



# Using HLS AXI Master on Zynq

*Targeting microzed board*

Author: Stefano Tabanelli



## Table of Contents

1 Introduction.....	3
2 Setting up the host.....	3
2.1 Create a VirtualBox Virtual Machine.....	3
2.2 Install packages required by Yocto.....	3
2.3 Xilinx tools installation.....	4
2.4 Copying project files to VM.....	4
2.5 Setting up git.....	4
2.6 Installing Architech Yocto “Dizzy”2.....	4
3 Vivado High Level Synthesis AXI Master project.....	6
3.1 Vivado HLS AXI Master project, compile and export IP (optional).....	7
3.2 Instantiate HLS AXI Master IP on a Vivado project, targeting microzed board.....	9
3.3 Create uSD boot file BOOT.BIN.....	11
3.4 Adding a Linux Kernel Module device driver for HLS Axi Master to rootfs (Yocto recipe).....	12
3.5 Device tree setup.....	13
3.6 MicroSD image creation, putting the pieces together.....	14
4 Show me it’s working!.....	15
5 Appendix.....	18
5.1 Assumptions and limits of AXI Master template.....	18
5.2 Flashing a 4GB uSD with a working image.....	18
5.3 More on Yocto.....	19



# 1 Introduction

The purpose of this document is to show how to rebuild the entire HLS Axi Master project and uSD boot image for microzed board. A detailed explanation of the project components and their functions are not included on this version and will be added in future releases.

The required steps are:

- Setup a VirtualBox virtual machine and install Ubuntu 14.04.2 64 bit, Xilinx Vivado 2015.1, Architech Yocto “Dizzy” (Ch. 2)
- Build and export HLS Axi Master IP to Vivado (Ch. 3.1 optional, requires a Vivado High Level Synthesis license)
- Build the Vivado project that instantiates the HLS Axi Master IP on Zynq (Ch. 3.2)
- Create the boot file required by Zynq (BOOT.BIN) containing First Stage Bootloader, Vivado project bitstream and uboot (Ch. 3.3)
- Add the Linux Kernel module for HLS AXI Master IP to the rootfs
- Flash a uSD containing BOOT.BIN, linux kernel, device tree on the first FAT partition and linux rootfs on the second ext3 partition

## 2 Setting up the host

We'll be working on a machine running Ubuntu 14.04.2 64 bit, Vivado 2015.1, Architech Yocto “Dizzy”

### 2.1 Create a VirtualBox Virtual Machine

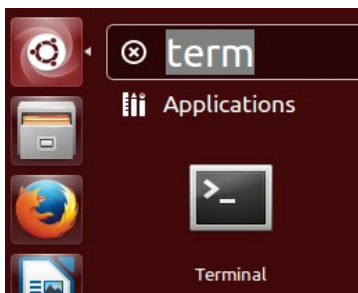
Install Virtualbox on your host, download Ubuntu 14.04.2 64 bit Desktop .iso image and install it to the Virtual Machine. To be aligned to this document create a user named “user”.

### 2.2 Install packages required by Yocto

First we need to open a terminal, click the top left icon



and write “term”, then click on Terminal icon





Run the following commands:

```
sudo dpkg-reconfigure dash
    when prompted hit <no>
sudo apt-get update
sudo apt-get install curl fxload gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential
chrpath libstd1.2-dev xterm vim curl minicom xsltproc cmake u-boot-tools1
```

## 2.3 Xilinx tools installation

Follow the steps showed on

<https://github.com/Architech-Silica/Installing-Vivado-2015.1-Ubuntu-14.04.2-64b>

tip: in case your host PC is a Linux one, you need the following command to recognize usb devices on VirtualBox

```
sudo usermod -a -G vboxusers $USER
```

## 2.4 Copying project files to VM

Create a directory using

```
mkdir $HOME/Utils
```

then copy the supplied directory HLS\_AxiMaster to it

## 2.5 Setting up git

We've previously installed the git utility, which is the versioning tool we'll use to sync with the Architech repository, before using it we need to set the following with our details:

```
git config --global user.email "you@example.com"
```

```
git config --global user.name "Your Name"
```

## 2.6 Installing Architech Yocto “Dizzy”<sup>2</sup>

```
cd $HOME
```

```
mkdir architech_sdk
```

```
cd architech_sdk
```

```
git clone -b dizzy https://github.com/architech-boards/microzed-splashscreen.git
```

```
cd microzed-splashscreen
```

```
./run_install
```

After the installation we need to configure and compile the Yocto packages, type:

---

<sup>1</sup> On other distributions can be named `uboot-mkimage`

<sup>2</sup> More informations on <http://architechboards-microzed.readthedocs.org/en/latest/create-sdk.html>



```
cd $HOME/architech_sdk/yocto
```

tip: \$HOME is an environment variable that point to user home directory (/home/user)

Now let's set the required environment variables using

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

tip: you may substitute the command *source* with a dot “.”

tip2: every time you open a new terminal for yocto developing you must source this script

we're automatically on the generated directory *yocto/build* where two important configuration files are located, in the subdirectory *conf*

let's have a look to *local.conf* with any editor (*gedit* / *vim* / *kate*...)

```
cd conf
```

```
gedit local.conf &
```

tip: appending an “&” at the end of any command will execute that command in a separate thread, so that the current terminal will be available for other commands, otherwise it'll be blocked until the end of the command (in our case *gedit*)

Currently there are 2 mistakes in the *local.conf* file, variables *DL\_DIR* and *SSTATE\_DIR* point to non existing directories, double check and in case amend this by editing *local.conf* as follow

```
DL_DIR = "/home/user/architech_sdk/yocto/downloads"
```

```
SSTATE_DIR = "/home/user/architech_sdk/yocto/sstate-cache"
```

then create the directories typing

```
mkdir $HOME/architech_sdk/yocto/downloads
```

```
mkdir $HOME/architech_sdk/yocto/sstate-cache
```

A second configuration file is important:

```
gedit bblayers.conf &
```

In this case the configuration is already correct so don't save and close the editor.

