



AMX™ 4-Thumb Tool Guide

First Printing: August 1, 1997

Last Printing: March 1, 2005

Copyright © 1997 - 2005

KADAK Products Ltd.

206 - 1847 West Broadway Avenue

Vancouver, BC, Canada, V6J 1Y5

Phone: (604) 734-2796

Fax: (604) 734-8114

TECHNICAL SUPPORT

KADAK Products Ltd. is committed to technical support for its software products. Our programs are designed to be easily incorporated in your systems and every effort has been made to eliminate errors.

Engineering Change Notices (ECNs) are provided periodically to repair faults or to improve performance. You will automatically receive these updates during the product's initial support period. For technical support beyond the initial period, you must purchase a Technical Support Subscription. Contact KADAK for details. Please keep us informed of the primary user in your company to whom update notices and other pertinent information should be directed.

Should you require direct technical assistance in your use of this KADAK software product, engineering support is available by telephone, fax or e-mail. KADAK reserves the right to charge for technical support services which it deems to be beyond the normal scope of technical support.

We would be pleased to receive your comments and suggestions concerning this product and its documentation. Your feedback helps in the continuing product evolution.

KADAK Products Ltd.
206 - 1847 West Broadway Avenue
Vancouver, BC, Canada, V6J 1Y5

Phone: (604) 734-2796
Fax: (604) 734-8114
e-mail: amxtech@kadak.com

**Copyright © 1997-2005 by KADAK Products Ltd.
All rights reserved.**

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of KADAK Products Ltd., Vancouver, B.C., CANADA.

DISCLAIMER

KADAK Products Ltd. makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability and fitness for any particular purpose. Further, KADAK Products Ltd. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of KADAK Products Ltd. to notify any person of such revision or changes.

TRADEMARKS

AMX in the stylized form and KwikNet are registered trademarks of KADAK Products Ltd. AMX, AMX/FS, InSight, *KwikLook* and *KwikPeg* are trademarks of KADAK Products Ltd. Microsoft, MS-DOS and Windows are registered trademarks of Microsoft Corporation. ARM is a trademark of Advanced RISC Machines Ltd. All other trademarked names are the property of their respective owners.

AMX 4-Thumb TOOL GUIDE
Table of Contents

	Page
1. Selecting a Tool Set	1-1
2. ARM Ltd. (RM) Tool Guide (for SDT, ADS)	2-1
3. MetaWare (MW) Tool Guide	3-1
4. ARM Ltd. (RV) Tool Guide (for RVDS)	4-1
5. Metrowerks (ME) Tool Guide	5-1

This page left blank intentionally.

1. Selecting a Tool Set

Available Toolsets

AMX™ 4-Thumb and the *KwikLook*™ Fault Finder have been developed on a PC with Microsoft® Windows® using the software development tools described in this guide.

To simplify the selection process, KADAK has prepared this Tool Guide. This chapter introduces the tools and defines the subsets which KADAK has used with success. Subsequent chapters provide specific guidelines for using each of the supported toolset combinations with AMX 4-Thumb.

Note that AMX 4-Thumb is delivered to you ready to use with each of the supported toolsets. Should you wish to rebuild the AMX 4-Thumb Library for any reason, follow the construction guidelines provided in Appendix D of the AMX User's Guide.

To construct your embedded application, you will require a C or C++ compiler, an assembler, a librarian (optional), a linker and/or locator and a remote debugger. The vendors listed below provide these tools. The tool name listed is the vendor's product name or the name of the executable program used to run the tool. The tool name listed will be used throughout this manual to reference the specific tool from a particular vendor.

Vendor	C/C++	Assembler	Librarian	Linker	Locator	Debugger
ARM	<i>ARMCC</i> or <i>TCC</i>	<i>ARMASM</i> or <i>TASM</i>	<i>ARMAR</i> or <i>ARMLIB</i>	<i>ARMLINK</i>		<i>AXD</i> or <i>ADW</i> or <i>RealView</i>
MetaWare	<i>HCARM</i>	<i>ASARM</i>	<i>ARARM</i>	<i>LDARM</i>		<i>SeeCode</i>
Metrowerks	<i>MWCCARM</i>	<i>MWASMARM</i>	<i>MWLDARM</i>	<i>MWLDARM</i>		<i>CodeWarrior</i>

Supported Toolsets

Unfortunately you cannot arbitrarily use any combination of the listed tools. Of all the tools listed, KADAK has identified several combinations which can be used with AMX 4-Thumb. The supported toolsets are divided into major classes according to the C/C++ compiler vendor and then, if necessary, into sub-classes, one for each locator and/or debugger.

Each supported toolset is given a three character mnemonic called a **toolset id** which is used by KADAK to identify the toolset combination. The first two characters of the mnemonic identify the compiler vendor. The third character, if needed, identifies the locator and/or debugger used.

Compiler

<i>RM</i>	Advanced RISC Machines Ltd. (for SDT, ADS tools)
<i>RV</i>	Advanced RISC Machines Ltd. (for RVDS tools)
<i>MW</i>	MetaWare Incorporated High C/C++
<i>ME</i>	Metrowerks Inc. C/C++

Debugger

--	Advanced RISC Machines Ltd. <i>AXD</i> ARM Extended Debugger
--	Advanced RISC Machines Ltd. <i>ADW</i> ARM Debugger for Windows
--	Advanced RISC Machines Ltd. RealView Debugger
--	MetaWare Incorporated SeeCode Debugger for ARM
--	Metrowerks CodeWarrior for ARM

The following toolset combinations are supported by KADAK.

Toolset id:	RM	RV	MW	ME
Vendor:	ARM (SDT, ADS)	ARM (RVDS)	MetaWare	Metrowerks
C/C++	<i>TCC</i>	<i>ARMCC</i>	<i>HCARM</i>	<i>MWCCARM</i>
Assembler	<i>ARMASM,</i> <i>TASM</i>	<i>ARMASM</i>	<i>ASARM</i>	<i>MWASMARM</i>
Librarian	<i>ARMAR</i> OR <i>ARMLIB</i>	<i>ARMAR</i>	<i>ARARM</i>	<i>MWLDARM</i>
Linker/ Locator	<i>ARMLINK</i>	<i>ARMLINK</i>	<i>LDARM</i>	<i>MWLDARM</i>
Debugger	<i>AXD</i> or <i>ADW</i> or <i>RVDEBUG</i>	<i>RVDEBUG</i>	<i>SeeCode</i>	<i>CodeWarrior</i>

2. ARM Ltd. (RM) Tool Guide (for SDT, ADS)

AMX™ 4-Thumb has been developed on a PC with Windows® NT v4.0 using the ARM Software Development Kit from Advanced RISC Machines (ARM). These tools are listed below. The AMX libraries and object modules on the product disks have been generated using the most recent tools listed. If you are not using this toolset, you may have to rebuild the AMX libraries in order to use your out-of-date tools.

ARM Tools		<u>v2.11</u> (SDT)	<u>v2.50</u> (SDT)	<u>v1.0</u> (ADS)	<u>v1.1</u> (ADS)	<u>v1.2</u> (ADS)
<i>TCC</i>	Thumb C/C++ compiler	4.71	4.90	1.0	1.1	1.2
<i>ARMASM</i>	Assembler	2.37	2.50	1.0	1.1	1.2
<i>ARMLINK</i>	Linker/locator	5.07	5.20	1.0	1.1	1.2
<i>ARMAR</i>	Librarian			1.0	1.1	1.2
<i>ARMLIB</i>	Librarian	4.38	4.50			
<i>ADW</i>	ARM Debugger for Windows	2.11	2.50	1.0	1.1	n/a
<i>AXD</i>	ARM Extended Debugger			1.0	1.1	1.2
<i>RVDEBUG</i>	ARM RealView Debugger					1.6

Note that the Thumb assembler *TASM* is NOT used to assemble any AMX 4-Thumb modules or your application Target Configuration Module. You will only need to use it to assemble your own assembly language modules which use Thumb instructions.

AMX 4-Thumb and *KwikLook* have been tested on the following platforms.

- Advanced RISC Machines ARM Development Board (ARM7TDMI Version)
- Advanced RISC Machines ARM Development Board (ARM940T Version)
- Advanced RISC Machines Integrator/AP Board (with ARM966E)
- Advanced RISC Machines Integrator/CP Board (with ARM920T)
- Intel XScale IQ80321 Evaluation Platform
- Atmel AT91EB40 Evaluation Board
- Atmel AT91EB42 Evaluation Board
- Atmel AT91SAM7S-EK Evaluation Kit
- Advanced RISC Machines ARM Evaluator-7T (with Samsung S3C4510 processor)
- Cogent Computer Systems, Inc. CSB238 Single Board Computer
(with Cirrus Logic EP7312 processor)
- Freescale i.MX21ADS board

Environment Variables

Set the following environment variables to provide access to all AMX and ARM tools, header files, object files and libraries.

<i>CJPATH</i>	Path to AMX installation directory (. . . \AMX422)
<i>PATH</i>	Path to AMX and ARM executable programs
<i>ARMCONF</i>	Path to ARM configuration files
<i>ARMDLL</i>	Path to ARM DLL support files
<i>ARMINC</i>	Path to ARM C include files
<i>ARMLIB</i>	Path to ARM C library files
<i>TMP</i>	Path to a temporary directory for use by ARM tools

AMX Libraries

The AMX libraries have been constructed using the following assumptions. The resulting AMX 4-Thumb libraries are ready for use with any ARM processor which is compatible with the ARM v4T or v5T architecture.

- AMX 4-Thumb target processor is little endian and of ARM v4T architecture
- AMX 4-Thumb is generated in ELF object format
- AMX 4-Thumb assumes software floating point emulation

Object Formats

ARM supports the Executable and Linking Format (ELF). The AMX 4-Thumb libraries and object modules are provided in ELF format. Your object modules and the AMX and ARM libraries and object modules, all in ELF format, can be combined to create an executable module in ARM Executable Format (AXF) suitable for use with the *AXD* ARM Extended Debugger or the ARM RealView Debugger.

