



# Synaptic Labs'

## Hyperbus Controller Full Featured Edition Upgrade

### Introduction

This short document outlines the basic steps for upgrading the tutorials in order to use the Full Featured Edition of S/Labs HBMC IP.

### Step 1 Remove HBMC Qsys Component from the project IP Folder or other IP installation location

1. Remove any copy of S/Labs' HBMC basic (open-core) edition IP bundle from your project or IP installation folder.

### 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/](http://opencore_license_001.synaptic-labs.com/)

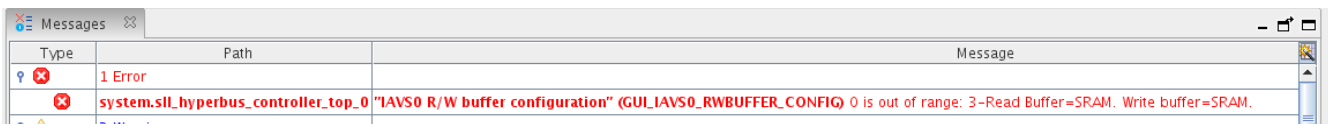
2. Synaptic Labs offers two Installation Guides that:
  - a. Begin by preparing you to enroll to receive a Full Featured Trial Edition license
  - b. Guide you on how to install the license file you will receive after enrolment
  - c. Guide you on how to install the Qsys components that you will receive 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. Other Qsys component installation methods are described in the above mentioned installation Guides.
2. Contact S/Labs for the latest version of Synaptic Labs' HBMC IP
3. Copy S/Labs' HBMC Full Featured edition IP into the **project/ip** folder or IP installation folder.
4. Also, copy sll\_memory\_region\_bridge IP into the **project/ip** folder or IP installation folder

### Step 4: Open Qsys and update S/Labs' HBMC configuration

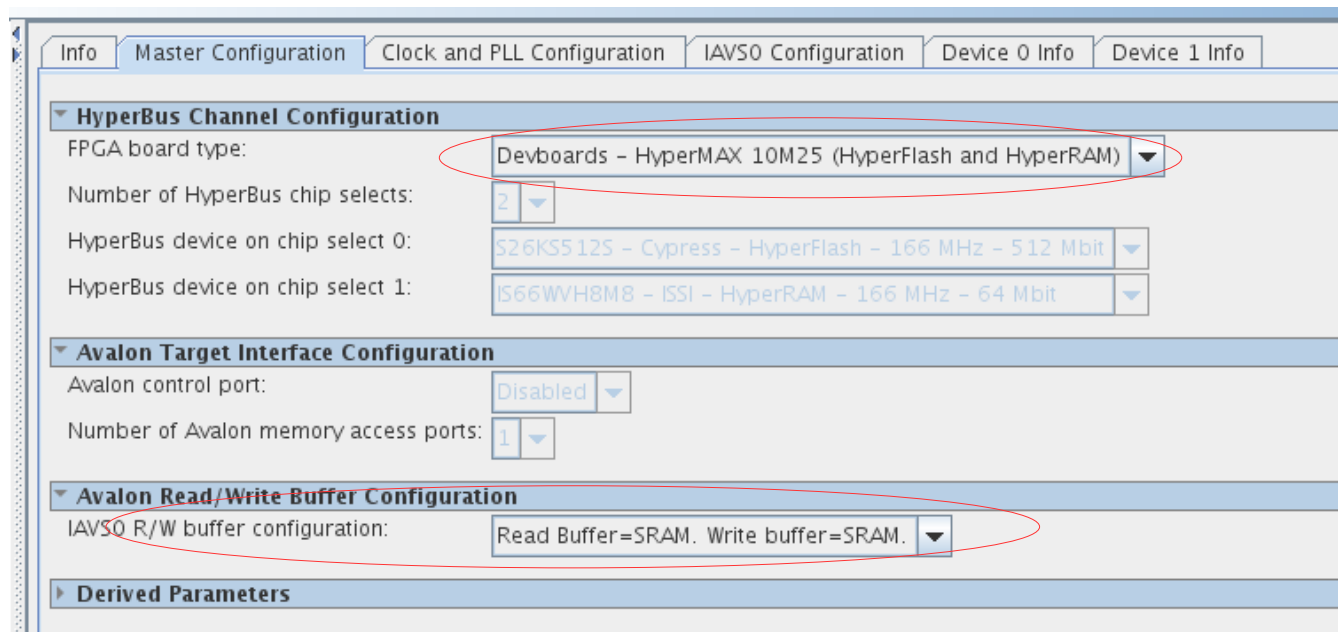
Ignore any errors and warning related to S/Labs' HBMC IP (if any).