It's time to build the Yocto image:

```
cd $HOME/architech_sdk/yocto/build
```

```
bitbake core-image-minimal
```

The first time you run the build it'll take a **lot** of cpu computation time, download bandwidth and hard drive space. Due to the fact that u-boot, the most used linux bootloader, is not included in *core-image-minimal* recipe, we have to explicitly invoke its build using command:

```
bitbake u-boot-xlnx
```



### 3 Vivado High Level Synthesis AXI Master project

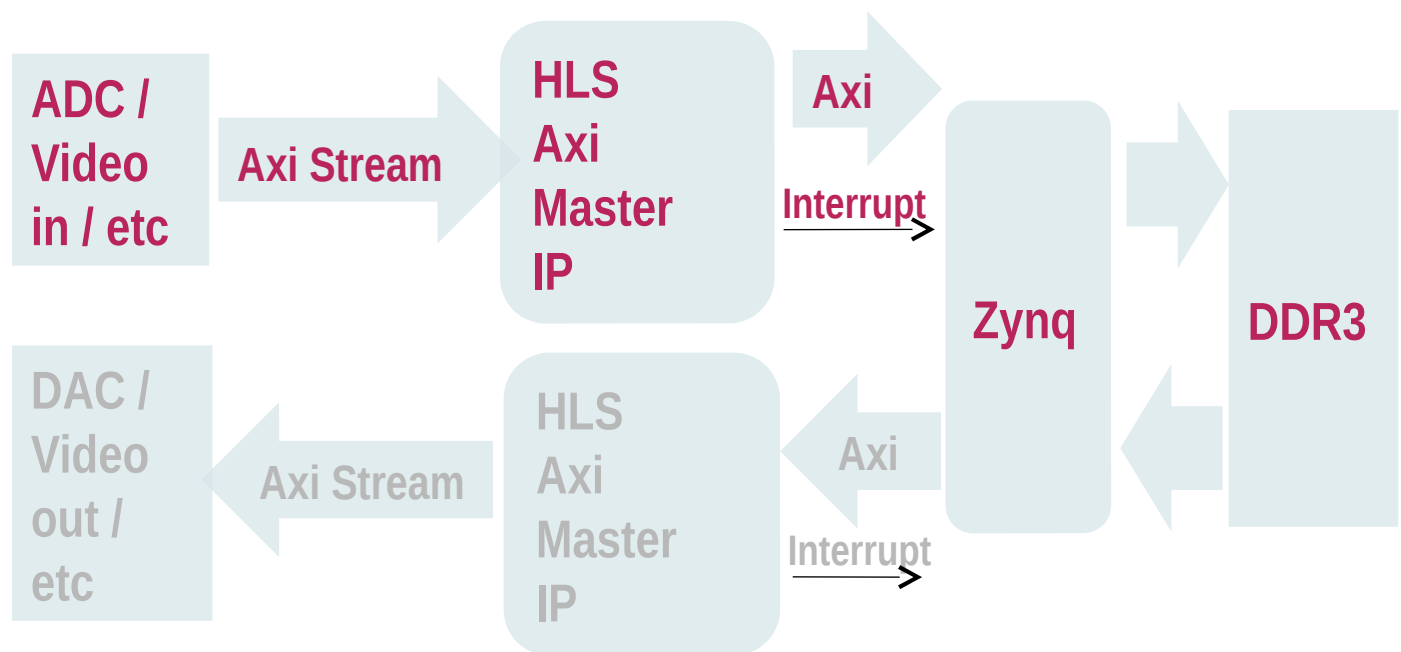
There is a common question often coming from Zynq users: how can I develop an AXI Master IP?

An AXI Master IP is able to access a resource (typically the DDR memory) independently from the processors (CortexA9 cores) to do tasks like:

- Video acquisition to memory / Video output from memory
- ADC data acquisition to memory / DAC output driving from memory
- Data filtering / elaboration on the fly or directly on memory

It's easy to imagine that a sync method is needed to coordinate these tasks to the Cortex A9 cores, likely an interrupt, and whenever an interrupt is involved a driver is suggested.

This chapter is aimed to show an AXI Master template that receives data from an AXI Stream source (eg video/data acquisition) and write these data to ZYNQ DDR memory:



The AXI Master is written in ANSI C language and we'll use a powerful Xilinx tool, Vivado High Level Synthesis, to translate it to VHDL/Verilog and to export the result as a Zynq peripheral in IP Integrator form. Moreover, we want to handle an interrupt and we'll show a Linux kernel module able to address all of this, supplied as Yocto recipe. Vivado High Level Synthesis is able to create very complex IPs with different optimizations, this doc is not targeted to explain in details the tool. Instead what we're going to do is using the bare minimum of HLS to create an AXI Master template, interrupt capable, without the need to know HDL/Verilog or AXI connection.