ARM v2.11 only supports the ARM Object Format (AOF). If you are using ARM v2.11 tools, you must rebuild the AMX 4-Thumb libraries and object modules using those tools as described in Appendix D of the AMX User's Guide. Your object modules and the AMX and ARM libraries and object modules, all in ARM format, can be combined to create an executable module in ARM Image Format (AIF) suitable for use with the *ADW* ARM Debugger for Windows.

Parameter Passing Conventions

The ARM tools support the C function parameter passing convention defined in the ARM-Thumb Procedure Call Standard (ATPCS). AMX 4-Thumb follows this standard, the parameter passing convention common to all Thumb toolsets supported by KADAK.

Register Usage

AMX 4-Thumb makes the following C interface register assumptions. Registers *a1*, *a2*, *a3*, *a4*, *ip* and *lr* and the flags in *CPSR* can always be altered by C procedures. All other registers are preserved by AMX according to the ATPCS rules for C procedures. Integers and pointers are returned from C procedures in register *a1*. You must NOT use any C compilation switch which changes these register assumptions.

Big or Little Endian

AMX 4-Thumb is delivered ready for use with the little endian model. AMX 4-Thumb will also operate, without modification, on big endian hardware. However, to use AMX on big endian hardware, you must first **rebuild the AMX Library for big endian operation**. Be sure to use the ARM Ltd. big endian C startup code and libraries.

To rebuild the AMX Library for big endian operation, set environment variable *AMX_ENDN=B*. If you must rebuild the AMX Library for little endian operation, set environment variable *AMX_ENDN=L* or leave it undefined. Then build the library as described in Appendix D of the AMX User's Guide.

Using the ARM Ltd. C Compiler

All AMX header files *CJ422xxx.H* and the generic AMX include file *CJZZZ.H* must be present in the current directory together with your source file being compiled.

Use the following compilation switches when you are compiling modules for use in the AMX environment.

by default	; generate 16-bit, Thumb code
by default	; structure members are aligned per type size
by default	; integers are 4-byte signed quantities
by default	; promotion of char and short int values to integers
	; is done by the function receiving the parameters
<i>-cpu 4T</i>	; ARM v4T Thumb instruction compliance
	; (ADS 1.0 and later)
<i>-c</i>	; compile only
<i>-Otime</i>	; optimize for speed of execution
<i>-apcs /noswst/inter</i>	; no stack checking; use interworking
<i>-li</i>	; compile little endian (<i>-bi</i> for big endian)
<i>-g+</i>	; (optional) generate debug information
<i>-o FILENAME.O</i>	; output object module <i>FILENAME.O</i>
<i>-errors FILENAME.ERR</i>	; redirect C error messages to <i>FILENAME.ERR</i>

For ADS 1.0 and later, the compilation command line is therefore of the following form.

```
TCC -cpu 4T -c -Otime -apcs /noswst/inter -li  
-o FILENAME.O -errors FILENAME.ERR FILENAME.C
```

v2.11 only	
by default	; v2.11 promotes char and short int values to integers
	; prior to passing them as function parameters
<i>-zpq8</i>	; avoid compiler code generation bug

v2.11 and v2.50 only	
<i>-fz</i>	; assume <i>SWI</i> corrupts the link register
<i>-arch 4T</i>	; ARM v4T Thumb instruction compliance

For ARM v2.11 and v2.50, the compilation command line is of the following form. Note that switch *-zpq8* is only required for ARM v2.11 tools. The braces {} must be omitted.

```
TCC {-zpq8} -fz -arch 4T -c -Otime -apcs /noswst/inter -li  
-o FILENAME.O -errors FILENAME.ERR FILENAME.C
```

Compiling the AMX System Configuration Module

Your AMX System Configuration Module *SYSCFG.C* is compiled as follows. All AMX header files *CJ422xxx.H* and the generic AMX include file *CJZZZ.H* must be present in the current directory together with file *SYSCFG.C*.

Use the *-li* switch for little endian systems and *-bi* for big endian systems.

For ARM v1.0 and later tools, use the following command line.

```
TCC -cpu 4T -c -Otime -apcs /noswst/inter -li
-o SYSCFG.O -errors SYSCFG.ERR SYSCFG.C
```

For ARM v2.11 and v2.50 tools, use the following command line. Note that switch *-zpq8* is only required for ARM v2.11 tools. The braces {} must be omitted.

```
TCC {-zpq8} -fz -arch 4T -c -Otime -apcs /noswst/inter -li
-o SYSCFG.O -errors SYSCFG.ERR SYSCFG.C
```

Assembling the AMX Target Configuration Module

Your AMX Target Configuration Module *HDWCFG.S* is assembled as follows. The generic AMX header file *CJZZZK.DEF* must be present in the current directory together with file *HDWCFG.S*.

Use the following command line switches when using the ARM assembler. You MUST use the ARM assembler *ARMASM*, not the Thumb assembler *TASM*. Note that all AMX 4-Thumb assembly language modules generate code areas that support interworking.

	by default	; generate 32-bit ARM code
	by default	; assemble with case sensitivity
	<i>-apcs /noswst/inter</i>	; define ATPCS register names
		; inhibit stack checks; support interworking
	<i>-cpu 4T</i>	; ARM v4T Thumb instruction compliance
	<i>-li</i>	; assemble little endian (<i>-bi</i> for big endian)
	<i>-o HDWCFG.O</i>	; output object module <i>HDWCFG.O</i>
	<i>-e HDWCFG.ERR</i>	; redirect error messages to <i>HDWCFG.ERR</i>
	v2.11 and v2.50 only	
	<i>-arch 4T</i>	; ARM v4T Thumb instruction compliance
	<i>-apcs 3/nosw</i>	; define ATPCS register names (v2.11 only)
		; inhibit stack checks
	<i>-apcs 3/noswst/inter</i>	; define ATPCS register names (v2.50 only)
		; inhibit stack checks; support interworking

For ARM v1.0 and later tools, the assembly command line is of the following form.

```
ARMASM -apcs /noswst/inter -cpu 4T -li
-e HDWCFG.ERR -o HDWCFG.O HDWCFG.S
```

For ARM v2.11 and v2.50 tools, use the following command line. Note that parameter *3/noswst/inter* must be replaced with *3/nosw* if you are using ARM v2.11 tools.

```
ARMASM -apcs 3/noswst/inter -arch 4T -li  
-e HDWCFG.ERR -o HDWCFG.O HDWCFG.S
```

v2.11 Thumb Assembler Only

Because of changes introduced by newer ARM tools, users of the ARM v2.11 Thumb assembler must also add the following definition to the command line used to assemble the AMX 4-Thumb Target Configuration Module.

```
-pd "K_THUMB211 SETA 1"
```

Making Libraries

To make a library from a collection of object modules, create a library specification file *YOURLIB.LBM*. Use the ARM Ltd. version of the AMX library specification file *CJ422.LBM* as a guide.

For ARM v1.0 and later tools, use the following librarian command line switches.

```
-create          ; create a library module
-c              ; create a new library module
-via YOURLIB.LBM ; use library command file YOURLIB.LBM
>YOURLIB.LBE    ; redirect librarian error messages to YOURLIB.LBE
```

Make your library as follows.

```
ARMAR -create -c -via YOURLIB.LBM >YOURLIB.LBE
```

For ARM v2.11 and v2.50 tools, use the following librarian command line switches.

```
-c              ; create a new library module
-o             ; include public symbols in the library
-v YOURLIB.LBM ; use library command file YOURLIB.LBM
>YOURLIB.LBE   ; redirect librarian error messages to YOURLIB.LBE
```

Make your library as follows.

```
ARMLIB -c -o -v YOURLIB.LBM >YOURLIB.LBE
```

Selecting an ARM Library

You must choose a variant of the ARM C Runtime Library that is 16-bit, little endian, supports interworking and has NO runtime stack checking.

Note that the AMX Sample Program specifies a generic ARM library called *ARMLIB.A*. To use the AMX Sample Program link specification file without alteration, you must make a copy of the ARM library file and rename it *ARMLIB.A*.

The AMX Sample Program was linked with the following ARM library files.

```
C_T__UN.L      ; ARM ADS v1.0 (and later) C Runtime Library
ARMLIB_I.16L  ; ARM v2.50 C Runtime Library
ARMLIB_I.16L  ; ARM v2.11 C Runtime Library
```

If you rebuild the AMX libraries for big endian operation, be sure to select the appropriate big endian ARM C Runtime Library for use with your application.

Linking with the ARM Ltd. Linker

When used with the ARM C tools, the modules which form your AMX system must be linked in the following order.

Your *MAIN* module

Other application modules

<i>SYSCFG.O</i>	; AMX System Configuration Module
<i>HDWCFG.O</i>	; AMX Target Configuration Module
<i>CHxxxxxT.O</i>	; AMX chip-specific clock driver or your equivalent
<i>CJ422UF.O</i>	; Launch and leave AMX (may be customized)
<i>CJ422UD.O</i>	; KADAK's ARM Library Semihosting Exclusion Module ; (Resolves ARM semihosting references if not provided ; by the debug monitor. For example, include this module ; when using the RedBoot monitor on the IQ80321 board.)
<i>CJ422RAC.O</i>	; AMX ROM Access Module (customized) ; (only if AMX placed in a separate ROM) ; (see Appendix C in AMX Target Guide)
<i>CJ422CV.A</i>	; AMX 4-Thumb vc Conversion Library ; (only if converting an AMX 86 v3, AMX 386 v1 or ; AMX 68000 v2 application)
<i>CJ422.A</i>	; AMX 4-Thumb Library
<i>ARMLIB.A</i>	; ARM C Runtime Library for target hardware

Create a link specification file *YOURLINK.LKS*. Use the AMX 4-Thumb Sample Program link specification file *CJSAMPLE.LKS* as a guide.

Start with the sample link specification file for the board which most closely resembles your hardware configuration.

Note

If you decide to omit any of the link and locate commands from the sample specification, you may encounter link errors or run-time faults.

