The AXIOM-board

PART ONE - Hardware guide

Contents

1 Introduction
2 Overview
3 Configuration
4 Power supply
5 Programming Cable
6 RAM

The AXIOM_ZU9EG board is based on the Zynq® UltraScale+™ MPSoC XCZU9EG-1FFC900.
List of the main features:

- XCZU9EG-1FFC900 MPSoC;
- Configuration/Booting from QSPI;
- Configuration/Booting from SD card;
- Configuration/Booting from eMMC on board;
- Configuration/Booting from JTAG;
- Clocks
  (Programmable Clock Generator for PS_CLK, PS/PL transceivers, and PL-system);
- PS DDR4 64-bit SODIMM w/ ECC;
- PL DDR4 Component (32-bit, 4Gb x 2); PS GTR assignment
- DisplayPort USB3
- PL GTH assignment
- USB Type C connectors
- PS/PL EMIO Trace Port
- Arduino Uno R3 connector
- PS MIO: UART (using USB-to-UART bridge)
3 Configuration

Boot Mode | Mode Pins [0:3]
---|---
QSPI32 | 0 1 0 0
uSD | 1 0 1 0
eMMC | 1 1 0 0
JTAG | 0 0 0 0

NOTE: logical value 0 refers to ON state. The switch state reported into the picture corresponds to SD boot mode.

| LED1 | PS_INIT_B | Initialization completion indicator after POR. High voltage indicates completion of initialization (PL). |
| LED2 | PS_DONE | If the LED glows red, the Zynq UltraScale+ device has configured successfully. |
| LED3 | USER_LED | Programmable LED form PS/PL. |
| LED4 | PS_G_PGD | If Power Good LED glows green, the power system is good. |

| SW2 | RST_BTN# | Reset button |
| SW3 | N/A | N/A |
4 Power supply

12V DC 5A power supply - 2.5mm plug (positive polarity). A suitable one can be found at [1] (https://www.digimax.it/acdc-adattatori-desktop/304-ea1050am03.html)

5 Programming Cable

A programming cable for Xilinx FPGAs can be useful for JTAG boot or FPGA programming and debug, we tested a low cost compatible one: JTAG-HS3 from digilent ([2] (http://store.digilentinc.com/jtag-hs3-programming-cable/))

The JTAG connection CN15 of the board does not have mechanical polarization, so keep attention to the polarization key of the programming cable (it must be facing outside the board - see the images below).

6 RAM

DDR4 SO-DIMM: At this time we tested 4GB RAM modules. 512MBx8, CL=15 DRAM have to be used, we tested several ones eg: KVR21SE15S8/4 from Kingstone; bigger ones (but same topology and CAS latency) can be used but, without modifications to Vivado project, 4GB will be available anyway.
Board Power Analysis

Hardware Sensor System

The board has eight INA219 chips for monitoring current and power, connected with the Zynq Ultrascale+ via I2C.SYSFS and IOCTL interfaces.


The following rails are monitored:

<table>
<thead>
<tr>
<th>Bus</th>
<th>Label</th>
<th>Nominal Voltage (V)</th>
<th>System ID (*)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.85V_INTFP</td>
<td>PM_VCC_INTFP</td>
<td>0.85</td>
<td>0</td>
<td>PS full-power domain supply voltage</td>
</tr>
<tr>
<td>0.85V_VCCINT</td>
<td>PM_VCC_INT</td>
<td>0.85</td>
<td>1</td>
<td>PL internal power supply</td>
</tr>
<tr>
<td>12V_ALW</td>
<td>PM_VIN</td>
<td>12</td>
<td>2</td>
<td>VIN power supply</td>
</tr>
<tr>
<td>0.85V_INTFP_DDR</td>
<td>PM_INTFP_DDR</td>
<td>0.85</td>
<td>3</td>
<td>PS DDR controller and PHY supply voltage</td>
</tr>
<tr>
<td>1V2_DDR_PS</td>
<td>PM_1V2_DDR_PS</td>
<td>1.2</td>
<td>4</td>
<td>PS DDR supply</td>
</tr>
<tr>
<td>1.2V_DDR_PL</td>
<td>PM_1V2_DDR_PL</td>
<td>1.2</td>
<td>5</td>
<td>PL DDR supply</td>
</tr>
<tr>
<td>MGTAVCC</td>
<td>PM_MGTAVCC</td>
<td>0.9</td>
<td>6</td>
<td>Analog supply voltage for GTH transceiver</td>
</tr>
<tr>
<td>1.2V_MGTAVTT</td>
<td>PM_MGTAVTT</td>
<td>1.2</td>
<td>7</td>
<td>Analog supply voltage for GTH transmitter and receiver termination circuits</td>
</tr>
</tbody>
</table>

(*) The System ID is a conventional zero based number used to uniquely identify the chip. This number is used also as suffix of the device file descriptor assigned at each driver instance.
## Test Setting

<table>
<thead>
<tr>
<th>OS Image:</th>
<th>XOS_v0.3_20171113</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW test:</td>
<td>hwmon_scan_all.sh</td>
</tr>
</tbody>
</table>

| Test parameters | duration: 240 s | sample time: 0.2 s |

### Client Setting

```
Node 1 started! IPoA: 192.168.17.1

root@axiom:/home/ubuntu# axiom-info -f
AXIOM NIC informations:
  interfaces[0] status: 0x02
    connected = 0
    tx-enabled = 0
    rx-enabled = 1
  interfaces[1] status: 0x01
    connected = 0
    tx-enabled = 1
    rx-enabled = 0
  interfaces[2] status: 0x02
    connected = 0
    tx-enabled = 0
    rx-enabled = 1
  interfaces[3] status: 0x01
    connected = 0
    tx-enabled = 1
    rx-enabled = 0
```

### Server Setting

```
Node 2 started! IPoA: 192.168.17.2

root@axiom:/home/ubuntu# axiom-info -f
AXIOM NIC informations:
  interfaces[0] status: 0x02
    connected = 0
    tx-enabled = 0
    rx-enabled = 1
  interfaces[1] status: 0x01
    connected = 0
    tx-enabled = 1
    rx-enabled = 0
  interfaces[2] status: 0x02
    connected = 0
    tx-enabled = 0
    rx-enabled = 1
  interfaces[3] status: 0x01
    connected = 0
    tx-enabled = 1
    rx-enabled = 0
```
1. Connect USB Type C cables on CN3-CN4-CN5-CN6 (it’s important to use USB Type C Gen 2 cables that have both lanes wired);
2. Select JTAG boot mode
3. Connect JTAG_HS3
Running the test

1. Open Hardware manager from Vivado;

2 Select Open target;

3 Select Program device end then xczu9eg_0;
4 Choose bitstream file from your PC

5. Select "Auto-detect Serial I/O Links";

6. So, in Serial I/O Links panel you can see the status of transmission;
AVR8 programming

Arduino Soft Core programming

The BRAM_prog_driver allows mapping the Soft-core program hex file in RAM blocks (for instance the program memory has been mapped to /dev/memX). We can use 32 bit AXI interface that allow, through AXI_BRAM_controller, to initiate write/read transactions from ARM-core to BRAM of AVR8 implemented in programmable logic.
In the Figure, the main components involved on the programming of the Soft Core are colored in the following way:

Linux standard interface through which the user space can interact with the IP designed in programmable logic.

Set of commands to:

- write/read to/from the program memory of the Arduino soft core
- start/stop program execution on the soft core

Linux standard interface used to communicate with the IP implemented in Programmable logic.

High level function set to manage BRAM access accordingly to the IP core’s status. For example, program’s execution is stopped when updating the program memory. Moreover it performs .hex file parsing and checksum verification.

Basic set of functions to interact with the BRAM controller.

The access to the IP takes place through a reserved memory range, through dedicated memory mapped registers.

User Space Interface

Driver: arduino.c

This driver provides an user space interface via PROCFS. The root folder is:

```
/proc/arduino
```
This last contains the following file:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>erase</td>
<td>r</td>
<td>Erase the AVR program memory</td>
</tr>
<tr>
<td>program_memory</td>
<td>r/w</td>
<td>Program the AVR program memory with the data streaming from an hex file.</td>
</tr>
<tr>
<td>run</td>
<td>r/w</td>
<td>Control/Show the state of the execution</td>
</tr>
</tbody>
</table>

During the programming operation a check of the hex file is performed in order to avoid wrong data writing. The default state of the execution is reset mode and the driver puts the execution in this state each time a dump or program operation is started.

**State control**

To read the current state of the execution:

```
root@axiom:/proc/arduino# cat run
```

To control the execution state:

```
root@axiom:/proc/arduino# echo <state> > run
```

Where `<state>` must be 1 to put the execution running, 0 to stop the execution.

**Memory Erase**

Write value 1 to "erase" file:

```
root@axiom:/proc/arduino# echo 1 > erase
```

After this operation the execution will be stopped in each case.
Memory Control
To dump the program memory:

```bash
root@axiom:/proc/arduino# cat program_memory
```

A well formatted output will be printed into the current terminal. To write a hex file into the memory:

```bash
root@axiom:/proc/arduino# cat <hex file> > program_memory
```

If the hex file has not a correct format the command returns an error. When this operation is started, the driver stops the execution state and at the end of the operation, it automatically reter the current state.

Example

Example: memory dump

```bash
root@axiom:/proc/arduino# cat program_memory
```

Example: program the memory with code.cpp.hex file

```bash
root@axiom:/proc/arduino# cat program_memory
```
Example: put execution running

Example: erase program memory

**AVR8 system communication**

Arduino Soft Core to Processing System communication

Beyond AVR processor programming, it is useful to provide a mechanism to allow communication between CPU complex integrated in Zynq’s Processing system (i.e., the ARM cores) and the AVR Soft Core embedded in Zynq’s Programmable Logic (Figure 5). The communication between user application and Arduino Soft Core is provided in a standard way, with a kernel driver and a set of APIs. The first one (see yellow block in Figure 6) has the double purpose to implement the communication interface between the Kernel and the Soft Core and the communication interface between the kernel space and the user space, provided as set of primitives. The second one (see blue block in the picture below) is a dynamic library, which uses these primitives and implements a higher set of functions that can be called by user applications. The two main interfaces (user space/kernel space and kernel/Soft Core) are asynchronous, so the driver implements a synchronization mechanism that queues the tasks to perform. The priority can be assigned at run-time or in a static way, based on the specific operation. In order to use the Soft Core as Slave device and allow it to notify events to the master, the driver handles an interrupt routine (ISR) in which the slave notification is decoded and the message (from the slave) is passed to the user space via kernel event system. In this way, an asynchronous communication from the slave to the user application is provided.

---

**Note:**
- **buffer size:** 64 bit.
- **IN_BUFFER:** send data from PS to AVR8.
- **OUT_BUFFER:** send data from AVR8 to PS.
Linux standard interface through which the user space can interact with the IP designed in programmable logic.

Set of to write/read to/from the bi-directional buffer and a set of commands to manage the FIFO controller.

Linux standard interface used into the custom code to interact with the IP designed in programmable logic.

High level function set. According to the current state of the controller, it allows or disable some user function.

Basic set of functions to interact with the FIFO controller.

The access to the IP takes place through a reserved memory range, through dedicated memory mapped registers.

