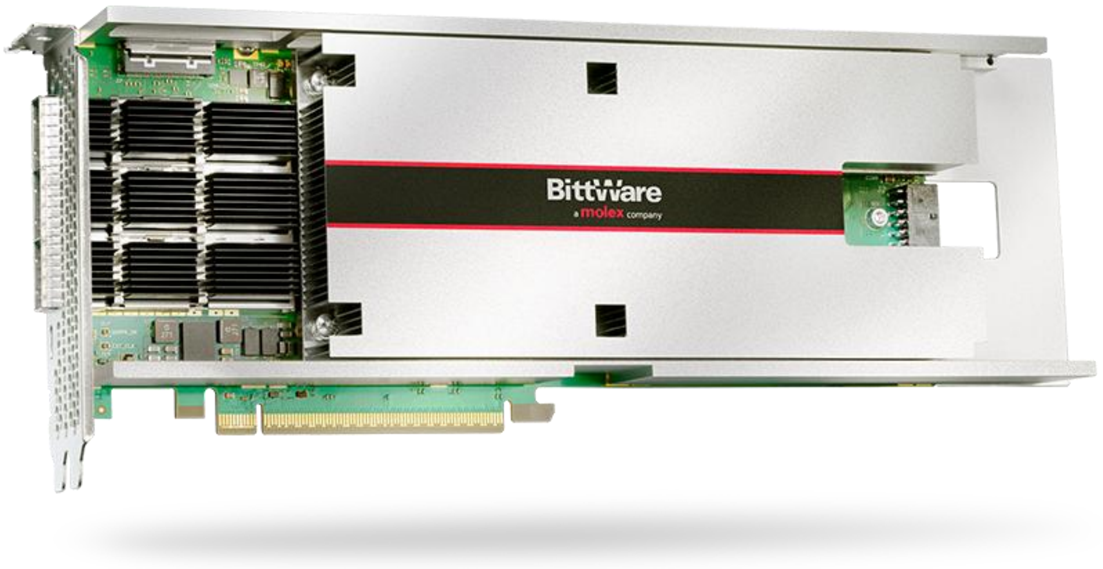


# IA-860m

## FPGA Developer Reference Guide

BittWare a molex company

# IA-860m FPGA Developer Reference Guide

Document Revision: 05-Dec-2025
Hardware Revisions: 0/1/2/3/4

**BittWare**
45 South Main Street, Suite L100
Concord, NH 03301 USA

+1 (603) 226 0404

## Intellectual Property

## Trademarks

## Disclaimer

The RF port connections on this product are for installation within the same building structure and indoor use only.

Please note that blowing fuses or making permanent changes to FPGA logic are considered as changes under the BittWare terms and conditions and can affect warranty. Please contact BittWare for further details.

# Contents

# 1 Introduction

## 1.1 About this Guide

This document provides FPGA related detail on integrating any BittWare provided IP into user designs. BittWare strongly advises FPGA developers to read the IA-860m Hardware Reference Guide (HRG) prior to this document. Since the IA-860m is an FPGA-centric card, the HRG provides relevant information to the FPGA developer that relates to the board's hardware.

## 1.2 Revision History

### 1.2.1 Document Revisions

The table below lists the revision history of this document.

*Table 1: Document revisions*

| Release Date | Document Revision | Hardware Revision | Notes/Changes |
|---|---|---|---|
| 05-Mar-2024 | 0.0 | 0 | Initial release for Early Access Unit (EAU) cards |
| 24-Apr-2024 | 1.0 | 0/1 | Includes Rev1 of PCB |
| 14-Jun-2024 | 2.0 | 0/1/2 | Includes Rev2 of PCB |
| 17-Sep-2024 | 3.0 | 0/1/2/3 | Includes Rev3 of PCB (extra Golden Top files)<br>BMC IP Pin table includes BBRev3 column (3 signals have moved location)<br>Section on PLL Locking added (Section 3.4)<br>In Section 3.1, ERROR_CHECK_FREQUENCY_DIVISOR set to 1 (was 256 previously). This resolves an issue where the SDM voltages weren't being reported correctly. |
| 16-Apr-2025 | 4.0 | 0/1/2/3/4 | Includes Rev4 of PCB (extra Golden Top files)<br>Section on Quartus Settings File (Section 3.1) includes note on ensuring Agilex DEVICE is selected in settings file (when not using Bittware files)<br>Section on PLL Locking (Section 3.4) now includes tables to show differences in PLL locations between ES silicon (Rev0/1/2/3) and production silicon (Rev4) |
| 12-Sep-2025 | 4.1 | 0/1/2/3/4 | Add note on UART speed |
| 20-Nov-2025 | 4.2 | 0/1/2/3/4 | Update HPS section |
| 05-Dec-2025 | 4.3 | 0/1/2/3/4 | Add Virtual Addresses for I$^2$C Access section |

## 1.3 Related Documentation

Refer to the following documents for more information about the IA-860m. All IA-860m **documentation** is available to download on the BittWare developer site.

| | |
|---|---|
| **BittWare** | IA-860m Hardware Reference Guide |
| **BittWare** | IA-860m Getting Started Guide |
| **BittWare** | BMC 3.0 Reference Guide |
| **BittWare** | BMC SPI Interface for IA-860m Datasheet |
| **Intel** | Intel Agilex Configuration User Guide |
| **Intel** | Intel Agilex FPGA External Memory Interface Overview |
| **Intel** | R-Tile Avalon Streaming Intel FPGA IP for PCI Express User Guide |
| **Intel** | F-tile Architecture and PMA and FEC Direct PHY IP User Guide |
| **Intel** | F-Tile Serial Lite IV Intel FPGA IP User Guide |
| **Intel** | M-Series FPGA Network-on-Chip (NoC) User Guide |
| **Intel** | High Bandwidth Memory (HBM2E) M-Series FPGA IP User Guide |
| **Intel** | AN1003: Multi Memory IP System Resource Planning for Intel Agilex® 7 M-Series FPGAs |

## 1.4 Abbreviations & Definitions

| | |
|---|---|
| **AVMM** | Avalon Memory Map |
| **AXI** | Advanced Extensible Interface |
| **BIST** | Built-In Self Test |
| **BRAM** | Block Random Access Memory |
| **DDR** | Double Data Rate |
| **DIMM** | Dual In-Line Memory Module |
| **DMA** | Direct Memory Access |
| **DRAM** | Dynamic Random Access Memory |
| **EAU** | Early Access Unit |
| **ECC** | Error Correction Code |
| **EMIF** | External Memory Interface |
| **FIFO** | First-In, First-Out |
| **FPGA** | Field Programmable Gate Array |
| **Gb/s** | Giga-bits per second |
| **GB/s** | Giga-Bytes per second |
| **GPIO** | General Purpose IO |
| **HBM** | High Bandwidth Memory |
| **HPS** | Hard Processor System |
| **I2C** | Inter IC |
| **MCIO** | Mini Cool Edge IO |
| **MTPS** | Mega Transfers Per Second |
| **NoC** | Network-on-Chip |
| **PCIe** | Peripheral Component Interconnect Express |
| **PMBus** | Power Management Bus |
| **PHY** | Physical Layer |
| **PMA** | Physical Medium Attachment |
| **PRBS** | Psuedo-Random Binary Sequence |
| **QSFP-DD** | Quad Small Form Factor (Double Density) |
| **QSPI** | Quad Serial Peripheral Interface |
| **SDM** | Secure Device Manager |
| **SMBus** | System Management Bus |
| **SPI** | Serial Peripheral Interface |

## 1.5 Contacting BittWare

BittWare is dedicated to providing customers with superior technical support:

- **BittWare Developer Site**: The BittWare Developer Site provides online access to our technical support resources. Go to developer.bittware.com to register for an account. Once you have set up an account, you will have access to BITTS (the BittWare Issue Tracking and Technical Support site), BittWare product documentation, software downloads, release notes, and examples. When you are logged into the Developer Site, follow the "Issue Tracking" link at the top right of the screen to access BITTS.
- **BittWare's website:** Our website at www.bittware.com provides a variety of literature, including whitepapers, datasheets, and articles.
- **Phone**: You can also call us directly at +1 (603) 226 0404 between the hours of 8:30 a.m.– 5:00 p.m. (US Eastern Time) or +44 (0) 1236 373500 between the hours of 8:30 a.m. – 5:00 p.m. (UK GMT).

