# Synaptic Labs'
# HyperBus Memory Controller (HBMC) Tutorial

## T002A: A Qsys based Nios II reference design using Intel's MSGDMA to benchmark memory copy operations on the HyperRAM device using S/Labs' HBMC IP

This stand-alone tutorial describes a simple benchmarking reference design for S/Labs HBMC IP targeted specifically to Intel Cyclone 10LP evaluation board or devboards GmbH HyperMAX 10M25 and 10M50 boards. This reference design is based on Intel's MSGDMA reference project[1]. This tutorial does not depend on, and does not require familiarity with, any of the other tutorials for S/Labs HBMC IP.

HBMC customers interested to benchmark the performance of large burst memory transfer requests on the HyperMAX/ Intel Cyclone 10LP evaluation board will want to explore this tutorial. Generally speaking, doubling the burst length of a memory transfer request addressed to a HyperBus devices results in around a 25% increase in effective bandwidth. When considering memory access contention latency vs. effective bandwidth trade off, the latency vs. performance sweet spot is has a burst memory transfer request length of around around 64 to 256 bytes. Additionally, increasing the HyperBus channel clock speed achieves a near linear increase in effective bandwidth along with a sub-linear decrease in contention access latencies. Developers can use this reference project to quickly explore these and various other trade-offs.

This tutorial describes key aspects of a pre-configured .qsys reference project and then walks through the process of generating and compiling that .Qsys project. This tutorial then describes how to compile the example Nios II source code that controls Intel's MSGDMA and then run the reference design on the development board.

[1] http://www.alterawiki.com/wiki/Modular_SGDMA

## Table of Contents

# Set-Up Requirements:

## a. Step 1: Obtain core materials

2. Download and install Quartus Prime Standard/Lite 17.0 on your PC, please ensure that your PC meets the required minimum specification.
3. For Intel's C10LP Evaluation board :

   a. Create a folder/directory for your work. We suggest: C:\C10_lab****

   b. Download reference design HyperNios_Project_C10LP from: http://media.synaptic-labs.com/pub/2017-Designs/SynapticLabs-HBMC-Tutorial-001/HyperNios_Project_C10LP.zip

   c. Extract to: C:\C10_lab****
4. For devboards HyperMAX board :

   a. Create a folder/directory for your work. We suggest: *C:\HyperMAX_lab*

   b. Download reference design HyperNios_Project_HM10M25 from: *http://media.synaptic-labs.com/pub/2017-Designs/SynapticLabs-HBMC-Tutorial-001/HyperNios_Project_HM10M25.zip*

   c. *Extract to: C:\HyperMAX_lab\*

## Step 2: License Setup

1. Next you need to apply for Synaptic Labs' HyperBus Memory Controller license. You can skip this step if you already installed the license at some earlier stage.

   Free enrollment can be obtained from:

   http://opencore_license_001.synaptic-labs.com/

2. Synaptic Labs offers two Installation Guides that:

   a. Begin by preparing you to enroll to receive a Basic Edition (OpenCore) license

   b. Guide you on how to install the license file you will receive after enrolment

   c. Guide you on how to install the Qsys components that you will receive after enrollment

3. Please download and read one of those Installation Guides:
    a. Developers familiar with installing third party IP into Quartus will probably prefer the streamlined:
    HBMC IP Installation Guide for Experience Developers.
    b. All other developers should download the:
    HBMC IP Installation Guide with Detailed Step-by-Step Instructions.

## Step 3: Install HBMC Qsys Component into the project IP Folder

1. In this tutorial we assume that S/Labs HyperBus Memory Controller (HBMC) will be located in the Project directory.
    a. Other Qsys component installation methods are described in the above mentioned installation Guides.
2. Download the latest version of Synaptic Labs' HBMC IP from:

    http://media.synaptic-labs.com/pub/SoftIP/latest_sll_ca_hbmc_001_be_cots.zip

3. For Intel's C10LP Evaluation board :

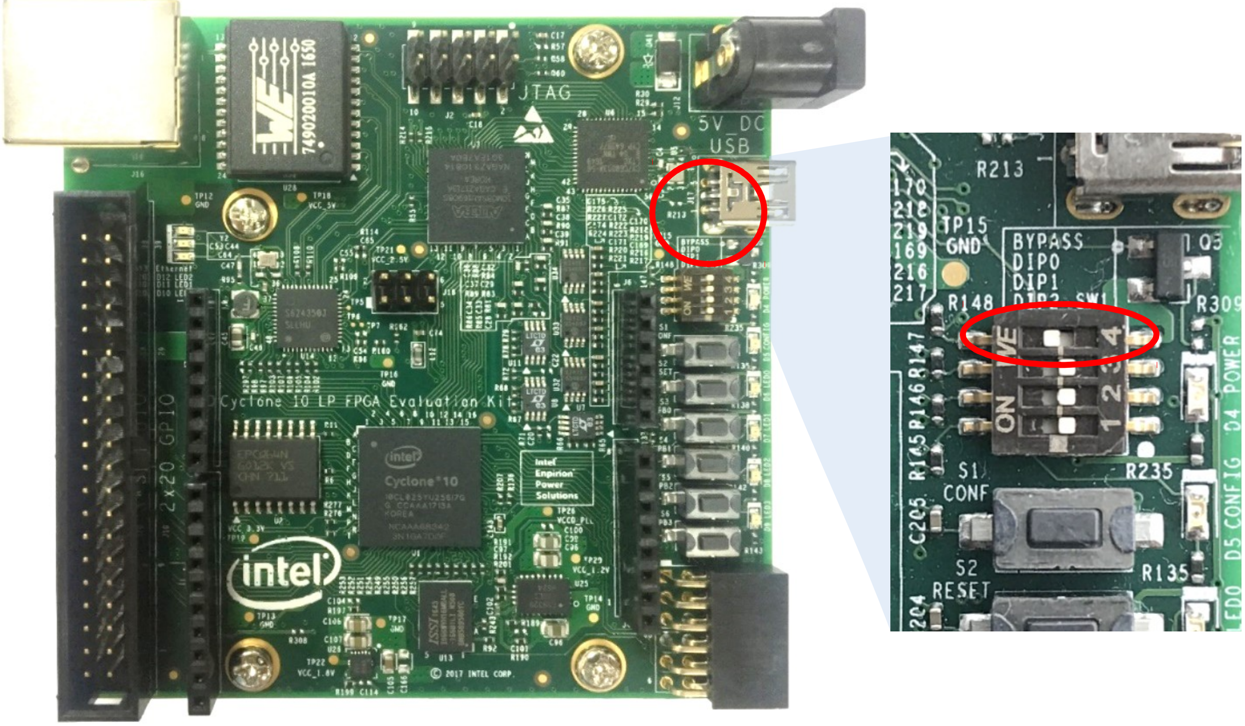
    ○ Extract to the project/ip directory :
    **C:\C10_lab\MSGDMA_project_C10LP\ip**

4. For devboards HyperMAX board :

    ○ Extract to the project/ip directory :
    **C:\C10_lab\MSGDMA_project_HM10M25\ip**

## Step 4:  Cyclone 10 LP Development Board  DIP Switches

You have the Cyclone 10 LP evaluation board and mini USB cable provided.
*Note: the board is powered over USB so no power supply is required.*



***Ensure that DIP Switch 4 on the Cyclone 10 board is set to ON, this bypasses the virtual JTAG system and simplifies board programming.***

# 1. Contents of the reference project

Synaptic Labs' HyperBus Memory Controller (HBMC) Reference design bundle includes the following files and directories:

**MSGDMA_Project_C10LP(MSGDMA_Project_HM10M25)** folder contains the Quartus Prime and Qsys project files for this reference project.

**MSGDMA_Project_C10LP(MSGDMA_Project_HM10M25)**→ **ip** folder will contain S/Labs HBMC encrypted ip (refer to Setup requirements: step3 for more info)

The **MSGDMA_Project_C10LP(MSGDMA_Project_HM10M25)** → s**oftware** folder is the workspace folder for Eclipse

The **MSGDMA_Project_C10LP(MSGDMA_Project_HM10M25)** →**source** folder contains the source code for:

- Intel's MSGDMA project, adapted for use with the HyperRAM memories.

Note: Synaptic Labs' HyperBus Memory Controller (HBMC) IP can ONLY be simulated with Altera's Modelsim Simulator. Please contact Synaptic Labs for a simulation model if required.

# 2. Open the reference Quartus Project

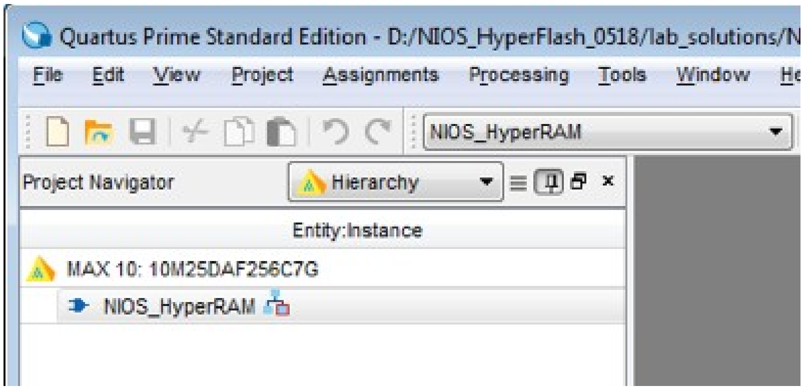
- In the menu bar of Quartus Prime, select **File** → **Open Project…**

- Select the file **HyperNios_MSGDMA.qpf** in the project directory

- Click the [ Open ] button.

## 2.1  Check the correct FPGA device is selected

- Every Quartus project is targeted to a specific FPGA device.  For  devboards GmbH HyperMAX 10M25 and 10M50 boards, check that the FPGA device matches the one being used.   This step is not needed for Intel C10LP evaluation board.



Quartus Prime Standard Edition - D:/NIOS_HyperFlash_0518/lab_solutions/N

Entity:Instance

MAX 10: 10M25DAF256C7G

NIOS_HyperRAM

- The HyperMAX 10M25 board employs the **`10M25DAF256C7G`** device.
  The HyperMAX 10M50 board employs the **`10M50DAF256I7G`** device.
- The Intel C10LP Evaluation board employs  the **`10CL025YU256I7G`** device

- If you need to change the FPGA device for your specific board:

  - Ensure that there are no instances of the Qsys application running.
  - Right click on the device name.
  - Select “Devices…” in the pop up window.
  - A new window will open.  Select the “Device” tab.
  - Copy the required device name into the “Name filter:” field.
  - A popup window will ask: “Do you want to remove all location assignments?” Click on the [ No ] button.
  - Select the requested device in the “available devices:” field so that it is highlighted in blue.
  - Then click on the [ Okay ] button.

- The Quartus project and the Qsys project are now configured for the FPGA device you selected.

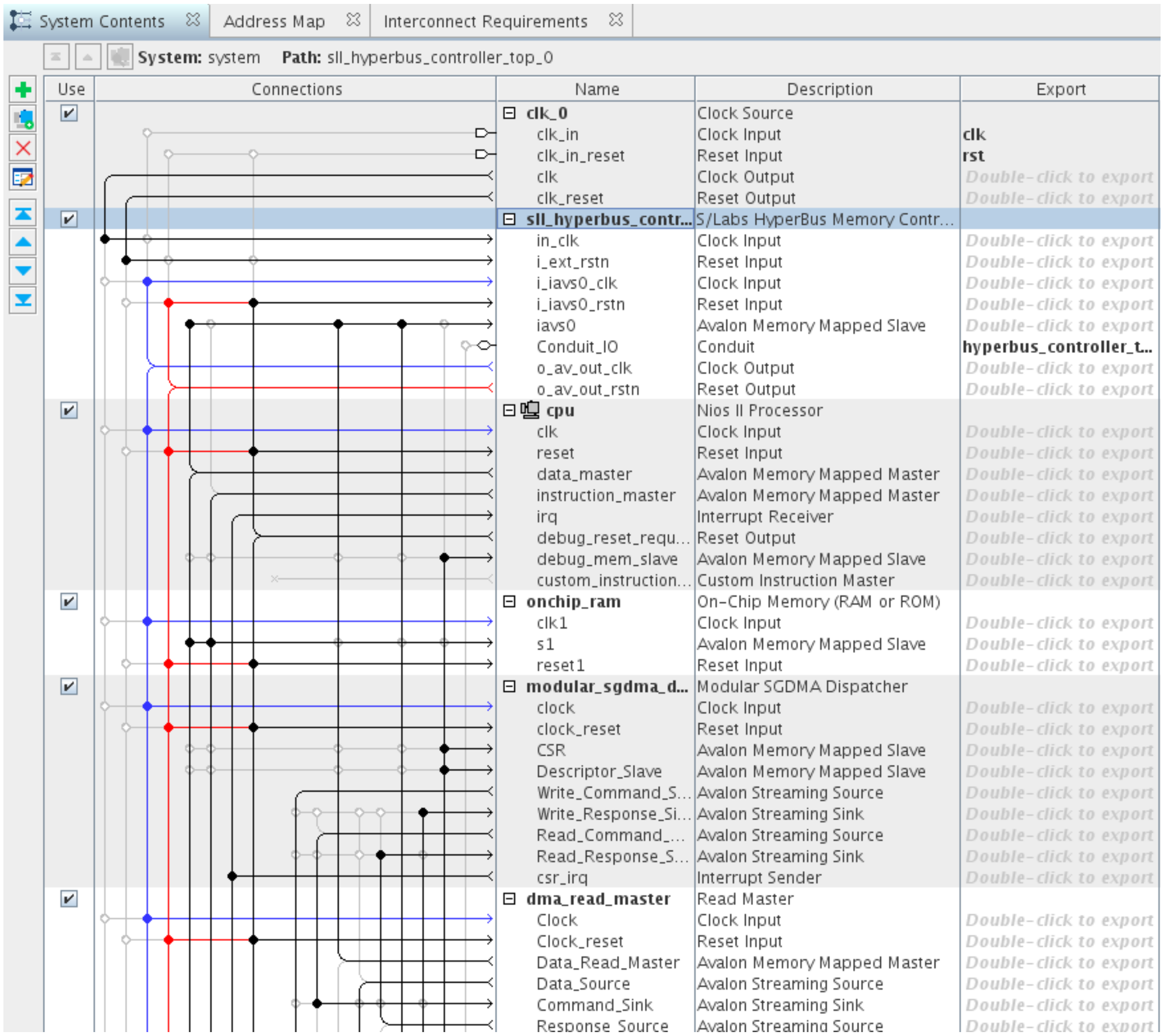
# 3.  Open the reference Qsys project

- In the menu bar of Quartus Prime, select  **Tools→Qsys**
- Select the file **system.qsys** in the project directory
- Click the [ Open ] button.

# 4. Explore and configure the reference Qsys project

## 4.1 Components employed in the reference project

The reference Qsys project in this tutorial employs a NiosII/f processor, Synaptic Labs' HyperBus Memory Controller (HBMC) IP, Altera’s On-chip Memory module to store code and data in on chip SRAM, the Altera Modular Scatter Gather Direct Memory Access (SGDMA) Dispatcher and various peripherals such as Altera’s JTAG UART and timer modules as illustrated below. All these Qsys components are connected together.



