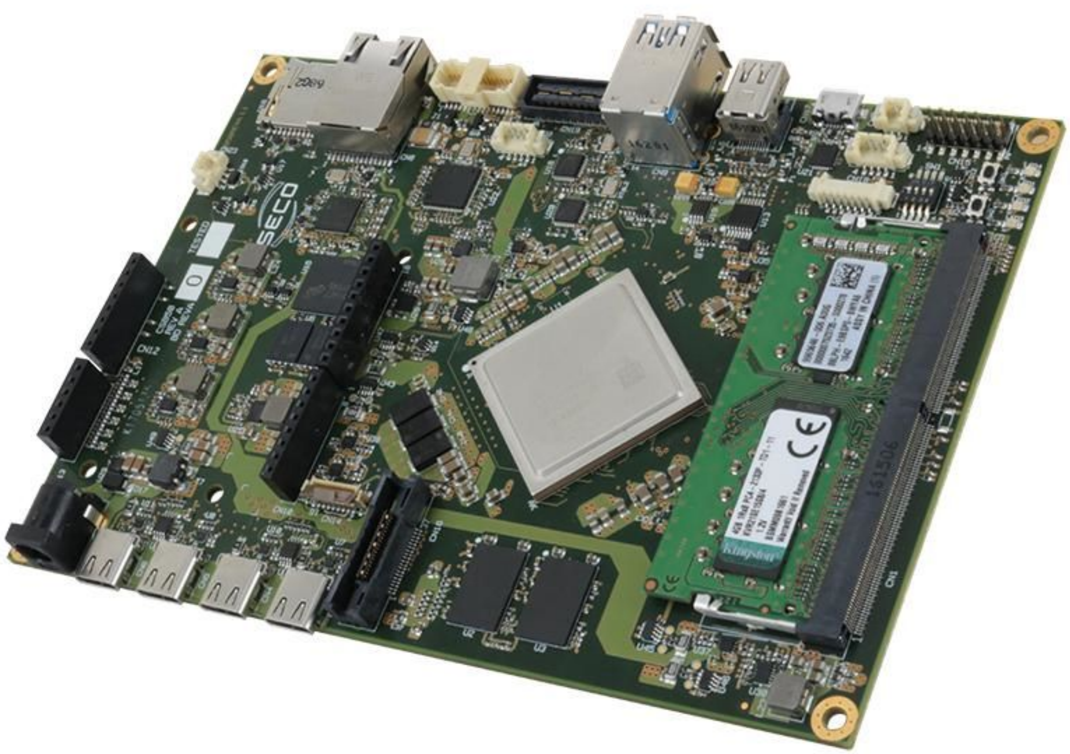
The AXIOM-board

PART ONE - Hardware guide



Contents

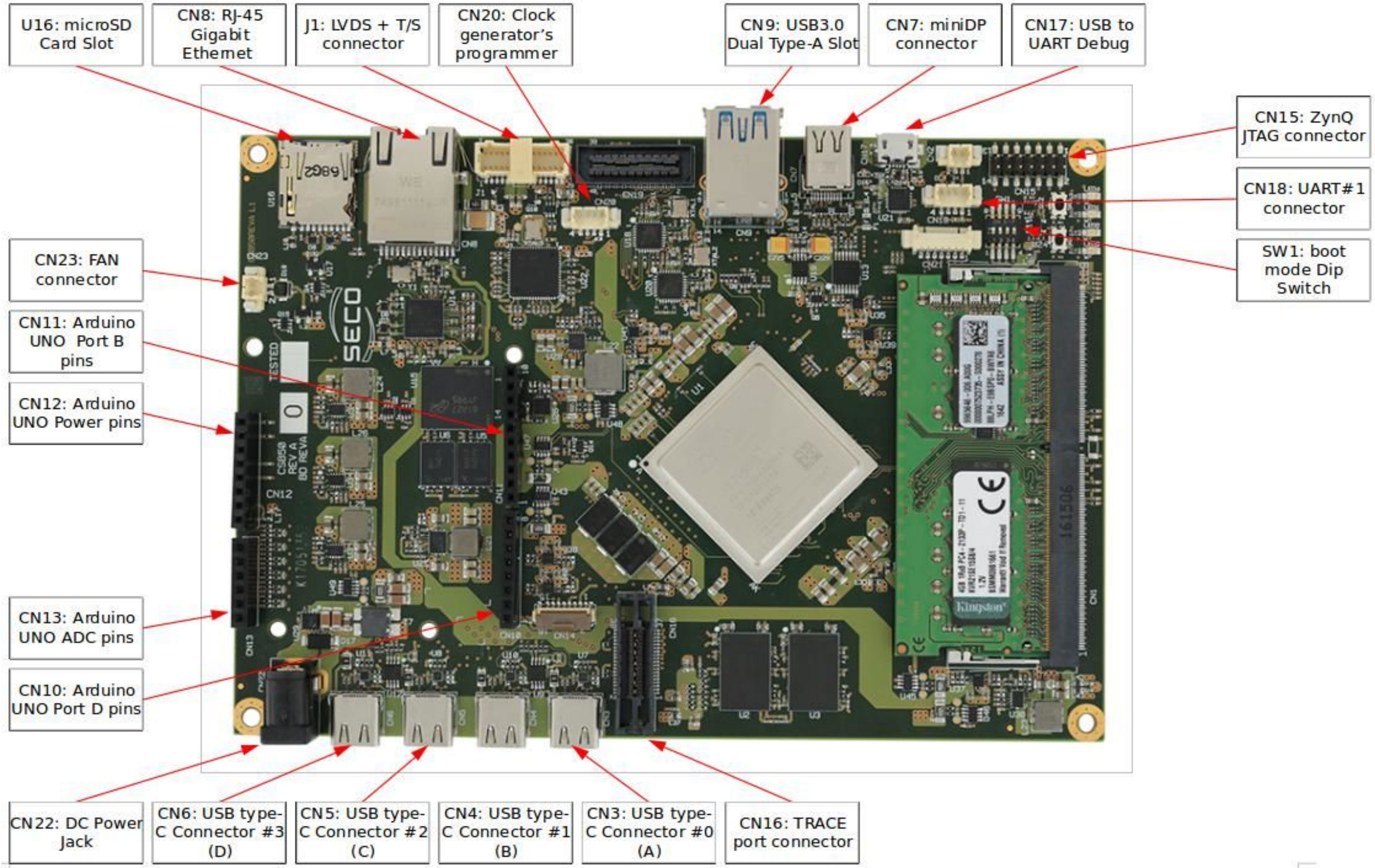
1 Introduction

The AXIOM_ZU9EG board is based on the Zynq® UltraScale+™ MPSoC XCZU9EG-1FFC900.

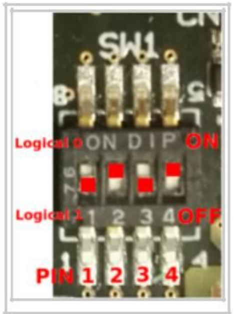
2 Overview

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)

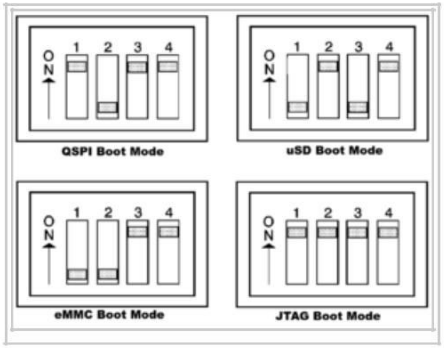


3 Configuration



Boot Mode	Mode Pins [0:3]			
	1	2	3	4
QSPI32	0	1	0	0
uSD	1	0	1	0
eMMC	1	1	0	0
JTAG	0	0	0	0

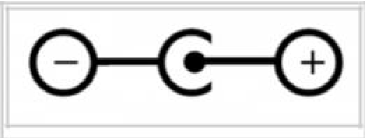
NOTE: logical value 0 refers to ON state. The switch state reported into the picture corresponds to SD boot mode.



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

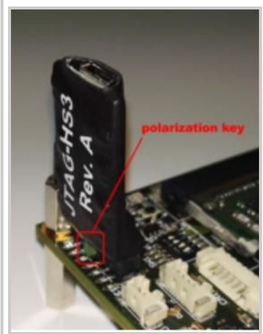
4 Power supply



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 compat ible one: JTAG-HS3 from digilent ([2] (http://stor e.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).



RIGHT connection


WRONG connection

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.