## 1.6 Customer Feedback

Thank you for using BittWare products. We appreciate you choosing BittWare for your FPGA devel- opment.

If you have a few minutes to spare, we would love to hear from you about your experience with our products or our staff. We know your time is valuable; we would be grateful for any comment at all.

- Please email us at support@bittware.com for any type of feedback
- Or, let us know about a recent Technical Support inquiry at http://koch.link/bwsupportfeedback

# 2 Design Quick Start Guide

## 2.1 Introduction

BittWare provides a selection of 'golden_top' files to aid in creation of Quartus projects for the IA-860m. The files can be found on the BittWare Developer Site (within `golden_top.zip`).

---

There are no new files for Rev2 of the PCB as there have been no changes to the FPGA pinout or settings between Rev1 and Rev2 of the PCB.

Note that Rev4 is the board revision that exclusively supports production silicon devices.

As a general rule, if there are no files for the board revision you are planning to build for, use the fileset from the board revision that most closely precedes it.

---

Files include the following:

- golden_top_release_notes.txt – Release Notes describing any changes for release
- ia860m_golden_top.qpf – Quartus Project File
- ia860m_golden_top_bbrev0_es.qsf – Quartus Settings File (Rev0)
- ia860m_golden_top_bbrev0_es.tcl – Contains the SDM Settings + Pinout (Rev0)
- ia860m_golden_top_bbrev0.sdc – Contains the input frequencies for all input clocks (Rev0)
- ia860m_golden_top_bbrev0.vhd – A VHDL file with all ports on the entity declaration (Rev0)
- ia860m_golden_top_bbrev0.v – A Verilog file with all ports on the entity declaration (Rev0)
- ia860m_golden_top_bbrev1_es.qsf – Quartus Settings File (Rev1/2)
- ia860m_golden_top_bbrev1_es.tcl – Contains the SDM Settings + Pinout (Rev1/2)
- ia860m_golden_top_bbrev1.sdc – Contains the input frequencies for all input clocks (Rev1/2)
- ia860m_golden_top_bbrev1.vhd – A VHDL file with all ports on the entity declaration (Rev1/2)
- ia860m_golden_top_bbrev1.v – A Verilog file with all ports on the entity declaration (Rev1/2)
- ia860m_golden_top_bbrev3_es.qsf – Quartus Settings File (Rev3)
- ia860m_golden_top_bbrev3_es.tcl – Contains the SDM Settings + Pinout (Rev3)
- ia860m_golden_top_bbrev3.sdc – Contains the input frequencies for all input clocks (Rev3)
- ia860m_golden_top_bbrev3.vhd – A VHDL file with all ports on the entity declaration (Rev3)
- ia860m_golden_top_bbrev3.v – A Verilog file with all ports on the entity declaration (Rev3)
- ia860m_golden_top_bbrev4.qsf – Quartus Settings File (Rev4)
- ia860m_golden_top_bbrev4.tcl – Contains the SDM Settings + Pinout (Rev4)
- ia860m_golden_top_bbrev4.sdc – Contains the input frequencies for all input clocks (Rev4)
- ia860m_golden_top_bbrev4.vhd – A VHDL file with all ports on the entity declaration (Rev4)
- ia860m_golden_top_bbrev4.v – A Verilog file with all ports on the entity declaration (Rev4)
- hps_dram_ddr2666_dm_enabled_preset – Preset file for HPS DDR4 SDRAM EMIF (Quartus 25.1)

## 2.2 Design Project Creation

Some builds of the IA-860m will not support all the I/O provided in the golden_top files. **Before starting your design**, be familiar with the features available on the card you are designing for.

When creating a new design, perform the following stages:

1. Rename all the files to an appropriate name.
2. Edit the .qpf file to ensure that the revision(s) match the name(s) of the .qsf file(s).
3. Edit the .qsf file(s) to ensure they link to the correct names of the .tcl file and .sdc file.
4. Comment in and rename the top-level HDL file.
5. Edit the .sdc file to remove any clocks that aren't required in the design.
6. Edit the .tcl file to comment/remove any signals not required in the design.
7. Create your design using the .vhd or .v file as a template for the top level, removing signals that aren't required by your design.

## 2.3 Beyond Project Creation

For further design considerations please refer to the following sections:

- For transceiver preservation, see Transceiver Preservation.
- For BMC 3.0 SPI IP integration, see BMC3.0 FPGA IP.
  - Includes pin instructions for when the BMC3.0 FPGA IP isn't integrated into the design.
- For HBM2E (and NoC) integration, see HBM2E and NoC.
- For configuration and RBF generation, see FPGA Configuration.

If you are using the HPS DRAM in a design, when creating the EMIF IP, enable the use of a custom preset and link to the preset contained with golden_top files.

Note that the HPS preset is for DRAM IP created in Quartus 25.1. The IP changed from Quartus 24.3 onwards. If building with older tools, please use the preset from an older golden_top release.

# 3 Physical Constraints

## 3.1 Quartus Settings File

The Card Test design includes the `ia860m_device_bbrev<x>.tcl` file, which contains the device, SmartVID, pin, and I/O standards for the IA-860m. This file can be included in a project using the following line in the QSF:

```
source ia860m_device_bbrev<x>.tcl
```

The `ia860m_device_bbrev<x>.tcl` assumes the F-tile transceivers connected to the QSFP-DD are included in the design. Refer to the section below about transceiver preservation if this is not the case.

To support device configuration and SmartVID, include the following in the Quartus Settings File (QSF) if you are not using the `ia860m_device_bbrev<x>.tcl` file:

```
set_global_assignment -name FAMILY "Agilex 7"
# IA-860m devices: AGMF039R47A1E2VR0, AGMF039R47A1E1VR0 BBRev0/1/2/3
# IA-860m devices: AGMF039R47A1E2VC BBRev4 (Production Silicon)
set_global_assignment -name DEVICE AGMF039R47A1E2VR0
set_global_assignment -name BOARD default
set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
set_global_assignment -name MAX_CORE_JUNCTION_TEMP 100
set_global_assignment -name ERROR_CHECK_FREQUENCY_DIVISOR 1
set_global_assignment -name ENABLE_ED_CRC_CHECK ON
set_global_assignment -name MINIMUM_SEU_INTERVAL 0
set_global_assignment -name DEVICE_INITIALIZATION_CLOCK OSC_CLK_1_125MHZ
set_global_assignment -name AUTO_RESTART_CONFIGURATION OFF
set_global_assignment -name STRATIXV_CONFIGURATION_SCHEME "AVST X8"
set_global_assignment -name USE_PWRMGT_SCL SDM_IO0
set_global_assignment -name USE_PWRMGT_SDA SDM_IO12
set_global_assignment -name USE_CONF_DONE SDM_IO16
set_global_assignment -name USE_INIT_DONE SDM_IO5
set_global_assignment -name USE_PWRMGT_ALERT SDM_IO9
set_global_assignment -name USE_HPS_COLD_RESET SDM_IO7
set_global_assignment -name VID_OPERATION_MODE "PMBUS SLAVE"
set_global_assignment -name PWRMGT_DEVICE_ADDRESS_IN_PMBUS_SLAVE_MODE 01
set_global_assignment -name GENERATE_PR_RBF_FILE ON
# These two settings below keep getting added by Quartus(22.3)
set_global_assignment -name PWRMGT_VOLTAGE_OUTPUT_FORMAT "LINEAR FORMAT"
set_global_assignment -name PWRMGT_LINEAR_FORMAT_N "-12"
# Generate a compressed sof
set_global_assignment -name GENERATE_COMPRESSED_SOF ON
```

