



# **DesignWare ARC AXC003 CPU Card User Guide**

---

Version 6323-018 May 2017

## Copyright Notice and Proprietary Information Notice

© 2017 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

### Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

### Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

### Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>.

All other product or company names may be trademarks of their respective owners.

### Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.  
690 E. Middlefield Road  
Mountain View, CA 94043  
[www.synopsys.com](http://www.synopsys.com)

# Contents

Contents.....	3
List of Figures .....	7
List of Tables.....	9
1 Package Contents.....	10
1.1 DesignWare ARC AXS103 Software Development Platform .....	10
1.2 DesignWare ARC AXC003 CPU Card (Standalone) .....	11
2 Getting Started.....	12
2.1 Mounting the CPU Card .....	12
2.2 Performing a Self-Test .....	12
3 Default Board Settings .....	15
3.1 Default Jumper Settings on the AXC003 CPU Card .....	15
3.2 Default Boot-Mode Settings on the ARC SDP Mainboard.....	16
4 CPU Core Selection.....	17
4.1 Supported CPU Cores.....	17
4.2 Core Selection.....	17
4.2.1 ARC HS36 CPU .....	18
4.2.2 ARC HS34 CPU .....	18
4.2.3 ARC HS38 Core 0.....	18
4.2.4 ARC HS38 Core 1 .....	18
5 Self-Tests.....	19
5.1 Self-Test Overview.....	19
5.2 Executing the Self-Test of the ARC HS36 Core .....	22
5.3 Executing the Self-Test of the ARC HS38x2 Core .....	23
5.4 Restoring the Self-Tests in the SPI Flash.....	25
6 Hardware Functional Description .....	27
6.1 Board Overview.....	27
6.2 Board Interface Overview.....	30
6.2.1 Power Supply Connector .....	30
6.2.2 HapsTrak II Connectors (Bottom).....	30
6.2.3 HapsTrak II Connectors (Top).....	31
6.2.4 Mictor Connectors .....	31
6.3 Jumpers .....	31
6.4 LEDs .....	32
6.5 Pushbutton .....	33
6.6 Seven-Segment Displays .....	34
6.7 AXC003 Processor FPGA Overview .....	35
6.7.1 Main Features of the ARC Cores .....	35
6.7.2 PAE.....	37
6.7.3 I/O Coherency.....	38
6.7.4 Interrupts.....	39
6.7.5 Clock .....	46
6.7.6 Reset.....	48

6.7.7	Debug .....	49
6.7.8	Control Registers .....	50
6.7.9	GPIO Registers .....	52
6.7.10	DIP Switches for FPGA Image Selection .....	55
6.7.11	ARC HS34 Emulation.....	56
6.8	Memories on the AXC003 CPU Card .....	57
6.9	Power Supply .....	58
6.10	Audio Support .....	59
6.11	Usage of ARC SDP Mainboard Resources .....	59
6.11.1	Usage of the Mainboard DIP Switches.....	59
6.11.2	Usage of the Mainboard Pushbuttons .....	62
6.11.3	Usage of the Mainboard LEDs .....	63
7	System Memory Map .....	65
7.1	System Memory Map After a Reset.....	65
7.2	System Memory Map After Pre-Bootloader Execution .....	65
7.3	Controlling the Memory Map .....	67
7.3.1	Setting Up the AXI Masters on the AXC003 CPU Card .....	67
7.3.2	Setting Up the AXI Masters on the ARC SDP Mainboard .....	68
7.3.3	Example Register Settings for the Default Memory Map.....	69
7.4	Memory Map of the Local Peripherals.....	71
8	Programmer's Reference .....	72
8.1	Supported Tools and Operating Systems.....	72
8.2	Boot Modes .....	72
8.2.1	Common Boot Modes .....	72
8.2.2	ARC HS36 Booting from ICCM0 .....	73
8.3	Pre-Boot.....	74
8.3.1	Pre-Boot Overview .....	74
8.4	Drivers.....	77
8.4.1	Drivers for Bare-Metal Applications.....	77
8.4.2	Drivers for MQX .....	78
8.5	Bare-Metal Package.....	78
8.5.1	Overview .....	78
8.5.2	Building Bare-Metal Applications Using the MetaWare IDE .....	79
8.5.3	Building Bare-Metal Applications Using gmake.....	83
8.5.4	Hardware Setup for Debugging.....	85
8.5.5	Running a Bare-Metal Application in the MetaWare IDE Debugger .....	87
8.5.6	Running a Bare-Metal Application in the MetaWare Debugger .....	90
8.5.7	Storing an Image in the SPI Flash and Running the Application .....	94
8.6	MQX Package .....	96
8.6.1	Overview .....	96
8.6.2	Building MQX Applications Using gmake .....	97
8.6.3	Hardware Setup for Debugging.....	98
8.6.4	Running MQX Applications in the MetaWare Debugger .....	98
8.7	Linux and U-Boot Packages.....	99
8.7.1	Overview .....	99
8.7.2	Hardware Setup for Debugging.....	100

8.7.3	Executing the Linux Image with U-Boot .....	100
8.8	ARCV2 Instruction Set: Usage Limitations.....	107
9	Software Interfaces .....	108
9.1	Clock-Generation Registers .....	108
9.1.1	TUNNEL PLL .....	108
9.1.2	ARC PLL .....	111
9.2	AXI Tunnel Address Decoder Registers.....	114
9.2.1	TUN_A_SLV0: AXI Tunnel Slave Select Register 0.....	114
9.2.2	TUN_A_SLV1: AXI Tunnel Slave Select Register 1.....	114
9.2.3	TUN_A_OFFSET0: AXI Tunnel Address Offset Register 0.....	115
9.2.4	TUN_A_OFFSET1: AXI Tunnel Address Offset Register 1.....	116
9.2.5	TUN_A_UPDATE: AXI Tunnel Update Register .....	116
9.3	ARC CPU Address Decoder Registers .....	117
9.3.1	CPU_A_SLV0: ARC CPU Slave Select Register 0 .....	117
9.3.2	CPU_A_SLV1: ARC CPU Slave Select Register 1 .....	117
9.3.3	CPU_A_OFFSET0: ARC CPU Address Offset Register 0.....	118
9.3.4	CPU_A_OFFSET1: ARC CPU Address Offset Register 1 .....	119
9.3.5	CPU_A_UPDATE: ARC CPU Update Register.....	119
9.4	ARC RTT Address Decoder Registers .....	120
9.4.1	RTT_A_SLV0: ARC RTT Slave Select Register 0 .....	120
9.4.2	RTT_A_SLV1: ARC RTT Slave Select Register 1 .....	120
9.4.3	RTT_A_OFFSET0: ARC RTT Address Offset Register 0 .....	121
9.4.4	RTT_A_OFFSET1: ARC RTT Address Offset Register 1 .....	122
9.4.5	RTT_A_UPDATE: ARC RTT Update Register .....	122
9.5	PAE Registers.....	123
9.5.1	PAE: PAE Register .....	123
9.5.2	PAE_UPDATE: PAE Update Register .....	123
9.6	CPU Start Registers .....	124
9.6.1	CPU_START: ARC CPU Start Register.....	124
9.6.2	CPU_0_ENTRY: ARC CPU-0 Kernel Entry Point Register .....	125
9.6.3	CPU_1_ENTRY: ARC CPU-1 Kernel Entry Point Register .....	125
9.6.4	CPU_BOOT: Boot Register.....	125
9.7	AXI Tunnel Registers .....	126
9.7.1	TUN_CTRL Register .....	126
9.7.2	TUN_STAT Register .....	126
9.8	GPIO Registers .....	127
9.8.1	GPIO_SWPORTA_DR: GPIO Port A Output Register .....	127
9.8.2	GPIO_SWPORTB_DR: GPIO Port B Output Register .....	128
9.8.3	GPIO_EXT_PORTA: GPIO Port A Input Register.....	129
9.8.4	GPIO_EXT_PORTB: GPIO Port B Input Register.....	130
Appendix A.....		132
A.1	Mounting the AXC003 CPU Card.....	132
Appendix B.....		134
B.1	Installing and Configuring PuTTY.....	134
Appendix C .....		137
C.1	Detailed Core Configurations .....	137

---

Glossary and References .....	144
Glossary .....	144
References .....	145

## List of Figures

Figure 1	DesignWare ARC AXS103 Software Development Platform	10
Figure 2	DesignWare ARC AXC003 CPU Card	11
Figure 3	Location of the ARC SDP Mainboard Power Supply and Power Switch	13
Figure 4	ARC SDP Mainboard Status LEDs After Power-On	13
Figure 5	AXC003 CPU Card Power-Control LEDs After Power-On	14
Figure 6	Default Jumper Settings on the AXC003 CPU Card	15
Figure 7	Default Settings of the DIP Switches on the ARC SDP Mainboard	16
Figure 8	Location of the CPU LEDs on the ARC SDP Mainboard	21
Figure 9	Location of the LED121x on the AXC003 CPU Card	21
Figure 10	Location of the ARC SDP Mainboard's Power Supply and Power Switch	22
Figure 11	Location of the CPU Start Button SW2504 for the ARC HS36 Core	22
Figure 12	ARC HS36 Self-Test	23
Figure 13	Location of the RESET Button on the ARC SDP Mainboard	23
Figure 14	Location of the ARC SDP Mainboard's Power Supply and Power Switch	24
Figure 15	Location of the CPU Start Button SW2504 for the ARC HS38x2 Core	24
Figure 16	Screen-Shot of ARC HS38x2 Self-Test	25
Figure 17	Location of the RESET Button on the ARC SDP Mainboard	25
Figure 18	Hardware Block Diagram (HS36)	28
Figure 19	Hardware Block Diagram (HS38x2)	29
Figure 20	Location of the Power Control LEDs on the AXC003 CPU Card	32
Figure 21	Location of the User LEDs on the AXC003 CPU Card	32
Figure 22	Location of the Pushbutton on the AXC003 CPU Card	34
Figure 23	AXC003 Memory Map	37
Figure 24	AXC003 I/O Coherency Architecture	38
Figure 25	I/O Coherency and PAE	39
Figure 26	HS36 Interrupt Architecture	41
Figure 27	HS38x2 Interrupt Architecture	42
Figure 28	Clock Architecture	47
Figure 29	Location of the RESET Button on the ARC SDP Mainboard	48
Figure 30	JTAG Daisy-Chain	49
Figure 31	Pinout of the Power Supply Connector (Bottom View)	59
Figure 32	Location of the Power Control LEDs on the AXC003 CPU Card	59
Figure 33	Function and Default Settings of the DIP Switches on the ARC SDP Mainboard	62
Figure 34	Location of the CPU Start Buttons on the ARC SDP Mainboard	63
Figure 35	Location of the CPU LEDs on the ARC SDP Mainboard	64
Figure 36	Default settings of the DIP Switches on the ARC SDP Mainboard	75
Figure 37	Pre-Boot Mechanism	76
Figure 38	MetaWare IDE - Select Workspace Directory	80
Figure 39	MetaWare IDE – Importing Existing Projects	80
Figure 40	MetaWare IDE - Set Active Build Configurations	81
Figure 41	MetaWare IDE – Build Results in Console Window	82
Figure 42	Build Script Options	83
Figure 43	Settings of the DIP Switches on the ARC SDP Mainboard for Using the Debugger	85
Figure 44	Location of the Debug Interfaces and the Corresponding Jumpers	86
Figure 45	Location of the ARC SDP Mainboard's Power Supply and Power Switch	86
Figure 46	Location of the CPU Start Buttons on the ARC SDP Mainboard	87

Figure 47	Creating a New Process .....	90
Figure 48	Debugger options – Command-Line Options.....	91
Figure 49	Debugger Options – Target Selection.....	92
Figure 50	Specifying a Path to the .elf File.....	92
Figure 51	Debugger Status .....	93
Figure 52	HyperTerminal Output.....	94
Figure 53	DIP Switch Settings for Autonomous Code Execution on the ARC Core .....	96
Figure 54	Default Settings of the DIP Switches on the ARC SDP Mainboard. ....	133
Figure 55	Identification of COM Port .....	135
Figure 56	PuTTY Configuration.....	136



## List of Tables

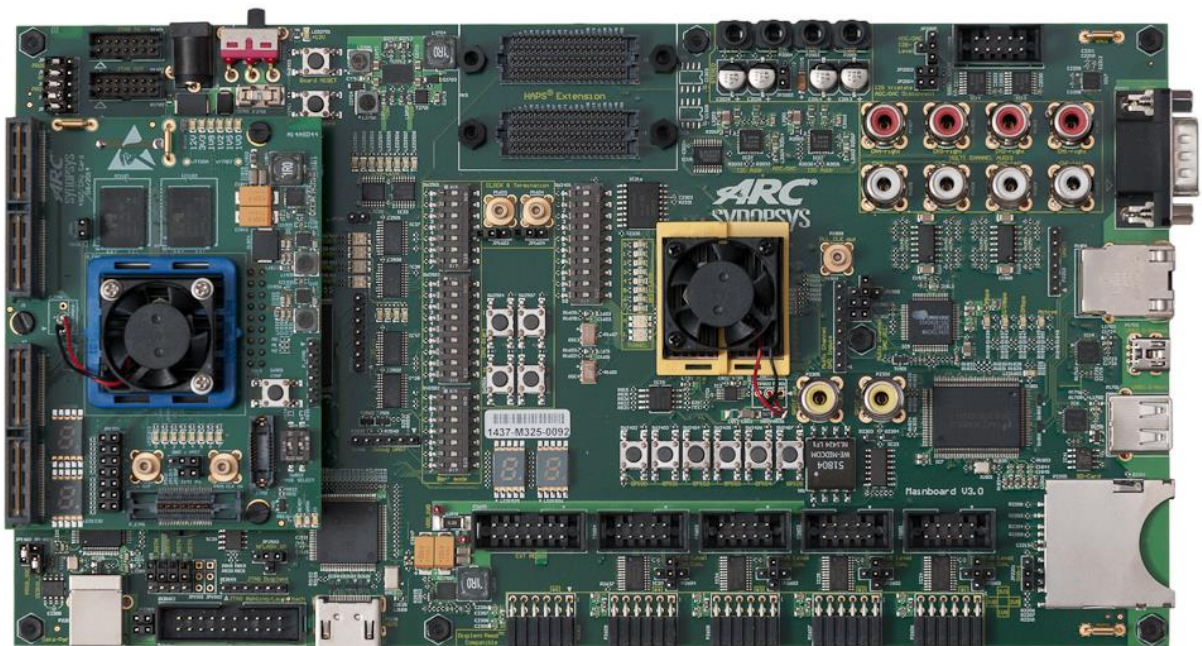
Table 1	Self-Test Start Buttons.....	20
Table 2	Characters on the Seven-Segment Display During the Self-Test.....	21
Table 3	Jumper Functionality.....	31
Table 4	LED Control Bits.....	33
Table 5	Control Bits of the Seven-Segment Displays.....	35
Table 6	Main Features of the ARC Cores.....	36
Table 7	Interrupt Mapping for ARC HS36.....	43
Table 8	Interrupt Mapping for ARC HS38.....	44
Table 9	Mainboard ICTL Interrupt Mapping.....	45
Table 10	Clock Frequencies.....	48
Table 11	JTAG ID Codes.....	49
Table 12	Control Register Memory Map.....	50
Table 13	GPIO Register Memory Map.....	52
Table 14	GPIO Port A Output Register Bit Function (SWPORTA_DR).....	52
Table 15	GPIO port A Input Register Function (EXT_PORTA).....	53
Table 16	GPIO port B Output Register Function (SWPORTB_DR).....	54
Table 17	GPIO port B input register function (EXT_PORTB).....	55
Table 18	Memory mapping for ARC HS36.....	56
Table 19	Memory mapping for HS34 Emulation.....	57
Table 20	Pinout of the Power-Supply Connector.....	58
Table 21	ARC Core Boot Configuration (Mainboard DIP Switch SW2501).....	60
Table 22	Multicore Configuration (Mainboard DIP Switch SW2503).....	61
Table 23	Usage of the CPU Start Buttons of the ARC SDP Mainboard.....	63
Table 24	Control Bits of the CPU LEDs on the ARC SDP Mainboard.....	64
Table 25	ARC CPU Memory Map After Pre-Bootloader Execution.....	65
Table 26	AXI Tunnel Memory Map After Pre-Bootloader Execution (ARC HS34 / HS36).....	67
Table 27	AXI Tunnel Memory Map After Pre-Bootloader Execution (ARC HS38).....	67
Table 28	AXC003 CPU Card Target Slaves.....	68
Table 29	ARC SDP Mainboard Target Slaves.....	68
Table 30	ARC CPU Memory Map Pre-Boot Programming on the AXC003 CPU Card.....	69
Table 31	Memory Map Pre-Boot Programming for All Masters on the ARC SDP Mainboard.....	70
Table 32	Peripheral Memory Map.....	71
Table 33	Meaning of the Left Character of the Seven-Segment Display.....	76
Table 34	Meaning of the Right Character of the Seven-Segment Display.....	77
Table 35	baremetal Folder Contents.....	78
Table 36	Build Options.....	81
Table 37	Command Line Options for build.bat.....	83
Table 38	CPU Start Buttons and Display Values for Running Applications in the Debugger.....	87
Table 39	Property Arguments for Selecting the CPU Core in the Debugger.....	91
Table 40	MQX folder Contents.....	97
Table 41	GPIO port A Output Register (GPIO_SWPORTA_DR).....	127
Table 42	GPIO port B output Register (GPIO_SWPORTB_DR).....	128
Table 43	GPIO Port A Input Register (GPIO_EXT_PORTA).....	129
Table 44	GPIO Port B Input Register (GPIO_EXT_PORTB).....	130

## 1.1 DesignWare ARC AXS103 Software Development Platform

The DesignWare ARC AXS103 Software Development Platform package contains the following items:

- DesignWare ARC AXC003 CPU Card mounted on ARC SDP Mainboard
- 100-240V AC power adapter (including power cables for U.S., UK, and EU outlets)
- USB cable
- Pen-sized plastic dipstick for actuating DIP switches

Figure 1 *DesignWare ARC AXS103 Software Development Platform*



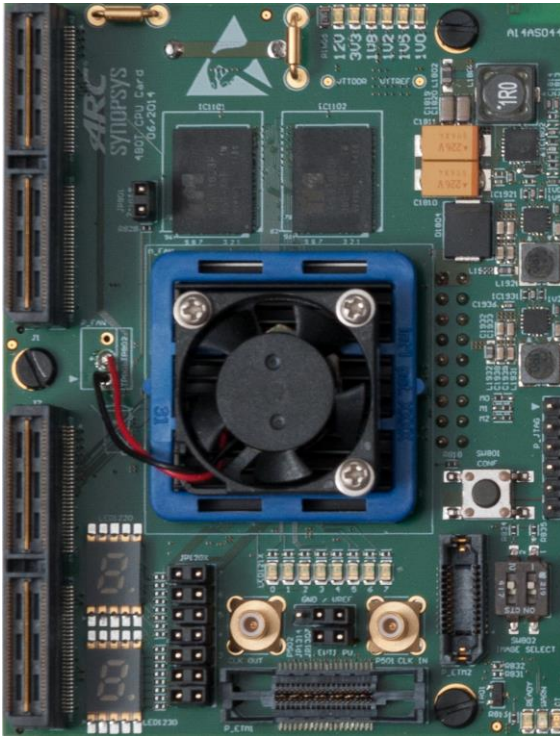
### Warning

The AXC003 CPU Card and the ARC SDP Mainboard contain static-sensitive devices.

## 1.2 DesignWare ARC AX003 CPU Card (Standalone)

The DesignWare ARC AX003 CPU Card package contains the DesignWare ARC AX003 CPU Card printed circuit board.

Figure 2 *DesignWare ARC AX003 CPU Card*



### Warning

The AX003 CPU Card contains static-sensitive devices.

# 2

## Getting Started

---

*This chapter contains a step-by-step guide for installing the software package for the AXS103 Software Development Platform, connecting the ARC SDP Mainboard, and performing a self-test.*

---

### 2.1 Mounting the CPU Card

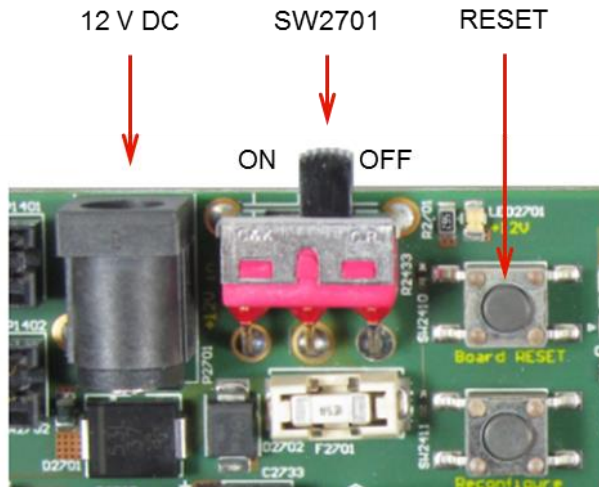
If you purchased a standalone AXC003 CPU Card, see the instructions in [Appendix A](#) for mounting the CPU Card on the ARC SDP Mainboard to obtain a complete AXS103 Software Development Platform.

### 2.2 Performing a Self-Test

Follow these steps to get the AXS103 Software Development Platform up and running and to perform a self-test.

1. Download and unzip the `axs103_software_<version>.zip` file from the ARC SDP download webpage [\[4\]](#).
2. Install the USB-JTAG and USB-UART drivers (Digilent Adept tool) according to the instructions provided in the *ARC SDP Mainboard User Guide* [\[5\]](#).
3. Connect the ARC SDP Mainboard to your PC by connecting the USB cable to the USB data port of the Mainboard and the PC.
4. Connect the power supply to the ARC SDP Mainboard and switch on the Mainboard.

Figure 3 Location of the ARC SDP Mainboard Power Supply and Power Switch

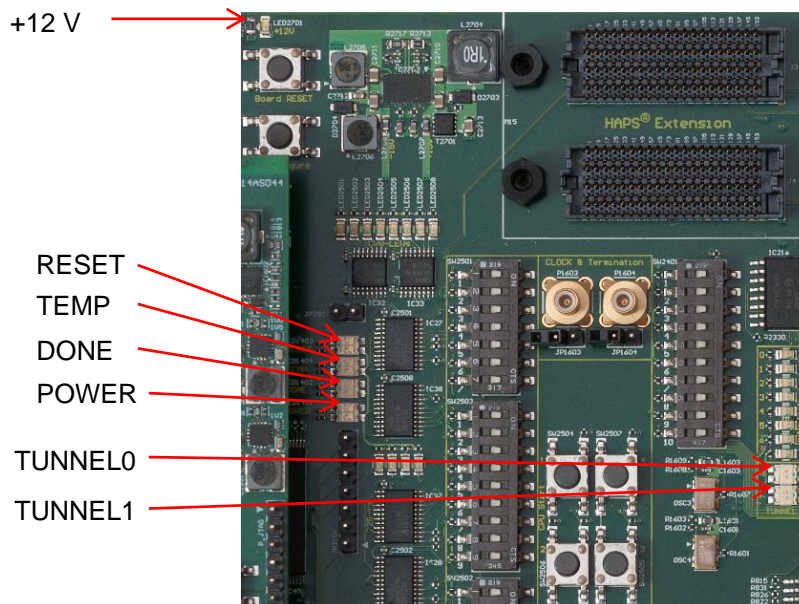


5. Install PuTTY on your computer as described in [Appendix B](#).

The FPGA on the ARC SDP Mainboard is now configured automatically and the Mainboard executes the reset sequence. The status LEDs `DONE`, `RESET`, `TUNNEL0`, and `TUNNEL1` on the Mainboard shine red during startup.

6. Wait until all status LEDs except `TUNNEL1` shine green. This may take several seconds. The LED `TUNNEL1` continues to shine red.

Figure 4 ARC SDP Mainboard Status LEDs After Power-On



7. Check that the six power LEDs on the AXC003 CPU Card are all on, shining green.

Figure 5 AXC003 CPU Card Power-Control LEDs After Power-On



8. Perform the self-test for one or both CPU cores as described in the “[Self-Tests](#)” section on page 19.

For the next steps, see “[Bare-Metal Package](#)” on page 78.

**Note**

The AXS103 Software Development Platform is supplied with the ARC SDP Mainboard version 3.x. The AXC003 CPU Card is also compatible with ARC SDP Mainboard version 2.x. Follow the instructions provided with the firmware package `axs103_firmware_<version>.zip` to configure the ARC SDP Mainboard.

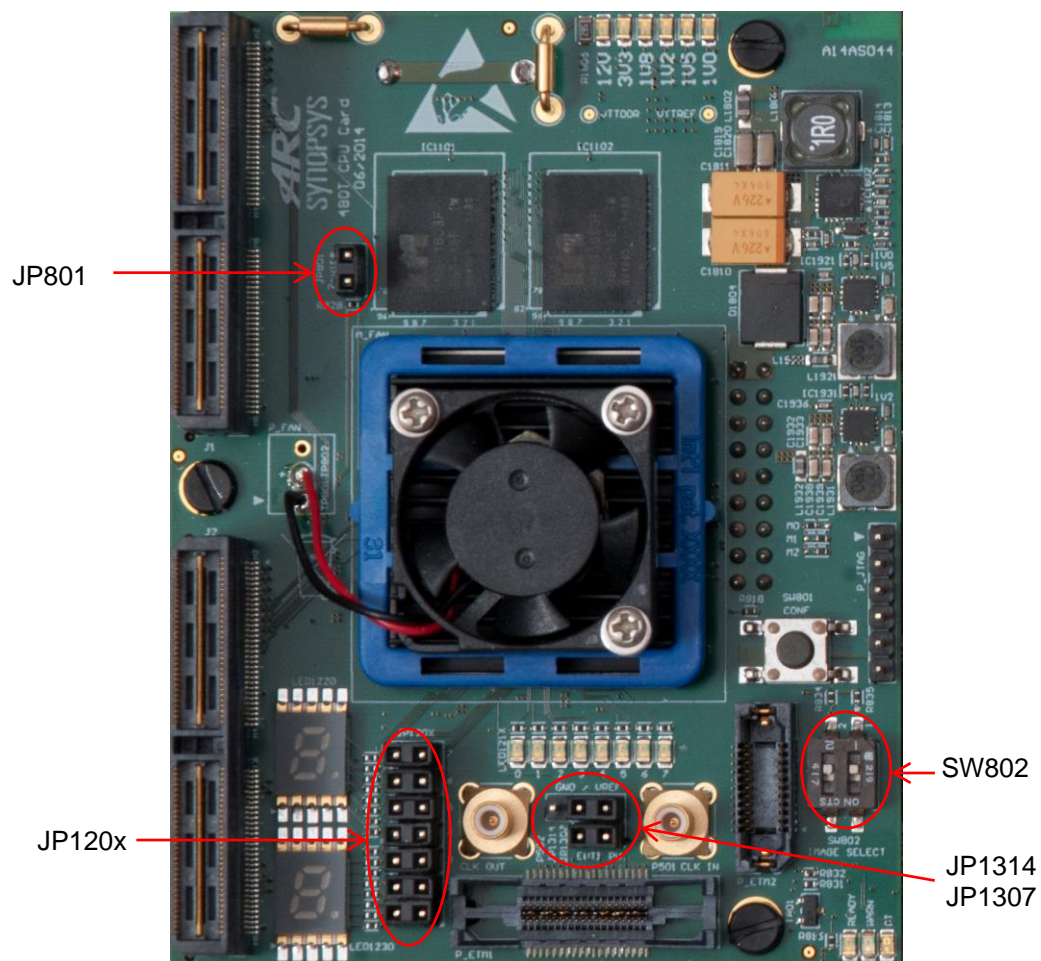
## Default Board Settings

This section describes the factory default settings of the jumpers on the AXC003 CPU Card and the default boot-mode settings for the cores on the AXC003 CPU Card, which can be selected by DIP switches on the ARC SDP Mainboard.

### 3.1 Default Jumper Settings on the AXC003 CPU Card

The jumpers on the AXC003 CPU Card must be set according to Figure 6.

Figure 6 Default Jumper Settings on the AXC003 CPU Card



### 3.2 Default Boot-Mode Settings on the ARC SDP Mainboard

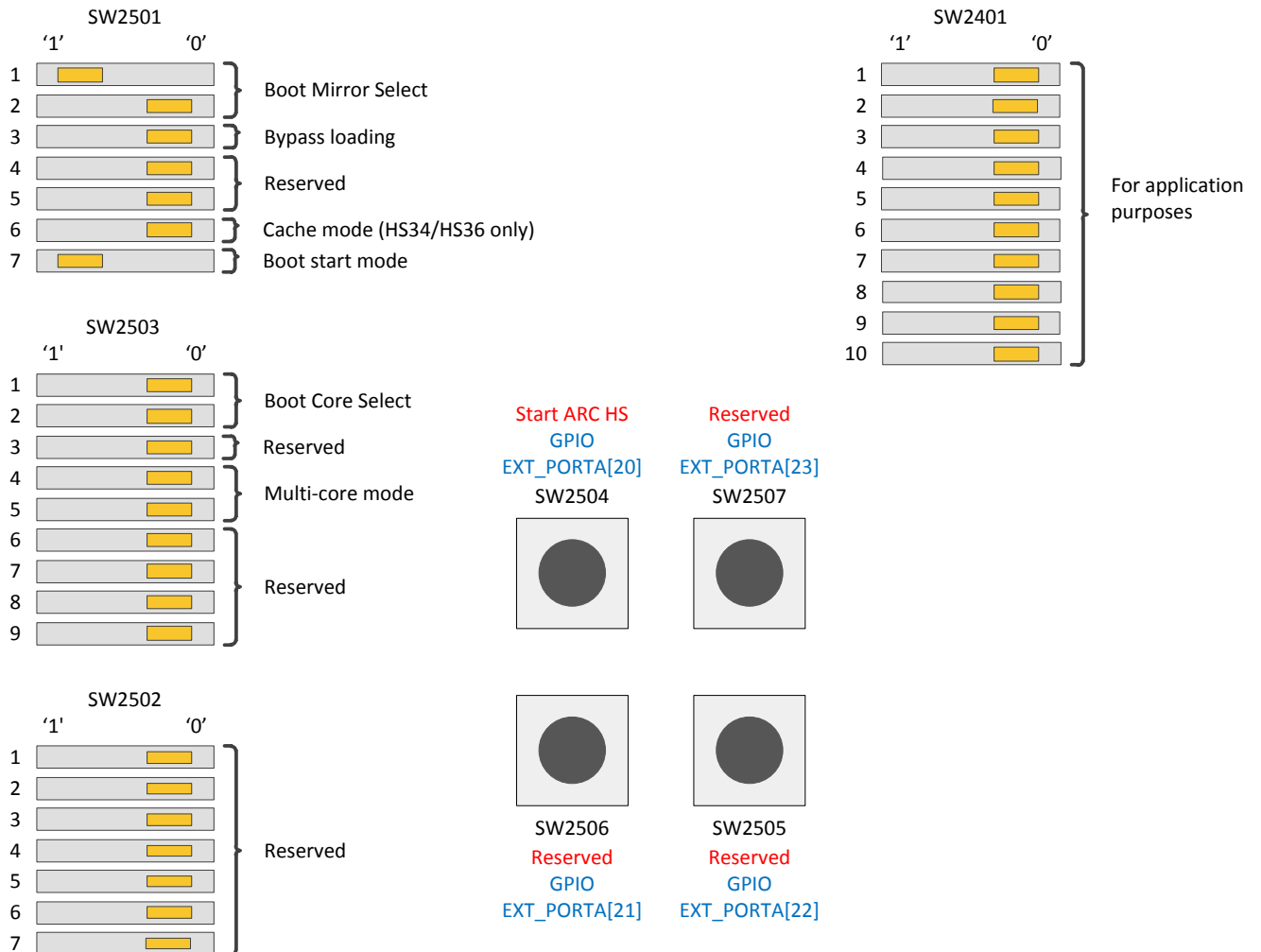
The DIP switches on the ARC SDP Mainboard are set according to Figure 7.

All cores are configured to boot from the internal ROM and automatically start the pre-bootloader application after reset. The pre-bootloader handles initialization of the system and sets the CPU in the halt state.

Application loading from the SPI flash is bypassed.

If you want to start an ARC core manually, set bit 7 (boot start mode) to the right-side position. In this case, the CPU delays code execution after reset until the corresponding CPU Start button on the ARC SDP Mainboard is pressed.

Figure 7 Default Settings of the DIP Switches on the ARC SDP Mainboard





This chapter provides instructions for selecting a CPU core.

## 4.1 Supported CPU Cores



Following CPU cores can be used with the *AXS103 Software Development Platform*:

- ARC HS36
- ARC HS36 in ARC HS34 emulation mode
- ARC HS38 single-core mode, core 0
- ARC HS38 single-core mode, core 1
- Dual-core ARC HS38x2

## 4.2 Core Selection

To select a CPU core, use the DIP switches on the AXC003 CPU Card and on the ARC SDP Mainboard. The configuration is selected in the reset state.

SW802 on the AXC003 CPU Card defines the FPGA image to be loaded at power-on reset. The following bits of SW802 define the FPGA image that is selected:

-  00 – FPGA image for ARC HS36 CPU
-  01 – FPGA image for ARC HS38x2 CPU

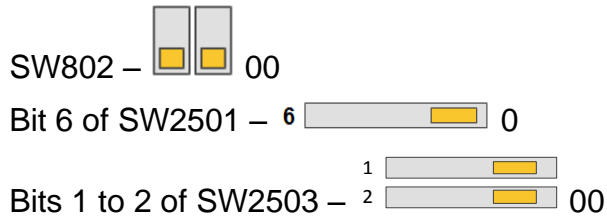
DIP switches SW2501 and SW2503 on the ARC SDP Mainboard select the core used within a particular FPGA image:

- Bit 6 of SW2501 defines whether the data cache and instruction cache are bypassed, that is, it selects ARC HS36 or ARC HS34 emulation. For more details, see [“ARC HS34 Emulation”](#) on page 56.
- Bits 1 to 3 of SW2503 select a dual-core configuration ARC HS38x2.

For the detailed description of DIP switches used for configuration, see [“Usage of the Mainboard DIP Switches”](#) on page 59.

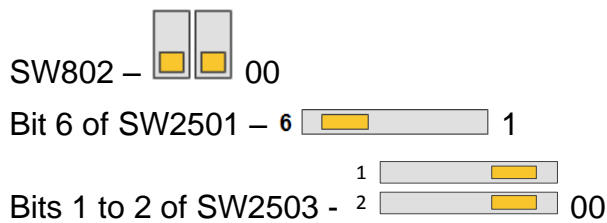
### 4.2.1 ARC HS36 CPU

To select ARC HS36 use following settings:



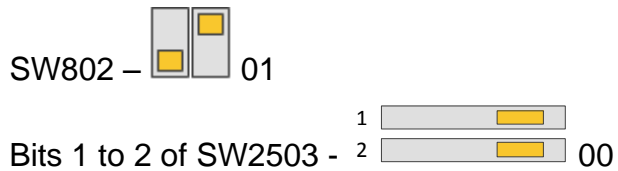
### 4.2.2 ARC HS34 CPU

To select ARC HS34, use following settings:



### 4.2.3 ARC HS38 Core 0

To select ARC HS38 core 0 use following settings:



### 4.2.4 ARC HS38 Core 1

To select ARC HS38 core 1 use following settings:



*This chapter provides an overview of the self-tests and includes detailed instructions for executing the self-test on each individual CPU core. The expected behavior during the self-test is described as well.*

### 5.1 Self-Test Overview

At the time of shipment, the SPI Flash on the ARC SDP Mainboard contains a self-test for each CPU core.

Descriptions in this section are based on the following assumptions:

- The SPI Flash contains self-tests
- The DIP switches on the ARC SDP Mainboard and on the AXC003 CPU Card are set as described in “[Default Board Settings](#)” on page 15.
- The steps described in “[Getting Started](#)” on page 12 have been performed.



#### Note

If you have programmed other applications in the SPI Flash, you can restore the self-test as described in the “[Restoring the Self-Tests in the SPI Flash](#)” on page 25.

With the default board settings, the CPU core runs autonomously and application loading from SPI flash is bypassed.

To perform a self-test, use the following settings of DIP switch SW2501:

Bit 3 “Bypass loading”:



The pre-bootloader looks for the appropriate application in the SPI flash and runs it if found.

Bit 7 “Boot start mode”:



Start ARC core manually.

Run the self-test for a particular CPU by pushing the corresponding CPU Start button listed in Table 1.

Table 1 Self-Test Start Buttons

CPU Core	CPU Start Button	Location
ARC HS36 ARC HS38x2	SW2504	
Reserved	SW2506	
Reserved	SW2505	
Reserved	SW2507	

The self-test accesses the peripherals of the peripheral subsystem that is implemented in the FPGA on the ARC SDP Mainboard. It displays information on the bitfile version and the available peripherals in a debug console on the PC. Additionally, the current CPU core speed is measured and displayed in the debug console. For a quick start, use a hyperterminal, such as PuTTY, as a debug console (see “[Getting Started](#)”).