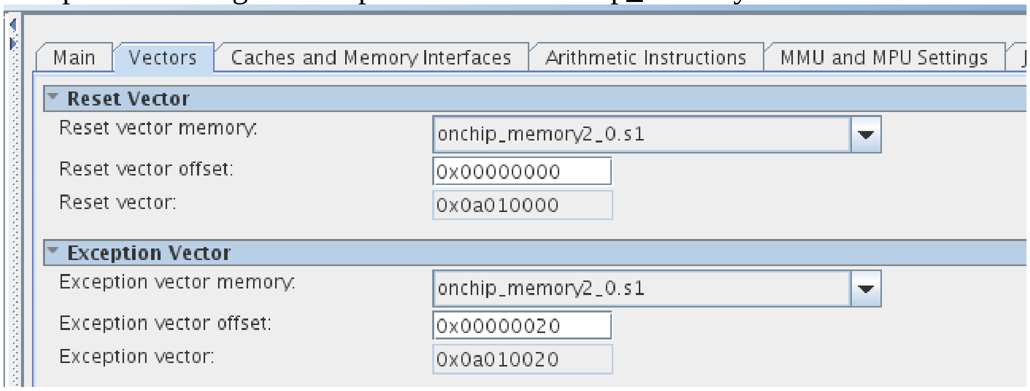
**System:** system **Path:** sll_hyperbus_controller_top_0

| Use | Connections | Name | Description | Export |
|---|---|---|---|---|
| ☑ | | **clk_0** | Clock Source | |
| | | clk_in | Clock Input | **clk** |
| | | clk_in_reset | Reset Input | **rst** |
| | | clk | Clock Output | *Double-click to export* |
| | | clk_reset | Reset Output | *Double-click to export* |
| ☑ | | **sll_hyperbus_contr...** | S/Labs HyperBus Memory Contr... | |
| | | in_clk | Clock Input | *Double-click to export* |
| | | i_ext_rstn | Reset Input | *Double-click to export* |
| | | i_iavs0_clk | Clock Input | *Double-click to export* |
| | | i_iavs0_rstn | Reset Input | *Double-click to export* |
| | | iavs0 | Avalon Memory Mapped Slave | *Double-click to export* |
| | | Conduit_IO | Conduit | **hyperbus_controller_t...** |
| | | o_av_out_clk | Clock Output | *Double-click to export* |
| | | o_av_out_rstn | Reset Output | *Double-click to export* |
| ☑ | | **cpu** | Nios II Processor | |
| | | clk | Clock Input | *Double-click to export* |
| | | reset | Reset Input | *Double-click to export* |
| | | data_master | Avalon Memory Mapped Master | *Double-click to export* |
| | | instruction_master | Avalon Memory Mapped Master | *Double-click to export* |
| | | irq | Interrupt Receiver | *Double-click to export* |
| | | debug_reset_requ... | Reset Output | *Double-click to export* |
| | | debug_mem_slave | Avalon Memory Mapped Slave | *Double-click to export* |
| | | custom_instruction... | Custom Instruction Master | *Double-click to export* |
| ☑ | | **onchip_ram** | On-Chip Memory (RAM or ROM) | |
| | | clk1 | Clock Input | *Double-click to export* |
| | | s1 | Avalon Memory Mapped Slave | *Double-click to export* |
| | | reset1 | Reset Input | *Double-click to export* |
| ☑ | | **modular_sgdma_d...** | Modular SGDMA Dispatcher | |
| | | clock | Clock Input | *Double-click to export* |
| | | clock_reset | Reset Input | *Double-click to export* |
| | | CSR | Avalon Memory Mapped Slave | *Double-click to export* |
| | | Descriptor_Slave | Avalon Memory Mapped Slave | *Double-click to export* |
| | | Write_Command_S... | Avalon Streaming Source | *Double-click to export* |
| | | Write_Response_Si... | Avalon Streaming Sink | *Double-click to export* |
| | | Read_Command_... | Avalon Streaming Source | *Double-click to export* |
| | | Read_Response_S... | Avalon Streaming Sink | *Double-click to export* |
| | | csr_irq | Interrupt Sender | *Double-click to export* |
| ☑ | | **dma_read_master** | Read Master | |
| | | Clock | Clock Input | *Double-click to export* |
| | | Clock_reset | Reset Input | *Double-click to export* |
| | | Data_Read_Master | Avalon Memory Mapped Master | *Double-click to export* |
| | | Data_Source | Avalon Streaming Source | *Double-click to export* |
| | | Command_Sink | Avalon Streaming Sink | *Double-click to export* |
| | | Response_Source | Avalon Streaming Source | *Double-click to export* |

Please note that the 50 MHz clock pin of the FPGA is mapped to the **in_clk** port of S/Labs HyperBus Memory Controller instance. The HBMC instance includes an integrated PLL that generates the clocks to drive the Avalon bus and the HyperBus memory channel. In particular the **o_av_out_clk** port of the HBMC is used to drive the Avalon bus and all of the components illustrated above.

## 4.2 Nios II/f Gen2 processor configuration

In this example, the Nios II/f Reset and Exception vectors are mapped to onchip_memory as illustrated below. This means that the Nios II/f processor will look for the boot code and exception handling / interrupt code in the onchip_memory module.



**Reset Vector**

Reset vector memory: onchip_memory2_0.s1

Reset vector offset: 0x00000000

Reset vector: 0x0a010000

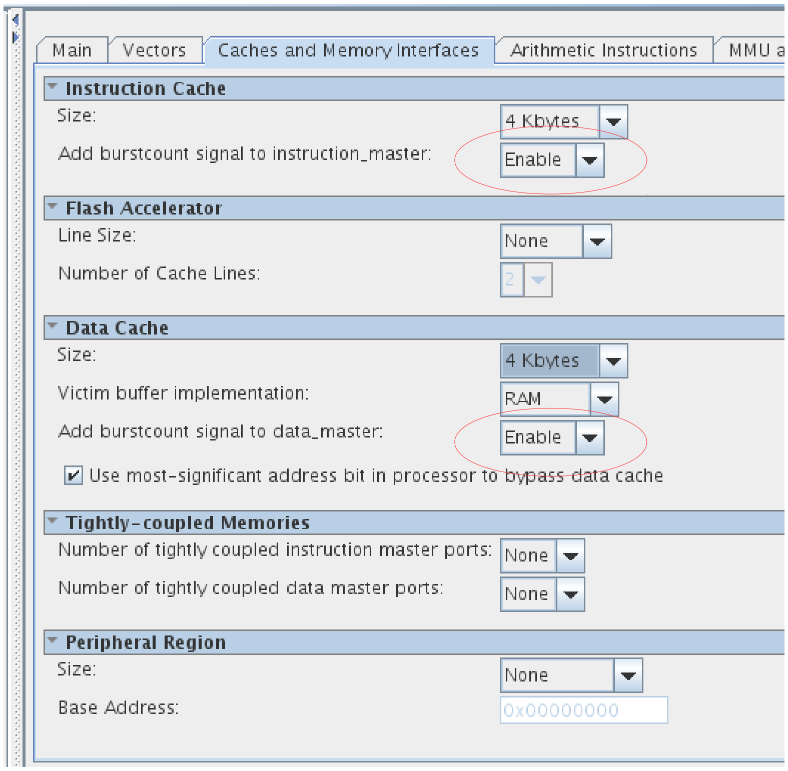
**Exception Vector**

Exception vector memory: onchip_memory2_0.s1

Exception vector offset: 0x00000020

Exception vector: 0x0a010020

As illustrated below, the instruction and data caches of the Nios II/f core have both been set to 4Kbytes in size to accelerate software performance. The instruction and data caches have both been configured with their **burstcount signal enabled** so that both caches issue 32-byte burst memory transfer requests. This is done because: (a) the HyperBus protocol employs burst memory transfer requests with closed page mode of operation; and (b) SLL's HBMC employs an Avalon interface with burst mode of operation.



**Instruction Cache**

Size: 4 Kbytes

Add burstcount signal to instruction_master: Enable

**Flash Accelerator**

Line Size: None

Number of Cache Lines: 2

**Data Cache**

Size: 4 Kbytes

Victim buffer implementation: RAM

Add burstcount signal to data_master: Enable

☑ Use most-significant address bit in processor to bypass data cache

**Tightly-coupled Memories**

Number of tightly coupled instruction master ports: None

Number of tightly coupled data master ports: None

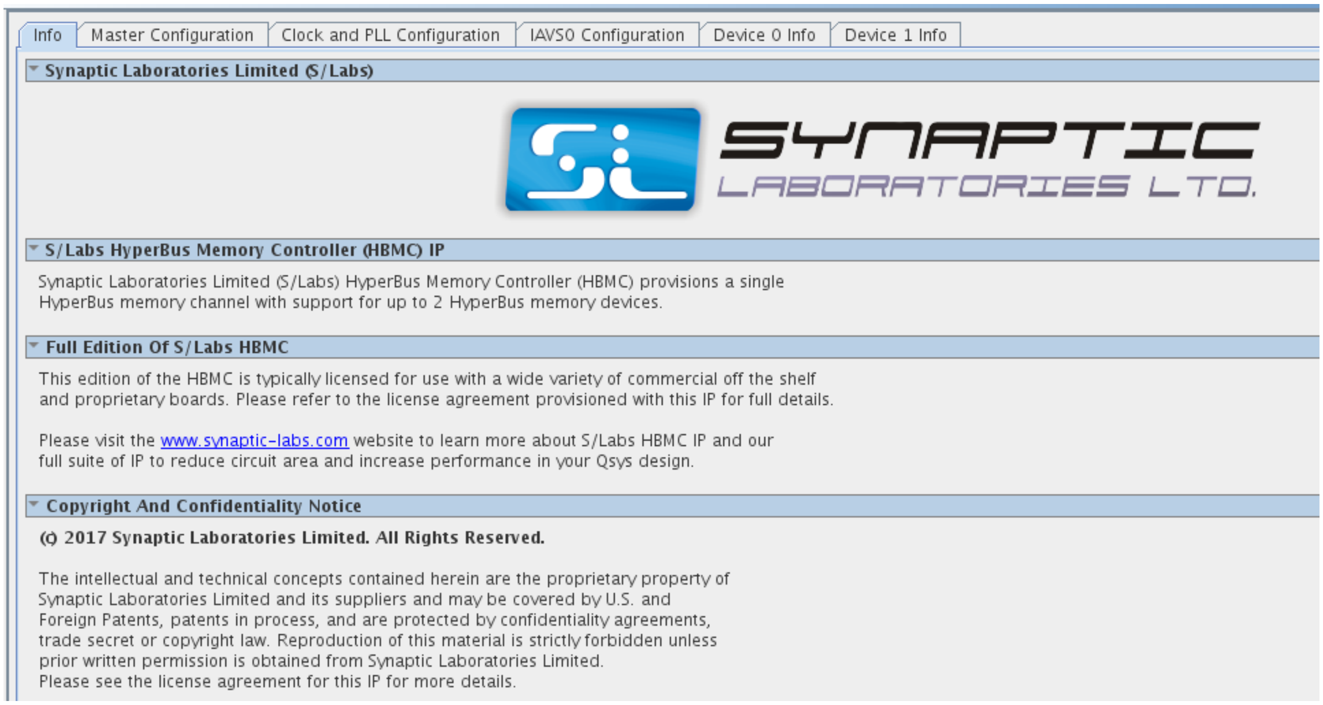
**Peripheral Region**

Size: None

Base Address: 0x00000000

## 4.3 Configuring S/Labs HyperBus Memory Controller

Synaptic Labs' HyperBus Memory Controller has been pre-configured in this reference project. In this section we will describe various configuration options that you may wish to change.



**Synaptic Laboratories Limited (S/Labs)**

**S/Labs HyperBus Memory Controller (HBMC) IP**

Synaptic Laboratories Limited (S/Labs) HyperBus Memory Controller (HBMC) provisions a single HyperBus memory channel with support for up to 2 HyperBus memory devices.

**Full Edition Of S/Labs HBMC**

This edition of the HBMC is typically licensed for use with a wide variety of commercial off the shelf and proprietary boards. Please refer to the license agreement provisioned with this IP for full details.

Please visit the www.synaptic-labs.com website to learn more about S/Labs HBMC IP and our full suite of IP to reduce circuit area and increase performance in your Qsys design.

**Copyright And Confidentiality Notice**

**(c) 2017 Synaptic Laboratories Limited. All Rights Reserved.**

The intellectual and technical concepts contained herein are the proprietary property of Synaptic Laboratories Limited and its suppliers and may be covered by U.S. and Foreign Patents, patents in process, and are protected by confidentiality agreements, trade secret or copyright law. Reproduction of this material is strictly forbidden unless prior written permission is obtained from Synaptic Laboratories Limited. Please see the license agreement for this IP for more details.

**In the "Master Configuration" tab**

The open-core edition of SLL HBMC IP only supports HyperRAM. The full edition of SLL HBMC IP supports any combination of HyperFlash and HyperRAM. Both editions offer preconfigured memory options for supported COTS FPGA development boards. The full edition of SLL HBMC IP also includes the option to manually configure the HyperBus devices.

To configure the project to just use HyperRAM on the HyperMAX/Intel C10LP evaluation board:

- The **FPGA board type** field is set to either:
  - **Devboards - HyperMAX 10M25 (HyperRAM)** or
  - **Devboards - HyperMAX 10M50 (HyperRAM) or**
  - **Intel – Cyclone 10LP Evaluation Kit (HyperRAM)**

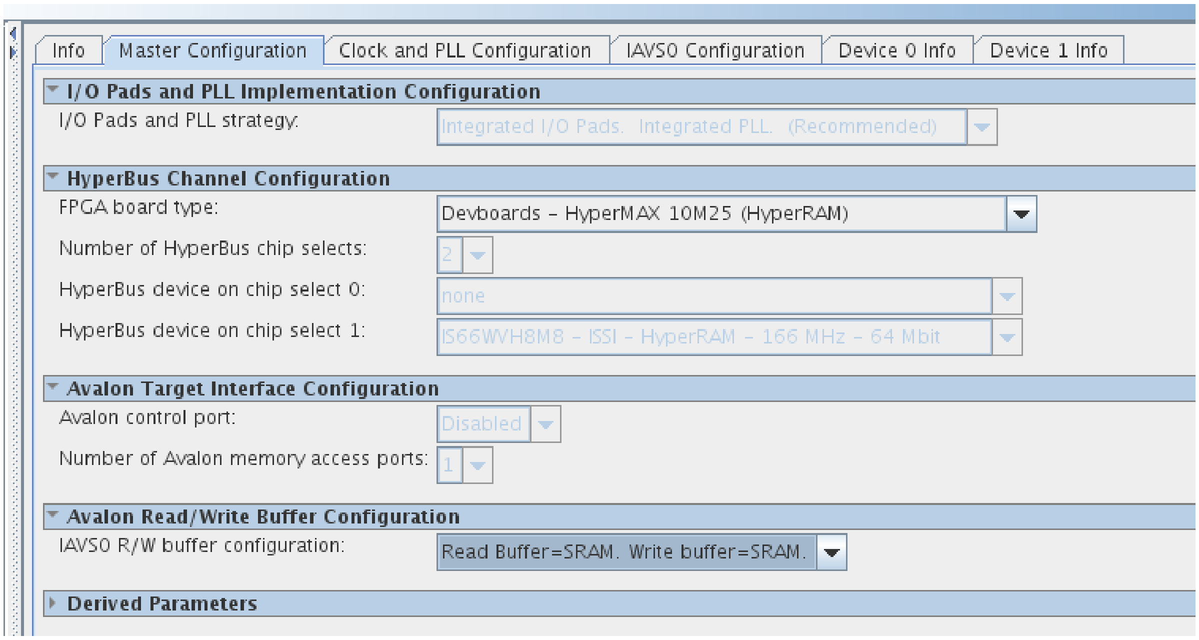
To configure the project to use both HyperFlash and HyperRAM on the HyperMAX board:

- The **FPGA board type** field is set to either:
  - **Devboards - HyperMAX 10M25 (HyperFlash and HyperRAM)** or
  - **Devboards - HyperMAX 10M50 (HyperFlash and HyperRAM)**