Chip Part Number: INA219 (http://www.ti.com/lit/ds/symlink/ina219.pdf)

The following rails are monitored:

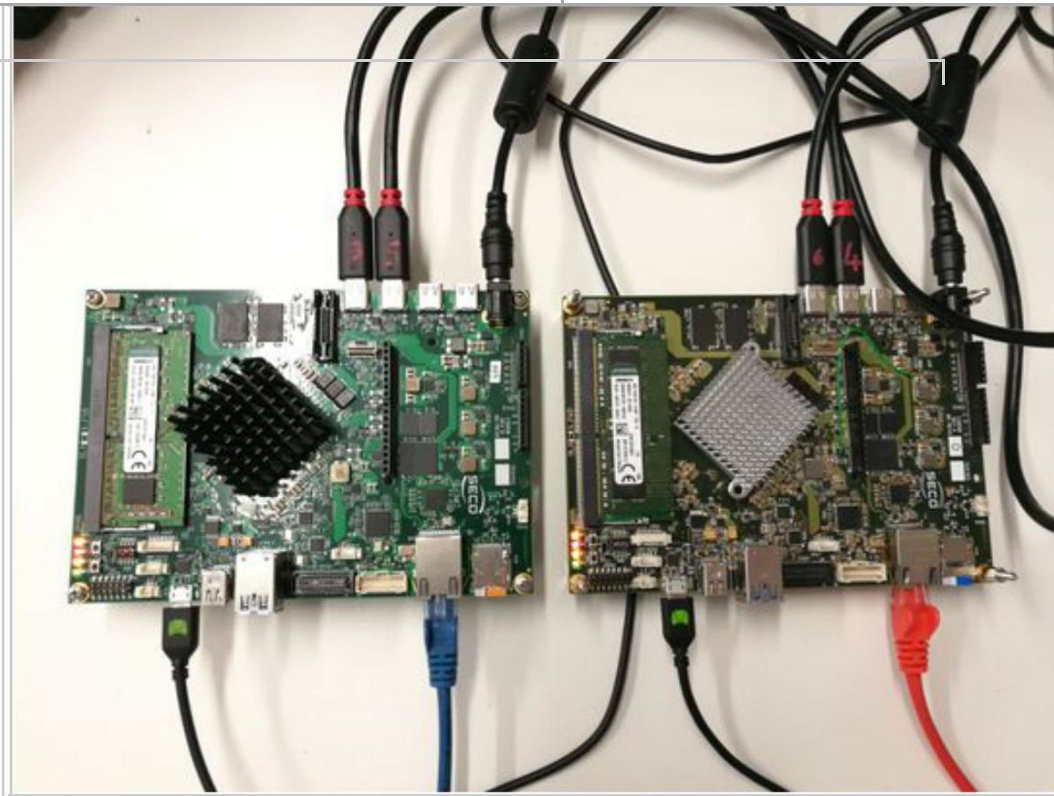
Bus	Label	Nominal Voltage (V)	System ID (*)	Description
0.85V_INTFP	PM_VCC_INTFP	0.85	0	PS full-power domain supply voltage
0.85V_VCCINT	PM_VCC_INT	0.85	1	PL internal power supply
12V_ALW	PM_VIN	12	2	VIN power supply
0.85V_INTFP_DDR	PM_INTFP_DDR	0.85	3	PS DDR controller and PHY supply voltage
1V2_DDR_PS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

(*) 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

OS Image:	XOS_v0.3_20171113
SW test:	hwmon_scan_all.sh

CLIENT	SERVER



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
                connceted = 0
                tx-enabled = 0
                rx-enabled = 1

        interfaces[1] status: 0x01
                connceted = 0
                tx-enabled = 1
                rx-enabled = 0

        interfaces[2] status: 0x02
                connceted = 0
                tx-enabled = 0
                rx-enabled = 1

        interfaces[3] status: 0x01
                connceted = 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
                connceted = 0
                tx-enabled = 0
                rx-enabled = 1

        interfaces[1] status: 0x01
                connceted = 0
                tx-enabled = 1
                rx-enabled = 0

        interfaces[2] status: 0x02
                connceted = 0
                tx-enabled = 0
                rx-enabled = 1

        interfaces[3] status: 0x01
                connceted = 0
                tx-enabled = 1
                rx-enabled = 0
```

Vivado/TestIBERT

Board setup

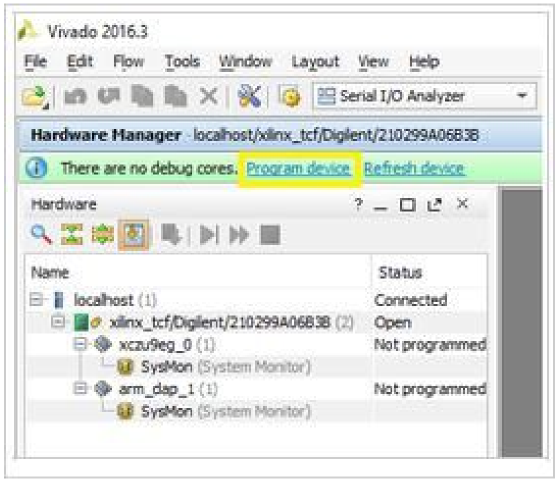
1. Connect USB Type C cables on CN3-CN4-CN5-CN6 (it's important to use USB Type C Gen 2 cables that have both lanes wired);
2. Select JTAG boot mode
3. Connect JTAG_HS3

Running the test

1. Open Hardware manager from Vivado;

2 Select Open target;

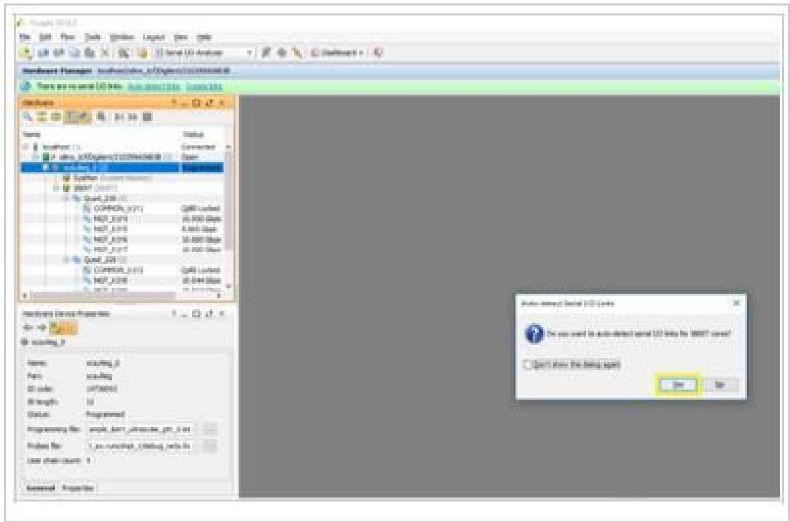
3 Select Program device end then xczu9eg_0;



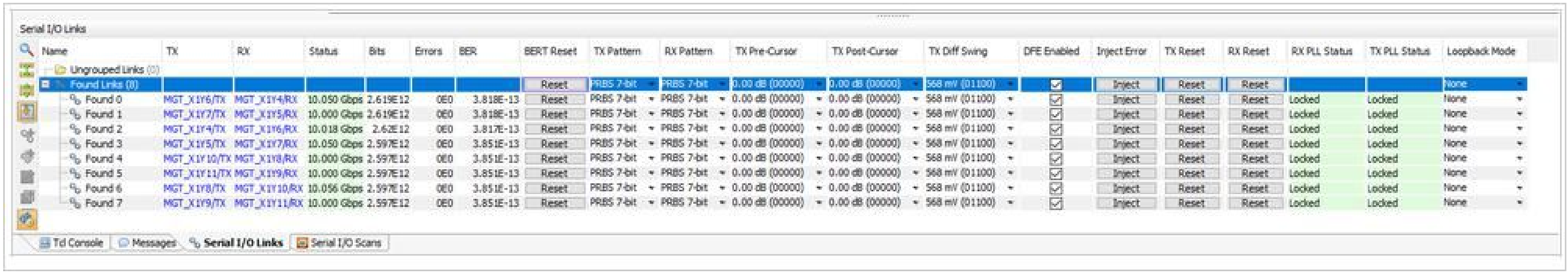
4 Choose bitstream file from your PC