Link and locate with the ARM linker using the following command line switches. To reduce the command line length, many of these command line switches should be placed in your link specification file *YOURLINK.LKS*.

```

by default                ; link with case sensitivity
-nodebug                  ; exclude debug records from output
-symbols                  ; generate a symbol listing
-elf                      ; generate ELF format executable file
                          ; suitable for use with the ARM AXD debugger
-errors YOURLINK.LKE    ; send link errors to file YOURLINK.LKE
-o YOURLINK.AXF         ; direct link output to file YOURLINK.AXF
-via YOURLINK.LKS      ; use link specification file YOURLINK.LKS
                          ; which has following command switches:
-info totals              ; generate information on code and data sizes
-map                      ; generate an area map listing
-list YOURLINK.MAP     ; send map listing to file YOURLINK.MAP
-ro-base 0x8000           ; locate read only areas at this address
-rw-base 0x18000         ; locate read/write areas at this address

v2.11 and v2.50 only
-symbols -                ; generate a symbol listing (note the extra -)

v2.11 only
-aif -bin                 ; generate ARM Image Format file suitable for use with
                          ; the ARM ADW debugger (replaces the -elf switch)

```

The link and locate command line is therefore of the following form. If you are using ARM v2.11 or v2.50 tools, be sure to add the extra - after the *-symbols* switch. If you are using ARM v2.11 tools, be sure to replace the *-elf* switch with the *-aif -bin* switches.

```

ARMLINK -nodebug -symbols -elf -errors YOURLINK.LKE
        -o YOURLINK.AXF -via YOURLINK.LKS

```

The resulting load module *YOURLINK.AXF* in ELF format is suitable for use with the *AXD* ARM Extended Debugger or the ARM RealView Debugger. If you are using ARM v2.11 tools, the resulting load module *YOURLINK.AXF* in AIF format is suitable for use with the *ADW* ARM Debugger for Windows. Load modules in other formats ready for burning into EPROM can also be created.

Linking a Separate AMX ROM

AMX can be committed to a separate ROM as described in Appendix C of the AMX Target Guide. Use the AMX Configuration Manager to edit your Target Parameter File *HDWCFG.UP* to define your ROM option parameters. Then use the Manager to generate your ROM Option Module *CJ422ROP.S*, ROM Access Module *CJ422RAC.S* and ROM Option link specification file *CJ422ROP.LKS*.

The ROM Option and ROM Access source modules are assembled as follows. Use the *-li* for little endian systems and *-bi* switch for big endian systems. Note that parameters */noswst/inter* and *-cpu* must be replaced with the proper switches if you are using ARM v2.11 or v2.50 tools (see page 2-4).

```
ARMASM -apcs /noswst/inter -cpu 4T -li -e CJ422ROP.ERR
        -o CJ422ROP.O CJ422ROP.S
```

```
ARMASM -apcs /noswst/inter -cpu 4T -li -e CJ422RAC.ERR
        -o CJ422RAC.O CJ422RAC.S
```

The AMX ROM is linked using link specification file *CJ422ROP.LKS* as follows.

```
ARMLINK -nodebug -errors AMXROM.LKE -o AMXROM.AXF -via CJ422ROP.LKS
```

This example generates file *AMXROM.AXF* in ELF format. The ARM *FROMELF.EXE* utility can be used to convert this ELF file to other formats suitable for transfer to ROM.

When you link your AMX application, be sure to include your customized AMX ROM Access Module *CJ422RAC.O* (created above) in your system link specification file.

Using the AMX Configuration Generator

If you cannot use the AMX Configuration Manager, you may still be able to use the stand-alone AMX Configuration Generator to generate the ROM Option Module *CJ422ROP.S*, ROM Access Module *CJ422RAC.S* and ROM Option link specification file *CJ422ROP.LKS*.

Copy the ROM Option and ROM Access template files *CJ422ROP.CT* and *CJ422RAC.CT* to the current directory. Also copy the ROM Option Link Specification Template file *CJ422ROP.LKT* to the current directory.

Use the AMX Configuration Generator to generate the ROM option source modules as follows.

```
CJ422CG HDWCFG.UP CJ422ROP.CT CJ422ROP.S
CJ422CG HDWCFG.UP CJ422RAC.CT CJ422RAC.S
CJ422CG HDWCFG.UP CJ422ROP.LKT CJ422ROP.LKS
```

Once the ROM option source modules have been created, you can proceed to build your AMX ROM image and your AMX application as described above.

ARM Debuggers for Windows

The ARM Debugger for Windows (*ADW*), the ARM Extended Debugger (*AXD*) and the ARM RealView[®] Debugger support source level debugging of your AMX 4-Thumb system.

The most effective way to use each of these ARM debuggers is to connect the debugger to the target system under test using ARM's EmbeddedICE[™] or Multi-ICE[®] tool as directed by ARM Ltd.

The ARM Debugger can also operate using a serial (or other) connection to the target system under test. When used in this fashion, you must install the ARM Angel Debug Monitor in your target hardware. Instructions for doing so are provided in the ARM Software Development Toolkit Reference Guide. Your version of the Angel Debug Monitor must provide a device driver for the serial (or other) device used for communication with the ARM Debugger. It is recommended that your driver use polled I/O so that the Angel Debug Monitor can operate with interrupts disabled.

When using the ARM Angel Debug Monitor, be sure to configure AMX to ignore exceptions for undefined instruction execution, SWI software interrupt and FIQ external interrupts, all of which are serviced by Angel. In particular, if the SWI exception is treated as fatal, your AMX application will halt (hang) as soon as the first Angel heartbeat tick attempts to inform the ARM Debugger that your application is alive. Of course, you can also configure the ARM Debugger's remote connection to disable the Angel heartbeat tick.

If you are using a debug monitor such as the Red Hat RedBoot bootstrap monitor on the Intel IQ80321 Evaluation Platform, you must add module *CJ422UD.0* to your link specification file to resolve references by the ARM startup code to Angel semihosting services not provided by the monitor.

Using the *KwikLook* Fault Finder

The *KwikLook*[™] Fault Finder is compatible with the RealView Debugger from the RVDS tools providing full screen, source level, task-aware debugging from within the Microsoft Windows[®] environment. *KwikLook* can be invoked directly from the debugger while at breakpoints giving you finger tip access to your application from the AMX perspective. Note that *KwikLook* and RealView share a common link to the target system.

The *KwikLook* Fault Finder does not provide support for the *AXD* ARM Extended Debugger or the *ADW* ARM Debugger for Windows. Hence, although these debuggers can be used to test your AMX application, the debuggers are not AMX-aware.

3. MetaWare (MW) Tool Guide

AMX™ 4-Thumb has been developed on a PC with Windows® NT v4.0 using the MetaWare tools listed below. The AMX libraries and object modules on the product disks have been generated using the most recent tools listed. If you are not using this toolset, you may have to rebuild the AMX libraries in order to use your out-of-date tools.

MetaWare Tools	<u>v4.1</u>	<u>v4.2</u>	<u>v4.5</u>	<u>v5.1</u>	<u>v5.4</u>
<i>HCARM</i> ARM High C/C++ compiler	R4.1C	R4.2d	R4.5a	R5.1a	R5.4a
<i>ASARM</i> ARM assembler	1.7i	3.07	3.34	3.38	3.4i
<i>LDARM</i> ARM linker/locator	4.8q	5.0o	5.1f	5.2a	5.3h
<i>ARARM</i> ARM archiver (librarian)	2.07	2.09	2.10	2.10	2.12
SeeCode ARM Debugger	1.5	4.2	4.5	5.1	7.0

AMX 4-Thumb and *KwikLook* have been tested on the following platforms.

- Advanced RISC Machines ARM Development Board (ARM7TDMI Version)
- Advanced RISC Machines ARM Development Board (ARM940T Version)
- Advanced RISC Machines Integrator/AP Board (with ARM966E)
- Advanced RISC Machines Integrator/CP Board (with ARM920T)
- Intel XScale IQ80321 Evaluation Platform
- Atmel AT91EB40 Evaluation Board
- Atmel AT91EB42 Evaluation Board
- Atmel AT91SAM7S-EK Evaluation Kit
- Advanced RISC Machines ARM Evaluator-7T (with Samsung S3C4510 processor)
- Cogent Computer Systems, Inc. CSB238 Single Board Computer
(with Cirrus Logic EP7312 processor)
- Freescale i.MX21ADS board

Environment Variables

Set the following environment variables to provide access to all AMX and MetaWare tools, header files, object files and libraries.

<i>CJPATH</i>	Path to AMX installation directory (. . . \AMX422)
<i>PATH</i>	Path to AMX and MetaWare executable programs
<i>TOOLS DIR</i>	Path to MetaWare executable programs
<i>HCDIR</i>	Path to MetaWare installation directory
<i>TMP, TMPPREFIX</i>	Path to a temporary directory for use by MetaWare tools

The AMX libraries have been constructed using the following assumptions. The resulting AMX 4-Thumb libraries are ready for use with any ARM processor which is compatible with the ARM v4T or v5T architecture.

- AMX 4-Thumb target processor is little endian and of ARM v4T architecture
- AMX 4-Thumb is generated in ELF object format
- AMX 4-Thumb assumes software floating point emulation

Object Formats

MetaWare only supports the ELF object format. The AMX 4-Thumb libraries and object modules are provided in ELF format. Your object modules and the AMX and MetaWare libraries and object modules, all in ELF format, can be combined to create an executable module in ELF format suitable for use with the MetaWare SeeCode ARM Debugger.

Parameter Passing Conventions

The MetaWare tools support the C function parameter passing convention defined in the ARM-Thumb Procedure Call Standard (ATPCS). AMX 4-Thumb follows this standard, the parameter passing convention common to all Thumb toolsets supported by KADAK.

Register Usage

AMX 4-Thumb makes the following C interface register assumptions. Registers *a1*, *a2*, *a3*, *a4*, *ip* and *lr* and the flags in *CPSR* can always be altered by C procedures. All other registers are preserved by AMX according to the ATPCS rules for C procedures. Integers and pointers are returned from C procedures in register *a1*. You must NOT use any C compilation switch which changes these register assumptions.