Double Click on S/Labs HBMC module

In the Master configuration Tab, change

- **FPGA board type** to [Devboards – HyperMAX 10M25 \(HyperFlash and HyperRAM\)](#) or [Devboards – HyperMAX 10M50 \(HyperFlash and HyperRAM\)](#)
- **IAVS R/W buffer configuration** to **Read Buffer=SRAM, Write Buffer = SRAM**



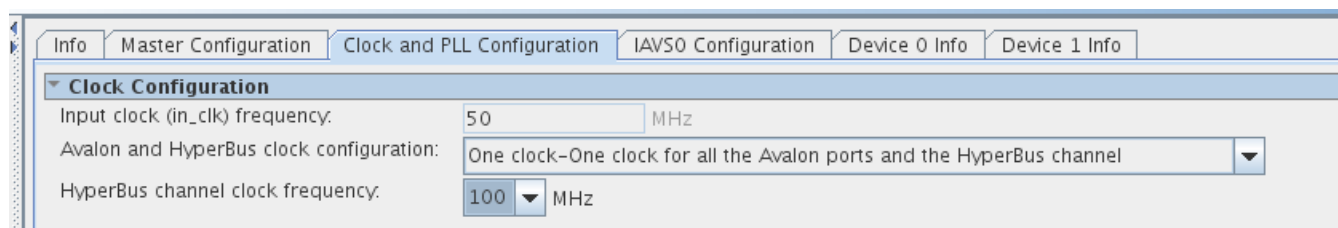
## Increasing throughput on the Hyperbus memory devices

Two methods are described for increasing the throughput on the Hyperbus memory devices. Method 2 offers higher throughput than Method 1.

### Method 1

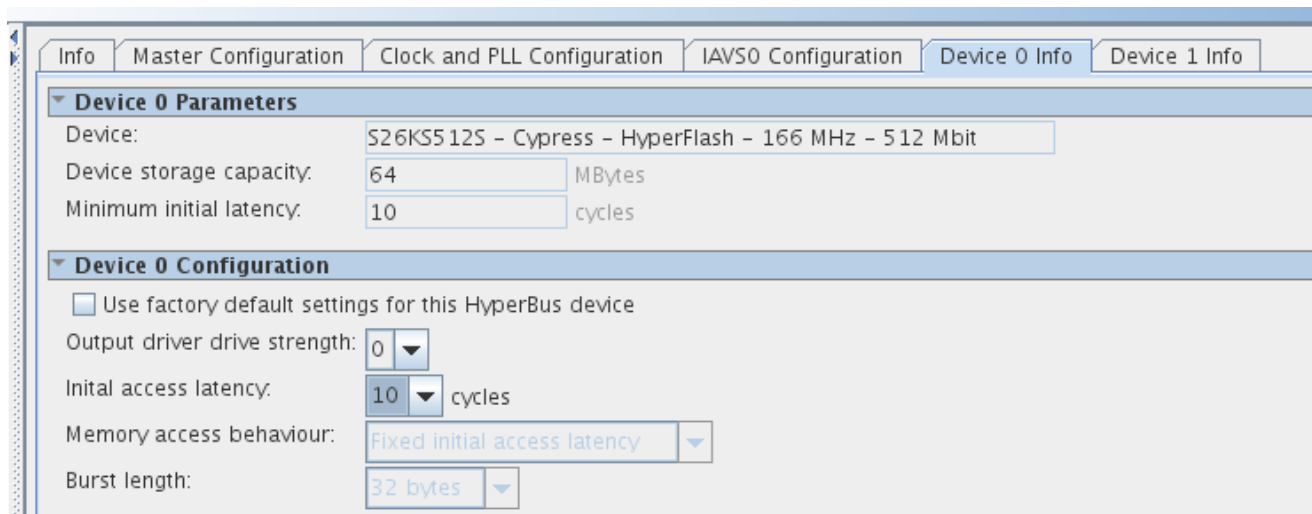
In the Clock and PLL configuration Tab, set

- The **Avalon and HyperBus clock configuration** field is set to **One clock**
- The **HyperBus channel clock frequency** field is set to **100 MHz** (*this is maximum frequency that is supported by the Nios II subsystem. For small designs, higher frequencies might be achievable*).