Next, the self-test enters an infinite loop, which creates walking patterns on the CPU-LEDs on the ARC SDP Mainboard and LED121x on the AXC003 CPU Card as follows:

The CPU-LEDs (LED2501, LED2502, LED2503, LED2504, LED2505, LED2506, LED2507 and LED2508) on the ARC SDP Mainboard display a walking pattern with one of the LEDs switched ON.

The eight LED121x LEDs on the AXC003 CPU Card show a walking pattern with one of the LEDs switched OFF.

The figures on page 21 show the locations of the LEDs.

Figure 8 Location of the CPU LEDs on the ARC SDP Mainboard

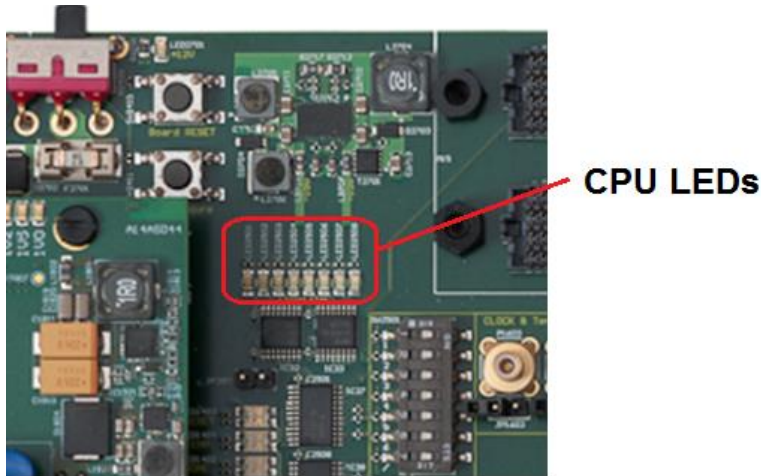
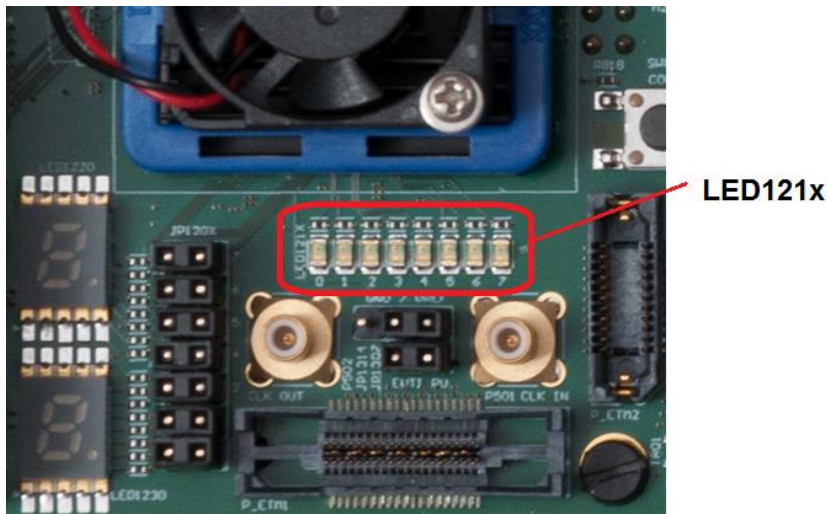


Figure 9 Location of the LED121x on the AXC003 CPU Card



The seven-segment displays on the ARC SDP Mainboard show the characters listed in Table 2. The number indicates the ARC core that is currently running.

Table 2 Characters on the Seven-Segment Display During the Self-Test

ARC Core	Characters
ARC HS36	10
ARC HS34	20
ARC HS38x2: core 0	30
ARC HS38x2: core 1	40

The seven-segment displays on the AXC003 CPU Card are not used by the self-test.

The expected results for each test are described in [Executing the Self-Test of the ARC HS36 Core](#) and Executing the Self-Test of the ARC HS38x2 Core on page 23.

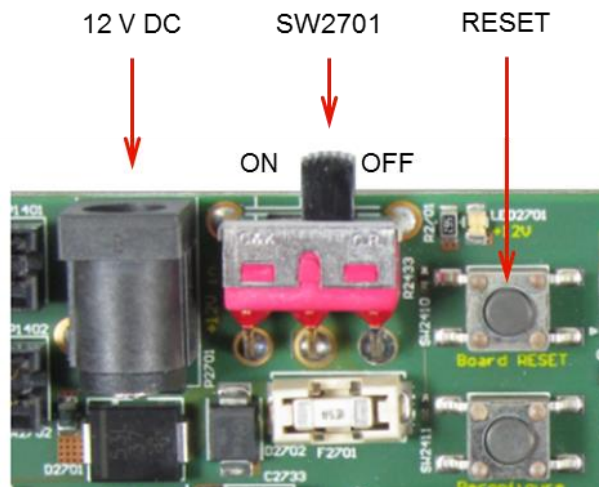
Press the RESET button to stop execution and to reset the seven-segment displays to their initial value.

## 5.2 Executing the Self-Test of the ARC HS36 Core

Follow the steps described below to perform the self-test of the ARC HS36 core:

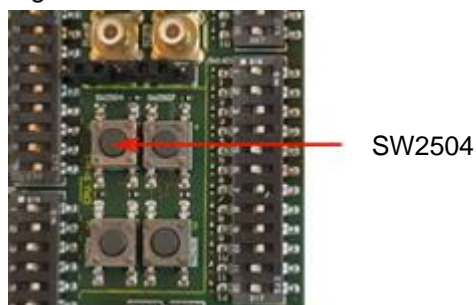
1. Connect the ARC SDP Mainboard to your PC using the USB cable, which must be connected to the USB data port of the Mainboard.
2. Connect the power supply to the Mainboard and switch ON the Mainboard or press the RESET button.

Figure 10 Location of the ARC SDP Mainboard's Power Supply and Power Switch



3. Launch PuTTY on your computer
4. Push the CPU Start button SW2504 on the Mainboard.

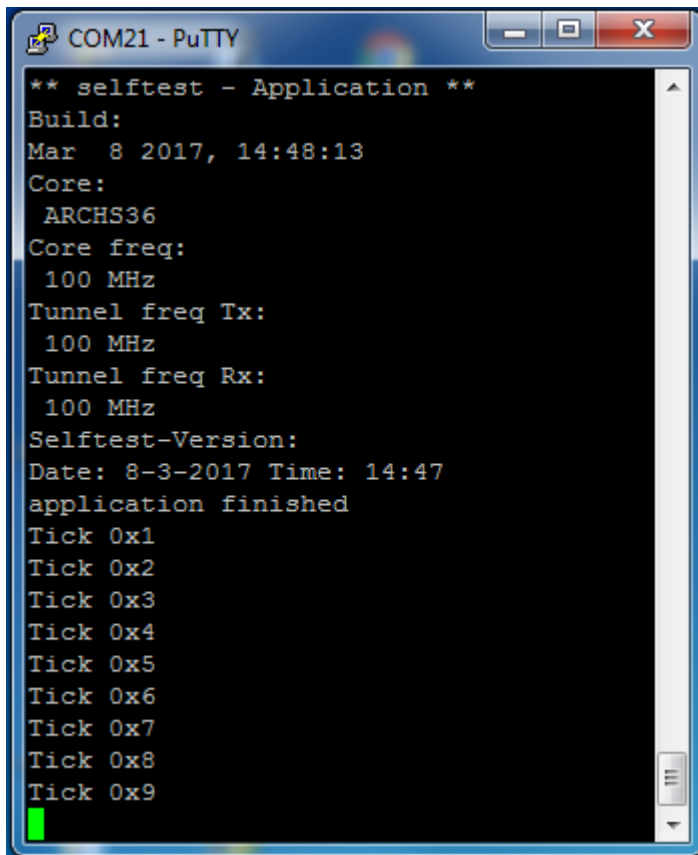
Figure 11 Location of the CPU Start Button SW2504 for the ARC HS36 Core



5. Check that the seven-segment displays 10.

6. Observe the output in the console, which should look similar to the console output shown in Figure 12.

Figure 12 ARC HS36 Self-Test



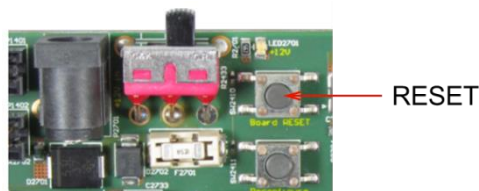
```

** selftest - Application **
Build:
Mar 8 2017, 14:48:13
Core:
ARCHS36
Core freq:
100 MHz
Tunnel freq Tx:
100 MHz
Tunnel freq Rx:
100 MHz
Selftest-Version:
Date: 8-3-2017 Time: 14:47
application finished
Tick 0x1
Tick 0x2
Tick 0x3
Tick 0x4
Tick 0x5
Tick 0x6
Tick 0x7
Tick 0x8
Tick 0x9

```

7. Observe the walking patterns shown by the CPU LEDs and the LED0121x LEDs.
8. Push the reset button on the ARC SDP Mainboard to stop the test.

Figure 13 Location of the RESET Button on the ARC SDP Mainboard

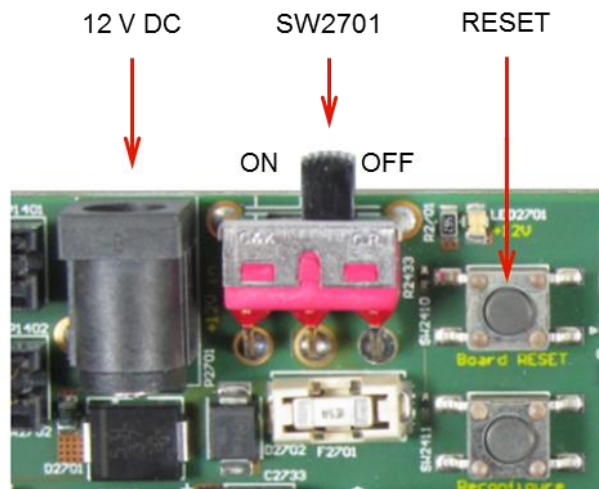


## 5.3 Executing the Self-Test of the ARC HS38x2 Core

Follow the steps described below to perform the self-test of the ARC HS38x2 core:

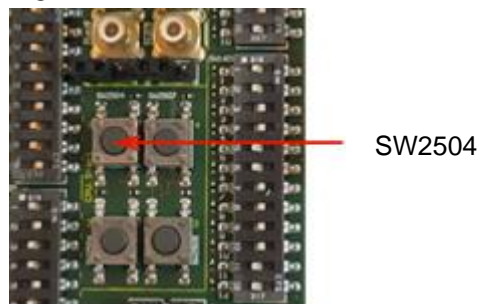
1. Connect the ARC SDP Mainboard to your PC using the USB cable, which must be connected to the USB data port of the Mainboard.
2. Connect the power supply to the Mainboard and switch ON the Mainboard or press the RESET button.

Figure 14 Location of the ARC SDP Mainboard's Power Supply and Power Switch



3. Launch PuTTY on your computer
4. Push the CPU Start button SW2504 on the ARC SDP Mainboard.

Figure 15 Location of the CPU Start Button SW2504 for the ARC HS38x2 Core



5. Check that the seven-segment displays 30.



6. Observe the output in the console, which should be similar to the console output shown in Figure 16.

Figure 16 Screen-Shot of ARC HS38x2 Self-Test

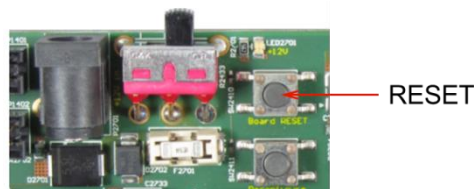
```

** selftest - Application **
Build:
Mar 8 2017, 14:48:17
Core:
ARCHS38[0]
Core freq:
100 MHz
Tunnel freq Tx:
100 MHz
Tunnel freq Rx:
100 MHz
Selftest-Version:
Date: 8-3-2017 Time: 14:47
application finished
Tick 0x1
Tick 0x2
Tick 0x3
Tick 0x4
Tick 0x5
Tick 0x6
Tick 0x7
Tick 0x8
Tick 0x9

```

7. Observe the walking patterns shown by the CPU LEDs and the LED0121x LEDs.
8. Push the reset button on the ARC SDP Mainboard to stop the test.

Figure 17 Location of the RESET Button on the ARC SDP Mainboard



## 5.4 Restoring the Self-Tests in the SPI Flash

If you store your own applications in the SPI Flash on the ARC SDP Mainboard, the factory-programmed self-tests are typically erased.

To restore the factory-programmed self-tests, follow the steps below:

1. If you have not done so earlier, download and unzip the `axs103_software_<version>.zip` file from the ARC SDP download webpage [4] and install the `axs_comm` tool as described in the *ARC SDP Mainboard User Guide* [5].
2. Download and unzip the `axs103_selftest_firmware_<version>.zip` file from the ARC SDP download webpage [4].
3. Navigate to the `/selftest_firmware` folder and double-click the `axs_comm_program_selftest.bat` batch file.

**Note**

Executing `axs_comm_program_selftest.bat` erases the content of the sectors 0 and 1 of the SPI Flash device on the ARC SDP Mainboard.

*This chapter provides information about the hardware used in the AXC003 CPU Card.*

---

## 6.1 Board Overview

The AXC003 CPU Card hardware is a daughter card for the ARC SDP Mainboard. It includes an FPGA with an SPI ROM image file that contains two FPGA images with ARC CPUs:

- Single core ARC HS36
- Dual-core ARC HS38x2

In addition to the ARC processors, the AXC003 CPU Card includes a 2 GByte DDR3 SDRAM, 256 KByte local SRAM physically implemented inside the FPGA, and internal ROM physically implemented as additional 32 KByte local SRAM inside the FPGA. The internal ROM area contains bootloader code and is not intended for application software.

Bare-metal applications are natively supported. OS and driver porting for operating systems are facilitated as well.

The AXS103 Software Development Platform is HAPS compliant and enables the use of high-frequency ARC CPU cores as a daughter card for HAPS, allowing full SoC prototyping.

[Figure 18](#) and [Figure 19](#) show the hardware block diagrams of the AXC003 CPU card.

Figure 18 Hardware Block Diagram (HS36)

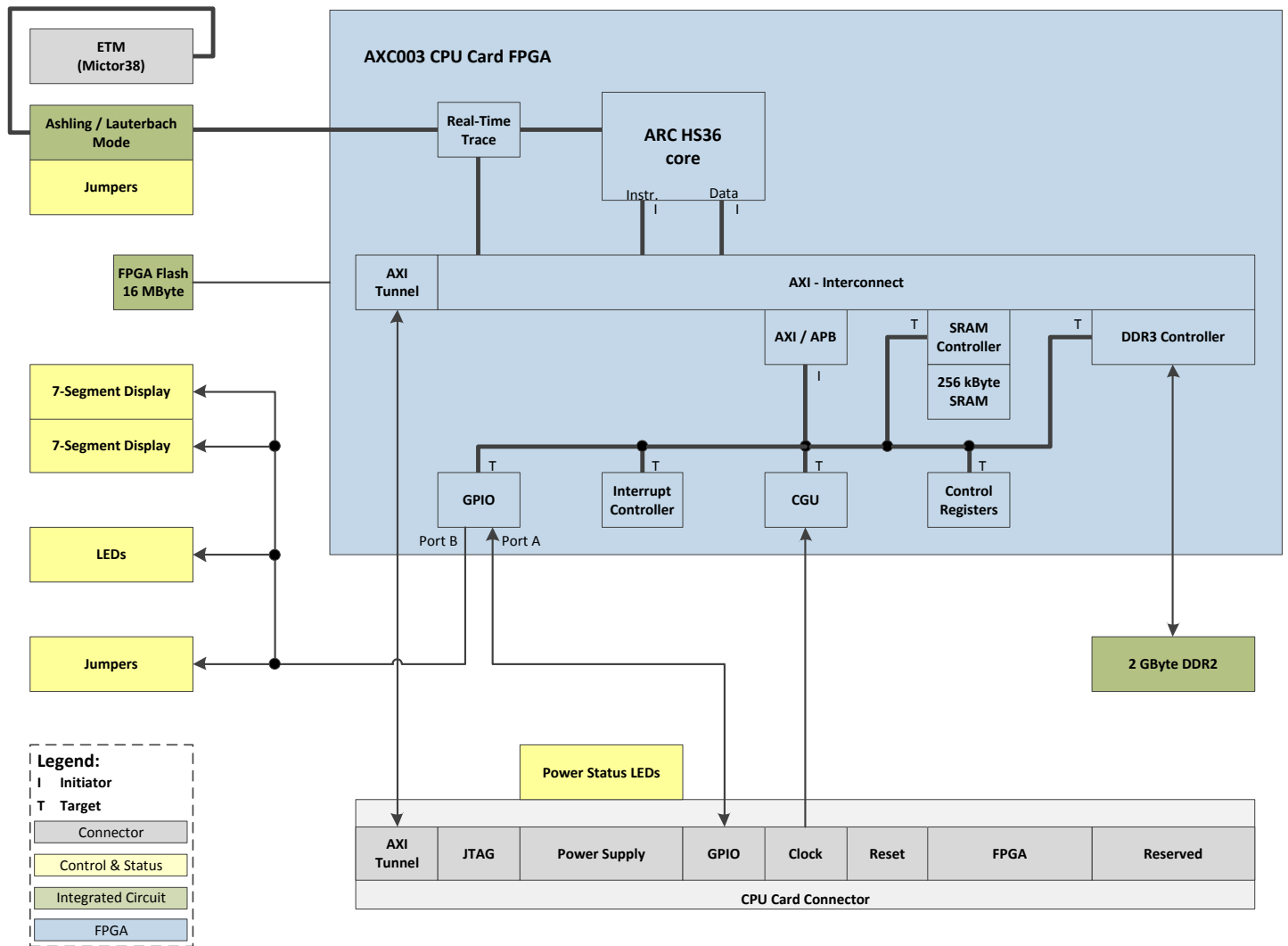
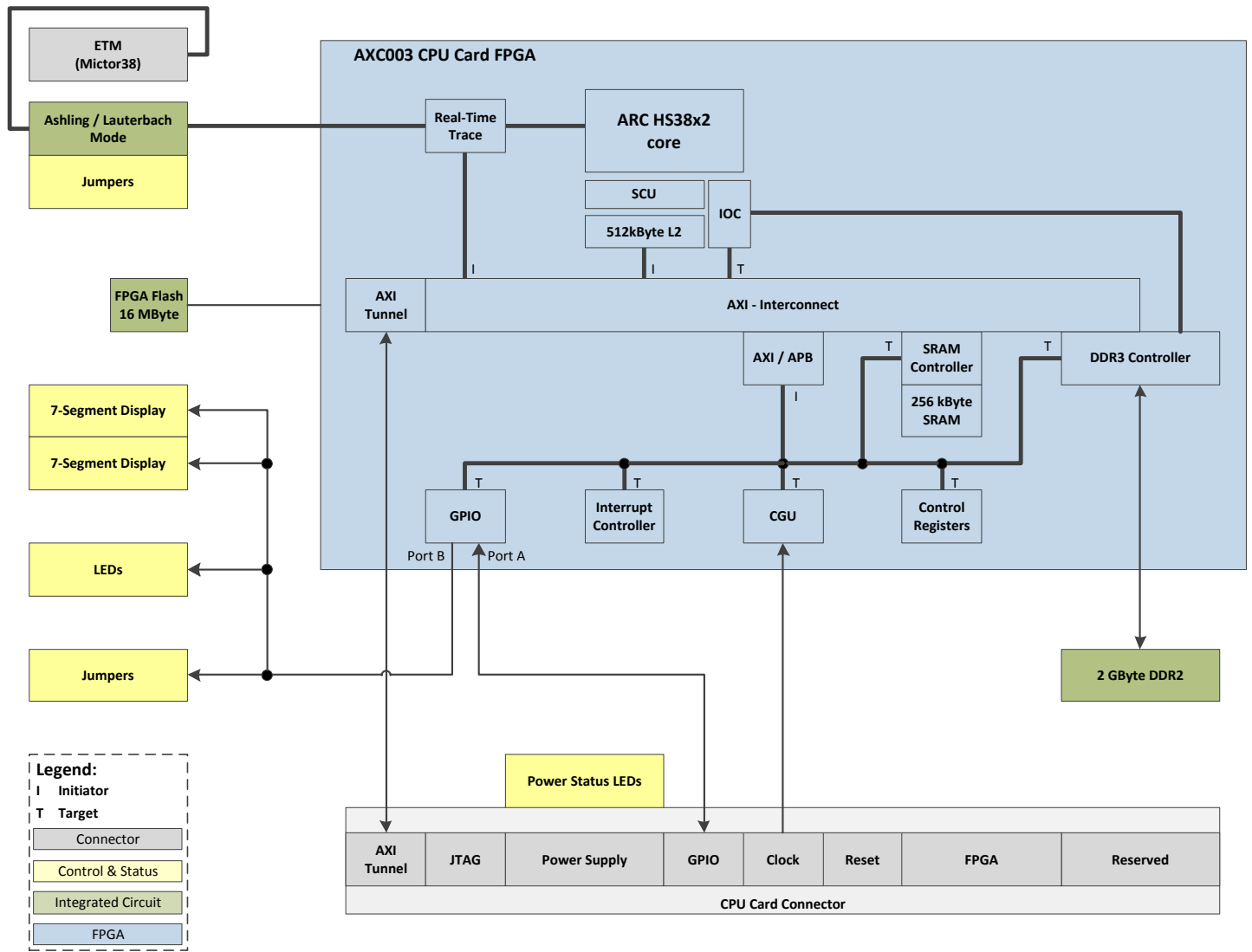


Figure 19 Hardware Block Diagram (HS38x2)



---

## 6.2 Board Interface Overview

The AXC003 CPU Card has two female HapsTrak II connectors and a female 18-pin power-supply connector on the bottom of the card. These connectors are provided for mounting the AXC003 CPU Card on the ARC SDP Mainboard.

The top of the AXC003 CPU Card has two male HapsTrak II connectors, which can be used to connect a HAPS logic-analyzer card for debugging.

The AXC003 CPU Card has two Mictor connectors, one of which can be used to connect an Ashling Ultra-XD debugger.

Additionally, the AXC003 CPU Card features two male SMB clock connectors, which are reserved for future extensions.

---

### 6.2.1 Power Supply Connector

Power is supplied to the AXC003 CPU Card by the ARC SDP Mainboard through the power-supply connector on the bottom of the AXC003 CPU Card board. The following voltage levels are provided: 1.1V, 1.8V, 2.5V, 3.3V and 12.0V.

See the [“Power Supply”](#) section for details.

---

### 6.2.2 HapsTrak II Connectors (Bottom)

The AXC003 CPU Card communicates with the ARC SDP Mainboard using two HapsTrak II connectors, which carry signal groups such as:

- AXI tunnel
- GPIO
- Clock
- Reset
- JTAG (ARC cores)
- FPGA (JTAG, control)

These connectors include 24 GPIO pins, which are connected to DIP switches on the ARC SDP Mainboard during reset. These switches are used to configure the boot mode of the ARC cores on the AXC003 CPU Card. At the end of reset the switch settings are latched inside the AXC003 Processor FPGA. After a reset these signals are connected to port A (bits [23:0]) of the GPIO peripheral of the AXC003 Processor FPGA.

Refer to the [“GPIO Registers”](#) section for details on the functionality.

### 6.2.3 HapsTrak II Connectors (Top)

A HAPS logic analyzer card can be connected to the HapsTrak II connectors at the top side of the AXC003 Processor FPGA. This setup can be used to observe the signals of the AXI tunnel between the AXC003 CPU Card and the ARC SDP Mainboard, for example.










### 6.2.4 Mictor Connectors

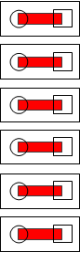
The AXC003 CPU Card has two Mictor connectors – P\_ETM1 and P\_ETM2. The P\_ETM1 connector can be used to connect an Ashling Ultra-XD debugger to ARC JTAG chain. The ARC JTAG connection is controlled by the JP1207 jumper setting; it is described in [Table 3](#) on page 31.

## 6.3 Jumpers

Table 3 describes the functionality of the jumpers on the AXC003 CPU Card. The default jumper settings are shown in [Figure 6](#) on page 15.

Table 3 Jumper Functionality

Jumper Name		Setting	Description
JP801			Normal operation
			Reserved
JP1314 JP1307			Normal operation
		Others	Reserved
SW802			ARC HS36
			ARC HS38x2
			Reserved
			Reserved
JP1207			ARC JTAG is connected to the AXS103 Mainboard
			ARC JTAG is connected to the Mictor P_ETM1 connector

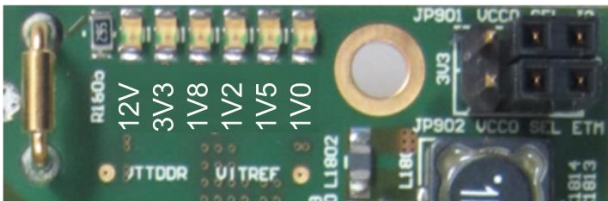
Jumper Name		Setting	Description
JP1206 JP1205 JP1204 JP1203 JP1202 JP1201			Normal Operation
		Others	Reserved

## 6.4 LEDs

The AXC003 CPU Card features six green power control LEDs and eight green LEDs for user applications.

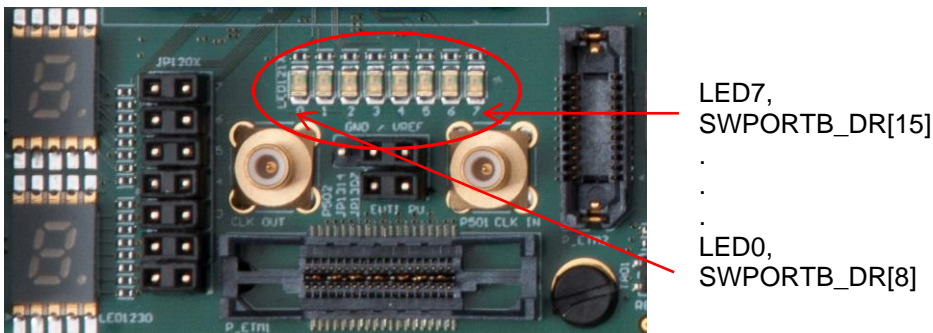
A power-control LED indicates when voltage is present for each supply voltage. In normal operation all six LEDs shine green. The location of these LEDs is shown in [Figure 20](#).

Figure 20 Location of the Power Control LEDs on the AXC003 CPU Card



The location of the user LEDs on the AXC003 CPU Card is shown in [Figure 21](#).

Figure 21 Location of the User LEDs on the AXC003 CPU Card



The LEDs are controlled by eight control bits within the SWPORTB\_DR register as listed in Table 4. The register is located at the offset 0x300C to the base address of the AXI2APB segment within the system memory map. By default the base address of the AXI2APB



segment is 0xF000\_0000 such that the default address of the register is 0xF000\_300C (see “[System Memory Map After Pre-Bootloader Execution](#)”).

Table 4 LED Control Bits

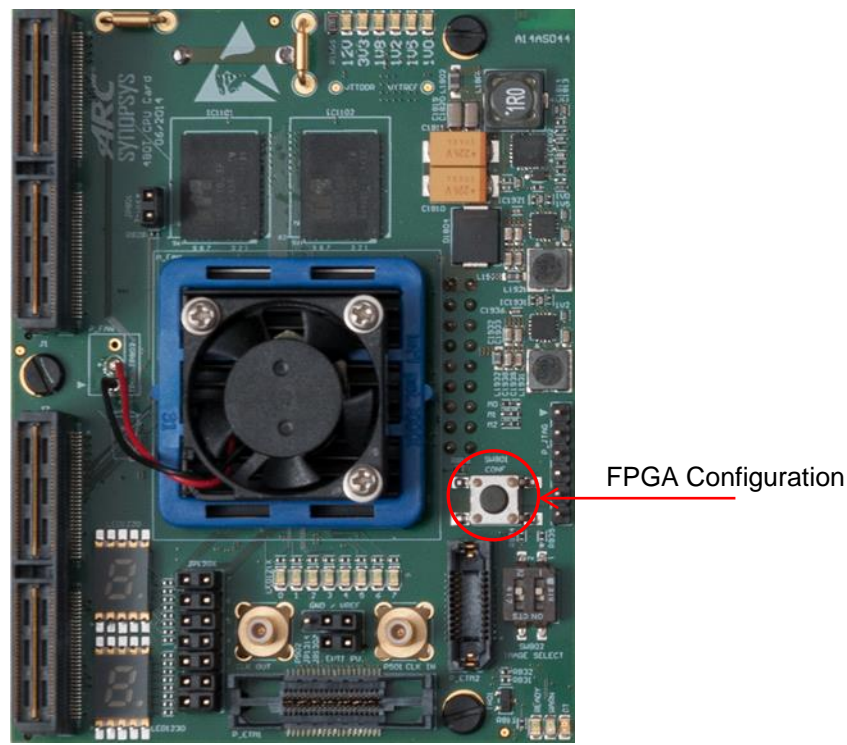
Control Bit	Description
SWPORTB_DR[8]	Connected to LED0 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
SWPORTB_DR[9]	Connected to LED1 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
SWPORTB_DR[10]	Connected to LED2 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
SWPORTB_DR[11]	Connected to LED3 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
SWPORTB_DR[12]	Connected to LED4 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
SWPORTB_DR[13]	Connected to LED5 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
SWPORTB_DR[14]	Connected to LED6 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
SWPORTB_DR[15]	Connected to LED7 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.

## 6.5 Pushbutton

Figure 22 shows the FPGA configuration button on the AXC003 CPU Card. This button can be used to re-load the FPGA configuration from the serial flash memory into the FPGA.

Normally you do not need to do this step manually. The FPGA configuration is initialized automatically at power-on.

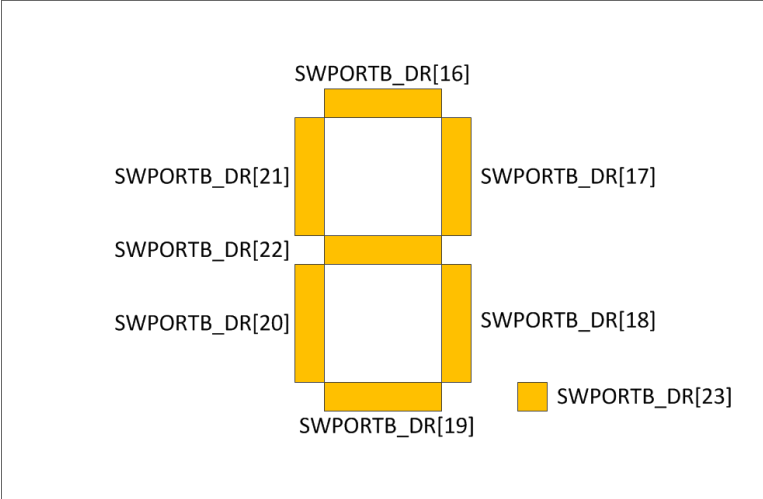
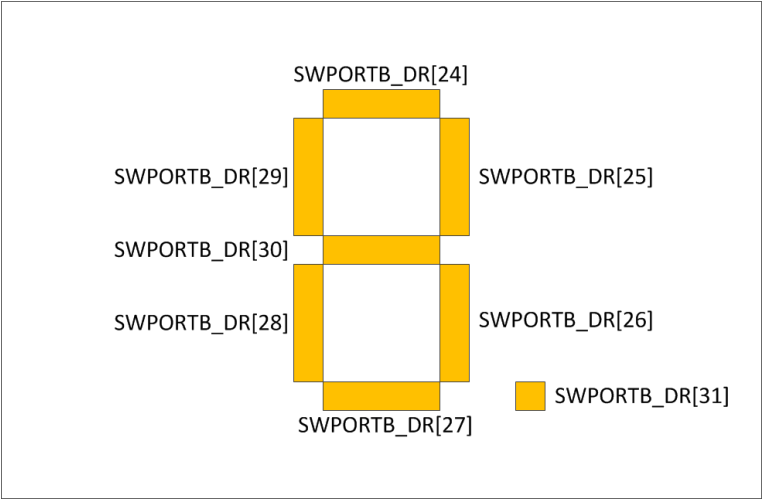
Figure 22 Location of the Pushbutton on the AXC003 CPU Card



## 6.6 Seven-Segment Displays

The AXC003 CPU Card features two seven-segment displays, which are controlled by a field of the `SWPORTB_DR` register as listed in [Table 5](#) on page 35. The register is located at the offset `0x300C` from the base address of the `AXI2APB` segment in the system memory map. By default the base address of the `AXI2APB` segment is `0xF000_0000`, so the default address of the register is `0xF000_300C` (see “[System Memory Map After Pre-Bootloader Execution](#)”).

Table 5 Control Bits of the Seven-Segment Displays

Control Bit	Description
SWPORTB_DR[23:16]	<p>Controls the upper seven-segment display. A segment of the display is ON when its control bit is set to 1.</p> 
SWPORTB_DR[31:24]	<p>Controls the lower seven-segment display. A segment of the display is ON when its control bit is set to 1.</p> 

## 6.7 AXC003 Processor FPGA Overview

### 6.7.1 Main Features of the ARC Cores

Table 6 on page 36 lists the main features of the core configurations and extensions available in the AXC003 Processor FPGA. See Appendix C for a complete list of the core configurations available in the AXC003 Processor FPGA.

Table 6 Main Features of the ARC Cores

Feature	ARC HS36	ARC HS38x2
Number of cores	1	2
I-Cache (bytes) Associativity Cache-line size	64K 2-way 32 bytes	64K 4-way 64 bytes
D-Cache (bytes) Cache-line size	64K 32 bytes	64K 64 bytes
Internal memory (bytes)	256K DCCM 256K ICCM0	No
Core interface	AXI (64-bit)	AXI (64-bit)
Core extensions	32x32 multiply Dual and quad MAC SIMD instructions Timer0 Timer1 Load/Store Unit LLOCK/SCOND instructions Branch prediction unit - branch cache: 512 - predictors: 8192 - return address stack: 4 - branch cache tag size: 4 - top of stack queue: 5 - instruction fetch buffer: 2 Radix-4 hardware divider Real-time clock Real-time trace	32x32 multiply Dual and quad MAC SIMD instructions Timer0 Timer1 Load/Store Unit LLOCK/SCOND instructions Branch prediction unit - branch cache: 512 - predictors: 8192 - return address stack: 4 - branch cache tag size: 4 - top of stack queue: 5 - instruction fetch buffer: 2 Radix-4 hardware divider Real-time trace
Floating point	Double-precision instructions Divide and square-root instructions Multiply and accumulate instructions	Double-precision instructions Divide and square-root instructions Multiply and accumulate instructions
Multi-Core		Inter-core Interrupt Unit Inter-core Semaphore Unit Inter-core Message Unit Inter-core Debug Unit Global Real-Time Counter Unit
Coherency Unit	no	yes
I/O Coherency	no	yes
Memory	Memory Protection Unit	Memory Management Unit System-level cache (512k)

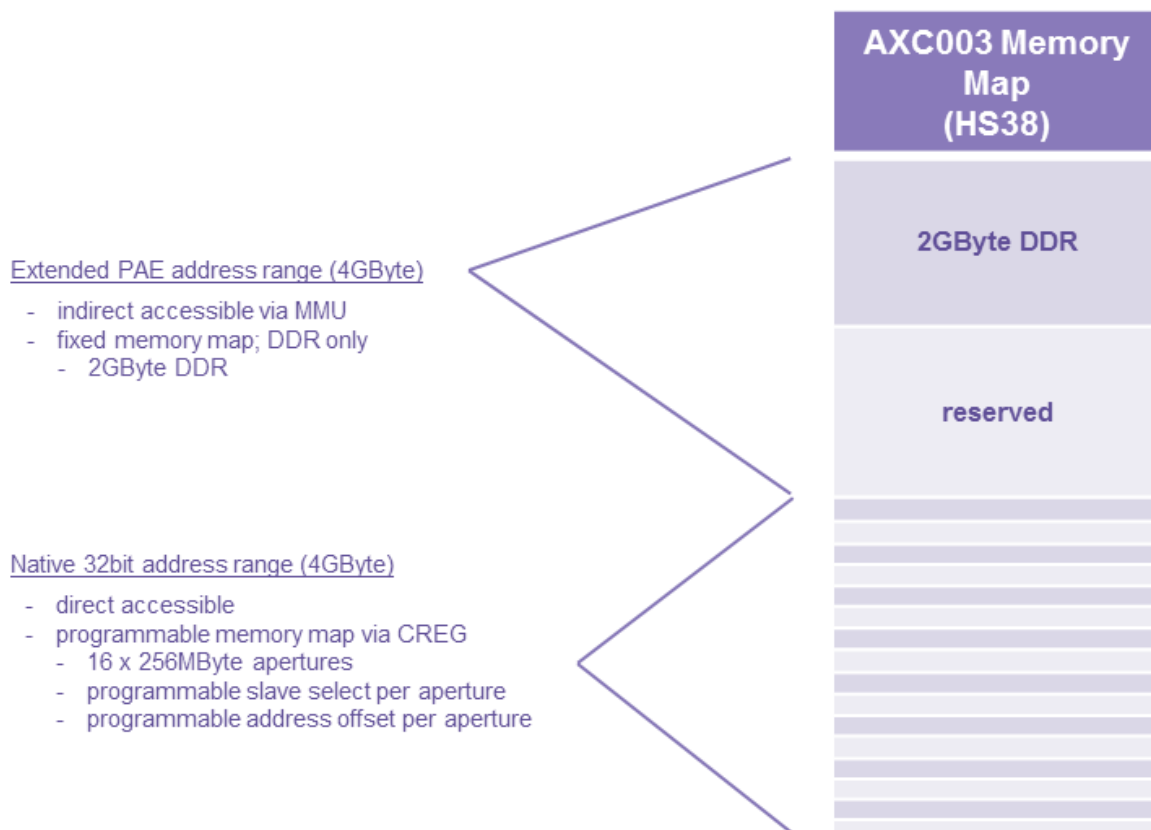
Feature	ARC HS36	ARC HS38x2
Maximum CPU frequency	100 MHz	100 MHz

### 6.7.2 PAE

The ARC HS38x2 core supports Physical Address Extension (PAE). PAE extends the physical address range beyond the core’s native 32-bit, 4GByte address range. In the AXC003, the PAE functionality is used to extend the physical address range from 4 to 8 GByte. A high-level overview of the memory map is shown in Figure 23. The memory map for the lower 4GByte of the physical address range is fully programmable via control registers. The sections “Controlling the Memory Map” and “Software Interfaces” provide additional information.