***Currently, Intel's Cyclone 10LP evaluation board does not contain a HyperFlash device.***

**This tutorial is written assuming the use of (HyperRAM) only.**

**…...for Devboards HyperMAX board**



I/O Pads and PLL Implementation Configuration

I/O Pads and PLL strategy: Integrated I/O Pads. Integrated PLL. (Recommended)

HyperBus Channel Configuration

FPGA board type: Devboards - HyperMAX 10M25 (HyperRAM)

Number of HyperBus chip selects: 2

HyperBus device on chip select 0: none

HyperBus device on chip select 1: IS66WVH8M8 - ISSI - HyperRAM - 166 MHz - 64 Mbit

Avalon Target Interface Configuration

Avalon control port: Disabled

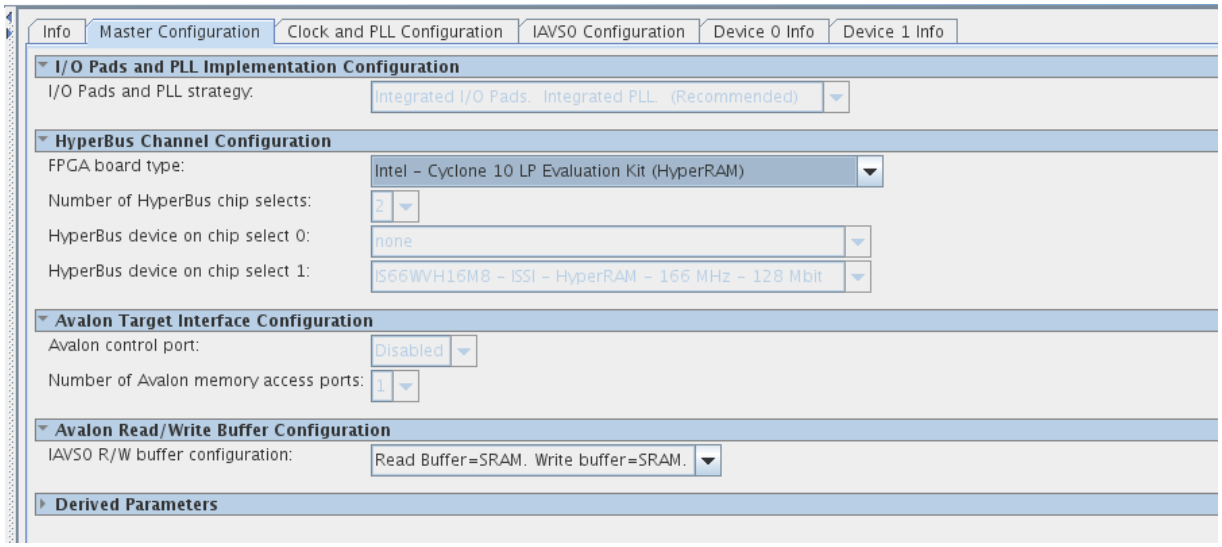
Number of Avalon memory access ports: 1

Avalon Read/Write Buffer Configuration

IAVS0 R/W buffer configuration: Read Buffer=SRAM. Write buffer=SRAM.

Derived Parameters

**…...for Intel Cyclone 10LP Evaluation Kit**



I/O Pads and PLL Implementation Configuration

I/O Pads and PLL strategy: Integrated I/O Pads. Integrated PLL. (Recommended)

HyperBus Channel Configuration

FPGA board type: Intel - Cyclone 10 LP Evaluation Kit (HyperRAM)

Number of HyperBus chip selects: 2

HyperBus device on chip select 0: none

HyperBus device on chip select 1: IS66WVH16M8 - ISSI - HyperRAM - 166 MHz - 128 Mbit

Avalon Target Interface Configuration

Avalon control port: Disabled

Number of Avalon memory access ports: 1

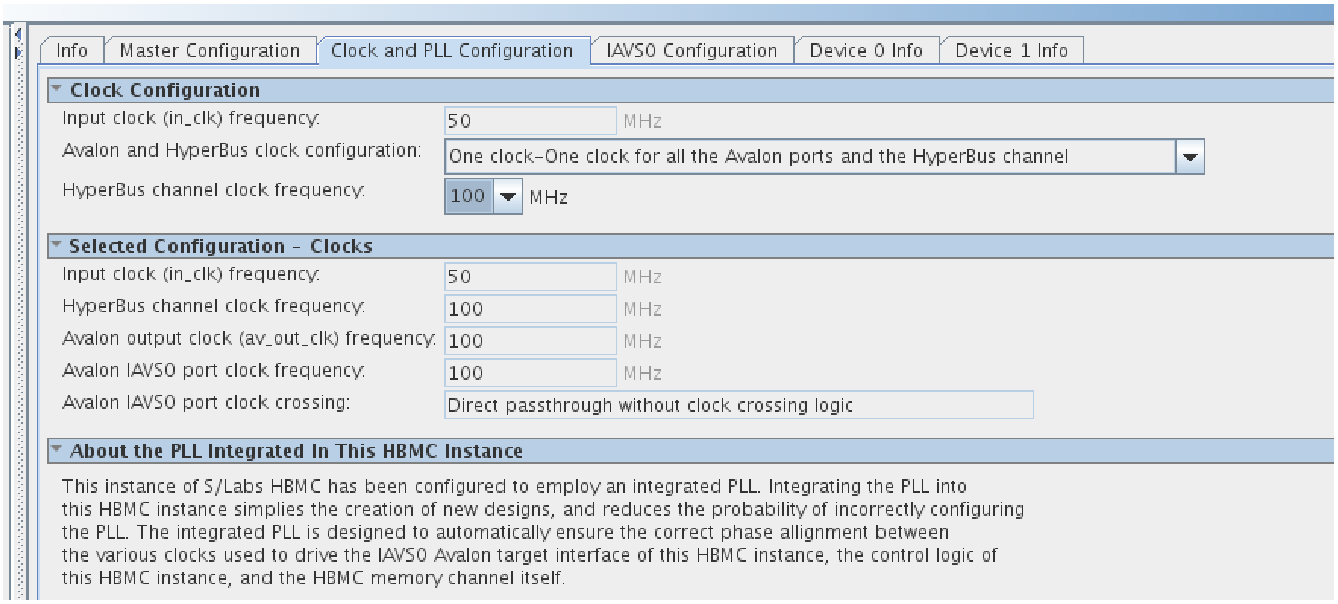
Avalon Read/Write Buffer Configuration

IAVS0 R/W buffer configuration: Read Buffer=SRAM. Write buffer=SRAM.

Derived Parameters

## In the Clock and PLL Configuration Tab (opencore edition)

- The **Avalon and HyperBus clock configuration** field is set to **One clock**
- The **HyperBus channel clock frequency** field is set to **100 MHz**



**Clock Configuration**

Input clock (in_clk) frequency: 50 MHz

Avalon and HyperBus clock configuration: One clock-One clock for all the Avalon ports and the HyperBus channel

HyperBus channel clock frequency: 100 MHz

**Selected Configuration - Clocks**

Input clock (in_clk) frequency: 50 MHz

HyperBus channel clock frequency: 100 MHz

Avalon output clock (av_out_clk) frequency: 100 MHz

Avalon IAVS0 port clock frequency: 100 MHz

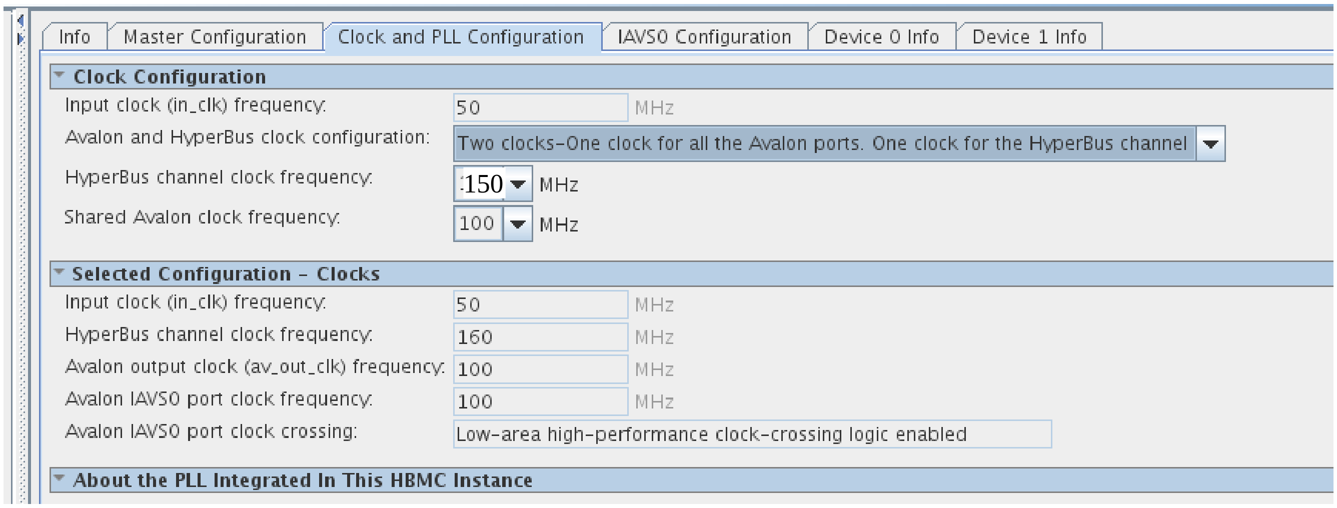
Avalon IAVS0 port clock crossing: Direct passthrough without clock crossing logic

**About the PLL Integrated In This HBMC Instance**

This instance of S/Labs HBMC has been configured to employ an integrated PLL. Integrating the PLL into this HBMC instance simplies the creation of new designs, and reduces the probability of incorrectly configuring the PLL. The integrated PLL is designed to automatically ensure the correct phase allignment between the various clocks used to drive the IAVS0 Avalon target interface of this HBMC instance, the control logic of this HBMC instance, and the HBMC memory channel itself.

## In the Clock and PLL Configuration Tab (full edition)

- The **Avalon and HyperBus clock configuration** field is set to **Two clocks**
- The **HyperBus channel clock frequency** field is set to **150 MHz**
- The **Shared Avalon clock frequency** field is set to **100 MHz**



**Clock Configuration**

Input clock (in_clk) frequency: 50 MHz

Avalon and HyperBus clock configuration: Two clocks-One clock for all the Avalon ports. One clock for the HyperBus channel

HyperBus channel clock frequency: 150 MHz

Shared Avalon clock frequency: 100 MHz

**Selected Configuration - Clocks**

Input clock (in_clk) frequency: 50 MHz

HyperBus channel clock frequency: 160 MHz

Avalon output clock (av_out_clk) frequency: 100 MHz

Avalon IAVS0 port clock frequency: 100 MHz

Avalon IAVS0 port clock crossing: Low-area high-performance clock-crossing logic enabled

**About the PLL Integrated In This HBMC Instance**

## In the Ingress Avalon Slave 0 (IAVS0) configuration tab

The IAVS0 port is used to access all HyperBus memories connected to the HyperBus Memory Controller IP. The most common configuration of the IAVS0 port is as follows:

IAVS0: Ingress Avalon port stage

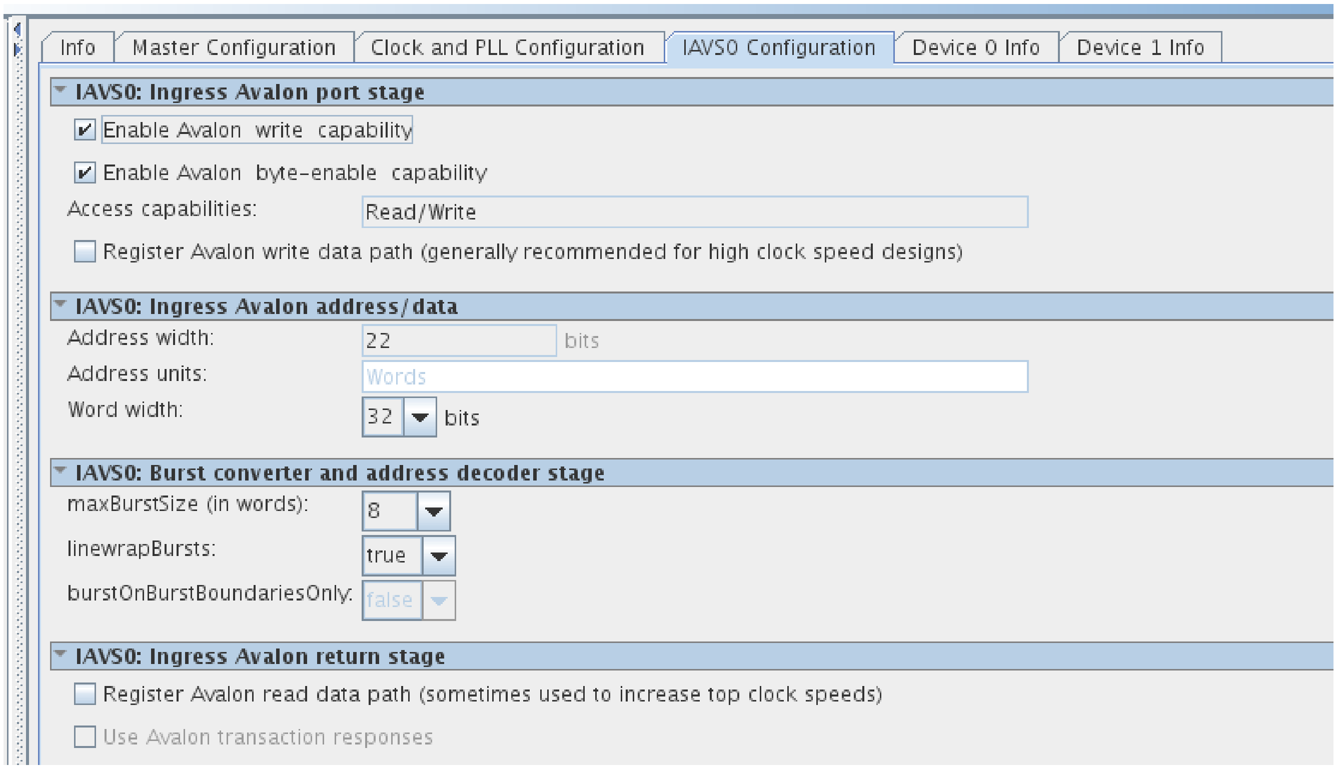
- The **Enable Avalon write capability** field is **checked**
- The **Enable Avalon byte-enable capability** field is **checked**
- The **Register Avalon Write data** field is left **unchecked**

IAVS0: Burst Converter and address decoder stage

- The **max BurstSize (in Words)** field is set to **8 words**
  - The full edition of S/Labs HBMC IP supports long bursts:
    8 words = 32 bytes
    16 words = 64 bytes
    32 words = 128 bytes
    64 words = 256 bytes
    …
  - Generally speaking, doubling the burst length of a memory transfer request addressed to a HyperBus devices results in around a 25% increase in effective bandwidth.
  - When considering memory access contention latency vs. effective bandwidth trade off, the latency vs. performance sweet spot is has a burst memory transfer request length of around around 64 to 256 bytes.
- In this tutorial, set the **lineWrapBursts** field to **true**
  - LineWrapBursts are used to accelerate the Nios II cache accesses to HyperBus devices.
  - LineWrapBursts are currently only supported by S/Labs HBMC IP when the value of the
    **max BurstSize** is **8 words.**

IAVS0: Ingress Avalon return stage

- The **Register Avalon Read data path** field is left **unchecked**
- The **Use Avalon Transaction Response** field is left **unchecked**



IAVS0: Ingress Avalon port stage

☑ Enable Avalon write capability

☑ Enable Avalon byte-enable capability

Access capabilities: Read/Write

□ Register Avalon write data path (generally recommended for high clock speed designs)

IAVS0: Ingress Avalon address/data

Address width: 22 bits

Address units: Words

Word width: 32 bits

IAVS0: Burst converter and address decoder stage

maxBurstSize (in words): 8

linewrapBursts: true

burstOnBurstBoundariesOnly: false

IAVS0: Ingress Avalon return stage

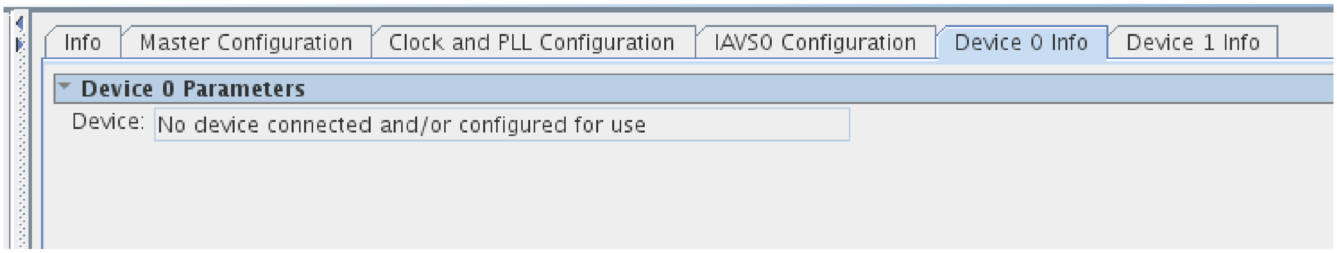
□ Register Avalon read data path (sometimes used to increase top clock speeds)

□ Use Avalon transaction responses

The GUI interface includes significant inbuilt documentation. Moving your mouse over a configuration field pops up the on-screen help for that field.

