Mercury SA1 SoC Module

Reference Design for Mercury+ ST1 Base Board User Manual

Purpose

The purpose of this document is to present to the user the overall view of the Mercury SA1 SoC module reference design and to provide the user with a step-by-step guide to the complete Intel® SoC design flow used for the Mercury SA1 SoC module.

Summary

This document first gives an overview of the Mercury SA1 SoC module reference design and then guides through the complete Intel SoC design flow for the Mercury SA1 SoC module in the getting started section. In addition, the internals and the boot options of the Mercury SA1 SoC module reference design are described.

Product Information	Code	Name
Module	ME-SA1	Mercury SA1 SoC Module
Baseboard	ME-ST1	Mercury+ ST1 Base Board

Document Information	Reference	Version	Date
Reference / Version / Date	D-0000-490-008	2023.1_v1.0.0	18.04.2024

Approval Information	Name	Position	Date
Written by	ARUD	FPGA/SoC Embedded Software Engineer	14.04.2024
Verified by	ABUE	FPGA/SoC Senior Embedded Software Engineer	18.04.2024
Approved by	IJOS	Manager, BU SP	18.04.2024

Table of Contents

1 Overview

1.1 Introduction

The Mercury SA1 SoC module reference design demonstrates a system using the Mercury SA1 SoC module in combination with the Mercury+ ST1 base board. It presents the basic configuration of the device and contains a guided getting started tutorial.

A troubleshooting section is included at the end of the document to help the user solve potential issues related to board connectivity and system functionality.

This reference design includes a basic HelloWorld software example. In addition, Enclustra provides Application Notes [6] for selected applications.

An introduction to the Intel tools is provided by the documents below:

- HPS SoC Boot Guide – Cyclone V SoC Development Kit [16]
- Intel® Quartus® Prime Pro and Standard Software User Guides [8]
- Rocketboards Bootloader Guide [10]

More information on the Mercury SA1 SoC module and the Mercury+ ST1 base board can be found in the Mercury SA1 SoC Module User Manual [2] and the Mercury+ ST1 Base Board User Manual [3].

The following directory structure applies to the SA1 Reference Design:

Directory	Description
`doc`	Reference Design documentation
`scripts`	Scripts directory required for project creation and settings
`src`	Design pinout, timing constraints and VHDL source code directory
`sw`	HelloWorld application source files

Table 1: Directory Structure

1.2 Path Variables

The following variables are used to substitute the full path to directories relevant for the reference design build process. Paths not already present in the reference design directory will be created during the build process.

Variable	Full Path	Description
<base_dir>	<base_dir>	Reference design root directory as described in Section 1.1
<quartus_install_dir>	<quartus_install_dir>	Path to the Quartus 23.1 installation directory
<quartus_dir>	<base_dir>/Quartus/<module_name>/<boot_mode>	Quartus project directory
<quartus_outdir>	<base_dir>/Quartus/<module_name>/<boot_mode>/output_files	Quartus output directory
<ebe_dir>	<base_dir>/sw/bsp-altera	Enclustra Build Environment repository [13]
<u-boot_dir>	<base_dir>/sw/bsp-altera/sources/altera-uboot	U-boot source directory
<app_dir>	<base_dir>/sw/HelloWorld	HelloWorld source directory
<hwlib_dir>	<base_dir>/sw/intel-socfpga-hwlib	Intel SoC FPGA HWLIB repository [9]
<hps_handoff_dir>	<base_dir>/Quartus/<module_name>/<boot_mode>/hps_isw_handoff/Mercury_SA1_pd_hps_0	HPS handoff file directory

Table 2: Path Variables

1.3 Prerequisites

- IT infrastructure
  - Computer with a microSD card slot (optional[1])
  - Supported Linux OS[2] [7]
- Software
  - Intel® Quartus® Prime Software Suite 23.1 Standard or Lite Edition[3]
  - ARM GNU Toolchain [11]
- Hardware
  - Enclustra Mercury SA1 SoC module
  - Enclustra Mercury+ ST1 base board
- Accessories
  - 12 V DC power supply
  - Micro-USB cable
  - microSD card (optional[1])
  - USB-Blaster/USB-Blaster II download cable (optional[4])

NOTICE

Damage to the device due to USB-Blaster cable incompatibilities

Refer to the Mercury SA1 SoC module User Manual [2] to find detailed information about a suitable download cable.

[1]Only required for SD card boot mode.
[2]This reference design was built and tested with Ubuntu 20.04 and Ubuntu 22.04.
[3]Check the Mercury SA1 SoC Module User Manual [2] for details on device support in Quartus tools.
[4]Only required for loading the bitstream and/or programming the QSPI flash via Quartus.