Big or Little Endian

AMX 4-Thumb is delivered ready for use with the little endian model. AMX 4-Thumb will also operate, without modification, on big endian hardware. However, to use AMX on big endian hardware, you must first **rebuild the AMX Library for big endian operation**. You must also be sure to use the MetaWare big endian C startup code and libraries.

To rebuild the AMX Library for big endian operation, set environment variable *AMX_ENDN=B*. If you must rebuild the AMX Library for little endian operation, set environment variable *AMX_ENDN=L* or leave it undefined. Then build the library as described in Appendix D of the AMX User's Guide.

Using the MetaWare C Compiler

All AMX header files *CJ422xxx.H* and the generic AMX include file *CJZZZ.H* must be present in the current directory together with your source file being compiled.

Use the following compilation switches when you are compiling modules for use in the AMX environment.

by default	; structure members are aligned per type size
by default	; integers are 4-byte signed quantities
by default	; promote char and short int values to integers
	; when passing them as function parameters
by default	; output object module <i>FILENAME.O</i>
<i>-c</i>	; compile only
<i>-HL</i>	; compile little endian (<i>-HB</i> for big endian)
<i>-Hthread</i>	; compile for thread-safe use (preferred)
or	
<i>-Hnothread</i>	; compile for non-thread-safe use
<i>-Hnoswst</i>	; no stack checking
<i>-Hthumb</i>	; generate Thumb code
<i>-Hinter</i>	; generate interworking code
<i>-O</i>	; full optimize for speed
<i>-Hon=Version4</i>	; ARM v4 instruction compliance
<i>-g</i>	; (optional) generate debug information
<i>-Hefile=FILENAME.ERR</i>	; redirect error messages to <i>FILENAME.ERR</i>

The compilation command line is therefore of the form:

```
HCARM -c -HL -Hthread -Hnoswst -Hthumb -Hinter -O -Hon=Version4  
-Hefile=FILENAME.ERR FILENAME.C
```

The following command line switches, although optional, are recommended.

-Hpragma=Offwarn(572) ; disable dangerous typecast warnings

Compiling the AMX System Configuration Module

Your AMX System Configuration Module *SYSCFG.C* is compiled as follows. All AMX header files *CJ422xxx.H* and the generic AMX include file *CJZZZ.H* must be present in the current directory together with file *SYSCFG.C*.

Use the *-HL* switch for little endian systems and *-HB* for big endian systems. Use the *-Hthread* switch for thread-safe systems and *-Hnothread* for non-threaded systems.

```
HCARM -c -HL -Hthread -Hthumb -Hnoswst -Hinter -O -Hon=Version4
      -Hefile=SYSCFG.ERR SYSCFG.C
```

Assembling the AMX Target Configuration Module

Your AMX Target Configuration Module *HDWCFG.S* is assembled as follows. The generic AMX header file *CJZZZK.DEF* must be present in the current directory together with file *HDWCFG.S*.

Use the following command line switches when using the MetaWare assembler. Note that all AMX 4-Thumb assembly language modules unconditionally declare that code areas support interworking.

```
by default          ; generate 32-bit ARM code
by default          ; assemble with case sensitivity
by default          ; output object module HDWCFG.O
-le                 ; assemble little endian (-be for big endian)
-noswst             ; no stack checking
-Eo >HDWCFG.ERR    ; redirect error messages to HDWCFG.ERR
                   ; (omit the -Eo switch for v4.2 and later)
```

```
ASARM -le -noswst HDWCFG.S >HDWCFG.ERR
```

Making Libraries

To make a library from a collection of object modules, create a library specification file *YOURLIB.LBM*. Use the MetaWare version of the AMX library specification file *CJ422.LBM* as a guide.

Use the following command line switches when using the MetaWare librarian.

```
-r                 ; replace (add) modules in (to) library
YOURLIB.A          ; output library module YOURLIB.A
>YOURLIB.LBE       ; redirect librarian error messages to YOURLIB.LBE
```

Make your library as follows.

```
ARARM -r YOURLIB.A @YOURLIB.LBM >YOURLIB.LBE
```

Building the AMX Library

When rebuilding the AMX Library using MetaWare tools, you must define the following environment variables. To build the library for little endian operation, set environment variable `AMX_ENDN=L` or leave it undefined. To build the library for big endian operation, set environment variable `AMX_ENDN=B`.

To build the AMX Library with support for MetaWare's thread-safe library, set environment variable `AMX_TSAFE=TSOON` or leave it undefined. To build the library without such thread-safe support, set environment variable `TSAFE=TSOFF`. The library is then built as described in Appendix D of the AMX User's Guide.

The AMX thread-safe support modules require access to MetaWare's `THREAD.H` header file. For compatibility with MetaWare's tools for other processors, AMX expects to find this header file in subdirectory `INC\INTERNAL` in the MetaWare installation directory referenced by environment variable `HCDIR`. Unfortunately, MetaWare's ARM tools provide copies of this header file in several subdirectories in their `LIB\SRC` directory. Hence, before building the AMX Library, you must copy file `THREAD.H` from MetaWare's `LIB\SRC\MW\INC\INTERNAL` installation directory into a new directory `INC\INTERNAL`.

Selecting a MetaWare Library

You must choose a variant of the MetaWare C Runtime Library that is 16-bit, little endian, supports interworking and has NO runtime stack checking. You can use variants of the library which include any of the floating point or frame pointer options if required by your application.

MetaWare provides both little endian and big endian versions of its libraries. All libraries are thread-safe by default. The libraries include function stubs which operate in a non-threaded fashion in the absence of thread-safe support from the AMX Library. Hence, to operate without thread-safe support, simply rebuild the AMX Library to exclude such support and link your system with the MetaWare thread-safe libraries as described later in this guide.

Warning

MetaWare libraries for release v4.2f and earlier have known problems with the stack and heap setup. If you encounter difficulties using the MetaWare C startup code in your embedded system, contact MetaWare for an updated library or contact KADAK for a temporary work around.

Thread-safe Linking with the MetaWare Linker

When used with MetaWare C thread-safe libraries, the modules which form your AMX system must be linked in the following order.

Your *MAIN* module

Other application modules

<i>SYSCFG.O</i>	; AMX System Configuration Module
<i>HDWCFG.O</i>	; AMX Target Configuration Module
<i>CHxxxxxT.O</i>	; AMX chip-specific clock driver or your equivalent
<i>CJ422UF.O</i>	; Launch and leave AMX (may be customized)
<i>CJ422RAC.O</i>	; AMX ROM Access Module (customized) ; (only if AMX placed in a separate ROM) ; (see Appendix C in AMX Target Guide)
<i>CJ422CV.A</i>	; AMX 4-Thumb vc Conversion Library ; (only if converting an AMX 86 v3, AMX 386 v1 or ; AMX 68000 v2 application)
<i>CJ422.A</i>	; AMX 4-Thumb Library

The compiler driver will select the required *CRT1.O* startup module and libraries automatically. The MetaWare C Thread-safe Runtime Libraries *LIBC.A* and *LIBMW.A* can also be specified on the link command line as illustrated in the link example provided later in this guide.

Note

Symbol *cj_mw_tls* (or *cj_mw_tlsmin*) must be undefined (using the linker *-u* switch) to force the full (or minimal) thread-safe support modules to be resolved from the AMX 4-Thumb Library.

Be sure to follow the guidelines for thread-safe library usage presented later in this Tool Guide.

Non-threaded Linking with the MetaWare Linker

When using the MetaWare C thread-safe libraries without AMX thread-safe support, the modules which form your AMX system must be linked in the following order.

Your *MAIN* module

Other application modules

<i>SYSCFG.O</i>	; AMX System Configuration Module
<i>HDWCFG.O</i>	; AMX Target Configuration Module
<i>CHxxxxxT.O</i>	; AMX chip-specific clock driver or your equivalent
<i>CJ422UF.O</i>	; Launch and leave AMX (may be customized)
<i>CJ422RAC.O</i>	; AMX ROM Access Module (customized) ; (only if AMX placed in a separate ROM) ; (see Appendix C in AMX Target Guide)
<i>CJ422CV.A</i>	; AMX 4-Thumb vc Conversion Library ; (only if converting an AMX 86 v3, AMX 386 v1 or ; AMX 68000 v2 application)
<i>CJ422.A</i>	; AMX 4-Thumb Library (rebuilt for non-threaded use)

The compiler driver will select the required *CRT1.O* startup module and libraries automatically. Although the MetaWare thread-safe libraries are used, they will operate in a non-threaded fashion without thread-safe support from AMX. The MetaWare C Runtime Libraries *LIBC.A* and *LIBMW.A* can also be specified on the link command line as illustrated in the link example provided later in this guide.

Note

You **must not** use the linker *-u* switch to force symbol *cj_mw_tls* or *cj_mw_tlsmin* to be undefined.

The AMX C startup module *CJ422SU.O* previously used to replace the MetaWare *CRT1.O* module is no longer used.

Create a link specification file *YOURLINK.LKS*. Use the MetaWare version of the AMX Sample Program link specification file *CJSAMPLE.LKS* as a guide.

Start with the sample link specification file for the board which most closely resembles your hardware configuration.

Note

If you decide to omit any of the link and locate commands from the sample specification, you may encounter link errors or run-time faults.

Link and locate with the MetaWare linker using the following command line switches.

```
-Hthread           ; link for thread-safe use (preferred)
-u cj_mw_tls       ; load AMX 4-Thumb thread-safe support (full)
-u cj_mw_tlsmin    ; load AMX 4-Thumb thread-safe support (minimal)
                  ; (see the topic Guidelines for Thread-safe Library Use)
or
-Hnothread         ; link for non-thread-safe use

-m                ; generate section and symbol map
-HL               ; link little endian (-HB for big endian)
-o YOURLINK.OUT   ; direct link output to file YOURLINK.OUT
YOURLINK.LKS      ; use link specification file YOURLINK.LKS
-Lmwp            ; path to mwpath library directory (for -lxxxx searches)
-lxxxx           ; link library named LIBxxxx.A
>YOURLINK.MAP     ; direct section and symbol map to file YOURLINK.MAP

-xm               ; create hex output in file YOURLINK.HEX
                  ; generate Motorola S-record format
                  ; other formats can be generated
                  ; (see MetaWare manual)
```

The compiler driver's link and locate command line is therefore of the following form.