```
($PC_folder\AXIOM_ZU9EG_IBERT\ibert_ultrascale_gth_0_10Gbps_228_229_refclk125\ibert_ultrascale_gth_0_ex.runs\impl_1\example_ibert_ultrascale_gth_0.bit)
```

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



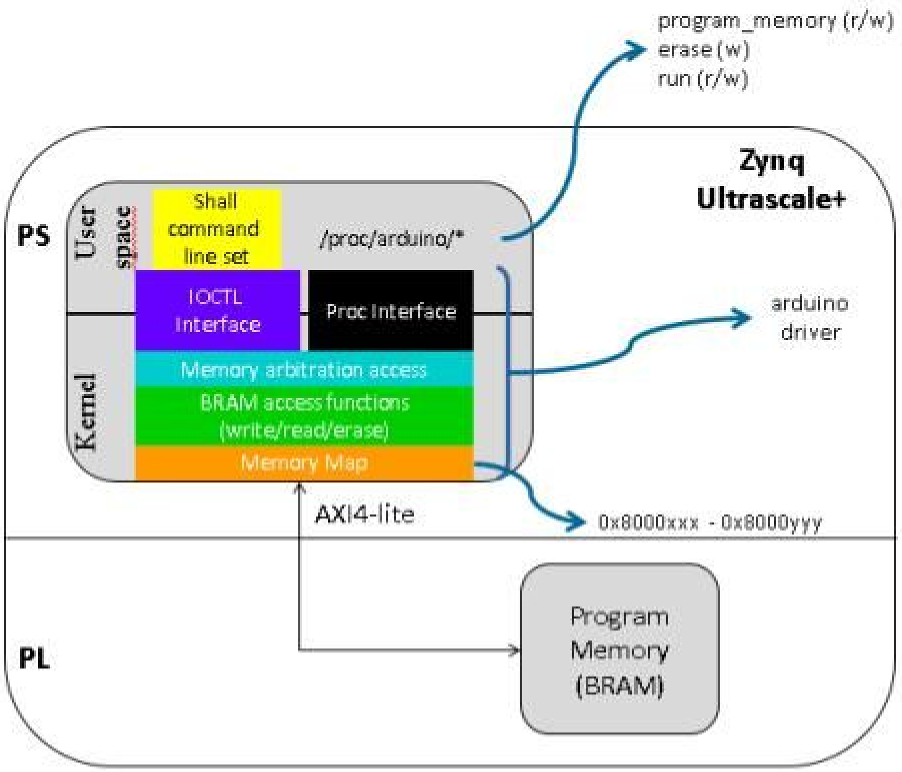
6. So, in Serial I/O Links panel you can see the status of transmission;



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:

PROC User Interface		
File Name	Access	Description
erase	r	Erase the AVR program memory
program_memory	r/w	Program the AVR program memory with the data straming from an hex file. Allows also the dumping of the memory.
run	r/w	Control/Show the state of the execution

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:

```
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:

```
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 e xecution state and at the end of the operation, it automatically rester the current state.

Example

Example: memory dump

```
root@axiom:/proc/arduino# cat program_memory
==================================== Arduino Program Memory =======================================
      |    0     1     2     3     4     5     6     7     8     9     A     B     C     D     E     F
--------------------------------------------------------------------------------------------------
 0000  | 940C   0031 940C  005F  940C  005F  940C  005F  940C  005F  940C  005F  940C  005F  940C  005F
 0010  | 940C   005F 940C  005F  940C  005F  940C  005F  940C  005F  940C  005F  940C  005F  940C  005F
 0020  | 940C   007F 940C  005F  940C  005F  940C  005F  940C  005F  940C  005F  940C  005F  940C  005F
 0030  | 0193   2411 BE1F  EFCF  E0DF  BFDE  BFCD  E010  E6A0  E0B0  ECE2  E0F6  EF0F  9503  BF0B  C004
 0040  | 95D8   920D 9631  F3C8  36AA  07B1  F7C9  E021  E6AA  E0B0  C001  921D  30A9  07B2  F7E1  E010
 0050  | E3C1   E0D0 C005  9721  2FFD  2FEC  940E  0354  33C0  07D1  F7C1  940E  01B8  940C  035F  940C
 0060  | 0000   E840 E255  E060  E070  E785  E090  940C  0144  9180  0060  940E  0116  9390  006B  9380
 0070  | 006A   E04A E050  2F68  2F79  E785  E090  940E  02D8  EF64  E071  E080  E090  940C  00D5  921F
 0080  | 920F   B60F 920F  2411  932F  933F  938F  939F  93AF  93BF  9180  006D  9190  006E  91A0  006F
 0090  | 91B0   0070 9130  006C  E023  0F23  372D  F420  9601  1DA1  1DB1  C005  E826  0F23  9602  1DA1
 00A0  | 1DB1   9320 006C  9380  006D  9390  006E  93A0  006F  93B0  0070  9180  0071  9190  0072  91A0
 00B0  | 0073   91B0 0074  9601  1DA1  1DB1  9380  0071  9390  0072  93A0  0073  93B0  0074  91BF  91AF
 00C0  | 919F   918F 913F  912F  900F  BE0F  900F  901F  9518  B72F  94F8  9160  006D  9170  006E  9180
 00D0  | 006F   9190 0070  BF2F  9508  928F  929F  92AF  92BF  92CF  92DF  92EF  92FF  2EC6  2ED7  2EE8
 00E0 |  2EF9  940E  00C9  2E86  2E97  2EA8  2EB9  940E  00C9  2FB9  2FA8  2F97  2F86  1988  0999  09AA
 ..................
 1FE0 |  0000   0000 0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
 1FF0 |  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
==================================================================================================
```

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

