illustration 12) and breakpoints can be set, however stepping through the kernel is still problematic. The connection often drops or loses communication across the JTAG connection without warning. Persistence is required and entering commands via the OpenOCD window or the Insight console window is needed. Illustration 12 shows the result of one session where the kernel ran and was halted in the kernel timekeeping function.

Being able to see the real time updates of the registers and memory save time and make the debugging task easier. The GUI allows all the locations of interest to be quickly and simultaneously displayed so that their status is immediately understood. This adds efficiency and reduces the time that would otherwise be required to issue separate commands and scroll through results to see how the information is changing.

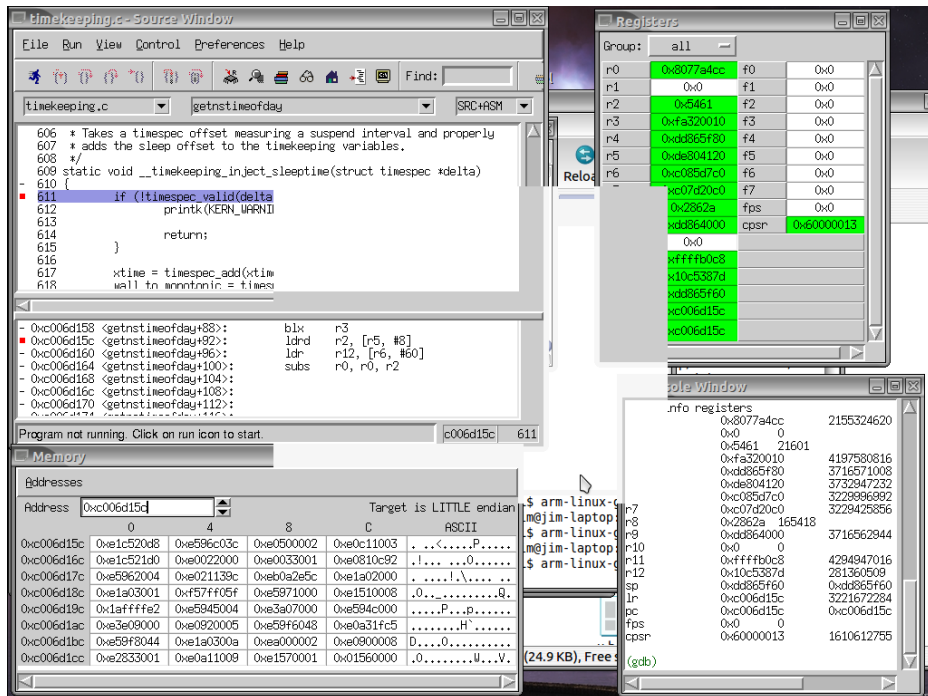
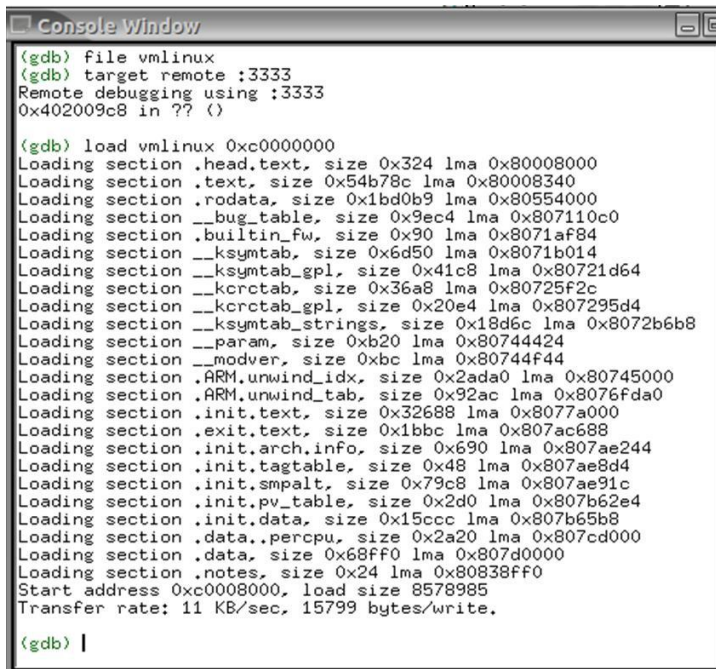


Illustration 12, Insight Debug Window

Chapter 3: Problems and Forward Work

There are still a few issues that should eventually be addressed. There are a significant number of programs, hardware, and parameters that work together. The current setup is difficult in practice and sometimes displays erratic behavior. The connection to the target computer will drop out and the symbol information will sometimes show up out of step with the target machine.