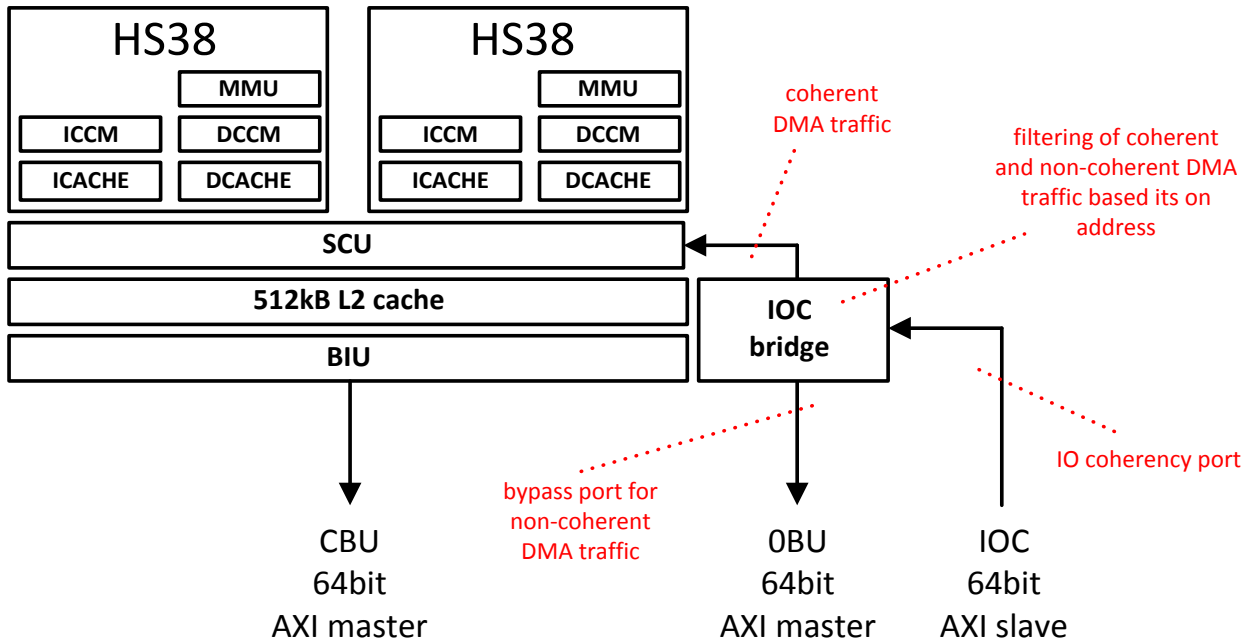
The memory map for the extended part of the physical address range is fixed, and used for DDR only. The AXC003 supports 2 GByte of DDR.

Figure 23 AXC003 Memory Map



### 6.7.3 I/O Coherency

Figure 24 AXC003 I/O Coherency Architecture



The AXC003 Processor FPGA supports the HS38x2 hardware I/O mechanism for ARC.

The I/O coherency architecture is illustrated Figure 24. The ARC HS38x2 provides an additional AXI slave port (the I/O coherency port) that can be used by any DMA client that needs to operate on cacheable data that is shared with the ARC core. Traffic on the I/O coherency port is filtered by the IOC bridge based on its address. When it falls within a certain (programmable) address window, the traffic is forwarded to the SCU. When it falls outside the address window, the traffic is forwarded to its final destination without being snooped.

#### 6.7.3.1 I/O Coherency and Physical Address Extension

The DMA clients on the AXS mainboard do not support PAE. Their physical address range is limited to 32 bits (4 G). The limitation of 4 G physical address range causes a problem when a DMA client needs to operate on cacheable data in the PAE region that is shared with the ARC core. Instead, a special PAE control register is provided (see Table 16). This register makes it possible to map coherent DMA traffic to the PAE region.

The basic principle of the PAE register is illustrated in Figure 25 on page 39. Note that the coherent DMA client in Figure 25 uses the same physical-address map as the ARC HS38x2; the MSB bits of the physical address are simply ignored by the DMA client.



The ICTL module combines the interrupt requests from the on-chip interrupt sources into a single interrupt request for each ARC processor.

Dual-core HS38x2 has cross-core interrupts between the cores instantiated in the inter-core interrupt unit.

The interrupt source responsible for generating the interrupt request can be determined by reading back the interrupt status register in the GPIO sub-module of the ICTL.

Interrupts should be cleared within the GPIO sub-module of the ICTL module. The control signals from the CREG and the CGU are pulses and are thus self-clearing.

Use the GPIO driver for accessing the registers inside the ICTL.

- External interrupts

The ICTL module on the Mainboard combines all Interrupt requests from the Mainboard peripherals into a single signal, which is received by the (top level) GPIO module on the AXC003 CPU Card. The interrupt output of the GPIO module is shared between all ARC cores.

In a first step the interrupt handler should read the status register of the ICTL module on the Mainboard to identify the source peripheral of the interrupt. In a second step the interrupt status register of the source peripheral provides the primary root cause. At both intermediate levels (Mainboard ICTL and CPU Card GPIO) the interrupts are level-sensitive. Interrupts should be cleared within the source peripheral.

The dual-core HS38x2 has an interrupt distribution unit that controls interrupt request connections from CREG, ICTL, and GPIO.

All interrupts inputs of the ICTL module are edge-sensitive. [Figure 26](#) on page 41 and [Figure 27](#) on page 42 show the top-level interrupt architecture of the AXC003 Processor FPGA.



Figure 26 HS36 Interrupt Architecture

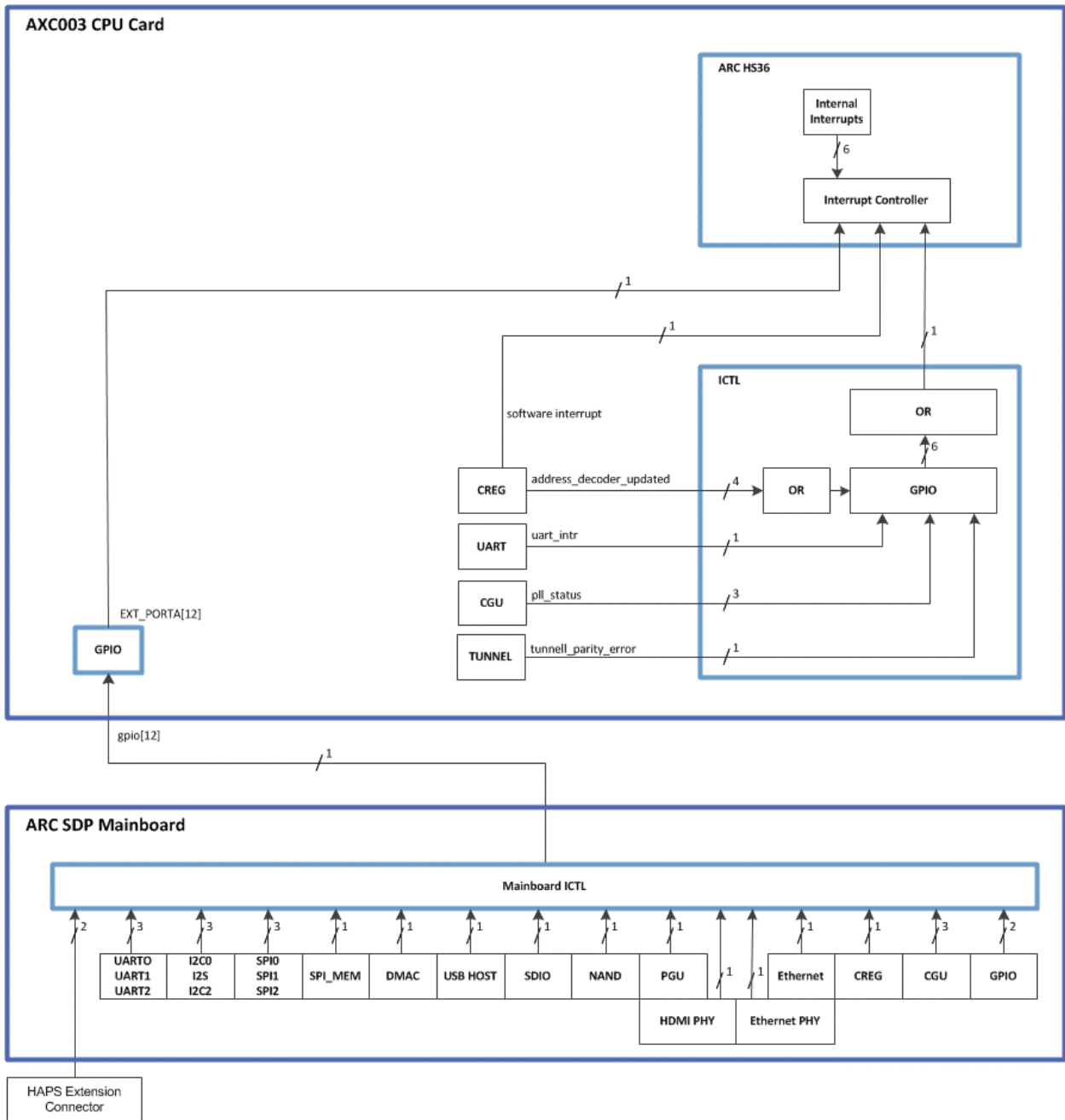
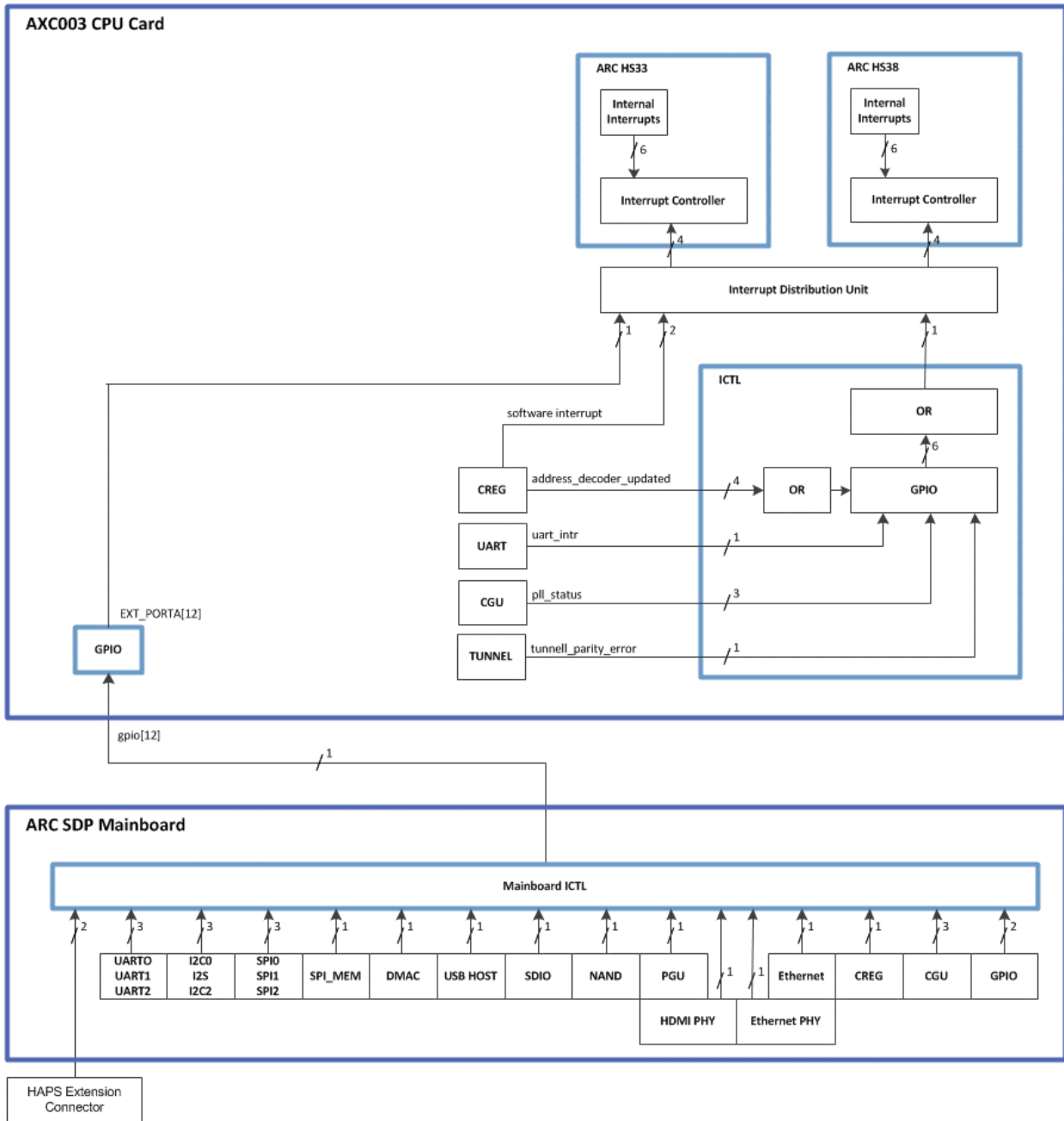


Figure 27 HS38x2 Interrupt Architecture



Each ARC core is configured with 8 external interrupt inputs.

- The interrupt mapping of the two ARC cores is listed in [Table 7](#) on page 43 and [Table 8](#) on page 44.
- [Table 9](#) on page 45 shows the correspondence of the bits in the interrupt status register of the Mainboard ICTL to the interrupt source peripherals.

Table 7 Interrupt Mapping for ARC HS36

Interrupt #	Interrupt source	Remarks																					
0	ARC internal interrupt	reset																					
1		memory error																					
2		instruction error																					
16		timer0																					
17		timer1																					
18		Reserved																					
19																							
20																							
21																							
22																							
23																							
24	Combined interrupt from Mainboard peripherals	The interrupt request is received by the ICTL module on the Mainboard and the GPIO register EXT_PORTA[12] on the AXC003 CPU Card. See <a href="#">Table 9</a> .																					
25	Interrupt from interrupt controller on AXC003 CPU Card	Note that interrupts that are edge-sensitive within the ICTL should also be cleared at the ICTL level.																					
		<table border="1"> <thead> <tr> <th>ICTL bit</th> <th>Description</th> <th>Sensitivity</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Tunnel parity error</td> <td>level</td> </tr> <tr> <td>1</td> <td>Address decoder updated</td> <td>edge</td> </tr> <tr> <td>2</td> <td>UART</td> <td>level</td> </tr> <tr> <td>3</td> <td>PLL locked</td> <td>edge</td> </tr> <tr> <td>4</td> <td>PLL unlocked</td> <td>edge</td> </tr> <tr> <td>5</td> <td>PLL lock error</td> <td>edge</td> </tr> </tbody> </table>	ICTL bit	Description	Sensitivity	0	Tunnel parity error	level	1	Address decoder updated	edge	2	UART	level	3	PLL locked	edge	4	PLL unlocked	edge	5	PLL lock error	edge
ICTL bit		Description	Sensitivity																				
0		Tunnel parity error	level																				
1		Address decoder updated	edge																				
2		UART	level																				
3		PLL locked	edge																				
4	PLL unlocked	edge																					
5	PLL lock error	edge																					
26	CREG	Interrupt request from AX003 Control Registers																					

Table 8 Interrupt Mapping for ARC HS38

Interrupt #	Interrupt source	Remarks																					
0	ARC internal interrupt	reset																					
1		memory error																					
2		instruction error																					
16		timer0																					
17		timer1																					
18		Reserved																					
19	Cross-core Interrupt	The interrupt request from Inter-core Interrupt Unit																					
20	ARC internal interrupt	Performance counter																					
21	Reserved																						
22																							
23																							
24	Combined interrupt from Mainboard peripherals	The interrupt request is received via the ICTL module on the Mainboard and the GPIO register EXT_PORTA[12] on the AXC003 CPU Card. See <a href="#">Table 9</a> .																					
25	Interrupt from interrupt controller on AXC003 CPU Card	Note that interrupts that are edge-sensitive within the ICTL should also be cleared at the ICTL level.																					
		<table border="1"> <thead> <tr> <th>ICTL bit</th> <th>Description</th> <th>Sensitivity</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Tunnel parity error</td> <td>level</td> </tr> <tr> <td>1</td> <td>Address decoder updated</td> <td>edge</td> </tr> <tr> <td>2</td> <td>UART</td> <td>level</td> </tr> <tr> <td>3</td> <td>PLL locked</td> <td>edge</td> </tr> <tr> <td>4</td> <td>PLL unlocked</td> <td>edge</td> </tr> <tr> <td>5</td> <td>PLL lock error</td> <td>edge</td> </tr> </tbody> </table>	ICTL bit	Description	Sensitivity	0	Tunnel parity error	level	1	Address decoder updated	edge	2	UART	level	3	PLL locked	edge	4	PLL unlocked	edge	5	PLL lock error	edge
ICTL bit		Description	Sensitivity																				
0		Tunnel parity error	level																				
1		Address decoder updated	edge																				
2		UART	level																				
3		PLL locked	edge																				
4		PLL unlocked	edge																				
5	PLL lock error	edge																					
26	CREG	Configurable interrupt request from AXC003 CREG																					
27	CREG	Configurable interrupt request from AXC003 CREG																					

In each core of HS38x2, interrupts 24...27 are controlled by the interrupt distribution unit. For more information about the interrupt distribution unit, see the *ARConnect Databook* provided with the ARConnect IP.

Table 9 Mainboard ICTL Interrupt Mapping

<b>ICTL_MB INT_STATUS Register Bit</b>	<b>Interrupt Source</b>
0	Mainboard CGU: PLL lock interrupt
1	Mainboard CGU: PLL unlock interrupt
2	Mainboard CGU: PLL lock error interrupt
3	Mainboard CREG interrupt
4	Ethernet interrupt
5	PGU interrupt
6	NAND interrupt
7	SDIO interrupt
8	USB HOST interrupt
9	DMAC interrupt
10	SPI_MEM interrupt
11	SPI0 interrupt
12	SPI1 interrupt
13	SPI2 interrupt
14	I2C0 interrupt
15	I2S interrupt
16	I2C2 interrupt
17	UART0 interrupt
18	UART1 interrupt
19	UART2 interrupt
20	Mainboard GPIO0 interrupt
21	Mainboard GPIO1 interrupt
22	Ethernet PHY interrupt
23	HDMI PHY interrupt
24	HAPS Extension 0 interrupt (signal HE_intr[0] at connector)

ICTL_MB INT_STATUS Register Bit	Interrupt Source
25	HAPS Extension 1 interrupt (signal HE_intr[1] at connector)

### 6.7.5 Clock

The AXC003 CPU Card has two clock inputs from which all other clocks are derived internally:

- 33 MHz reference clock
- AXI tunnel clock

These clocks are provided at the HapsTrak II connectors on the bottom side of the AXC003 CPU Card. [Figure 28](#) on page 47 shows the top-level clock architecture. The 33-MHz input clock is supplied to the CGU. The CGU generates all the internal clocks using the internal PLLs and clock dividers of the FPGA. The main clock domains are highlighted by the different colors in [Figure 28](#). The AXC003 Processor FPGA has the following main clock domains:

- 33 MHz clock  
This is the input clock of the AXC003 Processor FPGA. All other clocks are derived from this clock. The only exception is the source-synchronous input clock for the AXI tunnel.
- DDR reference clock  
This is the reference clock for the DDR3 controller and PHY. This clock is used to generate the 400 MHz DDR memory clock and the different internal clock phases (0°, 90°, 180°, 270°). The AXI bus runs at a quarter of the DDR memory clock frequency, that is, at 100 MHz.
- ARC HS core clock
- APB clock
- AXI Tunnel clock

Figure 28 Clock Architecture

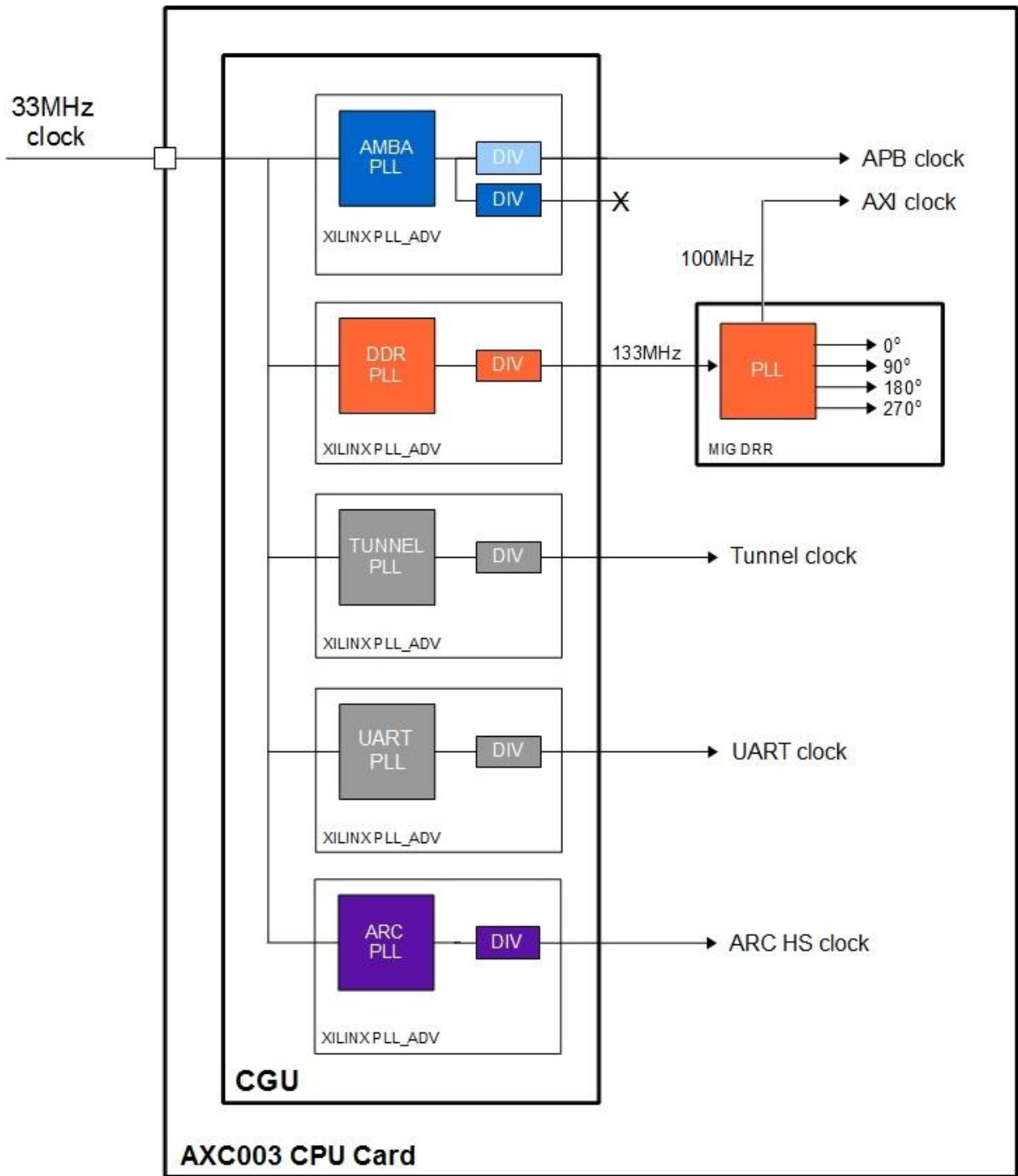


Table 10 Clock Frequencies

PLL	Clock	Clk Frequency (MHz)			Run-Time Programmable
		Maximum	After Reset	After Pre-Boot	
<b>AMBA</b>					
	apb_clk	100	100	100	No
	axi_clk	100	100	100	No
<b>DDR</b>					
	ddr_ref_clk	133	133	133	No
<b>TUNNEL</b>					
	tunnel_clk	100	50	100	Yes
<b>UART</b>					
	uart_ref_clk	33	33	33	No
<b>ARC</b>					
	arc_clk	100 / 100 <sup>1)</sup>	50	100	Yes

1)  $F_{max}$  for HS36 and HS38x2

## 6.7.6 Reset

The AXC003 Processor FPGA has one external reset pin (`rst_n_in`) that serves as an active low, hardware reset. When the external hardware reset is active, the entire FPGA is reset. This pin is routed to the HapsTrak II connector and controlled by reset circuitry on the ARC SDP Mainboard, which also implements the power-on-reset. The CGU on the AXC003 Processor FPGA performs reset synchronization to the internal clock domains.

Use the `Reset` button on the ARC SDP Mainboard to trigger a reset of the entire system.

Figure 29 Location of the RESET Button on the ARC SDP Mainboard



The AXC003 Processor FPGA has an external reset output pin (`rst_n_out`), which is routed to the HapsTrak II connector as well. This reset output is used by the reset controller on the ARC SDP Mainboard.





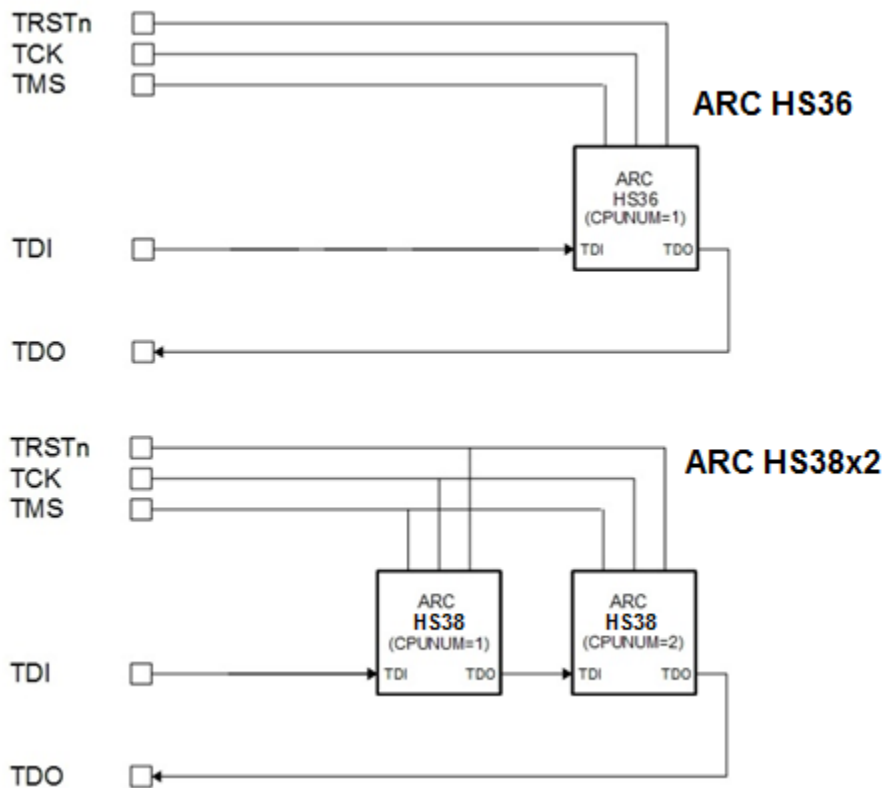
**Note**

The AXC003 Processor FPGA includes a pushbutton labeled SW801 CONF. This pushbutton is for reconfiguring the CPU card FPGA. Pushing this button forces loading the ARC CPU image to the FPGA, depending on the SW802 dipswitch settings

### 6.7.7 Debug

The ARC cores provide debug access using an IEEE 1149.1 JTAG port. In the AXC003 Processor FPGA the JTAG ports of the different ARC cores are daisy-chained into a JTAG chain, where the data output from the first core becomes the data input to the second core and so forth; the control and clock signals are common to all the cores in the chain. The JTAG chain for the AXC003 Processor FPGA is shown in Figure 30.

Figure 30 JTAG Daisy-Chain



To distinguish between the individual cores in the JTAG chain, each core has a unique JTAG IDCODE as listed in Table 11.

Table 11 JTAG ID Codes

Core	JTAG ID bit [31:0]	ARC ID	CPUNUM
ARC HS36	0x2014_24B1	0x0553	1
ARC HS38x2, core 1	0x2000_24B1	0x0053	1
ARC HS38x2, core 2	0x2004_24B1	0x0153	2

## 6.7.8 Control Registers

The CREG peripheral inside the AXC003 Processor FPGA provides software registers to control the following features:

- System memory map (see [System Memory Map](#) on page 65)
- Boot mode (see [Boot Modes](#) on page 72)

Table 12 lists the control registers and provides the address offsets to the base address of the AXI2APB segment in the system memory map. By default the base address of the AXI2APB segment is 0xF000\_0000 (see [System Memory Map After Pre-Bootloader Execution](#) on page 65).

For a detailed description of the control registers, see [Software Interfaces](#) on page 108.

Table 12 Control Register Memory Map

Name	Address Offset	R/W	Description
<b>Clock Generation Registers</b>			
TUN_PLL_IDIV	0x0000_0040	RW	Tunnel input divider register
TUN_PLL_FBIDIV	0x0000_0044	RW	Tunnel feedback divider register
TUN_PLL_ODIV	0x0000_0048	RW	Tunnel output divider register
ARC_PLL_IDIV	0x0000_0080	RW	ARC input divider register
ARC_PLL_FBIDIV	0x0000_0084	RW	ARC feedback divider register
ARC_PLL_ODIV	0x0000_0088	RW	ARC output divider register
TUN_PLL_LOCK	0x0000_0108	R	Tunnel PLL lock register
ARC_PLL_LOCK	0x0000_0110	R	ARC PLL lock register
<b>AXI Tunnel Address Decoder Registers</b>			
TUN_A_SLV_SEL0	0x0000_1000	RW	Slave select register for AXI tunnel
TUN_A_SLV_SEL1	0x0000_1004	RW	Slave select register for AXI tunnel
TUN_A_SLV_OFFSET0	0x0000_1008	RW	Address offset register for AXI tunnel
TUN_A_SLV_OFFSET1	0x0000_100C	RW	Address offset register for AXI tunnel
TUN_A_UPDATE	0x0000_1014	RW1C	Address decoder update register for AXI tunnel

Name	Address Offset	R/W	Description
<b>ARC CPU Address Decoder Registers</b>			
CPU_A_SLV_SEL0	0x0000_1020	RW	Slave select register for ARC CPU
CPU_A_SLV_SEL1	0x0000_1024	RW	Slave select register for ARC CPU
CPU_A_SLV_OFFSET0	0x0000_1028	RW	Address offset register for ARC CPU
CPU_A_SLV_OFFSET0	0x0000_102C	RW	Address offset register for ARC CPU
CPU_A_UPDATE	0x0000_1034	RW1C	Address decoder update register for ARC CPU
<b>RTT Address Decoder Registers</b>			
RTT_A_SLV_SEL0	0x0000_1040	RW	Slave select register for RTT
RTT_A_SLV_SEL1	0x0000_1044	RW	Slave select register for RTT
RTT_A_SLV_OFFSET0	0x0000_1048	RW	Address offset register for RTT
RTT_A_SLV_OFFSET0	0x0000_104C	RW	Address offset register for RTT
RTT_A_UPDATE	0x0000_1054	RW1C	Address decoder update register for RTT
<b>Physical Address Extensions Registers</b>			
CREG_PAE	0x0000_1060	RW	PAE register
CREG_PAE_UPDATE	0x0000_1074	RW1C	PAE update register
<b>CPU Start Registers</b>			
CPU_START	0x0000_1400	RW	ARC CPU start register
CPU_0_ENTRY	0x0000_1404	RW	ARC CPU-0 kernel entry point register
CPU_1_ENTRY	0x0000_1408	RW	ARC CPU-0 kernel entry point register
CPU_BOOT	0x0000_1010	RW	ARC CPU boot register

Name	Address Offset	R/W	Description
<b>AXI Tunnel Registers</b>			
TUN_CTRL	0x0000_14A0	RW	AXI tunnel control register
TUN_STAT	0x0000_14A4	R	AXI tunnel status register

### 6.7.9 GPIO Registers

The GPIO peripheral in the AXC003 Processor FPGA uses DesignWare `dw_apb_gpio` IP [3] and provides two GPIO ports.

Table 13 lists the GPIO registers and provides the address offsets to the base address of the AXI2APB section in the system memory map.

By default the base address of the AXI2APB segment is `0xF000_0000` (see [System Memory Map After Pre-Bootloader Execution](#) on page 65). Table 14 describes the function of the GPIO bits.

Table 13 GPIO Register Memory Map

Name	Address Offset	R/W	Description
GPIO_SWPORTA_DR	0x3000	R/W	Port A Data Register Controls LEDs on the ARC SDP Mainboard
GPIO_EXT_PORTA	0x3050	R	External Port A Register Input from CPU Start buttons on the ARC SDP Mainboard and from the Mainboard's interrupt request
GPIO_SWPORTB_DR	0x300C	R/W	Port B Data Register Controls LEDs and seven-segment displays on the AXC003 CPU Card
GPIO_EXT_PORTB	0x3054	R	External Port B Register Input from jumpers on the AXC003 CPU Card

Table 14 below and [Table 15](#) on page 53 describe the function of the Port A registers. [Table 16](#) on page 54 and [Table 17](#) on page 55 describe the Port B registers.

Table 14 GPIO Port A Output Register Bit Function (SWPORTA\_DR)

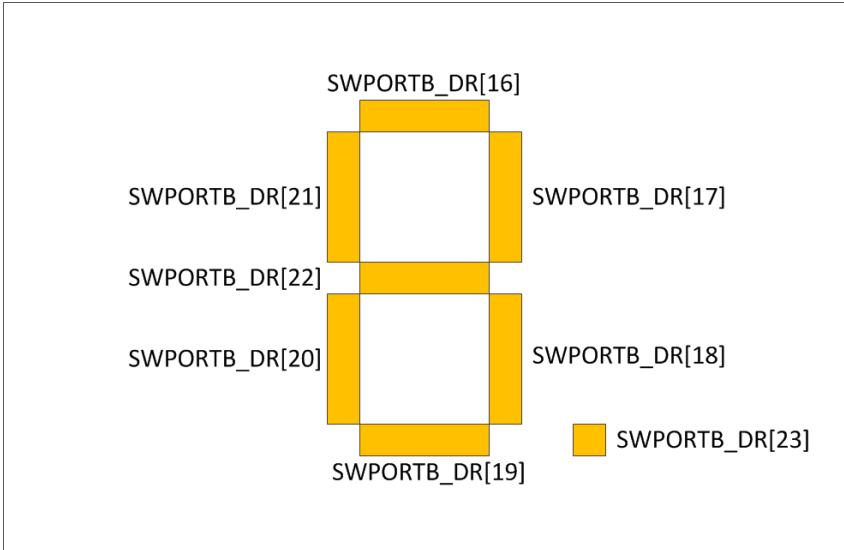
Bit	Description
0	Connected to LED2501 of the ARC SDP Mainboard The LED is ON when this bit is set to 1.