### AVR8 Side

Register List:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rx_reg</td>
<td>0x1FF0</td>
<td>Provides data from PS</td>
</tr>
<tr>
<td>Tx_reg</td>
<td>0x1FF1</td>
<td>Sends data to PS</td>
</tr>
<tr>
<td>INT_status_reg</td>
<td>0x1FF2</td>
<td>Interrupt status register</td>
</tr>
<tr>
<td>Control_reg</td>
<td>0x1FF3</td>
<td>Control register</td>
</tr>
<tr>
<td>Mask_reg</td>
<td>0x1FF4</td>
<td>Interrupt enable register</td>
</tr>
</tbody>
</table>

Register Mapping:

#### INT_status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-4</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>Rxsmmit_ready_IN_buffer</td>
<td>ROC</td>
<td>IN_buffer has data to provide</td>
</tr>
<tr>
<td>2</td>
<td>Rno_more_data_IN-buffer</td>
<td>ROC</td>
<td>IN_buffer no more data to provide</td>
</tr>
<tr>
<td>1</td>
<td>Wbuffer_busy_OUT_buffer</td>
<td>ROC</td>
<td>OUT_buffer busy</td>
</tr>
<tr>
<td>0</td>
<td>Wbuffer_empty_OUT_buffer</td>
<td>ROC</td>
<td>OUT_buffer empty</td>
</tr>
</tbody>
</table>

#### Control Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-3</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>Wbuffer_flush_OUT_buffer</td>
<td>WO</td>
<td>OUT_buffer data flush</td>
</tr>
<tr>
<td>1</td>
<td>Wbuffer_en_OUT_buffer</td>
<td>WO</td>
<td>OUT_buffer write enable</td>
</tr>
<tr>
<td>0</td>
<td>Rbuffer_en_IN_buffer</td>
<td>WO</td>
<td>IN_buffer read enable</td>
</tr>
</tbody>
</table>

#### Mash Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-4</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>Rxsmmit_ready_IN_buffer</td>
<td>WO</td>
<td>Enable flag of relative interrupt</td>
</tr>
<tr>
<td>2</td>
<td>Rno_more_busy_OUT_buffer</td>
<td>WO</td>
<td>Enable flag of relative interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Wbuffer_busy_OUT_buffer</td>
<td>WO</td>
<td>Enable flag of relative interrupt</td>
</tr>
<tr>
<td>0</td>
<td>Wbuffer_empty_OUT_buffer</td>
<td>WO</td>
<td>Enable flag of relative interrupt</td>
</tr>
</tbody>
</table>
Arduino Interface: In order to use the communication system an Arduino library is provided.

Library name: PSComm
Heder to include: include "PSCOMMClass.h"

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin</td>
<td>begin()</td>
<td>none</td>
<td>none</td>
<td>initializes the buffer interface. To call before any buffer operation.</td>
</tr>
<tr>
<td>end</td>
<td>end()</td>
<td>none</td>
<td>none</td>
<td>Reset the buffer interface. To call after any buffer operation.</td>
</tr>
<tr>
<td>write8</td>
<td>write8()</td>
<td>uint8_t data: data to read</td>
<td>none</td>
<td>Writes a 8bit data into the buffer and flush all.</td>
</tr>
<tr>
<td>write16</td>
<td>write16()</td>
<td>uint16_t data: data to read</td>
<td>none</td>
<td>Writes a 16bit data into the buffer and flush all.</td>
</tr>
<tr>
<td>write32</td>
<td>write32()</td>
<td>uint32_t data: data to read</td>
<td>none</td>
<td>Writes a 32bit data into the buffer and flush all.</td>
</tr>
<tr>
<td>write64</td>
<td>write64()</td>
<td>uint64_t data: data to read</td>
<td>none</td>
<td>Writes a 64bit data into the buffer and flush all.</td>
</tr>
<tr>
<td>hasDataToRead</td>
<td>hasDataToRead()</td>
<td>none</td>
<td>0 or 1</td>
<td>Return 1 if there are some data to read.</td>
</tr>
<tr>
<td>noMoreData</td>
<td>noMoreData()</td>
<td>none</td>
<td>0 or 1</td>
<td>Return 1 if the IN_BUFFER is empty.</td>
</tr>
<tr>
<td>read8</td>
<td>read8()</td>
<td>uint8_t *data: point to data to store the read value</td>
<td>none</td>
<td>Read 8bit of data from the buffer.</td>
</tr>
<tr>
<td>read16</td>
<td>read16()</td>
<td>uint16_t data: point to data to store the read value</td>
<td>none</td>
<td>Read 16bit of data from the buffer.</td>
</tr>
<tr>
<td>read32</td>
<td>read32()</td>
<td>uint32_t data: point to data to store the read value</td>
<td>none</td>
<td>Read 32bit of data from the buffer.</td>
</tr>
<tr>
<td>read64</td>
<td>read64()</td>
<td>uint64_t data: point to data to store the read value</td>
<td>none</td>
<td>Read 64bit of data from the buffer.</td>
</tr>
</tbody>
</table>

PS Side

Note:
- There are two 32bit slot.
- The data are available to the other side only when both slot are written.
- The data are immediately readable by other side if the flush flag is set.

Register List: BASE_ADDR = 0x80030000

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rx_reg</td>
<td>BASE_ADDR + 0x00</td>
<td>Provides data from PS</td>
</tr>
<tr>
<td>Tx_reg</td>
<td>BASE_ADDR + 0x04</td>
<td>Sends data to PS</td>
</tr>
<tr>
<td>INT_status_reg</td>
<td>BASE_ADDR + 0x08</td>
<td>Interrupt status register</td>
</tr>
<tr>
<td>Control_reg</td>
<td>BASE_ADDR + 0x0c</td>
<td>Control register</td>
</tr>
<tr>
<td>Mask_reg</td>
<td>BASE_ADDR + 0x10</td>
<td>Interrupt enable register</td>
</tr>
</tbody>
</table>
### INT_status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-4</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>Rxsmitt_ready_OUT_buffer</td>
<td>ROC</td>
<td>OUT_buffer has data to provide</td>
</tr>
<tr>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>Wbuffer_busy_IN_buffer</td>
<td>ROC</td>
<td>IN_buffer busy</td>
</tr>
<tr>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Control Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-1</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>0</td>
<td>Wbuffer_flush_IN_buffer</td>
<td>WO</td>
<td>IN_buffer data flush</td>
</tr>
</tbody>
</table>

### Mash Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-4</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>Rxsmitt_ready_OUT_buffer</td>
<td>WO</td>
<td>Enable flag of relative interrupt</td>
</tr>
<tr>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>Wbuffer_busy_IN_buffer</td>
<td>WO</td>
<td>Enable flag of relative interrupt</td>
</tr>
<tr>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

---

**User Space Interface Driver: fifo.c**

This driver provides an user space interface via PROCFS. The root folder is:

```
/proc/fifo
```

This last contains the following file:

**PROC User Interface**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regs_dump</td>
<td>r</td>
<td>Dump of all registers (not included out and in buffer)</td>
</tr>
<tr>
<td>ien</td>
<td>w</td>
<td>Interrupt enable flags (interrupt mask)</td>
</tr>
<tr>
<td>wb_flush</td>
<td>r/w</td>
<td>Show/Control flush flags</td>
</tr>
<tr>
<td>out</td>
<td>w</td>
<td>Write into the OUT_buffer</td>
</tr>
<tr>
<td>in</td>
<td>r</td>
<td>Read from the IN_buffer</td>
</tr>
</tbody>
</table>

**Example:**

- **Dump registers**
  ```bash
  root@axiom:/proc/fifo# cat regs_dump
  Control : 0x00000000
  Int en : 0x00000000
  Int state: 0x00000000
  TX data : 0x00000000
  ```

- **Write data:**
  ```bash
  root@axiom:/proc/fifo# echo 0xFFFFFFFF > ien
  ```
root@axiom:/proc/fifo# echo 0x1 > out
root@axiom:/proc/fifo# echo 0x2 > out

Read data:

root@axiom:/proc/fifo# echo 0xFFFFFFFF > ien
root@axiom:/proc/fifo# cat in
0x00000001

Enable flushing:

root@axiom:/proc/fifo# echo 1 > wb_flush
# Arduino system

## Contents

- 1 Arduino Function Support
- 2 Arduino Library Support
- 3 Arduino IDE
- 4 Arduino PWM Support and Examples
- 5 Example: Analog Voltage displayed in Linux
- 6 Example: PIR with LED indication
- 7 Example: SPI NOR Flash write/read
- 8 Example: Math, Trigonometry, Characters, Random Numbers, Bits and Bytes function sets

## 1 Arduino Function Support

Reference: [Arduino functions](https://www.arduino.cc/reference/en/)

<table>
<thead>
<tr>
<th>Function</th>
<th>Status</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital I/O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>digitalWrite()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>digitalWrite()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>pinMode()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>analogRead()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>analogReference()</td>
<td>Not supported</td>
<td></td>
</tr>
<tr>
<td>analogWrite()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>analogReadResolution()</td>
<td>NC</td>
<td>No compatible board</td>
</tr>
<tr>
<td>analogWriteResolution()</td>
<td>NC</td>
<td>No compatible board</td>
</tr>
<tr>
<td>Zero, Due &amp; MKR Family</td>
<td></td>
<td></td>
</tr>
<tr>
<td>noTone()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>pulseIn()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>pulseInLong()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>shiftIn()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>shiftOut()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>tone()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>delay()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>delayMicroseconds()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>pulseInLong()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>micros()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>millis()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>Analog I/O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isAlpha()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isAlphaNumeric()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isAscii()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isControl()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isDigit()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isGraph()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isHexadecimalDigit()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isLowerCase()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isPrintable()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isPunct()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isSpace()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isUpperCase()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>isWhitespace()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>Random Numbers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>random()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>randomSeed()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>Bits and Bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>bitClear()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>bitRead()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>bitSet()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>bitWrite()</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>highByte()</td>
<td>Supported</td>
<td></td>
</tr>
</tbody>
</table>
## 2 Arduino Library Support


<table>
<thead>
<tr>
<th>Function</th>
<th>Status</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI</td>
<td>Full supported</td>
<td>AXIOM library for custom communication (see here (<a href="https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTATION/BSP/AVR8_system_communication">https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTATION/BSP/AVR8_system_communication</a>))</td>
</tr>
<tr>
<td>PSCOMM</td>
<td>Full supported</td>
<td></td>
</tr>
</tbody>
</table>

### 3 Arduino IDE

Into the File System generated with the XGenImage tool (see here (https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTATION/BSP/XGenImage)) a version of Arduino IDE is already present. It is a AARCH64 compatible version and is aligned with the libc for ARV present into Ubuntu 16.04.

Current Arduino IDE version: 1.5.2

The AVR8 IP is denominated, into the IDE, as "AXIOM AVR8 IP" and is present into the board list of the software. So the user must to select the correct item from the board list before write a sketch. In this manner all features of the IP will be available.

The Serial Monitor is available over the default serial port ttyACM0.
4 Arduino PWM Support and Examples

Tag:
- analogWrite;
- PWM;


There are two independent PWM controller available through Arduino connector. This is the combination:

<table>
<thead>
<tr>
<th>Controller</th>
<th>Pin Num.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM0</td>
<td>3</td>
</tr>
<tr>
<td>PWM2</td>
<td>6</td>
</tr>
</tbody>
</table>

(Note: PWM1 not available)

The duty cycle of the signal can by controlled via the standard analogWrite() Arduino's function. In addition, it is possible:
- to change the default clock frequency from a set of frequencies, depending on the controller in use;
- to invert the polarity of the signal (the default one is non-inverted, so the duty cycle means duration of the signal high).