```
root@axiom:/proc/arduino#cat code.cpp.hex > program_memory
```

Example: put execution running

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

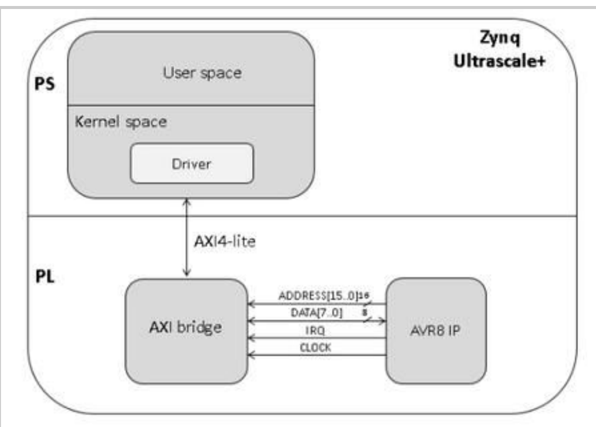
Example: erase program memory

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

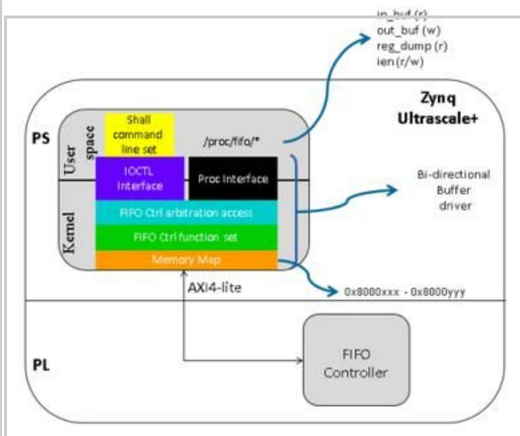
AVR8 system communication

Arduino Soft Core to Processing System communication

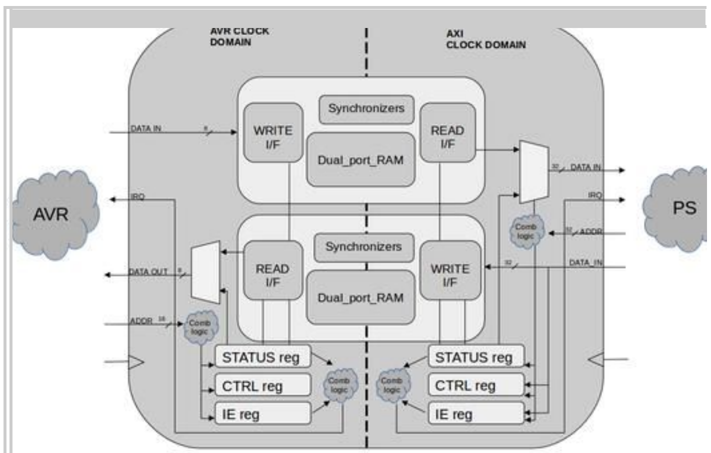
Beyond AVR processor programming, it is useful to provide a mechanism to allow communication between CPU complex integrated in Zynq's Processing system (i.e., the ARM cores) and the AVR Soft Core embedded in Zynq's Programmable Logic (Figure 5). The communication between user application and Arduino 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.



Asynchronous communication with the Arduino IP



Protocol management with the Arduino IP



Internal scheme of communication block

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 program mable logic
Set of to write/read to/from the bi-direction 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:

Name	Address	Description
Rx_reg	0x1FF0	Provides data from PS
Tx_reg	0x1FF1	Sends data to PS
INT_status_reg	0x1FF2	Interrupt status register
Control_reg	0x1FF3	Control register
Mask_reg	0x1FF4	Interrupt enable register

Register Mapping:

INT_status Register			
Bits	Name	Access	Description
7-4	N/A	N/A	N/A
3	Rxsmit_ready_IN_buffer	ROC	IN_buffer has data to provide
2	Rno_more_data_IN-buffer	ROC	IN_buffer no more data to provide
1	Wbuffer_busy_OUT_buffer	ROC	OUT_buffer busy
0	Wbuffer_empty_OUT_buffer	ROC	OUT_buffer empty

Control Register			
Bits	Name	Access	Description
7-3	N/A	N/A	N/A
2	Wbuffer_flush_OUT_buffer	WO	OUT_buffer data flush
1	Wbuffer_en_OUT_buffer	WO	OUT_buffer write enable
0	Rbuffer_en_IN_buffer	WO	IN_buffer read enable

Mash Register			
Bits	Name	Access	Description
7-4	N/A	N/A	N/A
3	Rxsmit_ready_IN_buffer	WO	Enable flag of relative interrupt
2	Rno_more_busy_OUT_buffer	WO	Enable flag of relative interrupt
1	Wbuffer_busy_OUT_buffer	WO	Enable flag of relative interrupt
0	Wbuffer_empty_OUT_buffer	WO	Enable flag of relative interrupt

Arduino Interface: In order to use the communication system an Arduino library is provided.
Library name: PSComm
Heder to include: include "PSCOMMClass.h"

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

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

Name	Address	Description
Rx_reg	BASE_ADDR + 0x00	Provides data from PS
Tx_reg	BASE_ADDR + 0x04	Sends data to PS
INT_status_reg	BASE_ADDR + 0x08	Interrupt status register
Control_reg	BASE_ADDR + 0x0c	Control register
Mask_reg	BASE_ADDR + 0x10	Interrupt enable register

Register Mapping:

INT_status Register			
Bits	Name	Access	Description
31-4	N/A	N/A	N/A
3	Rxsmit_ready_OUT_buffer	ROC	OUT_buffer has data to provide
2	N/A	N/A	N/A
1	Wbuffer_busy_IN_buffer	ROC	IN_buffer busy
0	N/A	N/A	N/A

Control Register			
Bits	Name	Access	Description
31-1	N/A	N/A	N/A
0	Wbuffer_flush_IN_buffer	WO	IN_buffer data flush

Mash Register			
Bits	Name	Access	Description
31-4	N/A	N/A	N/A
3	Rxsmit_ready_OUT_buffer	WO	Enable flag of relative interrupt
2	N/A	N/A	N/A
1	Wbuffer_busy_IN_buffer	WO	Enable flag of relative interrupt
0	N/A	N/A	N/A

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

Example:

- Dump registers

```
root@axiom:/proc/fifo# cat regs_dump
============ FIFO REGS ============
Control : 0x00000000
Inter. en: 0x00000000
Int state: 0x00000000
TX data : 0x00000000
```

- Write data:

```
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

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

Function	Status	Note
Digital I/O		
digitalRead()	Supported	
digitalWrite()	Supported	
pinMode()	Supported	
Analog I/O		
analogRead()	Supported	
analogReference()	Not supported	
analogWrite()	Supported	
Zero, Due & MKR Family		
analogReadResolution()	NC	No compatible board
analogWriteResolution()	NC	No compatible board
Advanced I/O		
noTone()	Supported	
pulseIn()	Supported	
pulseInLong()	Supported	
shiftIn()	Supported	
shiftOut()	Supported	
tone()	Supported	
Time		
delay()	Supported	
delayMicroseconds()	Supported	
pulseInLong()	Supported	
micros()	Supported	
millis()	Supported	

Function	Status	Note
Characters		
isAlpha()	Supported	
isAlphaNumeric()	Supported	
isAscii()	Supported	
isControl()	Supported	
isDigit()	Supported	
isGraph()	Supported	
isHexadecimalDigit()	Supported	
isLowerCase()	Supported	
isPrintable()	Supported	
isPunct()	Supported	
isSpace()	Supported	
isUpperCase()	Supported	
isWhitespace()	Supported	
Random Numbers		
random()	Supported	
randomSeed()	Supported	
Bits and Bytes		
bit()	Supported	
bitClear()	Supported	
bitRead()	Supported	
bitSet()	Supported	
bitWrite()	Supported	
highByte()	Supported	

Math		
abs()	Supported	
constrain()	Supported	
map()	Supported	
max()	Supported	
min()	Supported	
pow()	Supported	
sq()	Supported	
sqrt()	Supported	
Trigonometry		
cos()	Supported	
sin()	Supported	
tan()	Supported	

lowByte()	Supported	
External Interrupts		
attachInterrupt()	Supported	
detachInterrupt()	Supported	
Interrupts		
interrupts()	Supported	
noInterrupts()	Supported	
Communication		
serial	Supported	
stream	Supported	
USB		
Keyboard	NC	No compatible board
Mouse	NC	No compatible board

2 Arduino Library Support

Reference: Arduino libraries (https://www.arduino.cc/en/Reference/Libraries)

Function	Status	Note
SPI	Full supported	
PSCOMM	Full supported	AXIOM library for custom communication (see here (https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTATION/BSP/AVR8_system_communication))

3 Arduino IDE

Into the File Sytem 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

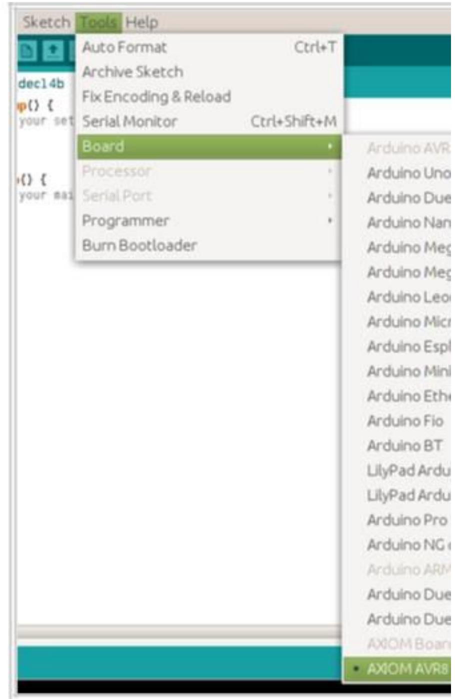
The AVR8 IP is denomined, into the IDE, as "AXIOM AVR8 IP" and is present into the board list of the software. So the user mu st to select the correct item from the board list before write a sketch. In this manner all features of the IP will be available.

The Serial Monitor is available over the default serial port ttyACM0.

Arduino IDE 1.5.2 About

Arduino IDE in AXIOM Ubuntu 16.04

AXIOM AVR IP board support



AXIOM AVR IP board selection

4 Arduino PWM Support and Examples

Tag:

- analogWrite;
- PWM;

Reference: Atmega103 ref manual (https://wiki.axiom-project.eu/images/8/85/Doc0945.pdf)

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

Controller	Pin Num.
PWM0	3
PWM2	6

(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 lever code and the register to use is the TCCRx (Timer/Counter Contol 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:

PWM0			
CS02	CS01	CS00	Clock frequency
0	0	1	32 KHz
0	1	0	4 KHz
0	1	1	1 KHz
1	0	0	500 Hz
1	0	1	250 KHz
1	1	0	125 KHz
1	1	1	31.25 Hz

PWM2			
CS02	CS01	CS00	Clock frequency
0	0	1	32 KHz
0	1	0	4 KHz
0	1	1	500 Hz
1	0	0	125 Hz
1	0	1	31.25 Hz