For little endian use with full thread-safe support, link as follows:

```
HCARM -o YOURLINK.OUT -m -HL -Hthread -xm -u cj_mw_tls
      YOURLINK.LKS -Lmwp -lc -lmw >YOURLINK.MAP
```

For little endian use without full thread-safe support, link as follows:

```
HCARM -o YOURLINK.OUT -m -HL -Hnothread -xm
      YOURLINK.LKS -Lmwp -lc -lmw >YOURLINK.MAP
```

The resulting load module *YOURLINK.OUT* is suitable for use with the MetaWare SeeCode Debugger. The resulting load module *YOURLINK.HEX* is ready for burning into EPROM.

Guidelines for Thread-safe Library Use

KADAK provides support for the thread-safe run-time libraries included with the MetaWare High C/C++ for ARM Embedded Development tools. You only need to use the thread-safe libraries if your application has multiple tasks which concurrently use the library functions which can benefit from thread-safe execution. For example, if only one task does floating point operations, there may be no need to use the thread-safe libraries.

Starting with AMX 4-Thumb v1.03a, support for the thread-safe libraries is built into the AMX 4-Thumb Library. The following files in the AMX 4-Thumb Library satisfy the kernel-dependent requirements of the MetaWare thread-safe libraries.

<i>AMX_LIST.C</i>	Low-level list manipulation
<i>AMX_MUTX.C</i>	Kernel-dependent mutex implementation
<i>AMX_TLS.C</i>	Thread-local storage allocation and freeing
<i>AMX_TLSK.C</i>	Kernel initialization and thread-safe <i>errno</i> support
<i>AMX_TLSX.C</i>	Mutex stubs for minimal thread-safe support

Add one the following statements to your link specification file or to your link command line to force the appropriate thread-safe support modules to be loaded from the AMX 4-Thumb Library.

```
-u cj_mw_tls ; load AMX 4-Thumb thread-safe support (full)
-u cj_mw_tlsmin ; load AMX 4-Thumb thread-safe support (minimal)
```

MetaWare recommends that you compile and link your application modules with the *-Heos=amx* command line switch to specify AMX as the target operating system. However, you can safely omit this switch since all MetaWare specific support for AMX is provided by KADAK, not MetaWare.

The following step is very **IMPORTANT!** If you miss it, *malloc()* will fail because the *bss* and *sbss* sections are not initialized. If your link/locate specification includes a data initialization directive such as

```
INITDATA !data
```

replace it with the following line:

```
INITDATA !data,.bss,.sbss
```

Thread-Local Storage Allocation

Thread-local storage is allocated, assigned and initialized automatically as the MetaWare Library demands. After the AMX kernel has been launched, each task that makes a call to a C library function that requires thread-local storage will be assigned its own private storage block.

If such a task is deleted with a call to AMX procedure `cjtkdelete()`, the task's thread-local storage will not be released!

To delete such a task, you must call the new procedure `cj_mw_tkdelete()`. You will have to add the procedure's prototype in the module in which it is used:

```
CJ_ERRST CJ_CCPP cj_mw_tkdelete(CJ_ID tid, int priority)
```

Note that only one thread storage block is used by the MetaWare Library prior to the AMX launch. After AMX shuts down, the MetaWare Library will be forced to resume using the same thread storage block which was in use before AMX was launched.

Mutex Allocation and Use

Some functions in the C/C++ run-time library must be guarded from concurrent access by multiple threads. Operations of this type include `malloc()`, `free()`, complex floating-point operations such as `log()` and, of course, the I/O library.

The thread-safe MetaWare Library uses a mutex construct to guard access to these functions. The mutex services are provided by AMX in module `AMX_MUTEX.C`.

Before AMX is launched and after AMX has shut down, mutex protection is not required. However, once AMX has been launched, AMX must ensure that the mutex protection is provided.

When a task calls a function such as `malloc()`, the AMX Semaphore Manager is called upon to create a resource semaphore to be used by the library to guard access to its memory allocation services.

Similarly, when a task calls a function such as `log()`, the AMX Semaphore Manager is called upon to create a resource semaphore to be used by the library to guard access to its floating-point services.

The AMX mutex services in file `AMX_MUTEX.C` allow a maximum of 128 concurrent mutex locks. These locks satisfy the startup and exit requirements of MetaWare. Only a small fraction of these mutex locks will be used by your AMX application and hence require the allocation of an AMX resource semaphore.

The number of resource semaphores required by the MetaWare Library depends on which library services your tasks actually use. In general, the number does not depend on how many tasks use those services.

You must adjust your AMX system configuration to include the AMX Semaphore Manager and to account for the additional semaphores that will be required.

Fatal Thread-safe Conditions

Any of the following conditions are considered fatal:

1. A thread-storage block cannot be allocated.
2. A mutex lock is not available because the supply is exhausted.
3. An AMX semaphore cannot be created for mutex locking purposes.
4. An attempt to reference an allocated AMX semaphore was rejected by AMX.

If any of these conditions are encountered, the application will hang forever in procedure `cj_mw_fatal()` in module `AMX_MUTX.C`. When testing your application, always run with a breakpoint on `cj_mw_fatal()`. Be sure to set the breakpoint BEFORE entering the MetaWare C/C++ start-up code.

Error 1 implies that `malloc()` cannot provide the required memory.

Error 2 indicates that more than 128 mutex locks are needed.
Adjust the definition in file `AMX_MUTX.C`.

Error 3 indicates that you need more AMX semaphores or that you may not have included the AMX Semaphore Manager in your configuration.

Error 4 usually implies that data corruption has occurred. The private AMX or MetaWare Library data structures used to manage thread-safe operation have been damaged.

Linking a Separate AMX ROM

AMX can be committed to a separate ROM as described in Appendix C of the AMX Target Guide. Use the AMX Configuration Manager to edit your Target Parameter File *HDWCFG.UP* to define your ROM option parameters. Then use the Manager to generate your ROM Option Module *CJ422ROP.S*, ROM Access Module *CJ422RAC.S* and ROM Option link specification file *CJ422ROP.LKS*.

The ROM Option and ROM Access source modules are assembled as follows. Use the *-le* switch for little endian systems and *-be* for big endian systems. For MetaWare v4.1 tools, you must use the *-Eo* switch without the braces.

```
ASARM -le -noswst CJ422ROP.S {-Eo} >CJ422ROP.ERR
```

```
ASARM -le -noswst CJ422RAC.S {-Eo} >CJ422RAC.ERR
```

The AMX ROM is linked using link specification file *CJ422ROP.LKS* as follows. Use the *-HL* switch for little endian systems and *-HB* for big endian systems. Use the *-Hthread* switch for thread-safe systems and *-Hnothread* for non-thread-safe systems. You must use the *-Hnocrt* switch to preclude linking the MetaWare C startup code and libraries.

```
HCARM -o AMXROM.OUT -m -HL -Hthread -Hnocrt -xm  
CJ422ROP.LKS >AMXROM.MAP
```

This example generates file *AMXROM.HEX* in Motorola S-record format suitable for transfer to ROM. Other formats supported by MetaWare can be selected with the appropriate command switch.

When you link your AMX application, be sure to include your customized AMX ROM Access Module *CJ422RAC.O* (created above) in your system link specification file.

Using the AMX Configuration Generator

If you cannot use the AMX Configuration Manager, you may still be able to use the stand-alone AMX Configuration Generator to generate the ROM Option Module *CJ422ROP.S*, ROM Access Module *CJ422RAC.S* and ROM Option link specification file *CJ422ROP.LKS*.

Copy the ROM Option and ROM Access template files *CJ422ROP.CT* and *CJ422RAC.CT* to the current directory. Also copy the ROM Option Link Specification Template file *CJ422ROP.LKT* to the current directory.

Use the AMX Configuration Generator to generate the ROM option source modules as follows.

```
CJ422CG HDWCFG.UP CJ422ROP.CT CJ422ROP.S  
CJ422CG HDWCFG.UP CJ422RAC.CT CJ422RAC.S  
CJ422CG HDWCFG.UP CJ422ROP.LKT CJ422ROP.LKS
```

Once the ROM option source modules have been created, you can proceed to build your AMX ROM image and your AMX application as described above.

MetaWare SeeCode Debugger

The MetaWare SeeCode Debugger supports source level debugging of your AMX 4-Thumb system.

The most effective way to use the MetaWare SeeCode Debugger is to connect it to the target system under test using the ARM Ltd. Multi-ICE[®] tool or one of the wiggler devices supported by SeeCode.

The MetaWare SeeCode Debugger can also operate using a serial (or other) connection to the target system under test. When used in this fashion, you must install the ARM Angel Debug Monitor in your target hardware. Instructions for doing so are provided in the ARM Software Development Toolkit Reference Guide. Your version of the Angel Debug Monitor must provide a device driver for the serial (or other) device used for communication with the MetaWare SeeCode Debugger. It is recommended that your driver use polled I/O so that the Angel Debug Monitor can operate with interrupts disabled.

When using SeeCode with the ARM Angel Debug Monitor, be sure to configure AMX to ignore exceptions for undefined instruction execution, SWI software interrupt and FIQ external interrupts, all of which are serviced by Angel. In particular, if the SWI exception is treated as fatal, your AMX application will halt (hang) as soon as the first Angel heartbeat tick attempts to inform SeeCode that your application is alive.

Using the *KwikLook* Fault Finder

The *KwikLook*[™] Fault Finder is compatible with the SeeCode Debugger providing full screen, source level, task-aware debugging from within the Microsoft Windows[®] environment. *KwikLook* can be invoked directly from the debugger while at breakpoints giving you finger tip access to your application from the AMX perspective. Note that *KwikLook* and SeeCode share a common link to the target system.