These two features are not implemented by standard Arduino's function set, so the user have to directly acts at low level code and the register to use is the TCCRx (Timer/Counter Control Register), where x is 0 for the PWM0 and 2 for the PWM2. The clock source of the controller come from a Prescaler, which provide seven different clock frequencies for PWM0 and five for PWM2. The selection of the frequency take place through a three-bit selector, corresponding to the first three bit of the TCCRx register: CS02 (MSB), CS01, CS00 (LSB). Follow the association bits' value and PWM frequency:
Example: set PWM0 to 1KHz

```
TCCR0 &= 0xF8;
TCCR0 |= 0x3; // to set clk to 1KHz
```

Example: set PWM0 non-inverted

```
TCCR0 &= (0x3 << 4);
TCCR0 |= (0x2 << 4);
```

Example: set PWM0 inverted

```
TCCR0 |= (0x3 << 4);
```

Example: set PWM0 @ 32KHz, 30% duty, not inverted

```
int pinPWM = 3;
void setup() {
  TCCR0 &= 0xF8;
  TCCR0 |= 0x1;
}
void loop() {
  analogWrite(pinPWM, 80);
  for (; ;);
}
```
5 Example: Analog Voltage displayed in Linux

Tag:
- AnalogRead;
- Send data from AVR8 to PS

A Trimmer is connected to the ADC[4] of Arduino. The sketch reads the Voltage as result of a resistor partition and sends this value to the PS. At PS side, there is a python script which reads the values sent by the AVR, through the buffer, and displays them in a window as show in the pictures below.

```cpp
#include "PSComm.h"
int analogPin = 4;
int val = 0;
int val1 = 0;

void setup() {
  PSComm.begin();
  val = 0;
  val1 = 0;
  delay (500);
}

void loop() {
  val = analogRead(analogPin);
  if (val != val1) {
    PSComm.write16 ((uint16_t)val);
    val1 = val;
    delay(200);
  } else {
    delay(100);
  }
}
```
6 Example: PIR with LED indication

Tag:
- pinMode;
- digitalRead;
- digitalWrite;

A PIR sensor is connected to the Arduino's shield in the follow mode: - Pin 6 -> sensor's signal (active Low); - Pin 8 Power group -> 3.3V - Pin 2 Power group -> GND When the PIR sensor recognises a movement the four leds of the shield will be sequentially put to ON. Otherwise, leds are all OFF.

Download demo video (https://wiki.axiom-project.eu/images/c/c8/Arduino_example_PIR.mp4.zip)

```c
#define DELAY_LED_ON 100
int pinPIR = 6;
int pinLED1 = 4;
int pinLED2 = 7;
int pinLED3 = 8;
int pinLED4 = 12;
int statePIR;

void setup() {
ipMode (pinPIR, INPUT);
ipMode (pinLED1, OUTPUT);
ipMode (pinLED2, OUTPUT);
ipMode (pinLED3, OUTPUT);
ipMode (pinLED4, OUTPUT);
digitalWrite (pinLED1, 0);
digitalWrite (pinLED2, 0);
digitalWrite (pinLED3, 0);
digitalWrite (pinLED4, 0);
}

void setLeds () {
digitalWrite (pinLED1, 1);
delay (DELAY_LED_ON);
digitalWrite (pinLED2, 1);
delay (DELAY_LED_ON);
digitalWrite (pinLED3, 1);
delay (DELAY_LED_ON);
digitalWrite (pinLED4, 1);
}

void clearLeds () {
digitalWrite (pinLED1, 0);
digitalWrite (pinLED2, 0);
digitalWrite (pinLED3, 0);
digitalWrite (pinLED4, 0);
}

void loop() {
static int state_old = LOW;
statePIR = !digitalRead (pinPIR);
if (statePIR == LOW) {
clearLeds ();
} else {
if (state_old != statePIR) {
setLeds ();
}
state_old = statePIR;
delay (200);}
}
```

AXIOM board with Relay Shield and PIR sensor
7 Example: SPI NOR Flash write/read

Tag:
- SPI library;
- Serial Port;
- NOR SPI Flash;

As showed into picture below, a NOR flash is connected with the AVR8 through SPI bus. The sketch read the vendor and device IDs, erase the whole memory and write the first twelve values of the Fibonacci's series. The values are read and showed via Serial Monitor.

External document

```cpp
#include <SPI.h>
byte v_id; // vendor ID
byte d_id; // device ID
byte dr, dw, state; int addr = 0x8;

void setup() { 
  SPI.begin(); 
  Serial.begin(9600); 
}

void chip_erase () { 
  SPI.begin(); 
  SPI.transfer(0x01); 
}

void get_id (byte *vendor_id, byte *device_id) { 
  SPI.begin(); 
  byte addr = 0x8; 
  void get_id(byte *vendor_id, byte *device_id) { 
    SPI.begin(); 
    Serial.begin(9600); 
}
```
```c
SPI.transfer(0x00);
*vendor_id = SPI.transfer(0x00);
*device_id = SPI.transfer(0x00);
SPI.disableChip();

void byte_write (int addr, byte data)
{ byte addr0; //LSB
 byte addr1;
 byte addr2; //MSB
 addr0 = addr & 0xFF;
 addr1 = (addr >> 8) & 0xFF;
 addr2 = (addr >> 16) & 0xFF;
 SPI.enableChip();
 SPI.transfer(0x01);
 SPI.transfer(0x00);
 SPI.disableChip();
 SPI.enableChip();
 SPI.transfer(0x06);
 SPI.disableChip();
 SPI.enableChip();
 SPI.transfer(0x02);
 SPI.transfer(addr2);
 SPI.transfer(addr1);
 SPI.transfer(addr0);
 SPI.transfer(data);
 SPI.disableChip();
}

void byte_read (int addr, byte *data)
{ byte addr0; //LSB
 byte addr1;
 byte addr2; //MSB
 addr0 = addr & 0xFF;
 addr1 = (addr >> 8) & 0xFF;
 addr2 = (addr >> 16) & 0xFF;
 SPI.enableChip();
 SPI.transfer(0x00);
 SPI.transfer(0x00);
 SPI.transfer(addr2);
 SPI.transfer(addr1);
 SPI.transfer(addr0);
 byte_write(addr2, (continue ...)

define N_WR 12

void loop() {
 int i;
 int n1 = 1;
 int n2 = 1;
 get_id(&v_id, &d_id);
 Serial.println(" ");
 Serial.println("-----------------------");
 Serial.println("Vendor ID: ");
 Serial.println(v_id, HEX);
 Serial.println("Device ID: ");
 Serial.println(d_id, HEX);
 Serial.println("-----------------------");
 Serial.println(" ");
 Serial.println("Chip erase...");
 Serial.println(" ");
 chip_erase();
 addr = 0x0;
 dw = 1;
 Serial.print("Write first ");
 Serial.print(N_WR);
 Serial.print(" of the Fibonacci's series");
 for (i = 0; i < N_WR; i++) {
   if (i > 1) {
     n2 = dw;
     dw += n1;
     n1 = n2;
   } else dw = 1;
   byte_write(addr + i, dw);
   delay(10);
 }
 Serial.println(" ");
 for (i = 0; i < N_WR; i++) {
   byte_read(addr + i, &dr);
   Serial.print("Byte read at address ");
   Serial.print(addr + i);
   Serial.print(" : ");
   Serial.println(dr);
   delay(5000);
 }
```

Schematic Arduino + SPI Flash NOR (build with Frizzing)

Screenshot of Arduino IDE (with the running program) and Serial port monitor with relative output
8 Example: Math, Trigonometry, Characters, Random Numbers, Bits and Bytes function sets

Tag:
- Math functions;
- Trigonometry functions;
- Characters functions;
- Random Numbers functions;
- Bits and Bytes functions;
- Serial Port.

The sketch simple shows the usage of all functions of the mentioned groups of the "Language Reference" of Arduino. All the results of those functions are shown via terminal emulation program, through Serial Port (in this regard, a UART-to-USB adapter was used).

```cpp
void setup() {
  Serial.begin(9600);
  Serial.println("\n");
  Serial.println("######### Math functions test #########");
  Serial.print("abs(-2) = ");
  Serial.println(abs(-2));
  Serial.print("constrain of 150 between 10 and 100 = ");
  Serial.println(constrain(150, 10, 100));
  Serial.print("constrain of 1 between 10 and 100 = ");
  Serial.println(constrain(1, 10, 100));
  Serial.print("map 100 in conversion from [0, 1023] to [0, 255] = ");
  Serial.println(map(100, 0, 1023, 0, 255));
  Serial.print("max between 10 and 100 = ");
  Serial.println(max(10, 100));
  Serial.print("min between 10 and 100 = ");
  Serial.println(min(10, 100));
  Serial.print("3 raised to power of 3 = ");
  Serial.println(pow(3, 3));
  Serial.print("square root of 25 = ");
  Serial.println(sqrt(25));
  Serial.print("'c' is lower case = ");
  Serial.println(isLowerCase('c'));
  Serial.print("'C' is lower case = ");
  Serial.println(isLowerCase('C'));
  Serial.print("'c' is upper case = ");
  Serial.println(isUpperCase('c'));
  Serial.print("'C' is upper case = ");
  Serial.println(isUpperCase('C'));
  Serial.print("'C' is punctuation = ");
  Serial.println(isPunct('C'));
  Serial.print("',' is punctuation = ");
  Serial.println(isPunct(','));
  Serial.print("'c' is the space character = ");
  Serial.println(isSpace('c'));
  Serial.print("' ' is the space character = ");
  Serial.println(isSpace(' '));
  Serial.print("' ' is a white space = ");
  Serial.println(isWhitespace(' '));
  Serial.print("'\n' is a white space = ");
  Serial.println(isWhitespace('
'));
  Serial.print("'\t' is a white space = ");
  Serial.println(isWhitespace('	'));
}
```
Serial.println(sqrt(25));
Serial.println("square of 20 = ");
Serial.println(square((double)20));
Serial.println("\n\n######### Trigonometry functions test #########");
Serial.print("cos of 0.5 rad = ");
Serial.println(cos(0.5));
Serial.print("sin of 0.5 rad = ");
Serial.println(sin(0.5));
Serial.print("tan of 0.5 rad = ");
Serial.println(tan(0.5));
Serial.println("\n\n######### Characters functions test #########");
Serial.print("'3' is an alpha = ");
Serial.println(isAlpha('3'));
Serial.print("'c' is an alpha = ");
Serial.println(isAlpha('c'));
Serial.print("'3' is an alphanumeric = ");
Serial.println(isAlphaNumeric('3'));
Serial.print("'@' is an alphanumeric = ");
Serial.println(isAlphaNumeric('@'));
Serial.print("'3' is an ASCII = ");
Serial.println(isAscii('3'));
Serial.print("' ' is a digit = ");
Serial.println(isDigit(' '));
Serial.print("' ' is a graph = ");
Serial.println(isGraph(' '));
Serial.print("' ' can be printed = ");
Serial.println(isPrintable(' '));
Serial.println(isWhitespace(' '));
Serial.println("\n\n######### Random Numbers functions test #########");
Serial.print("random value between 10 and 100 = ");
randomSeed(50);
Serial.println(random(10, 100));
Serial.println("\n\n######### Bits and Bytes functions test #########");
int val = 0xAC;
Serial.print("clear bit three of 0xAC = ");
val = 0xAC;
bitClear(val, 2);
Serial.println(val, HEX);
Serial.print("set bit two of 0xAC = ");
val = 0xAC;
bitSet(val, 1);
Serial.println(val, HEX);
Serial.print("value of bit one of 0xAC = ");
val = 0xAC;
Serial.println(bitRead(val, 0), HEX);
Serial.print("set to 1 the first bit of 0xAC = ");
val = 0xAC;
bitWrite(val, 0, 1);
Serial.println(val, HEX);
Serial.print("high byte of 0xABCD = ");
val = 0xABCD;
Serial.println(highByte(val), HEX);
Serial.print("low byte of 0xABCD = ");
val = 0xABCD;
Serial.println(lowByte(val), HEX);

void loop() {
Introduction

All needed materials are provided via remote sources. Moreover a Linux distribution is needed since all provided tools work under Linux. The tested distributions are Debian like (Ubuntu and last Debian release).