### 3.1 Vivado HLS AXI Master project, compile and export IP (optional)

Source the vivado.sh script to set environment using

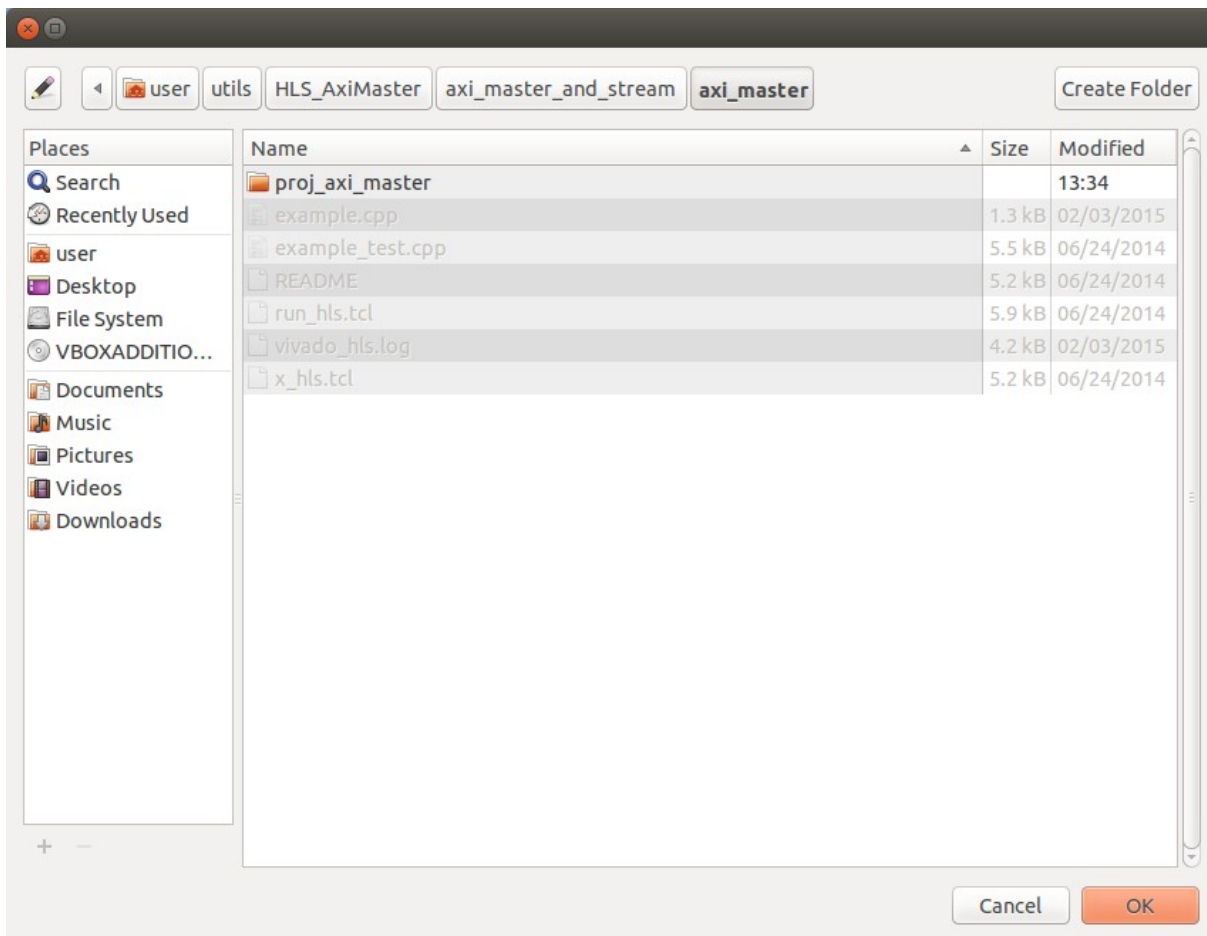
```
. vivado.sh
```

launch Vivado High Level Synthesis typing

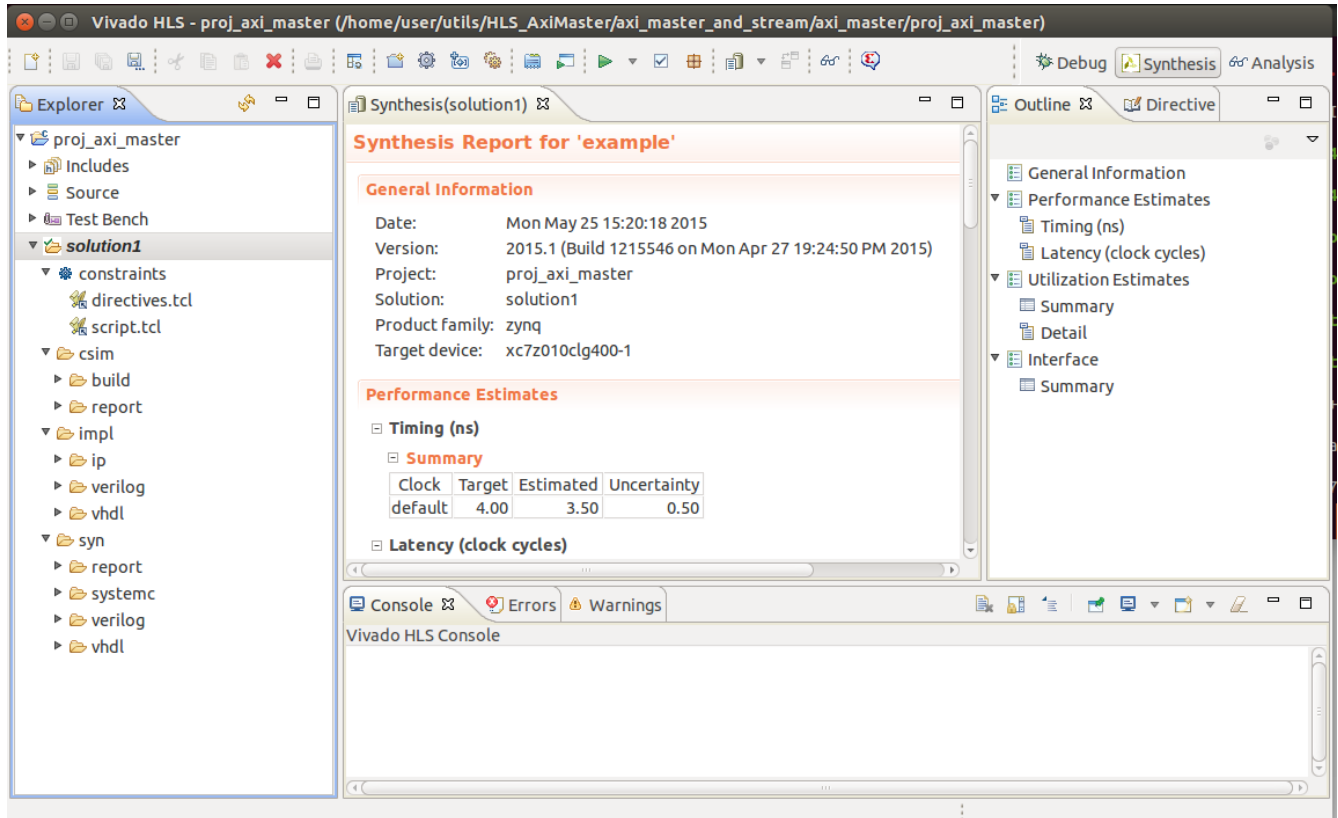
```
vivado_hls
```

then open project located in

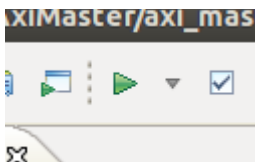
```
$HOME/utils/HLS_AxiMaster/axi_master_and_stream/axi_master/proj_axi_master/
```



You should see :



Now compile (= translate C to vhd/verilog) left-clicking on the green triangle icon