This page left blank intentionally.

4. ARM Ltd. (RV) Tool Guide (for RVDS)

AMX™ 4-Thumb has been developed on a PC with Windows® NT v4.0 using the ARM RealView® Development Suite from Advanced RISC Machines (ARM). These tools are listed below. The AMX libraries and object modules on the product disks have been generated using the most recent tools listed. If you are not using this toolset, you may have to rebuild the AMX libraries in order to use your out-of-date tools.

ARM Tools		<u>v2.0</u>	<u>v2.1</u>	<u>v2.2</u>
<i>ARMCC</i>	ARM C/C++ compiler	2.0.1	2.1	2.2
<i>ARMASM</i>	Assembler	2.0.1	2.1	2.2
<i>ARMLINK</i>	Linker/locator	2.0.1	2.1	2.2
<i>ARMAR</i>	Librarian	2.0.1	2.1	2.2
<i>RVDEBUG</i>	ARM RealView Debugger	1.7	1.7	1.8

AMX 4-Thumb and *KwikLook* have been tested on the following platforms.

Advanced RISC Machines ARM Development Board (ARM7TDMI Version)
Advanced RISC Machines ARM Development Board (ARM940T Version)
Advanced RISC Machines Integrator/AP Board (with ARM966E)
Advanced RISC Machines Integrator/CP Board (with ARM920T)
Intel XScale IQ80321 Evaluation Platform
Atmel AT91EB40 Evaluation Board
Atmel AT91EB42 Evaluation Board
Atmel AT91SAM7S-EK Evaluation Kit
Advanced RISC Machines ARM Evaluator-7T (with Samsung S3C4510 processor)
Cogent Computer Systems, Inc. CSB238 Single Board Computer
(with Cirrus Logic EP7312 processor)
Freescale i.MX21ADS board

Environment Variables

Set the following environment variables to provide access to all AMX and ARM tools, header files, object files and libraries.

<i>CJPATH</i>	Path to AMX installation directory (. . . \AMX422)
<i>PATH</i>	Path to AMX and ARM executable programs
<i>RVCT20INC</i>	Path to ARM C include files (use correct version number)
<i>RVCT20LIB</i>	Path to ARM C library files (use correct version number)
<i>TMP</i>	Path to a temporary directory for use by ARM tools

AMX Libraries

The AMX libraries have been constructed using the following assumptions. The resulting AMX 4-Thumb libraries are ready for use with any ARM processor which is compatible with the ARM v4T or v5T architecture.

- AMX 4-Thumb target processor is little endian and of ARM v4T architecture
- AMX 4-Thumb is generated in ELF object format
- AMX 4-Thumb assumes software floating point emulation

Object Formats

ARM supports the Executable and Linking Format (ELF). The AMX 4-Thumb libraries and object modules are provided in ELF format. Your object modules and the AMX and ARM libraries and object modules, all in ELF format, can be combined to create an executable module in ARM Executable Format (AXF) suitable for use with the ARM RealView Debugger.

Parameter Passing Conventions

The ARM tools support the C function parameter passing convention defined in the ARM Architecture Procedure Call Standard (AAPCS). AMX 4-Thumb follows this standard, the parameter passing convention common to all Thumb toolsets supported by KADAK.

Register Usage

AMX 4-Thumb makes the following C interface register assumptions. Registers *a1*, *a2*, *a3*, *a4*, *ip* and *lr* and the flags in *CPSR* can always be altered by C procedures. All other registers are preserved by AMX according to the AAPCS rules for C procedures. Integers and pointers are returned from C procedures in register *a1*. You must NOT use any C compilation switch which changes these register assumptions.

Big or Little Endian

AMX 4-Thumb is delivered ready for use with the little endian model. AMX 4-Thumb will also operate, without modification, on big endian hardware. However, to use AMX on big endian hardware, you must first **rebuild the AMX Library for big endian operation**. Be sure to use the ARM Ltd. big endian C startup code and libraries.

To rebuild the AMX Library for big endian operation, set environment variable *AMX_ENDN=B*. If you must rebuild the AMX Library for little endian operation, set environment variable *AMX_ENDN=L* or leave it undefined. Then build the library as described in Appendix D of the AMX User's Guide.

Using the ARM Ltd. C Compiler

All AMX header files *CJ422xxx.H* and the generic AMX include file *CJZZZ.H* must be present in the current directory together with your source file being compiled.

Use the following compilation switches when you are compiling modules for use in the AMX environment.

by default	; structure members are aligned per type size
by default	; integers are 4-byte signed quantities
by default	; promotion of char and short int values to integers
	; is done by the function receiving the parameters
<i>--cpu 4T</i>	; ARM v4T Thumb instruction compliance
<i>--thumb</i>	; compile for 16-bit, Thumb mode execution
<i>-c</i>	; compile only
<i>-Otime</i>	; optimize for speed of execution
<i>--apcs /noswst/inter</i>	; no stack checking; use interworking
<i>--li</i>	; compile little endian (<i>--bi</i> for big endian)
<i>-g</i>	; (optional) generate debug information
<i>-o FILENAME.O</i>	; output object module <i>FILENAME.O</i>
<i>--errors FILENAME.ERR</i>	; redirect C error messages to <i>FILENAME.ERR</i>

The compilation command line is therefore of the following form.

```
ARMCC --cpu 4T --thumb -c -Otime --apcs /noswst/inter --li  
-o FILENAME.O --errors FILENAME.ERR FILENAME.C
```

Compiling the AMX System Configuration Module

Your AMX System Configuration Module *SYSCFG.C* is compiled as follows. All AMX header files *CJ422xxx.H* and the generic AMX include file *CJZZZ.H* must be present in the current directory together with file *SYSCFG.C*.

Use the *--li* switch for little endian systems and *--bi* for big endian systems.

Use the following command line.

```
ARMCC --cpu 4T --thumb -c -Otime --apcs /noswst/inter --li
-o SYSCFG.O --errors SYSCFG.ERR SYSCFG.C
```

Assembling the AMX Target Configuration Module

Your AMX Target Configuration Module *HDWCFG.S* is assembled as follows. The generic AMX header file *CJZZZK.DEF* must be present in the current directory together with file *HDWCFG.S*.

Use the following command line switches when using the ARM assembler. Note that all AMX 4-Thumb assembly language modules generate code areas that support interworking.

by default	; generate 32-bit ARM code
by default	; assemble with case sensitivity
<i>--apcs /noswst/inter</i>	; define AAPCS register names
	; inhibit stack checks; support interworking
<i>--cpu 4T</i>	; ARM v4T Thumb instruction compliance
<i>--li</i>	; assemble little endian (<i>--bi</i> for big endian)
<i>-o HDWCFG.O</i>	; output object module <i>HDWCFG.O</i>
<i>-e HDWCFG.ERR</i>	; redirect error messages to <i>HDWCFG.ERR</i>

The assembly command line is of the following form.

```
ARMASM --cpu 4T --apcs /noswst/inter --li
-e HDWCFG.ERR -o HDWCFG.O HDWCFG.S
```

Making Libraries

To make a library from a collection of object modules, create a library specification file *YOURLIB.LBM*. Use the ARM Ltd. version of the AMX library specification file *CJ422.LBM* as a guide.

Use the following librarian command line switches.

```
--create          ; create a library module
-c                ; create a new library module (without warning)
--via YOURLIB.LBM ; use library command file YOURLIB.LBM
>YOURLIB.LBE     ; redirect librarian error messages to YOURLIB.LBE
```

Make your library as follows.

```
ARMAR --create -c --via YOURLIB.LBM >YOURLIB.LBE
```

Selecting an ARM Library

You must choose a variant of the ARM C Runtime Library that is 16-bit, little endian, supports interworking and has NO runtime stack checking.

The ARM linker will automatically select the correct library according the linker command line switches that you specify.

If you rebuild the AMX libraries for big endian operation, be sure that the linker selects the appropriate big endian ARM C Runtime Library for use with your application.

Linking with the ARM Ltd. Linker

When used with the ARM C tools, the modules which form your AMX system must be linked in the following order.

Your *MAIN* module

Other application modules

<i>SYSCFG.O</i>	; AMX System Configuration Module
<i>HDWCFG.O</i>	; AMX Target Configuration Module
<i>CHxxxxxT.O</i>	; AMX chip-specific clock driver or your equivalent
<i>CJ422UF.O</i>	; Launch and leave AMX (may be customized)
<i>CJ422UD.O</i>	; KADAK's ARM Library Semihosting Exclusion Module ; (Resolves ARM semihosting references if not provided ; by the debug monitor. For example, include this module ; when using the RedBoot monitor on the IQ80321 board.)
<i>CJ422RAC.O</i>	; AMX ROM Access Module (customized) ; (only if AMX placed in a separate ROM) ; (see Appendix C in AMX Target Guide)
<i>CJ422CV.A</i>	; AMX 4-Thumb vc Conversion Library ; (only if converting an AMX 86 v3, AMX 386 v1 or ; AMX 68000 v2 application)
<i>CJ422.A</i>	; AMX 4-Thumb Library ARM C Runtime Library for target hardware

Create a link specification file *YOURLINK.LKS*. Use the AMX 4-Thumb Sample Program link specification file *CJSAMPLE.LKS* as a guide.

Start with the sample link specification file for the board which most closely resembles your hardware configuration.

Note

If you decide to omit any of the link and locate commands from the sample specification, you may encounter link errors or run-time faults.