When not using our .tcl files, please ensure you enable the correct Agilex DEVICE in your design:

- **For Rev0/1/2/3** : AGMF039R47A1E2VR0 (or AGMF039R47A2E2VR0)
- **For Rev4** : AGMF039R47A1E2VC

## 3.2 Transceiver Preservation

The performance of Agilex high-speed transceivers deteriorates if they are powered up and unused. If there are unused transceivers in the design that might be required in the future, BittWare recommends that the unused transceivers be preserved.

For details, refer to the Intel *F-tile Architecture and PMA and FEC Direct PHY IP User Guide*, which has a section on preserving unused PMA lanes. If you do not follow these recommendations, it can result in a configuration error.

To preserve all unused PMA lanes across all F-Tiles and R-Tiles, use the following in the .qsf:

```
set_global_assignment -name PRESERVE_UNUSED_XCVR_CHANNEL ON
```

To preserve any unused PMA lanes in a single F-tile in a package, use the following single-pin F-tile .qsf:

```
# F-tile (Bank 12A) RX_Q2_CH7P Pin:
set_instance_assignment -name PRESERVE_UNUSED_XCVR_CHANNEL
ON -to DN66
# Fitter report snippet which confirms preservation:
# Info(22251): Empty 'F-tile' indicated by pin 'DN66' has been preserved due to
PRESERVE_ UNUSED_XCVR_CHANNEL instance assignment
```

If your design does not instantiate (does not use) a PMA lane, preservation of the unused PMA lane in a partially used F-tile takes place by default.

**Note:** If a design does not preserve transceivers and does not instantiate any PMA lane within an F-tile then it can result in a configuration error.

## 3.3 VSR Mode

Intel recommend enabling VSR mode on the RX path of all transceivers running at PAM4 rates or NRZ rates greater than 23Gbps.

We recommend that VSR mode be set for high loss.

An example constraint would be:

```
set_instance_assignment -name HSSI_PARAMETER "vsr_mode=VSR_MODE_HIGH_LOSS" -to
QSFP0_RX_P[0]
```

## 3.4 PLL Locations

Bittware has found occasions where it has place the IOPLL for User Clock 0 in a neighboring I/O Bank (Bank 2B rather than Bank 2A). This has led to problematic behavior (PLL going in and out of lock).

As such, we recommend that the PLLs for User Clock 0 and User Clock 1 be locked down. The proposed locations can be found in the ia860m_golden_top_bbrev<x>_es.tcl file (in the golden top reference files).

For User Clock 0, the proposed I/O Bank 2A locations are:

| Rev0/1/2/3 | Rev4 |
|---|---|
| IOPLL_X64_Y2_N293 | IOPLL_X65_Y2_N0 |
| IOPLL_X64_Y2_N384 | IOPLL_X65_Y2_N84 |
| IOPLL_X64_Y2_N377 | IOPLL_X65_Y2_N91 |

For User Clock 1, the proposed I/O Bank 3A locations are:

| Rev0/1/2/3 | Rev4 |
|---|---|
| IOPLL_X12_Y418_N293 | IOPLL_X13_Y418_N0 |
| IOPLL_X12_Y418_N384 | IOPLL_X13_Y418_N84 |
| IOPLL_X12_Y418_N377 | IOPLL_X13_Y418_N91 |

# 4 IP Integration

For the IA-860m, there are currently two IP components that may require further explanation:

- Bittware's BMC3.0 Interface IP
- Intel's HBM2E Controller IP (with NoC)

## 4.1 BMC3.0 FPGA IP

**Note**: When the BMC3.0 FPGA IP is not integrated into the design, the design must drive BMC_IF_PRESENT_L (PIN_FH55) logic HIGH to ensure proper BMC operation.

### 4.1.1 Overview

The BMC interface IP communicates with the BMC via two independent SPI interfaces. All host traffic is communicated via the ingress SPI interface. This interface is mastered from the BMC itself.

The egress SPI interface is mastered from the FPGA and is used to push/pull telemetry data to and from the BMC. Telemetry data currently consists of QSFP-DD sideband signals and EEPROM. This could be expanded to include other data (i.e. sensor data) in a future release.

Host access to the ingress SPI interface is provided via two AXI4-Lite interfaces. These interfaces are identical and share access to the ingress SPI controller. Priority is shared (round-robin) between the two interfaces.

### 4.1.2 BMC Host Interface 0 and 1

Both host interfaces (host0 and host1) are identical and connect to the same BMC slave IP. Priorirty is shared between the two interfaces (round-robin). In BittWare's utilities, host0 is used for MCTP communications and host1 is used for I$^2$C communications.

### 4.1.3 BMC Capability ROM

The capability ROM provides both the host and the BMC with information that details which features are supported (by the FPGA IP). The ROM is a dual port 64K deep memory.

The BMC accesses the ROM via the SPI Slave (Ingress) interface. The host accesses the ROM via the Capability ROM AXI4-Lite interface.

### 4.1.4 BMC Telemetry Interface

The BMC is responsible for direct management of board level features as a hardware RoT (Root of Trust). The telemetry mechanism allows the BMC to manage access yet still expose peripheral control and status signals, where relevant, to the FPGA user design without requiring these to be directly connected to FPGA I/O pins. This approach retains a level of control for security on the platform and scales well for other BittWare hardware by avoiding additional FPGA I/O pin requirements and improves ease of porting between different BittWare hardware. It avoids unnecessary I$^2$C control complexity for FPGA side access to these sideband, low speed, control, and status signals.

The telemetry interface allows the BMC and FPGA to pass information to each other in relative real-time. The primary use for this is to provide sideband control and status for the QSFP-DD module(s). The interface allows the FPGA logic to assert QSFP-DD Reset# and QSFP-DD LPMode despite not having direct access to those signals from the FPGA pins. Equally, the FPGA can be informed when a module is present or when there is an interrupt from the QSFP-DD module(s).

In the IA-860m implementation, the full EEPROM contents (256 bytes) are also provided.