In the Device 0 Info Tab, change

- Untick the **Use factory default settings** [ ]
- Set the **Initial access latency** to 10 (the initial access latency has to be equal or higher than the value in the minimum initial latency field)
- These are the timings related to HyperFlash memory device



In the Device 1 Info Tab, change

- Untick the **Use factory default settings** [ ]
- Set the **Initial access latency** to 4 (the initial access latency has to be equal or higher than the value in the minimum initial latency field)
- These are the timings related to HyperRAM memory device

The screenshot shows the 'Device 1 Info' tab. Under 'Device 1 Parameters', the 'Device' is 'IS66WVH8M8 - ISSI - HyperRAM - 166 MHz - 64 Mbit', 'Device storage capacity' is '8 MBytes', and 'Minimum initial latency' is '4 cycles'. Under 'Device 1 Configuration', the checkbox 'Use factory default settings for this HyperBus device' is unchecked. 'Output driver drive strength' is '0', 'Initial access latency' is '4 cycles', 'Memory access behaviour' is 'Fixed initial access latency', and 'Burst length' is '32 bytes'.

The Hyperbus memory devices are now running with lower initial latency and offer higher throughput.

## Method 2

In the Clock and PLL configuration Tab, change

- 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** *this is maximum frequency that is supported by the Nios II subsystem. For small designs, higher frequencies might be achievable).*

The screenshot shows the 'Clock and PLL Configuration' tab. Under 'Clock Configuration', 'Input clock (in\_clk) frequency' is '50 MHz', 'Avalon and HyperBus clock configuration' is 'Two clocks-One clock for all the Avalon ports. One clock for the HyperBus channel', 'HyperBus channel clock frequency' is '150 MHz', and 'Shared Avalon clock frequency' is '100 MHz'. Under 'Selected Configuration - Clocks', 'Input clock (in\_clk) frequency' is '50 MHz', 'HyperBus channel clock frequency' is '150 MHz', 'Avalon output clock (av\_out\_clk) frequency' is '100 MHz', 'Avalon IAVS0 port clock frequency' is '100 MHz', and 'Avalon IAVS0 port clock crossing' is 'Low-area high-performance clock-crossing logic enabled'.

In the Device 0 Info Tab,

- Tick the **Use factory default settings** [\[x\]](#)