**In the Device 0 Info tab**

The “Device 0 Info” tab provides information about the HyperBus device connected to chip select 0. If in the "Master Configuration" tab, the **FPGA board type** field is set to **Devboards - HyperMAX 10M25/50 (HyperRAM) or Intel – Cyclone 10LP Evaluation Kit (HyperRAM),**, these fields remain empty.



Device: No device connected and/or configured for use

If in the "Master Configuration" tab, the **FPGA board type** field is set to **Devboards - HyperMAX 10M25/50 (HyperFlash and HyperRAM),** the following table will show the parameters, configuration and timing for the HyperFlash memory.

Device 0 Configuration

- In this tutorial the **Use factory default settings for this HyperBus device** field is left **checked**

**In the Device1 Info tab**

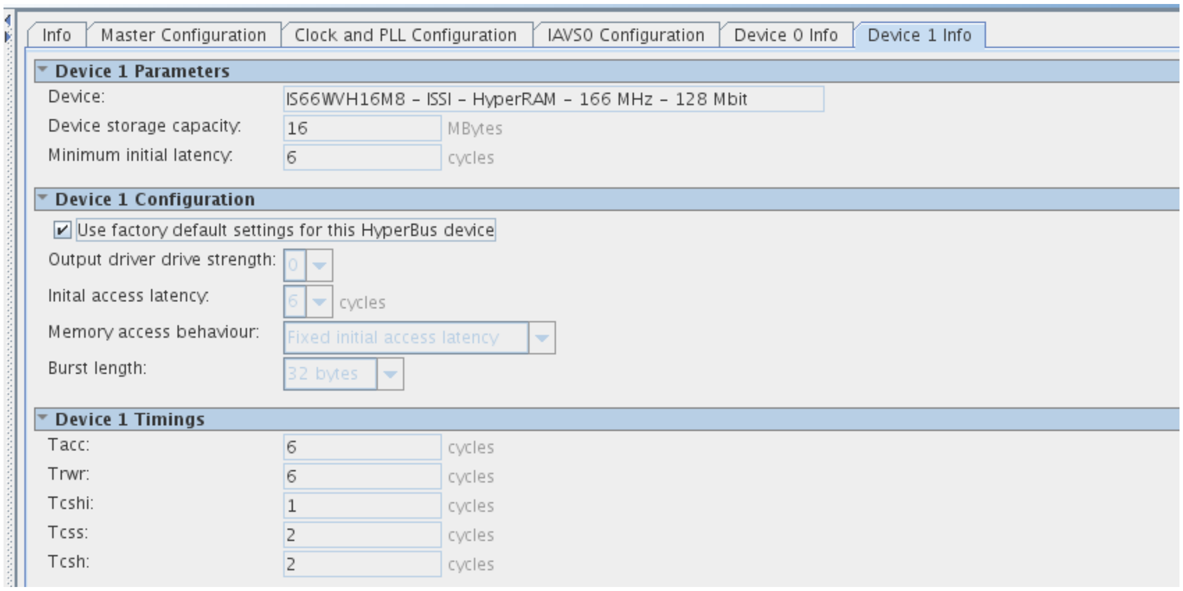
The table below will show the parameters, configuration and timing for the HyperRAM memory.

Device 1 Configuration

- In this tutorial the **Use factory default settings for this HyperBus device** field is left **checked**

The exact parameters on your screen may be different because:

- The HyperMAX 10M25 board employs a   64 Megabit HyperRAM device.
- The HyperMAX 10M50 board employs a 128 Megabit HyperRAM device.
- The Intel C10LP Evaluation kit employs a 128 Megabit HyperRAM device.



**Device 1 Parameters**

Device: IS66WVH16M8 - ISSI - HyperRAM - 166 MHz - 128 Mbit

Device storage capacity: 16 MBytes

Minimum initial latency: 6 cycles

**Device 1 Configuration**

☑ Use factory default settings for this HyperBus device

Output driver drive strength: 0

Inital access latency: 6 cycles

Memory access behaviour: Fixed initial access latency

Burst length: 32 bytes

**Device 1 Timings**

Tacc: 6 cycles

Trwr: 6 cycles

Tcshi: 1 cycles

Tcss: 2 cycles

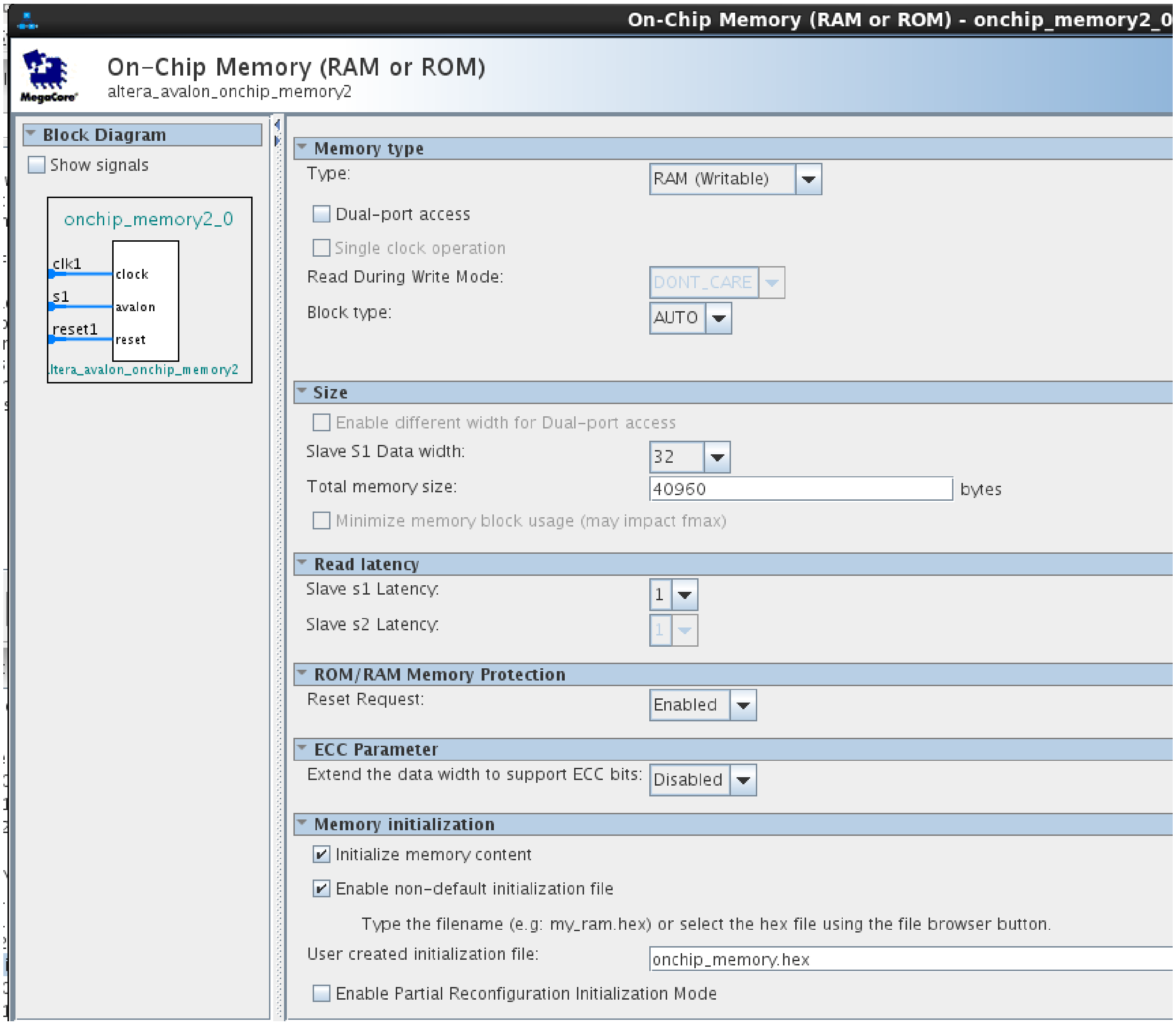
Tcsh: 2 cycles

## 4.4 Configuration of Altera’s On-Chip Memory

In this reference project, Altera's On-Chip Memory is configured as a 40 Kilobyte single port RAM. We will setup this memory to be initialized by the Nios II SBT for Eclipse

- Please ensure that:
    - **[x] Initialize memory content** is **Ticked**
    - **[x] Enable non-default initialization** file is **Ticked**
    - **[x] User created initialization file** is **onchip_memory.hex**

      Note: If you do not specify an initial .hex filename, Nios II SBT will **not** generate the memory initialization file. After we have generated that file, we will then need to update Qsys with the relative/full path to the .hex file.



On-Chip Memory (RAM or ROM) - onchip_memory2_0
MegaCore
Block Diagram
onchip_memory2_0
clk1
s1
reset1
clock
avalon
reset
ltera_avalon_onchip_memory2

On-Chip Memory (RAM or ROM)
altera_avalon_onchip_memory2

☐ Show signals

**Memory type**

Type: RAM (Writable)

☐ Dual-port access

☐ Single clock operation

Read During Write Mode: DONT_CARE

Block type: AUTO

**Size**

☐ Enable different width for Dual-port access

Slave S1 Data width: 32

Total memory size: 40960 bytes

☐ Minimize memory block usage (may impact fmax)

**Read latency**

Slave s1 Latency: 1

Slave s2 Latency: 1

**ROM/RAM Memory Protection**

Reset Request: Enabled

**ECC Parameter**

Extend the data width to support ECC bits: Disabled

**Memory initialization**

☑ Initialize memory content

☑ Enable non-default initialization file

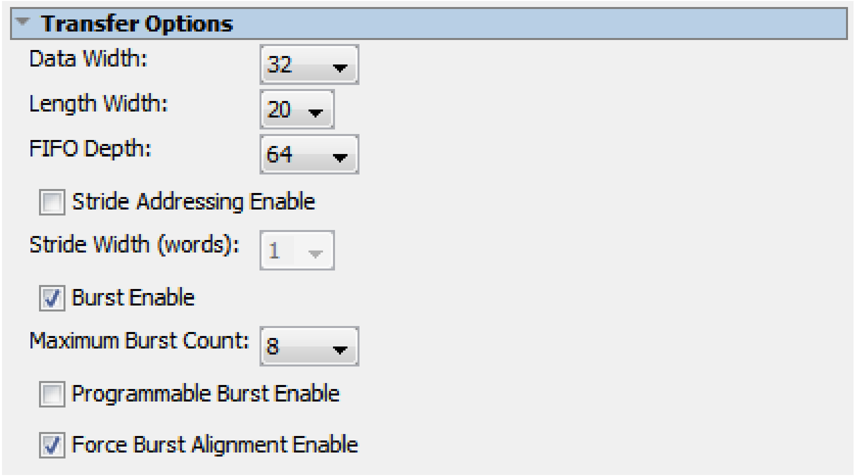
Type the filename (e.g: my_ram.hex) or select the hex file using the file browser button.

User created initialization file: onchip_memory.hex

☐ Enable Partial Reconfiguration Initialization Mode

## 4.5 Configuration of Intel’s dma_read_master and dma_write_master modules

In this reference project, both of Intel’s **dma_read_master** and **dma_write_master** modules are configured to employ burst memory transfer requests of **8 words** (32 bytes) in length.  You can adjust the burst length of each module in its respective graphical user interface:



Transfer Options

Data Width: 32

Length Width: 20

FIFO Depth: 64

□ Stride Addressing Enable

Stride Width (words): 1

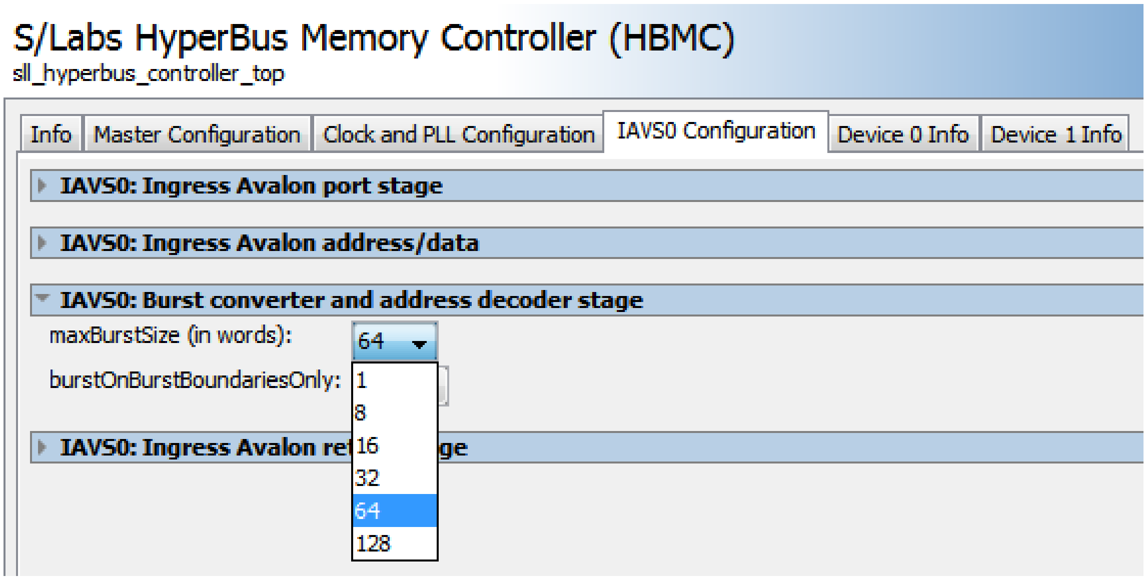
☑ Burst Enable

Maximum Burst Count: 8

□ Programmable Burst Enable

☑ Force Burst Alignment Enable

If you increase the burst length of the **dma_read_master** and/or **dma_write_master** module, you must also ensure that the **maxBurstSize** of S/Labs HBMC IP is set to a value of equal or larger burst length.