Bit	Description
1	Connected to LED2502 of the ARC SDP Mainboard The LED is ON when this bit is set to 1.
4 : 2	Reserved
5	Connected to LED2503 of the ARC SDP Mainboard. The LED is ON when this bit is set to 1.
6	Connected to LED2504 of the ARC SDP Mainboard. The LED is ON when this bit is set to 1.
9 : 7	Reserved
10	Connected to LED2505 of the ARC SDP Mainboard. The LED is ON when this bit is set to 1.
11	Connected to LED2506 of the ARC SDP Mainboard. The LED is ON when this bit is set to 1.
14 : 12	Reserved
15	Connected to LED2507 of the ARC SDP Mainboard. The LED is ON when this bit is set to 1.
16	Connected to LED2508 of the ARC SDP Mainboard. The LED is ON when this bit is set to 1.
31 : 17	Reserved

Table 15 GPIO port A Input Register Function (EXT\_PORTA)

Bit	Description
11 : 0	Reserved
12	Connected to the interrupt controller of the ARC SDP Mainboard. Can be used to provide an interrupt from the peripheral subsystem of the ARC SDP Mainboard to a core on the AXC003 CPU Card.
19 : 13	Reserved
20	Connected to pushbutton SW2504 of the ARC SDP Mainboard
21	Connected to pushbutton SW2506 of the ARC SDP Mainboard
22	Connected to pushbutton SW2505 of the ARC SDP Mainboard
23	Connected to pushbutton SW2507 of the ARC SDP Mainboard
31 : 24	Reserved

Table 16 GPIO port B Output Register Function (SWPORTB\_DR)

Bit	Description
7:0	Reserved
8	Connected to LED0 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
9	Connected to LED1 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
10	Connected to LED2 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
11	Connected to LED3 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
12	Connected to LED4 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
13	Connected to LED5 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
14	Connected to LED6 on the AXC003 CPU Card The LED is ON when this bit is set to 1.
15	Connected to LED7 on the AXC003 CPU Card. The LED is ON when this bit is set to 1.
23:16	Controls the upper seven-segment display on the AXC003 CPU Card. A segment of the display is ON when its control bit is set to 1.  <p>The diagram shows a seven-segment display with segments labeled as follows: SWPORTB_DR[16] (top), SWPORTB_DR[17] (top-right), SWPORTB_DR[18] (bottom-right), SWPORTB_DR[19] (bottom), SWPORTB_DR[20] (bottom-left), SWPORTB_DR[21] (top-left), and SWPORTB_DR[22] (middle). A legend indicates that SWPORTB_DR[23] is represented by a yellow square, which is used to highlight the segments SWPORTB_DR[16], SWPORTB_DR[17], SWPORTB_DR[18], SWPORTB_DR[19], SWPORTB_DR[20], SWPORTB_DR[21], and SWPORTB_DR[22] in the diagram.</p>

Bit	Description
31 : 24	<p>Controls the lower seven-segment display on the AXC003 CPU Card. A segment of the display is ON when its control bit is set to 1.</p> <p>The diagram shows a seven-segment display with segments labeled as follows: SWPORTB_DR[24] (top), SWPORTB_DR[25] (top-right), SWPORTB_DR[26] (bottom-right), SWPORTB_DR[27] (bottom), SWPORTB_DR[28] (bottom-left), SWPORTB_DR[29] (top-left), and SWPORTB_DR[30] (middle). A legend indicates that SWPORTB_DR[31] controls the segments shown in yellow.</p>

Table 17 GPIO port B input register function (EXT\_PORTB)

Bit	Description
0	Reserved
1	Connected to JP1201; usage reserved
2	Connected to JP1202; usage reserved
3	Connected to JP1203; usage reserved
4	Connected to JP1204; usage reserved
5	Connected to JP1205; usage reserved
6	Connected to JP1206; usage reserved
7	Connected to JP1207; usage reserved
31 : 8	Reserved

### 6.7.10 DIP Switches for FPGA Image Selection

AXC003 CPU card has SPI flash ROM that contains two FPGA images – HS36 and HS38x2. One of these images is selected on reset stage depending on DIP-switch settings. The DIP-switches are in the SW802 IMAGE SELECT component.

Bits of SW802 define the FPGA image that will be selected:

- 00 – HS36
- 01 – HS38x2

- 10 – Reserved
- 11 – Reserved

### 6.7.11 ARC HS34 Emulation

The HS36 configurations on the AXC003 CPU card also have closely coupled memories: 256k ICCM and 256k DCCM. This configuration can be used for an emulation of HS34 cores and for working with software packages built and compiled for HS34.

The difference between HS34 and HS36 with CCM memories on AXC003 is in memory mapping. The memory mapping of HS36 with CCM is shown in Table 18.

In HS34 emulation mode the DDR3 memory is also available, but its latency is larger than in closely coupled memory. This mode also has data cache and instruction cache disabled.

The ARC HS36 core has internal ICCM and DCCM memories. The locations of these memories depend on register settings in the ARC HS36 core. The pre-boot loader keeps the ICCM at its reset address 0x1000\_0000 but moves the DCCM base address to 0xC000\_0000. Therefore the ARC HS36 core can only access the RAM on the ARC SDP Mainboard using 0x0000\_0000 as the base address.

The start address of DDR3 SDRAM is 0x8000\_0000.

See Chapter 7, [System Memory Map](#), for a detailed mapping description.

*Table 18 Memory mapping for ARC HS36*

0xFFFF_FFFF 0xF000_0000	AXI2APB on AXC003 CPU Card (CREG)
0xEFFF_FFFF 0xE000_0000	AXI2APB on Mainboard
0xDFFF_FFFF 0xD000_0000	AXI Tunnel Slave for HAPS System
0xCFFF_FFFF 0xC000_0000	DCCM 256k
0xBFFF_FFFF 0x8000_0000	DDR3 SDRAM
0x7FFF_FFFF 0x4000_0000	Unused
0x3FFF_FFFF 0x3000_0000	Internal ROM
0x2FFF_FFFF 0x2000_0000	SRAM on Mainboard
0x1FFF_FFFF 0x1000_0000	ICCM 256k
0x0FFF_FFFF 0x0000_0000	SRAM on AXC003 CPU Card



**Table 19** Memory mapping for HS34 Emulation

0xFFFF_FFFF 0xF000_0000	AXI2APB on AXC003 CPU Card (CREG)
0xEFFF_FFFF 0xE000_0000	AXI2APB on Mainboard
0xDFFF_FFFF 0xD000_0000	AXI Tunnel Slave for HAPS System
0xCFFF_FFFF 0xC000_0000	DCCM 256k
0xBFFF_FFFF 0x8000_0000	<del>DDR3</del> SDRAM
0x7FFF_FFFF 0x4000_0000	Unused
0x3FFF_FFFF 0x3000_0000	Internal ROM
0x2FFF_FFFF 0x2000_0000	SRAM on Mainboard
0x1FFF_FFFF 0x1000_0000	ICCM 256k
0x0FFF_FFFF 0x0000_0000	SRAM on AXC003 CPU Card

For more booting information see ARC HS36 Booting from ICCM0 on page 73.

To make proper builds of software for HS36 with CCM memories, the `arc_hs34.tcf` file is provided in the AXC003 software package.

The GPIO pin located at SW2501[6] defines whether data cache and instruction cache are used. This pin does not change hardware configuration, and it is used by the pre-bootloader software. During initialization the pre-bootloader reads the value of this pin through CREG. If this bit is 1, data cache and instruction cache are enabled and the CPU operates as HS36. Otherwise the caches are disabled and CPU is in HS34 emulation mode.

## 6.8 Memories on the AXC003 CPU Card

The global memory is available on the AXC003 CPU Card:

- 2 GByte DDR3 SDRAM
- 256 KByte SRAM
- 32 KByte internal ROM

The memory controller for the DDR3 SDRAM supports a single port.

An internal ROM controller is implemented as FPGA RAM blocks initialized with a pre-bootloader code. This region is not intended to be used by application software.

Additional local memories (DCCM and ICCM0) are available for the ARC HS36 core; see [Main Features of the ARC Cores](#) section on page 35.

The ARC SDP Mainboard provides additional global memories.

## 6.9 Power Supply

Power is supplied to the AXC003 CPU Card by the ARC SDP Mainboard using the power supply connector on the bottom side of the AXC003 CPU Card. The following voltage levels are provided: 1.1V, 1.8V, 2.5V, 3.3V, and 12.0V.

Table 20 provides a pin description of the power-supply connector.

*Table 20 Pinout of the Power-Supply Connector*

<b>Pin</b>	<b>Name</b>	<b>Description</b>
1, 2	12V0	12.0 Volt power supply pin
3, 4	GND	Ground supply pin
5, 6	1V1	1.1 Volt power supply pin
7, 8	GND	Ground supply pin
9, 10	1V8	1.8 Volt power supply pin
11, 12	GND	Ground supply pin
13, 14	2V5	2.5 Volt power supply pin
15, 16	GND	Ground supply pin
17, 18	3V3	3.3 Volt power supply pin

Figure 31 on page 59 shows the pinout.

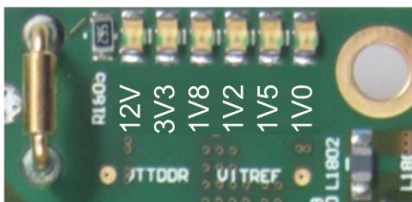
Figure 31 Pinout of the Power Supply Connector (Bottom View)

		P6			
12V0	2	○	○	1	12V0
GND	4	○	○	3	GND
1V1	6	○	○	5	1V1
GND	8	○	○	7	GND
1V8	10	○	○	9	1V8
GND	12	○	○	11	GND
2V5	14	○	○	13	2V5
GND	16	○	○	15	GND
3V3	18	○	○	17	3V3



For each supply voltage of the AXC003 CPU Card a power-control LED indicates that the supply voltage is present. In normal operation all six LEDs shine green. The location of these LEDs is shown in Figure 32.

Figure 32 Location of the Power Control LEDs on the AXC003 CPU Card



## 6.10 Audio Support

The AXC003 CPU Card does not use the audio resources of the ARC SDP Mainboard and drives all audio input signals of the Mainboard at the HapsTrak II Connector to their inactive state.

## 6.11 Usage of ARC SDP Mainboard Resources

### 6.11.1 Usage of the Mainboard DIP Switches

During reset the DIP switches SW2401, SW2501, SW2502 and SW2503 of the ARC SDP Mainboard are connected to the AXC003 Processor FPGA through the GPIO signal group of the HapsTrak II connector. At the end of reset, the switch settings are latched inside the AXC003 Processor FPGA and determine the boot behavior of the ARC CPUs as described in [Table 21](#) on page 60 [Table 22](#) on page 61.

The DIP-switch settings determine the initial values of some control registers.

**Note**



- When a DIP switch is in the right position (  ), the corresponding bit in the control register is 0.
- When a DIP switch is in the left position (  ), the corresponding bit in the control register is 1.

Table 21 ARC Core Boot Configuration (Mainboard DIP Switch SW2501)






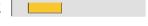

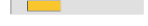

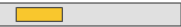


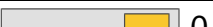

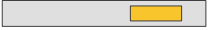


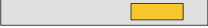

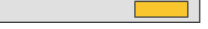
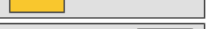




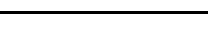
Bit	Description		
# 1	Boot mirror select	<b>Switch Position</b>	<b>Boot Mirror</b>
		1  2  00	Disabled
		1  2  01	Internal ROM
		1  2  10	Reserved
# 2		1  2  11	Reserved
# 3	Bypass loading	<b>Switch Position</b>	<b>Pre-bootloader Mode</b>
		3  0	Pre-bootloader bypasses loading application from SPI flash. Only default initialization is done.
		3  1	Pre-bootloader looks for the appropriate application in SPI flash and runs it if found.
# 4		Reserved	
# 5		Reserved	
# 6	Cache bypass	<b>Switch Position</b>	<b>Cache Mode (HS34/36 only)</b>
		6  0	Data cache and Instruction cache are used
		6  1	Data cache and Instruction cache are not used
# 7	Boot start mode	<b>Switch Position</b>	<b>Start Mode</b>
		7  0	Start ARC core manually (CREG, external start button or debugger)
		7  1	Start ARC core autonomously after reset

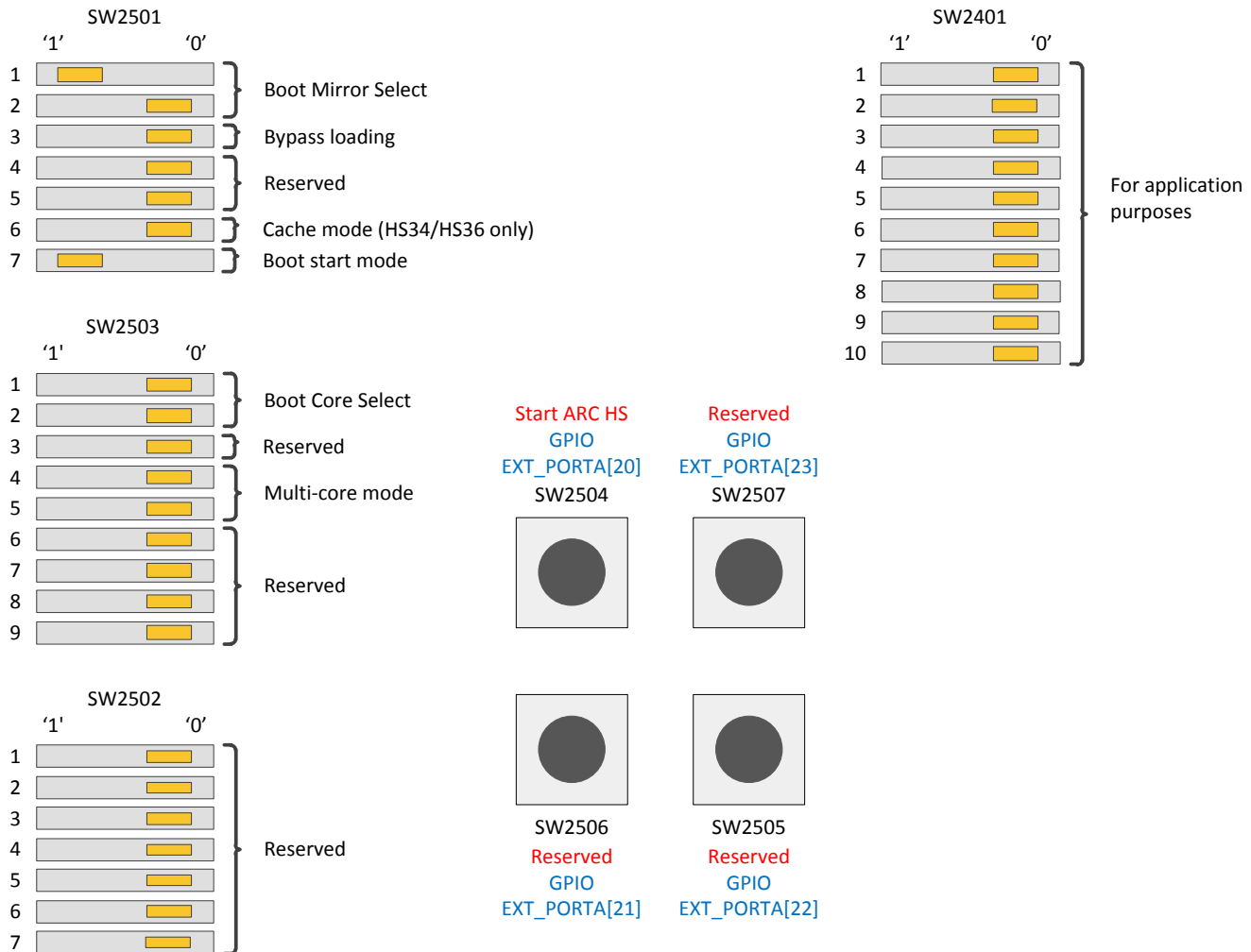
Table 22 Multicore Configuration (Mainboard DIP Switch SW2503)

Bit	Description	Switch Position	Boot Core
# 1	Boot core select		
# 2		1  2  00	HS38x2_0 or HS36
		1  2  01	HS38x2_1
		010	Reserved
		011	Reserved
# 4	Multicore mode	Switch Position	Multicore mode
# 5		4  5  00	Single core
		4  5  01	Dual core
		4  5  10	Reserved
		4  5  11	Reserved
# 6		Reserved	
# 7		Reserved	

The GPIO pin located at SW2501[6] defines whether data cache and instruction cache are used. This pin does not change the hardware configuration, and it is used by the pre-bootloader software.

Figure 33 shows the function and default settings of the DIP Switches on the ARC SDP Mainboard, which are used by the AXC003 CPU Card.

Figure 33 Function and Default Settings of the DIP Switches on the ARC SDP Mainboard.



The DIP switch settings shown in Figure 33 are the factory default settings. All cores are configured to boot from internal ROM after reset.

### 6.11.2 Usage of the Mainboard Pushbuttons

The start behavior of the ARC cores on the AXC003 CPU Card is configurable, one of the options being that the core starts automatically after a reset. For the remaining options the core halts after a reset, but multiple ways exist to start code execution. One of these ways is to start code execution when a button is pushed. The corresponding CPU Start buttons are located on the ARC SDP Mainboard. The behavior of the ARC cores is controlled individually using control registers, which are initialized using DIP switches on the ARC SDP Mainboard. See [Usage of the Mainboard DIP Switches](#) on page 59 for details.

Table 23 shows the usage of the CPU Start buttons of the ARC SDP Mainboard to start code execution on the individual cores. Figure 34 shows the location of the CPU Start buttons on the ARC SDP Mainboard.

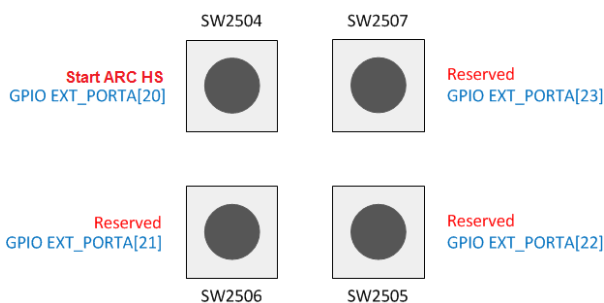
Table 23 Usage of the CPU Start Buttons of the ARC SDP Mainboard

Pushbutton	Description
SW2504	CPU start for ARC CPU when <code>CREG_CPU_START[4] = 0x0</code> (boot start mode switch SW2501[7] set to 0 during reset); GPIO_EXT_PORTA[20] otherwise
SW2505	GPIO_EXT_PORTA[22]
SW2506	GPIO_EXT_PORTA[21]
SW2507	GPIO_EXT_PORTA[23]



**Note** After an ARC core is running, the corresponding CPU Start button can be used for application purposes. This is however not recommended. A CPU Start button should only be used for application purposes when the corresponding core is not configured to start after pushing the button, but to start autonomously, through CREG access or the debugger.

Figure 34 Location of the CPU Start Buttons on the ARC SDP Mainboard.



### 6.11.3 Usage of the Mainboard LEDs

The ARC SDP Mainboard includes eight CPU LEDs that are controlled by the GPIO peripheral of the AXC003 Processor FPGA. These green LEDs are ON when the corresponding control bit is set to 1 and OFF when the control bit is cleared to 0. Table 24 on page 64 lists the control bits of the CPU LEDs and Figure 35 on page 64 shows the LED positions on the ARC SDP Mainboard.

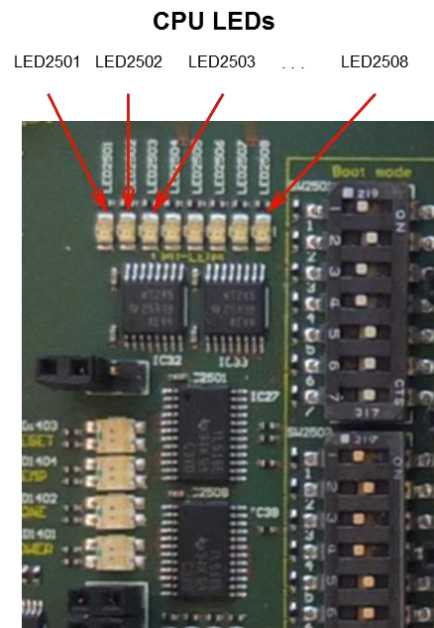


**Note** The ARC SDP Mainboard also has eight GPIO LEDs, which are controlled by the GPIO peripheral in the Mainboard's FPGA.

Table 24 Control Bits of the CPU LEDs on the ARC SDP Mainboard

Control Bit	Description
SWPORTA_DR[0]	Controls LED2501
SWPORTA_DR[1]	Controls LED2502
SWPORTA_DR[5]	Controls LED2503
SWPORTA_DR[6]	Controls LED2504
SWPORTA_DR[10]	Controls LED2505
SWPORTA_DR[11]	Controls LED2506
SWPORTA_DR[15]	Controls LED2507
SWPORTA_DR[16]	Controls LED2508

Figure 35 Location of the CPU LEDs on the ARC SDP Mainboard





## 7.1 System Memory Map After a Reset

Following a reset, the memory maps of all AXI masters on the AXC003 CPU Card and on the ARC SDP Mainboard have to be set up according to the application requirements. Control registers allow programming the system memory map individually for all AXI masters on the AXC003 CPU Card and on the ARC SDP Mainboard. These control registers are located in the local peripheral area of the AXC003 CPU Card and in the peripheral area of the Mainboard. Following a reset, all CPU cores on the AXC003 CPU Card can access the Mainboard peripherals in the address range 0xE000\_0000 to 0xEFFF\_FFFF and the AXC003 CPU Card peripherals in the address range 0xF000\_0000 to 0xFFFF\_FFFF.

The pre-bootloader programs the corresponding address decoders and, thus, provides the default memory map. It also initializes the DDR3 SDRAM.

The default memory map programmed by the Pre-Bootloader is described in the [System Memory Map After Pre-Bootloader Execution](#) on page 65.

If needed, alternative memory maps can be programmed for each AXI master individually by altering the settings of the corresponding control registers. The sections [Controlling the Memory Map](#) on page 67 and [Software Interfaces](#) on page 108 provide additional information.

## 7.2 System Memory Map After Pre-Bootloader Execution

The pre-bootloader sets up the memory maps of all AXI masters on the AXC003 CPU Card and on the ARC SDP Mainboard as shown in Table 25. The memory map has been chosen to be identical for all AXI masters. See [Example Register Settings for the Default Memory Map](#) on page 69 for information on the corresponding register settings.

Table 25 ARC CPU Memory Map After Pre-Bootloader Execution

Master Address	Selected Slave	Slave Address
0xFFFF_FFFF 0xF000_0000	AXI2APB on AXC003 CPU Card	0x0FFF_FFFF 0x0000_0000
0xEFFF_FFFF 0xE000_0000	AXI2APB on Mainboard	0x0FFF_FFFF 0x0000_0000
0xDFFF_FFFF 0xD000_0000	AXI Tunnel Slave for HAPS System <sup>1)</sup>	0xDFFF_FFFF 0xD000_0000

Master Address	Selected Slave	Slave Address
0xCFFF_FFFF 0xC000_0000	256K DCCM <sup>2)</sup>	
0xBFFF_FFFF  0x8000_0000	DDR3 SDRAM	0x3FFF_FFFF  0x0000_0000
0x7FFF_FFFF0  x4000_0000	Unused	
0x3FFF_FFFF 0x3000_0000	ROM on AXC003 CPU Card	0x0FFF_FFFF 0x0000_0000
0x2FFF_FFFF 0x2000_0000	SRAM on MB via AXI Tunnel	0x0FFF_FFFF 0x0000_0000
0x1FFF_FFFF 0x1000_0000	SRAM on AXC003 CPU Card/ 256K ICCM <sup>2)</sup>	0x0FFF_FFFF 0x0000_0000
0x0FFF_FFFF 0x0000_0000	SRAM on AXC003 CPU Card	0x0FFF_FFFF 0x0000_0000

1) *The slave address is transparently forwarded to the AXI tunnel master on the HAPS system. Further address decoding depends on your custom design.*

2) *ICCM and DCCM available only for the ARC HS36 CPU configuration.*

The memory map shown in Table 25 is an aggregate of the individual memory-map settings on the AXC003 CPU Card and the ARC SDP Mainboard, with the AXI Tunnel between the AXC003 CPU Card and the ARC SDP Mainboard abstracted away.

The ARC HS36 core has internal ICCM0 and DCCM memories. The locations of these memories depend on register settings in the ARC HS36 core. The pre-boot loader keeps the ICCM at reset address 0x1000\_0000 but moves the DCCM base address to 0xC000\_0000. Therefore the ARC HS36 core can only access the RAM on the ARC SDP Mainboard using 0x0000\_0000 as the base address. The other cores can access the RAM either using 0x0000\_0000 or 0x1000\_0000 as the base address. Either start address supports accessing the entire RAM.

The SRAM on the AXC003 CPU Card has a size of 256 KBytes. The lower 256 KBytes within the master's 256 MByte aperture access the SRAM. The remaining 261888 KBytes are not accessible.

The pre-bootloader RAM on the ARC SDP Mainboard has a size of 256 KBytes. The lower 256 KBytes within the master's 256 MByte aperture access the RAM. The remaining 261888 KBytes are not accessible.

Table 26 AXI Tunnel Memory Map After Pre-Bootloader Execution (ARC HS34 / HS36)

Master Address	Selected Slave	Slave Address
0xFFFF_FFFF	DDR3 SDRAM	0x7FFF_FFFF
0x8000_0000		0x0000_0000
0x7FFF_FFFF	DDR3 SDRAM	0x7FFF_FFFF
0x0000_0000		0x0000_0000

Table 27 AXI Tunnel Memory Map After Pre-Bootloader Execution (ARC HS38)

Master Address	Selected Slave	Slave Address
0xFFFF_FFFF	DDR3 SDRAM via ARC IOC port	0xFFFF_FFFF
0x8000_0000		0x8000_0000
0x7FFF_FFFF	DDR3 SDRAM	0x7FFF_FFFF
0x0000_0000		0x0000_0000

## 7.3 Controlling the Memory Map

### 7.3.1 Setting Up the AXI Masters on the AXC003 CPU Card

Control registers are available for each AXI master (for each core and for the AXI tunnel) to customize its memory map. The full 4 GByte AXI memory map is partitioned into 16 address apertures of 256 Mbytes each:

- aperture[0]: base address is 0x0000\_0000
- aperture[1]: base address is 0x1000\_0000
- aperture[2]: base address is 0x2000\_0000
- ...
- aperture[15]: base address is 0xF000\_0000

The address map configuration consists of two steps for each 256 MByte aperture within the AXI address space of an AXI master.

First, a target slave is selected from the list shown in Table 28 on page 68. Then, the desired address offset within the memory map of the target slave is programmed. This offset can be selected in steps of 256 MBytes. The specified offset refers to the address offset within the target slave only; the base address of the aperture is not taken into account.

Table 28 AXC003 CPU Card Target Slaves

Slave Number	Target Slave
0	No slave selected (default slave provides response)
1	DDR controller
2	SRAM controller
3	AXI tunnel
4	AXI2APB bridge
5	Internal ROM controller
6	I/O Coherency port
7	Reserved

The AXI tunnel slave transparently forwards the received address to the corresponding AXI tunnel master on the ARC SDP Mainboard. The address map as seen by this master is defined by control registers of the Mainboard. For apertures selecting the AXI Tunnel, it is best to set the address offset such that the address issued by the master on the other side and the original address are identical. This can be achieved by setting the `SLV_OFFSET` field of the corresponding register to the aperture number.

The memory map as seen by the AXI2APB bridge is described in [Memory Map of the Local Peripherals](#) on page 71.

### 7.3.2 Setting Up the AXI Masters on the ARC SDP Mainboard

The address map of the AXI masters on the ARC SDP Mainboard is defined in a similar way as described in the “[Setting Up the AXI Masters on the AXC003 CPU Card](#)” section above.

Table 29 lists the target slaves that are available on the ARC SDP Mainboard. Refer to the ARC SDP Mainboard User Guide [5] for more details.

Table 29 ARC SDP Mainboard Target Slaves

Slave Number	Target Slave
0	No slave selected (default slave provides response)
1	TUNNEL0 (AXI tunnel to/from AXC003 CPU Card)
2	TUNNEL1 (AXI tunnel to/from HAPS System)
3	SRAM controller (for Mainboard RAM)
4	AXI2APB (control/status interface of peripherals)

The AXI tunnel slaves transparently forward the received address to the corresponding AXI tunnel masters on the AXC003 CPU Card or the HAPS system. For TUNNEL0 this address is then decoded according to the memory map programmed for the AXI tunnel master on the

AXC003 CPU Card. For TUNNEL1 the address issued by the AXI tunnel master on the HAPS system is decoded according to your custom design.

### 7.3.3 Example Register Settings for the Default Memory Map

The pre-bootloader memory map as shown in Table 30 below and [Table 31](#) on page 70.

Table 30 ARC CPU Memory Map Pre-Boot Programming on the AXC003 CPU Card

Aperture #	Master Address	SLV_SEL	SLV_OFFSET	Selected Slave	Slave Address
15	0xFFFF_FFFF 0xF000_0000	4	0x0	AXI2APB	0x0FFF_FFFF 0x0000_0000
14	0xEFFF_FFFF 0xE000_0000	3	0xE	AXI Tunnel	0xEFFF_0000 0xE000_0000
13	0xDFFF_FFFF 0xD000_0000	3	0xD		0xDFFF_FFFF 0xD000_0000
12	0xCFFF_FFFF 0xC000_0000	0	0x0	Unused	
11	0xBFFF_FFFF 0xB000_0000	1	0x3	DDR3 SDRAM	0x3FFF_FFFF 0x0000_0000
10	0xAFFF_FFFF 0xA000_0000	1	0x2		
9	0x9FFF_FFFF 0x9000_0000	1	0x1		
8	0x8FFF_FFFF 0x8000_0000	1	0x0		
7	0x7FFF_FFFF 0x7000_0000	0		Unused	
6	0x6FFF_FFFF 0x6000_0000	0			
5	0x5FFF_FFFF 0x5000_0000	0			
4	0x4FFF_FFFF 0x4000_0000	0			
3	0x3FFF_FFFF 0x3000_0000	5	0x0	Internal ROM	0x0003_FFFF 0x0000_0000
2	0x2FFF_FFFF 0x2000_0000	3	0x0	SRAM (Mainboard)	0x0FFF_FFFF 0x0000_0000
1	0x1FFF_FFFF 0x1000_0000	2	0x0	SRAM (CPU card)	0x0000_0000
0	0x0FFF_FFFF 0x0000_0000	2	0x0	SRAM (CPU card)	0x0000_0000

The slave address of the AXI tunnel on the AXC003 CPU Card is transparently forwarded to the AXI TUNNEL0 master of the ARC SDP Mainboard and becomes the address issued by this master. This master address is then decoded according to the memory map of the AXI TUNNEL0 master on the Mainboard as shown in [Table 31](#). For example, if a core issues the

address 0xE000\_0000 the AXI tunnel is selected. The AXI Master on the other side of the tunnel issues the address 0xE000\_0000 and thus selects the AXI2APB bridge of the Mainboard's peripheral area.

Table 31 Memory Map Pre-Boot Programming for All Masters on the ARC SDP Mainboard

Aperture #	Master Address	SLV_SEL	SLV_OFFSET	Selected Slave	Slave Address
15	0xFFFF_FFFF 0xF000_0000	1	0xF	TUNNEL0 (CPU Card)	0xFFFF_FFFF 0xF000_0000
14	0xEFFF_FFFF 0xE000_0000	4	0x0	AXI2APB (Peripherals)	0x0FFF_0000 0x0000_0000
13	0xDFFF_FFFF 0xD000_0000	2	0xD	TUNNEL1 (HAPS System)	0xDFFF_0000 0xD000_0000
12	0xCFFF_FFFF 0xC000_0000	0	0x0	Unused	
11	0xBFFF_FFFF 0xB000_0000	1	0xB	TUNNEL0 (CPU Card)	0xBFFF_FFFF 0xB000_0000
10	0xAFFF_FFFF 0xA000_0000	1	0xA		0xAFFF_FFFF 0xA000_0000
9	0x9FFF_FFFF 0x9000_0000	1	0x9		0x9FFF_FFFF 0x9000_0000
8	0x8FFF_FFFF 0x8000_0000	1	0x8		0x8FFF_FFFF 0x8000_0000
7	0x7FFF_FFFF 0x7000_0000	0	0x0	Reserved	
6	0x6FFF_FFFF 0x6000_0000	0	0x0		
5	0x5FFF_FFFF 0x5000_0000	0	0x0		
4	0x4FFF_FFFF 0x4000_0000	0	0x0		
3	0x3FFF_FFFF 0x3000_0000	0	0x0		
2	0x2FFF_FFFF 0x2000_0000	0	0x0		
1	0x1FFF_FFFF 0x1000_0000	3	0x0	SRAM (Mainboard)	0x0FFF_FFFF 0x0000_0000
0	0x0FFF_FFFF 0x0000_0000	3	0x0	SRAM (Mainboard)	0x0FFF_FFFF 0x0000_0000

The slave address of the AXI TUNNEL0 slave on the ARC SDP Mainboard as listed in Table 31 is transparently forwarded over the AXI tunnel and becomes the address issued by the AXI Tunnel master on the AXC003 CPU Card. It is then decoded according to the memory map programmed for the AXI Tunnel master on the AXC003 CPU Card, which is shown in [Table 30](#) on page 69.

Likewise, the slave address of the AXI TUNNEL1 slave on the ARC SDP Mainboard is forwarded to the AXI Tunnel master on the HAPS system. It is then decoded according to your custom design.

## 7.4 Memory Map of the Local Peripherals

All peripherals of the APB subsystem inside the AXC003 Processor FPGA are mapped into the AXI2APB segment of the system memory map, which has the default base address `0xF000_0000`. Table 32 shows the address offsets of the individual peripherals and the corresponding aperture within the AXI2APB section. This means that the address offset listed in Table 32 has to be added to the base address of the AXI2APB section to obtain the correct base address (to be used by the master) of the peripheral.

**Example:** If the AXI2APB segment is located at its default address `0xF000_0000`, the CREG base address within the memory map of the master is `0xF000_1000`.

Table 32 Peripheral Memory Map

Peripheral	Address Offset	Aperture [Bytes]	Description
CGU	<code>0x0000_0000</code>	4096	Clock Generation Unit
CREG	<code>0x0000_1000</code>	4096	Control Registers
ICTL	<code>0x0000_2000</code>	128	Interrupt Controller
GPIO	<code>0x0000_3000</code>	128	General Purpose I/O
UART	<code>0x0000_5000</code>	128	UART

This chapter is intended for programmers of the AXC003 CPU Card. It includes an overview of the example applications provided with the AXC003 CPU Card and explains how to use the AXS103 Software Development Platform for software development.

## 8.1 Supported Tools and Operating Systems

For an overview of the supported tools and operating systems, refer to the *Release Notes* document at the ARC SDP download webpage [4].

## 8.2 Boot Modes

### 8.2.1 Common Boot Modes

All the ARC cores on the AXC003 Processor FPGA are configured to halt after reset. Hence, after a reset the ARC cores go into the halt state and must be started explicitly before they can start executing the boot code. Each of the ARC cores can be started individually in one of the four following ways:

- Starting the ARC core with the debugger
- Starting the ARC core using a CPU Start button on the Mainboard (HW)
- Starting the ARC core using a CREG register bit (SW)
- Starting the ARC core autonomously after a reset

When an ARC core is started, it starts fetching instructions from the reset vector location. The default reset vector locations for the ARC cores are as follows: <sup>1</sup>

- ARC HS36                                   0x0000\_0000
- ARC HS38x2                               0x0000\_0000

To ensure maximum flexibility the ARC cores can boot from different boot sources and from different locations within a certain boot source. For this purpose, each 256 MB aperture of the memory map can be designated as a *boot mirror*.

<sup>1</sup> the reset vector address is programmable at run time through the INT\_VECTOR\_BASE register, and may be set to any 1KB aligned address.



- The boot-mirror configuration options are described in [Usage of the Mainboard DIP Switches](#) on page 59.
- The default values of the boot-mirror switches are listed in [Default Boot-Mode Settings on the ARC SDP Mainboard](#) on page 16.

For the AXC003 Processor FPGA the following *basic* boot scenarios are anticipated:

#### boot with debugger

- 1) Select boot mode and boot mirror via the CPU DIP switches on the Mainboard.
  - Boot mode is “start ARC core with the debugger”
  - Boot mirror is set to DDR3 SDRAM or local SRAM
- 2) Push the RESET button on the Mainboard
- 3) Download code into the selected boot source with the debugger
- 4) Start the ARC core with the debugger

#### boot without debugger

- 1) Select boot mode and boot mirror via the CPU DIP switches on the Mainboard
  - Boot mode is “start ARC core by CPU Start button”
  - Boot mirror is set to “AXI tunnel”
- 2) Push the RESET button on the Mainboard
- 3) Download code into the configured boot source (i.e. into the SPI Flash)
- 4) Start the ARC core by pushing the corresponding CPU Start button on the Mainboard

---

## 8.2.2 ARC HS36 Booting from ICCM0

The ARC HS36 core supports 256KB of Instruction Closely Coupled Memory (ICCM0). The ICCM0 provides the processor with fast and predictably timed access to a fixed-size region of memory, which normally contains program code. After a reset, the base address of the ICCM0 is located at address `0x1000_0000`. The default reset vector is set to the address `0x0000_0000`. Direct booting from ICCM0 is therefore not supported.

During program execution the ICCM0 base address and the reset vector can be re-programmed, and may be mapped to any 256MB-aligned aperture in the memory map.