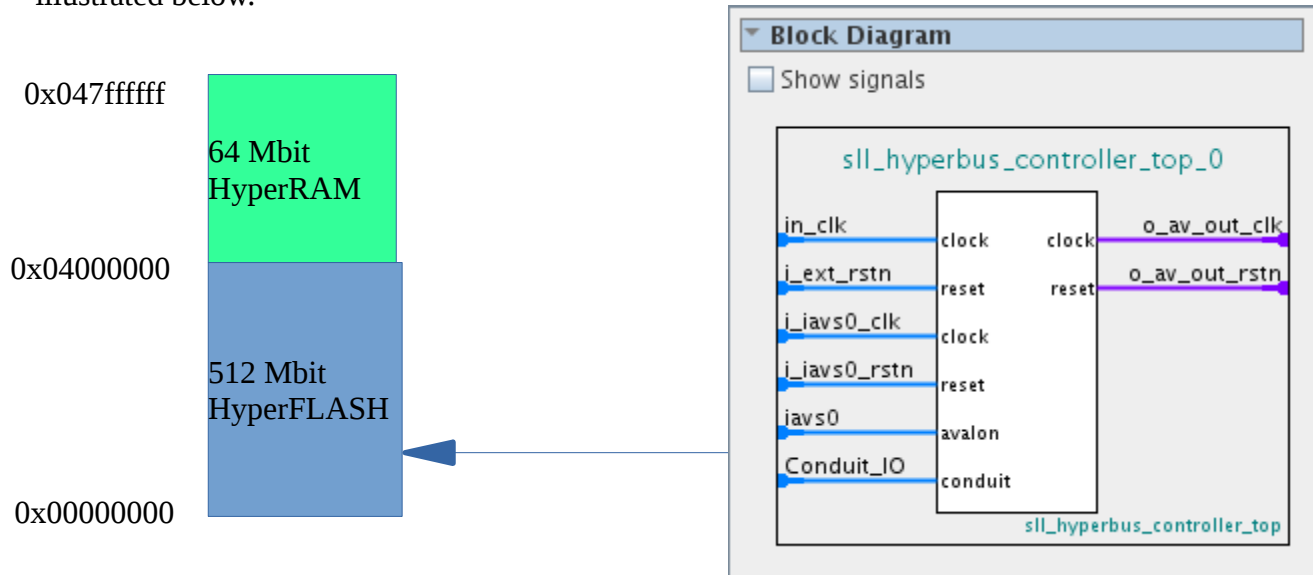
In the Device 1 Info Tab, change

- Tick the **Use factory default settings** [\[x\]](#)

The Hyperbus memory devices are now running at a higher frequency and offer higher throughput.

## Step 5: Supporting HyperFlash and HyperRAM memory regions in Eclipse

For now we will assume that you have selected the Devboards – HyperMAX 10M25 board with HyperFlash and HyperRAM enabled (as illustrated above). In this case, the 512 Megabit HyperFlash memory associated with chip select 0 will be mapped to the base (0x0) of the address space of the Avalon ingress address port (iavs0). The 64 Megabit HyperRAM memory associated with chip select 1 will be mapped to the address (0x04000000) immediately above the HyperFlash memory as visually illustrated below.



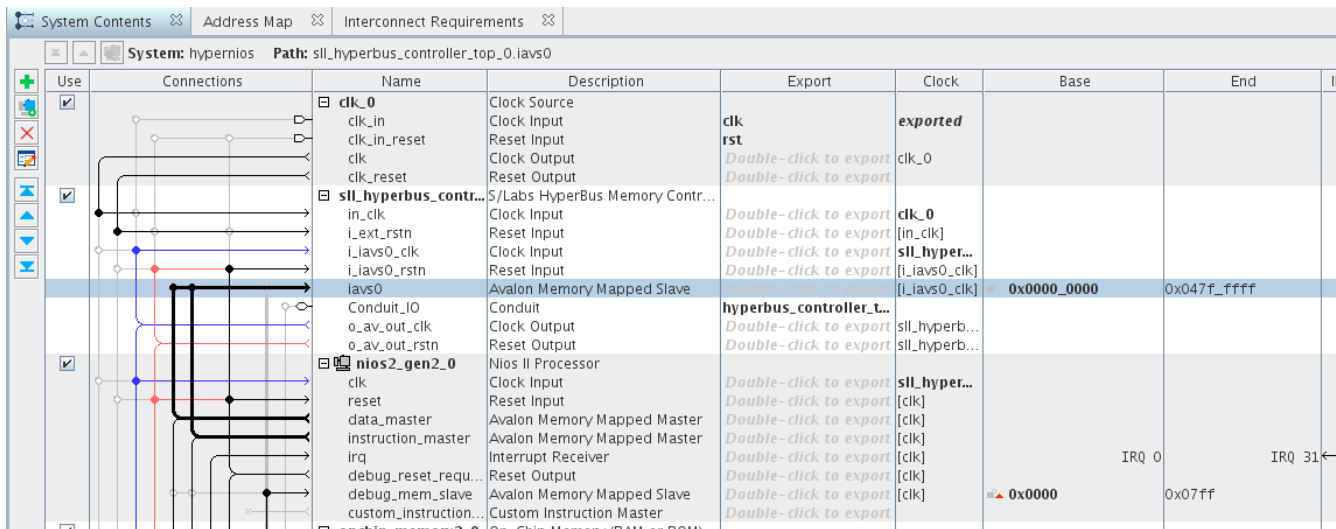
However, Eclipse will generate a single memory region in the linker section. Software developers find this difficult to work with and thus it would be better to split the memory region into respective HyperFlash and HyperRAM memory regions. This can be done in 2 ways :

- either manually modify the Nios II BSP Linker Memory Regions so that we can map two linker regions (one for HyperFlash, and one for HyperRAM) to this HyperBus Memory Controller IP instance. More info is available in document [SynapticLabs-HBMC\\_Splitting\\_memory\\_bsp\\_editor.pdf](#)
- Or add S/Labs memory regions mapper component (`sll_memory_region_mapper`) freely available with S/Labs HBMC Full Featured edition. This is further described in the steps below.