Link and locate with the ARM linker using the following command line switches. To reduce the command line length, many of these command line switches should be placed in your link specification file *YOURLINK.LKS*.

by default	; link with case sensitivity
<i>--debug</i>	; include debug records in output
<i>--nodebug</i>	; exclude debug records from output
<i>--symbols</i>	; generate a symbol listing
<i>--elf</i>	; generate ELF format executable file
	; suitable for use with the ARM RealView Debugger
<i>--errors YOURLINK.LKE</i>	; send link errors to file <i>YOURLINK.LKE</i>
<i>-o YOURLINK.AXF</i>	; direct link output to file <i>YOURLINK.AXF</i>
<i>--via YOURLINK.LKS</i>	; use link specification file <i>YOURLINK.LKS</i>
	; which has following command switches:
<i>--info totals</i>	; generate information on code and data sizes
<i>--map</i>	; generate an area map listing
<i>--list YOURLINK.MAP</i>	; send map listing to file <i>YOURLINK.MAP</i>
<i>--ro-base 0x8000</i>	; locate read only areas at this address
<i>--rw-base 0x18000</i>	; locate read/write areas at this address

The link and locate command line is therefore of the following form.

```
ARMLINK --nodebug --symbols --elf --errors YOURLINK.LKE  
-o YOURLINK.AXF --via YOURLINK.LKS
```

The resulting load module *YOURLINK.AXF* in ELF format is suitable for use with the ARM RealView Debugger. Load modules in other formats ready for burning into EPROM can also be created.

Linking a Separate AMX ROM

AMX can be committed to a separate ROM as described in Appendix C of the AMX Target Guide. Use the AMX Configuration Manager to edit your Target Parameter File *HDWCFG.UP* to define your ROM option parameters. Then use the Manager to generate your ROM Option Module *CJ422ROP.S*, ROM Access Module *CJ422RAC.S* and ROM Option link specification file *CJ422ROP.LKS*.

The ROM Option and ROM Access source modules are assembled as follows. Use the *--li* for little endian systems and *--bi* switch for big endian systems.

```
ARMASM --cpu 4T --apcs /noswst/inter --li -e CJ422ROP.ERR
      -o CJ422ROP.O CJ422ROP.S
```

```
ARMASM --cpu 4T --apcs /noswst/inter --li -e CJ422RAC.ERR
      -o CJ422RAC.O CJ422RAC.S
```

The AMX ROM is linked using link specification file *CJ422ROP.LKS* as follows.

```
ARMLINK --nodebug --errors AMXROM.LKE -o AMXROM.AXF
      --via CJ422ROP.LKS
```

This example generates file *AMXROM.AXF* in ELF format. The ARM *FROMELF.EXE* utility can be used to convert this ELF file to other formats suitable for transfer to ROM.

When you link your AMX application, be sure to include your customized AMX ROM Access Module *CJ422RAC.O* (created above) in your system link specification file.

Using the AMX Configuration Generator

If you cannot use the AMX Configuration Manager, you may still be able to use the stand-alone AMX Configuration Generator to generate the ROM Option Module *CJ422ROP.S*, ROM Access Module *CJ422RAC.S* and ROM Option link specification file *CJ422ROP.LKS*.

Copy the ROM Option and ROM Access template files *CJ422ROP.CT* and *CJ422RAC.CT* to the current directory. Also copy the ROM Option Link Specification Template file *CJ422ROP.LKT* to the current directory.

Use the AMX Configuration Generator to generate the ROM option source modules as follows.

```
CJ422CG HDWCFG.UP CJ422ROP.CT CJ422ROP.S
CJ422CG HDWCFG.UP CJ422RAC.CT CJ422RAC.S
CJ422CG HDWCFG.UP CJ422ROP.LKT CJ422ROP.LKS
```

Once the ROM option source modules have been created, you can proceed to build your AMX ROM image and your AMX application as described above.

ARM Debuggers for Windows

The ARM RealView[®] Debugger supports source level debugging of your AMX 4-Thumb system.

The most effective way to use the ARM RealView Debugger is to connect the debugger to the target system under test using ARM's RealView ICE or Multi-ICE[®] tool as directed by ARM Ltd.

The RealView Debugger can also operate using a serial (or other) connection to the target system under test. When used in this fashion, you must install the ARM Angel Debug Monitor in your target hardware. Your version of the Angel Debug Monitor must provide a device driver for the serial (or other) device used for communication with the ARM Debugger. It is recommended that your driver use polled I/O so that the Angel Debug Monitor can operate with interrupts disabled.

When using the ARM Angel Debug Monitor, be sure to configure AMX to ignore exceptions for undefined instruction execution, SWI software interrupt and FIQ external interrupts, all of which are serviced by Angel. In particular, if the SWI exception is treated as fatal, your AMX application will halt (hang) as soon as the first Angel heartbeat tick attempts to inform the debugger that your application is alive. Of course, you can also configure the debugger's remote connection to disable the Angel heartbeat tick.

If you are using a debug monitor such as the Red Hat RedBoot bootstrap monitor on the Intel IQ80321 Evaluation Platform, you must add module *CJ422UD.0* to your link specification file to resolve references by the ARM startup code to Angel semihosting services not provided by the monitor.

Using the *KwikLook* Fault Finder

The *KwikLook*[™] Fault Finder is compatible with the RealView Debugger providing full screen, source level, task-aware debugging from within the Microsoft Windows[®] environment. *KwikLook* can be invoked directly from the debugger while at breakpoints giving you finger tip access to your application from the AMX perspective. Note that *KwikLook* and RealView share a common link to the target system.

This page left blank intentionally.

5. Metrowerks (ME) Tool Guide

AMX™ 4-Thumb has been developed on a PC with Windows® NT v4.0 using the Metrowerks tools listed below. The AMX libraries and object modules on the product disks have been generated using the most recent tools listed. If you are not using this toolset, you may have to rebuild the AMX libraries in order to use your out-of-date tools.

Metrowerks Tools	<u>v2.1.1</u>
<i>MWCCARM</i>	ARM C++ compiler 2.0
<i>MWASMARM</i>	ARM assembler 1.0
<i>MWLDARM</i>	ARM linker/librarian 2.0
	CodeWarrior IDE
	CodeWarrior ARM Debugger
	<i>MetroTRK</i> Target Resident Kernel

AMX 4-Thumb and *KwikLook* have been tested on the following platforms.

- Advanced RISC Machines ARM Development Board (ARM7TDMI Version)
- Advanced RISC Machines ARM Development Board (ARM940T Version)
- Advanced RISC Machines Integrator/AP Board (with ARM966E)
- Advanced RISC Machines Integrator/CP Board (with ARM920T)
- Intel XScale IQ80321 Evaluation Platform
- Atmel AT91EB40 Evaluation Board
- Atmel AT91EB42 Evaluation Board
- Atmel AT91SAM7S-EK Evaluation Kit
- Advanced RISC Machines ARM Evaluator-7T (with Samsung S3C4510 processor)
- Cogent Computer Systems, Inc. CSB238 Single Board Computer
(with Cirrus Logic EP7312 processor)
- Freescall i.MX21ADS board

Environment Variables

Set the following environment variables to provide access to all AMX and Metrowerks tools, header files, object files and libraries.

<i>CJPATH</i>	Path to AMX installation directory (. . . \AMX422)
<i>PATH</i>	Path to AMX and Metrowerks executable programs
<i>TMPDIR</i>	Path to a temporary directory for use by Metrowerks tools
<i>CWFolder</i>	Path to Metrowerks installation directory
<i>MWCIncludes</i>	Path to Metrowerks include directories
<i>MWLibraries</i>	Path to Metrowerks library directories
<i>MWLibraryFiles</i>	Filenames of Metrowerks C libraries to be searched

Command Line Tools

The Metrowerks CodeWarrior Integrated Development Environment (IDE) provides a software development environment within which you can readily create a project which incorporates AMX. However, the AMX library construction process is independent of the CodeWarrior IDE.

To make the AMX libraries and to construct an AMX application as described in this Tool Guide, you must use the Metrowerks command line tools. It is assumed that the following Metrowerks tools have been copied from the Metrowerks installation directory to the Metrowerks *BIN* directory and renamed as follows.

```
... \ARM_EABI_Tools\Command_Line_Tools\mwccarm.exe           to MWCC.EXE
... \ARM_EABI_Tools\Command_Line_Tools\mwasmarm.exe         to MWASM.EXE
... \ARM_EABI_Tools\Command_Line_Tools\mwldarm.exe          to MWLD.EXE
```

AMX Libraries

The AMX libraries have been constructed using the following assumptions. The resulting AMX 4-Thumb libraries are ready for use with any ARM processor which is compatible with the ARM v4T or v5T architecture.

- AMX 4-Thumb target processor is little endian and of ARM v4T architecture
- AMX 4-Thumb is generated in ELF object format
- AMX 4-Thumb assumes software floating point emulation

Object Formats

Metrowerks supports the Executable and Linking Format (ELF). The AMX 4-Thumb libraries and object modules are provided in ELF format. Your object modules and the AMX and Metrowerks libraries and object modules, all in ELF format, can be combined to create an executable module in ELF format suitable for use with the Metrowerks CodeWarrior Debugger.

Parameter Passing Conventions

The Metrowerks tools support the C function parameter passing convention defined in the ARM-Thumb Procedure Call Standard (ATPCS). AMX 4-Thumb follows this standard, the parameter passing convention common to all ARM toolsets supported by KADAK.

Register Usage

AMX 4-Thumb makes the following C interface register assumptions. Registers *a1*, *a2*, *a3*, *a4*, *ip* and *lr* and the flags in *CPSR* can always be altered by C procedures. All other registers are preserved by AMX according to the ATPCS rules for C procedures. Integers and pointers are returned from C procedures in register *a1*. You must NOT use any C compilation switch which changes these register assumptions.

Using the Metrowerks C Compiler

All AMX header files *CJ422xxx.H* and the generic AMX include file *CJZZZ.H* must be present in the current directory together with your source file being compiled.