2 Reference Design Description

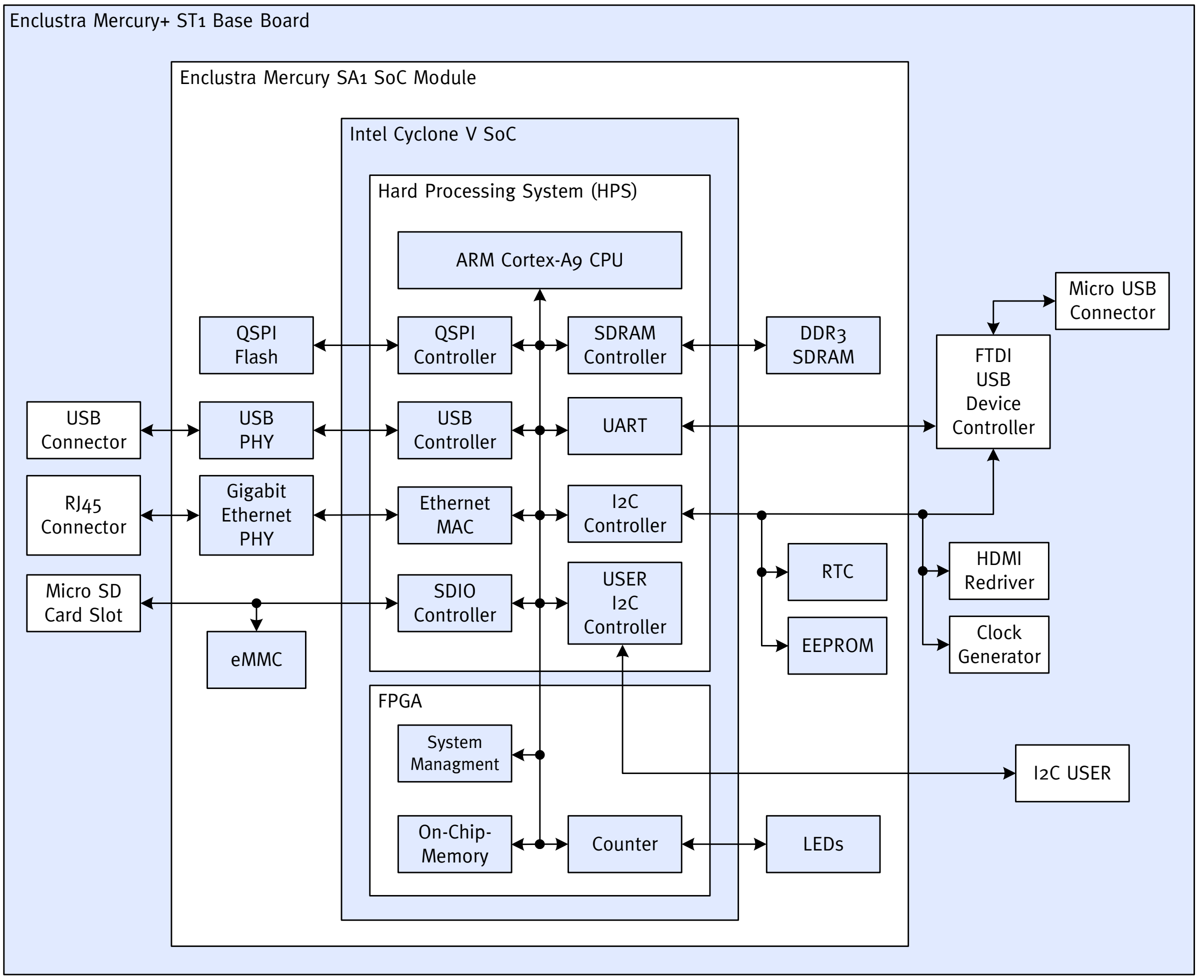

Figure 1: Hardware Block Diagram

2.1 Hard Processor System (HPS)

2.1.1 Clocks

The HPS input clock frequency is configured to 50 MHz. The HPS CPU (MPU) clock frequency is configured to its corresponding maximum frequency, as specified in the Intel® Cyclone V Device Datasheet [15]. The maximum MPU clock performance depends on the device speed grade and the VCC_INT voltage. Refer to the Mercury SA1 SoC Module User Manual [2] for details.

In addition, a 50 MHz clock and a 100 MHz clock are exported from the HPS to the FPGA. These clocks can be modified within the Platform Designer in the Cyclone V HPS properties.

2.1.2 PS DDR3 SDRAM

The DDR3 SDRAM memory is attached to the HPS and runs at its corresponding maximum DDR frequency.

The DDR3 configuration can be modified in the Cyclone V Hard Processor System properties in the Platform Designer.

2.1.3 SD Card/eMMC

The SD/MMC controller is mapped to the HPS dedicated pins HPS_GPIO[36,38:39,45:47] on HPS I/O Set 0. The data width is 4 bit. The controller can be connected to an external SD card on the base board or to the on-board eMMC flash. This enables SD card and eMMC flash access, as well as booting from the SD card, respectively eMMC flash.

For configuring the Mercury SA1 SoC module to boot from the SD card, the configuration DIP switches on the Mercury+ ST1 base board must be set according to Section 4.2.3. For details on the eMMC flash boot mode, refer to the Mercury SA1 SoC Module User Manual [2].

The SDIO interface signals are shared between the SD card and the eMMC flash. Only one of those storage devices can be used at a given moment. Switching between them at runtime is possible with the EMMC_EN# signal on HPS_GPIO[53].

2.1.4 I2C

The I2C controller I2C0 is mapped to HPS dedicated pins HPS_GPIO[54:56] on HPS I/O Set 0. The pins are connected to the I2C bus on the Mercury+ ST1 base board.

The I2C controller I2C1 is connected through the FPGA fabric to the I2C_FPGA bus on the Mercury+ ST1 base board.

For available I2C devices refer to the Mercury SA1 SoC Module User Manual [2] and the Mercury+ ST1 Base Board User Manual [3].

2.1.5 Quad SPI Flash Controller

The quad SPI flash controller is mapped to the HPS dedicated pins HPS_GPIO[29:34] on HPS I/O Set 1 in single slave select mode (1SS).

To configure the Mercury SA1 SoC module to boot from the QSPI flash, the hardware configuration on the Mercury+ ST1 base board must be done according to Section 4.3.4. Refer to the Mercury SA1 SoC Module User Manual [2] for details about flash programming and usage.

2.1.6 UART

The UART0 controller is mapped to the HPS dedicated pins HPS_GPIO[65:66] on HPS I/O Set 2. It is connected to the FTDI USB device controller on the Mercury+ ST1 base board. The UART is configured as shown in Table 3.