### 4.1.5 Interface Pins

The pins as described in the 'BMC SPI Interface for IA-860m Datasheet' should be connected as follows:

| BMC Interface Name | I/O Name | Pin Location (BBRev0/1/2) | Pin Location (BBRev3) |
|---|---|---|---|
| bmc_if_ready_n | BMC_IF_PRESENT_N | PIN_FH55 | PIN_FM59 |
| spi_slv_sclk | FPGA_IG_SPI_SCK | PIN_FJ62 | PIN_FJ62 |
| spi_slv_ss_n | FPGA_IG_SPI_PCS0 | PIN_FE60 | PIN_FE60 |
| spi_slv_mosi | FPGA_IG_SPI_MOSI | PIN_FF59 | PIN_FF59 |
| spi_slv_miso | FPGA_IG_SPI_MISO | PIN_FH61 | PIN_FH61 |
| f2b_irq_n | FPGA_TO_BMC_IRQ | PIN_FJ56 | PIN_FM57 |
| spi_mst_sclk | FPGA_EG_SPI_SCK | PIN_FR58 | PIN_FR58 |
| spi_mst_ss_n | FPGA_EG_SPI_PCS0 | PIN_FF61 | PIN_FF61 |
| spi_mst_mosi | FPGA_EG_SPI_MOSI | PIN_FE62 | PIN_FE62 |
| spi_mst_miso | FPGA_EG_SPI_MISO | PIN_FP57 | PIN_FP57 |
| b2f_irq_n | FPGA_TO_BMC_IRQ | PIN_FE56 | PIN_FF57 |

### 4.1.6 Virtual Addresses for I$^2$C Access

This section documents the I$^2$C virtual addresses (referenced in the *BMC SPI Interface for IA-xxx Cards Datasheet*) that should be used with this IP for accessing the following devices:

*Table 2: I$^2$C Device Virtual Addresses*

| I$^2$C Resource Name | I$^2$C Virtual Address |
|---|---|
| SiT95148 0 | 0x70 |
| SiT95148 1 | 0x71 |
| ID PROM | 0x57 |
| QSFP-DD 0 | 0x50 |
| QSFP-DD 1 | 0x52 |
| QSFP-DD 2 | 0x54 |

### 4.1.7 Further Reading

For more information regarding the FPGA BMC Interface IP and how to implement it within a design, please see 'BMC SPI Interface for IA-860m Datasheet'. The 'Designing With The IP' section towards the end of the datasheet has details on how to integrate the IP into either Platform Designer or HDL.

## 4.2 HBM2E and NoC

### 4.2.1 Overview

HBM2E is the next generation of High Bandwidth Memory (HBM) as defined in JEDEC specification JESD-235C. The Agilex7 M-Series devices are compliant with that JEDEC specification.

The FPGA device on the IA-860m contains two HBM2E memory DRAMs in a single package thus supporting two HBM2E interfaces.

Each HBM2E DRAM consists of stacked DRAMs. As such, each HBM2E memory provides 8-channels, each targeting a different part of the stack. Each channel is 128 bits, but they are typically broken down further into two pseudo-channels, each targeting 64 bits of the memory width.

The HBM2E pseudo-channel ports are not directly accessible to the fabric. Instead, they are connected to the fabric via a hard memory NoC (network-on-chip). The fabric connects into the NoC via NoC initiator ports into the network.

### 4.2.2 HBM2E Locations

As already mentioned, the FPGA device on the IA-860m contains two HBM2E memory DRAMs. Each HBM2E instance has a corresponding NoC. An HBM/NoC pair is situated at the top of the device and another is situated at the bottom of the device.

The HBM2E IP has two inputs that must be connected to the top level of the design: *hbm_cattrip_i* and *hbm_temp_i*. There are no pins assigned to these ports and no pins should be assigned to them, but they must be connected to the top level of the design.

For the bottom HBM2E and NoC, the following pins are required:

| Pin Name | IP Name | Description |
|---|---|---|
| HBM_REFCLK0 | PIN_EC36 | Lower HBM2E Reference Clock |
| HBM_REFCLK0(n) | PIN_ED35 | Lower HBM2E Reference Clock (negative) |
| NOC_CLK0 | PIN_EE56 | Lower NoC Clock |

For the upper HBM2E and NoC, the following pins are required:

| Pin Name | Pin Location | Description |
|---|---|---|
| HBM_REFCLK1 | PIN_AR36 | Upper HBM2E Reference Clock |

| | | |
|---|---|---|
| HBM_REFCLK1(n) | PIN_AN36 | Upper HBM2E Reference Clock (negative) |
| NOC_CLK1 | PIN_AU52 | Upper NoC Clock |

### 4.2.3 NoC Connectivity

Design constraints are required to control which fabric NoC initiators are connected to which pseudo channels on the HBM2E. This is handled via the ‘Network on Chip (NoC) Assignment Editor’ from the Assignments menu in Quartus. The process for this is well described in the Intel documentation (see

Related Documentation).

On top of these constraints, BittWare has found that for optimal HBM bandwidth, the NoC initiators and NoC targets should be LOC'd to specific locations in the constraints.

For the following examples, we have removed the Card Test design instance and replaced with a general description in <descriptor>.

For the **lower HBM2E/NoC** we have:

```
set_location_assignment NOCINITIATOR_X149_Y6_N202 -to
<ch0_0_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X215_Y6_N202 -to
<ch0_1_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X188_Y6_N202 -to
<ch1_0_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X258_Y6_N202 -to
<ch1_1_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X134_Y6_N202 -to
<ch2_0_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X204_Y6_N202 -to
<ch2_1_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X160_Y6_N202 -to
<ch3_0_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X242_Y6_N202 -to
<ch3_1_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X296_Y6_N202 -to
<ch4_0_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X365_Y6_N202 -to
<ch4_1_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X323_Y6_N202 -to
<ch5_0_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X404_Y6_N202 -to
<ch5_1_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X269_Y6_N202 -to
<ch6_0_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X350_Y6_N202 -to
<ch6_1_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X312_Y6_N202 -to
<ch7_0_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X376_Y6_N202 -to
<ch7_1_initiator>||intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCTARGET_X204_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch0_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X231_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch0_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X215_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch1_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X242_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch1_u1|target_0.target_inst_0
```

```
set_location_assignment NOCTARGET_X194_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch2_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X221_Y6_N200 -to <
<hbm_inst>|hbm_fp_0|tniu_ch2_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X210_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch3_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X237_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch3_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X269_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch4_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X296_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch4_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X285_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch5_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X312_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch5_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X264_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch6_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X291_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch6_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X275_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch7_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X302_Y6_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch7_u1|target_0.target_inst_0
```

For the **upper HBM2E/NoC** we have:

```
set_location_assignment NOCINITIATOR_X322_Y417_N202 -to
<ch0_0_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X258_Y417_N202 -to
<ch0_1_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X296_Y417_N202 -to
<ch1_0_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X215_Y417_N202 -to
<ch1_1_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X357_Y417_N204 -to
<ch2_0_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X269_Y417_N202 -to
<ch2_1_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X311_Y417_N202 -to
<ch3_0_initiator>intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X242_Y417_N202 -to
<ch3_1_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X188_Y417_N202 -to
<ch4_0_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X105_Y417_N202 -to
<ch4_1_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0
```

```
set_location_assignment NOCINITIATOR_X150_Y417_N202 -to
<ch5_0_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X79_Y417_N202 -to
<ch5_1_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X204_Y417_N202 -to
<ch6_0_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X134_Y417_N202 -to
<ch6_1_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X161_Y417_N202 -to
<ch7_0_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCINITIATOR_X94_Y417_N202 -to
<ch7_1_initiator>|intel_noc_initiator_0|iniu_0|initiator_inst_0

set_location_assignment NOCTARGET_X248_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch0_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X221_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch0_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X237_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch1_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X210_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch1_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X258_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch2_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X231_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch2_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X242_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch3_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X215_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch3_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X183_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch4_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X156_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch4_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X167_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch5_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X140_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch5_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X188_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch6_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X161_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch6_u1|target_0.target_inst_0

set_location_assignment NOCTARGET_X177_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch7_u0|target_0.target_inst_0

set_location_assignment NOCTARGET_X150_Y417_N200 -to
<hbm_inst>|hbm_fp_0|tniu_ch7_u1|target_0.target_inst_0
```

These constraints are for 1-to-1 initiator/target mapping rather than 16x16 crossbar mapping. We also have a 1-port initiator per pseudo-channel.