The startup procedure described below does not need any compile procedure, the download of the source code of the BSP and the toolchain is not needed.

You just need to download pre-compiled image and FileStystem, as outlined below.

Board Setting

To properly setting the board, refer to the documentation AXIOM board (https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTATION/AXIOM_board#Programming_Cable)

Getting BSP components

To obtain the materials to create a bootable BSP, refer to the documentation Getting BSP (https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTATION/BSP/ComponentsAndSources#Getting_the_pre-built_BSP)
Creating SD Card

Preparing the SD card
Steps to prepare the SD card for PetaLinux SD card ext filesystem boot:

1. The SD card is formatted with two partitions using a partition editor such as gparted or fdisk.
2. The first partition should be at least 40MB in size and formatted as a FAT32 filesystem. Ensure that there is 4MB of free space preceding the partition. The first partition will contain the bootloader, u-boot, devicetree and kernel images. Label this partition as BOOT.
3. The second partition should be formatted as an ext4 filesystem and can take up the remaining space on the SD card. This partition will store the system root filesystem. Label this partition as rootfs.

Flashing the SD card
Note: plnx-proj-root is the root folder of the pre-built AXIOM BSP.
Steps to Boot a PetaLinux Image on Hardware with SD Card:

1. Mount the SD card on your host machine.
2. Copy the following files from <plnx-proj-root>/pre-built/linux/images/ into the root directory of the first partition which is in FAT32 format in the SD card: BOOT.BIN and image.ub.
3. Extract the downloaded FileSystem archive into the second partition which is in EXT4 format in the SD card.
4. Connect the serial port on the board to your workstation.
5. Open a console on the workstation and start the preferred serial communication program (e.g. kermit, minicom, gtkterm) with the baud rate set to 115200 on that console.
6. Power off the board.
7. Set the boot mode of the board to SD boot. Refer to the board documentation for details.
8. Plug the SD card into the board.
9. Power on the board

Actually, the boot environment is working in progress. So to properly boot the system from uSD, the u-boot variables must to be changed with the following command:

```
setenv bootargs 'earlycon=cdns,mmio,0xFFF00000,115200n8 root=/dev/mmcblk1p2 rw rootwait console=ttyPS0,115200'
setenv bootcmd 'mmc dev 1 && mmcinfo && fatload mmc 1:1 ${clobstart} ${kernel_img} && bootm' saveenv
```

Note that this step is temporary until the environment configuration command (bconfig) will be released.

10. Watch the serial console, you will see the boot messages similar to the following:
SD1 Boot Mode
SD: rc= 0
File name is 1:/BOOT.BIN
Multiboot Reg : 0x0
Image Header Table Offset 0x8C0
*****Image Header Table Details*******
Boot Gen Ver: 0x1020000
No of Partitions: 0x5
Partition Header Address: 0x280
Partition Present Device: 0x0
Initialization Success
====== In Stage 3, Partition No:1 =======
UnEncrypted data Length: 0x65216F
Data word offset: 0x65216F
Total Data word length: 0x65216F
Destination Load Address: 0xFFFFFFFF
Execution Address: 0x0
Data word offset: 0x7B00
Partition Attributes: 0x20
Destination Device is PL, changing LoadAddress
Bitstream download to start now
DMA transfer done
PL Configuration done successfully
Partition 1 Load Success
====== In Stage 3, Partition No:2 =======
UnEncrypted data Length: 0x1800
Data word offset: 0x1800
Total Data word length: 0x1800
Destination Load Address: 0xFFFFE000
Execution Address: 0xFFFFE000
Data word offset: 0x659C70
Partition Attributes: 0x107
Partition 2 Load Success
====== In Stage 3, Partition No:3 =======
UnEncrypted data Length: 0x8
Data word offset: 0x8
Total Data word length: 0x8
Destination Load Address: 0xFFFF0000
Execution Address: 0x0
Data word offset: 0x65B470
Partition Attributes: 0x107
Partition 3 Load Success
====== In Stage 3, Partition No:4 =======
UnEncrypted data Length: 0x1FF8C
Data word offset: 0x1FF8C
Total Data word length: 0x1FF8C
Destination Load Address: 0x8000000
Execution Address: 0x8000000
Data word offset: 0x65B480
Partition Attributes: 0x104
Partition 4 Load Success
All Partitions Loaded
========== In Stage 4 ===========
Protection configuration applied
# ATF running on XCZU9EG/silicon v3/RTL5.1 at 0xffe000
NOTICE: BL31: Secure code at 0x0
NOTICE: BL31: Non secure code at 0x8000000
NOTICE: BL31: v1.2(release):5d8a33b
NOTICE: BL31: Built : 08:56:19, Feb 24 2017

U-Boot 2016.07-gac66543-dirty (Feb 24 2017 - 08:57:14 +0100) AXIOM ZynqMP ZU9EG

I2C:  ready
DRAM: 4 GiB
EL Level:  EL2
Chip ID:  xczuunknown
MMC:  sdhci@ff160000: 0, sdhci@ff170000: 1
SF: Detected N25Q512A with page size 512 Bytes, erase size 128 KiB, total 128 MiB
In:  serial
Out:  serial
Err:  serial
Net:  ZYNQ GEM: ff0b0000, phyaddr 0, interface rgmii-id
eth0: ethernet@ff0b0000
Hit any key to stop autoboot:

Ubuntu 16.04 LTS secolnx01 ttyPS0
secolnx01 login:
1 Notice

This page provides the description of a sub-set of the PetaLinux's commands, useful to the usage of the AXIOM BSP. For more information, please refer to Xilinx documents about Petalinux 3016.3.

2 Introduction

PetaLinux Tools provide a simple and fast method to build and deploy Linux-based software on Xilinx Zynq UltraScale+ MPSoC and MicroBlaze devices. With PetaLinux, you can:

- Build and configure Linux components; Linux kernel, u-boot and file system, for Zynq UltraScale+ MPSoC and MicroBlaze based designs
- Easily synchronize your Linux software platform and hardware platform in a single step
- Easily target and migrate your Linux user application to Zynq UltraScale+ MPSoC or MicroBlaze based Linux platform
  Quickly get started with building Linux-based software on Xilinx and partner development boards using the pre-built Board Support Packages (BSPs)
- Test your Zynq UltraScale+ MPSoC or MicroBlaze Linux system without any hardware in a virtual machine environment using QEMU
3 Download Installation Package


(N.B. The used Petalinux installation package is also available into AXIOM server: downlaod link (https://wiki.axiom-project.eu/index.php/WP6_ARCHITECTURE_IMPLEMENTATION/BSP/ComponentsAndSources) - requires user access -)

4 Prerequisites

This section lists the requirements for the PetaLinux Tools Installation:

- Minimum workstation requirements:
  - 4 GB RAM (recommended minimum for Xilinx tools)
  - Pentium 4 2GHz CPU clock or equivalent
  - 20 GB free HDD space
  - Supported OS:
    - RHEL 6.6/6.7/7.1/7.2 (64-bit)
    - CentOS 7.1 (64-bit)
    - SUSE Enterprise 12.0 (64-bit)
    - Ubuntu 14.04.3 (64 bit)

• You need to have root access to perform some operations. • PetaLinux requires a number of standard development tools and libraries to be installed on your Linux host workstation. Please install the libraries and tools listed in the following table on your host Linux. You must install the appropriate 32-bit compatible libraries due to some tools such as toolchains are 32bit executables.

The table below describes the required packages, and how to install them on different Linux workstation environments.
<table>
<thead>
<tr>
<th>Tool/Library</th>
<th>CentOS 7.1</th>
<th>RHEL 6.6</th>
<th>RHEL 6.7</th>
<th>RHEL 7.2</th>
<th>SuSE 12.0</th>
<th>RHEL 7.2</th>
<th>Ubuntu 14.04</th>
</tr>
</thead>
<tbody>
<tr>
<td>dos2unix</td>
<td>dos2unix 6.0.3</td>
<td>dos2unix 3.1-37</td>
<td>dos2unix 3.1-37</td>
<td>dos2unix 6.0.3</td>
<td>dos2unix 6.0.3</td>
<td>dos2unix 6.0.4</td>
<td>tofrods 1.7.13</td>
</tr>
<tr>
<td>ip</td>
<td>iproute 3.10.0</td>
<td>iproute 2.6.32</td>
<td>iproute 2.6.32</td>
<td>iproute 3.10.0</td>
<td>iproute 3.10.0</td>
<td>iproute 2</td>
<td>iproute2</td>
</tr>
<tr>
<td>gawk</td>
<td>gawk 4.0.2</td>
<td>gawk 3.1.7</td>
<td>gawk 3.1.7</td>
<td>gawk 4.0.2</td>
<td>gawk 4.0.2</td>
<td>gawk 4.1.0</td>
<td>gawk 4.0.1</td>
</tr>
<tr>
<td>gcc</td>
<td>gcc 4.8.3</td>
<td>gcc 4.4.7</td>
<td>gcc 4.4.7</td>
<td>gcc 4.8.3</td>
<td>gcc 4.8.5</td>
<td>gcc 4.8</td>
<td>gcc 4.8</td>
</tr>
<tr>
<td>git</td>
<td>git 1.8.3</td>
<td>git 1.7.1</td>
<td>git 1.7.1</td>
<td>git 1.8.3</td>
<td>git 1.8.3</td>
<td>git 1.7.1 or above</td>
<td>git 1.7.1 or above</td>
</tr>
<tr>
<td>make</td>
<td>make 3.81</td>
<td>make 3.81</td>
<td>make 3.81</td>
<td>make 3.82</td>
<td>make 3.82</td>
<td>make 4.0</td>
<td>make 3.81</td>
</tr>
<tr>
<td>netstat</td>
<td>net-tools 2.0</td>
<td>net-tools 1.60</td>
<td>net-tools 1.60</td>
<td>net-tools 2.0</td>
<td>net-tools 2.0</td>
<td>net-tools</td>
<td>net-tools</td>
</tr>
<tr>
<td>ncurses-devel</td>
<td>ncurses-devel 5.9-13</td>
<td>ncurses-devel 5.7-3</td>
<td>ncurses-devel 5.7-4</td>
<td>ncurses-devel 5.9-13</td>
<td>ncurses-devel 5.9-13</td>
<td>ncurses-devel</td>
<td>libncurses5-dev</td>
</tr>
<tr>
<td>tftp server</td>
<td>tftp-server</td>
<td>tftp-server</td>
<td>tftp-server</td>
<td>tftp-server</td>
<td>tftp-server</td>
<td>atftp or yast2-tftp-server</td>
<td>tftpd</td>
</tr>
<tr>
<td>zlib-devel</td>
<td>zlib-devel 1.2.7</td>
<td>zlib-devel 1.2.3</td>
<td>zlib-devel 1.2.3</td>
<td>zlib-devel 1.2.7</td>
<td>zlib-devel 1.2</td>
<td>zlib-devel 1.2</td>
<td>zlib-devel</td>
</tr>
<tr>
<td>openssl-devel</td>
<td>openssl-devel 1.0</td>
<td>openssl-devel 1.0</td>
<td>openssl-devel 1.0</td>
<td>openssl-devel 1.0</td>
<td>openssl-devel 1.0</td>
<td>libopenssl-devel</td>
<td>libssl-dev</td>
</tr>
<tr>
<td>flex</td>
<td>flex 2.5.37</td>
<td>flex 2.5.35</td>
<td>flex 2.5.35</td>
<td>flex 2.5.37</td>
<td>flex 2.5.37</td>
<td>flex</td>
<td>flex</td>
</tr>
<tr>
<td>bison</td>
<td>bison-2.7</td>
<td>bison-2.4.1</td>
<td>bison-2.4.1</td>
<td>bison-2.7.4</td>
<td>bison-2.7.4</td>
<td>bison</td>
<td>bison</td>
</tr>
<tr>
<td>libselinux</td>
<td>libselinux 2.2.2</td>
<td>libselinux 2.0.94</td>
<td>libselinux 2.0.94</td>
<td>libselinux 2.2.2</td>
<td>libselinux 2.2.2</td>
<td>libselinux 2.3.2</td>
<td>libselinux1</td>
</tr>
</tbody>
</table>