S/Labs HyperBus Memory Controller (HBMC)

sll_hyperbus_controller_top

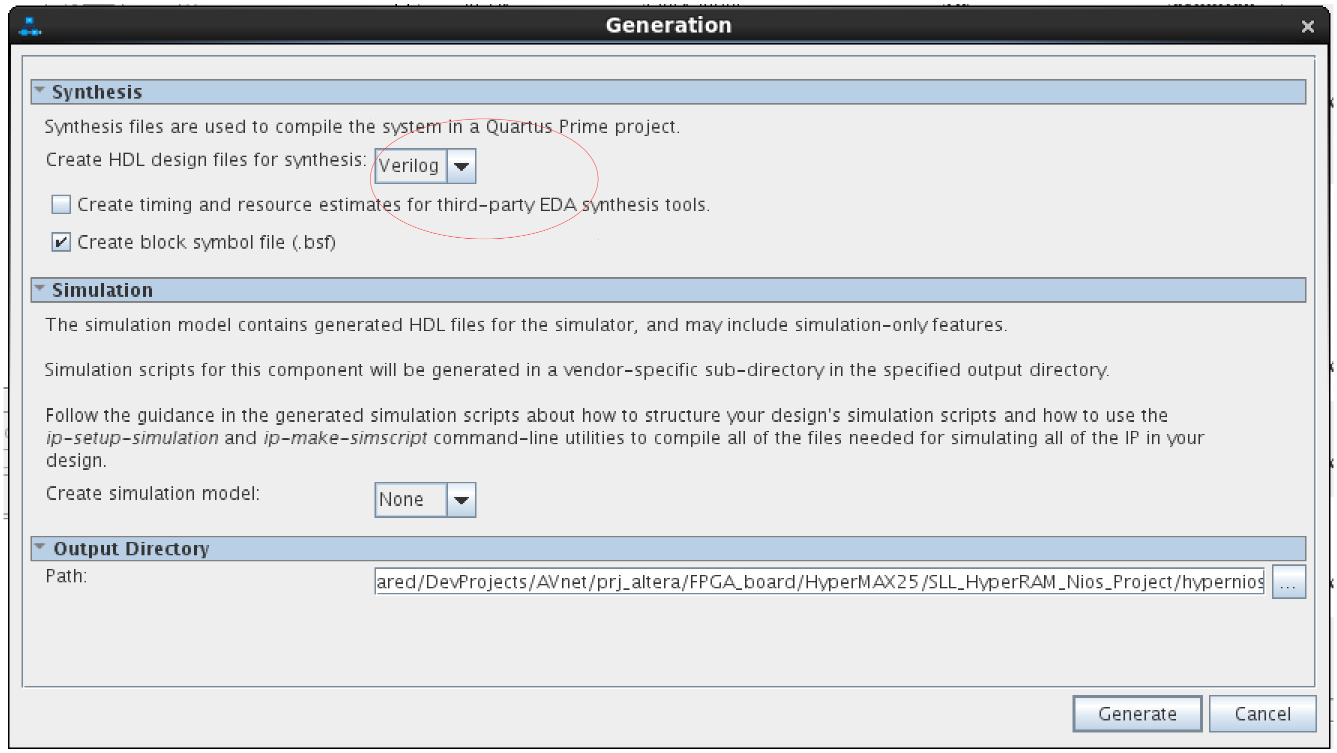
Info
Master Configuration
Clock and PLL Configuration
IAVS0 Configuration
Device 0 Info
Device 1 Info
IAVS0: Ingress Avalon port stage
IAVS0: Ingress Avalon address/data
IAVS0: Burst converter and address decoder stage
IAVS0: Ingress Avalon re
ge
1
8
16
32
64
128

maxBurstSize (in words): 64

burstOnBurstBoundariesOnly:

# 5. Generating the Qsys Design

Once the Qsys project has been correctly configured, press the [ **Generate HDL… ]** button on the bottom right hand side of the Qsys window**.**



Generation

**Synthesis**

Synthesis files are used to compile the system in a Quartus Prime project.

Create HDL design files for synthesis: Verilog

□ Create timing and resource estimates for third-party EDA synthesis tools.

☑ Create block symbol file (.bsf)

**Simulation**

The simulation model contains generated HDL files for the simulator, and may include simulation-only features.

Simulation scripts for this component will be generated in a vendor-specific sub-directory in the specified output directory.

Follow the guidance in the generated simulation scripts about how to structure your design's simulation scripts and how to use the *ip-setup-simulation* and *ip-make-simscript* command-line utilities to compile all of the files needed for simulating all of the IP in your design.

Create simulation model: None

**Output Directory**

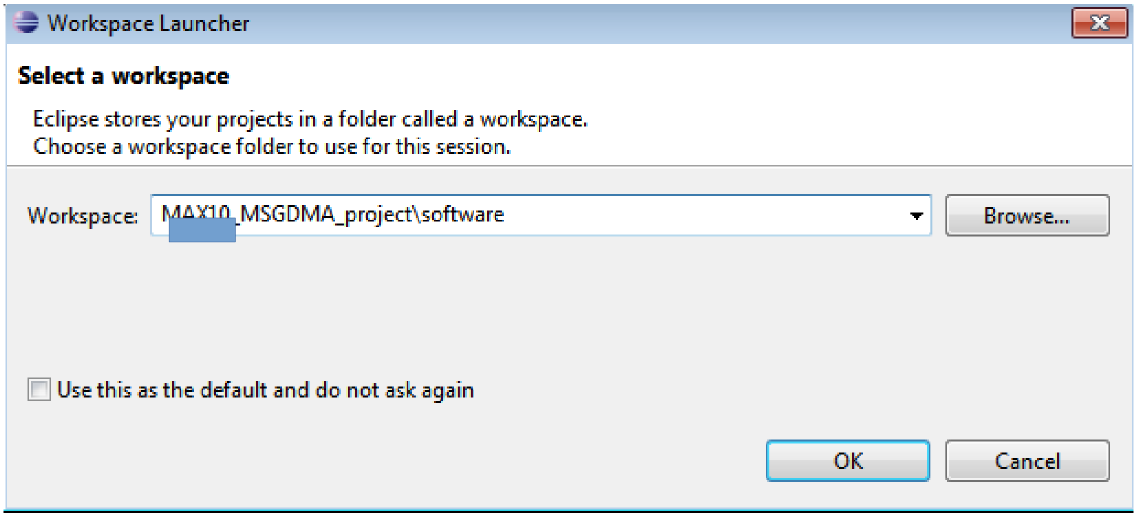
Path: ared/DevProjects/AVnet/prj_altera/FPGA_board/HyperMAX25/SLL_HyperRAM_Nios_Project/hypernios

- In the Synthesis section, set the **Create HDL design files for synthesis** field to **Verilog**.
- In the Simulation section, set the **Create simulation model** field to **None**.
- Then click on the [ Generate ] button.
- You may see a Save System window. Click the [ Close ] button to close the save window.
- Generating the .qsys project updates the .SOPC file which will be used by the Nios II Software Build Tools (SBT) environment.
- Click the [ Close ] button to close the generate window.
- You may want to close the Qsys window.

# 6. Preparing the firmware

## 6.1 Open the NIOS II Software Built Tools for Eclipse

- In Quartus Prime, go to the menu bar and select **Tools → NIOS II Software Built Tools for Eclipse**.



Workspace Launcher

**Select a workspace**

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: MAX10_MSGDMA_project\software

□ Use this as the default and do not ask again

- Click the [Browse…] button. A new file selector window will open. In this tutorial we are going to select the **software** folder located inside the project folder as the workspace and then click the [ OK ] button.

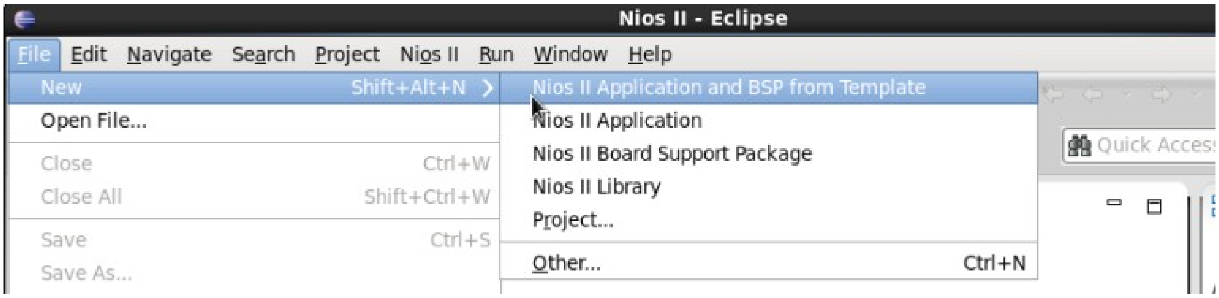
- Be sure to leave the **[ ] Use this as the default** field unticked.

- Click the [ OK ] button.

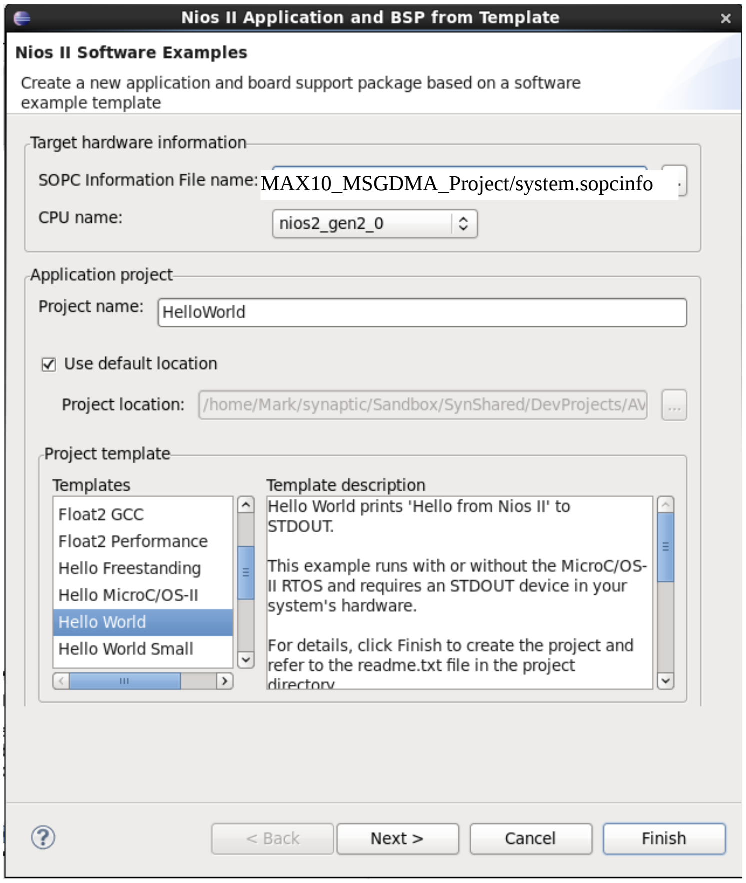
## 6.2 Create a simple application and BSP

The software folder in the reference project is empty. This is because problems can be experienced when moving the Eclipse Workshop folder between Windows and Linux Systems. We need to create a Nios II application, and Nios II board support package for that Nios II application:

- In the Eclipse window, go the menu bar and select:
  **File → New → NIOS II Application and BSP from Templat**e



- A new window will pop up: (most of the fields below will initially be empty)



Nios II Application and BSP from Template

**Nios II Software Examples**

Create a new application and board support package based on a software example template

Target hardware information

SOPC Information File name: MAX10_MSGDMA_Project/system.sopcinfo

CPU name: nios2_gen2_0

Application project

Project name: HelloWorld

☑ Use default location

Project location: /home/Mark/synaptic/Sandbox/SynShared/DevProjects/AV

Project template

Templates: Float2 GCC, Float2 Performance, Hello Freestanding, Hello MicroC/OS-II, Hello World (selected), Hello World Small

Template description:
Hello World prints 'Hello from Nios II' to STDOUT.

This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware.

For details, click Finish to create the project and refer to the readme.txt file in the project directory

- In the **Target hardware information**, click on the […] button
- A file browser window will open. Locate and select the **system.sopcinfo** file generated by Qsys and stored in the project directory. Click [Open].
- It may take around 30 seconds for the Eclipse application to parse the .sopcinfo file.
- Select a **Project name**. In this example, we are using **HelloWorld** as the project name.
- Ensure that: **[x] Use default location** is **ticked**.
- We now need to select a template from the **Project Template** list.
  In this example, select the **Hello World** template.
- Press the [ **Finish ]** button to complete the current step.

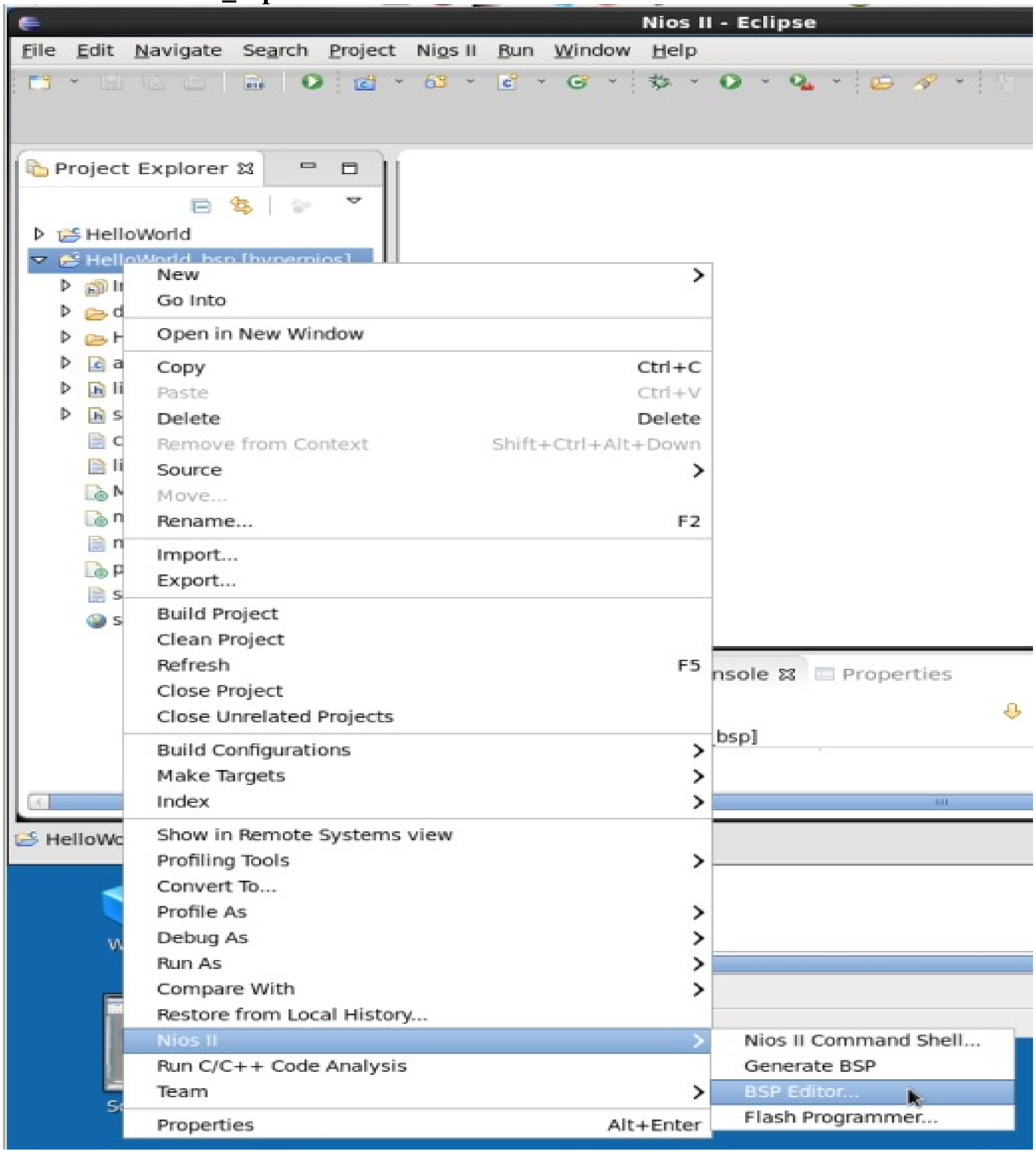
The Nios II SBT will now generate:

- a **HelloWorld** application folder that contains the **hello_world.c** file. We will replace that **hello_world.c** file with a custom program that tests the HyperRAM device later in this tutorial.
- a **HelloWorld_bsp** folder that contains the Nios II Board Support Package (BSP) hardware abstraction layer (HAL).