## Step 6: Adding S/Labs Memory region mapper IP to support easy integration of the HyperRAM and HyperFlash memory regions in Eclipse ( optional)

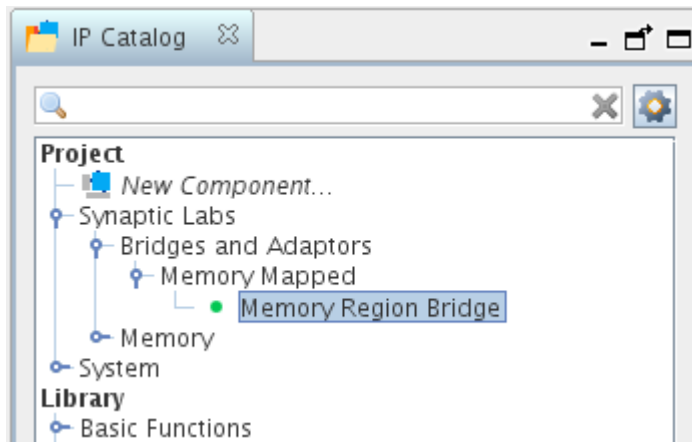
Initial configuration maps the Nios II instruction and data master directly to S/Labs HBMC avalon slave.

- HyperFlash memory region is from 0x00000000 to 0x03FFFFFF
- HyperRAM memory region is from 0x04000000 to 0x047FFFFFFF (or 0x04FFFFFFF depending on the HyperRAM device size)



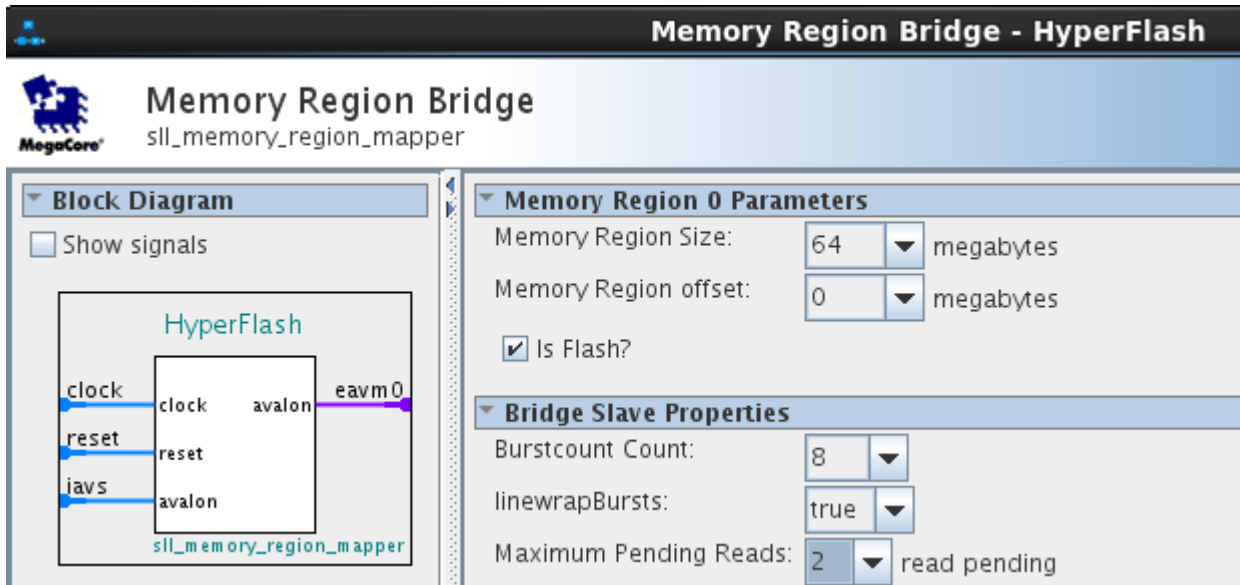
## Using S/Labs Memory Region bridge

S/Labs Memory region bridge is located in the IP catalog library.