Parameter	Value
Baud rate	115'200
Data	8 bit
Parity	None
Stop	1 bit
Flow control	None

Table 3: UART Configuration

2.1.7 Ethernet

The Ethernet MAC EMAC1 is mapped to the HPS dedicated pins HPS_GPIO[14:25] on HPS I/O Set 0. It is connected to a Micrel KSZ9031 Ethernet PHY on the Mercury SA1 SoC module using an RGMII interface. The PHY can be configured via the MDIO interface on PHY address 3.

2.1.8 USB

The USB controller USB1 is mapped to the HPS dedicated pins HPS_GPIO[0:13] on HPS I/O Set 0. It is connected to a USB3320C USB 2.0 PHY. This interface can be configured for USB host or device mode.

2.1.9 HPS GPIOs

The unused pins from the HPS are available as GPIOs. For details on the I/O assignment, refer to the Mercury SA1 SoC module User Manual [2], Section "HPS dedicated and FPGA/HPS shared I/O Pins". For details on the combination of the I/O assignment with the Mercury+ ST1 base board, refer to the Mercury+ ST1 Base Board User Manual [3].

2.2 FPGA Fabric

2.2.1 GPIOs

The FPGA firmware contains a 24-bit counter freely running at 50 MHz. The MSB of this counter is used to blink LED0# with a frequency of approximately 3 Hz.

FPGA Pin	Signal	Function
AH12	LED0#	Blinking LED counter MSB

Table 4: FPGA Firmware I/O Configuration

3 Getting Started

This section describes the steps required to configure the Mercury SA1 SoC module and Mercury+ ST1 base board in order to run a simple HelloWorld application example:

1. Mount the module and configure the Mercury+ ST1 base board.
2. Generate the FPGA bitstream.
3. Prepare the software workspace.
4. Run a software application.

Read the Mercury SA1 SoC module User Manual [2] and the Mercury+ ST1 Base Board User Manual [3] carefully before proceeding.

3.1 Essential Information

Pre-generated binaries may be used instead of building them manually as described in the following sections. The binaries for any supported SA1 variant and boot mode are released on the SA1 Reference Design Github release page [5].

> **Tip**
>
> Workarounds and fixes for potential issues can be found in Section 5.

NOTICE

Damage to the device due to overheating

Depending on the user application, the Mercury SA1 SoC module may consume more power than can be dissipated without additional cooling measures.

- Ensure that the SoC is always adequately cooled by installing a heat sink and/or providing air flow.

3.2 Setting up the Hardware

NOTICE

Damage to the device when mounting or removing the module

Mounting or removing the module while the base board is powered can lead to damage to the module or the base board.

- Ensure that the base board is not powered before mounting or removing the module.

The assembly drawing of the Mercury+ ST1 base board is shown in Figure 2. The relevant interfaces are marked in red in the figure as well as in the instructions below.