The ARC HS36 core does not allow other initiators in the system to access its ICCM0. For initializing the ICCM0 you can use the debugger (see below) or write a boot loader application that copies the code into ICCM0 and re-programs the reset vector.

Besides the basic boot scenarios “boot with debugger” and “boot without debugger” described in [Common Boot Modes](#), the ARC HS36 also supports the following boot option:

### boot from ICCM0 with debugger

- 1) Select boot mode and boot mirror via the CPU DIP switches on the Mainboard
  - Boot mode is “start ARC core with the debugger”
  - Boot mirror is arbitrary
- 2) Push the RESET button on the Mainboard
- 3) Download code into the ICCM0 with the debugger
- 4) Re-program the reset vector to address `0x1000_0000` (i.e. reset vector now points to ICCM0)
- 5) Start the ARC core with the debugger

---

## 8.3 Pre-Boot

---

### 8.3.1 Pre-Boot Overview

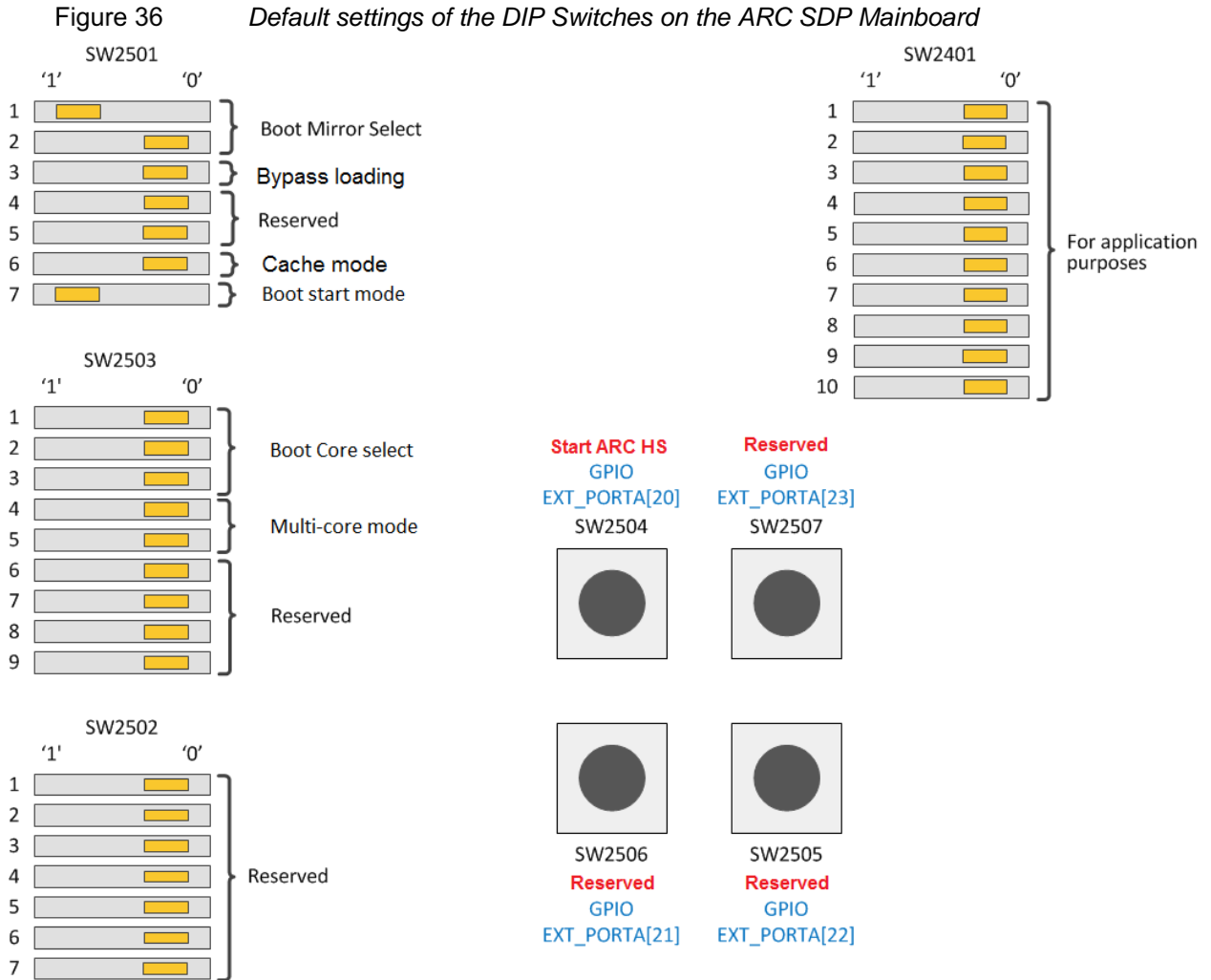
The AXS103 Software Development Platform includes a set of pre-bootloaders, which are functionally identical but compiled for the different cores. The pre-bootloader performs two main tasks:

- Board initialization
- Loading an image from SPI flash

The pre-bootloaders are included in the bitfile for the FPGA of the AXC003 CPU Card. A RAM, which is implemented in this FPGA, gets initialized with the Pre-Bootloader and is then used as boot memory.

Booting with the pre-bootloader makes use of the boot mirror mechanism and requires that the CPU DIP Switches on the Mainboard are set as shown in [Figure 36](#) on page 75:

The boot-mirror select switches need to be set to “Pre-Bootloader ROM on AXC003 CPU Card.”



During the board initialization, the pre-bootloader programs the clock dividers and the system memory map and initializes the DDR3 SDRAM.

The pre-bootloader supports loading an application image from the SPI-flash on the ARC SDP Mainboard into the SRAM or the DDR3 SDRAM memory of the AXC003 CPU Card. See the [“Building Bare-Metal Application”](#) section for instructions on creating the image and for storing the image in the SPI flash.

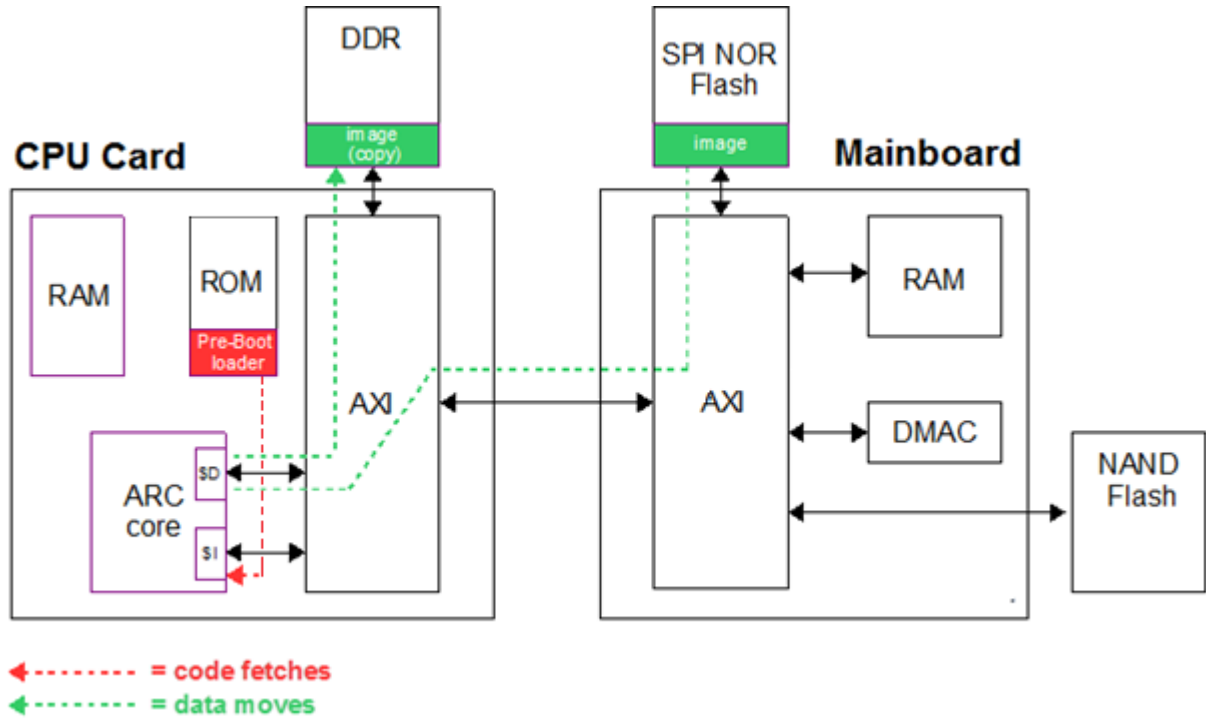
Loading an image can be bypassed by setting bit SW2501[3] to 0. In that case the pre-bootloader does not load any image, but only performs a board initialization and sets the ARC core into the HALT state.

**Note** This operation mode is useful if you want to load your application using the debugger, but want the board to be initialized automatically. In that case, set the *Boot Mirror Select* switches to “Internal ROM,” set the *Boot Mode Select* bits to “Autonomously” and bypass the image loading. See [Usage of the Mainboard DIP Switches](#) on page 59 for the exact switch settings.

If loading an image is not bypassed, the last instruction of the Pre-Bootloader is a jump to the start address of the loaded image, such that the loaded application starts execution.

As an example Figure 37 shows how the pre-bootloader loads an image from the SPI-flash on the ARC SDP Mainboard to the DDR3 SDRAM on the AXC003 CPU Card.

Figure 37 Pre-Boot Mechanism



The pre-bootloader uses the seven-segment displays on the Mainboard to show status information. The left character shows the CPU number that is also used in the debugger as listed in Table 33. If loading of an image is bypassed (SW2501[3]), a dot is displayed next to the CPU number.

The right character shows an error code as explained in Table 34 on page 77.

Table 33 Meaning of the Left Character of the Seven-Segment Display

CPU Number	ARC core	Pre-Bootloader Mode
1	ARC HS36	image loaded and code execution started
1.		no image loaded
2	ARC HS34	image loaded and code execution started
2.		no image loaded
3	ARC HS38x2: core 0	image loaded and code execution started
3.		no image loaded
4	ARC HS38x2: core 1	image loaded and code execution started
4.		no image loaded

. (dot)		Pre-Bootloader not yet executed. Check the DIP switch settings and press a CPU start button.
---------	--	--

Table 34 Meaning of the Right Character of the Seven-Segment Display

Error Code	Description
0	No error
1	DDR3 SDRAM initialization error
2	SPI initialization error: SPI IP is not detected
3	CGU lock error
4	SPI-Flash error: Cannot read the Flash ID
5	SPI-Flash error: Problem reading data
6	SPI-Flash error: Incorrect Flash ID
7	SPI-Flash error: Cannot select next 128 Mbit segment (Problem related to 4 <sup>th</sup> address byte)
8	No valid image in SPI Flash for the ARC core running the Pre-Bootloader
9	CRC error: after copying the image to the target memory location the calculated CRC differs from the one in the header
blank	When the right character is blank (off) but the left character shows a digit, the pre-bootloader hangs. Push the RESET button again

Additional board settings are performed by the **board\_init()** function from the file `/software/baremetal/board/axs103/src/board_axs103.c`, which should be executed at the start of every application.

## 8.4 Drivers

The AXS103 Software Development Platform includes drivers for bare-metal applications and MQX.

For a list of available drivers, see the release notes at the ARC SDP download webpage [\[4\]](#).

### 8.4.1 Drivers for Bare-Metal Applications

Drivers for bare-metal applications are included in the file `axs103_software_<version>.zip`, which is available from the ARC SDP download webpage [\[4\]](#). Download and unzip this file. The drivers are located in the `/software/baremetal/io` subdirectory.

Within the `/io` directory:

- Subdirectories with the `_axs1xx` postfix contain specific drivers for IP located in the peripheral subsystem on the ARC SDP Mainboard.
- Subdirectories with the `_axc003` postfix contain specific drivers for peripherals on the AXC003 CPU Card.
- The remaining directories contain generic drivers suited for peripherals located on the ARC SDP Mainboard or a CPU Card (or both).

Each driver directory contains a `makefile` and two subdirectories: `/inc` and `/src`. They contain `include` files and driver source code respectively.

See [Building Bare-Metal Applications](#) on page 83 for more details about the remaining content of this zip file.

## 7.4.2 Drivers for MQX

Peripheral drivers are included in the pre-built MQX library, which is contained in the file `axs103_software<version>.zip`. This zip file is available on the ARC SDP download webpage [4]. After unzipping this file, you can find MQX-related files in the `/software/mqx<mqx_version>` directory. Some examples for using MQX peripheral drivers are included in the zip file as well; see [MQX Package](#) on page 96. A detailed description of the drivers is contained in the documentation provided with the MQX product.

## 8.5 Bare-Metal Package

Before using the bare-metal package, make sure that you have installed the MetaWare toolchain (compiler/linker/debugger). This is a separate product, and is not part of the AXS103 Software Development Platform.

### 8.5.1 Overview

The bare-metal package is part of the zip file `axs103_software_<version>.zip`, which is available on the ARC SDP download webpage [4]. The package includes bare-metal example applications, peripheral drivers and the corresponding makefiles. Unzip the file and change to the folder `/software/baremetal`. The directory structure of the `baremetal` folder is shown in Table 35.

Table 35 *baremetal Folder Contents*

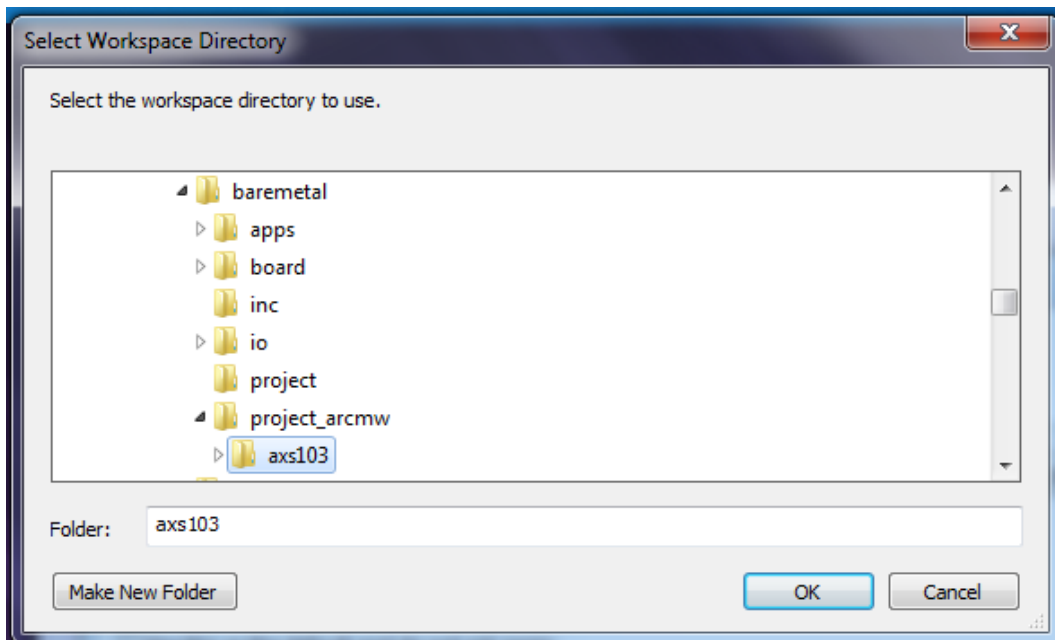
Folder	Description
Root folder path:	<code>/software/baremetal</code>

/apps	This folder contains individual subdirectories for all application examples. See the release notes on the ARC SDP download webpage [4] for an overview of the available application examples.
/board	This folder contains board-specific header files. Also, it contains linker files that specify the definition of the memory map, located in the /board/axs103/src/ folder. Two linker files are included, ap_axs103_ddr.met and map_axs103_ram.met, for building the code for the DDR3 SDRAM or local SRAM.
/inc	This folder contains a general header file for type definitions
/io	This folder contains all basic drivers for the AXS103 Software Development Platform. See the <a href="#">Drivers for Bare-Metal Applications</a> on page 77 for details.
/project	This folder contains the files related to the <code>gmake</code> build flow, including the following: <ul style="list-style-type: none"> <li>• <code>rules.mk</code> contains all makefile rules</li> <li>• <code>options.mk</code> contains more compiler/linker related options</li> </ul>
/project_arcmw	This folder contains files related to the MetaWare IDE flow.
build.bat	This batch script can be used to build the bare-metal applications using the <code>gmake</code> build flow. See <a href="#">Building Bare-Metal Applications Using gmake</a> on page 83 for details.

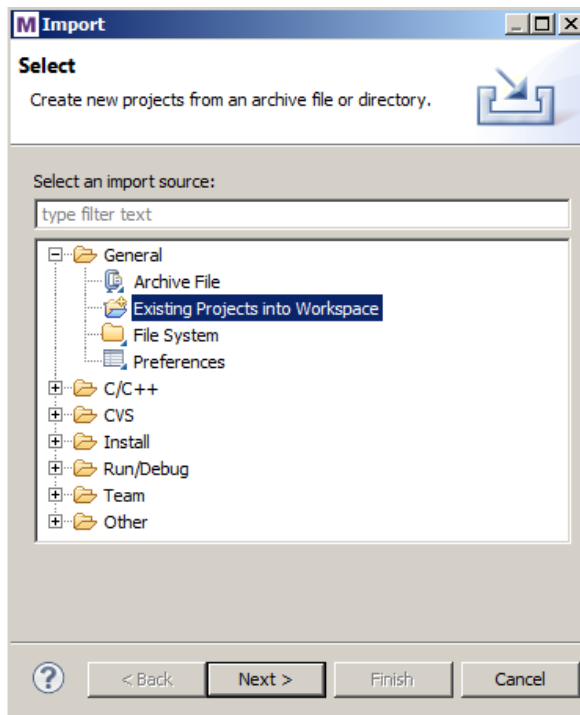
## 8.5.2 Building Bare-Metal Applications Using the MetaWare IDE

The files belonging to the example projects have to be imported in the Eclipse IDE workspace. The instructions below illustrate this process step-by-step:

1. Run MWIDE and select `/software/baremetal/project_arcmw/axs103` as the current workspace.

Figure 38 *MetaWare IDE - Select Workspace Directory*

2. Open the workspace, and select **File – Import** from the top menu.
3. Expand the **General** folder, then select **Existing Projects into Workspace** and click the **Next** button

Figure 39 *MetaWare IDE – Importing Existing Projects*

4. Select the `/software/baremetal/project_arcmw/axs103` folder as the root directory.

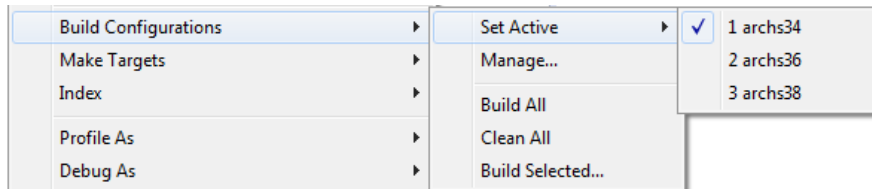


5. Select all projects available there for import, and click **Finish**.

The IDE loads and displays the example projects in your workspace.

6. Right-click one or multiple projects (excluding the `common` project).
7. Select **Build Configurations > Set Active** and select your build target, for example **archs36**.

Figure 40 *MetaWare IDE - Set Active Build Configurations*



This step selects the `archs36` configuration of the project(s) for building. You can choose another option to match your debug target.

8. By default the project is built for
  - code execution from DDR (AXS\_MEMORY\_TYPE=ddr)
  - console UART via USB data port (AXS\_CONSOLE\_TPYPE=usb\_uart)

If you wish to set other options, edit the file

`/software/baremetal/project_arcmw/axs103/common/settings.mk` and set the variables `AXS_CONSOLE_TYPE` and `AXS_MEMORY_TYPE` according to your requirements. Table 36 lists the available options. Changes that are made in this file affect all projects in the workspace.

Table 36 *Build Options*

Variable	Value	Description
AXS_MEMORY_TYPE	ddr	Image executed from DDR3 SDRAM; default. The code is built using the start address 0x8000_0000.
	ram	Image executed from local SRAM. The code is built using the start address 0x2000_0000.
	ccm	Available only for ARC HS36. Image executed from ICCM from 0x1000_0000; DATA segment is located to DCCM starting from 0xC000_0000.

Variable	Value	Description
AXS_CONSOLE_TYPE	uart0	Debug console is connected to the UART0 interface at the DB9 connector <sup>1)</sup> or at the Pmod connector <sup>1) 2)</sup>
	uart1	Debug console is connected to the UART1 interface at the 6-pin header <sup>1)</sup> or at the Pmod connector <sup>1) 2)</sup>
	usb_uart	Debug console is connected via the USB Dataport <sup>1)</sup> (using the UART2 peripheral); default
	nouart	No console output

1) This connector is located on the ARC SDP Mainboard.

2) In order to use the Pmod connectors, the `PMOD_MUX_CTRL` register (see the ARC SDP Mainboard User Guide [5]) needs to be modified.

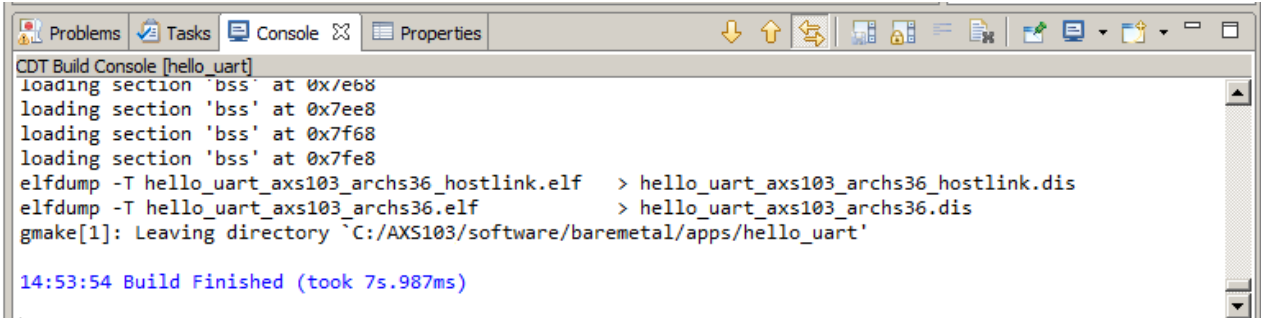
9. Build the project(s) using any of the following methods:

- Right-click the selected project(s) again and select **Build Project**.
- Select **Build Project** from the **Project** menu.
- Enter CTRL-B.

You can see the build results in the console window.

An example is shown in Figure 41. Two images get created. One of them includes HOSTLINK, the other doesn't. The file name of the image with HOSTLINK includes the string `_hostlink`. The images are located in the folder `/software/baremetal/apps/<project_name>`.

Figure 41 MetaWare IDE – Build Results in Console Window



```

CDT Build Console [hello_uart]
loading section '.bss' at 0x/e68
loading section '.bss' at 0x7ee8
loading section '.bss' at 0x7f68
loading section '.bss' at 0x7fe8
elfdump -T hello_uart_axs103_archs36_hostlink.elf > hello_uart_axs103_archs36_hostlink.dis
elfdump -T hello_uart_axs103_archs36.elf > hello_uart_axs103_archs36.dis
gmake[1]: Leaving directory `C:/AXS103/software/baremetal/apps/hello_uart'

14:53:54 Build Finished (took 7s.987ms)

```

- For instructions on running the code in the MetaWare IDE Debugger, see [Hardware Setup for Debugging](#) on page 85 and [Running a Bare-Metal Application in the MetaWare IDE Debugger](#) on page 87.
- For instructions on storing the code in the SPI Flash memory of the ARC SDP Mainboard, see [Storing an Image in the SPI Flash and Running the Application](#) on page 94.

### 8.5.3 Building Bare-Metal Applications Using gmake

The batch script `build.bat` can be used to build the applications. This script is based on `gmake`. It executes `gmake`, passing variables according to the command line options provided. Figure 42 and Table 37 explain the available options of this script.

Figure 42 *Build Script Options*

```
c:\_axs103\software\baremetal>build -h
=====
Usage: build -app <appname> -core <coretype>
      -console <consoletype>
      -memory <memorytype>

Arguments:
  <appname>      : application name
  <coretype>     : core type      <archs36|archs38>
  <consoletype> : console type   <uart0|uart1|usb_uart|nouart>
  <memorytype>  : memory type   <ccm|ram|ddr>
```

Table 37 *Command Line Options for build.bat*

Option	Value	Description
-app		The name of the subdirectory of the <code>/software/baremetal/apps</code> directory, which contains the makefile for the selected application
-core	archs36	Select ARC HS36 core
	archs38	Select ARC HS38 core
-console	uart0	Debug console is connected to the UART0 interface at the DB9 connector <sup>1)</sup> or at the Pmod connector <sup>1) 2)</sup>
	uart1	Debug console is connected to the UART1 interface at the 6-pin header <sup>1)</sup> or at the Pmod connector <sup>1) 2)</sup>
	usb_uart	Debug console is connected using the USB data port <sup>1)</sup> (using the UART2 peripheral); default
	nouart	No console output
-memory	ddr	Image to be executed from DDR3 SDRAM; default The code is built using the start address <code>0x8000_0000</code> .
	ram	Image to be executed from local SRAM The code is built using the start address <code>0x2000_0000</code> .
	ccm	Enabling this option in combination with <code>archs36</code> emulates the ARCV2HS34 CPU. The code is built using start address <code>0x1000_0000</code> , data sections are mapped to DCCM starting from <code>0xC000_0000</code>

- 1) *This connector is located on the ARC SDP Mainboard.*
- 2) *In order to use the Pmod connectors, the `PMOD_MUX_CTRL` register (see the ARC SDP Mainboard User Guide [5]) needs to be modified.*

To build an application example, do the following:

1. Open a Windows command prompt
2. Navigate to the `/software/baremetal` folder
3. Enter the `build` command followed by with the desired options.

The result of the build can be found in the `/software/baremetal/apps/<app_name>` folder. The build generates seven types of files:

<code>*.o</code>	Object
<code>*.a</code>	Archive library
<code>*.dis</code>	Disassembly file
<code>*.elf</code>	Elf file for debugger
<code>*.bin</code>	Bin file for SPI-Flash or SD-Card
<code>*.hex</code>	Hex file
<code>*.map</code>	Map file

Two sets of files are generated: One set includes `hostlink`, the other doesn't. For the set with `hostlink`, the string `_hostlink` is appended to the file name.

If the arguments for `-console` or `-memory` differ from the default values, the selected option is included in the filename as shown in the following examples:

#### **Example 1: Build the selftest Application with Default Arguments**

```
build -app selftest
```

This command generates the files

```
selftest_axs103_archs36.*
selftest_axs103_archs36_hostlink.*
```

#### **Example 2: Build the selftest Application for a Different ARC core**

```
build -app selftest -core archs38
```

This command generates the files

```
selftest_axs103_archs38.*
selftest_axs103_archs38_hostlink.*
```

#### **Example 3: Build the selftest Application for a Different Memory and Console**

```
build -app selftest -memory ram -console uart0
```

This command creates the files

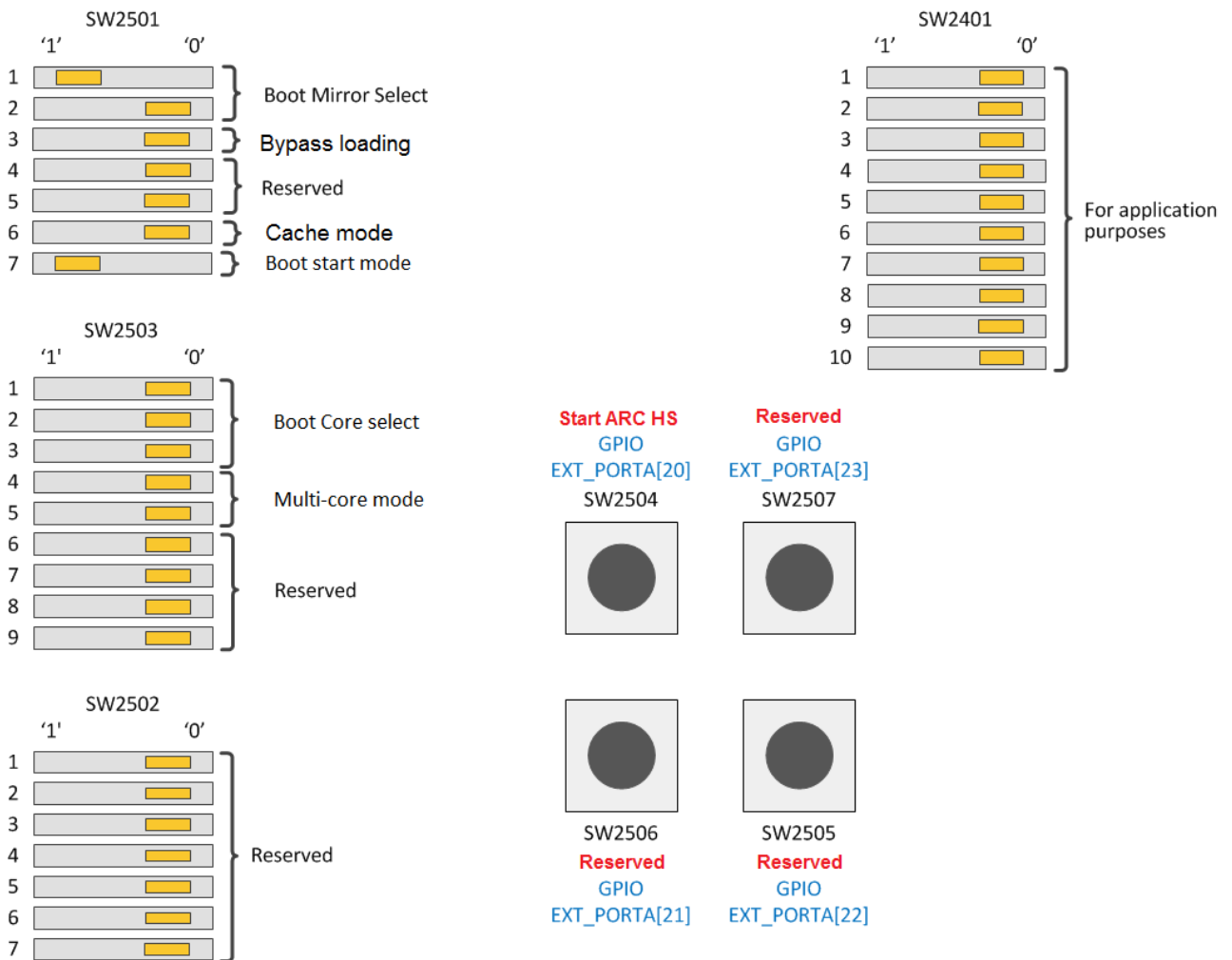
```
selftest_axs103_ram_uart0_archs36.*
selftest_axs103_ram_uart0_archs36_hostlink.*
```

### 8.5.4 Hardware Setup for Debugging

Follow the steps below to execute a sample application in a debugger:

1. Set the jumpers to their default settings (see [Jumpers](#) on page 31). The JTAG interface is in daisy-chained mode.

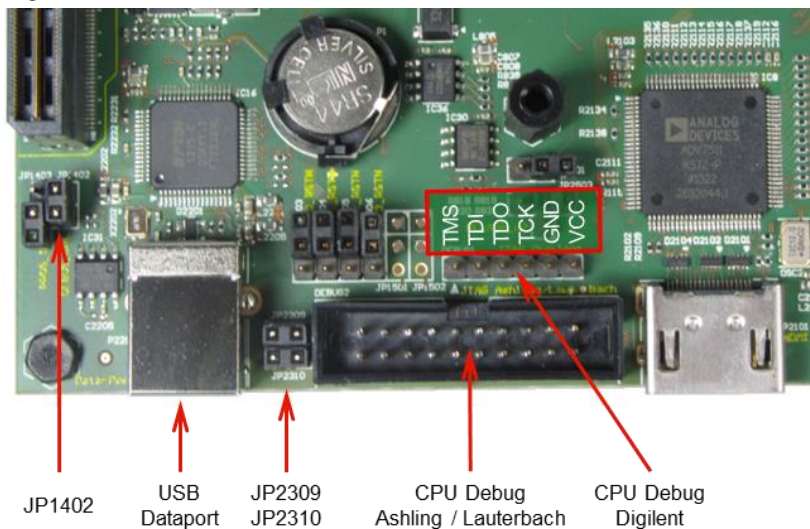
Figure 43 Settings of the DIP Switches on the ARC SDP Mainboard for Using the Debugger



2. Connect the USB data port of the ARC SDP Mainboard to your PC.
3. If you are using a debug probe rather than the JTAG channel of the USB data port, connect the probe to the appropriate connector and remove the jumper JP1402 on

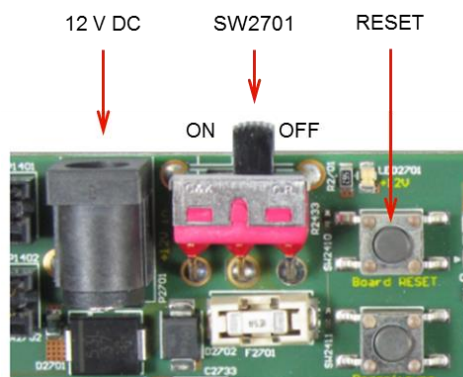
the ARC SDP Mainboard. If you are using a Lauterbach probe, also remove the jumpers JP2309 and JP2310.

Figure 44 Location of the Debug Interfaces and the Corresponding Jumpers



- Switch on the power supply or push the `RESET` button.

Figure 45 Location of the ARC SDP Mainboard's Power Supply and Power Switch



- Push the `CPU Start` button of your target CPU core according to [Table 38](#) on page 87. For the location of the `CPU Start` button see [Figure 46](#) on page 87. The pre-

bootloader then automatically initializes the DDR3 SDRAM memory and sets the ARC core frequency correctly, but bypasses loading an image from the SPI Flash.

Figure 46 Location of the CPU Start Buttons on the ARC SDP Mainboard.

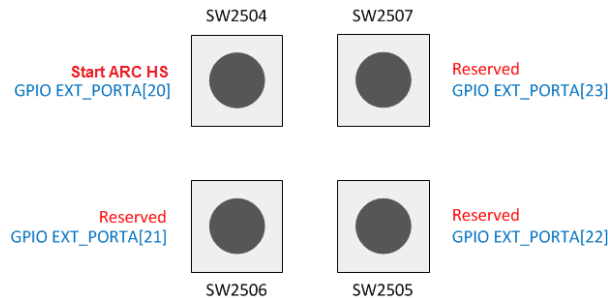


Table 38 CPU Start Buttons and Display Values for Running Applications in the Debugger

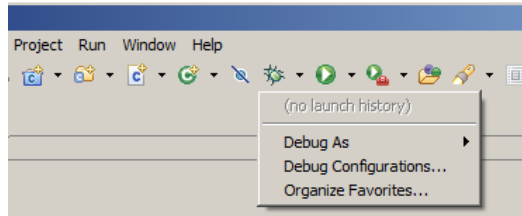
ARC Core	Start Button	Seven-Segment Display
ARC HS36	SW2504	1.0
ARC HS34	SW2504	2.0
ARC HS38x2: core 0	SW2504	3.0
ARC HS38x2: core 1	SW2504	4.0

When the seven-segment display shows the value listed in Table 38 (note the dot between the two digits) the AXS103 Software Development Platform is ready for loading and executing your application.

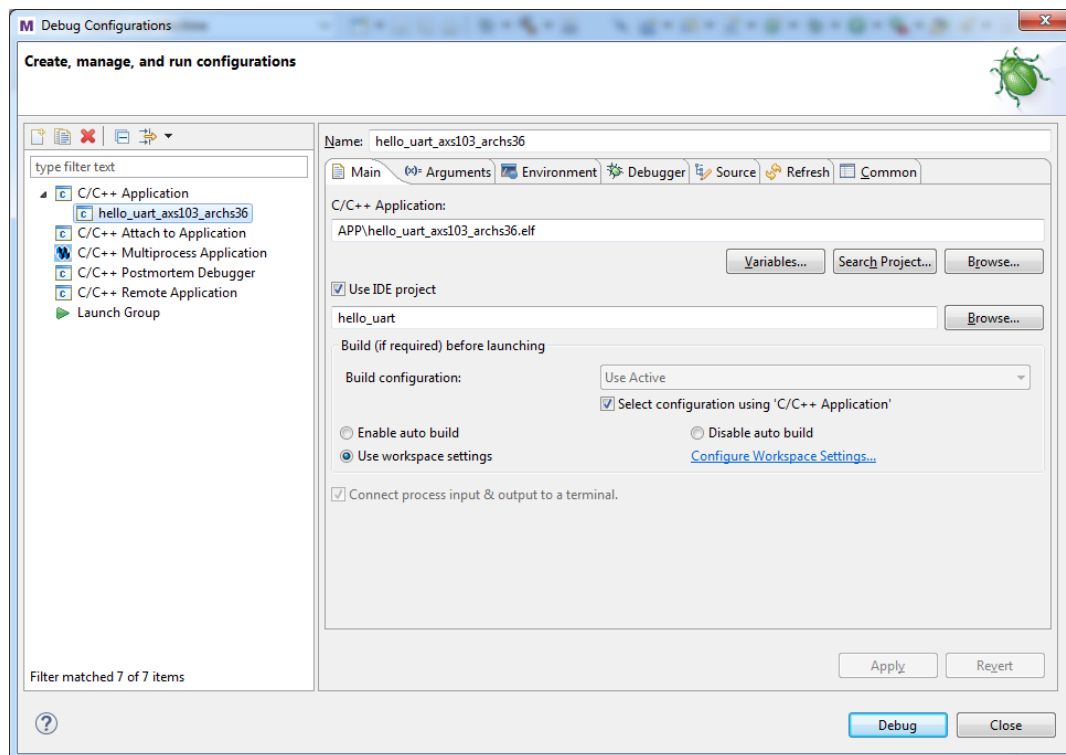
### 8.5.5 Running a Bare-Metal Application in the MetaWare IDE Debugger

After you successfully build the C Project, you can debug the executable on the AXS103 Software Development Platform. This section provides step-by-step instructions how to configure and run a debug session in the MetaWare IDE.