## Adding HyperFlash Memory Region bridge

Open S/Labs Memory region bridge located in the IP catalog library



In the Memory Region 0 Parameters

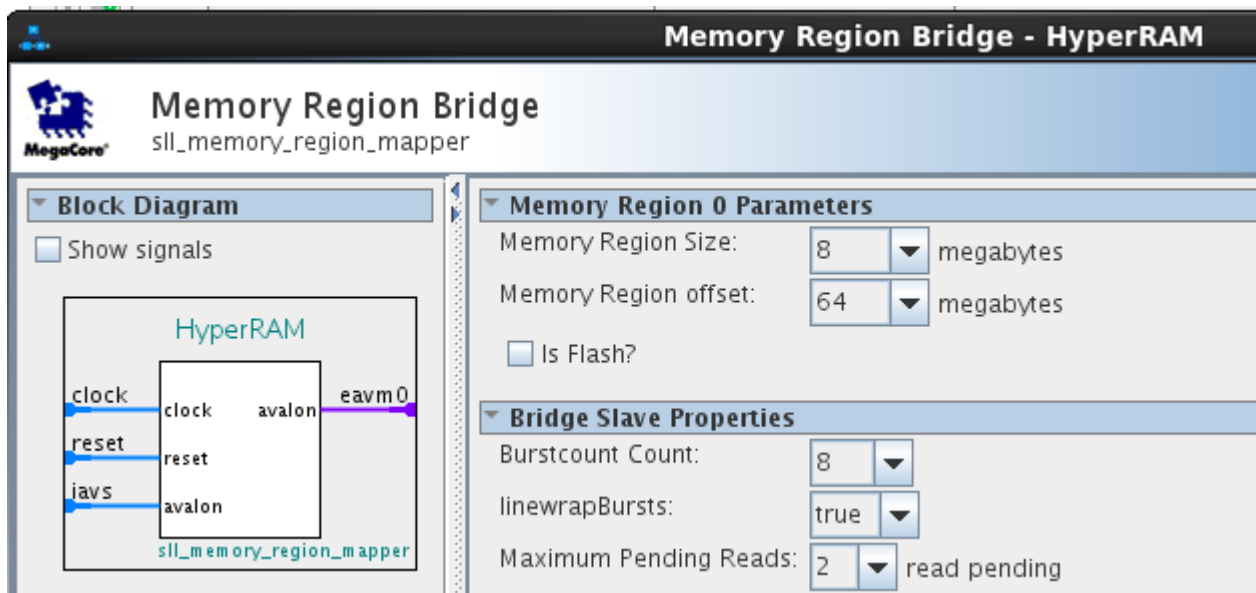
- set the **Memory region Size** to 64 megabytes (Size of the HyperFlash)
- set the **Memory region Offset** to 0 (HyperFlash is located at the start of the address space)
- Tick the **IsFlash** button to [x]
- Leave the other options unchanged
- Ignore any reported messages
- Click on Finish

Rename the instance to HyperFlash



## Adding HyperRAM Region bridge

Open S/Labs Memory region bridge located in the IP catalog library



In the Memory Region 0 Parameters

- set the **Memory region Size** to 8 megabytes (Size of the HyperRAM)
- set the **Memory region Offset** to 64 megabytes (offset is defined as the size of the HyperFlash)
- UnTick the **IsFlash** button to []
- Leave the other options unchanged
- Ignore any reported messages
- Click on Finish
- 

Rename the instance to HyperRAM.