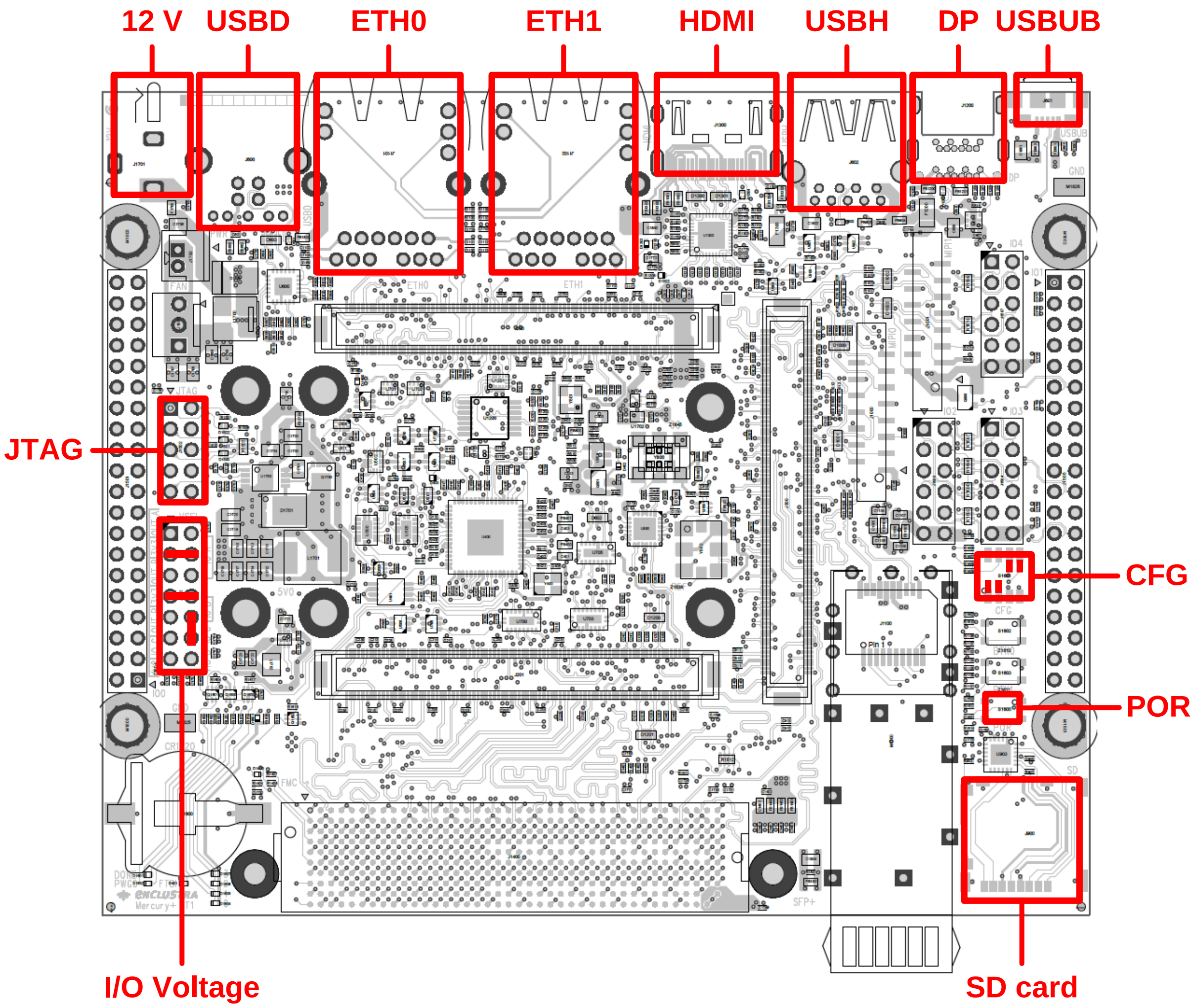

Figure 2: Mercury+ ST1 Base Board Assembly Drawing (Top View)

1. Make sure no power is connected to the power connector of the Mercury+ ST1 base board (label **12 V**).
2. Set the I/O voltage jumpers on the Mercury+ ST1 base board according to label **I/O Voltage** (the jumpers are marked with red rectangles).
    - VCC_IO_A to VCC_3V3 (3.3 V)
    - VCC_IO_B to VCC_3V3 (3.3 V)
    - VCC_IO_C to VCC_IO_B (3.3 V)
3. Set the configuration DIP switch on the Mercury+ ST1 base board as follows (label **CFG**):
    - CFG = [1: OFF, 2: OFF, 3: ON, 4: ON]
4. Mount the Mercury SA1 SoC module to the Mercury+ ST1 base board.
5. Connect the Micro-USB cable between the computer and the Mercury+ ST1 base board using the Micro-USB port labeled **USBUB**.
6. Two new serial ports should be detected by the Linux OS[5], where `<X>` denotes the lowest number of those:
    - `/dev/ttyUSB<X>`
    - `/dev/ttyUSB<X+1>`
7. Open the serial port with the highest number of the two detected ports and configure it using the parameters specified in Section 2.1.6. Any suitable utility for establishing a serial connection can

[5]For issues related to serial port detection, refer to Section 5.3.

be used. Below, the `screen` command is used executed in a Linux terminal:
`screen /dev/ttyUSB<X+1> 115200`
The output of the HelloWorld application will be printed in this terminal after following Section 4.

8. Optional: Connect a compatible JTAG adapter (see Section 1.3) to the JTAG connector (label **JTAG**).
9. **NOTICE: Damage to the device when applying power. Ensure that the mounting holes on the base board are aligned with the mounting holes of the Mercury SA1 SoC module.**
   Connect the 12 V DC power supply plug to the power connector of the Mercury+ ST1 base board (label **12 V**).

3.3 Generating the FPGA Bitstream

Follow the steps described in this section to generate the reference design bitstream manually.

> **Tip**
>
> Pre-generated bitstreams for all supported SA1 variants are available on the SA1 reference design Github release page [5]. These can be used directly instead of generating the bitstream manually. If using those files directly, the bitstream does not need to be generated manually and this section can be skipped.

1. Configure the `<base_dir>/scripts/settings.tcl` file[6]:
   (a) Set the `module_name` variable to the desired module variant.
   (b) Set the `boot_mode` variable to the desired boot mode.
   (c) Save the file after editing.
2. Start Quartus Prime 23.1, for example, `<quartus_install_dir>/quartus/bin/quartus &`
3. Create the Mercury SA1 SoC module reference design project:
   (a) Select **View** > **Utility Windows** > **Tcl Console** to open the Tcl console.
   (b) Click inside the Tcl console:
      i. Type `cd {<base_dir>}`
         Note the curly brackets around the path.
      ii. Type `source ./scripts/create_project.tcl`
      iii. Wait for completion.
4. Open the Mercury SA1 SoC module reference design project:
   (a) Select **File** > **Open Project...**.
   (b) Navigate to `<base_dir>/Quartus/<module_name>/<boot_mode>`
   (c) Select the **Mercury_SA1_ST1.qpf** project file.
   (d) Click **Open**.
5. Compile the design and generate the programming files:
   (a) Go to **Tasks**.
   (b) Double-click **Compile Design**.
   (c) Wait for completion.
   (d) Check for errors and critical warnings in the **Messages** window.

The build output is located in `<base_dir>/Quartus/<module_name>/<boot_mode>/output_files` and is described in Table 5:

File	Description
`Mercury_SA1_ST1.sof`	Bitstream

Table 5: Quartus Programming Files

[6]Valid values for the variables are listed at the beginning of the file.

3.4 Programming the FPGA

Follow the steps described in this section to program the FPGA bitstream using the Quartus Programmer from Quartus 23.1 (see Figure 3).

1. Connect a compatible JTAG adapter (see Section 1.3) to the JTAG connector (see label **JTAG** in Figure 2).
2. Select **Tools** > **Programmer**.
3. Select **Hardware Setup**.
4. Under **Currently selected hardware**, select **USB-Blaster/USB-Blaster II** from the drop-down list.
5. Close the **Hardware Setup** window.
6. In the **Programmer** window, click **Auto Detect**.
7. If prompted, select the device according to the product variant:
    - ME-SA1-C6-7I-D10: 5CSXFC6C6 or 5CSXFC6C6U23

    and click **OK**.
8. If the correct programming file is not automatically selected by the tool, do the following:
    (a) Click on the device just added.
    (b) Select **Change File...**.
    (c) Navigate to `<base_dir>/Quartus/<module_name>/<boot_mode>/output_files`
    (d) Select the bitstream file (`Mercury_SA1_ST1.sof`).
    (e) Click **Open**.
9. Enable the **Program/Configure** checkbox for the device.
10. Click **Start**.
11. After the FPGA is successfully configured, the **DONE** LED lights up.