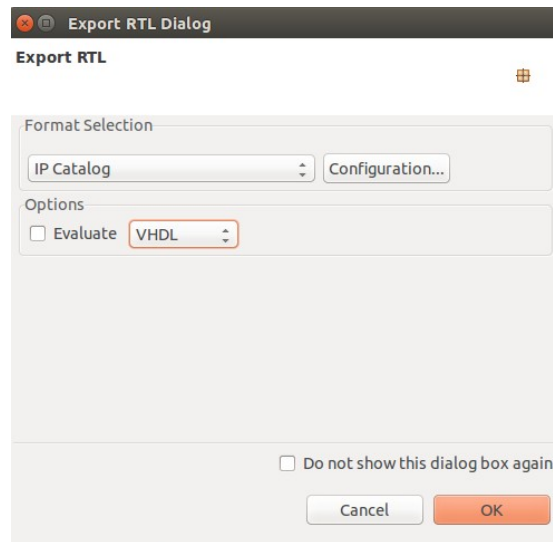
and export this project as Vivado IP Integrator module right-clicking on “solution1”







and selecting “Export RTL”



### 3.2 Instantiate HLS AXI Master IP on a Vivado project, targeting microzed board

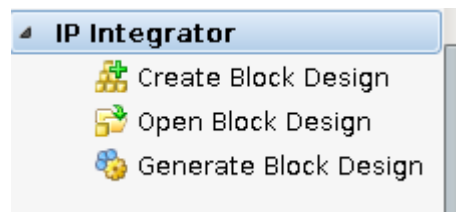
If not already done, source the vivado.sh script to set environment using

. vivado.sh

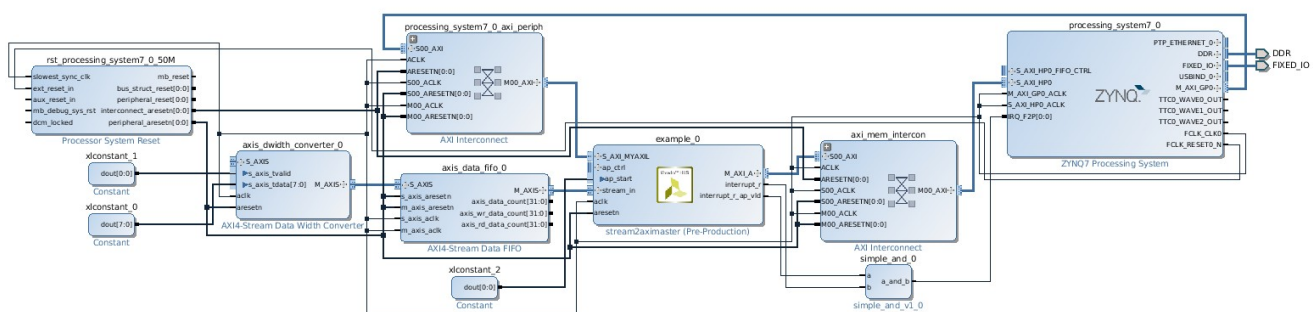
then launch Vivado while opening project project\_1.xpr typing