Small program files are able to be directly loaded to the target machine. When an attempt was made to load the Linux kernel remotely, the “head.text” and “.text” sections loaded successfully. Other sections would fail. This appears to be due to a memory address failure and further research provided an interim solution. By resetting and initializing the CPU, the vmlinux file would load with an offset specified as demonstrated in Illustration 13.



```
(gdb) file vmlinux
(gdb) target remote :3333
Remote debugging using :3333
0x402009c8 in ?? ()

(gdb) load vmlinux 0xc0000000
Loading section .head.text, size 0x324 lma 0x80008000
Loading section .text, size 0x54b78c lma 0x80008340
Loading section .rodata, size 0x1bd0b9 lma 0x80554000
Loading section __bug_table, size 0x9ec4 lma 0x807110c0
Loading section .builtin_fw, size 0x90 lma 0x8071af84
Loading section __ksymtab, size 0x6d50 lma 0x8071b014
Loading section __ksymtab_gpl, size 0x41c8 lma 0x80721d64
Loading section __kcrctab, size 0x36a8 lma 0x80725f2c
Loading section __kcrctab_gpl, size 0x20e4 lma 0x807295d4
Loading section __ksymtab_strings, size 0x18d6c lma 0x8072b6b8
Loading section __param, size 0xb20 lma 0x80744424
Loading section __modver, size 0xbc lma 0x80744f44
Loading section .ARM.unwind_idx, size 0x2ada0 lma 0x80745000
Loading section .ARM.unwind_tab, size 0x92ac lma 0x8076fda0
Loading section .init.text, size 0x32688 lma 0x8077a000
Loading section .exit.text, size 0x1bbc lma 0x807ac688
Loading section .init.arch.info, size 0x690 lma 0x807ae244
Loading section .init.tagtable, size 0x48 lma 0x807ae8d4
Loading section .init.smpalt, size 0x79c8 lma 0x807ae91c
Loading section .init.pv_table, size 0x2d0 lma 0x807b62e4
Loading section .init.data, size 0x15ccc lma 0x807b65b8
Loading section .data.percpu, size 0x2a20 lma 0x807cd000
Loading section .data, size 0x68ff0 lma 0x807d0000
Loading section .notes, size 0x24 lma 0x80838ff0
Start address 0xc0008000, load size 8578985
Transfer rate: 11 KB/sec, 15799 bytes/write.

(gdb) |
```

Illustration 13, Loading the Kernel

This method requires further work to set up a boot loader and initialization file that will allow a remote load of the kernel to the correct address in RAM and run it. Lastly, a script file with a means to acknowledge timing and connection successes would simplify and shorten the process required to get to the debugging screen in Insight with one command or click. Eclipse may work for this task as well, but would require a potentially large amount of setup and configuration.

Chapter 4: *Conclusion*

This project demonstrated one hardware and software combination for a toolchain to debug the Linux kernel on an ARM[®] Cortex-A8 processor via the JTAG connection. Components and their connections as well as operation of the software to support this project were explained.