## 6.3 Configure the Board Support Package (BSP)

The Nios II BSP must be configured before we can compile the source code.

- In the Project Explorer tab, right click on:
  **HelloWorld_bsp → Nios II -> BSP Editor...**



In the **Main Tab** of the BSP editor, in the panel on the left hand side, select:

Settings → Common

- Set the **sys_clk_timer** field to **none**
  This is used to generate a recurring system clock interrupt for the hardware abstraction layer.

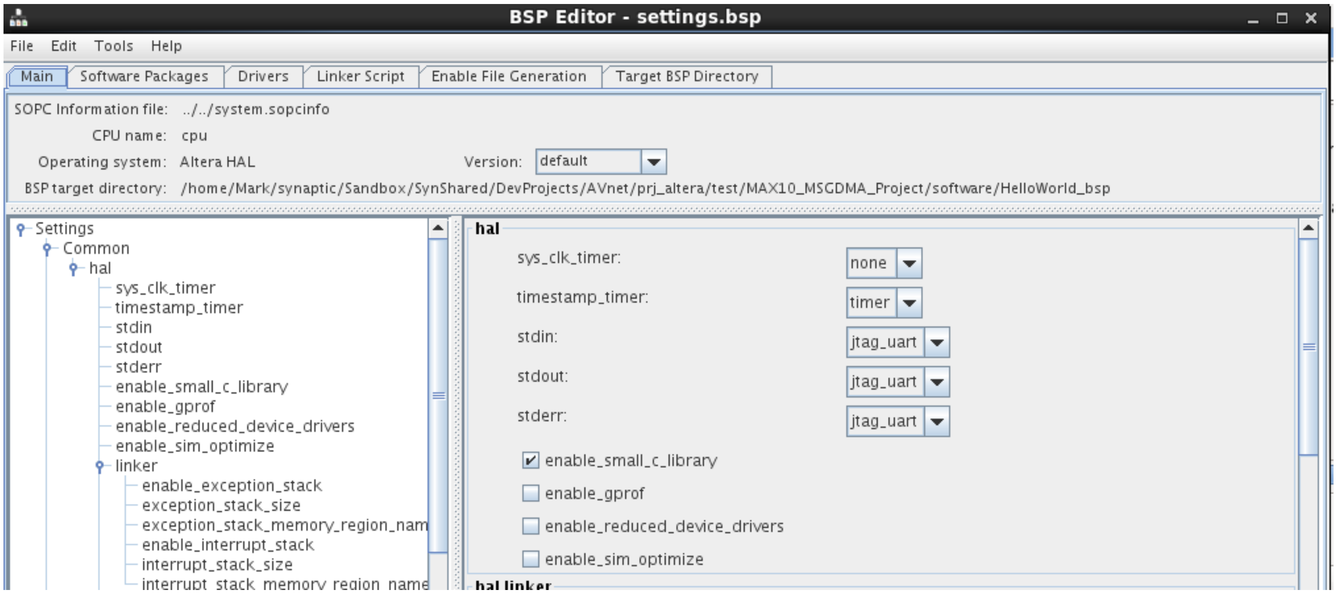
- Set the **timestamp_timer** field to **timer_0**
  This field is used to enable the hardware abstraction layer to perform fine precision timing.

- The Newlib ANSI C standard library can be configured as small or normal

- Generally, when mapping code and data to on-chip memory:
  - **Tick** the **[x] Enable small C library** field to reduce the size of the executable code generated by the hardware abstraction layer (HAL). Ticking this option also reduces the functionality and performance of the HAL. Please note that the inbuilt **memset()** and **memcpy()** routines will be very slow.

- Generally, when mapping code and data to HyperRAM and/or HyperFlash:
  - **Untick** the **[ ] Enable small C library** field to increase the functionality and performance of the executable code generated by the hardware abstraction layer (HAL). The inbuilt **memset()** and **memcpy()** routines will achieve good performance. However, the executable code will be considerably larger.

- It is important to **Tick** the **[x] Enable small C library** for this specific tutorial.



BSP Editor - settings.bsp

SOPC Information file: ../../system.sopcinfo
CPU name: cpu
Operating system: Altera HAL
Version: default
BSP target directory: /home/Mark/synaptic/Sandbox/SynShared/DevProjects/AVnet/prj_altera/test/MAX10_MSGDMA_Project/software/HelloWorld_bsp

hal

sys_clk_timer: none
timestamp_timer: timer
stdin: jtag_uart
stdout: jtag_uart
stderr: jtag_uart

☑ enable_small_c_library
☐ enable_gprof
☐ enable_reduced_device_drivers
☐ enable_sim_optimize

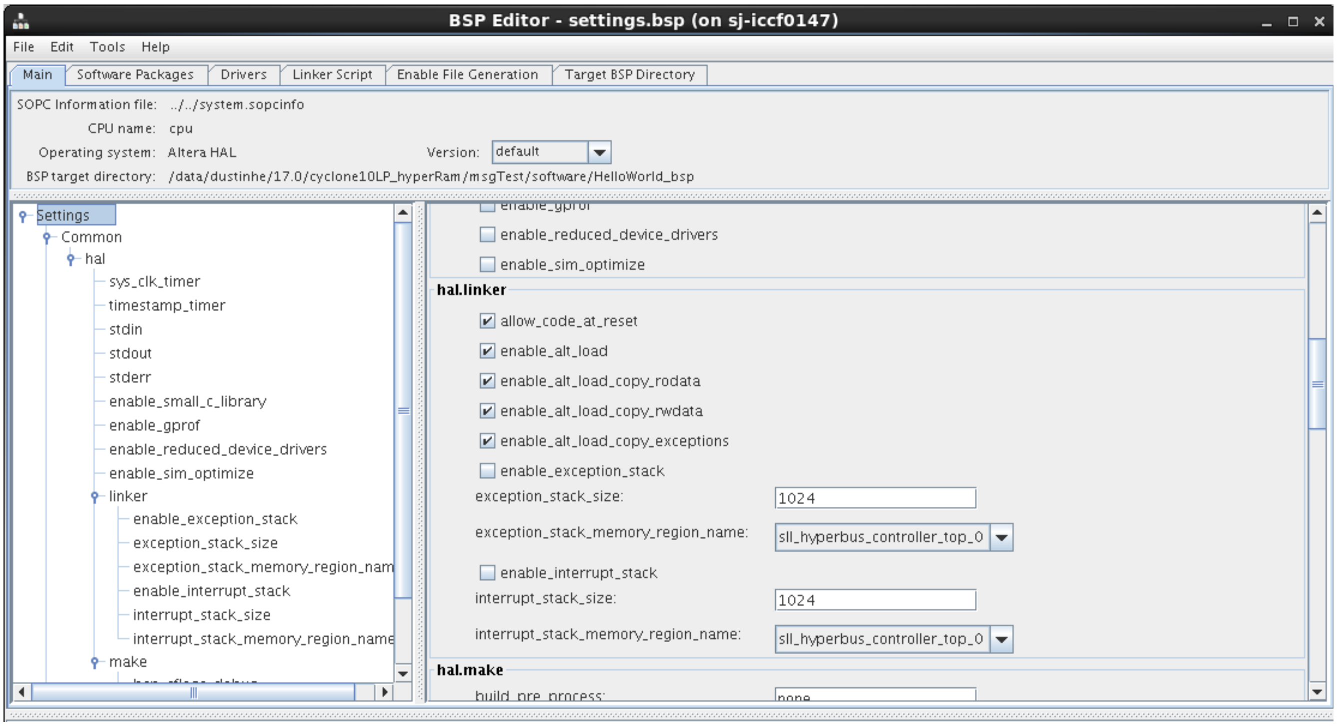
hal.linker

In the **Main Tab** of the BSP editor, in the panel on the left hand side, select:

Settings → Advanced → **hal.linker**

For the purpose of this tutorial, the following configuration will generally work:

- Tick **[x] allow_code_at_reset**
- Tick **[x] enable_alt_load**
- Tick **[x] enable_alt_load_copy_rodata**
- Tick **[x] enable_alt_load_copy_rwdata**
- Tick **[x] enable_alt_load_copy_exception**
- UnTick **[ ] enable_exception_stack**



SOPC Information file: ../../system.sopcinfo
CPU name: cpu
Operating system: Altera HAL Version: default
BSP target directory: /data/dustinhe/17.0/cyclone10LP_hyperRam/msgTest/software/HelloWorld_bsp

□ enable_gprof
□ enable_reduced_device_drivers
□ enable_sim_optimize

**hal.linker**

☑ allow_code_at_reset
☑ enable_alt_load
☑ enable_alt_load_copy_rodata
☑ enable_alt_load_copy_rwdata
☑ enable_alt_load_copy_exceptions
□ enable_exception_stack
exception_stack_size: 1024
exception_stack_memory_region_name: sll_hyperbus_controller_top_0
□ enable_interrupt_stack
interrupt_stack_size: 1024
interrupt_stack_memory_region_name: sll_hyperbus_controller_top_0

**hal.make**

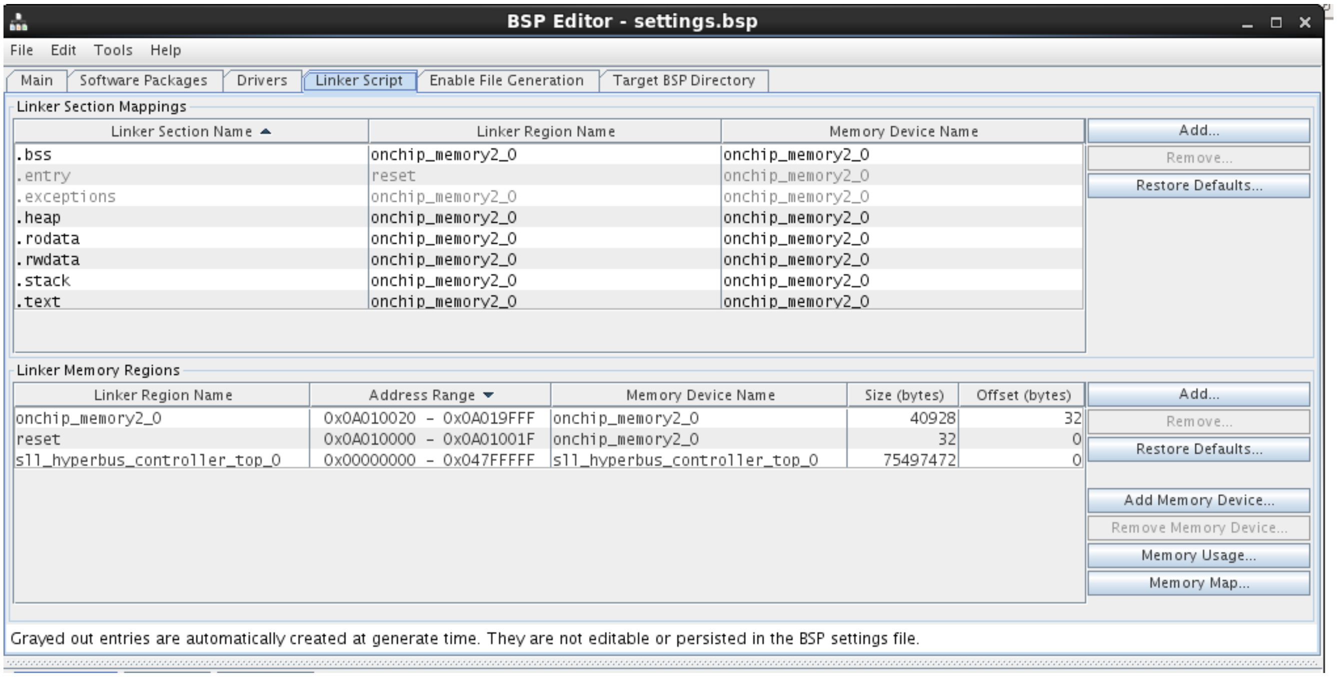
build_pre_process: none

However, this specific configuration may **not** be the best configuration for your project's needs.

Please refer to Altera's documentation for detailed information on how to setup the hal.linker fields:

**Generic Nios II Booting Methods User Guide, UG-20001, 2016.05.24**
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/niosii_generic_booting_methods.pdf

Select the **Linker Script Tab** of the BSP editor.



Linker Section Mappings

| Linker Section Name ▲ | Linker Region Name | Memory Device Name |
|---|---|---|
| .bss | onchip_memory2_0 | onchip_memory2_0 |
| .entry | reset | onchip_memory2_0 |
| .exceptions | onchip_memory2_0 | onchip_memory2_0 |
| .heap | onchip_memory2_0 | onchip_memory2_0 |
| .rodata | onchip_memory2_0 | onchip_memory2_0 |
| .rwdata | onchip_memory2_0 | onchip_memory2_0 |
| .stack | onchip_memory2_0 | onchip_memory2_0 |
| .text | onchip_memory2_0 | onchip_memory2_0 |

Linker Memory Regions

| Linker Region Name | Address Range ▼ | Memory Device Name | Size (bytes) | Offset (bytes) |
|---|---|---|---|---|
| onchip_memory2_0 | 0x0A010020 - 0x0A019FFF | onchip_memory2_0 | 40928 | 32 |
| reset | 0x0A010000 - 0x0A01001F | onchip_memory2_0 | 32 | 0 |
| sll_hyperbus_controller_top_0 | 0x00000000 - 0x047FFFFF | sll_hyperbus_controller_top_0 | 75497472 | 0 |

Grayed out entries are automatically created at generate time. They are not editable or persisted in the BSP settings file.