1. Perform the hardware setup as described in [Hardware Setup for Debugging](#) on page 85.
2. Open a hyperterminal (such as PuTTY) on your PC and select the COM port that is connected to the USB data port of the ARC SDP Mainboard.
3. Set the **Connection type** to **Serial** and the **Speed** to **115200**.
4. Launch the MetaWare IDE and open the workspace `/software/baremetal/project_arcmw/axs103`
5. Select **Debug Configurations** from the **Run** menu or by clicking on the down arrow next to the bug icon:



6. Double click **C/C++ Application** to create a new debug configuration for the project or select an existing debug configuration.
7. Select the **Main** tab and enter a name of your choice in the **Name** field. It is best to compose the name from the project name and the ARC core.
8. Enter the name of the ELF-file (with or without `_hostlink`) in the **C/C++ Application** field.

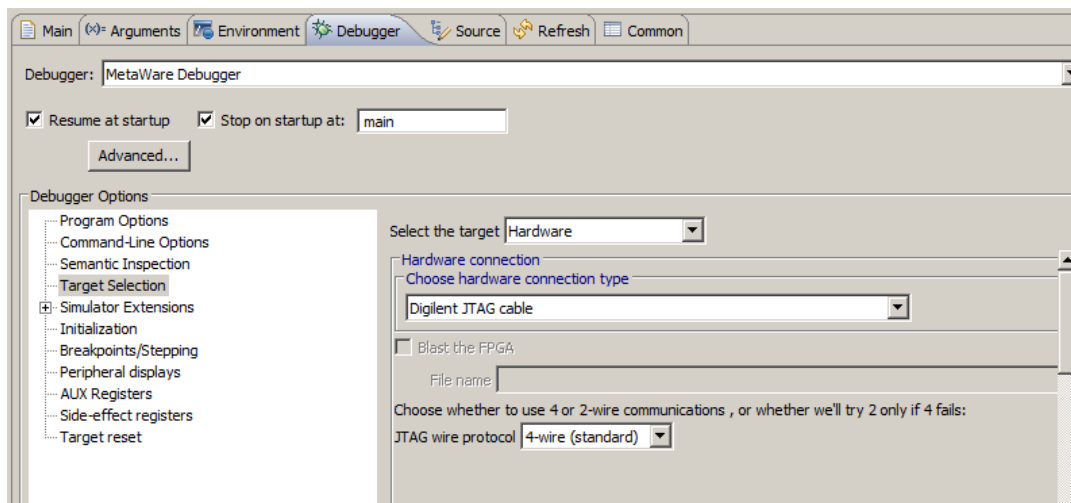


9. Click the **Debugger** tab.
10. Configure the target to use **Hardware** and connect the hardware the Digilent JTAG cable.



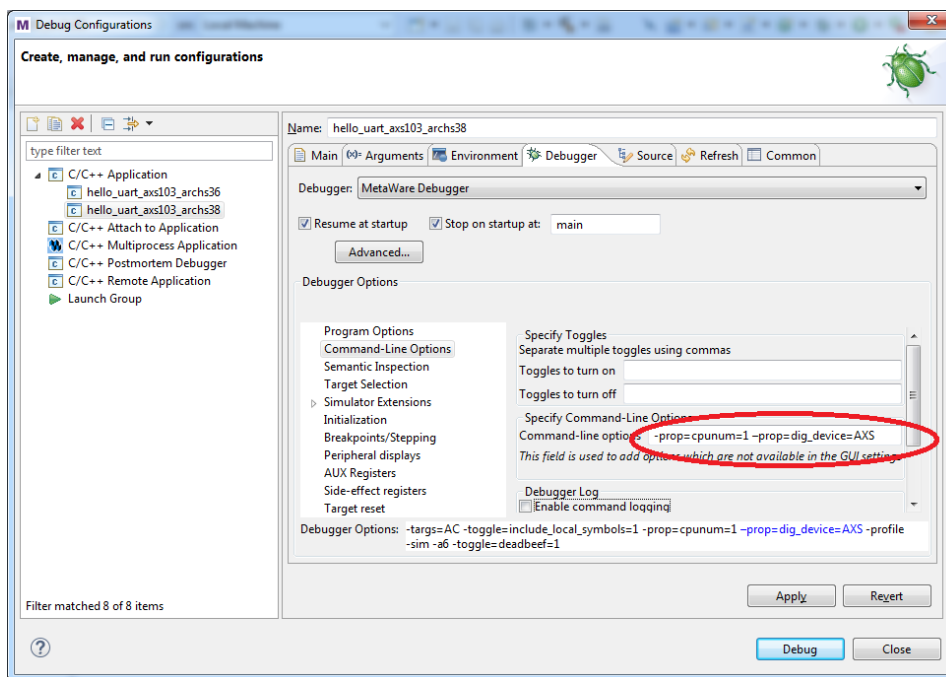
An ELF file must be selected before you can select the target.





11. In **Command-line Options > Debugger Options**, set a property to select the correct core and device:

- `-prop=cpunum=1`                      Select ARC HS34 core
- `-prop=cpunum=1`                      Select ARC HS36 core
- `-prop=cpunum=1`                      Select ARC HS38x2 core 1
- `-prop=cpunum=2`                      Select ARC HS38x2 core 2
- `-prop=dig_device=AXS`              Select the USB data port



If you are using a Digilent probe (rather than the USB data port) set `-prop=dig_device=JtagHs1` or

`-prop=dig_device=JtagHs2`  
depending on the type of your probe.

- Click the **Debug** button in the **Debug configurations** dialog.

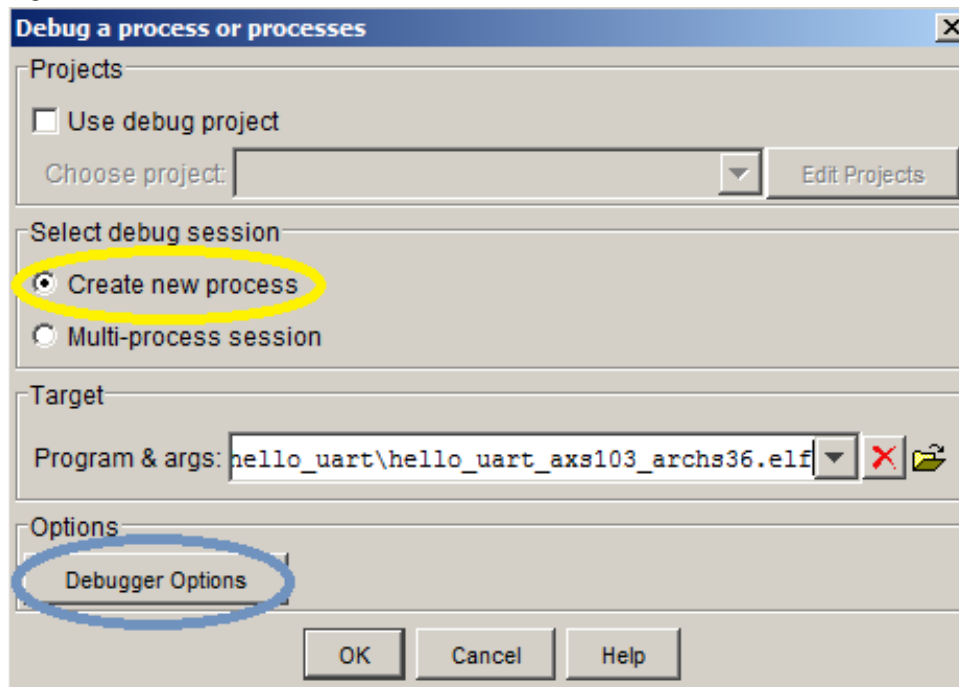
The IDE switches to the **Debug** perspective and initiates the debug session.

## 8.5.6 Running a Bare-Metal Application in the MetaWare Debugger

This section describes how to execute an image with the MetaWare debugger. This example assumes that the application `hello_uart` has been built for execution from the DDR3 SDRAM. The result of building is a file with the extension `.elf`. Use this file for the debugger.

- Perform the hardware setup as described in [Hardware Setup for Debugging](#) on page 85.
- Open a hypertext terminal (such as PuTTY) on your PC and connect to the COM port that is connected to the USB data port of the ARC SDP Mainboard.
- Set the **Connection type** to `Serial` and the **Speed** to `115200`.
- Start the MetaWare debugger.
- Select **Create new process** (yellow ellipse) and open **Debugger Options** (blue ellipse) as shown in Figure 47:

Figure 47 Creating a New Process



6. In **Command-line Options > Debugger Options**, enter a property to select the correct core and device:

`-prop=cpunum=1`                      select the ARC CPU (see Table 39).  
`-prop=dig_device=AXS`            selects the USB data port

Table 39 Property Arguments for Selecting the CPU Core in the Debugger

Option	Description
<code>-prop=cpunum=1</code>	Select ARC HS36 core
<code>-prop=cpunum=1</code>	Select ARC HS38x2 core 1
<code>-prop=cpunum=2</code>	Select ARC HS38x2 core 2

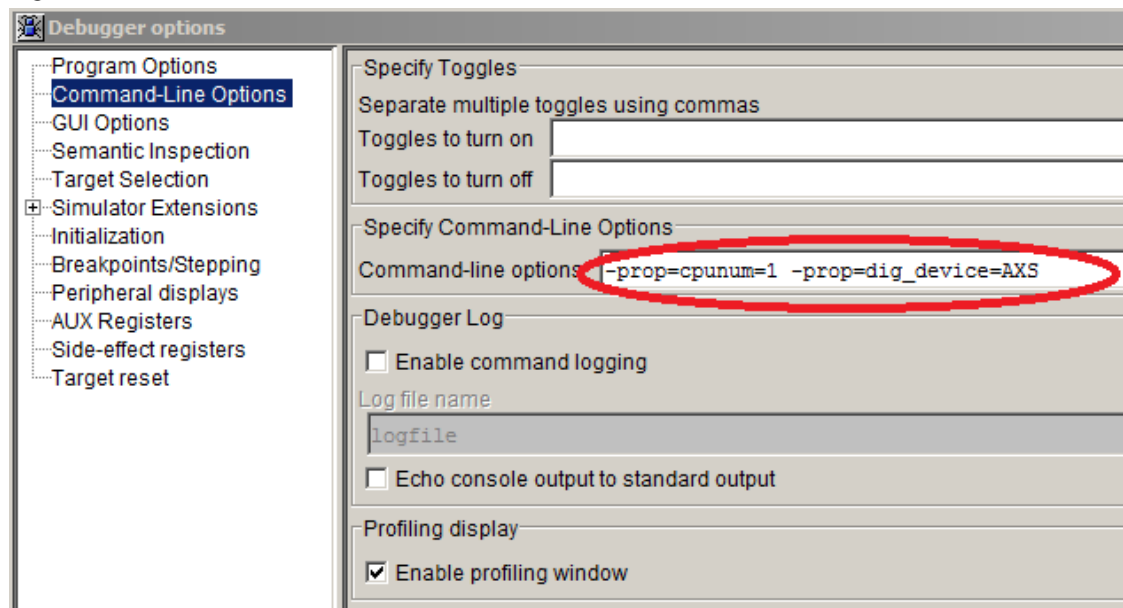
If you are using a Digilent probe (rather than the USB data port) set one of the following options depending on the type of your probe:

`-prop=dig_device=JtagHs1`

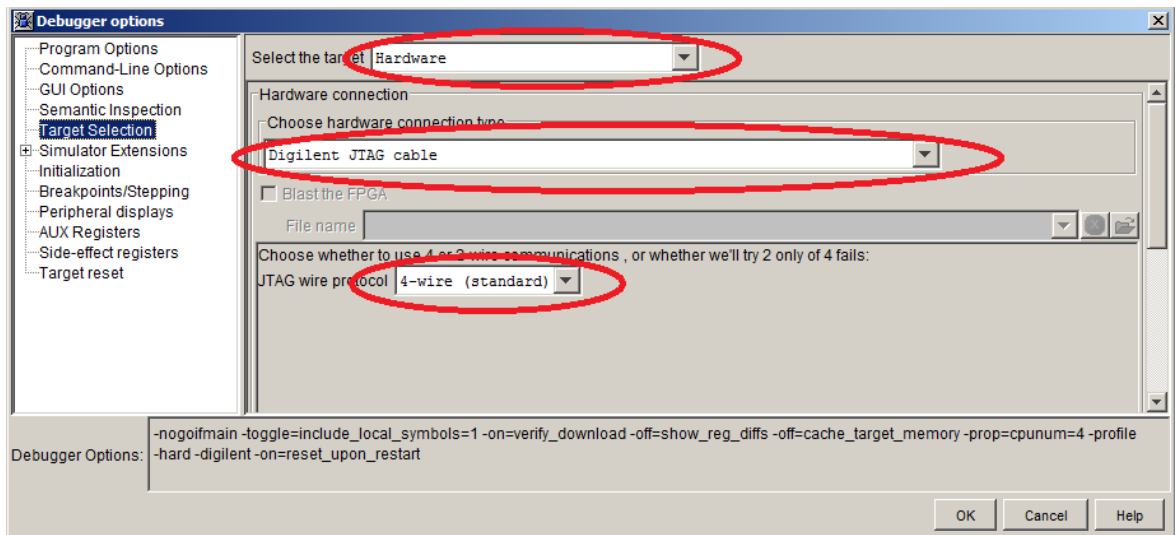
or

`-prop=dig_device=JtagHs2`

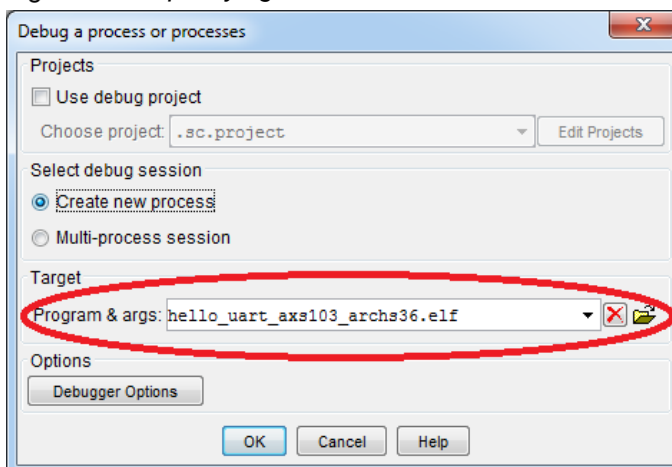
Figure 48 Debugger options – Command-Line Options



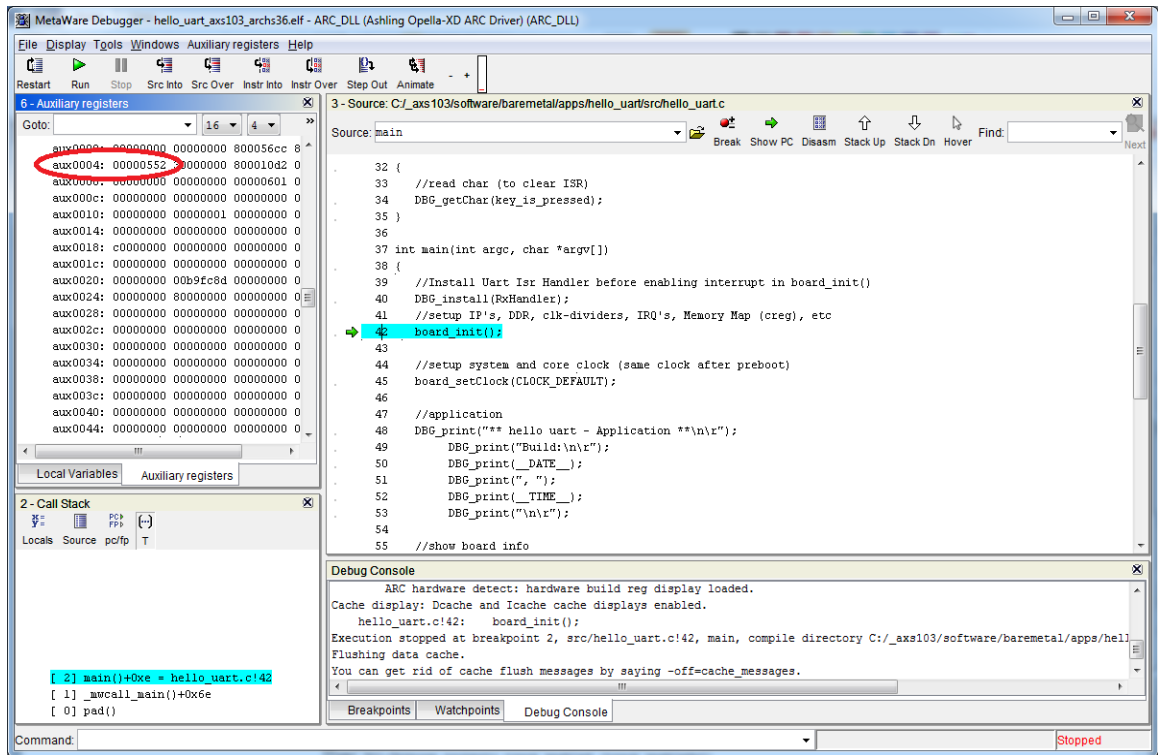
7. In **Debugger Options** sub-tab **Target Selection**, select the correct settings for your debug probe. Figure 49 on page 92 explains this using the example of the USB data port: Select the **Digilent JTAG cable** option and set the **JTAG wire protocol**:

Figure 49 *Debugger Options – Target Selection*

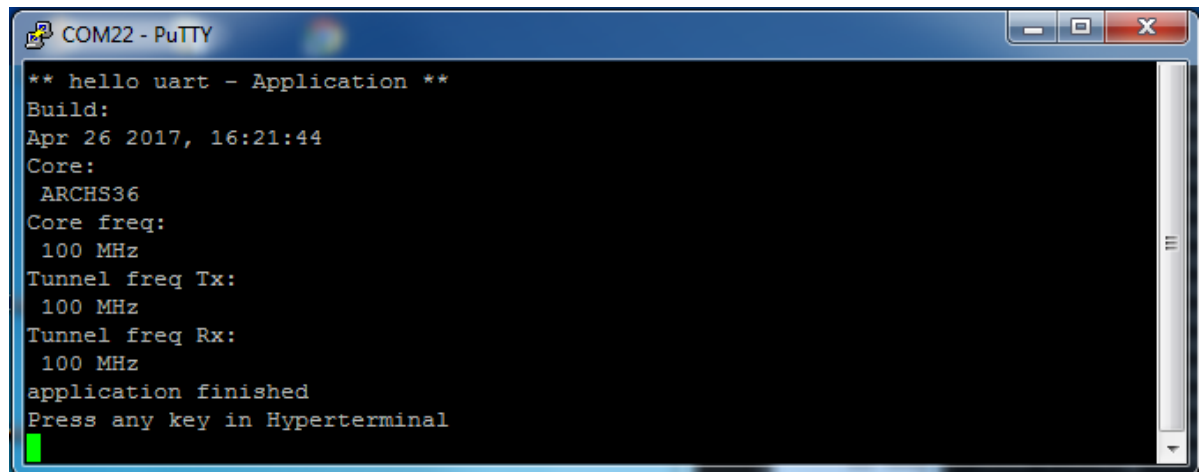
8. Back in the **Debug a process or processes** window, select the correct `.elf` file and press **OK**:

Figure 50 *Specifying a Path to the .elf File*

The debugger is now ready for executing the image. In the auxiliary register `AUX0004`, the ARC HS36 ID is visible (red ellipse, `0x553`). The program counter points to the **board\_init()** function, which should be called at the start of every application.

Figure 51 *Debugger Status*

9. Execute the program by clicking **run** button in the debugger.
10. Observe the output in the console, which should be similar to the screenshot shown in Figure 52:

Figure 52 *HyperTerminal Output*


```

COM22 - PuTTY
** hello uart - Application **
Build:
Apr 26 2017, 16:21:44
Core:
ARCHS36
Core freq:
100 MHz
Tunnel freq Tx:
100 MHz
Tunnel freq Rx:
100 MHz
application finished
Press any key in Hyperterminal

```

### 8.5.7 Storing an Image in the SPI Flash and Running the Application

The steps below explain the process of storing an image for the SPI Flash and running the application. This is done using the application `hello_uart` and the ARC HS36 core as an example. If you wish to use another core, modify the instructions marked with **yellow** highlighting.

Only images without HOSTLINK should be programmed in the SPI Flash.

1. After building the image, use the `axs_comm.exe` tool to store the image in the SPI flash. First, it erases sectors 0 and 1 in section 0 of the SPI flash and then programs the image starting at sector 0. To program the `hello_uart` example for the ARC HS36 core navigate to the `/software/baremetal/apps/hello_uart/` folder and use the following command:

```
axs_comm -c 0553 -a 00000000 -p 80000000 -f
hello_uart_axs103_archs36.bin
```

The parameters used here have the following meanings:

0553	ARC ID of the ARC HS36 processor
80000000	Program is built for DDR3 SDRAM @ 0x8000_0000s so the pre-bootloader stores the image at this address.
hello_uart_axs103_	Selected BIN file (result of build flow)

```
archs36.bin
```

For images built for the SRAM use the following command instead:

```
axs_comm -c 0553 -a 00000000 -p 20000000 -f  
hello_uart_axs103_ram_archs36.bin
```

The parameters used here have the following meanings:

0553	ARC ID of the ARC HS36 processor
20000000	Program is built for SRAM @ 0x2000_0000, so the pre-bootloader stores the image at this address,
hello_uart_axs103_ archs36.bin	Selected BIN file (result of build flow)

This command erases the factory-programmed self-tests. If you wish to restore the self-tests later, see [Restoring the Self-Tests in the SPI Flash](#) on page 25.

**Note**

When flashing an image into SPI flash, the `axs_comm` tool prefixes the BIN file with a 256-byte header. Hence, the total size occupied by an image is increased by 256 Bytes. To prevent corruption of existing image(s), take the increased image size(s) into account when flashing an additional image into SPI flash


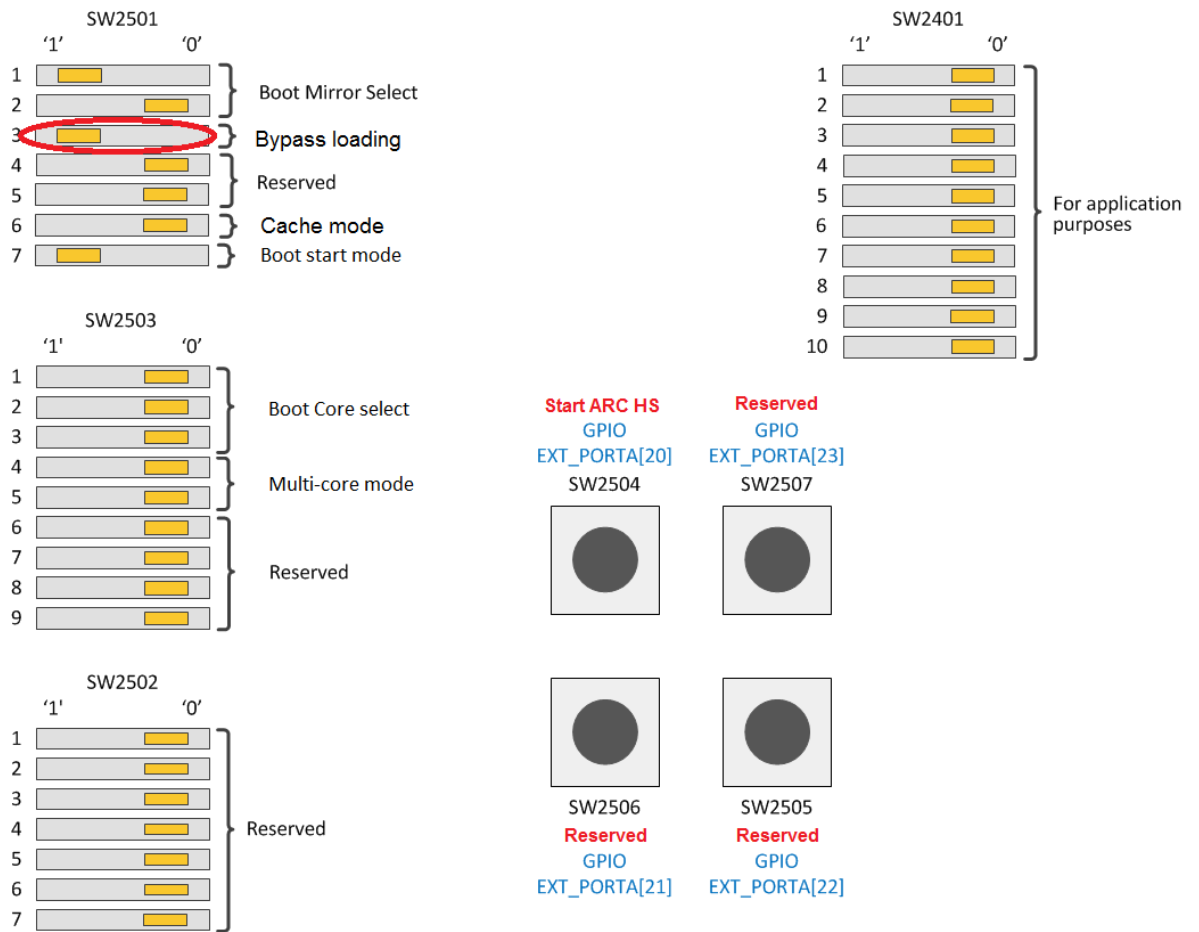
2. The typical setting is to load and execute the application autonomously after a reset. Set the DIP switches on the ARC SDP Mainboard to the default settings, then set bit 3 of the SW2501 DIP switch to the left position (  ). In this position the pre-bootloader looks for an application in SPI flash, and if it finds it the application runs. [Figure 53](#) on page 96 shows the DIP switch settings. The change compared to the default settings is marked by a red ellipse.

Figure 53 DIP Switch Settings for Autonomous Code Execution on the ARC Core



3. Push the `RESET` button on the ARC SDP Mainboard

The Pre-Bootloader detects a valid image in sector 0 of the SPI flash, loads this image in DDR3 SDRAM and executes it.

## 8.6 MQX Package

Before using the MQX package, make sure that you have installed the MetaWare toolchain (compiler/linker/debugger). This is a separate product, which is not part of the AXS103 Software Development Platform.

### 8.6.1 Overview

The MQX package is part of the zip file `axs103_software_<version>.zip`, which can be obtained from the ARC SDP download webpage [4]. The package includes a pre-built binary version of the MQX operating system, source code of BSP and peripheral drivers and example code that demonstrates how to build a simple MQX application and work with the peripheral device drivers.



Unzip the software package and change to the directory `/software/mqx<version>`. The directory structure of the `mqx<version>` folder is shown in Table 40.

Table 40 MQX folder Contents

Folder	Description
Root folder path: <code>/software/mqx&lt;version&gt;</code>	
<code>/examples</code>	<p>This folder contains an <code>axs103</code> subdirectory with application examples. Its subdirectory <code>hs38x2</code> contains examples for the ARC HS38x2 dual-core configuration.</p> <ul style="list-style-type: none"> <li><code>axs103/hs38x2/grtc</code> – demonstrates usage of the global real-time counter unit of ARConnect;</li> <li><code>axs103/hs38x2/i2c_selftest</code> – demonstrates access to AXS103 on-board RTC using the internal i2c bus;</li> <li><code>axs103/leds</code> – demonstrates usage of AXC003 seven-segment indicator and LEDs</li> <li><code>axs103/selftest</code> – self-test application</li> <li><code>axs103/eping</code> – demonstrates use of the Ethernet device driver API</li> </ul>
<code>/build</code>	This folder contains configuration files for building applications
<code>/docs</code>	MQX documentation
<code>/library</code>	MQX libraries
<code>/mkscripts</code>	This folder contains <i>make</i> scripts for building applications

## 8.6.2 Building MQX Applications Using gmake

- Set `MQX_ROOT` and `MQX_CONFIG` environment variables on the command line:

```
set MQX_ROOT=<path_to_axs103_software>\software\mqx<version>
set MQX_CONFIG=%MQX_ROOT%\build\axs103\<variant>_config.mk
```

where `<variant>` is your target HS core configuration: `hs34`, `hs36` or `hs38x2`
- Change to the application directory under `%MQX_ROOT\examples\axs103`.

```
cd <path_to_application>
```
- Build the application example:

```
gmake all
```

The output directory named `arcv2hs_<variant>.met` is created and includes the `test.elf` executable file to be run in the debugger.

### Building the grtc Application for HS38x2

```
set MQX_ROOT=C:\AXS103\software\mqx<version>
set MQX_CONFIG=%MQX_ROOT%\build\axs103\hs38x2_config.mk
cd %MQX_ROOT%\examples\axs103\hs38x2\grtc
gmake all
```

See the *DesignWare MQX RTOS Getting Started* manual for more information on how to build MQX applications

---

### 8.6.3 Hardware Setup for Debugging

See [Hardware Setup for Debugging](#) on page 85 for details.

---

### 8.6.4 Running MQX Applications in the MetaWare Debugger

The easiest way to run MQX applications in the MetaWare debugger is using the batch files in the `%MQX_ROOT%\build\axs103` directory:

- `gohs34.bat` for ARC HS34
- `gohs36.bat` for ARC HS36

These batch files invoke the MetaWare debugger with the required options either to run the application or to start the GUI for debugging

4. Add the `%MQX_ROOT%\build\axs103` to the `PATH` variable in your environment:

```
set PATH=%PATH%;%MQX_ROOT%\build\axs103
```

5. To run an application built for HS34 or HS36 configuration without starting the GUI enter the following command on the command line:

```
gohs34.bat test
```

or

```
gohs36.bat test
```

The application makefiles specify `test` as the name of the executable file

6. To start GUI for debugging of HS34 or HS36 application, execute the following command on the command line:

```
gohs34.bat test debug
```

or

```
gohs36.bat test debug
```



#### Note

To run HS38x2 dual-core application examples located in the `%MQX_ROOT%\examples\axs103\hs38x2` directory, use the `run.bat` file from the application subdirectory. When this file is invoked with no arguments it launches

---

executables on the corresponding HS38x2cores. To debug the dual-core example application in the GUI, pass `debug` as an argument to `run.bat`

---

### Running the leds Application Built for HS34

```
cd %MQX_ROOT%\examples\axs103\leds
gohs34.bat test
```

### Debugging the selftest Application Built for HS36

```
cd %MQX_ROOT%\examples\axs103\selftest
gohs36.bat test debug
```

### Running the grtc Application Built for Dual-Core HS38

```
cd %MQX_ROOT%\examples\axs103\hs38x2\grtc
run.bat
```

---

## 8.7 Linux and U-Boot Packages

Before using the Linux and U-Boot packages, make sure that you have installed the MetaWare toolchain (compiler/linker/debugger). This is a separate product, which is not part of the AXS103 Software Development Platform.

You can find the latest information about how to download Linux and U-Boot sources and build binary images you can find in the article [Getting Started with Linux on ARC AXS103 Software Development Platform \(SDP\)](#).

---

### 8.7.1 Overview

The Linux and U-Boot packages are part of the zip file `axs103_software_<version>.zip`, which is available from the ARC SDP download webpage [4]. The package includes a pre-built Linux and U-Boot binary versions.

The `axs103_uboot_v2017.01` folder contains the following:

- `u-boot_axs103.elf` - ELF file of the U-Boot bootloader.
- `u-boot_axs103.bin` - binary image of the U-Boot bootloader, to be programmed into the AXS SPI flash to autostart on power-on.
- `axs_comm_progam_uboot.bat` - batch file that flashing `u-boot_axs103.bin` into SPI flash.

The `axs103_linux_4.10.9` folder contains the following:

- `uImage_axs103` - Linux kernel prepared for loading by the U-Boot bootloader.

If you program the `u-boot_axs103.bin` file to SPI flash on AXS 103 hardware, set bit 3 on the SW2501 switch to off so that the pre-bootloader looks for the U-Boot in SPI flash. By default, this switch is on and bypasses loading the U-Boot or any application in SPI flash.

When `u-boot_axs103.bin` is programmed into the AXS SPI flash and autostarts on power-on, it attempts to find `uImage` on the first partition of the SD card.

---

## 8.7.2 Hardware Setup for Debugging

1. Select the ARC HS38 Core 0 configuration as described in [ARC HS38 Core 0 on page 18](#).
2. Perform the hardware setup as described in [Hardware Setup for Debugging](#) on page 85.
3. Open a hyperterminal (such as PuTTY) on your PC and select the COM port that is connected to the USB data port of the ARC SDP Mainboard.
4. Set the **Connection type** to **Serial** and the **Speed** to **115200**.

Additionally, you may connect:

- USB keyboard to the USB port on the board
- Ethernet cable
- HDMI Monitor

---

## 8.7.3 Executing the Linux Image with U-Boot

You can use the U-Boot bootloader for loading and starting a Linux image on an AXS103 board. U-Boot can be pre-programmed in on-board SPI flash and start automatically on power-on or you can load it using JTAG.

### 8.7.3.1 Loading U-Boot with the Debugger Using JTAG

If you are using an Ashling Opella-XD debug probe push the RESET button (SW2410) on the Mainboard and run the following command:

```
mdb -DLL=c:/AshlingOpellaXDforARC/opxdarc -prop=jtag_frequency=12MHz -
prop=jtag_optimise=1 -nooptions -run -cl -memxfersize=0x8000 u-boot_axs103.elf
```

If you are using a Digilent debug probe push the RESET button (SW2410) on the Mainboard and run the following command:

```
mdb -digilent -prop=dig_speed=10000000 -run -cl u-boot_axs103.elf
```

On execution of U-Boot you see the following in the serial console:

```
U-Boot 2017.01 (Apr 11 2017 - 11:54:33 +0300)

I2C:   ready
DRAM:  512 MiB
NAND:  0 MiB
MMC:   Synopsys Mobile storage: 0
*** Warning - bad CRC, using default environment
```

```

In:      serial0@e0022000
Out:     serial0@e0022000
Err:     serial0@e0022000
Net:
Warning: ethernet@e0018000 (eth0) using random MAC address - d2:83:38:6d:af:37
eth0: ethernet@e0018000
AXS#

```

Depending on the `bootcmd` variable, U-Boot may automatically execute the boot sequence. To prevent it press any key within three seconds to stop the countdown.

U-Boot is controlled with its specific set of commands. You can list the available commands using the `?` or `help` commands.

You can also create scripts of commands. A script with the name `bootcmd` is automatically executed after boot delay expires and may be saved into EEPROM with the `saveenv` command.