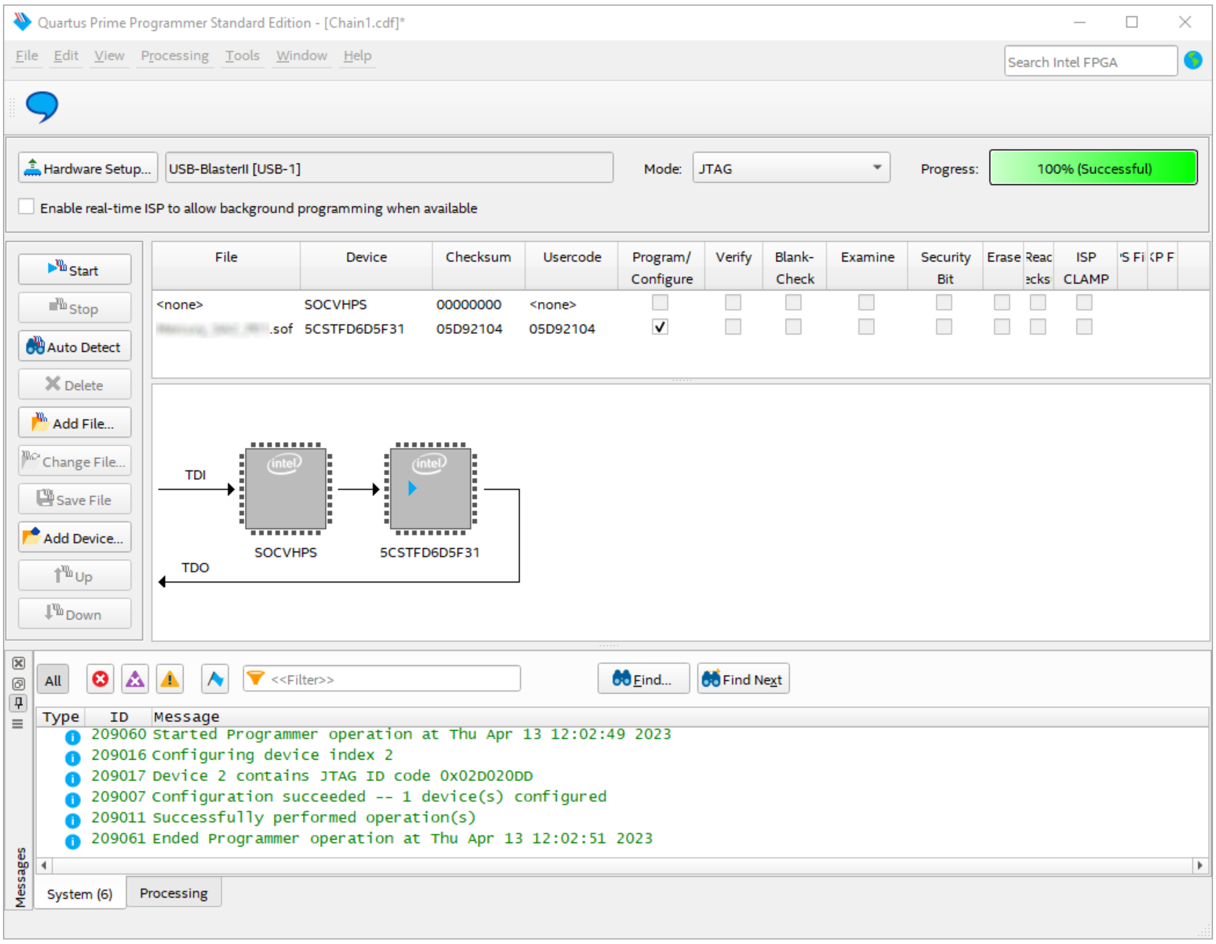

Figure 3: Quartus Programmer Program Bitstream

3.5 Generating the Bootloader

The bootloader (SPL) is used in the boot image creation process described further in Sections 4.2.1 and 4.3.1. The SPL is built alongside U-Boot and requires handoff files from the Quartus design. These handoff files contain HPS settings, memory parameters and address mappings.

Tip

Pre-generated bootloader binaries for the respective boot mode are available on the reference design github release page [5]. However, the bootloader needs to be re-generated whenever the HPS settings are changed.

Open a Linux terminal and execute the following commands to generate the SPL binary:

1. ```
   git clone https://github.com/enclustra-bsp/bsp-altera \
   --branch intel-v1.9 \
   <base_dir>/sw/bsp-altera
   ```
2. ```
   cd <base_dir>/sw/bsp-altera
   ```
3. ```
   ./build.sh \
   -d Mercury_SA1/Mercury_ST1/<boot_mode>[7] \
   -o <module_name> \
   -x U-Boot \
   -B handoff <hps_handoff_dir>[8]
   ```

[7] In EBE, the boot mode can be "MMC", "EMMC" or "QSPI".
[8] Use the absolute path to the HPS handoff directory.
   ```

The build output is located in `<base_dir>/sw/bsp-altera/sources/altera-uboot` and is described in Table 6:

File	Description
`spl/u-boot-spl`	SPL ELF executable
`u-boot`	U-Boot ELF executable
`u-boot-with-spl.sfp`	Bootable file containing four SPL copies and one copy of the U-Boot image

Table 6: SPL Output Binaries

For more details regarding the SPL build process refer to [14].