Appendix

BeagleBoard-xM Features

Feature		
Processor	Texas Instruments Cortex A8 1GHz processor	
POP Memory	Micron 4Gb MDDR SDRAM (512MB) 200MHz	
PMIC TPS65950	Power Regulators	
	Audio CODEC	
	Reset	
	USB OTG PHY	
Debug Support	14-pin JTAG	GPIO Pins
	UART	3 LEDs
PCB	3.1" x 3.0" (78.74 x 76.2mm)	6 layers
Indicators	Power, Power Error	2-User Controllable
	PMU	USB Power
HS USB 2.0 OTG Port	Mini AB USB connector	
	TPS65950 I/F	
USB Host Ports	SMSC LAN9514 Ethernet HUB	
	4 FS/LS/HS	Up to 500ma per Port if adequate power is supplied
Ethernet	10/100	From USB HUB
Audio Connectors	3.5mm	3.5mm
	L+R out	L+R Stereo In
SD/MMC Connector	MicroSD	
User Interface	1-User defined button	Reset Button
Video	DVI-D	S-Video
Camera	Connector	Supports Leopard Imaging Module
Power Connector	USB Power	DC Power
Overvoltage Protection	Shutdown @ Over voltage	
Main Expansion Connector	Power (5V & 1.8V)	UART
	McBSP	McSPI
	I2C	GPIO
	MMC2	PWM
2 LCD Connectors	Access to all of the LCD control signals plus I2C	3.3V, 5V, 1.8V
Auxiliary Audio	4 pin connector	McBSP2
Auxiliary Expansion	MMC3	MMC3,GPIO,ADC,HDQ

References

- [1] beagleboard.org, "BeagleBoard-xM Rev C System Reference Manual," 2010.
- [2] ARM Limited, Cortex A-8 Technical Reference Manual, Revision: r3p0, 2006-2008, p. 758.
- [3] ARM Limited, ARM Debug Interface v5 Architecture Specification, 2006.
- [4] ARM Limited, CoreSight Architecture Specification, v1.0, 2004-2005.
- [5] Wikipedia, "Application Binary Interface," Wikimedia Foundation, Inc., 11 April 2012. [Online]. Available: http://en.wikipedia.org/wiki/Application_binary_interface. [Accessed April 2012].
- [6] J. Masters, "Prorting Linux," WordPress Entries, 5 June 2011. [Online]. Available: <http://www.jonmasters.org/blog/category/general/linux-kernel/>. [Accessed March 2012].
- [7] Texas Instruments, AM/DM37x Multimedia Device Technical Reference Manual, Version O, 2012.
- [8] Tin Can Tools, "BeagleBoard Adapter Kit," 2012. [Online]. Available: <http://www.tincantools.com/product.php?productid=16144&cat=0&page=1&featured>. [Accessed 2012].
- [9] ARM Limited, CoreSight(TM) DAP-Lite Technical Reference Manual, ARM Limited, 2006-2008.
- [10] T. Bauermann, "Toolchain Working Group," WorkingGroups/ToolChain, 9 April 2012. [Online]. [Accessed April 2012].
- [11] DENX Software Engineering, "U-Boot Source Code," DetlevZundel, 19 October 2011. [Online]. Available: <http://www.denx.de/wiki/U-Boot/SourceCode>. [Accessed March 2012].
- [12] R. Nelson, "BeagleBoard wiki," 2012. [Online]. Available: <http://eewiki.net/display/linuxonarm/BeagleBoard#>. [Accessed March 2012].
- [13] Media Wiki, "OMAppedia Patches Accepted," Creative Commons Attribution-Share Alike 3.0 license., 25 May 2010. [Online]. Available: http://omappedia.org/wiki/Patches_Accepted. [Accessed 2011 March].
- [14] S. Oliver, O. Harboe, D. Ellis and D. Brownwell, Open On-Chip Debugger: OpenOCD User's Guide, Boston: Free Software Foundation, 2011.
- [15] freedesktop.org, "Index of / releases," 28 May 2010. [Online]. Available: <http://pkgconfig.freedesktop.org/releases/>. [Accessed March 2012].
- [16] K. Seitz, J. Ingham, I. Taylor, T. Tromej and E. Zannoni, "The GDB GUI," 19 July 2009. [Online]. Available: <http://sources.redhat.com/insight/index.php>. [Accessed Februray 2012].

Vita

Warren Clay Grant has a Bachelor of Science in Information Technology from the University of Phoenix and an MBA from Regis University. He is a graduate of the Navy Nuclear Power School, Navy Nuclear Prototype training, and a qualified Submariner. He worked in the Department of Energy's nuclear weapons complex at various sites across the country before coming to work at the Johnson Space Center for NASA in Houston, Texas.

Permanent email: wcgrant2@gmail.com

This report was typed by Warren Clay Grant.