## Wiring the connections

- Connect the clocks and the reset signals
- Remove Nios II instruction and data masters → S/Labs HBMC iavs0 slave connection
- HyperRAM Bridge connections and Base address
  - Connect Nios II instruction and data masters → HyperRAM iavs slave
  - Connect HyperRAM eavm0 slave → S/Labs HBMC iavs0 slave connection
  - Set Base Address to 0x04000000
- HyperFlash Bridge connections and Base address
  - Connect Nios II instruction and data masters → HyperFlash iavs slave
  - Connect HyperFlash eavm0 slave → S/Labs HBMC iavs0 slave connection
  - Set Base Address to 0x00000000

Use	Connections	Name	Description	Export	Clock	Base	End
<input checked="" type="checkbox"/>		clk_0	Clock Source	clk	exported		
<input checked="" type="checkbox"/>		clk_in	Clock Input	Double-click to export	clk_0		
<input checked="" type="checkbox"/>		clk_in_reset	Reset Input	Double-click to export			
<input checked="" type="checkbox"/>		clk	Clock Output	Double-click to export			
<input checked="" type="checkbox"/>		clk_reset	Reset Output	Double-click to export			
<input checked="" type="checkbox"/>		sll_hyperbus_controller	S/Labs HyperBus Memory Contr...	Double-click to export	clk_0		
<input checked="" type="checkbox"/>		in_clk	Clock Input	Double-click to export	clk_0		
<input checked="" type="checkbox"/>		i_ext_rstn	Reset Input	Double-click to export	sll_hyper...		
<input checked="" type="checkbox"/>		i_iavs0_clk	Clock Input	Double-click to export	[i_iavs0_clk]		
<input checked="" type="checkbox"/>		i_iavs0_rstn	Reset Input	Double-click to export	[i_iavs0_clk]		
<input checked="" type="checkbox"/>		iavs0	Avalon Memory Mapped Slave	Double-click to export	hyperbus_controller.t...	0x0000_0000	0x047f_ffff
<input checked="" type="checkbox"/>		Conduit_IO	Conduit	Double-click to export	sll_hyperb...		
<input checked="" type="checkbox"/>		o_av_out_clk	Clock Output	Double-click to export	sll_hyperb...		
<input checked="" type="checkbox"/>		o_av_out_rstn	Reset Output	Double-click to export	sll_hyperb...		
<input checked="" type="checkbox"/>		HyperRAM	Memory Region Bridge	Double-click to export	sll_hyper...		
<input checked="" type="checkbox"/>		clock	Clock Input	Double-click to export	[clock]	0x0400_0000	0x047f_ffff
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	[clock]		
<input checked="" type="checkbox"/>		iavs	Avalon Memory Mapped Slave	Double-click to export	sll_hyper...		
<input checked="" type="checkbox"/>		eavm0	Avalon Memory Mapped Master	Double-click to export	[clock]	0x0000_0000	0x03ff_ffff
<input checked="" type="checkbox"/>		HyperFlash	Memory Region Bridge	Double-click to export	sll_hyper...		
<input checked="" type="checkbox"/>		clock	Clock Input	Double-click to export	[clock]		
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	[clock]		
<input checked="" type="checkbox"/>		iavs	Avalon Memory Mapped Slave	Double-click to export	[clock]		
<input checked="" type="checkbox"/>		eavm0	Avalon Memory Mapped Master	Double-click to export	[clock]		
<input checked="" type="checkbox"/>		nios2_gen2_0	Nios II Processor	Double-click to export	sll_hyper...		
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		irq	Interrupt Receiver	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		debug_reset_requ...	Reset Output	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		custom_instruction	Custom Instruction Master	Double-click to export	[clk]		