Example: set PWM0 to 1KHz

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

Example: set PWM0 non-inverted

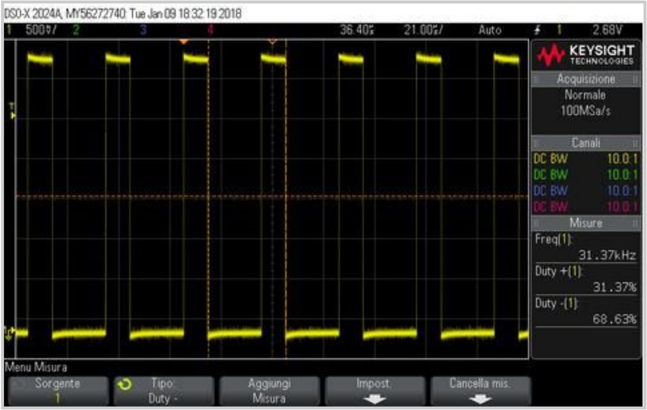
```
TCCR0 &= (0x3 << 4);
TCCR0 |= (0x2 << 4);
```

Example: set PWM0 inverted

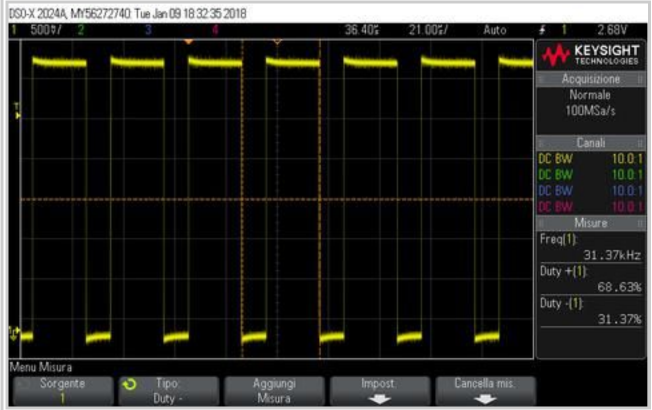
```
TCCR0 |= (0x3 << 4);
```

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

```
int pinPWM = 3;

void setup() {
    TCCR0 &= 0xF8;
    TCCR0 |= 0x1;
}

void loop() {
    analogWrite (pinPWM, 80);
    for ( ;; );
}
```



PWM0 @32KHz,30% duty, non inverted



PWM0 @32KHz,30%, duty, inverted

PWM0 @4KHz,30% duty, non inverted

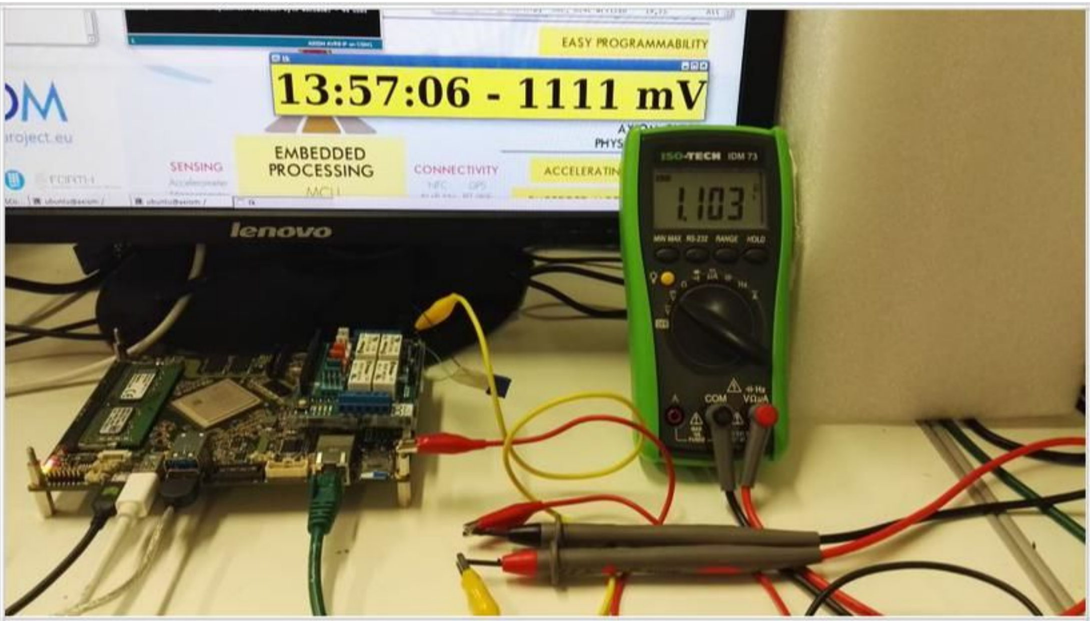
5 Example: Analog Voltage displayed in Linux

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 thi s value to the PS. At PS side, there is a python script witch reads the values sent by the AVR, through the buffer, and displays them in a window as show in the pictures below.

```
#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);
  }
}
```



Arduino IDE in AXIOM Ububtu 16.04

6 Example: PIR with LED indication

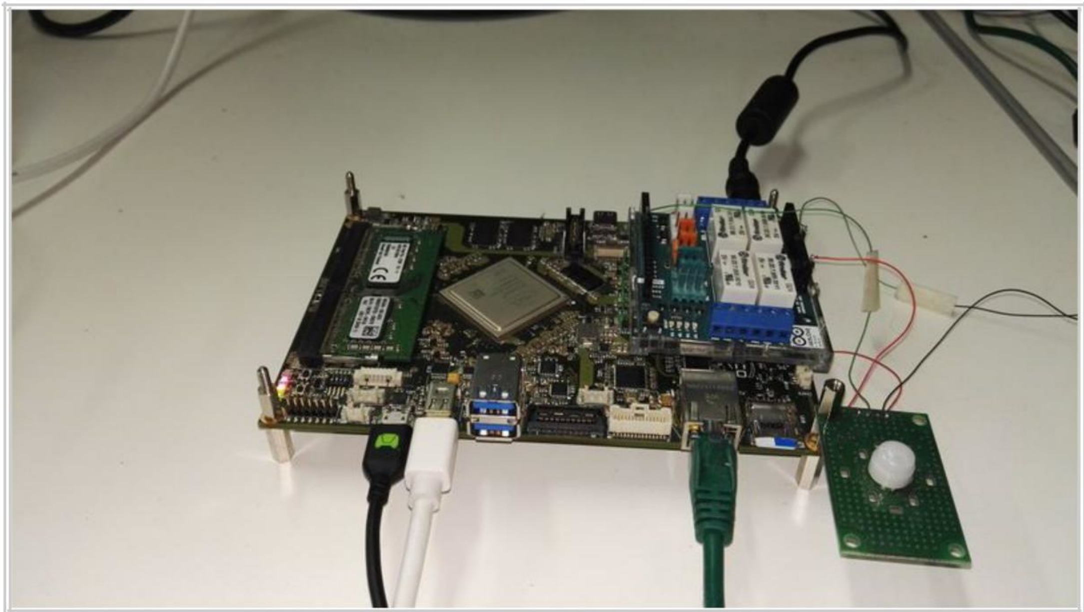
Tag:

- pinMode;
- digitalRead;
- digitalWrite;

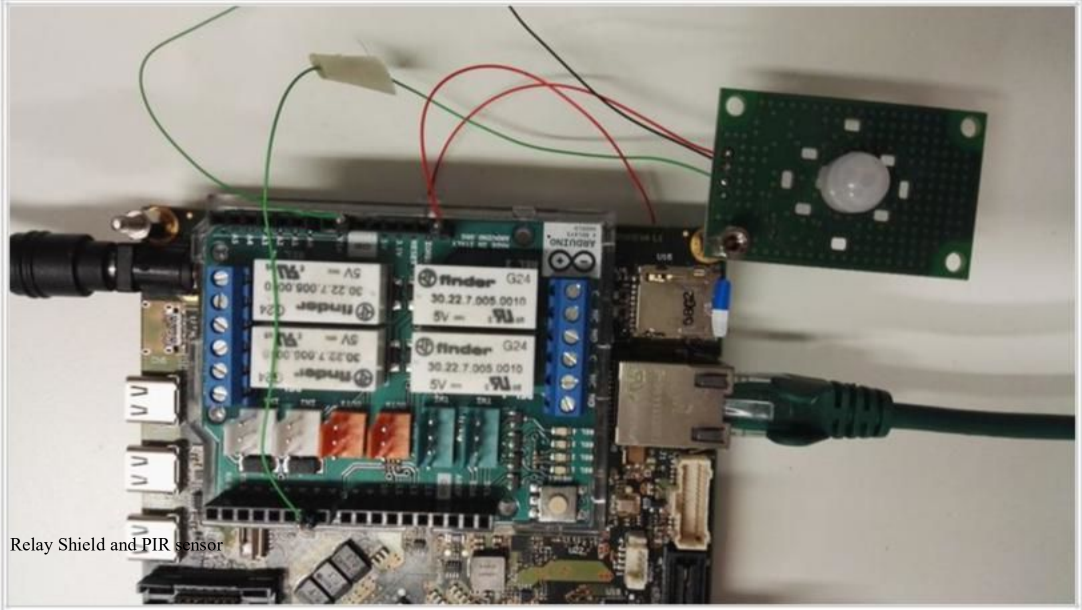
A PIR sensor is connected to the Arduino's shield in he follow mode: - Pin 6 -> sensor's signal (active Low); - Pin 8 Power group -> 3.3V - Pin 2 Power proup -> GND When the PIR sensor recognises a movement the four leds of the shield will be sequentialy put to ON. Otherwise, leds are all OFF.