`vivado $HOME/utils/HLS_AxiMaster/using_hls_master_stream/project_1.xpr`

Select “Open Block Design” to have a graphical view of the project:



block design view:

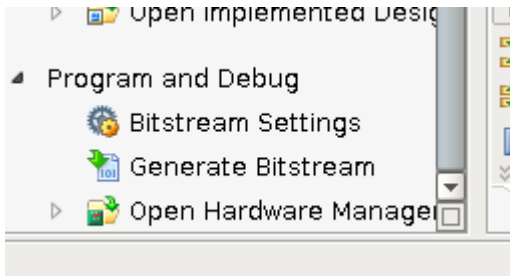


Instance name	Description
axis_dwidth_converter_0	Convert an AXI Stream Slave data input (S_AXIS) to a different data width AXI Stream Master output (M_AXIS), in this case 8bit → 32bit
axis_data_fifo_0	Implement a fifo, source data from slave AXI Stream Slave connection(S_AXIS), output data via AXI Stream Master connection (M_AXIS)
example_0	Source data from slave AXI Stream Slave connection(stream_in) and write to AXI Master connection (M_AXI_A), in this case Zynq's DDR3

Signal	Value	Note
xlconstant_0[7:0]	0xCA	Constant used as s_axis_tdata[7:0] to be written to DDR3, in real applications this could be video or ADC data acquired
xlconstant_1[0:0]	1	Assigned to s_axis_tvalid so that S_AXIS will be always enabled to receive s_axis_tdata[7:0]
xlconstant_2[0:0]	1	Assigned to ap_start so that block example_0 is always enabled and ready to receive data on stream_in port
S_AXI_MYAXIL		Mapped on Zynq AXI_GP0 to read/write status registers of instance example_0
interrupt_r		Corresponds to boolean C variable “interrupt” as per source code of HLS AXI Master project, but can only be evaluated when signal interrupt_r_ap_valid is active



create the bitstream left-clicking on “Generate Bitstream”:



### 3.3 Create uSD boot file BOOT.BIN

To create the boot file required by Zynq use the command:

```
cd /home/user/utls/HLS_AxiMaster/using_hls_master_stream
```

```
bootgen -image boot.bif -o /home/user/utls/HLS_AxiMaster/using_hls_master_stream/BOOT.BIN
```

as an alternative, you may use xsdk, menu “Xilinx Tools → Create Zynq Boot Image”:

**Create Zynq Boot Image**  
Creates Zynq Boot Image in .bin and .mcs formats from given FSBL elf and partition files in specified output folder.

☒ Create new BIF file ☐ Import from existing BIF file

Output BIF file path:

☐ Use Authentication

Authentication keys:

PPK:   PSK:

SPK:   SSK:

SPK signature:

☐ Use encryption

Encryption key:

Key file:

Key store: ☒ BRAM ☐ EFUSE

Part name:

Boot image partitions

File path	Encrypt	
(bootloader) /home/user/utls/HLS_AxiMaster/using_hls_master_stream/project_1.sdk/fsbl/Debug/fsbl.elf	none	<input type="button" value="Add"/>
/home/user/utls/HLS_AxiMaster/using_hls_master_stream/project_1.runs/impl_1/design_1_wrapper.bit	none	<input type="button" value="Delete"/>
/home/user/yocto/build/tmp/deploy/images/microzed/u-boot.elf	none	<input type="button" value="Edit"/>
		<input type="button" value="Up"/>
		<input type="button" value="Down"/>



### 3.4 Adding a Linux Kernel Module device driver for HLS Axi Master to rootfs (Yocto recipe)

A minimal kernel device module is available under directory `$HOME/utils`, copy it to Yocto hierarchy using:

```
cd $HOME/utils/HLS_AxiMaster
```

```
cp -r recipes-hlsaxim/ $HOME/yocto/meta-xilinx/
```

to check the correct module compilation type:

```
cd $HOME/yocto
```

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

```
bitbake hello-mod
```

then we'd like to include this module (and an utility used later) into the root file system, to do it edit the yocto configuration file typing:

```
gedit conf/local.conf
```

then append the following line to the end of the file and save it:

```
CORE_IMAGE_EXTRA_INSTALL += " hello-mod rwmem "
```

tip: DO NOT FORGET the space character before hello-mod and after rwmem!

To rebuild the rootfs use:

```
bitbake core-image-minimal
```

```
user@user-VirtualBox: ~/yocto/build

Build Configuration:
BB_VERSION      = "1.24.0"
BUILD_SYS       = "x86_64-linux"
NATIVELSBSTRING = "Ubuntu-14.04"
TARGET_SYS      = "arm-poky-linux-gnueabi"
MACHINE         = "microzed"
DISTRO          = "poky"
DISTRO_VERSION  = "1.7.1"
TUNE_FEATURES   = " arm armv7a vfp neon zynq"
TARGET_FPU      = "vfp-neon"
meta
meta-yocto
meta-yocto-bsp   = "(nobranch):016d607e23e2f4e0cd3c6c76bebe482651d56e7a"
meta-oe         = "(nobranch):6413cdb66acf43059f94d0076ec9b8ad6a475b35"
meta-xilinx     = "(nobranch):7f759048bb0aeef3c0b3938be81d2bcade7acb7e"
meta-microzed   = "(nobranch):7191185326997111ece432a4c62fcec8fb5c4213"

NOTE: Preparing runqueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
Currently 1 running tasks (2145 of 2147):
0: core-image-minimal-1.0-r0 do_rootfs (pid 16083)
```



### 3.5 Device tree setup

Recent linux kernels are relying on a device tree file that describes the specific hw we'll be running on. For microzed the text version of the device tree is located under Yocto hierarchy.