When finished, press Generate HDL

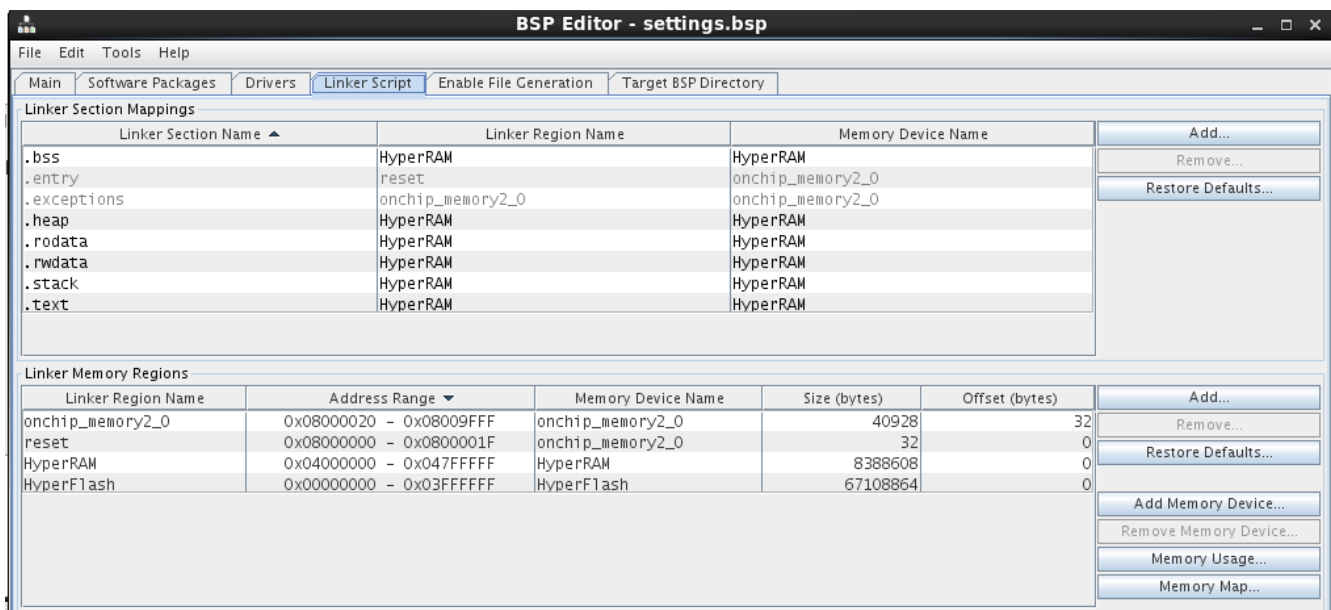
## 6.3 Configure the Linker regions inside Eclipse

This steps assumes that the user knows how to generate a Nios II application and BSP package. If needed, please refer to other tutorials for guidance.

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 linker section, one can see the HyperRAM and HyperFlash memory regions. In this example, all the data and text regions (.text, .bss, .heap, .rodata, .rdata, .stack) regions are mapped to HyperRAM memory device.



Another tutorial is available for running firmware from HyperFlash. Please refer to [SynapticLabs-HBMC-Tutorial005A-NiosII-HyperFlash\\_Test.pdf](#)

**Note: C Source code provided with some tutorials included reference to SLL\_HYPERBUS\_CONTROLLER\_TOP\_0\_BASE parameter. This parameter pointed to the start of S/Labs' HBMC IP memory region. Please note that this parameter is no longer present. Please change the c test program according to the guide lines mentioned below.**

**With the updated configuration, new defines are available to the programmer. These are included in the system.h located in the bsp directory**

- **HYPERAM\_BASE** - this is the start address of the HyperRAM memory region
- **HYPERAM\_SPAN** - this is the HyperRAM memory span
- **HYPERFLASH\_BASE** - this is the start address of the HyperFlash memory region
- **HYPERFLASH\_SPAN** - this is the HyperFlash memory span

**Here, HyperRAM\_.... and HyperFLASH\_... denotes the names given to the HyperRAM and HyperFlash memory bridges in Qsys. Change accordingly if different names are used**

Replace to following code lines with

```
#define SLL_HYPERRAM_ONLY

#ifdef SLL_HYPERRAM_ONLY
    #define SLL_HYPERRAM_BASE (SLL_HYPERBUS_CONTROLLER_TOP_0_BASE)
#else
    #define SLL_HYPERRAM_BASE (SLL_HYPERBUS_CONTROLLER_TOP_0_BASE + 0x04000000)
#endif
#define SLL_HYPERRAM_SPAN (0x00800000)
```

```
error += db_tests_hypermax_check_cfi (SLL_HYPERBUS_CONTROLLER_TOP_0_BASE, USE_PRINTF);
```

with

```
// #define SLL_HYPERRAM_ONLY

#define SLL_HYPERRAM_BASE  HYPERRAM_BASE
#define SLL_HYPERRAM_SPAN  HYPERRAM_SPAN
```

```
error += db_tests_hypermax_check_cfi (HYPERFLASH_BASE, USE_PRINTF);
```