For initiator placement, guidance has been taken from the Intel HBM2E documentation (Intel HBM2E User Guide - System Performance), see 'Initiator placements' on the linked page.

### 4.2.4 Further Reading

For more information on HBM2E and NoC, please refer to the linked Intel documentation (see

Related Documentation).

# 5 FPGA Configuration

## 5.1 Power-On Configuration

The FPGA configuration is set to Avalon-ST x8 (MSEL[2:0] pins set to "110". The BMC holds off configuration until all the power supplies and clocks are good; it then drives configuration and controls the transfer of configuration data from flash memory to the FPGA.

## 5.2 Configuring the User FPGA

The IA-860m BMC automatically loads an FPGA design from Flash at power on by default. The IA-860m supports the following methods to program the FPGA:

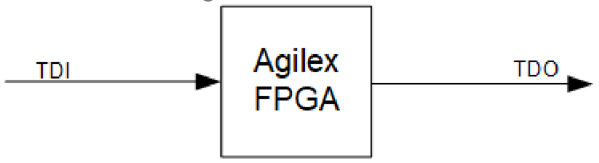
- **JTAG Chain using external Intel/Altera Blaster II Cable** – Connecting the Intel/Altera Blaster II Cable to J7 (see the Hardware Reference Guide for details) allows access to the JTAG chain with Quartus Programmer enabling configuration with a .sof file.
- **User Flash** – The BMC can program the on-board 256MByte configuration flash with multiple .rbf files. One of these can be set to configure the FPGA at power on by setting its priority to ‘0’. Each file can be given a boot priority, with ‘0’ being top priority..
  Once an. rbf has been loaded into flash and its priority is set to 0 then rebooting the BMC will configure the FPGA with this design. Refer to the Getting Started Guide and the SDK documentation for details on bw_bmc_fpga_load and bw_bmc_update utilities for further details.

### 5.2.1 Programming the FPGA using the Intel Quartus Programmer GUI

Once the Quartus Prime Pro tools have compiled the design, the output is a programming file (SOF). To download the file directly into the FPGA:

1. Connect the Intel/Altera Blaster II Cable to the J7 connector on the IA-860m via the J7 cable and breakout board.
2. Open the Quartus Prime Pro Programmer.
3. In the Programmer window select: **Hardware Setup** > **USB-Blaster** (this name may be different depending upon the Altera/Intel Blaster connected).
4. Click on the **Auto Detect** button. If a **Select Device** window pops up, select the **AGMF039R47A** device, and click on **OK**. The Programmer window should now display the devices and the JTAG chain as shown in **JTAG Chain**.
5. Click on the Agilex FPGA to highlight it and then click on the **Change File** button. Navigate to the SOF that is to be programmed into the FPGA and open this file.
6. In the upper section of the Programmer window, tick the **Program/Configure** box associated with the Agilex FPGA. Then click on the **Start** button.
7. Observe the **Progress** bar at the top right of the Programmer window. When it shows “**100% Successful**”, the FPGA is programmed.

*Figure 1: JTAG Chain*



### 5.2.2 Generating a Raw Binary File (.rbf)

Once the Quartus Prime Pro tools have compiled the design, the output is a programming file (SOF). To program the design into the onboard configuration flash the .sof needs to be converted into a .rbf. There are multiple ways to generate the .rbf. The Card Test includes Quartus settings to enable the assembler to generate the .rbf automatically after the .sof generation. The Quartus Programming File Generator utility can also convert a .sof into a .rbf.

#### Using Quartus Programming File Generator Utility to Generate a .rbf

1. Open Quartus. In the file menu select “**Programming File Generator…**”
2. Set the Device family to “**Agilex 7**” and the Configuration mode to “**AVST x8**”
3. Set the filename and output directory.
4. Select “**Raw Binary File (.rbf)**”.
5. Switch from the Output Files tab to the Input Files tab.
6. Use the “**Add Bitstream…**” button to add the bitstream (.sof).
7. Click on “**Generate**” to generate the .rbf file.

#### Enabling .rbf Generation Through Project Settings

1. In the Quartus settings file add the following settings:

```
# Assembler settings to generate an .rbf
set_global_assignment -name GENERATE_PROGRAMMING_FILES ON
set_global_assignment -name CONVERT_PROGRAMMING_FILES_COMMANDS <path to file containing
commands>
```

2. In the file containing the programming file commands, add the following where filename.sof is the path and name of the input .sof file and similarly filename.rbf is the path and filename for the output .rbf file.

```
-c <filename.sof> <filename.rbf>
```

Examples of this can be found in each revision of the cardtest in a revisions qsf file.

## 5.3 Managing Quartus Revisions

The Agilex FPGA contains a Secure Device Manager (SDM) that handles device programming. The microcode for the SDM is contained within the programming file that is programmed into the IA-860m flash for power on device configuration. Or, if a SOF is used to program the FPGA, the SDM microcode is pre-pended by Quartus Programmer. If the version of SDM code pre-pended by Quartus Programmer is ‘newer’ than the current SDM version, the SDM micro-code is updated. The Quartus version used to generate the RBF or program the SOF therefore determines the SDM microcode version. If the FPGA is programmed from flash at power-on, all subsequent FPGA reconfigurations should use at least the same Quartus version that was used to generate the RBF file. Also use the same version of Quartus Programmer for all device programming within a power cycle. Refer to the Intel Agilex Configuration User Guide for further details on the SDM.

Best practice is to generate all images stored in the flash using the same Quartus tools revision.

## 5.4 Autonomous Hard IP Feature

FPGA configuration from flash takes around a second. The PCIe hard IP core in the Agilex is configured first from the programming file and operates in autonomous mode. The bootup time for the PCIe interface may still lie outside the PCIe protocol specifications, but it should be fast enough for most servers.

# 6 Hard Processor System

## 6.1 Overview

The IA-860m includes an ARM Hard Processor System (HPS). A bootloader image can be stored in flash and loaded upon power up.

This example was tested using Ubuntu 22.04, and Quartus Pro 25.3.0 build109 was used to compile the fpga image (SOF) used to program the FPGA and HPS.

The BittWare IA-860m non-volatile storage is a NAND flash and is not accessible via JTAG.

## 6.2 Hardware Setup

The following connections are required:

- Intel USB-Blaster for JTAG connection to the HPS (either the embedded USB Blaster 3 or an external pod)
- USB cable for HPS UART console connectivity

Refer to the IA-860m Hardware Reference Guide for connection options.

## 6.3 Software Requirements

This example was compiled with Ubuntu 22.04.

### 6.3.1 Software Dependencies

#### Host Packages

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install openssh-server mc libgmp3-dev libmpc-dev gawk wget git diffstat unzip
texinfo gcc build-essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils
debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint3 xterm
python3-subunit mesa-common-dev zstd liblz4-tool git fakeroot ncurses-dev xz-utils libssl-dev bc
flex libelf-dev bison xinetd tftpd tftp nfs-kernel-server libncurses5 libc6-i386 libstdc++6:i386
libgcc++1:i386 lib32z1 device-tree-compiler curl mtd-utils u-boot-tools net-tools swig -y
```

#### BittWare HPS Release Package

Obtain the bittware-ia860m-hps package from the BittWare Developer website.

### Arm Developer Studio

This example uses ARM Developer Studio – Intel Edition, version 2023.1. A license is required to use ARM-DS on Intel SoC platforms. Contact Intel to obtain a license file. For more information, visit:

https://www.intel.com/content/www/us/en/software-kit/816394/arm-development-studio-version-2023-1-for-intel-soc-fpga.html

### 6.3.2 Set up the Working Directory

```
sudo rm -rf bittware_ia860m_hps
tar xvfz bittware-ia860m-hps-1.6.tgz
cd bittware_ia860m_hps
export TOP_FOLDER=`pwd`
```

### 6.3.3 Install the Cross Compiler Tool Chain

GCC-ARM is required to compile the HPS binaries.