3.6 Compiling the HelloWorld Application

The ARM GNU Toolchain [11] is used to compile the HelloWorld application example. The steps described in the following sections target a Linux Ubuntu OS and have been verified with Ubuntu 20.04 and Ubuntu 22.04.

3.6.1 Installing the ARM GNU Toolchain

The easiest way to install the ARM GNU Toolchain [11] is to use the provided package manager of the Linux OS. Execute these commands in a Linux terminal to install the toolchain:

1. `sudo apt-get update`
2. `sudo apt-get upgrade`
3. `sudo apt-get install gcc-arm-none-eabi`

Alternatively, the ARM GNU Toolchain can be installed manually. Download the appropriate package (AArch32 bare-metal target (arm-none-eabi)) from the ARM GNU Toolchain download page [12]. Extract the archive to the desired location. Add the `bin` folder to the **PATH** variable to make the toolchain available. In addition, install the following packages:

1. `sudo apt-get install build-essential`
2. `sudo apt-get install git`
3. `sudo apt-get install u-boot-tools`
4. `sudo apt-get install mtools`

3.6.2 Building the HelloWorld Application

Before running the commands below, ensure that the ARM GNU Toolchain was successfully installed, as described in Section 3.6.1.
Open a Linux terminal and execute the following commands to build the HelloWorld application:

1. `git clone https://github.com/altera-opensource/intel-socfpga-hwlib \`
   `--branch rel_master_23.12.02_pr \`
   `<hwlib_dir>`
2. `export INTEL_HWLIB_DIR=<absolute-path-to-<hwlib_dir>>`
3. `cd <base_dir>/sw/HelloWorld`
4. `make`

The build output is located in `<base_dir>/sw/HelloWorld` and described in Table 7:

File	Description
`hello.axf`	Arm Executable Format file
`hello.axf.objdump`	Disassembly information of the HelloWorld binary
`hello.axf.map`	Memory map of the HelloWorld binary
`hello.bin`	Binary copy of hello.axf generated with `arm-none-eabi-objcopy`
`hello.img`	HelloWorld image file generated with `mkimage`

Table 7: HelloWorld Output Files

4 Boot Configurations

A boot image can be created containing the previously generated binaries to enable booting from various boot modes. The boot image contains the bootloader and the software bare-metal application. This section describes how to generate a boot image for the Mercury SA1 SoC module for each supported boot mode.

> **Tip**
>
> Pre-generated images may be used for booting instead of rebuilding the image. The binaries for any supported SA1 variant and boot mode are released on the SA1 Reference Design Github release page [5].

4.1 Preparing the Boot Binaries

Follow these steps to prepare the boot binaries:

1. Build the SPL as described in Section 3.5.
2. Build the HelloWorld application as described in Section 3.6.

The Enclustra reference design does not provide functionality for loading the bitstream via SPL. Intel provides a bare-metal example for loading the bitstream targeting Cyclone V devices [9].

The next step is optional and shows the conversion of the bitstream to a suitable format if the functionality for loading the bitstream should be added to the SPL by the user. In a Linux terminal, run the following command:

```
<quartus_install_dir>/quartus/bin/quartus_cpf -c \
<base_dir>/Quartus/<module_name>/<boot_mode>/output_files/Mercury_SA1_ST1.sof \
<base_dir>/Quartus/<module_name>/<boot_mode>/output_files/Mercury_SA1_ST1.rbf
```

4.2 SD Card Boot

4.2.1 Generating the Image Files

1. Create a 200 MB sd card image with the required partitions[9]. In a Linux terminal, run the following commands:
   (a)
   ```
   mkdir -p <app_dir>/bootimage
   ```
   (b)
   ```
   dd if=/dev/zero of=<app_dir>/bootimage/sdmmc.img bs=64K count=3216
   ```
   (c)
   ```
   printf "type=c, size=190MiB \n type=A2, size=10MiB" | sfdisk \
   <app_dir>/bootimage/sdmmc.img
   ```
2. Prepare the 190 MB FAT file system:
   (a)
   ```
   dd if=/dev/zero of=<app_dir>/bootimage/fat.img bs=64K count=3040
   ```
   (b)
   ```
   mkfs.vfat -F 32 <app_dir>/bootimage/fat.img
   ```
3. (Optional) Copy the bitstream to the FAT file system:
   ```
   mcopy -i <app_dir>/bootimage/fat.img <quartus_outdir>/Mercury_SA1_ST1.rbf \
   ::Mercury_SA1_ST1.rbf
   ```
4. Merge the FAT file system with the SD card image:
   ```
   dd if=<app_dir>/bootimage/fat.img of=<app_dir>/bootimage/sdmmc.img seek=16 bs=64K
   ```
5. Append the HelloWorld image to the SPL:
   ```
   cat <u-boot_dir>/spl/u-boot-splx4.sfp <app_dir>/hello.img > <app_dir>/hello-with-spl.sfp
   ```
6. Write the SPL containing the HelloWorld image to the Intel A2 boot partition:
   ```
   dd if=<app_dir>/hello-with-spl.sfp of=<app_dir>/bootimage/sdmmc.img seek=3056 bs=64K
   ```

4.2.2 Programming the SD Card

1. Insert a microSD card into the SD card slot of the computer.
2. In a Linux terminal, check the device node of the inserted microSD card:
   ```
   dmesg | tail
   ```
   An example for the console output is given in Figure 4, where the device node is named `sdd` .
3. **NOTICE: Data loss due to overwriting. Use the `lsblk` command to identify the correct device node.**
   Write the SD card image created in Section 4.2.1 to the SD card, replacing `sd<X>` by the identified device node:
   ```
   dd if=<app_dir>/bootimage/sdmmc.img of=/dev/sd<X> bs=4M && sync
   ```