### 8.7.3.2 Linux Execution with U-Boot Started Manually

To run a Linux image from the SD-card, do the following:

1. Rename the `uImage_axs103` file to `uImage`.
2. Copy `uImage` to the SD card.
3. Insert the SD card
4. Run the following U-Boot commands:

```

setenv bootcmd fatload mmc 0\; bootm
boot

```

#### Result

You see following log in the serial console:

```

AXS# setenv bootcmd fatload mmc 0\; bootm
AXS# boot
reading uImage
13670495 bytes read in 2323 ms (5.6 MiB/s)
## Booting kernel from Legacy Image at 82000000 ...
   Image Name:   Linux-4.10.9
   Image Type:   ARC Linux Kernel Image (gzip compressed)
   Data Size:    13670431 Bytes = 13 MiB
   Load Address: 80000000
   Entry Point:  809c0000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK

```

Starting kernel ...

```

Linux version 4.10.9 (abrodkin@ru20arcgnul) (gcc version 6.2.1 20160824 (Buildroot
2017.02-00002-gce00ac569) ) #2 SMP PREEMPT Tue Apr 11 11:55:47 MSK 2017
Memory @ 80000000 [512M]

```

```

OF: fdt:Machine model: snps,axs103-smp
earlycon: uart8250 at MMIO32 0xe0022000 (options '115200n8')
bootconsole [uart8250] enabled
Freq is 100MHz
AXS: AXC003 CPU Card FPGA Date: 13-4-2017
AXS: MainBoard v3 FPGA Date: 14-4-2017
archs-intc      : 2 priority levels (default 1)

IDENTITY       : ARCOVER [0x53] ARCNUM [0x0] CHIPID [ 0x0]
processor [0]  : ARC HS38 R3.0 (ARCV2 ISA)
Timers        : Timer0 Timer1 GFRC [SMP 64-bit]
ISA Extn      : atomic ll64 unalign (not used)
               : mpy[opt 9] div_rem norm barrel-shift swap minmax swape
BPU           : full match, cache:512, Predict Table:8192
MMU [v4]      : 8k PAGE, 2M Super Page (not used) JTLB 512 (128x4), uDTLB 8,
uITLB 4, PAE40 (not used)
I-Cache       : 64K, 4way/set, 64B Line, VIPT aliasing
D-Cache       : 64K, 2way/set, 64B Line, PIPT
SLC           : 512K, 64B Line
Peripherals   : 0xe0000000, IO-Coherency
Vector Table  : 0x80000000
FPU           : SP DP
DEBUG         : ActionPoint smaRT RTT
OS ABI [v4]   : 64-bit data any register aligned
Extn [SMP]    : ARConnect (v2): 2 cores with IPI IDU DEBUG GFRC
CONFIG_ARC_FPU_SAVE_RESTORE needed for working apps
Reserved memory: created DMA memory pool at 0xbe000000, size 32 MiB
OF: reserved mem: initialized node frame_buffer@be000000, compatible id shared-
dma-pool
percpu: Embedded 6 pages/cpu @9fd64000 s16384 r8192 d24576 u49152
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 65248
Kernel command line: earlycon=uart8250,mmio32,0xe0022000,115200n8 console=tty0
console=ttyS3,115200n8 print-fatal-signals=1 consoleblank=0 video=1280x720@60
PID hash table entries: 2048 (order: 0, 8192 bytes)
Dentry cache hash table entries: 65536 (order: 5, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 4, 131072 bytes)
Memory: 504528K/524288K available (4558K kernel code, 149K rwddata, 888K rodata,
10112K init, 273K bss, 19760K reserved, 0K cma-reserved)
Preemptible hierarchical RCU implementation.
      Build-time adjustment of leaf fanout to 32.
NR_IRQS:128
MCIP: IDU referenced from Devicetree 2 irqs
clocksource: ARConnect GFRC: mask: 0xffffffffffffffff max_cycles: 0x171024e7e0,
max_idle_ns: 440795205315 ns
Console: colour dummy device 80x25
console [tty0] enabled
Calibrating delay loop... 73.72 BogoMIPS (lpj=368640)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 2048 (order: 0, 8192 bytes)
Mountpoint-cache hash table entries: 2048 (order: 0, 8192 bytes)
smp: Bringing up secondary CPUs ...
Idle Task [1] 9f053880
Trying to bring up CPU1 ...
archs-intc      : 2 priority levels (default 1)

IDENTITY       : ARCOVER [0x53] ARCNUM [0x1] CHIPID [ 0x0]
processor [1]  : ARC HS38 R3.0 (ARCV2 ISA)

```

```

Timers          : Timer0 Timer1
ISA Extn       : atomic ll64 unalign (not used)
                : mpy[opt 9] div_rem norm barrel-shift swap minmax swape
BPU            : full match, cache:512, Predict Table:8192
MMU [v4]       : 8k PAGE, 2M Super Page (not used) JTLB 512 (128x4), uDTLB 8,
uITLB 4, PAE40 (not used)
I-Cache        : 64K, 4way/set, 64B Line, VIPT aliasing
D-Cache        : 64K, 2way/set, 64B Line, PIPT
SLC            : 512K, 64B Line
Peripherals    : 0xe0000000, IO-Coherency
Vector Table   : 0x80000000
FPU            : SP DP
DEBUG          : ActionPoint smaRT RTT
OS ABI [v4]    : 64-bit data any register aligned
Extn [SMP]     : ARConnect (v2): 2 cores with IPI IDU DEBUG GFRC
CONFIG_ARC_FPU_SAVE_RESTORE needed for working apps
## CPU1 LIVE ##: Executing Code...
Idle Task [2] 9f0533c0
Trying to bring up CPU2 ...
Timeout: CPU2 FAILED to comeup !!!
Idle Task [3] 9f052f00
Trying to bring up CPU3 ...
random: fast init done
Timeout: CPU3 FAILED to comeup !!!
smp: Brought up 1 node, 2 CPUs
devtmpfs: initialized
clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns:
19112604462750000 ns
futex hash table entries: 1024 (order: 3, 65536 bytes)
NET: Registered protocol family 16
irq: no irq domain found for /cpu_card/dw-apb-gpio@0x2000/gpio-controller@0 !
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
i2c_designware e001e000.i2c: Unknown Synopsys component type: 0x00000030
pps_core: LinuxPPS API ver. 1 registered
pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti
<giometti@linux.it>
PTP clock support registered
clocksource: Switched to clocksource ARConnect GFRC
NET: Registered protocol family 2
TCP established hash table entries: 4096 (order: 1, 16384 bytes)
TCP bind hash table entries: 4096 (order: 2, 32768 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
UDP hash table entries: 256 (order: 0, 8192 bytes)
UDP-Lite hash table entries: 256 (order: 0, 8192 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
ARC perf       : 8 counters (48 bits), 121 conditions, [overflow IRQ support]
workingset: timestamp_bits=30 max_order=16 bucket_order=0
ntfs: driver 2.1.32 [Flags: R/O].
Block layer SCSI generic (bsg) driver version 0.4 loaded (major 251)
io scheduler noop registered

```

```
io scheduler deadline registered
io scheduler cfq registered (default)
Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
f0005000.dw-apb-uart: ttyS0 at MMIO 0xf0005000 (irq = 6, base_baud = 2083312) is a
16550A
e0020000.uart: ttyS1 at MMIO 0xe0020000 (irq = 17, base_baud = 2083333) is a 16550A
e0021000.uart: ttyS2 at MMIO 0xe0021000 (irq = 18, base_baud = 2083333) is a 16550A
e0022000.uart: ttyS3 at MMIO 0xe0022000 (irq = 22, base_baud = 2083333) is a 16550A
console [ttyS3] enabled
console [ttyS3] enabled
bootconsole [uart8250] disabled
bootconsole [uart8250] disabled
loop: module loaded
libphy: Fixed MDIO Bus: probed
stmmaceth e0018000.ethernet: no reset control found
stmmac - user ID: 0x10, Synopsys ID: 0x37
stmmaceth e0018000.ethernet: Ring mode enabled
stmmaceth e0018000.ethernet: DMA HW capability register supported
stmmaceth e0018000.ethernet: Normal descriptors
stmmaceth e0018000.ethernet: RX Checksum Offload Engine supported
stmmaceth e0018000.ethernet: COE Type 2
stmmaceth e0018000.ethernet: TX Checksum insertion supported
stmmaceth e0018000.ethernet: Enable RX Mitigation via HW Watchdog Timer
libphy: stmmac: probed
stmmaceth e0018000.ethernet (unnamed net_device) (uninitialized): PHY ID 20005c7a
at 1 IRQ POLL (stmmac-0:01) active
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
ehci-platform: EHCI generic platform driver
ehci-platform e0040000.ehci: EHCI Host Controller
ehci-platform e0040000.ehci: new USB bus registered, assigned bus number 1
ehci-platform e0040000.ehci: irq 8, io mem 0xe0040000
ehci-platform e0040000.ehci: USB 2.0 started, EHCI 1.00
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
ohci-platform: OHCI generic platform driver
ohci-platform e0060000.ohci: Generic Platform OHCI controller
ohci-platform e0060000.ohci: new USB bus registered, assigned bus number 2
ohci-platform e0060000.ohci: irq 8, io mem 0xe0060000
hub 2-0:1.0: USB hub found
hub 2-0:1.0: 1 port detected
usbcore: registered new interface driver usb-storage
mousedev: PS/2 mouse device common for all mice
usbcore: registered new interface driver synaptics_usb
i2c /dev entries driver
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright (c) Pierre Ossman
Synopsys Designware Multimedia Card Interface Driver
dw_mmc e0015000.mmc: IDMAC supports 32-bit address mode.
dw_mmc e0015000.mmc: Using internal DMA controller.
dw_mmc e0015000.mmc: Version ID is 290a
dw_mmc e0015000.mmc: DW MMC controller at irq 7, 32 bit host data width, 16 deep fifo
mmc_host mmc0: Bus speed (slot 0) = 50000000Hz (slot req 400000Hz, actual 396825Hz
div = 63)
dw_mmc e0015000.mmc: 1 slots initialized
sdhci-pltfm: SDHCI platform and OF driver helper
usbcore: registered new interface driver usbhid
```



```

usbhid: USB HID core driver
NET: Registered protocol family 17
NET: Registered protocol family 15
ttyS3 - failed to request DMA
Freeing unused kernel memory: 10112K
This architecture does not have kernel memory protection.
mmc_host mmc0: Bus speed (slot 0) = 50000000Hz (slot req 25000000Hz, actual
25000000Hz div = 1)
mmc0: new SDHC card at address 59b4
blk_queue_max_segment_size: set to minimum 8192
mmcblk0: mmc0:59b4 USD 7.51 GiB
mmcblk0: p1
Starting logging: OK
Initializing random number generator... done.
Starting network: stmmaceth e0018000.ethernet eth0: device MAC address
d2:83:38:6d:af:37
stmmaceth e0018000.ethernet eth0: fail to init PTP.
udhcpc: started, v1.26.2
udhcpc: sending discover
stmmaceth e0018000.ethernet eth0: Link is Up - 100Mbps/Full - flow control off
udhcpc: sending discover
udhcpc: sending discover
udhcpc: no lease, failing
FAIL
ssh-keygen: generating new host keys: RSA
DSA random: crng init done
ECDSA ED25519
Starting sshd: OK

Welcome to the ARC Software Development Platform
axs103 login: root
#

```

## Saving

You can save the U-Boot environment with the `saveenv` command as follows:

```

setenv bootcmd fatload mmc 0\; bootm
saveenv

```

## Cleaning

To clean the `bootcmd` variable, run the following commands:

```

setenv bootcmd
saveenv

```

### 8.7.3.3 Linux Execution with U-Boot Autostarted on Power-On

ARC AXS103 boards have built-in SPI-flash storage that you can use to host auto-started applications such as bootloaders.

The batch file `axs_comm_progam_uboot.bat` flashes `u-boot_axs103.bin` into SPI flash. Before running `axs_comm_progam_uboot.bat` close any hyperterminals (such as PuTTY) and debugger windows that are using the USB data port of the AXS Software Development Platform.

The batch file uses the `axs_comm` utility described in the *ARC SDP Mainboard User Guide* [\[5\]](#).

After the flashing process is completed do the following:

1. Set the third pin on dip switch SW2501 in to OFF (otherwise the image from SPI flash is not read).
2. Push the RESET button (SW2410) on the Mainboard.

You see following log in serial console:

```
AXS# ** PREBOOT **
Apr 14 2017
13:44:39
ARCHS38[0]
53
CPU-FPGA-VERSION:
1342017
15:10
MB-FPGA-VERSION:
1442017
13:21
MB-CPLD-VERSION:
312017
18:30
FLASH SPEED: 50MHz
FLASH READ MODE: FAST
FLASH ADDR MODE: 3byte
READ INFO FROM FLASH:
UPDATE FLASH
IMAGE: SPIFLASH
VALID IMAGE FOUND IN:
00
0
** HEADER **
ARCID:
53
SIZE:
4D9F8
CRC:
46
COPYTO:
80FFFFFFC
LOAD
VRF:
OK
JUMP

U-Boot 2017.01 (Apr 11 2017 - 11:54:33 +0300)

I2C: ready
DRAM: 512 MiB
NAND: 0 MiB
MMC: Synopsys Mobile storage: 0
*** Warning - bad CRC, using default environment
```

```
In:    serial0@e0022000
Out:   serial0@e0022000
Err:   serial0@e0022000
Net:
Warning: ethernet@e0018000 (eth0) using random MAC address - b2:80:7f:24:64:c4
eth0: ethernet@e0018000
AXS#
```

## Running

The following commands run Linux after U-Boot autostart on power-on:

```
setenv bootcmd fatload mmc 0\; bootm
saveenv
```

Push the RESET button (SW2410) on the Mainboard to run Linux.

To prevent Linux autostart press any key within three seconds to stop the countdown.

## Cleaning

To clean `bootcmd` variable run following commands:

```
setenv bootcmd
saveenv
```

---

## 8.8 ARCV2 Instruction Set: Usage Limitations

The ARCV2 architecture provides special load-locked and store-conditional instructions (`lock`, `llockd`, and `scond`, and `scond`) to enable the implementation of lock-free data structures.

These instructions cannot be used in the DesignWare ARC AXS103 Software Development Platform when Data Cache and Instruction Caches are disabled.

See the *ARCV2 ISA Programmer's Reference Manual* for additional information about these instructions.

This section describes the control registers inside the AXC003 Processor FPGA. Add the address offset listed for each register to the base address of the AXI2APB bridge (default address: 0xF000\_0000) to obtain the register address.

## 9.1 Clock-Generation Registers

This section describes the registers used for clock generation.

### 9.1.1 TUNNEL PLL

Reference input clock for Tunnel PLL is 33Mhz

Minimum input clock frequency is 10MHz

VCO range for Tunnel PLL is 600 - 1440MHz

#### 9.1.1.1 TUN\_PLL\_IDIV Register

31	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					NOUPDATE	BYPASS	EDGE	HIGHTIME				LOWTIME				

Address offset: 0x0040

Reset Value: 0x0000\_2041 (0x0000\_3001 after pre-boot)

Access: RW

Register to control setting of the Tunnel PLL input divider

LOWTIME[5:0] sets the amount of time in input cycles that the divided input clock remains low

HIGHTIME[5:0] sets the amount of time in input cycles that the divided input clock remains high

IDIV = LOWTIME + HIGHTIME

EDGE chooses the edge that the High Time counter transitions on (0=rising, 1=falling)

BYPASS	bypass the input divider
NOUPDATE	prevent update of the PLL with new settings. Debug only; can be used for register RW test

To obtain a 50% duty-cycle the divider shall be programmed as follows:

- even divider ratio => LOWTIME = HIGHTIME  
EDGE = 0
- odd divider ratio => LOWTIME = HIGHTIME + 1  
EDGE = 1

### 9.1.1.2 TUN\_PLL\_FBDIV Register

31	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					NOUPDATE	BYPASS	EDGE	HIGHTIME				LOWTIME				

Address offset: 0x0044

Reset Value: 0x0000\_03CF (0x0000\_03CF after pre-boot)

Access: RW

Register to control setting of the Tunnel PLL feedback divider

LOWTIME[5:0] sets the amount of time in VCO cycles that the feedback clock remains low

HIGHTIME[5:0] sets the amount of time in VCO cycles that the feedback clock remains high

$$FBDIV = LOWTIME + HIGHTIME$$

$$VCOFREQ = (33MHz / IDIV) * FBDIV$$

EDGE chooses the edge that the High Time counter transitions on (0=rising, 1=falling)

BYPASS bypass the feedback divider

NOUPDATE prevent update of the PLL with new settings. Debug only; can be used for register RW test

To obtain a 50% duty-cycle the divider shall be programmed as follows:

- even divider ratio => LOWTIME = HIGHTIME  
EDGE = 0
- odd divider ratio => LOWTIME = HIGHTIME + 1  
EDGE = 1

### 9.1.1.3 TUN\_PLL\_ODIV Register

31	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					NOUPDATE	BYPASS	EDGE	HIGHTIME				LOWTIME				

Address offset: 0x0048

Reset Value: 0x0001\_028A (0x0000\_0145 after pre-boot)

Access: RW

Register for controlling the clock setting for AXI tunnel.

LOWTIME[5:0] sets the amount of time in VCO cycles that the output clock remains low

HIGHTIME[5:0] sets the amount of time in VCO cycles that the output clock remains high

$ODIV = LOWTIME + HIGHTIME$

$OFREQ = VCOFREQ / ODIV$

EDGE chooses the edge that the High Time counter transitions on (0=rising, 1=falling)

BYPASS bypass the output divider

NOUPDATE prevents update of the PLL with new settings. Debug only; can be used for register RW test

To obtain a 50% duty-cycle the divider shall be programmed as follows:

- even divider ratio =>  $LOWTIME = HIGHTIME$   
 $EDGE = 0$
- odd divider ratio =>  $LOWTIME = HIGHTIME + 1$   
 $EDGE = 1$

### 9.1.1.4 TUN\_PLL\_LOCK Register

31	2	1	0
Reserved		ERROR	LOCK

Address offset: 0x0108

Reset Value: 0x0000\_0001

Access: R

Register for Tunnel PLL lock status

LOCK	PLL lock indication 0 = PLL is unlocked 1 = PLL is locked
ERROR	PLL error indication. Asserted high to indicate that PLL was programmed with an illegal value. PLL can be re-programmed after the ERROR status bit is reset to 0

## 9.1.2 ARC PLL

Reference input clock for ARC PLL is 33Mhz

Minimum input clock frequency is 10MHz

VCO range for ARC PLL is 600 - 1440MHz

### 9.1.2.1 ARC\_PLL\_IDIV Register

31	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			NOUPDATE	BYPASS	EDGE	HIGHTIME			LOWTIME							

Address offset: 0x0080

Reset Value: 0x0000\_2041 (0x0000\_3001 after pre-boot)

Access: RW

Register to control setting of the ARC PLL input divider

LOWTIME[5:0] sets the amount of time in input cycles that the divided input clock remains low

HIGHTIME[5:0] sets the amount of time in input cycles that the divided input clock remains high

$IDIV = LOWTIME + HIGHTIME$

EDGE chooses the edge that the High Time counter transitions on (0=rising, 1=falling)

BYPASS bypass the input divider

NOUPDATE prevent update of the PLL with new settings. Debug only; can be used for register RW test

To obtain a 50% duty-cycle the divider shall be programmed as follows:

- even divider ratio =>  $LOWTIME = HIGHTIME$   
 $EDGE = 0$

- odd divider ratio =>  $LOWTIME = HIGHTIME + 1$   
 $EDGE = 1$

### 9.1.2.2 ARC\_PLL\_FBDIV Register

31	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					NOUPDATE	BYPASS	EDGE	HIGHTIME				LOWTIME				

Address offset: 0x0084

Reset Value: 0x0000\_03CF (0x0000\_03CF after pre-boot)

Access: RW

Register to control setting of the ARC PLL feedback divider

LOWTIME[5:0] sets the amount of time in VCO cycles that the feedback clock remains low

HIGHTIME[5:0] sets the amount of time in VCO cycles that the feedback clock remains high

$$FBDIV = LOWTIME + HIGHTIME$$

$$VCOFREQ = (33MHz / IDIV) * FBDIV$$

EDGE chooses the edge that the High Time counter transitions on (0=rising, 1=falling)

BYPASS bypass the feedback divider

NOUPDATE prevent update of the PLL with new settings. Debug only; can be used for register RW test

To obtain a 50% duty-cycle the divider shall be programmed as follows:

- even divider ratio =>  $LOWTIME = HIGHTIME$   
 $EDGE = 0$
- odd divider ratio =>  $LOWTIME = HIGHTIME + 1$   
 $EDGE = 1$

### 9.1.2.3 ARC\_PLL\_ODIV Register

31	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					NOUPDATE	BYPASS	EDGE	HIGHTIME				LOWTIME				

Address offset: 0x0088

Reset Value: 0x0001\_028A (0x0000\_0145 after pre-boot)

Access: RW



Register for controlling the clock setting for the ARC CPU

LOWTIME[5:0] sets the amount of time in VCO cycles that the output clock remains low

HIGHTIME[5:0] sets the amount of time in VCO cycles that the output clock remains high

$$\text{ODIV} = \text{LOWTIME} + \text{HIGHTIME}$$

$$\text{OFREQ} = \text{VCOFREQ} / \text{ODIV}$$

EDGE chooses the edge that the High Time counter transitions on (0=rising, 1=falling)

BYPASS bypass the output divider

NOUPDATE prevents update of the PLL with new settings. Debug only; can be used for register RW test

To obtain a 50% duty-cycle the divider shall be programmed as follows:

- even divider ratio => LOWTIME = HIGHTIME  
EDGE = 0
- odd divider ratio => LOWTIME = HIGHTIME + 1  
EDGE = 1

#### 9.1.2.4 ARC\_PLL\_LOCK Register

31	Reserved	2	1	0
		ERROR	LOCK	

Address offset: 0x0110

Reset Value: 0x0000\_0001

Access: R

Register for ARC PLL lock status

LOCK PLL lock indication

0 = PLL is unlocked

1 = PLL is locked

ERROR PLL error indication. Asserted high to indicate that PLL was programmed with an illegal value. PLL can be re-programmed once the ERROR status bit is reset to 0

## 9.2 AXI Tunnel Address Decoder Registers

The AXI Tunnel Address Decoder Registers described below are re-programmed by the pre-bootloader. The reset values mentioned here are the reset values prior to running the pre-bootloader. See [Example Register Settings for the Default Memory Map](#) on page 69 for the register settings after running the pre-bootloader.

### 9.2.1 TUN\_A\_SLV0: AXI Tunnel Slave Select Register 0

Address offset: 0x1000

Reset value: 0x1111\_1111

Table 1 TUN\_A\_SLV0 Register

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_SEL0	RW	1 *	Slave select for address aperture[0]
			0	no slave selected
			1	slave 1 selected (=> DDR controller)
			2	slave 2 selected (=> SRAM controller)
			3	slave 3 selected (=> AXI tunnel)
			4	slave 4 selected (=> AXI2APB bridge)
			5	slave 5 selected (=> ROM Controller)
			6	slave 6 selected (=> IOC port)
7	Reserved			
7:4	SLV_SEL1	RW	1 *	Slave select for address aperture[1] <sup>1)</sup>
11:8	SLV_SEL2	RW	1 *	Slave select for address aperture[2] <sup>1)</sup>
15:12	SLV_SEL3	RW	1 *	Slave select for address aperture[3] <sup>1)</sup>
19:16	SLV_SEL4	RW	1 *	Slave select for address aperture[4] <sup>1)</sup>
23:20	SLV_SEL5	RW	1 *	Slave select for address aperture[5] <sup>1)</sup>
27:24	SLV_SEL6	RW	1 *	Slave select for address aperture[6] <sup>1)</sup>
31:28	SLV_SEL7	RW	1 *	Slave select for address aperture[7] <sup>1)</sup>

2) Same encoding as SLV\_SEL0

### 9.2.2 TUN\_A\_SLV1: AXI Tunnel Slave Select Register 1

Address offset: 0x1004

Reset value: 0x1111\_1111      ARC core without I/O Coherency  
 0x6666\_6666      ARC core with I/O Coherency

Table 2 *TUN\_A\_SLV1 Register*

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_SEL8	RW	1 / 6*	Slave select for address aperture[8]
*			0	no slave selected
			1	slave 1 selected (=> DDR controller)
			2	slave 2 selected (=> SRAM controller)
			3	slave 3 selected (=> AXI tunnel)
			4	slave 4 selected (=> AXI2APB bridge)
			5	slave 5 selected (=> ROM Controller)
			6	slave 6 selected (=> IOC port)
			7	Reserved
7:4	SLV_SEL9	RW	1 / 6*	Slave select for address aperture[9] <sup>1)</sup>
11:8	SLV_SEL10	RW	1 / 6*	Slave select for address aperture[10] <sup>1)</sup>
15:12	SLV_SEL11	RW	1 / 6*	Slave select for address aperture[11] <sup>1)</sup>
19:16	SLV_SEL12	RW	1 / 6*	Slave select for address aperture[12] <sup>1)</sup>
23:20	SLV_SEL13	RW	1 / 6*	Slave select for address aperture[13] <sup>1)</sup>
27:24	SLV_SEL14	RW	1 / 6*	Slave select for address aperture[14] <sup>1)</sup>
31:28	SLV_SEL15	RW	1 / 6*	Slave select for address aperture[15] <sup>1)</sup>

1) Same encoding as SLV\_SEL8

### 9.2.3 TUN\_A\_OFFSET0: AXI Tunnel Address Offset Register 0

Address offset: 0x1008

Reset value: 0x7654\_3210

Table 3 *TUN\_A\_OFFSET0 Register*

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_OFFSET0	RW		Address offset select for address aperture[0]
			0*	0*256MB
			1	1*256MB
			...	...
			15	15*256MB
7:4	SLV_OFFSET1	RW	1*	Address offset for address aperture[1] <sup>1)</sup>
11:8	SLV_OFFSET2	RW	2*	Address offset for address aperture[2] <sup>1)</sup>
15:12	SLV_OFFSET3	RW	3*	Address offset for address aperture[3] <sup>1)</sup>
19:16	SLV_OFFSET4	RW	4*	Address offset for address aperture[4] <sup>1)</sup>

23:20	SLV_OFFSET5	RW	5*	Address offset for address aperture[5] <sup>1)</sup>
27:24	SLV_OFFSET6	RW	6*	Address offset for address aperture[6] <sup>1)</sup>
31:28	SLV_OFFSET7	RW	7*	Address offset for address aperture[7] <sup>1)</sup>

1) Same encoding as SLV\_OFFSET0

## 9.2.4 TUN\_A\_OFFSET1: AXI Tunnel Address Offset Register 1

Address offset: 0x100C

Reset value: 0x7654\_3210 ARC core without I/O Coherency port  
0xFEDC\_BA98 ARC core with I/O Coherency port

Table 4 TUN\_A\_OFFSET1 Register

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_OFFSET8	RW	0 / 8*	Address offset for address aperture[8]
			0	0*256MB
			1	1*256MB
			...	...
			15	15*256MB
7:4	SLV_OFFSET9	RW	1 / 9*	Address offset for address aperture[9] <sup>1)</sup>
11:8	SLV_OFFSET10	RW	2 / 10*	Address offset for address aperture[10] <sup>1)</sup>
15:12	SLV_OFFSET11	RW	3 / 11*	Address offset for address aperture[11] <sup>1)</sup>
19:16	SLV_OFFSET12	RW	4 / 12*	Address offset for address aperture[12] <sup>1)</sup>
23:20	SLV_OFFSET13	RW	5 / 13*	Address offset for address aperture[13] <sup>1)</sup>
27:24	SLV_OFFSET14	RW	6 / 14*	Address offset for address aperture[14] <sup>1)</sup>
31:28	SLV_OFFSET15	RW	7 / 15*	Address offset for address aperture[15] <sup>1)</sup>

1) Same encoding as OFFSET8

## 9.2.5 TUN\_A\_UPDATE: AXI Tunnel Update Register

Address offset: 0x1014

Reset value: 0x0000\_0000

Table 5 TUN\_A\_UPDATE Register

Legend: * reset value				
Bit	Name	Access	Value	Description
0	UPDATE	RW1C	0*	All the address aperture configuration registers (i.e. *_A_SLV) are double-buffered. The newly programmed values will be only be forwarded to the address decoder after writing a '1' to this bit.

31:1	Reserved	R	0x0*	
------	----------	---	------	--

## 9.3 ARC CPU Address Decoder Registers

The ARC CPU Address Decoder Registers are re-programmed by the pre-bootloader. The reset values mentioned here are the reset values prior to running the pre-bootloader. See the [Example Register Settings for the Default Memory Map](#) on page 69 for the register settings after the pre-bootloader runs.

### 9.3.1 CPU\_A\_SLV0: ARC CPU Slave Select Register 0

Address offset: 0x1020

Reset value: 0x0000\_5322

Table 6 CPU\_A\_SLV0 Register

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_SEL0	RW		Slave select for address aperture[0]
			0	no slave selected
			1	slave 1 selected (=> DDR controller)
			2*	slave 2 selected (=> SRAM controller)
			3	slave 3 selected (=> AXI tunnel)
			4	slave 4 selected (=> AXI2APB bridge)
			5	slave 5 selected (=> ROM Controller)
			6	slave 6 selected (=> IOC port)
7	Reserved			
7:4	SLV_SEL1	RW	2*	Slave select for address aperture[1] <sup>1)</sup>
11:8	SLV_SEL2	RW	3*	Slave select for address aperture[2] <sup>1)</sup>
15:12	SLV_SEL3	RW	5*	Slave select for address aperture[3] <sup>1)</sup>
19:16	SLV_SEL4	RW	0*	Slave select for address aperture[4] <sup>1)</sup>
23:20	SLV_SEL5	RW	0*	Slave select for address aperture[5] <sup>1)</sup>
27:24	SLV_SEL6	RW	0*	Slave select for address aperture[6] <sup>1)</sup>
31:28	SLV_SEL7	RW	0*	Slave select for address aperture[7] <sup>1)</sup>

1) Same encoding as SLV\_SEL0

### 9.3.2 CPU\_A\_SLV1: ARC CPU Slave Select Register 1

Address offset: 0x1024

Reset value: 0x4330\_1111

Table 7 CPU\_A\_SLV1 Register

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_SEL8	RW		Slave select for address aperture[8]
			0	no slave selected
			1*	slave 1 selected (=> DDR controller)
			2	slave 2 selected (=> SRAM controller)
			3	slave 3 selected (=> AXI tunnel)
			4	slave 4 selected (=> AXI2APB bridge)
			5	slave 5 selected (=> ROM Controller)
			6	slave 6 selected (=> IOC port)
7	Reserved			
7:4	SLV_SEL9	RW	1*	Slave select for address aperture[9] <sup>1)</sup>
11:8	SLV_SEL10	RW	1*	Slave select for address aperture[10] <sup>1)</sup>
15:12	SLV_SEL11	RW	1*	Slave select for address aperture[11] <sup>1)</sup>
19:16	SLV_SEL12	RW	0*	Slave select for address aperture[12] <sup>1)</sup>
23:20	SLV_SEL13	RW	3*	Slave select for address aperture[13] <sup>1)</sup>
27:24	SLV_SEL14	RW	3*	Slave select for address aperture[14] <sup>1)</sup>
31:28	SLV_SEL15	RW	4*	Slave select for address aperture[15] <sup>1)</sup>

1) Same encoding as SLV\_SEL8

### 9.3.3 CPU\_A\_OFFSET0: ARC CPU Address Offset Register 0

Address offset: 0x1028

Reset value: 0x0000\_0000

Table 8 CPU\_A\_OFFSET0 Register

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_OFFSET0	RW		Address offset select for address aperture[0]
			0*	0*256MB
			1	1*256MB
			...	....
			15	15*256MB
7:4	SLV_OFFSET1	RW	0*	Address offset for address aperture[1] <sup>1)</sup>
11:8	SLV_OFFSET2	RW	0*	Address offset for address aperture[2] <sup>1)</sup>
15:12	SLV_OFFSET3	RW	0*	Address offset for address aperture[3] <sup>1)</sup>
19:16	SLV_OFFSET4	RW	0*	Address offset for address aperture[4] <sup>1)</sup>

23:20	SLV_OFFSET5	RW	0*	Address offset for address aperture[5] <sup>1)</sup>
27:24	SLV_OFFSET6	RW	0*	Address offset for address aperture[6] <sup>1)</sup>
31:28	SLV_OFFSET7	RW	0*	Address offset for address aperture[7] <sup>1)</sup>

1) Same encoding as SLV\_OFFSET0

### 9.3.4 CPU\_A\_OFFSET1: ARC CPU Address Offset Register 1

Address offset: 0x102C

Reset value: 0x0ED0\_3210

Table 9 CPU\_A\_OFFSET1 Register

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_OFFSET8	RW		Address offset for address aperture[8]
			0*	0*256MB
			1	1*256MB
			...	....
			15	15*256MB
7:4	SLV_OFFSET9	RW	1*	Address offset for address aperture[9] <sup>1)</sup>
11:8	SLV_OFFSET10	RW	2*	Address offset for address aperture[10] <sup>1)</sup>
15:12	SLV_OFFSET11	RW	3*	Address offset for address aperture[11] <sup>1)</sup>
19:16	SLV_OFFSET12	RW	0*	Address offset for address aperture[12] <sup>1)</sup>
23:20	SLV_OFFSET13	RW	D*	Address offset for address aperture[13] <sup>1)</sup>
27:24	SLV_OFFSET14	RW	E*	Address offset for address aperture[14] <sup>1)</sup>
31:28	SLV_OFFSET15	RW	0*	Address offset for address aperture[15] <sup>1)</sup>

1) Same encoding as SLV\_OFFSET8

### 9.3.5 CPU\_A\_UPDATE: ARC CPU Update Register

Address offset: 0x1034

Reset value: 0x0000\_0000

Table 10 CPU\_A\_UPDATE Register

Legend: * reset value				
Bit	Name	Access	Value	Description
0	UPDATE	RW1C	0*	All the address aperture configuration registers (i.e. *_A_SLV and *_A_BOOT) are double-buffered. The newly programmed values will be only be forwarded to the address decoder after writing a '1' to this bit.
31:1	Reserved	R	0x0*	

## 9.4 ARC RTT Address Decoder Registers

### 9.4.1 RTT\_A\_SLV0: ARC RTT Slave Select Register 0

Address offset: 0x1040

Reset value: 0x1111\_1111

Table 11 RTT\_A\_SLV0 Register

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_SEL0	RW		Slave select for address aperture[0]
			0	no slave selected
			1*	slave 1 selected (=> DDR controller)
			2	slave 2 selected (=> SRAM controller)
			3	slave 3 selected (=> AXI tunnel)
			4	slave 4 selected (=> AXI2APB bridge)
			5	slave 5 selected (=> ROM Controller)
			6	slave 6 selected (=> IOC port)
7	Reserved			
7:4	SLV_SEL1	RW	1*	Slave select for address aperture[1] <sup>1)</sup>
11:8	SLV_SEL2	RW	1*	Slave select for address aperture[2] <sup>1)</sup>
15:12	SLV_SEL3	RW	1*	Slave select for address aperture[3] <sup>1)</sup>
19:16	SLV_SEL4	RW	1*	Slave select for address aperture[4] <sup>1)</sup>
23:20	SLV_SEL5	RW	1*	Slave select for address aperture[5] <sup>1)</sup>
27:24	SLV_SEL6	RW	1*	Slave select for address aperture[6] <sup>1)</sup>
31:28	SLV_SEL7	RW	1*	Slave select for address aperture[7] <sup>1)</sup>

1) Same encoding as SLV\_SEL0

### 9.4.2 RTT\_A\_SLV1: ARC RTT Slave Select Register 1

Address offset: 0x1044

Reset value: 0x1111\_1111

Table 12 RTT\_A\_SLV1 Register

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0		RW		Slave select for address aperture[8]



	SLV_SEL8		0	no slave selected
			1*	slave 1 selected (=> DDR controller)
			2	slave 2 selected (=> SRAM controller)
			3	slave 3 selected (=> AXI tunnel)
			4	slave 4 selected (=> AXI2APB bridge)
			5	slave 5 selected (=> ROM Controller)
			6	slave 6 selected (=> IOC port)
			7	Reserved
7:4	SLV_SEL9	RW	1*	Slave select for address aperture[9] <sup>1)</sup>
11:8	SLV_SEL10	RW	1*	Slave select for address aperture[10] <sup>1)</sup>
15:12	SLV_SEL11	RW	1*	Slave select for address aperture[11] <sup>1)</sup>
19:16	SLV_SEL12	RW	1*	Slave select for address aperture[12] <sup>1)</sup>
23:20	SLV_SEL13	RW	1*	Slave select for address aperture[13] <sup>1)</sup>
27:24	SLV_SEL14	RW	1*	Slave select for address aperture[14] <sup>1)</sup>
31:28	SLV_SEL15	RW	1*	Slave select for address aperture[15] <sup>1)</sup>

2) Same encoding as SLV\_SEL8

### 9.4.3 RTT\_A\_OFFSET0: ARC RTT Address Offset Register 0

Address offset: 0x1048

Reset value: 0x7654\_3210

Table 13 RTT\_A\_OFFSET0 Register

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_OFFSET0	RW		Address offset select for address aperture[0]
			0*	0*256MB
			1	1*256MB
			...	....
			15	15*256MB
7:4	SLV_OFFSET1	RW	1*	Address offset for address aperture[1] <sup>1)</sup>
11:8	SLV_OFFSET2	RW	2*	Address offset for address aperture[2] <sup>1)</sup>
15:12	SLV_OFFSET3	RW	3*	Address offset for address aperture[3] <sup>1)</sup>
19:16	SLV_OFFSET4	RW	4*	Address offset for address aperture[4] <sup>1)</sup>
23:20	SLV_OFFSET5	RW	5*	Address offset for address aperture[5] <sup>1)</sup>
27:24	SLV_OFFSET6	RW	6*	Address offset for address aperture[6] <sup>1)</sup>
31:28	SLV_OFFSET7	RW	7*	Address offset for address aperture[7] <sup>1)</sup>