For this tutorial example, we are going to:

- Map the reset vector **(.reset)** to the onchip memory (*onchip_memory2_0*) .
  *This is generated by Qsys and depends on the location of the Nios II reset vector.*

- Map the exception vector **(.exceptions)** to the onchip memory (*onchip_memory2_0*).
  *This is generated by Qsys and depends on the location of the Nios II exception vector.*

- Map the instruction code (***.text***) in the onchip memory (*onchip_memory2_0*)

- Map all other data regions (***.bss***, ***.heap***, ***.rodata***, ***.rwdata***, ***.stack***) to the onchip memory (*onchip_memory2_0*)

This will map all memory regions generated by the GCC tools to the on-chip SRAM. Also see:
**Nios II Gen2 Software Developer's Handbook**, NII5V2Gen2, 2017.05.08
Section 5, Nios II Software Build Tools
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2sw_nii5v2gen2.pdf

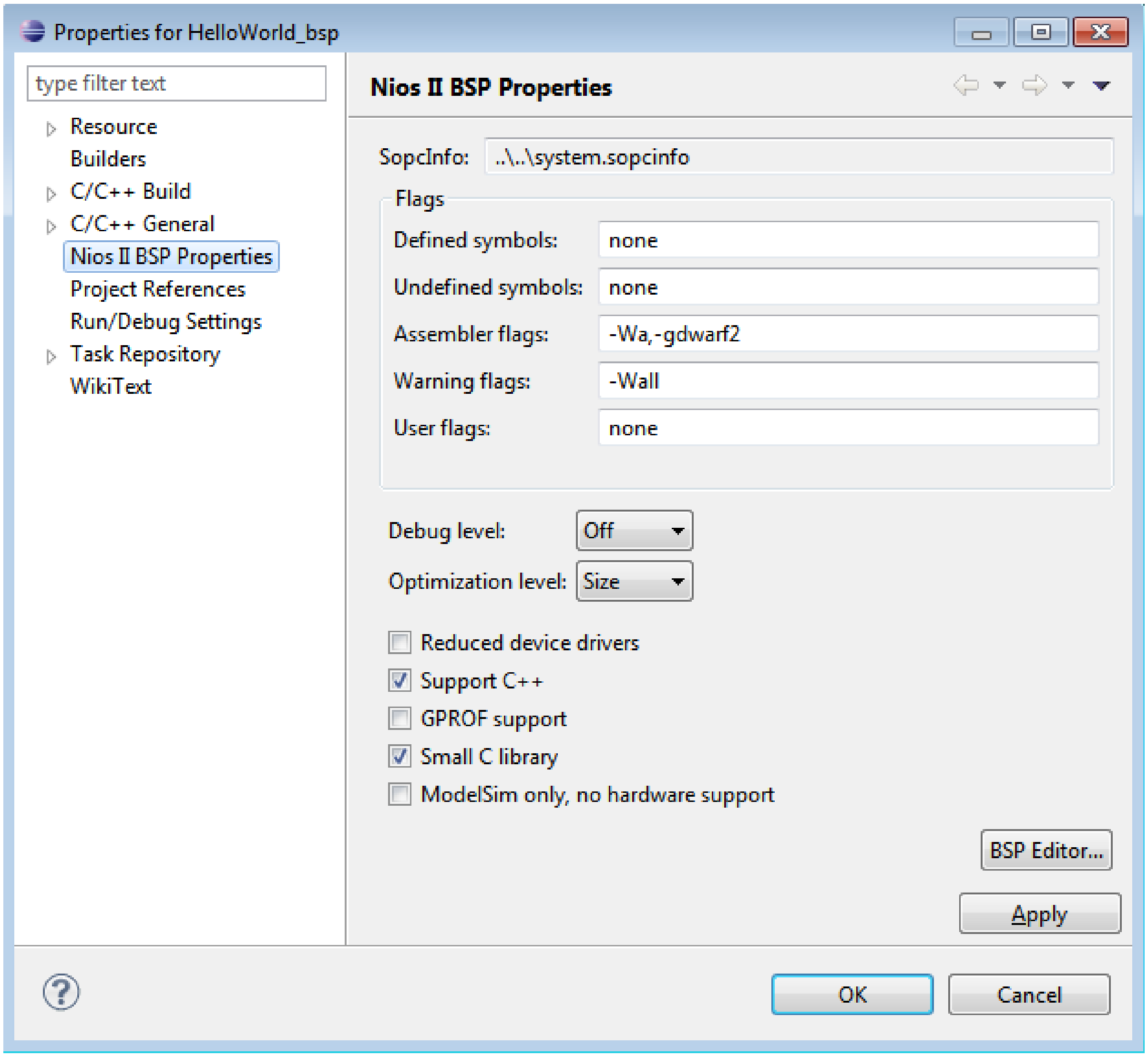
Now:

- Click on the [ Exit ] button on the bottom right hand corner of the BSP Editor window.

- Then click on the [Yes, Save] button on the Save Changes window to save the BSP settings.

For this tutorial, **we need to reduce the size of the executable code generated** for the Newlib ANSI C standard library:

- Right click on **HelloWorld_bsp**
- A popup window appears. Select **Properties**
- A new window called "Properties for HelloWorld_bsp" opens.
- Left click on **Nios II BSP Properties**
- Set **Optimization Level: SIZE.** ← Compiler will now generate smaller object code
- **Click the [ Apply ] button.**
- **Click the [ OK ] button.**



SopcInfo: ..\..\system.sopcinfo

Flags

Defined symbols: none

Undefined symbols: none

Assembler flags: -Wa,-gdwarf2

Warning flags: -Wall

User flags: none

Debug level: Off

Optimization level: Size

☐ Reduced device drivers
☑ Support C++
☐ GPROF support
☑ Small C library
☐ ModelSim only, no hardware support

The standard ANSI C functions will now be optimized by gcc for smaller object code size, resulting in more on-chip SRAM being available for the **.stack** and **.heap** sections.

## 6.4 Generate the BSP and clean the project

The software developer must re-generate the BSP every time the Qsys project is regenerated. This ensures that the device drivers and addresses of peripherals are reflected correctly in the hardware abstract library. To (re)generate the BSP:

- Go to the Nios II eclipse window.
- Right click on **HelloWorld_bsp** project then select Nios II then select **Generate BSP**.
- Right click on the **HelloWorld_bsp** project then select **Clean Project** to delete any intermediate files generated by the gcc compiler for this application library.
- Right click on the **HelloWorld** project then select **Clean Project** to delete any intermediate files generated by the gcc compiler for this application folder.

## 6.5 Install the memory bandwidth benchmarking source code

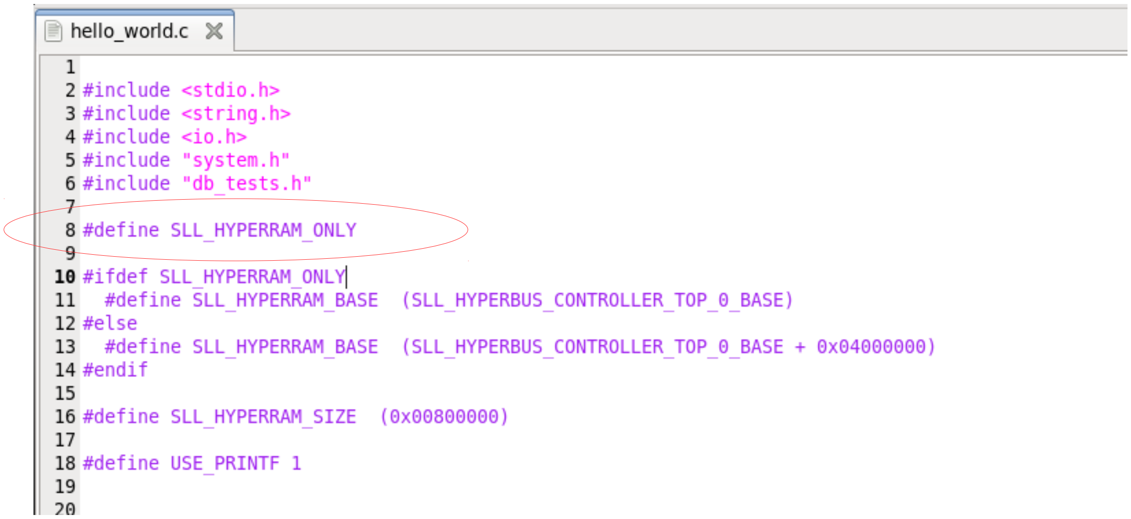
- We now want to replace the original HelloWorld.c source code with software that benchmarks the HyperRAM.  Copy and replace the files located in:
  - **MSGDMA_Project/source/**

  to:
  - **MSGDMA_Project/software/HelloWorld**
- In the project explorer window of Eclipse, right click on the **HelloWorld** folder. Then select **Refresh**.  The new source code files should now be visible within Eclipse.

---------------------------------------- **Important** ----------------------------------------

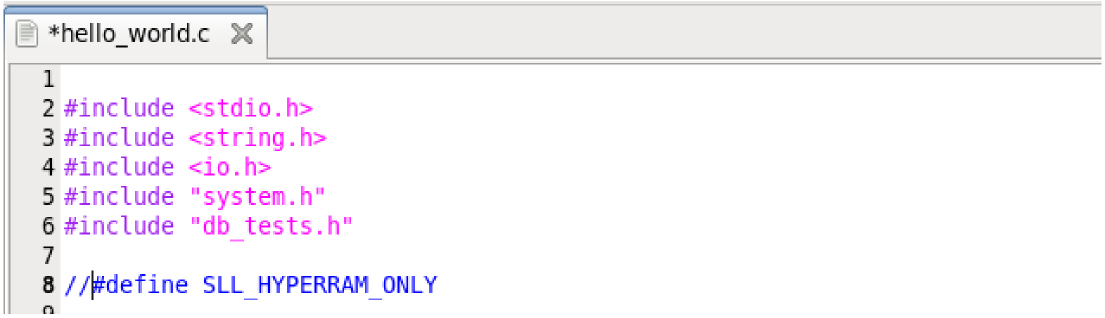
If the **FPGA board type** field of the **"Master Configuration" tab** of the HBMC IP, is set to **Devboards - HyperMAX 10M25/50 (HyperFlash and HyperRAM)**, then please **// COMMENT out line 8** in the new **hello_world.c** file.

From a HyperRAM only configuration:



```
 1
 2 #include <stdio.h>
 3 #include <string.h>
 4 #include <io.h>
 5 #include "system.h"
 6 #include "db_tests.h"
 7
 8 #define SLL_HYPERRAM_ONLY
 9
10 #ifdef SLL_HYPERRAM_ONLY
11   #define SLL_HYPERRAM_BASE  (SLL_HYPERBUS_CONTROLLER_TOP_0_BASE)
12 #else
13   #define SLL_HYPERRAM_BASE  (SLL_HYPERBUS_CONTROLLER_TOP_0_BASE + 0x04000000)
14 #endif
15
16 #define SLL_HYPERRAM_SIZE  (0x00800000)
17
18 #define USE_PRINTF 1
19
20
```

To a HyperFlash and HyperRAM configuration:



```
1
2 #include <stdio.h>
3 #include <string.h>
4 #include <io.h>
5 #include "system.h"
6 #include "db_tests.h"
7
8 //#define SLL_HYPERRAM_ONLY
9
```

## 6.6 Build the Nios II Application

We now want to run the compiler and linker:

- Go to the Nios II eclipse window.

- Go to the menu bar and select:
  **Project** ->**Build All**

- If the project produces warning / error messages, you may need to build the project twice.

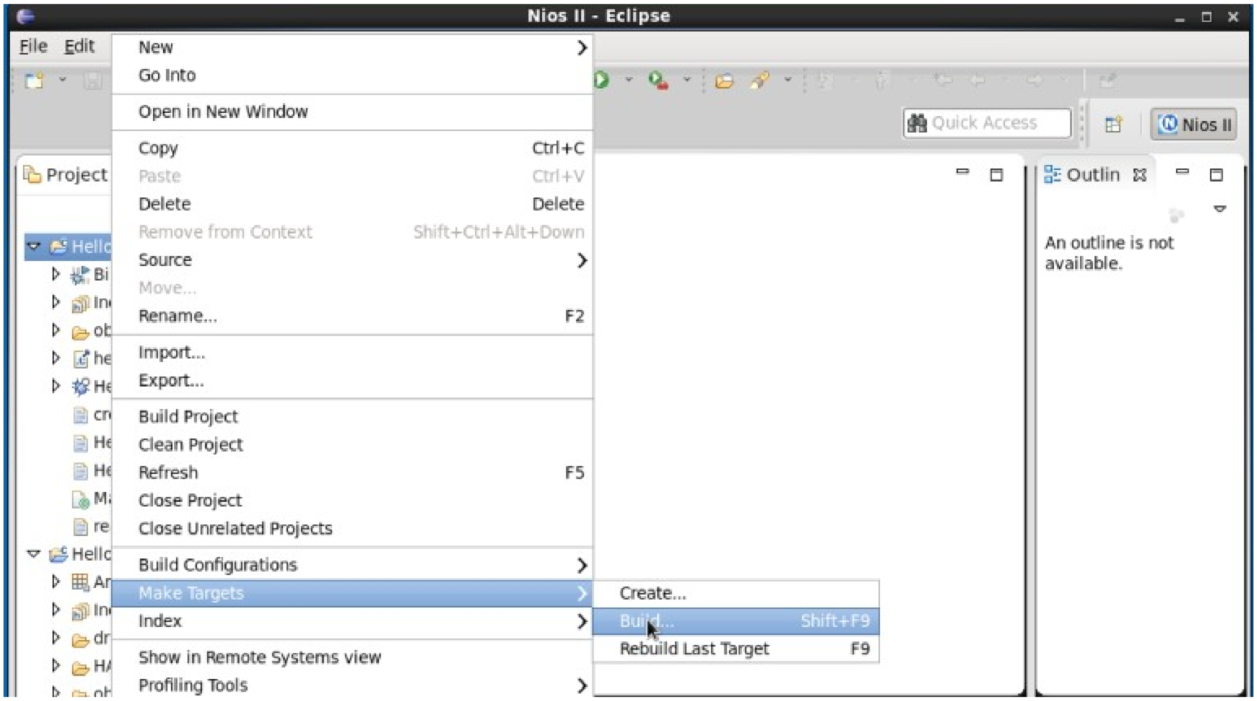
The **HelloWorld** executable firmware (.ELF) is now generated. The .ELF can be downloaded directly into on-chip SRAM using the Nios II Software Development files.

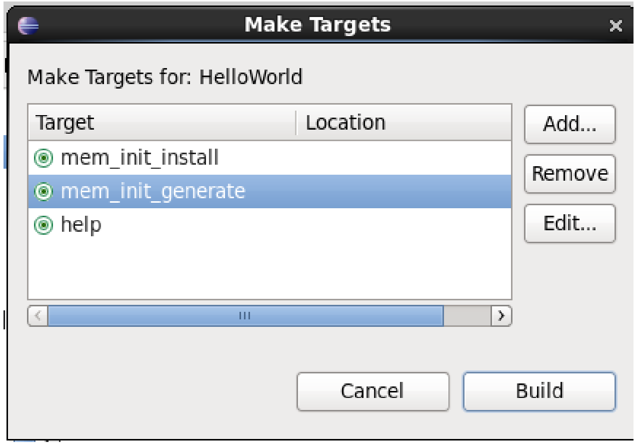
However, the .ELF file itself cannot be embedded directly into the FPGA bitstream file, or programmed directly into the HyperFlash memory. To do that, we need to convert the .ELF file into one or more memory initialization files.

## 6.7 Generate memory initialization files

If we want to embed the firmware into the on-chip SRAM when configuration the FPGA device using a FPGA bitstream, or if we wish to program the HyperFlash memories, we need to generate “memory initialisation” **.hex** files from the **.elf** file.

- In the Project Explorer tab, right click on:
  **HelloWorld** -> **Make Targets** → **Build…**





Make Targets for: HelloWorld

| Target | Location |
|---|---|
| mem_init_install | |
| mem_init_generate | |
| help | |

- A Make Targets window will open.
- Select **mem_init_generate** and click on the **Build** button.
- New hex files will be generated.

  These files will be located in
  Project_dir →
  Software →
  HelloWorld →
  mem_init

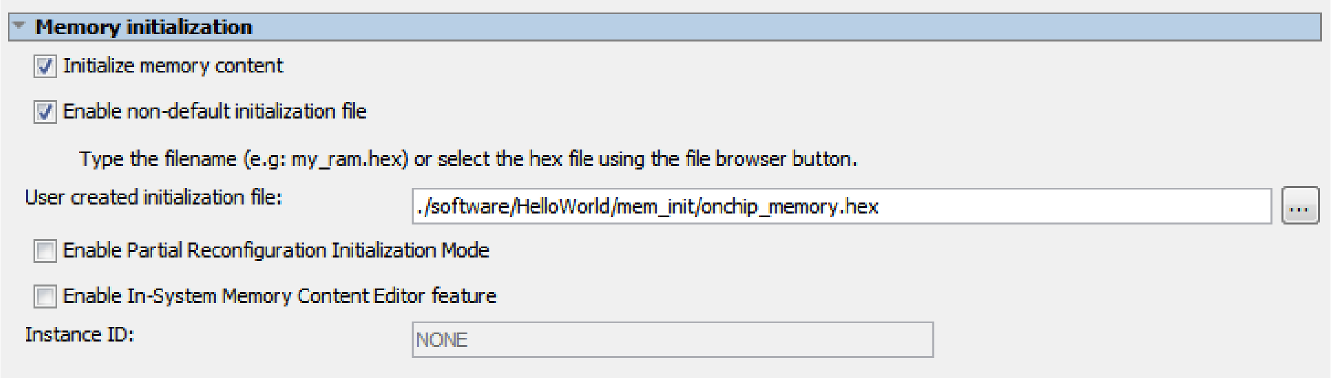
The file **onchip_memory.hex** can be embedded into the FPGA bitstream to initialize the onchip-memory as described in the next section.