5 Installation Tool

PetaLinux Tool has not a specific installation folder. You can select the target folder during the installation procedure. In fact, the install accepts as argument the root installation folder.
Without any options, the installer will install as a subdirectory of the current directory.

E.g. To specify the "/opt/" folder as installation folder:
N.B. Since the initial checking procedure, performed by the install will take you many time, please be sure to have the writing right for the specified (or current) installation folder.

This is the output of the installation procedure:

```
$ mkdir /opt/petalinux-v2016.3
$ ./petalinux-v2016.3-final-installer.run /opt/petalinux-v2016.3
INFO: Checking installer checksum...
INFO: Extracting PetaLinux installer...
INFO: Installing PetaLinux...
INFO: Checking PetaLinux integrity...
INFO: Extracting Installation files...

LICENSE AGREEMENTS

PetaLinux SDK contains software from a number of sources. Please review the following licenses and indicate your acceptance of each to continue.

You do not have to accept the licenses, however if you do not then you may not use PetaLinux SDK.

Use PgUp/PgDn to navigate the license viewer, and press 'q' to close

Press Enter to display the license agreements
Do you accept Xilinx End User License Agreement? [y/N] > y
Do you accept Webtalk Terms and Conditions? [y/N] > y
Do you accept Third Party End User License Agreement? [y/N] > y
INFO: Checking installation environment requirements...
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
INFO: Installing PetaLinux SDK to "/opt/petalinux-v2016.3/.
INFO: PetaLinux SDK has been installed to /opt/petalinux-v2016.3/.
INFO: Installing PetaLinux Yocto SDK to "/opt/petalinux-v2016.3/tools/yocto-sdk"
Build tools installer version 2016.3
===============================================================================
You are about to install the SDK to "/opt/petalinux-v2016.3/tools/yocto-sdk". Proceed[Y/n] Y
Extracting SDK.......done
Setting it up...done
INFO: PetaLinux Yocto SDK has been successfully installed.
```
6 Getting to use Petalinux

Before each usage of the tool, you have to run the setup script in order to have the right environment. This last describes the folder (with absolute path) of the single runnable command used by the tool. In the following, the single main commands will be described.

```
$ cd /opt/petalinux-v2016.3
$. ./settings.sh
PetaLinux environment set to '/opt/petalinux-v2016.3' WARNING: /bin/sh is not bash!
bash is PetaLinux recommended shell. Please set your default shell to bash.
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
```

This is the environment added with the execution of the script above:

```
$ export | grep -i petalinux
microblaze/bin:/opt/petalinux-v2016.3"
ddclare -x PETALINUX="/opt/petalinux-v2016.3"
ddclare -x PETALINUX_VER="2016.3"
ddclare -x PWD="/opt/petalinux-v2016.3"
```

N.B. The new environment is volatile: it is present only for the current session. To manage the BSP in various terminal sessions, you have to run the same script for each of them.

7 PetaLinux commands

Follow a brief description of used command tool.

petalinux-config Tool

Allows the user to customize the specified project. It is used for two main purposes:
 initialize (or update) the project to reflect the specified hardware
- configuration customize the single components of the BSP (kernel, u-boot, ...)

The table below details the main available options for the petalinux-config tool.

<table>
<thead>
<tr>
<th>Option</th>
<th>Function Description</th>
<th>Value Range</th>
<th>default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--get-hw-description PATH</td>
<td>Initialize or update the hardware configuration for the PetaLinux project. Mutually exclusive with -c.</td>
<td>user-specified</td>
<td>none</td>
</tr>
<tr>
<td>-c,--component COMPONENT</td>
<td>Configured the specified system component. Mutually exclusive with --get-hw-description.</td>
<td>none kernel rootfs</td>
<td>none</td>
</tr>
</tbody>
</table>

N.B. The components affected by this process may include FSBL configuration, U-Boot options, Linux kernel options, and the Linux device tree configuration.

petalinux-package Tool

Packages a PetaLinux project into a format suitable for deployment. The petalinux-package tool is executed using the package type name to specify a specific workflow in the format petalinux-package --PACKAGETYPE.

- The boot package type creates a file (.BIN or .MCS) that allows the target device to boot.
- The bsp package type creates a .bsp file which includes the entire contents of the target PetaLinux project.
- The firmware package type creates a .tar.gz file which includes the needed files to update a PROM device on a board which has already been configured. This package format is only compatible with the upgrade-firmware PetaLinux demonstration application.
- The pre-built package type creates a new directory within the target PetaLinux project called "pre-built" and contains pre-built content that is useful for booting directly on a physical board. This package type is commonly used as a precursor for creating a bsp package type.

By default, the petalinux-package tool loads default files from the `<plnx-proj-root>/images/linux/` directory.

The table below details the command line options that are common to all of the petalinux-package workflows.

<table>
<thead>
<tr>
<th>Option</th>
<th>Function Description</th>
<th>Value Range</th>
<th>default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p,--project PROJECT</td>
<td>OPTIONAL. PetaLinux project directory path.</td>
<td>user-specified</td>
<td>current directory</td>
</tr>
</tbody>
</table>
Details for the petalinux-package --boot

The petalinux-package --boot workflow generates a bootable image that can be used directly with a Zynq family device (e.g. bootable format is BOOT.BIN which can be booted from an SD card). The table below details the options that are valid when creating a bootable image with the petalinux-package --boot workflow.

<table>
<thead>
<tr>
<th>Option</th>
<th>Function Description</th>
<th>Value Range</th>
<th>default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--get-hw-description PATH</td>
<td>Initialize or update the hardware configuration for the PetaLinux project. Mutually exclusive with -c.</td>
<td>user-specified</td>
<td></td>
</tr>
<tr>
<td>--format FORMAT</td>
<td>REQUIRED. Image file format to generate.</td>
<td>BIN, MCS</td>
<td>BIN</td>
</tr>
<tr>
<td>--fsbl FSBL</td>
<td>REQUIRED. Path on disk to FSBL elf binary.</td>
<td>user-specified</td>
<td>zynqmp_fsbl.elf for Zynq UltraScale+ MPSoC</td>
</tr>
<tr>
<td>--force</td>
<td>OPTIONAL. Overwrite existing files on disk.</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>--fpga BITSTREAM</td>
<td>OPTIONAL. Path on disk to bitstream file.</td>
<td>user-specified</td>
<td>none</td>
</tr>
<tr>
<td>--atf ATF-IMG</td>
<td>OPTIONAL. Path on disk to ARM trusted firmware elf binary.</td>
<td>user-specified</td>
<td>&lt;plnx-projroot&gt;/images/linux/bl31.elf</td>
</tr>
<tr>
<td>--uboot UBOOT-IMG</td>
<td>OPTIONAL. Path on disk to U-Boot binary. It is U-Boot ELF for Zynq family device</td>
<td>user-specified</td>
<td>u-boot.elf for Zynq family device; The default image is in &lt;plnx-proj-root&gt;/images/linux</td>
</tr>
<tr>
<td>--kernel KERNEL-IMG</td>
<td>OPTIONAL. Path on disk to Linux Kernel image.</td>
<td>user-specified</td>
<td>&lt;plnx-projroot&gt;/images/linux/image.ub</td>
</tr>
<tr>
<td>--pmufw PMUFW-ELF</td>
<td>OPTIONAL. Only for Zynq UltraScale+ MPSoC. Path on disk to PMU firmware image.</td>
<td>user-specified</td>
<td>&lt;plnx-proj-root&gt;/pre-built/linux/images/pmufw.elf</td>
</tr>
<tr>
<td>--add DATAFILE</td>
<td>OPTIONAL. Path on disk to arbitrary data to include.</td>
<td>user-specified</td>
<td>none</td>
</tr>
</tbody>
</table>

Details for the petalinux-package --bsp

Compiles all contents of the specified PetaLinux project directory into a BSP file with the provided file name. This .bsp file can be distributed and later used as a source for creating a new PetaLinux project. This workflow is generally used as the last step in producing a project image that can be distributed to other users. All Xilinx reference BSP's for PetaLinux are packaged using this workflow.
The table below details the options that are valid when packaging a PetaLinux BSP file with the `petalinuxpackage --bsp` workflow.

<table>
<thead>
<tr>
<th>Option</th>
<th>Function Description</th>
<th>Value Range</th>
<th>default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-o, --output BSPNAME</td>
<td>REQUIRED. Path on disk to store the BSP file. File name will be of the form BSPNAME.bsp.</td>
<td>user-specified</td>
<td>current directory</td>
</tr>
<tr>
<td>-p, --project PROJECT</td>
<td>OPTIONAL. PetaLinux project directory path. In the BSP context, multiple project areas can be referenced and included in the output BSP file.</td>
<td>user-specified</td>
<td>current directory</td>
</tr>
<tr>
<td>--force</td>
<td>OPTIONAL. Overwrite existing files on disk.</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>--clean</td>
<td>OPTIONAL. Clean the hardware implementation results to reduce package size.</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>--hwsource HWPROJECT</td>
<td>OPTIONAL. Path to a Vivado project to include in the BSP file.</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>--no-extern</td>
<td>OPTIONAL. Exclude components external to the project referenced using the --searchpath option. This may prevent the BSP from building for other users.</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>--no-local</td>
<td>OPTIONAL. Exclude components referenced in the local PetaLinux project. This may prevent the BSP from building for other users.</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

Details for the `petalinux-package --image` Package image for component. You can use it to generate uImage for kernel.