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

```
#define DELAY_LED_ON 100

int pinPIR = 6;
int pinLED1 = 4;
int pinLED2 = 7;
int pinLED3 = 8;
int pinLED4 = 12;
int statePIR;

void setup() {
  pinMode (pinPIR, INPUT);
  pinMode (pinLED1, OUTPUT);
  pinMode (pinLED2, OUTPUT);
  pinMode (pinLED3, OUTPUT);
  pinMode (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



Relay Shield and PIR sensor

7 Example: SPI NOR Flash write/read

Tag:

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

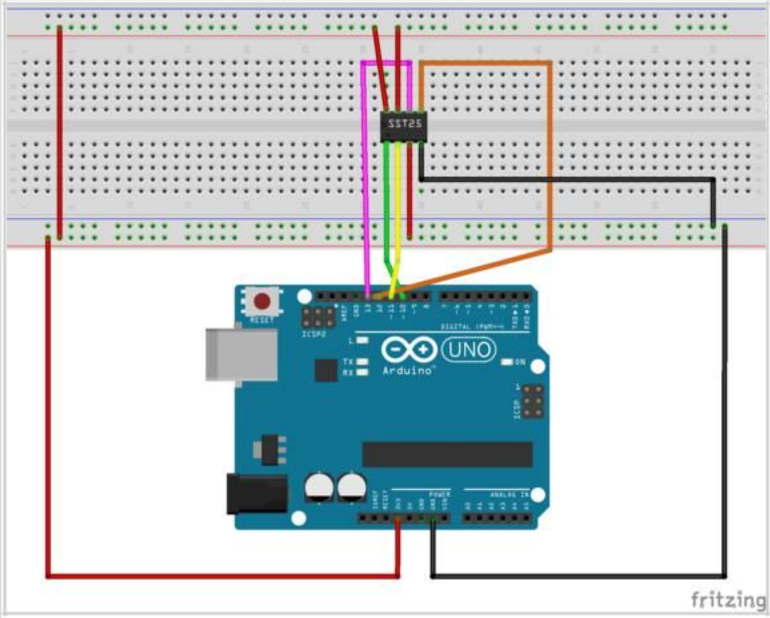
As showed into picture below, a NOR flash is connected with the AVR8 through SPI bus. The sketch read the vendor and device IDs, erase the whole memory and write the first twelve values of the Fibonacci's series. The values are read and showed via Serial Monitor.

External document

- Data Sheet of SST25VF032B (http://ww1.microchip.com/downloads/en/DeviceDoc/20005045C.pdf)

```
#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 get_id (byte *vendor_id, byte *device_id) {
  SPI.enableChip ();
  SPI.transfer(0x90);
  SPI.transfer(0x00);
  SPI.transfer(0x00);
```

```
(continue ...)
  SPI.transfer(addr1);
  SPI.transfer(addr0);
  *data = SPI.transfer(0x00);
  SPI.disableChip ();
}

 void chip_erase () {
  SPI.enableChip ();
  SPI.transfer(0x01);
  SPI.transfer(0x00);
  SPI.disableChip ();

  SPI.enableChip ();
  SPI.transfer(0x06);
  SPI.disableChip ();
  SPI.enableChip ();
```



Schematic Arduino + SPI Flash NOR (build with Frizzing)

Screenshot of Arduino IDE (with the running program) and Serial port monitor with relative output

```
  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 get_status_reg (byte *state)
  { SPI.enableChip ();
  SPI.transfer(0x05);
  *state = SPI.transfer(0x00);
  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(0x03);
  SPI.transfer(addr2);
 (continue ...)
```

```
  SPI.transfer(0xC7);
  SPI.disableChip ();
}

#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.print ("Vendor ID:  ");
 Serial.println (v_id, HEX);
 Serial.print ("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);
}
```

Oscilloscope screenshot of the SPI signals during the reading of the vendor and device IDs

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.

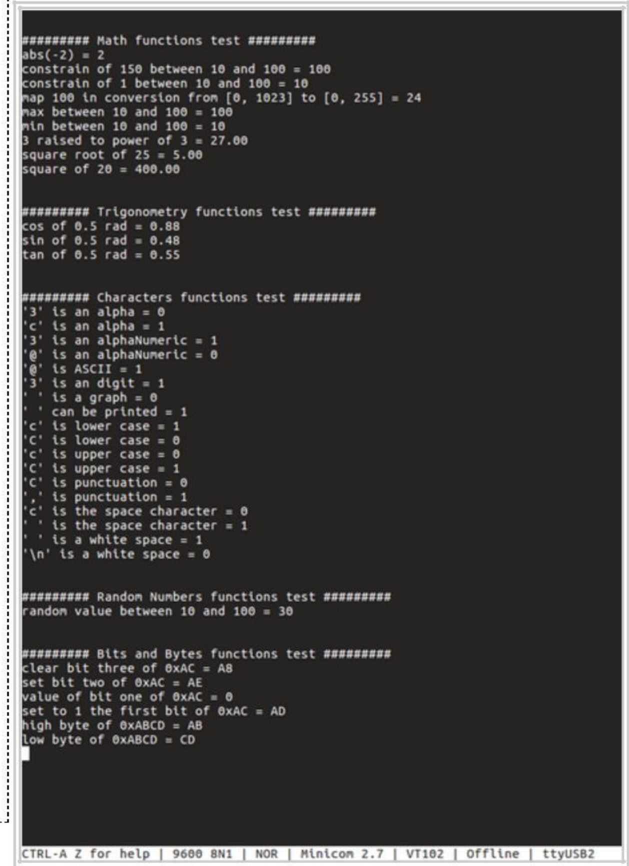
The sketch simple shows the usage of all functions of the mentioned groups of the "Language Reference" of Arduino. All the re sults of those functions are shown via terminal emulation program, through Serial Port (in this regard, a UART-to-USB adapter was used).

```
void setup() {
  Serial.begin(9600);

  Serial.println("");
  Serial.println("");
  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.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(sqrt(25));

Serial.print("square of 20 = ");
Serial.println(square((double)20));


Serial.println("");
Serial.println("");
Serial.println("######### 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("");
Serial.println("");
Serial.println("######### 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("'@' is ASCII = ");
Serial.println(isAscii('@'));

Serial.print("'3' is an digit = ");
Serial.println(isDigit('3'));

Serial.print("' ' is a graph = ");
Serial.println(isGraph(' '));

Serial.print("' ' can be printed = ");
Serial.println(isPrintable(' '));
.....
```

```
Serial.println(isWhitespace('\n'));

Serial.println("");
Serial.println("");
Serial.println("######### Random Numbers functions test #########");

Serial.print("random value between 10 and 100 = ");
randomSeed(50);
Serial.println(random(10, 100));

Serial.println("");
Serial.println("");
Serial.println("######### 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() {

}
```



Serial Port output

▪

FirstBoot

Contents

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_IMPLEM ENTATION/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_-_ARCHITECT URE_IMPLEMENTATION/BSP/ComponentsAndSources#Getting_the_pre-built_BSP)

Creating SD Card

<u>Preparing the SD card</u>

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.

<u>Flashing the SD card</u>

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 witch 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,0xFF000000,115200n8 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:

```
Release 2016.3   Feb 24 2017  -  08:56:12
Platform: Silicon (3.0), Cluster ID 0x80000000
Running on A53-0 (64-bit) Processor, Device Name: XCZU9EG
Processor Initialization Done
================= In Stage 2 ============
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: 0xFFFEA000
Execution Address: 0xFFFEA000
 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 0xfffea000
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:
```

-

PetalinuxTool

Contents

1 Notice

This page provides the desctiption 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

The actual version of Petalinux Tool is the 2016.3 one. You can download PetaLinux installation package directly from Xilinx Downloads page (https:// www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/archive.html).
(N.B. The download require autetication. So please, if needed, regist own account form Xilinx Account Page (https://www.xilinx.com/registration/create -account.html))

Download package: petalinux 2016.03 (https://www.xilinx.com/member/forms/download/xef.html?filename=petalinux-v2016.3-final-installer.run &akdm=1)

(N.B. The used Petalinux installation package is also available into AXIOM server: downlaod link (https://wiki.axiom-project.eu/index.php/WP6_-_AR CHITECTURE_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.

Tool/Library	CentOS 7.1	RHEL 6.6	RHEL 6.7	RHEL 7.2	SuSE 12.0	RHEL 7.2	Ubuntu 14.04
dos2unix	dos2unix 6.0.3	dos2unix 3.1-37	dos2unix 3.1-37	dos2unix 6.0.3	dos2unix 6.0.3	dos2unix 6.0.4	tofrodos 1.7.13
ip	iproute 3.10.0	iproute 2.6.32	iproute 2.6.32	iproute 3.10.0	iproute 3.10.0	iproute2	iproute2
gawk	gawk 4.0.2	gawk 3.1.7	gawk 3.1.7	gawk 4.0.2	gawk 4.0.2	gawk 4.1.0	gawk 4.0.1
gcc	gcc 4.8.3	gcc 4.4.7	gcc 4.4.7	gcc 4.8.3	gcc 4.8.5	gcc 4.8	gcc 4.8
git	git 1.8.3	git 1.7.1	git 1.7.1	git 1.8.3	git 1.8.3	git 1.7.1 or above	git 1.7.1 or above
make	make 3.81	make 3.81	make 3.81	make 3.82	make 3.82	make 4.0	make 3.81
netstat	net-tools 2.0	net-tools 1.60	net-tools 1.60	net-tools 2.0	net-tools 2.0	net-tools	net-tools
ncurses devel	ncurses-devel 5.9-13	ncurses-devel 5.7-3	ncurses-devel 5.7-4	ncurses-devel 5.9-13	ncurses-devel 5.9-13	ncurses-devel	libncurses5-dev
tftp server	tftp-server	tftp-server	tftp-server	tftp-server	tftp-server	atftp or yast2-tftp-server	tftpd
zlib devel	zlib-devel 1.2.7	zlib-devel 1.2.3	zlib-devel 1.2.3	zlib-devel 1.2.7	zlib-devel 1.2	zlib-devel	zlib1g-dev
openssl devel	openssl-devel 1.0	openssl-devel 1.0	openssl-devel 1.0	openssl-devel 1.0	openssl-devel 1.0	libopenssl-devel	libssl-dev
flex	flex 2.5.37	flex 2.5.35	flex 2.5.35	flex 2.5.37	flex 2.5.37	flex	flex
bison	bison-2.7	bison-2.4.1	bison-2.4.1	bison-2.7.4	bison-2.7.4	bison	bison
libselinux	libselinux 2.2.2	libselinux 2.0.94	libselinux 2.0.94	libselinux 2.2.2	libselinux 2.2.2	libselinux 2.3.2	libselinux1

5 Installation Tool

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

E.g. To specify the "/opt/" folder as installation folder:

```
$ ./petalinux-v2016.3-final-installer.run /opt/
```

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 installer 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 describs th e folder (with absolute path) of the single runnable command used by te tool. In the following, the single main co mmands 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
declare -x PATH="/opt/petalinux-v2016.3/tools/linux-i386/petalinux/bin:/opt/petalinux-v2016.3/tools/common/petalinux/bin:/opt/petalinux-v2016.3/tools/linux-i386/  micr
declare -x PETALINUX="/opt/petalinux-v2016.3"
declare -x PETALINUX_VER="2016.3"
declare -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 sessi on, you have to run the same script for each of them.

7 PetaLinux commnads

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 pourposes:

- 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.

Option	Function Description	Value Range	default Value
--get-hw-description PATH	Initialize or update the hardware configuration for the PetaLinux project. Mutually exclusive with -c.	user-specified	none
-c,--component COMPONENT	Configured the specified system component. Mutually exclusive with --get-hw-description.	none kernel rootfs	none

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.

Option	Function Description	Value Range	default Value
-p,--project PROJECT	OPTIONAL. PetaLinux project directory path.	user-specified	current directory

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 (eg. 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.

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

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.

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

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 alid when packaging a iamge with the *petalinux-package --image* workflow.

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

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.

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

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.

Option	Function Description	Value Range	default Value
-p,--project PROJECT	OPTIONAL. PetaLinux project directory path.	user-specified	current directory
--force	OPTIONAL. Overwrite existing files on disk.	none	none
--clean	OPTIONAL. Remove all files from the <plnx-proj-root>/prebuilt directory.	none	none
--fpga BITSTREAM	OPTIONAL. Include the BITSTREAM file in the prebuilt directory.	user-specified	none
-a,--add src:dest	OPTIONAL. Add the file/directory specifed by src to the directory specifed by by dest in the pre-built directory.	user-specified	none

Other tool

PetaLinux contains other command, whitch 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

XGenImage

Contents

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.

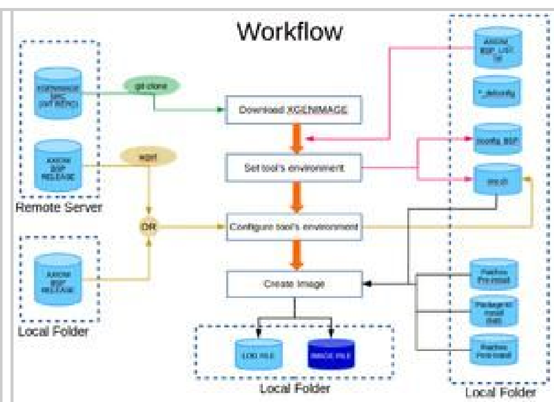
2 Download

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

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

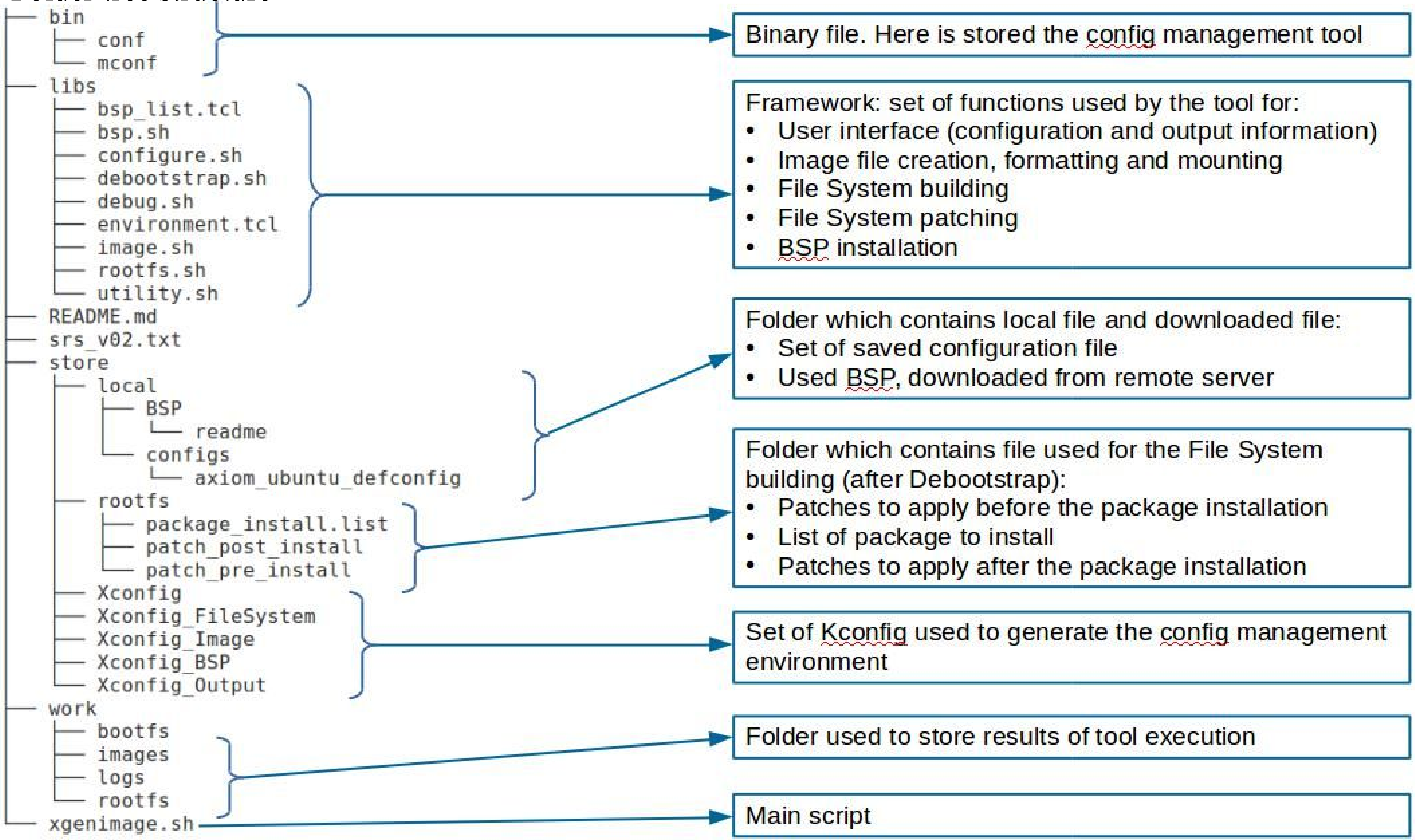
3 Workflow

The main task are:

<table>
<tr><td>Task</td><td>Function Description</td><td rowspan="4">

center Tool Workflow</td></tr>
<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>
</table>

4 Folder tree structure



5 Usage overview

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

1. ./xgenimage --usage