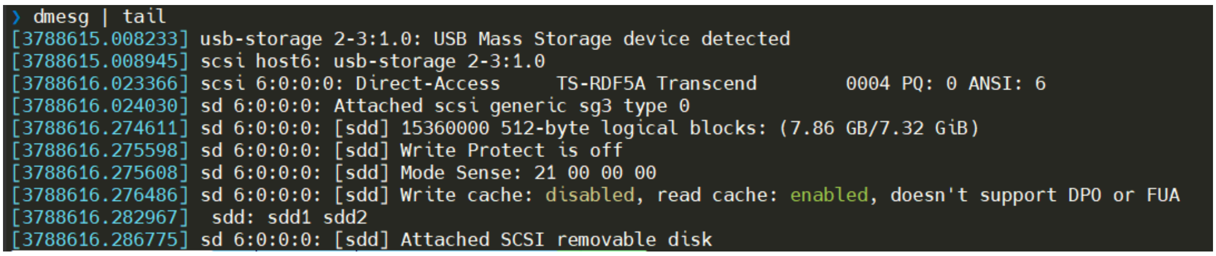

Figure 4: Example of an SD Card Device Node

4.2.3 Preparing the Hardware

1. Detach the power supply of the Mercury+ ST1 base board(see label **12 V** in Figure 2).
2. Enable the SD card boot mode (default) by setting the configuration DIP switch on the Mercury+ ST1 base board as follows (see label **CFG** in Figure 2):
   - CFG = [1: OFF, 2: OFF, 3: ON, 4: ON]

[9] The FAT partition is only used for copying the RBF file if it was created in Section 3.5 and otherwise empty.

4.2.4 Booting from the SD Card

1. Insert the SD card into the SD card slot of the Mercury+ ST1 base board (see label **SD Card** in Figure 2).
2. Connect the power supply to the Mercury+ ST1 base board (see label **12 V** in Figure 2).

4.3 QSPI Flash Boot

4.3.1 Preparing the QSPI Binaries

Before preparing the QSPI binaries, the procedure described in Sections 3.3, 3.5 and 4.1 must be successfully completed with the `boot_mode` variable set to `qspi`. The files needed for programming the QSPI flash are listed in Table 8.

File Path	Description
`<u-boot_dir>/spl/u-boot-splx4.sfp`	Bootloader (SPL)
`<app_dir>/hello.img`	Image file containing the HelloWorld application

Table 8: QSPI Programming Files

4.3.2 Preparing the Hardware

1. Disconnect the power supply of the Mercury+ ST1 base board (see label **12 V** in Figure 2).
2. Disconnect all USB cables from the Mercury+ ST1 base board.
3. Set the configuration DIP switch on the Mercury+ ST1 base board as follows (see label **CFG** in Figure 2):
    - CFG = [1: ON, 2: OFF, 3: ON, 4: ON]
4. Connect the Micro-USB cable between the computer and the Mercury+ ST1 base board using the Micro-USB port labeled **USBUB** in Figure 2.
5. Reconnect the UART terminal if necessary.
6. Connect a compatible JTAG adapter (see Section 1.3) to the JTAG connector (see label **JTAG** in Figure 2).
7. Reconnect the power supply of the Mercury+ ST1 base board (see label **12 V** in Figure 2).

4.3.3 Programming the QSPI Flash

The process of programming large files (such as the FPGA bitstream) to the QSPI flash via JTAG using the `quartus_hps` tool can take up to 30 minutes[10].

> **Tip**
>
> The pre-generated binaries include an image file containing all individual parts of the QSPI flash at the respective offsets. Instead of programming each file one by one, this QSPI flash image can be used to program the complete flash at once. The image file is provided as part of the release binaries on the SA1 Reference Design Github release page [5].

Execute the following commands in a Linux terminal to program the individual files generated in the previous sections to the respective offsets:

1. Erase the whole QSPI flash chip before programming the binaries:
   ```
   <quartus_install_dir>/bin/quartus_hps -c 1 -o E
   ```
2. Program the SPL to QSPI flash address `0`:
   ```
   <quartus_install_dir>/bin/quartus_hps -c 1 -o PV -a 0x0 \
   <u-boot_dir>/spl/u-boot-splx4.sfp
   ```
3. Program the HelloWorld application image file to the QSPI flash address offset `0x40000`:
   ```
   <quartus_install_dir>/bin/quartus_hps -c 1 -o PV -a 0x40000 \
   <app_dir>/hello.img
   ```

[10] Alternatively, the Enclustra MCT [4] can be used to program the QSPI flash.

4.3.4 Booting from the QSPI Flash

1. Check that the hardware configuration is done according to Section 4.3.2.
2. Press the power-on reset button (see label **POR** in Figure 2) and release it after one second.

The expected output in the serial terminal console is given in Figure 5.

```
U-Boot SPL 2023.01-g2b1bf9af (Nov 30 2023 - 16:28:24 +0100)
Trying to boot from SPI

Enclustra Hello World Example
Compiled at: Nov 28 2023 10:04:59
Example completed successfully.
```

Figure 5: Example of the Terminal Output in QSPI Boot Mode

4.4 eMMC Boot

4.4.1 Generating the Image Files

Before preparing the eMMC binaries, the procedure described in Sections 3.3, 3.5 and 4.1 must be successfully completed with the `boot_mode` variable set to `emmc`. The steps described in Section 4.2.1 can be used to generate an eMMC image file.

4.4.2 Preparing the Hardware

1. Disconnect the power supply of the Mercury+ ST1 base board.
2. Disconnect all USB cables from the Mercury+ ST1 base board.
3. Enable the eMMC boot mode by setting the configuration DIP switch on the Mercury+ ST1 base board as follows (see label **CFG** in Figure 2):
   (a) Set CFG = [1: OFF, 2: ON, 3: ON, 4: ON]
4. Connect the Micro-USB cable between the computer and the Mercury+ ST1 base board using the Micro-USB port labeled **USBUB** in Figure 2.
5. Reconnect the UART terminal if necessary.
6. Connect the power supply to the Mercury+ ST1 base board (see label **12 V** in Figure 2).