# 7. Update the memory initialization field(s) in Qsys

Before synthesizing the design, it is important to ensure that the initialization file for the onchip-memory is **set to the exact path** of the .hex file in the Qsys project. This is because it is possible to create multiple applications within a Nios II Eclipse workspace, each with its own firmware and memory initialisation files.

- In **Qsys**, open the **hypernios.qsys** project.
- Open the parameters for the **On-Chip Memory** module.
- Ensure that the **User created initialization file** points to the **onchip_memory.hex** file generated by NIOS II SBT for Eclipse for the application you want to use. In this tutorial we only have one Nios II application located in the **software/HelloWorld** folder.
- We recommend typing in the filename with a relative path:

**./software/HelloWorld/mem_init/onchip_memory.hex**



Memory initialization

☑ Initialize memory content

☑ Enable non-default initialization file

Type the filename (e.g: my_ram.hex) or select the hex file using the file browser button.

User created initialization file: ./software/HelloWorld/mem_init/onchip_memory.hex

□ Enable Partial Reconfiguration Initialization Mode

□ Enable In-System Memory Content Editor feature

Instance ID: NONE

- Each time you change the Qsys project, you will need to:
  - Save and regenerate the Qsys design
  - Then regenerate the Nios II BSP
  - Then clean the projects and rebuild the (.elf) Nios II Application
  - Then regenerate the (.hex) memory initialization binaries

- As we have now changed the Qsys project, you will need to execute the above steps.

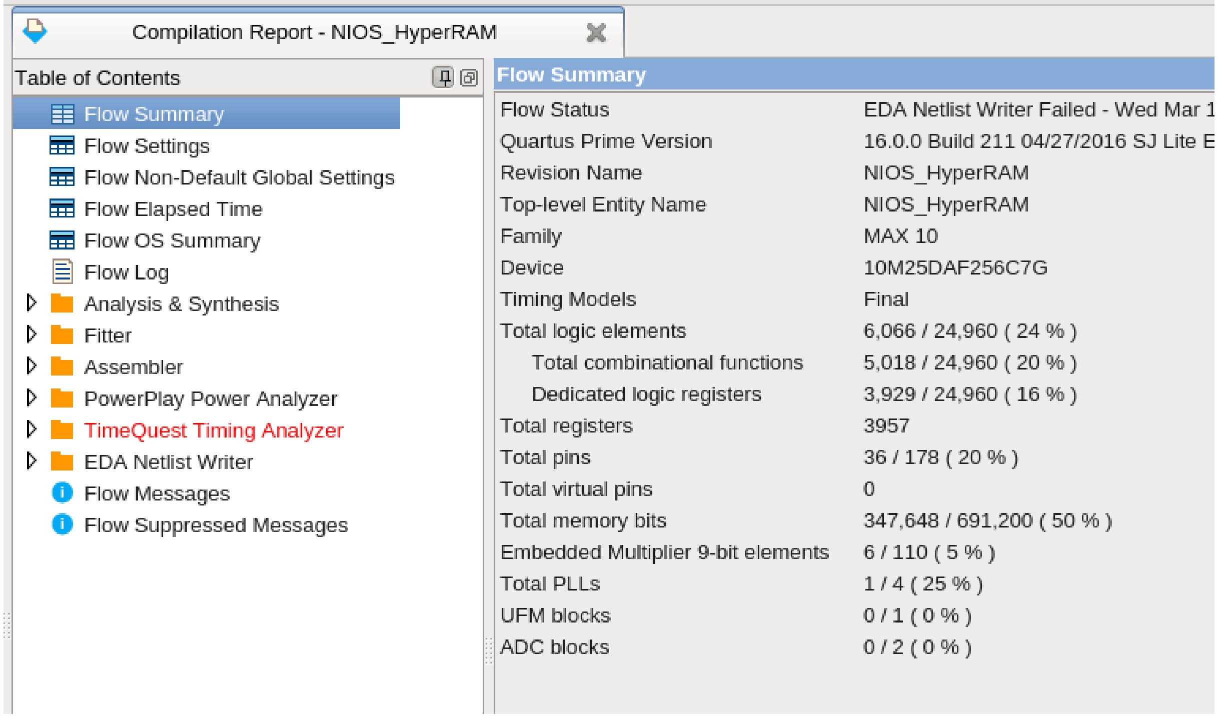
# 8. Synthesize and assemble the Design

- Go to the Quartus Prime window.

- In the menu bar, select:
  **Processing → Start Compilation**

  ***Windows users, please note:*** *If the compilation fails to start you may need to reduce the path length of your project folder. This is because some versions of Windows have a maximum path length of 260 characters which can be exceeded when compiling projects in Quartus Prime.*
  *If you change the path, make sure all .hex files in the Qsys project are set correctly.*

- The assembler step will create the SRAM FPGA Bitstream file (.SOF) with the memory initialisation file (.hex) for the SRAM embedded in that FPGA Bitstream file.



| | |
|---|---|
| Flow Status | EDA Netlist Writer Failed - Wed Mar 1 |
| Quartus Prime Version | 16.0.0 Build 211 04/27/2016 SJ Lite E |
| Revision Name | NIOS_HyperRAM |
| Top-level Entity Name | NIOS_HyperRAM |
| Family | MAX 10 |
| Device | 10M25DAF256C7G |
| Timing Models | Final |
| Total logic elements | 6,066 / 24,960 ( 24 % ) |
| Total combinational functions | 5,018 / 24,960 ( 20 % ) |
| Dedicated logic registers | 3,929 / 24,960 ( 16 % ) |
| Total registers | 3957 |
| Total pins | 36 / 178 ( 20 % ) |
| Total virtual pins | 0 |
| Total memory bits | 347,648 / 691,200 ( 50 % ) |
| Embedded Multiplier 9-bit elements | 6 / 110 ( 5 % ) |
| Total PLLs | 1 / 4 ( 25 % ) |
| UFM blocks | 0 / 1 ( 0 % ) |
| ADC blocks | 0 / 2 ( 0 % ) |

- If at a later time you recompile your Nios II Code, you will need to embed the new .hex files into the FPGA bitstream. The fastest way to do this is:

  - In the menu bar, select:
    **Processing → Update Memory Initialization File**
    This ensures the latest version of the .hex file will be used during the assembly step.

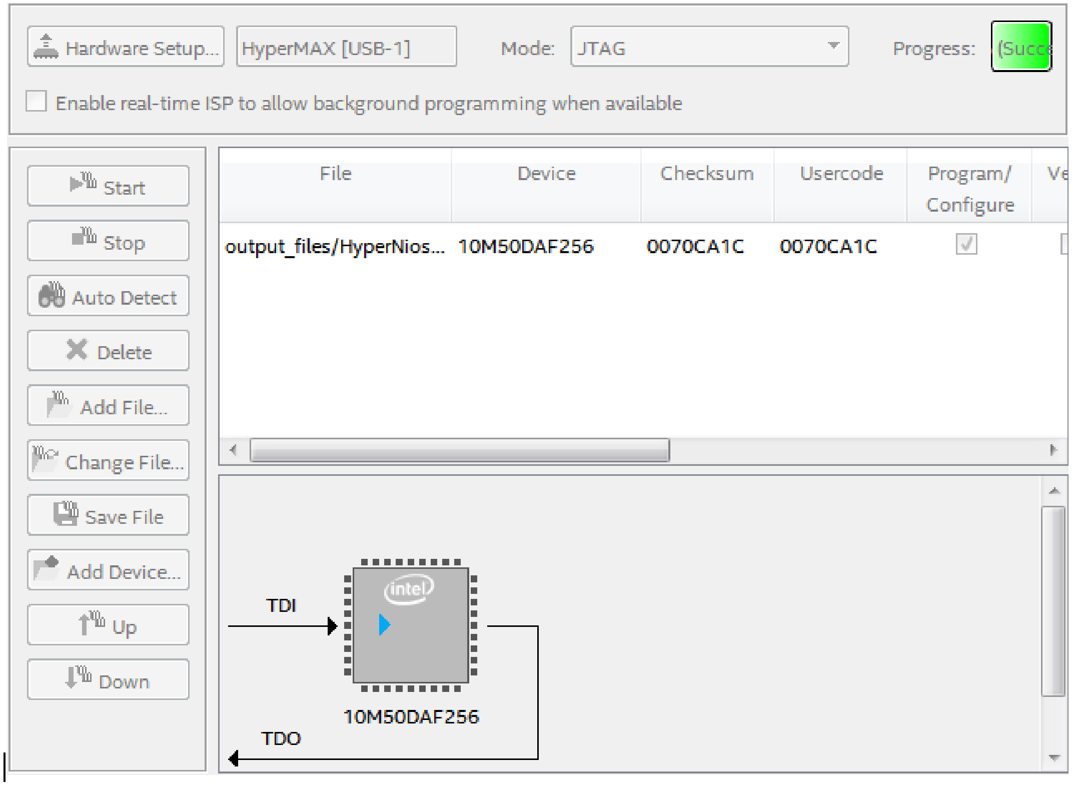
  - In the menu bar, select:
    **Processing → Start → Start Assembler**

# 9. Program the FPGA Bitstream into the FPGA device

- Connect the HyperMAX/Intel C10LP Evaluation kit to the USB port of your computer

- Open the Quartus Prime window

- In the menubar, click on **Tools** then **Programmer** to start the Altera Programmer



Hardware Setup... HyperMAX [USB-1] Mode: JTAG Progress: (Succe

□ Enable real-time ISP to allow background programming when available

| File | Device | Checksum | Usercode | Program/ Configure | Ve |
|---|---|---|---|---|---|
| output_files/HyperNios... | 10M50DAF256 | 0070CA1C | 0070CA1C | ☑ | □ |

- Click on "**Hardware Setup…**". A new window will open.
- Double Click on the **HyperMax** (or **Intel C10LP Evaluation kit**) device, then click the [ Close ] button.

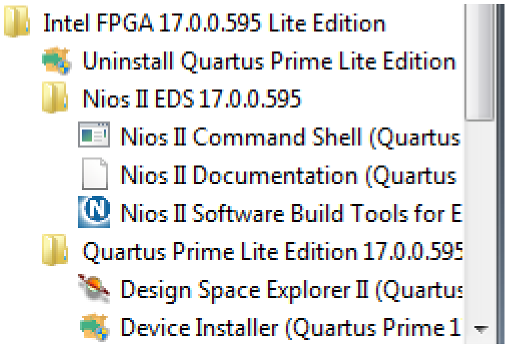
- If the **HyperNios_MSGDMA_time_limited.sof** is not already selected:
  - Click "**Add File...**" in the programmer window.
  - Go to the **output_files** folder
  - Double click on **HyperNios_MSGDMA_time_limited.sof**

- Click the [ Start ] button and the FPGA bitstream will be programmed into the SRAM configuration memory of the FPGA device.

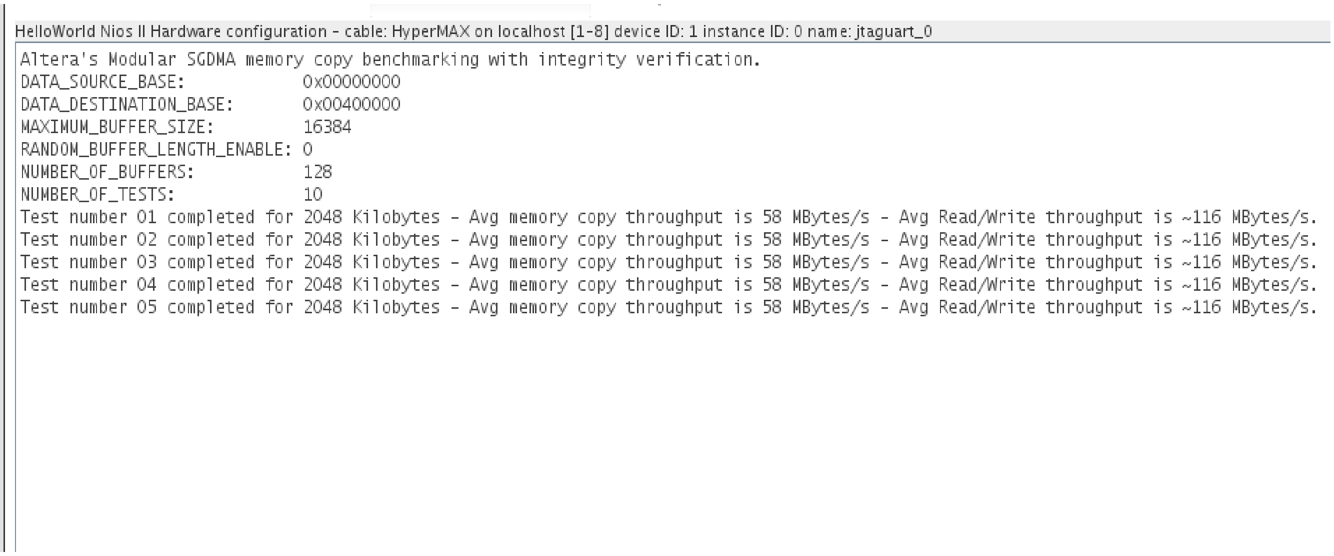
- A window called "OpenCore Plus Status" should open.

# 10. Run the nios2-terminal application

- In Linux: Open a Linux command shell / terminal
- In Windows: Run the **Nios II Command Shell** application from the Windows start menu.


Intel FPGA 17.0.0.595 Lite Edition
Uninstall Quartus Prime Lite Edition
Nios II EDS 17.0.0.595
Nios II Command Shell (Quartus
Nios II Documentation (Quartus
Nios II Software Build Tools for E
Quartus Prime Lite Edition 17.0.0.595
Design Space Explorer II (Quartus
Device Installer (Quartus Prime 1

- Run the **nios2-terminal** command from the terminal.
- Messages similar to the one below should be displayed in the command shell.


HelloWorld Nios II Hardware configuration - cable: HyperMAX on localhost [1-8] device ID: 1 instance ID: 0 name: jtaguart_0

```
Altera's Modular SGDMA memory copy benchmarking with integrity verification.
DATA_SOURCE_BASE:            0x00000000
DATA_DESTINATION_BASE:       0x00400000
MAXIMUM_BUFFER_SIZE:         16384
RANDOM_BUFFER_LENGTH_ENABLE: 0
NUMBER_OF_BUFFERS:           128
NUMBER_OF_TESTS:             10
Test number 01 completed for 2048 Kilobytes - Avg memory copy throughput is 58 MBytes/s - Avg Read/Write throughput is ~116 MBytes/s.
Test number 02 completed for 2048 Kilobytes - Avg memory copy throughput is 58 MBytes/s - Avg Read/Write throughput is ~116 MBytes/s.
Test number 03 completed for 2048 Kilobytes - Avg memory copy throughput is 58 MBytes/s - Avg Read/Write throughput is ~116 MBytes/s.
Test number 04 completed for 2048 Kilobytes - Avg memory copy throughput is 58 MBytes/s - Avg Read/Write throughput is ~116 MBytes/s.
Test number 05 completed for 2048 Kilobytes - Avg memory copy throughput is 58 MBytes/s - Avg Read/Write throughput is ~116 MBytes/s.
```