Use the following compilation switches when you are compiling modules for use in the AMX environment.

by default	; no stack checking
by default	; structure members are aligned per type size
by default	; integers are 4-byte signed quantities
by default	; promotion of char and short int values to integers
	; is done by the function receiving the parameters
<i>-c</i>	; compile only
<i>-proc v4t</i>	; ARM v4T instruction compliance
<i>-thumb</i>	; compile for 16-bit, Thumb mode execution
<i>-interworking</i>	; generate interworking code
<i>-little</i>	; compile little endian (<i>-big</i> for big endian)
<i>-sdatathreshold 0</i>	; small mutable data section is empty
<i>-nopic -nopid</i>	; no position independent code or data
<i>-Cpp_exceptions off</i>	; disable C++ exceptions
<i>-o</i>	; output object module <i>FILENAME.O</i>
<i>>FILENAME.ERR</i>	; redirect C error messages to <i>FILENAME.ERR</i>
<i>-Op</i>	; optimize for speed
<i>-g</i>	; (optional) generate debug information

The compilation command line is therefore of the form:

```
MWCC -c -proc v4t -thumb -interworking -little -sdatathreshold 0
      -nopic -nopid -Cpp_exceptions off -Op
      -o FILENAME.O FILENAME.C >FILENAME.ERR
```

If the command line becomes too long, use a command file. For example, create a text file *CCSW.CMD* which contains the following command string.

```
-c -proc v4t -thumb -interworking -little -sdatathreshold 0
-nopic -nopid -Cpp_exceptions off -Op
```

The compilation command line can then reduce to the form:

```
MWCC @CCSW.CMD -o FILENAME.O FILENAME.C >FILENAME.ERR
```

Compiling the AMX System Configuration Module

Your AMX System Configuration Module *SYSCFG.C* is compiled as follows. All AMX header files *CJ422xxx.H* and the generic AMX include file *CJZZZ.H* must be present in the current directory together with file *SYSCFG.C*.

```
MWCC -c -proc v4t -thumb -interworking -little -sdatathreshold 0
      -nopic -nopic -Cpp_exceptions off -Op
      -o SYSCFG.O SYSCFG.C >SYSCFG.ERR
```

Assembling the AMX Target Configuration Module

Your AMX Target Configuration Module *HDWCFG.S* is assembled as follows. The generic AMX header file *CJZZZK.DEF* must be present in the current directory together with file *HDWCFG.S*.

The Metrowerks assembler requires the following command line switches.. Note that all AMX 4-Thumb assembly language modules generate code areas that support interworking.

```
by default          ; generate 32-bit ARM code
by default          ; assemble with case sensitivity
by default          ; inhibit stack checking
-proc arm4t        ; ARM v4T instruction compliance
-little            ; assemble little endian (-big for big endian)
-o HDWCFG.O        ; output object module HDWCFG.O
-e HDWCFG.ERR     ; redirect error messages to HDWCFG.ERR
```

The Metrowerks command to invoke the assembler is as follows.

```
MWASM -proc arm4t -little -o HDWCFG.O HDWCFG.S >HDWCFG.ERR
```

Making Libraries

To make a library from a collection of object modules, create a library specification file *YOURLIB.LBM*. Use the Metrowerks version of the AMX library specification file *CJ422.LBM* as a guide.

Use the following command line switches when using the Metrowerks linker/librarian.

```
-library           ; create a library
-o YOURLIB.A       ; output library module YOURLIB.A
>YOURLIB.LBE      ; redirect librarian error messages to YOURLIB.LBE
```

Make your library as follows.

```
MWLD -library -o YOURLIB.A @YOURLIB.LBM >YOURLIB.LBE
```

Linking with the Metrowerks Linker

When used with Metrowerks C, the modules which form your AMX system must be linked in the following order.

Your *MAIN* module

Other application modules

<i>SYSCFG.O</i>	; AMX System Configuration Module
<i>HDWCFG.O</i>	; AMX Target Configuration Module
<i>CHxxxxxT.O</i>	; AMX chip-specific clock driver or your equivalent
<i>CJ422UF.O</i>	; Launch and leave AMX (may be customized)
<i>CJ422RAC.O</i>	; AMX ROM Access Module (customized) ; (only if AMX placed in a separate ROM) ; (see Appendix C in AMX Target Guide)
<i>CJ422CV.A</i>	; AMX 4-Thumb vc Conversion Library ; (only if converting an AMX 86 v3, AMX 386 v1 or ; AMX 68000 v2 application)
<i>CJ422.A</i>	; AMX 4-Thumb Library ; Metrowerks C Runtime Libraries are automatically loaded ; per environment variable <i>MWLibraryFiles</i>

Create a link specification file *YOURLINK.LKS*. Use the Metrowerks version of the AMX Sample Program link specification file *CJSAMPLE.LKS* as a guide.

Create a linker command file *YOURLINK.LCF*. Use the Metrowerks version of the AMX Sample Program linker command file *CJSAMPLE.LCF* as a guide.

Start with the sample link specification file and linker command file for the board which most closely resembles your hardware configuration.

Note

If you decide to omit any of the link and locate commands from the sample specification, you may encounter link errors or run-time faults.

Link and locate with the Metrowerks linker and locator using the following command line switches.

```
by default           ; main entry point is at symbol __start
-proc v4t          ; ARM v4T instruction compliance
-thumb             ; compile for 16-bit, Thumb mode execution
-interworking     ; generate interworking code
-little           ; target processor is little endian (-big for big endian)
-nopic -nopic     ; no position independent code or data
-map unused       ; direct section and symbol summary to
                   ; file YOURLINK.OUT.XMAP
                   ; include unused symbols in map file output
-g                 ; (optional) add debug information to the output to file
-srec             ; direct HEX output to file YOURLINK.OUT.S19
                   ; generate Motorola S-record format
-o                ; direct link output to file YOURLINK.OUT
                   ; use linker command file YOURLINK.LCF
@                   ; use link specification file YOURLINK.LKS
>                   ; direct link error messages to file YOURLINK.LKE
```

To avoid an overlength command line, create a command file *LDSW.CMD*, a text file containing the command line switches which you require.

```
-proc v4t -thumb -interworking -little
-nopic -nopic -map unused -srec
```

The link and locate command line is then of the form:

```
MWLD @LDSW.CMD -o YOURLINK.OUT
      YOURLINK.LCF @YOURLINK.LKS >YOURLINK.LKE
```

The resulting load module *YOURLINK.OUT* is suitable for use with the Metrowerks CodeWarrior ARM debugger.

The resulting load module *YOURLINK.OUT.S19* is ready for burning into EPROM.

Linking a Separate AMX ROM

AMX can be committed to a separate ROM as described in Appendix C of the AMX Target Guide. Use the AMX Configuration Manager to edit your Target Parameter File *HDWCFG.UP* to define your ROM option parameters. Then use the Manager to generate your ROM Option Module *CJ422ROP.S*, ROM Access Module *CJ422RAC.S* and ROM Option linker command file *CJ422ROP.LCF*.

The AMX Configuration Manager must have access to the ROM Option Linker Command Template file *CJ422ROP.LCT*. If you have installed AMX for multiple toolsets, the Manager may not be referencing the Metrowerks toolset directory *TOOLME* for its template files. Go to the File, Templates... menu and, from the list of selectors, choose the selector for the ROM Option Link/Locate File. Adjust the configuration template by browsing for the file *TOOLME\CFG\CJ422ROP.LCT*.

The ROM Option and ROM Access source modules are assembled as follows.

```
MWASM -proc arm4t -little -o CJ422ROP.O CJ422ROP.S >CJ422ROP.ERR
```

```
MWASM -proc arm4t -little -o CJ422RAC.O CJ422RAC.S >CJ422RAC.ERR
```

The AMX ROM is linked using linker command file *CJ422ROP.LCF* and link specification file *CJ422ROP.LKS* as follows.

```
MWLD MWASM -proc arm4t -thumb -interworking -little -nopic -nopid  
-map unused -srec -main "cjksenter"  
-o AMXROM.OUT CJ422ROP.LCF @CJ422ROP.LKS >AMXROM.LKE
```

This example generates file *AMXROM.OUT.S19* in Motorola S-record format suitable for transfer to ROM.

Note that command line switch *-main "cjksenter"* is used to prevent loading of the default C startup module.

When you link your AMX application, be sure to include your customized AMX ROM Access Module *CJ422RAC.O* (created above) in your system link specification file.

Using the AMX Configuration Generator

If you cannot use the AMX Configuration Manager, you may still be able to use the stand-alone AMX Configuration Generator to generate the ROM Option Module *CJ422ROP.S*, ROM Access Module *CJ422RAC.S* and ROM Option linker command file *CJ422ROP.LCF*.

Copy the ROM Option and ROM Access template files *CJ422ROP.CT* and *CJ422RAC.CT* to the current directory. Also copy the ROM Option Linker Command Template file *CJ422ROP.LCT* to the current directory.

Use the AMX Configuration Generator to generate the ROM option source modules as follows.

```
CJ422CG HDWCFG.UP CJ422ROP.CT CJ422ROP.S
CJ422CG HDWCFG.UP CJ422RAC.CT CJ422RAC.S
CJ422CG HDWCFG.UP CJ422ROP.LCT CJ422ROP.LCF
```

Once the ROM option source modules have been created, you can proceed to build your AMX ROM image and your AMX application as previously described.

Metrowerks CodeWarrior Debugger

The Metrowerks CodeWarrior[®] ARM Debugger supports source level debugging of your AMX 4-Thumb system.

The most effective way to use the CodeWarrior Debugger is to connect it to the target system under test using the ARM Ltd. Multi-ICE[®] tool or one of the wiggler devices supported by CodeWarrior.

The CodeWarrior Debugger can also operate using a serial (or other) connection to the target ARM system under test. When used in this fashion, you must install the CodeWarrior *MetroTRK* Target Resident Kernel in your target hardware. Instructions for doing so are provided in the CodeWarrior Reference Manual. Your version of the Target Resident Kernel must provide a device driver for the serial (or other) device used for communication with the CodeWarrior Debugger. It is recommended that your driver use polled I/O so that the Target Resident Kernel can operate with interrupts disabled.

Using the *KwikLook* Fault Finder

The *KwikLook*[™] Fault Finder is compatible with the CodeWarrior Debugger providing full screen, source level, task-aware debugging from within the Microsoft Windows[®] environment. *KwikLook* can be invoked directly from the debugger while at breakpoints giving you finger tip access to your application from the AMX perspective. Note that *KwikLook* and CodeWarrior share a common link to the target system.

This page left blank intentionally.