The table below details the options that are valid when packaging a PetaLinux firmware image with the `petalinux-package --image` workflow.

<table>
<thead>
<tr>
<th>Option</th>
<th>Function Description</th>
<th>Value Range</th>
<th>default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p, --project PROJECT</td>
<td>OPTIONAL. PetaLinux project directory path.</td>
<td>user-specified</td>
<td>current directory</td>
</tr>
<tr>
<td>-c, --component COMPONENT</td>
<td>OPTIONAL. PetaLinux project component.</td>
<td>user-specified</td>
<td>kernel, rootfs</td>
</tr>
<tr>
<td>--format FORMAT</td>
<td>OPTIONAL. Image format. It relies on the component.</td>
<td>user-specified</td>
<td>kernel: image.ub, Image for Zynq UltraScale+ MPSoC</td>
</tr>
</tbody>
</table>

Details for the `petalinux-package --firmware` Creates a firmware update package based on the specified PetaLinux project. The firmware package allows the user to selectively update components of a deployed system. This package may contain components such as U-Boot, the Linux kernel, a Linux device tree, or a Linux root filesystem.

The table below details the options that are valid when packaging a PetaLinux firmware image with the `petalinux-package --firmware` workflow.
<table>
<thead>
<tr>
<th>Option</th>
<th>Function Description</th>
<th>Value Range</th>
<th>default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-o, --output PACKAGENAME</td>
<td>OPTIONAL. Full path and name on disk to store the firmware image.</td>
<td>user-specified</td>
<td>firmware.tar.gz</td>
</tr>
<tr>
<td>-p, --project PROJECT</td>
<td>OPTIONAL. PetaLinux project directory path.</td>
<td>user-specified</td>
<td>current directory</td>
</tr>
<tr>
<td>--format FORMAT</td>
<td>OPTIONAL. Image format. It relies on the component.</td>
<td>user-specified</td>
<td>kernel: image.ub, Image for Zynq UltraScale+ MPSoC</td>
</tr>
<tr>
<td>--linux UBIMAGE</td>
<td>OPTIONAL. Update the Linux kernel partition with the specified UBIMAGE.</td>
<td>none</td>
<td>image.ub</td>
</tr>
<tr>
<td>--dtb DTBFILE</td>
<td>OPTIONAL. Update the device tree DTB partition with the specified DTBFILE.</td>
<td>none</td>
<td>system.dtb</td>
</tr>
<tr>
<td>--fpga BITSTREAM</td>
<td>OPTIONAL. Update the FPGA bitstream partition with the specified BITSTREAM.</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>--u-boot UBOOT-S</td>
<td>OPTIONAL. Update the U-Boot binary partition with the specified UBOOT-S binary.</td>
<td>none</td>
<td>u-boot-s.bin</td>
</tr>
<tr>
<td>-a, --add dev:file</td>
<td>OPTIONAL. Update the flash partition named dev with the file specified by file. This option can be repeated multiple times.</td>
<td>user-specified</td>
<td>none</td>
</tr>
<tr>
<td>--flash FLASHTYPE</td>
<td>OPTIONAL. Specify the type of flash device with which the image is compatible.</td>
<td>spi</td>
<td>parallel</td>
</tr>
<tr>
<td>---data-width SIZE</td>
<td>OPTIONAL. Specify the bit width of the data bus for the target parallel flash device.</td>
<td>8 16 32</td>
<td>none</td>
</tr>
<tr>
<td>--pre SCRIPT</td>
<td>OPTIONAL. Specify a SCRIPT that should be run on the target platform prior to updating the flash partitions.</td>
<td>user-specified</td>
<td>none</td>
</tr>
</tbody>
</table>

Details for the petalinux-package --prebuilt: Creates a new directory named "pre-built" inside the directory hierarchy of the specified PetaLinux project. This directory contains the required files to facilitate booting a board immediately without completely rebuilding the project. This workflow is intended for users who will later create a PetaLinux BSP file for distribution using the petalinux-package --bsp workflow. The table below details the options that are valid when including pre-built data in the project with the petalinuxpackage --prebuilt workflow.
<table>
<thead>
<tr>
<th>Option</th>
<th>Function Description</th>
<th>Value Range</th>
<th>default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p,--project PROJECT</td>
<td>OPTIONAL. PetaLinux project directory path.</td>
<td>user-specified</td>
<td>current directory</td>
</tr>
<tr>
<td>--force</td>
<td>OPTIONAL. Overwrite existing files on disk.</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>--clean</td>
<td>OPTIONAL. Remove all files from the &lt;plnx-proj-root&gt;/prebuilt directory.</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>--fpga BITSTREAM</td>
<td>OPTIONAL. Include the BITSTREAM file in the prebuilt directory.</td>
<td>user-specified</td>
<td>none</td>
</tr>
<tr>
<td>-a,--add src:dest</td>
<td>OPTIONAL. Add the file/directory specified by src to the directory specified by dest in the pre-built directory.</td>
<td>user-specified</td>
<td>none</td>
</tr>
</tbody>
</table>

Other tool

PetaLinux contains other command, which are not described in this guide because are out of the scope of the AXIOM BSP usage. The commands are:

- petalinux-create
- petalinux-util
  - petalinux-util --gdb
  - petalinux-util --xsdb-connect
  - petalinux-util --jtag-logbuf
  - petalinux-util --update-sdcard
  - petalinux-util --webtalk
1 Software Requirements Specification

- Tool name:
  - xgenimage
- Interface:
  - Command line with options
- Description:
  - Generic tool to automate the image creation process
- Programming Language:
  - Bash Shell Scripting as main language, plus external tools like: debootstrap
- Compatibility:
  - The tool must be compatible with Ubuntu 16.04, others distributions are a plus
- INPUT:
  - Import all image characteristics from a configuration file,
    see this as reference: https://git-private.axiom-project.eu/genautoinstall/blob/master/ks.cfg
The most important parameters are:
- Linux Kernel Image (binary)
- Bootloader Image (binary)
- Partition size definition (boot + rootfs)
- package list (example: openssh-server, libopenmpi, etc.)
- custom deb packages (for example OmpSs and all evidence axiom packages)
- Variantis: desktop, minimal
- all operating sistem configuration: default user, timezone, keyboard settings etc.

OUTPUT:
- sd-card image ready to flash (dd if=axiom.img of=/dev/mmcblock bs=1M)
- rootfs partition (partition is better, .tar is dangerous)

EXTRA:
- Is a good idea provide a very small sd-card image and at first startup make an automatic partition resize to fill all remaining space in sd-card.

2 Download

The tool is available under git repository (https://git-private.axiom-project.eu/xgenimage/)
To obtain it:

```bash
$ git clone https://git-private.axiom-project.eu/xgenimage/
```

3 Workflow

The main task are:

<table>
<thead>
<tr>
<th>Task</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>(REQUIRED) Set the tool with the default configuration</td>
</tr>
<tr>
<td>Configuration</td>
<td>(OPTIONAL) User configuration manipulation</td>
</tr>
<tr>
<td>Image creation</td>
<td>(FINAL TASK) creation of bootable SD image</td>
</tr>
</tbody>
</table>
4 Folder tree structure

- **bin**
  - conf
  - mconf

- **libs**
  - bsp_list.tcl
  - bsp.sh
  - configure.sh
  - debootstrap.sh
  - debug.sh
  - environment.tcl
  - image.sh
  - rootfs.sh
  - utility.sh

- **README.md**
- **srs_v02.txt**

- **store**
  - local
    - readme
    - configs
      - axiomUbuntu_defconfig
  - rootfs
    - package_install.list
    - patch_post_install
    - patch_pre_install
  - Xconfig
  - Xconfig_Filesystem
  - Xconfig_Images
  - Xconfig_BSP
  - Xconfig_Output

- **work**
  - bootfs
  - images
  - logs
  - rootfs
  - xgenimage.sh

Binary file. Here is stored the config management tool

Framework: set of functions used by the tool for:
- User interface (configuration and output information)
- Image file creation, formatting and mounting
- File System building
- File System patching
- BSP installation

Folder which contains local file and downloaded file:
- Set of saved configuration file
- Used BSP, downloaded from remote server

Folder which contains file used for the File System building (after Debootstrap):
- Patches to apply before the package installation
- List of package to install
- Patches to apply after the package installation

Set of Kconfig used to generate the config management environment

Folder used to store results of tool execution

Main script

5 Usage overview

This is a command line tool. Each main task has the own argument set:

1. `./xgenimage --usage`
6 Logging

For each image creation procedure a log file is created into the log folder: `work/logs/`. The log file contain all the information, warning and error divided step by step. At the start of the procedure the name of relative log file is mentioned:

```
# ./xgenimage.sh -I
* Create BSP configuration file... OK
* Created new log file: ./work/logs/xgenimage_log_20171024_1259.log
```
7 Configuration initialization
The default configuration file are stored into the folder: xgenimage/store/local/configs All file are name with “_defconfig” as suffix. With the command --defconfig, the tool automatically search a file into this folder with name passed as argument (without _defconfig suffix). The main output of this command is the file .config store into the root folder of the tool. This file is used as current configuration file. All user configuration selection are booked into this file.

To use the current official configuration:

```bash
./xgenimage.sh --defconfig=axiom_ubuntu
```

With the execution of this command, some file are created:

- Store/Xconfig_BSP : Kconfig file for the list of available BSP revisions, as mentioned before.
- Store/env.sh : script used to export end unset the current configuration environment, used during the image creation procedure.

8 Import/export configuration

It is possible to export the current setting in a new default configuration file:

```bash
./xgenimage.sh --exportconfig=my_config
```

The file my_config will be stored into the root folder of the tool.