```
cd $TOP_FOLDER
wget https://developer.arm.com/-/media/Files/downloads/gnu/11.2-2022.02/binrel/gcc-arm-11.2-
2022.02-x86_64-aarch64-none-linux-gnu.tar.xz
tar xf gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz
rm -f gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz
export PATH=`pwd`/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin:$PATH
export ARCH=arm64
export CROSS_COMPILE=aarch64-none-linux-gnu-
```

Quartus is a requirement to compile the flash and FPGA image. This example was compiled using Quartus Pro 23.4.0-79; however other versions of Quartus may work.

The Quartus binaries must be on the path and available from the terminal. The udev rules [i]must also be set properly to access the USB-Blaster JTAG pod.

## 6.4 Boot Flow

### 6.4.1 Boot Flow Overview

The flow begins with an FPGA project that contains an instantiation of the HPS component in the design. The output (in SOF form) must then be modified to embed a first stage bootloader (FSBL) into it. The FSBL in our case is the u-boot-spl bootloader.

**Note**: FSBL and SPL [secondary program loader] are terms that can be used interchangeably. This document will refer to it as the SPL.

When configured, the FPGA holds the HPS in reset while the SDM copies the SPL into the HPS on-chip SDRAM and then releases the HPS from reset. The SPL then initializes any relevant peripherals necessary to obtain the full U-Boot.

**Note**: the SPL is a minimalist build of U-Boot having only the necessary functionality to load and transfer execution to the full U-Boot version, hereafter referred to as “U-Boot proper”.

### 6.4.2 The Empty Flash Problem

Since the NAND flash for the HPS is not on the JTAG chain, there are some challenges in programming the flash initially, or when it is empty. The SPL does not provide a console, and it cannot send/receive files over the UART. The are two solutions to this problem: using a debugger such as Arm Developer Studio with a USB-Blaster, or implementing Intel’s “HPS Copy Engine” strategy. We recommend obtaining a license to Arm Developer Studio (Intel Edition) as the debugger is a valuable add-on when developing any future HPS solutions.

### 6.4.3 NAND Flash Layout

When booting, the SPL will initialize peripherals and load U-Boot Proper from MTD partition 0 and transfer control to it. You can use U-Boot to perform basic operations or load an application.

| MTD Partition | UBI Volume | Volume Name | Type | Image/File | | Flash Offset | Size | Size in Hex |
|---|---|---|---|---|---|---|---|---|
| 0 (u-boot) | N/A | N/A | RAW | u-boot.itb | | 0x00000000 | 2MB | 0x00200000 |
| 1 (root) | 0 | env | UBI | | root.ubi | 0x00200000 onwards | 256KB | 0x40000 |
| | 1 | script | UBI | u-boot.scr | | | 128KB | 0x00020000 |
| | 2 | kernel | UBI | kernel.itb | | | 64MB | 0x04000000 |
| | 3 | dtb | UBI | linux.dtb | | | 256KB | 0x00040000 |
| | 4 | rootfs | UBIFS | agilex_rootfs.img | | | <400MB | <0x19000000 |

**Note**: BittWare IA-860m products are shipped with only the U-Boot (mtd partition 0) and the U-Boot environment space (mtd partition 1, ubi volume 0 “env”) populated. However, this document contains the necessary steps to build a full Linux installation as well.

## 6.5 Building the Binaries

### 6.5.1 Build Arm Trusted Firmware

```
cd $TOP_FOLDER

rm -rf arm-trusted-firmware
```

```
git clone https://github.com/altera-opensource/arm-trusted-firmware

cd arm-trusted-firmware

# comment out next line to use the latest ATF branch

git checkout -b test -t origin/socfpga_v2.8.0

make bl31 PLAT=agilex DEPRECATED=1

cd ..
```

### 6.5.2 Build Linux

U-Boot will generate a “FIT” image containing the reference to the kernel and its device tree binary. So, generate those first.

```
cd $TOP_FOLDER

git clone -b socfpga-5.15.90-lts https://github.com/altera-opensource/linux-socfpga

sudo chown -R "$USER:$USER" linux-socfpga

cd linux-socfpga

cp $TOP_FOLDER/patches/860/kernel/socfpga_agilex_bittware.dts arch/arm64/boot/dts/intel/

make clean && make mrproper

git apply $TOP_FOLDER/patches/860/kernel/kernel.patch

make defconfig

make -j $(nproc) Image dtbs

make -j $(nproc) Image.lzma dtbs
```

### 6.5.3 Build U-Boot

The steps described in this section will build both the SPL and U-Boot proper.

**Prepare the sources:**

```
cd $TOP_FOLDER

rm -rf u-boot-socfpga

git clone -b socfpga_v2025.04 https://github.com/altera-opensource/u-boot-socfpga

cd u-boot-socfpga

tar xvfz $TOP_FOLDER/patches/860/u-boot/u-boot-fit.tgz

cd fit

ln -sf "$TOP_FOLDER/linux-socfpga/arch/arm64/boot/Image" Image

ln -sf "$TOP_FOLDER/linux-socfpga/arch/arm64/boot/dts/intel/socfpga_agilex_bittware.dtb"
linux.dtb

# link to atf

cd ..
```

```
ln -s $TOP_FOLDER/arm-trusted-firmware/build/agilex/release/bl31.bin .

make clean && make mrproper

cp -f $TOP_FOLDER/patches/860/u-boot/{fi2c.*,hdm.*} cmd/

make socfpga_agilex7m_defconfig

git apply $TOP_FOLDER/patches/860/u-boot/u-boot.patch

make -j `nproc` dtbs all

cd fit

make
```

Among the output files in $TOP_FOLDER/u-boot-socfpga are

| File | Description |
|---|---|
| spl/u-boot-spl-dtb.hex | The SPL binary with an attached device tree binary blob, in hex format. This will be embedded in the original SOF produced by Quartus, producing the final SOF image. |
| u-boot.itb | The full u-boot proper image, with DTB and ATF embedded in a FIT image. This is the file that should be programmed into flash, using one of the methods discussed previously. The ARM DS JTAG method is described below.<br><br>This file is also loaded and executed by the ARM-DS debugger script as part of the target initialization. |
| u-boot-spl-dtb.bin | Binary form of the SPL with the DTB appended. The ARM-DS debugger script will load and execute this as part of the target initialization. |
| fit/kernel.itb | Kernel FIT image, with linux kernel and flattened device tree. |

### 6.5.4 Generate the Root Filesystem Image

The root filesystem is built using the Poky Linux sources. Perform the following commands to build the root fs image:

```
cd $TOP_FOLDER

mkdir yocto && cd yocto

git clone -b nanbield git://git.yoctoproject.org/poky.git

git clone -b nanbield git://git.yoctoproject.org/meta-intel-fpga.git

git clone -b nanbield https://github.com/openembedded/meta-openembedded.git

tar xvfz ${TOP_FOLDER}/patches/860/rootfs/meta-bittware.tgz

source poky/oe-init-build-env ./build

echo 'BBLAYERS += " ${TOPDIR}/../meta-intel-fpga "' >> conf/bblayers.conf

echo 'BBLAYERS += " ${TOPDIR}/../meta-openembedded/meta-oe "' >> conf/bblayers.conf

echo 'BBLAYERS += " ${TOPDIR}/../meta-bittware "' >> conf/bblayers.conf
```

```
{
    echo 'MACHINE = "agilex7_dk_si_agf014ea"'
    echo 'IMAGE_FSTYPES = "tar.gz ubi ubifs"'
    echo 'CORE_IMAGE_EXTRA_INSTALL += " fio h2f"'
    echo 'UBI_VOLNAME = "rootfs"'
    echo 'MKUBIFS_ARGS = "-F -e 126976 -m 2048 -c 3400"'
    echo 'UBINIZE_ARGS = "-p 128KiB -m 2048"'
} >> conf/local.conf
bitbake core-image-minimal
```

This will create the ubifs image (“agilex_rootfs.img” in the NAND Flash Layout section above) needed in the next section. The generated file is build/tmp/deploy/images/agilex/core-image-minimal-agilex.ubifs, though a symbolic link is created to this file in the following section.

### 6.5.5 Generate UBI Volume Image

At this point, all the components of the root.ubi (see NAND Flash Layout section above) file should exist. To generate the root.ubi file to be written to NAND MTD partition 1, perform the following:

```
cd $TOP_FOLDER/ubi
ln -sf $TOP_FOLDER/yocto/build/tmp/deploy/images/agilex7_dk_si_agf014ea/core-image-minimal-
agilex7_dk_si_agf014ea.rootfs.ubifs agilex_rootfs.img
ln -sf $TOP_FOLDER/u-boot-socfpga/fit/kernel.itb kernel.itb
ubinize -o root.ubi -p 128KiB -m 2048 ubinize.cfg
```

### 6.5.6 Embed the SPL in the SOF

The initial generation of the bitstream (in SOF form) from a project containing an instantiation of the HPS does not yet contain the required SPL. To generate the final SOF with the SPL embedded, use the following command:

```
cd $TOP_FOLDER
quartus_pfg -c bare.sof ia860m-hps.sof -o hps_path=u-boot-socfpga/spl/u-boot-spl-dtb.hex
```

Where “bare.sof” is the initial SOF file and ia860m-hps.sof is the desired output filename.

After you have configured the FPGA with the generated SOF file, the JTAG chain should contain both the Agilex device and the HPS device in the JTAG chain. For example:

```
testfpga@nppdev06:~/ $ jtagconfig
1) USB-BlasterII [1-10]
  6BA00477   S10HPS/AGILEX_HPS/N5X_HPS
  034BD0DD   AGIB023R18AR0
```

## 6.6 Programming NAND with ARM-DS

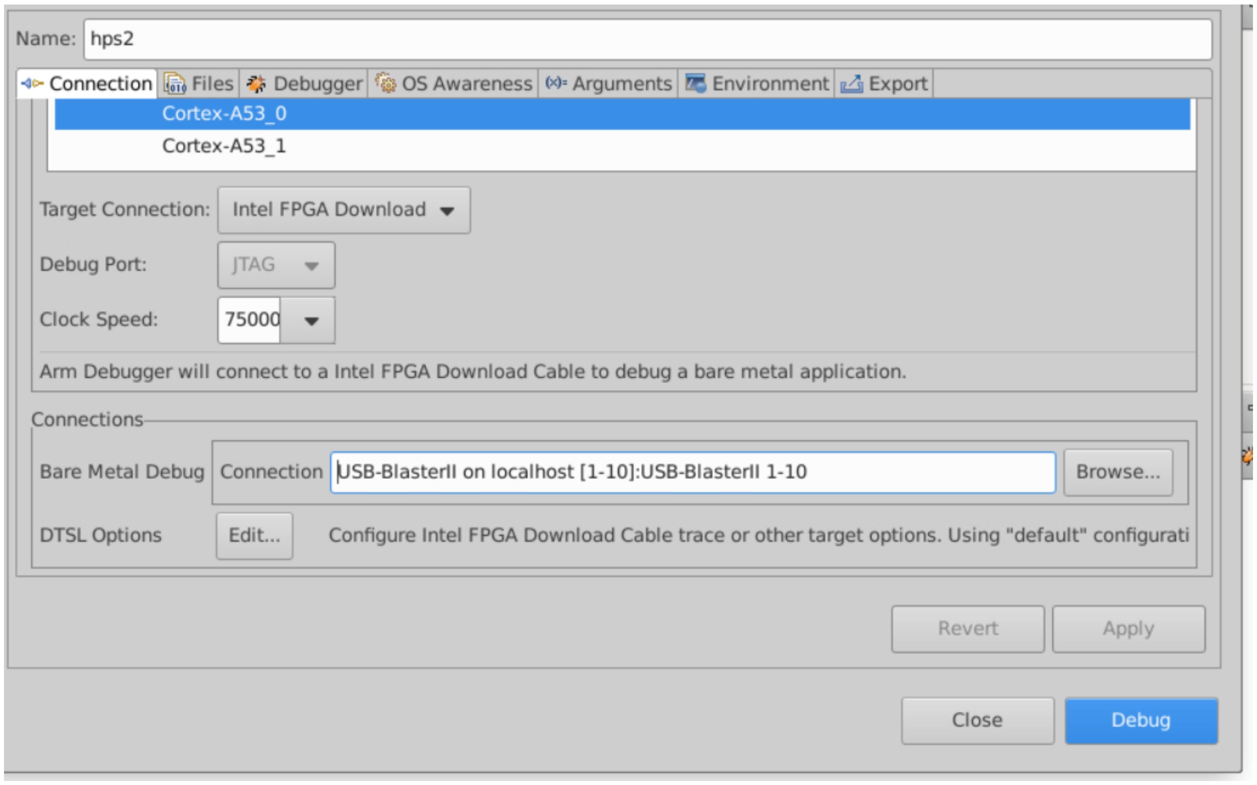
Now that you have loaded the appropriate SOF above, attach to the HPS target using ARM-DS.

**Note**: To make the ARM-DS debug scripts easier to refer to files, we recommend creating a symlink to the u-boot-socfpga directory from the IDE's workspace directory. By default, ARM-DS uses ~/developmentstudio-workspace as the working directory.