4.4.3 Programming the eMMC

Direct programming of the eMMC is not supported by the Intel tools. To program the eMMC, first boot into U-Boot or Linux from another device (SD card or QSPI). A documentation how to program the eMMC with U-Boot or Linux can be found in the Enclustra Build Environment documentation [13].

4.4.4 Booting from the eMMC

1. Check that the hardware configuration is done according to Section 4.4.2.
2. Press the power-on reset button (see label **POR** in Figure 2) and release it after one second.

The expected output in the serial terminal console is the same as in Section 4.2.4.

5 Troubleshooting

5.1 Enclustra Build Environment Issues

- Ensure that the arguments specifying the product model and the binary paths are correct when running the `build.sh` script.
- Run `build.sh clean` or make a fresh clone of the whole repository.
- Verify that all the dependencies given within the EBE documentation [13] are met and all required packages are installed.
- Examine the `<base_dir>/sw/bsp-altera/build.log` file.

If none of the above helps resolving the issue, contact Enclustra Support [1] for assistance.

5.2 JTAG Issues

5.2.1 Detecting the JTAG Cable Fails

1. Make sure that the hardware configuration complies with Section 3.2.
2. Disconnect and reconnect the USB blaster connection and power cycle the Mercury+ ST1 base board.
3. If the problem perisists, reboot the computer.

5.3 UART Connection Issues

5.3.1 Recognizing USB UART Fails

- Check that the Micro-USB cable is connected properly.
- Try different `/dev/ttyUSBx` .
- Check that the FTDI VCP drivers are available.
- Unload the FTDI D2XX drivers and reconnect the USB cable.
- Try a different USB cable.

5.3.2 No Output in the Serial Console

1. Check that the baud rate for the UART in the design matches the baud rate set in the serial console.
2. Reconnect to the serial console.

5.4 Toolchain Issues

5.4.1 A Command is not Found

Run the following commands to verify that all the tools are installed correctly:

1. `which arm-none-eabi-gcc`
2. `which make`
3. `which git`
4. `which mkimage`
5. `which mcopy`

The commands should output the path to the respective binary. An example is given in Figure 6.

```
❯ which arm-none-eabi-gcc && which make && which git && which mkimage && which mcopy
/usr/bin/arm-none-eabi-gcc
/usr/bin/make
/usr/bin/git
/usr/bin/mkimage
/usr/bin/mcopy
```

Figure 6: Example of Binary Path Output

5.5 SD Boot Issues

- Make sure that the SD card is formatted according to Section 4.2.1.
- Make sure that the hardware configuration complies with Section 4.2.3.
- Use the SD card image provided on the SA1 Reference Design Github release page [5].
- Try another SD card.

5.6 QSPI Boot Issues

- Make sure that the QSPI binaries are generated according to Section 4.3.1.
- Make sure that the hardware configuration complies with Section 4.3.2.
- Use the provided binaries for QSPI boot mode on the SA1 Reference Design Github release page [5].
- Verify that the binaries are written to the correct offset address given in Section 4.3.3.
- Use the Enclustra Module Configuration Tool (MCT) [4] to program the QSPI binaries.

5.7 eMMC Boot Issues

- Make sure that the hardware configuration complies with Section 4.4.2.
- Use the eMMC image provided on the SA1 Reference Design Github release page [5].
- Make sure that the image is written correctly to the eMMC flash.

List of Figures

List of Tables

References

[1] Enclustra Support Channel
support@enclustra.com

[2] Mercury SA1 SoC Module User Manual
→ Ask Enclustra for details

[3] Mercury+ ST1 Base Board User Manual
→ Ask Enclustra for details

[4] Enclustra Module Configuration Tool (MCT)
http://www.enclustra.com/en/products/tools/module-configuration-tool/

[5] Mercury SA1 ST1 Reference Design Releases
https://github.com/enclustra/Mercury_SA1_ST1_Reference_Design/releases

[6] Enclustra Application Notes
https://github.com/enclustra/I2CAppNote
https://github.com/enclustra/GigabitEthernetAppNote

[7] OS Support for Intel® FPGA Complete Design Suite
https://www.intel.com/content/www/us/en/support/programmable/support-resources/design-software/os-support.html

[8] Intel® Quartus® Prime Pro and Standard Software User Guides
https://www.intel.com/content/www/us/en/support/programmable/support-resources/design-software/user-guides.html

[9] Intel SoC FPGA HWLIB repository
https://github.com/altera-opensource/intel-socfpga-hwlib

[10] Rocketboards Bootloader Guide
https://www.rocketboards.org/foswiki/Documentation/BuildingBootloaderCycloneVAndArria10

[11] ARM GNU Toolchain
https://developer.arm.com/Tools and Software/GNU Toolchain

[12] ARM GNU Toolchain download Page
https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads

[13] Enclustra Build Environment
https://github.com/enclustra-bsp/bsp-altera

[14] Intel SoC FPGA Readme
https://github.com/altera-opensource/u-boot-socfpga/blob/socfpga_v2023.07/doc/README.socfpga

[15] Intel® Cyclone V Device Datasheet
https://www.intel.com/content/www/us/en/docs/programmable/683801/current/cyclone-v-device-datasheet.html

[16] HPS SoC Boot Guide – Cyclone V SoC Development Kit
https://www.intel.com/content/www/us/en/docs/programmable/683265/current/an-709-hps-soc-boot-guide-cyclone-v.html