9 Clean

Interested files:

- Configuration file
- All file created at run-time

Command:

```bash
./xgenimage.sh --cleanall
```
10 Configuration

The configuration is managed via menuconfig interface. To access to it:

```
# ./xgenimage.sh --config
```

### Main menu

<table>
<thead>
<tr>
<th>Item</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Output</td>
<td>All the options that involve the output image file</td>
</tr>
<tr>
<td>Image Setup</td>
<td>Source of the BSP package (remote or local)</td>
</tr>
<tr>
<td>BSP package path</td>
<td>Local source path of the BSP location</td>
</tr>
<tr>
<td>File System sources</td>
<td>Source of the base Ubuntu File System</td>
</tr>
<tr>
<td>File System Setup</td>
<td>Ubuntu setting to apply after the installation into the root partition</td>
</tr>
<tr>
<td>File System checking</td>
<td>Set of item about the File System integrity checking operation</td>
</tr>
</tbody>
</table>
### Image Output

<table>
<thead>
<tr>
<th>Item</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image file name</td>
<td>name of the output file (default: <code>axiom_image</code>)</td>
</tr>
</tbody>
</table>

### Image setup

<table>
<thead>
<tr>
<th>Item</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of the boot partition</td>
<td>size, in MB, of the boot partition (default: 64MB)</td>
</tr>
<tr>
<td>Label of the boot partition</td>
<td>label of boot partition (default: <code>bootfs</code>)</td>
</tr>
<tr>
<td>Size of the root partition</td>
<td>size, in MB, of the root partition (default: 4096MB)</td>
</tr>
<tr>
<td>Label of the root partition</td>
<td>label of root partition (default: <code>rootfs</code>)</td>
</tr>
<tr>
<td>Use data #1 partition</td>
<td>Create a generic data partition after the rootfs partition (default: <code>diable</code>)</td>
</tr>
<tr>
<td></td>
<td>If select, allows setting of name and size of this partition</td>
</tr>
<tr>
<td>Use data #2 partition</td>
<td>Create a generic second data partition after the data1 partition (default: <code>diable</code>)</td>
</tr>
<tr>
<td></td>
<td>If select, allows setting of name and size of this partition</td>
</tr>
</tbody>
</table>
### BSP source

<table>
<thead>
<tr>
<th>Item</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSP package form local folder</td>
<td>local source for the BSP package (default: <em>enable</em></td>
</tr>
<tr>
<td>BSP package form remote server</td>
<td>Remote source for the BSP package (default: <em>disable</em></td>
</tr>
</tbody>
</table>

### BSP location

<table>
<thead>
<tr>
<th>Item</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSP package path</td>
<td>local source for the BSP package (default: <em>./axiom_bsp.bsp</em></td>
</tr>
<tr>
<td>release</td>
<td>release to use of the BSP present into the server (default: <em>release 1.0</em></td>
</tr>
</tbody>
</table>
### File System sources

<table>
<thead>
<tr>
<th>Item</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base File System from local archive</td>
<td>use local archive as base for the File System building (default: disable)</td>
</tr>
<tr>
<td>Base File System from remote archive</td>
<td>use remote archive as base for the File System building (default: enable)</td>
</tr>
<tr>
<td>Base File System from debootstrap procedure</td>
<td>use full debootstrap (first and second stage) for the File System building (default: disable)</td>
</tr>
</tbody>
</table>

### File System setup

<table>
<thead>
<tr>
<th>Item</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board's hostname</td>
<td>system hostname (default: axiom)</td>
</tr>
<tr>
<td>Account user name</td>
<td>name of the default user (default: ubuntu)</td>
</tr>
<tr>
<td>Account user password</td>
<td>password of the default user (default: ubuntu)</td>
</tr>
<tr>
<td>Root password access</td>
<td>password of the system administrator (default: root)</td>
</tr>
</tbody>
</table>
Image creation

This procedure is divided in tasks:

- Download of the BSP at specified revision;
- Creation of an empty image file of size and name specified by configuration environment;
- Partitioning and formatting;
- Bootfs and Rootfs partition mounting;
- Install BSP into Bootfs partition;
- Perform first and second stage into Rootfs partition;
- Set the user access (specified by configuration environment);
- Apply patches “pre-installation”
- Install packages listened into a file
- Apply patches “post-installation”
- Unmount both Bootfs and Rootfs partitions

- the image file is now available for SD device flashing -

At the end of each tasks there is a check test to verify the good completion of the task. For each execution of this procedure a log file is created

Command:

```bash
# ./xgenimage.sh --full-image
```

Patches

All patches are grouped in two folder:

- patch_pre_install : all patched which must be applied after the debootstrap steps and before the installation of the list of packages;
- patch_post_install : all patched which must be applied after the installation of the list of packages.

The patches are intended to be applied in no chroot mode and in order to make this application more generic as possible a standard format of them is used:

1. Each patch must be included in a folder with name composed by four digits as prefix and label separated by character ‘_’;
2. In each folder must be present a file to patch (or install);
3. In each folder must be present a text file named “patch”, with the follow structure:

```bash
###DESCRIPTION###
<brief description>
###DESCRIPTION###
###FILE###
<file, object of the patch>
###FILE###
```
With:

- **<brief description>:** Description printed into the log during the execution;
- **<file, object of the patch>:** File name to patch or to install;
- **<command>:** Command to perform. Use `@FILE` to refer to the file reported above and `@ROOTFS` to refer to the rootfs folder (mount point of the partition image)

The four digits are used to create a sequence into the application of the patch.

Actual patches:

**patch_post_install:**

```
0000_apt-source_list
  └── patch
      └── sources.list

0001_apt-no-recommands
  └── patch
      └── sources.list
```

**patch_post_install:**

```
0000_gpu_link
  └── gpu_link.tar.gz
  └── patch

0001_xorg_driver
  └── armsoC_drv.so
  └── patch

0002_xorg_conf
  └── patch
      └── xorg.conf

0003_slim_conf
  └── patch
      └── slim.conf

0004_wallpaper
  └── AXIOM_InfoGraphic_Comp.jpg
  └── patch
```
1 Introduction

Simple set of bash scripts used to acquire data about power consumption.
The tool is provided as deb package (actual version: 2.0).
Download: archive (https://wiki.axiom-project.eu/images/4/4d/Pwrm-plot_2.0.zip)
This contains both deb package and archive of all file.

2 Installation

If the package 1.0 is already installed into the working system, place remove it:

```bash
dpkg --remove pwrm-plot1.0
```

To install the newest version:

```bash
dpkg -i pwrm-plot_2.0.deb
```
3 Placement

The script to launch is placed, by default, into the Desktop folder:

```
/home/ubuntu/Desktop/hwmon_scan_all.sh
```

4 Working and Output

The script creates eight files (one for each power monitor) to store the acquired data. These data are composed by:

- sampling time
- Current consumption (in mA)
- Power consumption (in mW)

and are presented in tabular form in order to allow the importing into spreadsheet software (e.g. Microsoft Excel), as CSV file. Each file has a standard name:

```
rail_id_<name>.dat
```

where:

- id: is the id of the power monitor (from 0 to 7);
- name: is the name of the power monitor

Example of file output:

```
root@axiom:/home/ubuntu/Desktop# cat rail_0_PM_VCC_INTFP.dat
Time(ms)  Current(mA)  Power(mW)
135   1677    1380
438   1665    1380
711   1619    1320
999   1696    1380
1288  1673    1380
1570  1670    1380
1881  1675    1340
2156  1691    1340
2452  1636    1340
```
5 Change Parameters

Available parameters:

- Sampling time (in msec)
- Sampling duration (in sec)
- folder to store the output file

To change these parameters the user have to edit the script file and set the follow variables:

- STIME for the sampling duration;
- LTIME for the sampling time;
- FOLDER for the placement of the output file
**UBOOT/EnvironmentConfiguration**

<table>
<thead>
<tr>
<th></th>
<th>SPI</th>
<th>eMMC</th>
<th>uSD</th>
<th>USB</th>
<th>TFTP</th>
<th>NFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>image.ub</td>
<td>ToDo</td>
<td>ToDo</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FileSystem</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>ToDo</td>
<td></td>
</tr>
</tbody>
</table>
1 Introduction
The board has eight INA219 chips for monitoring current and power. These chips are connected with the Zynq Ultrascale+ via I2C and at each of it is assigned a kernel driver for the user access to the consumption data. In fact, this driver provides both SYSFS and IOCTL interfaces.

2 Hardware References
The following rail are monitored:

<table>
<thead>
<tr>
<th>Bus</th>
<th>Label</th>
<th>Nominal Voltage (V)</th>
<th>System ID (*)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.85V_INTFP</td>
<td>PM_VCC_INTFP</td>
<td>0.85</td>
<td>0</td>
<td>PS full-power domain supply voltage</td>
</tr>
<tr>
<td>0.85V_VCCINT</td>
<td>PM_VCC_INT</td>
<td>0.85</td>
<td>1</td>
<td>PL internal power supply</td>
</tr>
<tr>
<td>12V_ALW</td>
<td>PM_VIN</td>
<td>12</td>
<td>2</td>
<td>VIN power supply</td>
</tr>
<tr>
<td>0.85V_INTFP_DDR</td>
<td>PM_INTFP_DDR</td>
<td>0.85</td>
<td>3</td>
<td>PS DDR controller and PHY supply voltage</td>
</tr>
<tr>
<td>1V2_DDR_PS</td>
<td>PM_1V2_DDR_PS</td>
<td>1.2</td>
<td>4</td>
<td>PS DDR supply</td>
</tr>
<tr>
<td>1.2V_DDR_PL</td>
<td>PM_1V2_DDR_PL</td>
<td>1.2</td>
<td>5</td>
<td>PL DDR supply</td>
</tr>
<tr>
<td>MGTAVCC</td>
<td>PM_MGTAVCC</td>
<td>0.9</td>
<td>6</td>
<td>Analog supply voltage for GTH transceiver</td>
</tr>
<tr>
<td>1.2V_MGTAVTT</td>
<td>PM_MGTAVTT</td>
<td>1.2</td>
<td>7</td>
<td>Analog supply voltage for GTH transmitter and receiver termination circuits</td>
</tr>
</tbody>
</table>

(*) The System ID is a conventional zero based number used to uniquely identify the chip. This number is used also as suffix of the device file descriptor assigned at each driver instance.
All the eight chip are connected to the Zynq Ultrascale+ through the bus I2C0 of the processor and the addresses from 0x40 to 0x47 are assigned to them. The user can see this at File System console with the i2c-tool:

```
root@secolnx01:~# i2cdetect -y -r 0
  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  UU UU UU UU UU UU UU UU --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
root@secolnx01:~#
```

**Compensation**

For each chip there is an RC input filter circuit. For this reason the relative driver has compensation value to neglected the filter effect. This value is represented in ‰.

**NB:** The compensation parameter has an initial value reported into the DTS file of the kernel source code. This value has been evaluated through external measurements (as reported in 8.5.2.1 Calibration Register and Scaling of the datasheet) and the changing of it could lead to incorrect evaluation of power and power by the chip.
3 SYSFS Interface

As mentioned before, each chip has an own SYSFS interface. In that folder there are all file description needed to access to consumption data. All these file are read-only, with the exception of the 'compensation' (for more info about the compensation, please see above).