We need to modify the amount of DDR3 memory dedicated to the kernel, because our HLS AXI Master will write directly to absolute address 0x10010000, to do so let's copy the .dts file to a proper location:

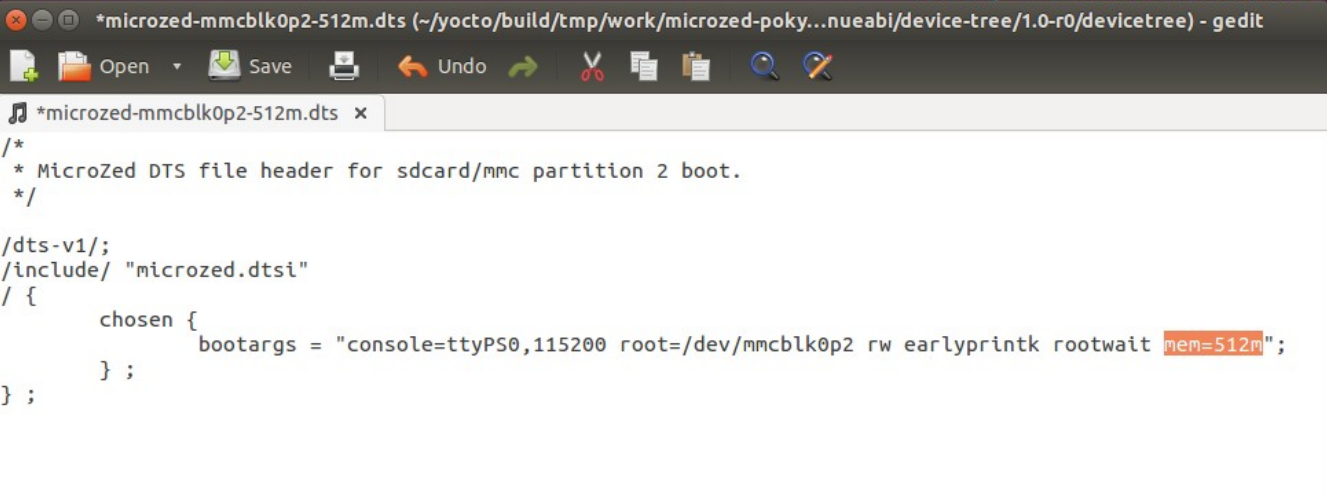
```
cd $HOME/yocto/build/tmp/work/microzed-poky-linux-gnueabi/device-tree/1.0-r0/devicetree
```

```
cp microzed-mmcbk0p2.dts microzed-mmcbk0p2-512m.dts
```

open an editor using:

```
gedit microzed-mmcbk0p2-512m.dts
```

and append parameter “mem=512m” as shown below, then save file:



```
*microzed-mmcbk0p2-512m.dts (-/yocto/build/tmp/work/microzed-poky...nueabi/device-tree/1.0-r0/devicetree) - gedit
/*
 * MicroZed DTS file header for sdcard/mmc partition 2 boot.
 */

/dts-v1/;
/include/ "microzed.dtsi"
/ {
    chosen {
        bootargs = "console=ttyPS0,115200 root=/dev/mmcbk0p2 rw earlyprintk rootwait mem=512m";
    };
};
```

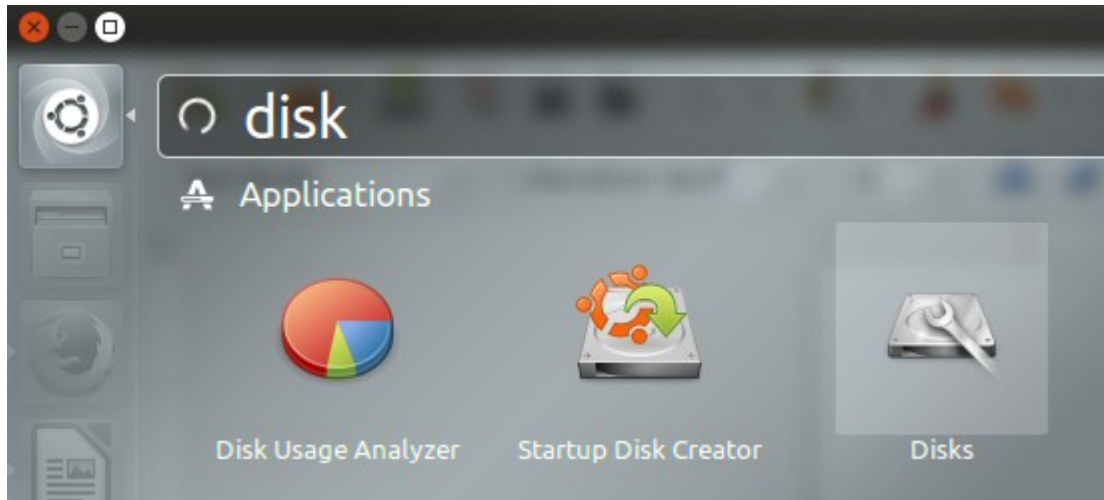
we need to convert the .dts file into a format known by the kernel (.dtb), to do it execute the command dtc:

```
$HOME/yocto/build/tmp/sysroots/x86_64-linux/usr/bin/dtc -I dts -O dtb -o $HOME/Utils/HLS_AxiMaster/microzed-mmcbk0p2-512m.dtb microzed-mmcbk0p2-512m.dts
```