```
pushd ~/developmentstudio-workspace
ln -sf $TOP_FOLDER/u-boot-socfpga u-boot-socfpga
popd
```

### 6.6.1 Establish a Debug Session

1. Create a debug configuration for Agilex7 Cortex-A53_0 and select the "Intel FPGA Download" as the target connection type. Under "connections", use the browse button to select your USB-Blaster device.



Name: hps2

Cortex-A53_0
Cortex-A53_1

Target Connection: Intel FPGA Download
Debug Port: JTAG
Clock Speed: 75000

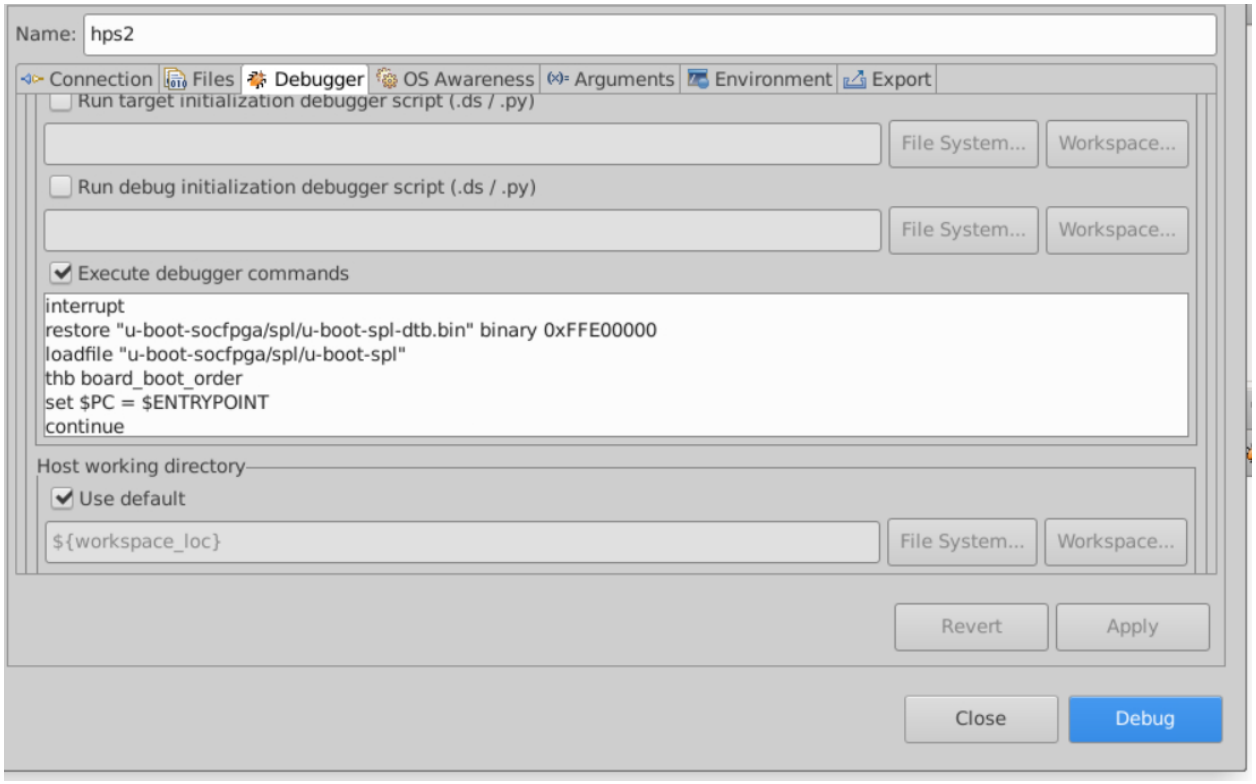
Arm Debugger will connect to a Intel FPGA Download Cable to debug a bare metal application.

Connections

Bare Metal Debug Connection USB-BlasterII on localhost [1-10]:USB-BlasterII 1-10

DTSL Options Configure Intel FPGA Download Cable trace or other target options. Using "default" configurati

2. Switch to the "Debugger" tab and check the box labeled "Execute debugger commands."



Execute debugger commands:

```
interrupt
restore "u-boot-socfpga/spl/u-boot-spl-dtb.bin" binary 0xFFE00000
loadfile "u-boot-socfpga/spl/u-boot-spl"
thb board_boot_order
set $PC = $ENTRYPOINT
continue
```

Host working directory: ${workspace_loc}

3. Copy and paste the following into the command box.

```
interrupt

restore "u-boot-socfpga/spl/u-boot-spl-dtb.bin" binary 0xFFE00000

loadfile "u-boot-socfpga/spl/u-boot-spl"

thb board_boot_order

set $PC = $ENTRYPOINT

continue

wait 60s

set spl_boot_list[0]=0

set $PC=$LR

restore "u-boot-socfpga/u-boot.itb" binary 0x2000000

thb el3:0x1000

continue

wait 60s

symbol-file "u-boot-socfpga/u-boot"

thb el2:relocate_code

continue

wait 60s

symbol-file "u-boot-socfpga/u-boot" ((gd_t*)$x18)->reloc_off
```

```
thb board_init_r

continue

continue
```

4. **Do not** click the Debug button yet.

### 6.6.2 Open the HPS Console

1. Before clicking the “Debug” button in ARM-DS, open a terminal for the HPS UART console.
2. The IA-860m has an FTDI USB device that appears as a four-port USB hub. The fourth port on this device is the HPS UART. Connect to this port using minicom. The port should appear on the host as /dev/ttyUSB3.

```
testfpga@nppdev06:~/djm$ sudo minicom -D /dev/ttyUSB3 -b 115200
```

3. Once the connection is established (you will not likely see any output yet), continue with ARM-DS and click the Debug button.

**Note**: in the example above, minicom is run as superuser. Permissions may need to be adjusted on the tty node for regular users.

### 6.6.3 Writing Firmware to NAND Flash

#### U-Boot

After the debugger loads the SPL and subsequently U-Boot proper, you should see the boot messages appear in the HPS console window. The messages end with an AGILEX# prompt in U-Boot.

**Note**: The communication speed for the UART interface is set at 115,200 baud. However, because of additional data overhead, the actual transfer speed is somewhat lower. This means that sending large amounts of data over UART may take a significant amount of time. Consider using BittWare’s “HPS Data Mover” to transfer large images to the HPS over PCIe.

To transfer the U-Boot FIT image to NAND, you must get it into memory.

1. At the U-Boot prompt, enter “loady $loadaddr” to begin reading the file over the UART using the “ymodem” protocol.

   You should see a stream of “C” characters appearing on the console. You have a limited amount of time to begin the next step while these characters are being printed, so do not delay.

2. Command minicom to send the file to be programmed by pressing “CTRL + A” followed by “S”.  This should open the “Send Files” options menu.
3. Select “ymodem” and then set the cursor over the “[ . . ]” entry. You can then enter the full path to the u-boot.itb file. Take note of the file size in hexadecimal reported after the transfer.  It should be automatically stored in the “filesize” environment variable.

```
SOCFPGA_AGILEX # loady ${loadaddr}

## Ready for binary (ymodem) download to 0x02000000 at 115200 bps...

CC## Total Size      = 0x000de9cc = 911820 Bytes

SOCFPGA_AGILEX # ubi detach

SOCFPGA_AGILEX # nand erase.part u-boot

SOCFPGA_AGILEX # nand write ${loadaddr} u-boot ${filesize}
```

4. It is a good practice to erase the NAND device before programming new content. Perform a chip erase and finally write the FIT image into offset 0.

**Note**: the file size argument to “nand write” assumes hexadecimal, without the “0x” prefix.

## UBI Volumes

**Note**: This operation can take a significant amount of time to complete. Sending large files over uart is extremely slow. Consider using the “HPS DataMover” example to transfer the root.ubi file containing the Linux kernel, flattened device tree, and root filesystem.

1. Perform the same operation as above, but for the root.ubi image created earlier.

```
SOCFPGA_AGILEX # loady $loadaddr

## Ready for binary (ymodem) download to 0x02000000 at 115200 bps...

CC## Total Size      = 0x00f0e9da = 15,788,506 Bytes
```

You should see a stream of “C” characters appearing on the console.

2. Command minicom to send the file to be programmed by pressing “CTRL + A” followed by “S”.  This should open the “Send Files” options menu.
3. Select “ymodem” and then enter the full path to the root.ubi file. Take note of the file size in hexadecimal reported after the transfer.

```
SOCFPGA_AGILEX # ubi detach

SOCFPGA_AGILEX # nand erase.part root

SOCFPGA_AGILEX # nand write.trimffs ${loadaddr} root ${filesize}
SOCFPGA_AGILEX # ubi detach

SOCFPGA_AGILEX # ubi part root

SOCFPGA_AGILEX # saveenv
```

4. Reboot the HPS by reprogramming the FPGA. The SPL should find and load the U-boot Proper from NAND flash and transfer control.
5. U-Boot will count down 5 seconds and will execute bootcmd which will load Linux.

## 6.7 References

- (Yocto Documentation, n.d.)
- (Device Tree Layout, n.d.)
- https://www.rocketboards.org/foswiki/Documentation/BuildingBootloaderAgilex7

[i] Intel Quartus udev rules