<table>
<thead>
<tr>
<th>Bus Name</th>
<th>SYSFS Folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM_VCC_INTFP</td>
<td>/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0040/hwmon/hwmon0</td>
</tr>
<tr>
<td>PM_VCC_INT</td>
<td>/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0041/hwmon/hwmon1</td>
</tr>
<tr>
<td>PM_VIN</td>
<td>/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0042/hwmon/hwmon2</td>
</tr>
<tr>
<td>PM_INTFP_DDR</td>
<td>/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0043/hwmon/hwmon3</td>
</tr>
<tr>
<td>PM_1V2_DDR_PS</td>
<td>/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0044/hwmon/hwmon4</td>
</tr>
<tr>
<td>PM_1V2_DDR_PL</td>
<td>/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0045/hwmon/hwmon5</td>
</tr>
<tr>
<td>PM_MGTAVCC</td>
<td>/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0046/hwmon/hwmon6</td>
</tr>
<tr>
<td>PM_MGTAVTT</td>
<td>/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0047/hwmon/hwmon7</td>
</tr>
</tbody>
</table>

In each SYSFS folder there are the following file descriptor:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus_name</td>
<td>r</td>
<td>Name of the bus of the board under measurement</td>
</tr>
<tr>
<td>shunt_resistor</td>
<td>r</td>
<td>Value in mOhm of the shunt resistor</td>
</tr>
<tr>
<td>in0_input</td>
<td>r</td>
<td>Shunt voltage measurement data (in mV)</td>
</tr>
<tr>
<td>in1_input</td>
<td>r</td>
<td>Bus voltage measurement data (in mV)</td>
</tr>
<tr>
<td>curr1_input</td>
<td>r</td>
<td>Current (in mA) flowing into the shunt resistor</td>
</tr>
<tr>
<td>power1_input</td>
<td>r</td>
<td>Power (in mW) supplied to the load over the bus</td>
</tr>
<tr>
<td>compensation</td>
<td>r/w</td>
<td>Compensation value (in %) to delete the load of the filter</td>
</tr>
<tr>
<td>dump_regs</td>
<td>r</td>
<td>Raw value list of all chip registers</td>
</tr>
</tbody>
</table>
Below is reported an example in which we read the power consumption on the PM_VIN:

```
ubuntu@secolnx01:~$ su
Password:
root@secolnx01:~# cd /sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0042/hwmon/hwmon2
root@secolnx01:/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0042/hwmon/hwmon2# ll
```

```
total 0
drwxr-xr-x 2 root root 0 Feb 11 16:28 ./
drwxr-xr-x 3 root root 0 Feb 11 16:28 ../
-r-r--r-- 1 root 4096 Feb 11 16:54 bus_name
-rw-r--r-- 1 root 4096 Feb 11 16:54 compensation
-r-r--r-- 1 root 4096 Feb 11 16:54 curr1_input
lrwxrwxrwx 1 root 0 Feb 11 16:54 device -> ../../../0-0-0042/
-r-r--r-- 1 root 4096 Feb 11 16:54 dump_regs
-r-r--r-- 1 root 4096 Feb 11 16:54 in0_input
-r-r--r-- 1 root 4096 Feb 11 16:54 in1_input
-r-r--r-- 1 root 4096 Feb 11 16:54 name
lrwxrwxrwx 1 root 0 Feb 11 16:54 of_node -> ../../../firmware/devicetree/base/amba/i2c@ff020000/ina219@42/
-r-r--r-- 1 root 4096 Feb 11 16:54 power1_input
```

```
lrwxrwxrwx 1 root 0 Feb 11 16:54 bus_name
```

```
rwxrwxrwx 1 root 0 Feb 11 16:54 of_node -> ../../../firmware/devicetree/base/amba/i2c@ff020000/ina219@42/
```

```
root@secolnx01:/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0042/hwmon/hwmon2# cat bus_name
PM_VIN
```

```
root@secolnx01:/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0042/hwmon/hwmon2# cat power1_input
5240 # in mW
```

4 IOCTL Interface

For each instance of the driver relative to the INA219 chip, a device file descriptor is created into the /dev/ folder. Its name is ina219.X where X is the System ID assigned to the bus (for the combination bus/ID, please see the table in Hardware References).

Follow the kind of accesses allowed from the driver:

<table>
<thead>
<tr>
<th>IOCTL Programming Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flag</strong></td>
</tr>
<tr>
<td>INA2XX_BUS_NAME_GET</td>
</tr>
<tr>
<td>INA2XX_BUS_VOLTAGE_GET</td>
</tr>
<tr>
<td>INA2XX_SHUNT_VOLTAGE_GET</td>
</tr>
<tr>
<td>INA2XX_CURRENT_GET</td>
</tr>
<tr>
<td>INA2XX_POWER_GET</td>
</tr>
<tr>
<td>INA2XX_CALIBRATION GET</td>
</tr>
</tbody>
</table>
Example of IOCTL access to read the bus name of the driver with ID=2:

```c
#define PM_BUS_NAME_LEN 128

char fname_device[64];
char bus_name[PM_BUS_NAME_LEN];
struct stat buf;
int file = 0;
int ret = 0;

fname = "/dev/ina219.2"

if ( stat (fname_device, &buf) ) {
    // file does not exist
    return -20;
}

file = open (fname_device, O_RDWR);
if ( file < 0 ) {
    // no file open
    access return -errno;
}

ret = ioctl (file, INA2XX_BUS_NAME_GET, bus_name);

close (file);
```
The AXIOM-board

PART TWO - Software guide

Intro

This part contains instruction on how to get, install, and run the AXIOM NIC drivers and user-space applications developed for the AXIOM board.
The main components of the AXIOM infrastructure are the following (see the figure above):

- **The AXIOM NIC**: Implemented in the FPGA by partner FORTH. The AXIOM NIC exports a set of registers to the computing part of the FPGA. These registers are described in the Datasheet, that can be found in the Documentation.
- **Axiom NIC Device driver**: Implements the kernel-related part of the infrastructure, which is responsible for a proper handling of the AXIOM NIC registers, and which is responsible for providing abstractions for communication ports, an IOCTL interface to the user space, and for extending the HW buffering space in memory thanks to the usage of kernel threads.
- **Axiom User Library**: The AXIOM User Library is responsible for providing a comprehensive C API, which can be used by the AXIOM applications to interact with the network interface in a simpler way.

The main features of the AXIOM NIC are the following:

- **Interrupt moderation**: The AXIOM NIC only generates interrupts when the queue status has changed. Link status monitoring: The AXIOM NIC is able to provide information about the fact that a specific interface of the board is connected or not to another board.
- **Routing decision based on a Routing Table**: The routing of messages in the AXIOM NIC is based on a store & forward technique, where each node maintains a routing table in the FPGA, holding the information about the interface to use to reach another node.
- **Multiple type of messages**: The network interface is able to concurrently send two kind of messages, small and big messages.
- **Multiple queues available**: The AXIOM NIC provides 2 set of queues for reasons of efficiency to separate message (one for small and one for big messages)

During the design of the network infrastructure, we took special care about defining a complete system that could be efficiently mapped on the parallel programming libraries running in user space. For this reason, we followed these guidelines during the implementation of the AXIOM Drivers and during the design of the AXIOM NIC registers:

- **Network management in user space**: Compared to the standard TCP/IP Linux stack, we tried to move various features and network support in user space, limiting the AXIOM NIC driver to its main task of handling the delivery of the packets between the user space applications and the AXIOM NIC registers. Possibility to support for memory mapped registers exposed to the user space: We left open the possibility to use memory mapped registers to avoid the use of IOCTL on each packet transfer, and to reduce memory copies. On the other hand, this requires the availability of separate per-process network queues to limit the overhead that is linked to mutual exclusion on the NIC register sets.
- **Possibility to use the NIC as tunnel for ethernet packets**: The idea is that the long messages will be used as datagrams, giving the possibility for TCP/IP packet to be sent on top of the AXIOM NIC layers. This has the advantage to reduce cabling between the boards (only the AXIOM Link is needed), allowing legacy applications based on TCP/IP to run unmodified on top of the AXIOM-link network.

How to get the software

To use AXIOM SW Stack on the AXIOM board:

- you can #Download the SD for the AXIOM board
- you can #Clone the repository and compile the packages
To use AXIOM SW Stack on the AXIOM QEMU simulator (up to axiom-v0.13):

- you can #Download_the_VDI_image_-_QEMU_sim you can #Download_the_tarball_-_QEMU_sim
- you can #Clone_the_repository_and_compiles_deb_packages

**Download the SD for the AXIOM board**

The following links contain the SD for the AXIOM board with the AXIOM SW Stack already installed: [SD image with axiom-v1.2](https://upload.axiom-project.eu/uploads/WP7/SDIMAGE/XOS_v1.1_20180319.tar.bz2) [SD image with axiom-v1.1](https://upload.axiom-project.eu/uploads/WP7/SDIMAGE/XOS_v1.0_20180315.tar.bz2) [SD image with axiom-v1.0](https://upload.axiom-project.eu/uploads/WP7/SDIMAGE/XOS_v0.9_20180208.tar.bz2) [SD image with axiom-v0.15](https://upload.axiom-project.eu/uploads/WP7/SDIMAGE/XOS_v0.3_20171113.tar.bz2) [SD image with axiom-v0.14](https://upload.axiom-project.eu/uploads/WP5/axiom-nic/20170911_axiom_evi_sd.img.7z)

**Download the DEBs, kernel and bitstream for the AXIOM board**

The following links contain the DEBs of the AXIOM SW Stack, and the archive with kernel and bitstream for the AXIOM board: [AXIOM SW Stack v1.0 DEBs](https://upload.axiom-project.eu/uploads/WP5/axiom-nic/axiom-evi-debs-v1.0.7z) [Kernel and bitstream for AXIOM SW Stack v1.0](https://upload.axiom-project.eu/uploads/WP5/axiom-nic/axiom-evi-boot-v1.0.7z) [AXIOM SW Stack v0.15 DEBs](https://upload.axiom-project.eu/uploads/WP5/axiom-nic/axiom-evi-debs-v0.15.7z) [Kernel and bitstream for AXIOM SW Stack v0.15](https://upload.axiom-project.eu/uploads/WP5/axiom-nic/axiom-evi-boot-v0.15.7z)

**Clone the repository and compiles deb packages**

```bash
# clone the repository
`git clone https://git.axiom-project.eu/axiom-evi cd axiom-evi`

# init the submodules (may take awhile)
`./scripts/submodules_update.sh`

# read the README (https://git.axiom-project.eu/axiom-evi/blob/master/README) with the instructions to compile
```
How to run the AXIOM utilities and tests

During the boot of AXIOM boards, all daemons are automatically started through a systemd service. You only need to:

- connect USB-C cables between boards
  - you can use any of the four USB-C port
  - cables are bi-directional, so one cable is enough to connect two boards
  - boot all boards
- access to shell of one board (using serial port, or ethernet cable or keyboard and monitor)
- start discovery of other nodes in the cluster

Terminal:
```
sudo axiom-make-master.sh
```

- at the end will be printed the network topology and the routing table of the master node (node where axiom-make-master.sh is launched)

  **Notes:**
  - Node 0 is not assigned in the cluster
  - IF 0 is an internal interface used to send loopback messages
  - In the topology table, 255 means cable disconnected
  - In the routing table, 0 means node is not reachable with specified IF

<table>
<thead>
<tr>
<th>Node</th>
<th>IF0</th>
<th>IF1</th>
<th>IF2</th>
<th>IF3</th>
<th>IF4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>1</td>
<td>255</td>
<td>255</td>
<td>2</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>2</td>
<td>255</td>
<td>1</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>IF0</th>
<th>IF1</th>
<th>IF2</th>
<th>IF3</th>
<th>IF4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

IP: 192.168.17.0/24
Other nodes are reachable with IP: 192.168.17.NODEID
Local IP is 192.168.17.1

The following links explain how to run tests and utilities (must be root to run all AXIOM application):

- axiom-* application (https://git.axiom-project.eu/axiom-evi-apps/README)
- AXIOM tests [[1] (https://git.axiom-project.eu/axiom-evi-apps/blob/master/tests/README)]
- OmpSS tests [[2] (https://git.axiom-project.eu/axiom-evi/blob/master/tests/ompss/README)]
- GASNet tests [[3] (https://git.axiom-project.eu/axiom-evi/blob/master/tests/gasnet/README)]

To compile the software in the repository follow this link [[4] (https://git.axiom-project.eu/axiom-evi/blob/master/README)]