### 3.6 MicroSD image creation, putting the pieces together

We have everything in place to finally create the uSD image for microzed, first plug a uSD (4GB suggested) into the VM, use Ubuntu utility “Disks” to create a first FAT partition (eg 256MB size) named FATBOOT and a second ext3 partition (as big as the rest of the uSD) named ROOTFS



```
cp $HOME/utils/HLS_AxiMaster/using_hls_master_stream/BOOT.bin /media/FATBOOT/  
cd $HOME/yocto/build/tmp/deploy/images/microzed  
cp uImage /media/FATBOOT  
cp $HOME/utils/HLS_AxiMaster/microzed-mmcbld0p2-512m.dtb /media/FATBOOT/  
sudo tar -C /media/ROOTFS/ -xzf core-image-minimal-microzed.tar.gz
```

tip: note that in case of any issue you may find an alternative procedure to flash a prebuild uSD image in Appendix Ch 5.2





## 4 Show me it's working!

Insert the uSD into microzed, connect a microUSB cable to power up the board, then launch a serial terminal typing:

`sudo minicom -s`

check the terminal settings as follow, then press Enter to accept, then select “Exit”:

```
stabanelli@xeon-ubuntu: ~  
  
+-----+  
| A -   Serial Device       : /dev/ttyUSB0  
| B - Lockfile Location    : /var/lock  
| C -   Callin Program     :  
| D -   Callout Program    :  
| E -     Bps/Par/Bits      : 115200 8N1  
| F - Hardware Flow Control : No  
| G - Software Flow Control : No  
|  
|   Change which setting?  |  
+-----+  
| Screen and keyboard |  
| Save setup as dfl   |  
| Save setup as..     |  
| Exit                |  
| Exit from Minicom   |  
+-----+
```

Now we should be on the uboot prompt, copy paste the following command lines for uboot:

`fatload mmc 0:1 0x30000000 uImage`

`fatload mmc 0:1 0x2ff0000 microzed-mmcb1k0p2-512m.dtb`

`bootm 0x30000000 - 0x2fF0000`

this will boot the linux kernel and end up to the login prompt, type “root” to log in:

```
stabanelli@xeon-ubuntu: ~  
INIT: version 2.88 booting  
Starting udev  
[ 1.956868] udevd[591]: starting version 182  
bootlogd: cannot allocate pseudo tty: No such file or directory  
[ 3.627156] random: dd urandom read with 55 bits of entropy available  
Populating dev cache  
hwclock: can't open '/dev/misc/rtc': No such file or directory  
Wed May 27 14:33:41 UTC 2015  
hwclock: can't open '/dev/misc/rtc': No such file or directory  
INIT: Entering runlevel: 5  
Configuring network interfaces... udhcpc (v1.22.1) started  
Sending discover...  
Sending discover...  
Sending discover...  
No lease, forking to background  
done.  
Starting system message bus: dbus.  
hwclock: can't open '/dev/misc/rtc': No such file or directory  
Starting syslogd/klogd: done  
Starting Lighttpd Web Server: lighttpd.  
  
Poky (Yocto Project Reference Distro) 1.7.1 microzed /dev/ttyPS0  
microzed login: |
```



to be sure that linux is using only the first 512MB of DDR3 memory, execute the command:

`cat /proc/meminfo`

```
stabanelli@xeon-ubuntu: ~  
root@microzed:~# cat /proc/meminfo  
MemTotal:      512264 kB  
MemFree:       496796 kB  
MemAvailable:  496700 kB  
Buffers:       1216 kB  
Cached:        5028 kB  
SwapCached:    0 kB  
Active:        5180 kB  
Inactive:      2480 kB  
Active(anon):  1460 kB  
Inactive(anon): 104 kB  
Active(file):  3720 kB  
Inactive(file): 2376 kB  
Unevictable:   0 kB  
Mlocked:       0 kB  
HighTotal:     0 kB  
HighFree:      0 kB  
LowTotal:      512264 kB  
LowFree:       496796 kB  
SwapTotal:     0 kB  
SwapFree:      0 kB  
Dirty:         4 kB
```

now let's load the kernel module typing:

`modprobe hello`

```
stabanelli@xeon-ubuntu: ~  
microzed login: [ 33.564684] random: nonblocking pool is initialized  
Poky (Yocto Project Reference Distro) 1.7.1 microzed /dev/ttyPS0  
microzed login: root  
root@microzed:~#  
root@microzed:~# modprobe hello  
[ 74.568804] Init syscall module.  
[ 74.572085] Registered IRQ.  
[ 74.574934] Interrupt! Status counter value : 128 Cycles  
[ 74.574937] Rising edge enabled on interrupt 61  
[ 74.574946] Type: mknod /dev/syscall c 22 0  
[ 74.574948] And remove it after unloading the module.  
[ 74.593948] Interrupt! Status counter value : 256 Cycles  
[ 74.599209] Interrupt! Status counter value : 256 Cycles  
[ 74.604520] Interrupt! Status counter value : 512 Cycles  
[ 74.609798] Interrupt! Status counter value : 512 Cycles  
[ 74.615107] Interrupt! Status counter value : 768 Cycles  
[ 74.620387] Interrupt! Status counter value : 768 Cycles  
[ 74.625696] Interrupt! Status counter value : 1024 Cycles  
[ 74.631063] Interrupt! Status counter value : 1024 Cycles  
[ 74.636459] Interrupt! Status counter value : 1280 Cycles  
[ 74.641843] Interrupt! Status counter value : 1408 Cycles
```

the kernel driver module will start the HLS AXI Master transactions and will print several debug messages, after having received 100 interrupts from the HLS AXI Master the kernel driver module will disable the transactions.