2) Same encoding as SLV\_OFFSET0

### 9.4.4 RTT\_A\_OFFSET1: ARC RTT Address Offset Register 1

Address offset: 0x104C

Reset value: 0x7654\_3210

Table 14 RTT\_A\_OFFSET1 Register

Legend: * reset value				
Bit	Name	Access	Value	Description
3:0	SLV_OFFSET8	RW		Address offset for address aperture[8]
			0*	0*256MB
			1	1*256MB
			...	....
15			15*256MB	
7:4	SLV_OFFSET9	RW	1*	Address offset for address aperture[9] <sup>1)</sup>
11:8	SLV_OFFSET10	RW	2*	Address offset for address aperture[10] <sup>1)</sup>
15:12	SLV_OFFSET11	RW	3*	Address offset for address aperture[11] <sup>1)</sup>
19:16	SLV_OFFSET12	RW	4*	Address offset for address aperture[12] <sup>1)</sup>
23:20	SLV_OFFSET13	RW	5*	Address offset for address aperture[13] <sup>1)</sup>
27:24	SLV_OFFSET14	RW	6*	Address offset for address aperture[14] <sup>1)</sup>
31:28	SLV_OFFSET15	RW	7*	Address offset for address aperture[15] <sup>1)</sup>

2) Same encoding as SLV\_OFFSET8

### 9.4.5 RTT\_A\_UPDATE: ARC RTT Update Register

Address offset: 0x1054

Reset value: 0x0000\_0000

Table 15 RTT\_A\_UPDATE Register

Legend: * reset value				
Bit	Name	Access	Value	Description
0	UPDATE	RW1C	0*	All the address aperture configuration registers (i.e. *_A_SLV and *_A_BOOT) are double-buffered. The newly programmed values will be only be forwarded to the address decoder after writing a '1' to this bit.
31:1	Reserved	R	0x0*	

## 9.5 PAE Registers

### 9.5.1 PAE: PAE Register

Address offset: 0x1060

Reset value: 0x5500\_0000

Table 16 PAE Register

Legend: * reset value				
Bit	Name	Access	Value	Description
0	PAE_0	RW		Physical address extension bits. These bits can be used to remap the 1 <sup>st</sup> 256MByte of IOC DMA traffic into the extended PAE region of the memory map
			0*	no PAE offset
			1	PAE offset = 4GByte
2	PAE_1	RW	0*	Physical address extension bits for 2 <sup>nd</sup> 256MByte <sup>1)</sup>
4	PAE_2	RW	0*	Physical address extension bits for 3 <sup>rd</sup> 256MByte <sup>1)</sup>
6	PAE_3	RW	0*	Physical address extension bits for 4 <sup>th</sup> 256MByte <sup>1)</sup>
8	PAE_4	RW	0*	Physical address extension bits for 5 <sup>th</sup> 256MByte <sup>1)</sup>
10	PAE_5	RW	0*	Physical address extension bits for 6 <sup>th</sup> 256MByte <sup>1)</sup>
12	PAE_6	RW	0*	Physical address extension bits for 7 <sup>th</sup> 256MByte <sup>1)</sup>
14	PAE_7	RW	0*	Physical address extension bits for 8 <sup>th</sup> 256MByte <sup>1)</sup>
16	PAE_8	RW	0*	Physical address extension bits for 9 <sup>th</sup> 256MByte <sup>1)</sup>
18	PAE_9	RW	0*	Physical address extension bits for 10 <sup>th</sup> 256MByte <sup>1)</sup>
20	PAE_10	RW	0*	Physical address extension bits for 11 <sup>th</sup> 256MByte <sup>1)</sup>
22	PAE_11	RW	0*	Physical address extension bits for 12 <sup>th</sup> 256MByte <sup>1)</sup>
24	PAE_12	RW	1*	Physical address extension bits for 13 <sup>th</sup> 256MByte <sup>1)</sup>
26	PAE_13	RW	1*	Physical address extension bits for 14 <sup>th</sup> 256MByte <sup>1)</sup>
28	PAE_14	RW	1*	Physical address extension bits for 15 <sup>th</sup> 256MByte <sup>1)</sup>
30	PAE_15	RW	1*	Physical address extension bits for 16 <sup>th</sup> 256MByte <sup>1)</sup>

1) Same encoding as PAE\_0

### 9.5.2 PAE\_UPDATE: PAE Update Register

Address offset: 0x1074

Reset value: 0x0000\_0000

Table 17 PAE\_UPDATE Register

Legend: * reset value				
Bit	Name	Access	Value	Description
0	UPDATE	RW1C	0*	The PAE configuration registers is double-buffered. The newly programmed value will be only be forwarded to the IOC port after writing a '1' to this bit.
31:1	Reserved	R	0x0*	

## 9.6 CPU Start Registers

### 9.6.1 CPU\_START: ARC CPU Start Register

Address offset: 0x1400

Reset value: depends on DIP switch settings on the ARC SDP Mainboard

Table 18 CPU\_START Register

Legend: * reset value				
Bit	Name	Access	Value	Description
0	START_0	RW1C	0x0*	Writing a '1' to this bit will generate a <code>cpu_start</code> pulse for the 1 <sup>st</sup> ARC
1	START_1	RW1C	0x0*	Writing a '1' to this bit will generate a <code>cpu_start</code> pulse for the 2 <sup>nd</sup> ARC
4	START_MODE	RW		Boot mode select Boot start mode
			0x0 <sup>[1]</sup>	Start ARC core manually (CREG, external start button or debugger)
			0x1	Start ARC core autonomously after reset
8	POL	RW		Polarity of <code>cpu_start</code> pulse
			0x0	active low
			0x1*	active high
10:9	CORE_SEL	RW		Boot Core Select
			0x0 <sup>[2]</sup>	HS38x2_0   HS36
			0x1	HS38x2_1
			0x2	Reserved
			0x3	Reserved
13:12	MULTI_CORE	RW		Multi Core Mode
			0x0 <sup>[3]</sup>	Singe-core
			0x1	Dual-core
			0x2	Reserved
			0x3	Reserved

- 1) Reset value for *START\_MODE* is sampled from *SW2501[7]* pin during power-on-reset on the ARC SDP Mainboard
- 2) Reset value for *CORE\_SEL* is sampled from *SW2503[2:1]* pins during power-on-reset
- 3) Reset value for *MULTI\_CORE* is sampled from *SW2503[5:4]* pins during power-on-reset

### 9.6.2 CPU\_0\_ENTRY: ARC CPU-0 Kernel Entry Point Register

Address offset: 0x1404

Reset value: 0x0000\_0000

Table 19 CPU\_0\_ENTRY Register

Legend: * reset value				
Bit	Name	Access	Value	Description
31:0	ENTRY	RW	0*	Kernel entry point for ARC CPU-0

### 9.6.3 CPU\_1\_ENTRY: ARC CPU-1 Kernel Entry Point Register

Address offset: 0x1408

Reset value: 0x0000\_0000

Table 20 CPU\_1\_ENTRY Register

Legend: * reset value				
Bit	Name	Access	Value	Description
31:0	ENTRY	RW	0*	Kernel entry point for ARC CPU-1

### 9.6.4 CPU\_BOOT: Boot Register

Address offset: 0x1010

Reset value: depends on DIP switch settings on the ARC SDP Mainboard

Table 21 CPU\_BOOT Register

Legend: * reset value				
Bit	Name	Access	Value	Description
1:0	MIRROR	RW		Boot mirror
			0x0 <sup>1</sup>	Disabled
			0x1	Internal ROM
			0x2	Reserved
5:4		RW		Image source location

	IMAGE_SRC		0x0 [2]	bypass (i.e. ARC core will enter HALT state after pre-boot)
			0x1	SPI FLASH
			0x2	Reserved
			0x3	Reserved
			0x4	Reserved
			0x5	Reserved
			0x6	Reserved
			0x7	Reserved
6	Reserved	RW	0x0	Reserved
7	MODE_HS34	RW	0x0 [3]	HS36 (Cache enabled)
			0x1	HS34 emulation (Cache disabled)

- 1) Reset value for MIRROR[1:0] is sampled from SW2501[2:1] pin during power-on-reset
- 2) Reset value for IMAGE\_SRC[1:0] is sampled from SW2501[4:3] pin during power-on-reset
- 3) Reset value for MODE\_HS34 is sampled from SW2501[6] pin during power-on-reset.

## 9.7 AXI Tunnel Registers

### 9.7.1 TUN\_CTRL Register

31	Reserved	2 1 0	PR/O[1:0]
----	----------	-------	-----------

Address offset: 0x14A0  
 Reset Value: 0x0000\_0000  
 Access: RW

PRI0[1:0] controls the priority setting for the tunnel arbitration

- PRI0 = 0 → AXI master and slave have equal priority (round-robin)
- PRI0 = 1 → axis master has the highest priority
- PRI0 = 2 → AXI slave has the highest priority
- PRI0 = 3] → illegal

### 9.7.2 TUN\_STAT Register

31	Reserved	4 3 2 1 0	STAT0
----	----------	-----------	-------

Address offset: 0x1\_14A4  
 Reset Value: 0x0000\_0000  
 Access: R

STAT[3:0] reflects the status of the tunnel after reset completion

STAT[0] → initialization sequence done (1=done, 0=not yet done)  
 STAT[1] → initialization sequence error (1=error, 0=no error)  
 STAT[2] → BIST sequence done (1=done, 0=not yet done)  
 STAT[3] → best sequence error (1=error, 0=no error)

## 9.8 GPIO Registers

### 9.8.1 GPIO\_SWPORTA\_DR: GPIO Port A Output Register

Address offset: 0x3000  
 Reset value: 0x0000\_0000

Table 41 GPIO port A Output Register (GPIO\_SWPORTA\_DR)

Legend: * reset value				
Bit	Name	Access	Value	Description
0	MB_LED2501	RW	0x0*	LED2501 on the ARC SDP Mainboard is OFF
			0x1	LED2501 on the ARC SDP Mainboard is ON
1	MB_LED2502	RW	0x0*	LED2502 on the ARC SDP Mainboard is OFF
			0x1	LED2502 on the ARC SDP Mainboard is ON
4:2		RW	0x0*	Reserved
5	MB_LED2503	RW	0x0*	LED2503 on the ARC SDP Mainboard is OFF
			0x1	LED2503 on the ARC SDP Mainboard is ON
6	MB_LED2504	RW	0x0*	LED2504 on the ARC SDP Mainboard is OFF
			0x1	LED2504 on the ARC SDP Mainboard is ON
9:7		RW	0x0*	Reserved
10	MB_LED2505	RW	0x0*	LED2505 on the ARC SDP Mainboard is OFF
			0x1	LED2505 on the ARC SDP Mainboard is ON
11	MB_LED2506	RW	0x0*	LED2506 on the ARC SDP Mainboard is OFF
			0x1	LED2506 on the ARC SDP Mainboard is ON
14:12		RW	0x0*	Reserved

15	MB_LED2507	RW	0x0*	LED2507 on the ARC SDP Mainboard is OFF
			0x1	LED2507 on the ARC SDP Mainboard is ON
16	MB_LED2508	RW	0x0*	LED2508 on the ARC SDP Mainboard is OFF
			0x1	LED2508 on the ARC SDP Mainboard is ON
23:17		RW	0x0*	Reserved
31:24		R	0x0*	Reserved

## 9.8.2 GPIO\_SWPORTB\_DR: GPIO Port B Output Register

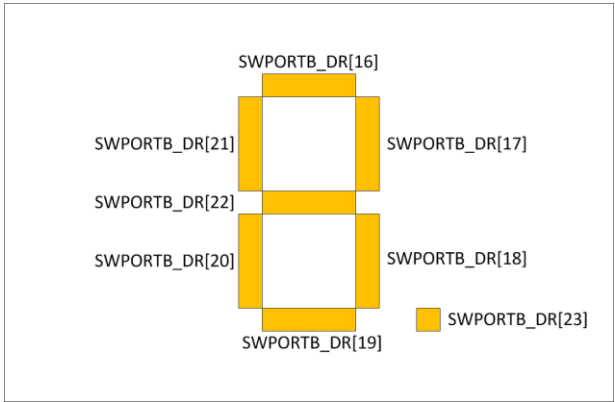
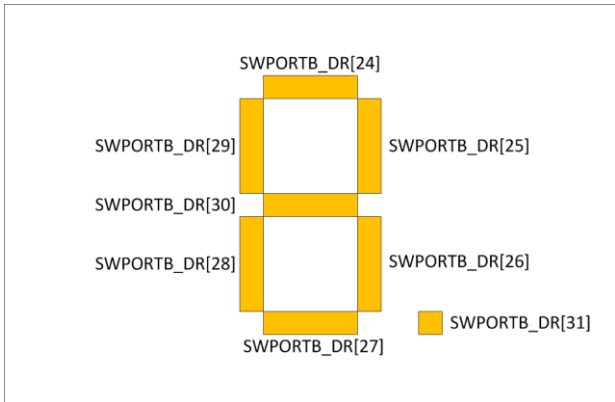
Address offset: 0x300C

Reset value: 0x0000\_0000

Table 42 GPIO port B output Register (GPIO\_SWPORTB\_DR)

Legend: * reset value				
Bit	Name	Access	Value	Description
7:0	Reserved	RW	0x0*	
8	LED0	RW	0*	LED0 on the AXC003 CPU Card is OFF
			1	LED0 on the AXC003 CPU Card is ON
9	LED1	RW	0*	LED1 on the AXC003 CPU Card is OFF
			1	LED1 on the AXC003 CPU Card is ON
10	LED2	RW	0*	LED2 on the AXC003 CPU Card is OFF
			1	LED2 on the AXC003 CPU Card is ON
11	LED3	RW	0*	LED3 on the AXC003 CPU Card is OFF
			1	LED3 on the AXC003 CPU Card is ON
12	LED4	RW	0*	LED4 on the AXC003 CPU Card is OFF
			1	LED4 on the AXC003 CPU Card is ON
13	LED5	RW	0*	LED5 on the AXC003 CPU Card is OFF
			1	LED5 on the AXC003 CPU Card is ON
14	LED6	RW	0*	LED6 on the AXC003 CPU Card is OFF
			1	LED6 on the AXC003 CPU Card is ON
15	LED7	RW	0*	LED7 on the AXC003 CPU Card is OFF
			1	LED7 on the AXC003 CPU Card is ON



23:16	UPPER_7SEG	RW	0*	<p>Controls the upper seven-segment display on the AXC003 CPU Card. A segment of the display is on when its control bit is set to 1.</p> 
31:24	LOWER_7SEG	RW	0*	<p>Controls the lower seven-segment display. A segment of the display is on when its control bit is set to 1.</p> 

### 9.8.3 GPIO\_EXT\_PORTA: GPIO Port A Input Register

Address offset: 0x3050

Reset value: 0x00FD\_0000

Table 43 GPIO Port A Input Register (GPIO\_EXT\_PORTA)

Legend: * reset value				
Bit	Name	Access	Value	Description
11:0		R	0x0*	Reserved
12	MB_IntrReq	R	0x0*	<p>Connected to the interrupt controller of the ARC SDP Mainboard.</p> <p>Can be used to provide an interrupt from the peripheral subsystem of the ARC SDP Mainboard to a core on the AXC003 CPU Card.</p> <p>This bit is configured as a level sensitive, active low interrupt.</p>
15:13		R	0x0*	Reserved












19:16		R	0xD*	Reserved
20	MB_SW2504	R/W	0x0	CPU Start button SW2504 on the ARC SDP Mainboard pressed
			0x1*	CPU Start button SW2504 on the ARC SDP Mainboard not pressed
21	MB_SW2506	R/W	0x0	CPU Start button SW2506 on the ARC SDP Mainboard pressed
			0x1*	CPU Start button SW2506 on the ARC SDP Mainboard not pressed
22	MB_SW2505	R	0x0	CPU Start button SW2505 on the ARC SDP Mainboard pressed
			0x1*	CPU Start button SW2505 on the ARC SDP Mainboard not pressed
23	MB_SW2507	R	0x0	CPU Start button SW2507 on the ARC SDP Mainboard pressed
			0x1*	CPU Start button SW2507 on the ARC SDP Mainboard not pressed
31:24		R	0x0*	Reserved

### 9.8.4 GPIO\_EXT\_PORTB: GPIO Port B Input Register

Address offset: 0x3054

Reset value: depends on jumper settings on the AXC003 CPU Card

Table 44 GPIO Port B Input Register (GPIO\_EXT\_PORTB)

Legend: * reset value				
Bit	Name	Access	Value	Description
0	Reserved	R	0 *	
1	JP1201	R	0	 Default setting <sup>1)</sup>
			1	 Reserved
2	JP1202	R	0	 Default setting <sup>1)</sup>
			1	 Reserved
3	JP1203	R	0	 Default setting <sup>1)</sup>
			1	 Reserved
4	JP1204	R	0	 Default setting <sup>1)</sup>
			1	 Reserved
5	JP1205	R	0	 Default setting <sup>1)</sup>
			1	 Reserved
6	JP1206	R	0	 Default setting <sup>1)</sup>

			1	<input type="checkbox"/>	Reserved
7	JP1207	R	0	<input checked="" type="checkbox"/>	Default setting <sup>1)</sup>
			1	<input type="checkbox"/>	Reserved
31:8	Reserved	R	0*		

1) Reset value depends on jumper settings on the AXC003 CPU Card.

*This appendix describes how to mount an AXC003 CPU Card on an ARC SDP Mainboard.*

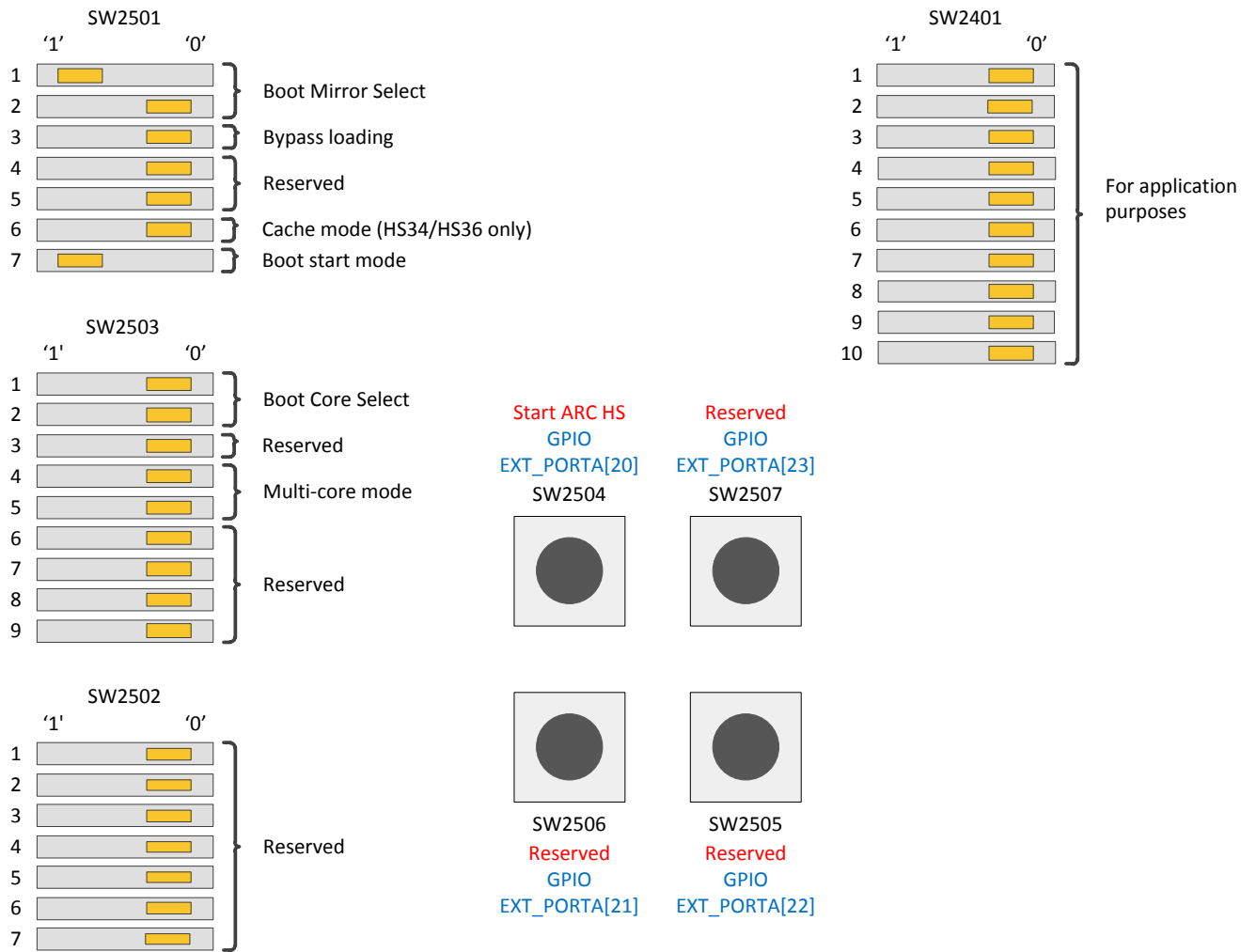
---

### A.1 Mounting the AXC003 CPU Card

Take the following steps to mount the AXC003 CPU Card on an ARC SDP Mainboard:

1. Make sure that the ARC SDP Mainboard is switched off.
2. Mount the AXC003 CPU Card on the ARC SDP Mainboard and make sure that the Power Supply Connector and the HapsTrak II connectors for the CPU Card are connected properly.
3. Make sure that the CPU Card specific DIP switches on the ARC SDP Mainboard are set according to [Figure 54](#) on page 133.

Figure 54 Default Settings of the DIP Switches on the ARC SDP Mainboard.



*This appendix describes how to install and configure PuTTY, a serial console that can be used as a debug console.*

---

### B.1 Installing and Configuring PuTTY

PuTTY is a serial console that can be used intermediately as a simple debug console when the MetaWare Development Toolkit has not yet been installed. This is an optional step, and is only needed if you are interested in the console output of the built-in self-test.

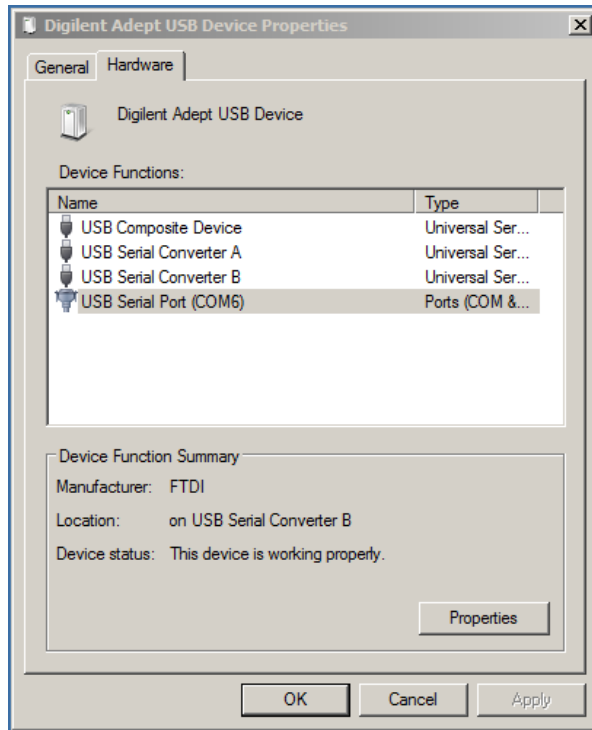
1. Download `putty.exe` from <http://www.putty.org>
2. Make sure that you have connected the USB cable to your computer and that the USB device drivers have been installed as described in the ARC SDP Mainboard User Guide [5].
3. Open the Windows **Control Panel**.
4. In the category **Hardware and Sound**, click **View devices and printers**, then **Digilent Adept USB Device**.

The **Digilent Adept USB Device Properties** windows opens.

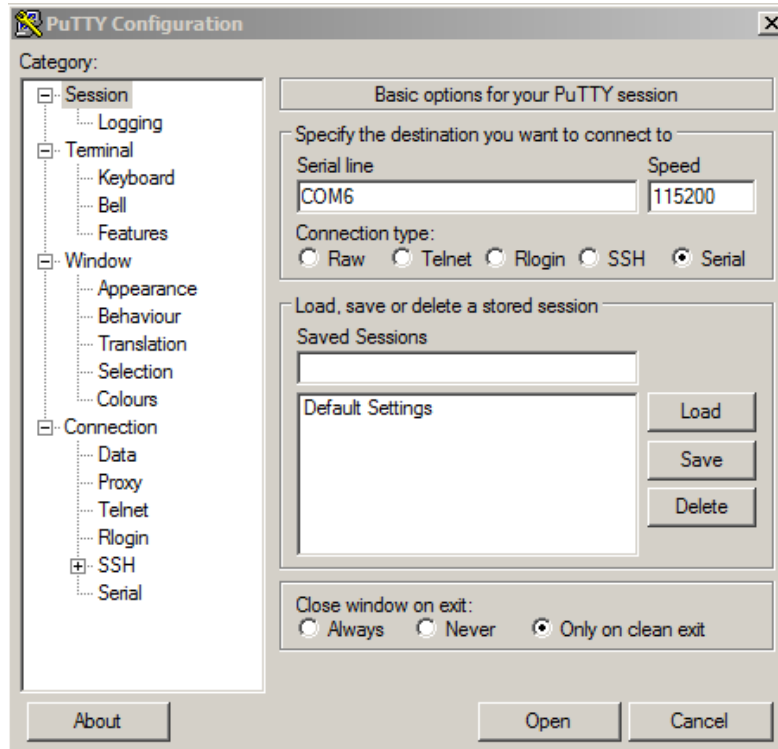
Select the **Hardware** tab and take note of the COM port assigned to the USB Serial Port.

The example in [Figure 55](#) on page 135 uses the COM6 port:

Figure 55 Identification of COM Port



5. Execute `putty.exe`  
The **PuTTY Configuration** window appears.
6. Set the **Connection type** to **Serial**.
7. Enter the name of the COM port in the **Serial line** field
8. Set the **Speed** field to **115200** as shown in [Figure 56](#) on page 136.

Figure 56 *PuTTY Configuration*

9. Click on **Open** to launch the PuTTY terminal.



## Appendix C

This appendix describes all features of the core configurations available in the AXC003 Processor FPGA.

### C.1 Detailed Core Configurations

Configuration Option	Description	HS36	HS38x2
<b>Cluster</b>			
Cluster	Cluster component	+	+
cluster_id	CLUSTER_ID auxiliary register	0	0
pipeline_ibps	Place pipeline registers on internal busses within the cluster to assist timing closure at high frequency	false	false
<b>CPUisle</b>			
arc_num	Processor number as read back in the ARCNUM	5	0
instances	The number of instantiations of this core	1	2
<b>ARCV2HS</b>			
halt_on_reset	The core is halted initially on reset	true	true
byte_order	The endianness of the core	little	little
atomic_option	Enables the LLOCK/SCOND instructions	true	true
div_rem_option	Adds non-blocking multi-cycle implementation of integer divide/remainder functions	radix4_enhanced	radix4_enhanced
mpy_option	The multiplier ISA option	plus_qmacw	plus_qmacw
stack_checking	A mechanism for checking stack accesses	true	true
ll64_option	Support for load and store instructions that transfer register pairs to/from memory	true	true
intvbase_preset	The upper 22 bits of the interrupt vector base configuration register	0	0

Configuration Option	Description	HS36	HS38x2
rgf_num_regs	Size (in 32b registers) of the processor register file	32	32
rgf_num_banks	Number of register banks	1	1
infer_alu_adder	Datapath infer/instantiate	infer	infer
infer_mpy_wtree	Datapath infer/instantiate	infer	infer
mem_initiator_ports	0 no access to system memory 2 instruction and data accesses go through separate interfaces. 1 instruction and data accesses go through a single interface	1	1
power_domains	Adds isolation and power-switch signal inputs for use in UPF-base flow when configuring power domains and generates UPF constraints	false	false
clock_gating	Insert of architectural clock-gating elements	false	false
<b>Branch-Prediction Unit</b>			
br_bc_entries	Number of entries in the branch cache of the branch-prediction unit	512	512
br_pt_entries	Number of two-bit predictors in the branch prediction unit to predict the direction of conditional branches (taken or not taken)	8192	8192
br_rs_entries	Number of entries in the return-address stack of the branch-prediction unit.	4	4
br_bc_full_tag	The size of the tag used in the branch cache of the branch-prediction unit	true	true
br_bc_tag_size	If a partial tag is used, this option sets the size of that tag	4	4
br_tosq_entries	The top-of-stack queue	5	5
br_fb_entries	The size of the buffer that stores fetched instructions that are not yet executed	2	2
ecc_option	Error checking for on-chip RAMs	none	none
grad_entries	Graduation entries track post-commit instructions awaiting result retirement	8	8
uaux_option	User auxiliary register interface	false	false

Configuration Option	Description	HS36	HS38x2
<b>Timer 0</b>		+	+
timer_0_int_level	Interrupt level (and implicitly the priority: level 0 is highest)	1	1
<b>Timer 1</b>		+	+
timer_1_int_level	Interrupt level (and implicitly the priority: level 0 is highest)	0	0
<b>Real-time Counter</b>		+	-
<b>Interrupt Controller</b>			
number_of_interrupts	Total number of interrupts	32	32
number_of_levels	Priority levels in the interrupt controller	2	2
external_interrupts	Total interrupt pins available for external system components	12	27
firq_option	Fast-interrupts option	false	false
<b>Actionpoints</b>		+	+
num_actionpoint	Number of trigger events	8	8
aps_feature	Actionpoint feature set	min	min
<b>SmaRT</b>		+	+
smart_stack_entries	Number of entries in the trace buffer	8	128
smart_implementation	Flip-flop or memory-based	flip-flop	flip-flop
<b>Performance Monitor</b>		-	+
pct_counters	Number of counters for performance monitoring	-	8
pct_interrupt	When a counter overflows, an interrupt is generated	-	true
<b>Real-time trace producer</b>		+	+
rtt_feature_level	'small' - program trace only is available. 'medium' adds data trace. 'full' adds core and aux register trace.	full	full

Configuration Option	Description	HS36	HS38x2
<b>ARC RTT</b>			
has_nexus_if	Nexus interface to offload the data from RTT	true	true
has_on_chip_mem	On-chip memory option to store the trace data in shared memory	true	true
nexus_data_wdt	Nexus data width to offload the data from RTT	16	16
internal_memory_size	Internal memory size to capture the trace data	16k	16k
ram_type	Types of internal memories to be inferred for the logic	1_PORT	1_PORT
rtt_power_domains	Isolation signal inputs and power-switch controls for use in UPF flow when configuring power domains	false	false
<b>Memory Protection Unit</b>		+	-
mpu_num_regions	Number of configured memory regions	8	-
<b>Memory Management Unit</b>		-	+
mmu_ntlb_num_entries	Number of joint TLB normal page entries	-	512
mmu_page_size_sel_0	Page size of each joint TLB normal page	-	8K
mmu_stlb_num_entries	Number of joint TLB super-page entries	-	16
mmu_page_size_sel_1	Page size of each joint TLB super page	-	2M
mmu_pae_enabled	PAE provides for a 40-bit physical memory address.	-	true
mmu_shared_lib	Shared-library ASID feature	-	true
<b>Floating Point Unit</b>			
fpu_dp_option	This enables double-precision floating point instructions	true	true
fpu_div_option	This enables divide & square-root instructions	true	true
fpu_fma_option	This enables the multiply & accumulate instructions	true	true
<b>Data Cache</b>			
dc_size	Total size of the data cache in bytes	65536	65536

Configuration Option	Description	HS36	HS38x2
dc_bsize	Cache-line length in bytes	32	64
dc_bus_data_width	Cache-bus width for refills and evictions	64	64
dc_mem_cycles	Number of cycles dedicated to the data cache data memories	2	2
dc_mem_posedge	Clock the data--cache memories on the positive edge of the clock	true	true
dc_uncached_region	Uncached data cache region specified by a single auxiliary register	false	true
<b>Instruction Cache</b>			
ic_size	Total size of the instruction cache in bytes	65536	65536
ic_ways	Number of cache ways	2	4
ic_bsize	Cache-line length in bytes	32	64
ic_disable_on_reset	Instruction cache is disabled on reset	false	false
ic_pipeline_bus	Insert a pipeline register on the instruction cache's refill bus from memory. This option can be used to ease timing in configurations where cores are not closely located with system-level cache, for example.	false	false
<b>DCCM</b>			
dccm_size	Size of the Data Closely Coupled Memory (DCCM) in bytes	262144	-
dccm_dmi	External access through a DMI port	false	-
dccm_mem_cycles	Number of cycles dedicated to the each DCCM memory bank	2	-
dccm_mem_posedge	Clock the DCCM memory banks on the positive edge of the clock	true	-
dccm_mem_banks	Number of DCCM memory banks	4	-
<b>ICCM0</b>			
iccm0_size	Size of ICCM0 in bytes	262144	-
iccm0_base	Initial memory region assignment for ICCM0	1	-
iccm0_dmi	External access through a DMI port	false	-

Configuration Option	Description	HS36	HS38x2
<b>Bus Interface Unit</b>			
biu_mem_bus_num	Number of memory busses (ignored if system-level cache is present)	2	1
biu_mem_bus_option	Protocol to connect to external memory	AXI	AXI
biu_mem_bus_data_w	Data width of the memory busses	64	64
biu_per_bus_option	Protocol to connect to external peripherals	AXI	AXI
biu_dmi_bus_num	Valid when one of the following is configured: <ul style="list-style-type: none"> <li>iccm0_dmi</li> <li>iccm1_dmi</li> <li>dccm_dmi</li> </ul>	1	n/a
biu_dmi_bus_option	Protocol to access CCMs from external bus devices	AXI	AXI
biu_dmi_bus_data_w	Data width of the DMI busses	64	64
biu_ioc_bus_num	The number of I/O coherency busses.	0	1
biu_ioc_bus_axi_idw	This specifies the AXI ID width of the IOC bus.	n/a	16
<b>Coherency Unit</b>		-	+
has_coherent_dma	Coherent I/O through the system bus	-	True
stb_entries	Maximum number of active coherency transactions	-	8
<b>System-level cache</b>		-	+
slc_size	SLC size in bytes	-	524288
slc_line_size	SLC line size	-	64
slc_ways	Number of SLC ways	-	4
slc_tag_banks	Number of tag banks	-	4
slc_tram_delay	Cycle delay for the tag RAM	-	2
slc_data_banks	Number of SLC data banks	-	8
slc_dram_delay	Cycle delay for the data RAM	-	3
slc_data_halfcycle_steal	Adds a register in front of <code>slc_data_ram</code> and clocks <code>slc_data_ram</code> on negative edge. Use only when <code>slc_data_size &gt;= 512KB</code>	-	false

Configuration Option	Description	HS36	HS38x2
slc_clock_gating	Inserts architectural clock-gating elements in the design. Set to <code>false</code> for certain FPGA tools	-	<code>false</code>
slc_mem_bus_width	Width of data connection to external memory	-	64
slc_ecc_option	Type of protection for the memories	-	None
<b>ARConnect</b>		-	+
mcip_has_intrpt	Inter-core interrupt unit	-	<code>true</code>
mcip_has_sema	Inter-core semaphore unit	-	<code>true</code>
mcip_sema_num	The number of semaphores	-	16
mcip_has_msg_sram	Inter-core message unit	-	<code>true</code>
mcip_msg_sram_size	The bytes of SRAM in the Inter-core message unit	-	512
mcip_has_debug	Inter-core debug unit	-	<code>true</code>
mcip_has_grtc	Global real-time counter unit	-	<code>true</code>
mcip_has_pmu	Power-management unit	-	<code>false</code>
mcip_has_llm	Shared low-latency memory unit	-	<code>false</code>
mcip_has_idu	Interrupt-distribution unit	-	<code>true</code>
mcip_idu_cirq_num	Number of common interrupts supported by IDU	-	8
mcip_has_bsu	Bus slave unit	-	<code>false</code>

# Glossary and References

---

*This chapter contains a list of specific terms used in this document and references for further reading.*

---

## Glossary

**AHB**

Advanced High Performance Bus

**AXI**

Advanced eXtensible Interface

**CGU**

Clock Generator Unit

**DDR3**

Double Data Rate 2

**GPIO**

General Purpose Input/Output

**HW**

Hardware

**HAPS**

High performance ASIC Prototyping System; FPGA based prototyping system of Synopsys

**HapsTrak II**

Standard (SAMTEC) connector type used on HAPS

**IC**

Integrated Circuit

**I<sup>2</sup>S**

Inter-IC Sound, serial bus interface standard for the transfer of audio data

**JTAG**

Joint Test Action Group

**R**

Read-only register

**RW**

Read-write register

**RW1C**

Read-write register; writing a one clears the corresponding bit



**SDP**

Software Development Platform

**SPDIF**

Sony/Philips Digital Interface

**SDRAM**

Synchronous Dynamic Random Access Memory

**SRAM**

Static Random Access Memory

**SW**

Software

---

## References

- [1] *HapsTrak II standard*
- [2] *C/C++ Programmer's Guide for the MetaWare Compiler*
- [3] *Synopsys DesignWare dw\_apb\_gpio Databook* <http://www.synopsys.com>
- [4] *ARC SDP download webpage*  
*You have received the corresponding URL during the purchasing process*
- [5] *ARC SDP Mainboard User Guide*  
*This document can be downloaded from the ARC SDP download webpage.*