To exchange data or commands with the kernel driver module we have to create a node, using 22 as device major number, to do so use the command:

```
mknod /dev/syscall c 22 0
```

then send the character “1” to the device typing:

```
echo 1 > /dev/syscall
```

this will trigger one more transaction on the HLS AXI Master:

```
stabanelli@xeon-ubuntu: ~
[ 75.095197] Interrupt! Status counter value : 12160 Cycles
[ 75.100652] Interrupt! Status counter value : 12160 Cycles
[ 75.106134] Interrupt! Status counter value : 12416 Cycles
[ 75.111603] Interrupt! Status counter value : 12544 Cycles
[ 75.117057] Interrupt! Status counter value : 12544 Cycles
[ 75.122539] Interrupt! Status counter value : 12800 Cycles
[ 75.127991] 100 interrupts have been registered.
Disabling masterInterrupt! Status counter value : 12800 Cycles
[ 75.138146] 100 interrupts have been registered.
Disabling masterroot@microzed:~#
root@microzed:~#
root@microzed:~# mknod /dev/syscall c 22 0
root@microzed:~# echo 1 > /dev/syscall

[ 175.022345] syscall_write.
[ 175.025141] Received: 1
[ 175.025141] u tried to open the syscall module.
[ 175.025141]
[ 175.033728] Driver enables master.
[ 175.037062] Interrupt! Status counter value : 12928 Cycles
[ 175.042517] 100 interrupts have been registered.
Disabling masterroot@microzed:~# Interrupt! Status counter value : 12928 Cycles
[ 175.054146] 100 interrupts have been registered.
Disabling master
```

we want now to check if the destination addresses (0x10010000 .. 0x100101FF) of our HLS AXI Master peripheral are really populated with the expected constant 0xCA, to do so we'll use an utility named `rwmem` that work as follow:

`rwmem [address] [optional value to write]`

so in case of single parameter it will dump the address content, type the following:

```
rwmem 0x10010000
```

```
rwmem 0x10010100
```

```
rwmem 0x100101FE
```

```
rwmem 0x10010200
```

```
stabanelli@xeon-ubuntu: ~
root@microzed:~# rwmem 0x10010000
0x0000000010010000 = 0xcacacaca
root@microzed:~# rwmem 0x10010100
0x0000000010010100 = 0xcacacaca
root@microzed:~# rwmem 0x100101fe
0x00000000100101fe = 0xb81fcaca
root@microzed:~# rwmem 0x10010200
0x0000000010010200 = 0xf77fb81f
root@microzed:~#
```



as final check, overwrite a value, retrigger a HLS AXI Master transaction and dump the same address:

```
rwmem 0x10010000 0
```

```
rwmem 0x10010000
```

```
echo 1 > /dev/syscall
```

```
rwmem 0x10010000
```

```
stabanelli@xeon-ubuntu: ~
root@microzed:~#
root@microzed:~#
root@microzed:~#
root@microzed:~#
root@microzed:~#
root@microzed:~#
root@microzed:~# rwmem 0x10010000 0
root@microzed:~# rwmem 0x10010000
0x0000000010010000 = 0x00000000
root@microzed:~# echo 1 > /dev/syscall

[ 574.494056] syscall_write.
[ 574.496848] Received: 1
[ 574.496848] u tried to open the syscall module.
[ 574.496848]
[ 574.505423] Driver enables master.
[ 574.508748] Interrupt! Status counter value : 13184 Cycles
[ 574.514206] 100 interrupts have been registered.
Disabling masterroot@microzed:~# Interrupt! Status counter value : 13184 Cycles
[ 574.525835] 100 interrupts have been registered.
root@microzed:~# rwmem 0x10010000
0x0000000010010000 = 0xcacacaca
root@microzed:~#
```

## 5 Appendix

### 5.1 Assumptions and limits of AXI Master template

The supplied linux kernel device driver module assumes the following limitations:

1. No Device Tree integration
  - a. It will map to a fixed 22 major number device , easier to read
  - b. A Device Tree based version will be available in future
2. No dynamic kernel memory allocation
  - a. It will write to absolute address 0x1001 0000, easier to read, must instruct kernel to avoid this memory area at boot time
  - b. It's the right approach when big memory space is required
  - c. A dynamic allocation version will be available in future
3. No inclusion of header files automatically generated by HLS/Vivado
  - a. Registers and interrupt are hard coded in the source code
  - b. Easier to read, a cleaner version will be available in future

### 5.2 Flashing a 4GB uSD with a working image

In case you prefer not to compile the entire flow showed above, a 4GB uSD card image is provided.



First you need to uncompress it using commands:

```
cd $HOME/utis
```

```
tar xvf uzed_uSD_dump_dd.tar.bz2
```

To flash it insert an empty uSD into your VM, triple-check the device name (eg using command *dmesg*), be sure to unmount it, then use command

```
sudo dd if=$HOME/utis/uzed_uSD_dump_dd of=/dev/sdx bs=512
```

where x needs to be updated with your device letter.

Please note that in case of mistake you may overwrite your VM hard drive content!

### **5.3 More on Yocto**

As Silica we've been creating and running Yocto trainings to help the startup phase of our customers, don't hesitate to contact your local Avnet Silica FAE to know more.