```
* Usage: xgenimage [OPTIONS...]
*
* Examples:
 *  xgenimage -f                        # set current configuration from default one
 *  xgenimage -c                        # open the menu configuration file for editing
 *  xgenimage -e=my_config              # save the current configuration in my_config file
 *
*  General options:
*
 *    -h | --help          : help message
 *    -u | --usage         : usage message
 *    -d | --debug         : enable debug messages
 *    -v | --verbose       : enable verbose messages
 *    -V | --version       : print software version
 *
*  Configuration control:
*
*     -c | --config        : open the config file editor
*     -D | --defconfig     : restore the configuration with default one
*     -e | --exportconfig  : export the current configuration in OUTCONFIG file
*                            The default output file is defconfig. If you want
*                            to change the output file:
*                                   -e=<output file>
*                                   --exportconfig=<output file
*     -C | --cleanall      : clean all user settings
*
*  Image creation:
*
*     -I | --full-image    : create a complete image file with the given specification
*
```

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:

```
# ./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:

```
# ./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:

```
# ./xgenimage.sh --cleanall
```

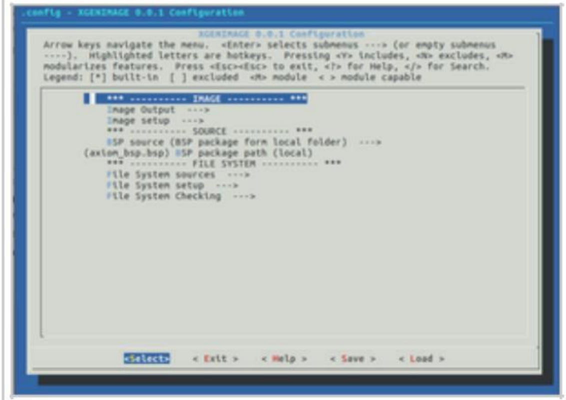
10 Configuration

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

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

Main menu

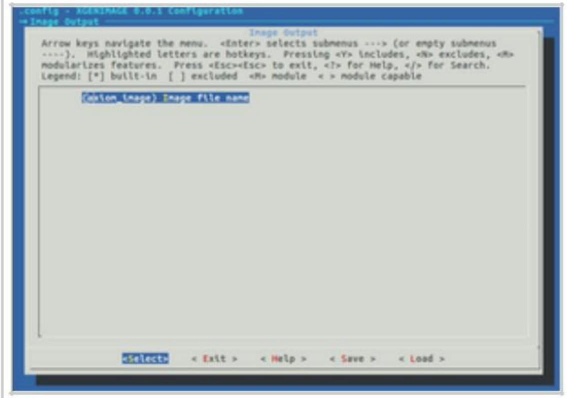
Item	Function Description
Image Output	All the options that involve the output image file
Image Setup	Source of the BSP package (remote o local)
BSP package path	Local source path of the BSP location
File System sources	Source of the base Ubuntu File System
File System Setup	Ubuntu setting to apply after the installation into the root partition
File System checking	Set of item about the File System integrity checking operation



center Main menu

Image Output

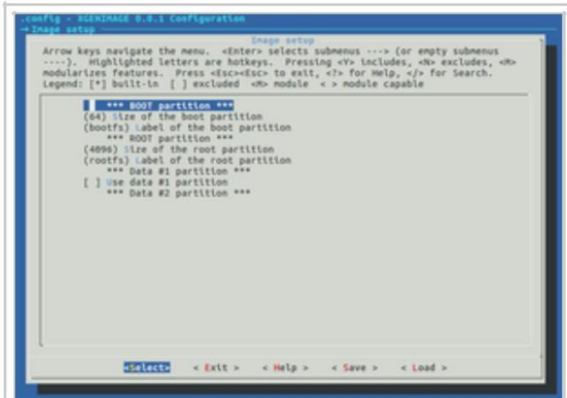
Item	Function Description
Image file name	name of the output file (default: *axiom_image*)



center Image Output

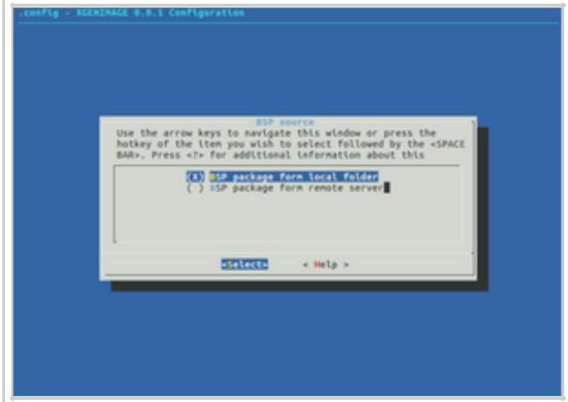
Image setup

Item	Function Description
Size of the boot partition	size, in MB, of the boot partition (default: *64MB*)
Label of the boot partition	label of boot partition (default: *bootfs*)
Size of the root partition	size, in MB, of the root partition (default: *4096MB*)
Label of the root partition	label of root partition (default: *rootfs*)
Use data #1 partition	Create a generic data partition after the rootfs partition (default: *diable*) If select, allows setting of name and size of this partition
Use data #2 partition	Create a generic second data partition after the data1 partition (default: *diable*) If select, allows setting of name and size of this partition



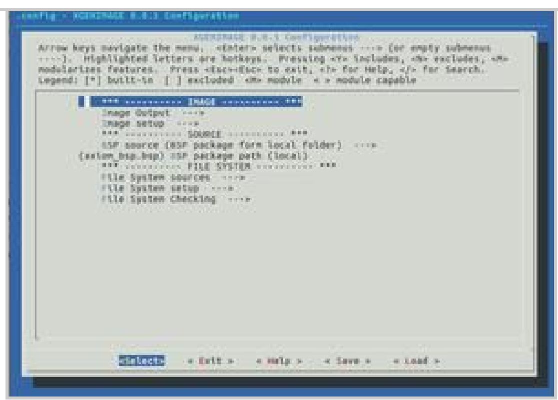
center Image setup

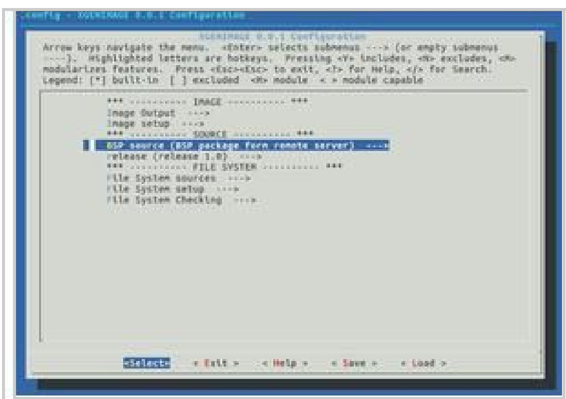
BSP source

<table>
<tr><th>Item</th><th>Function Description</th><td rowspan="3">

center BSP source</td></tr>
<tr><td>BSP package form local folder</td><td>local source for the BSP package (default: enable)</td></tr>
<tr><td>BSP package form remote server</td><td>Remote source for the BSP package (default: disable)</td></tr>
</table>

BSP location

<table>
<tr><th>Item</th><th>Function Description</th><td rowspan="3">

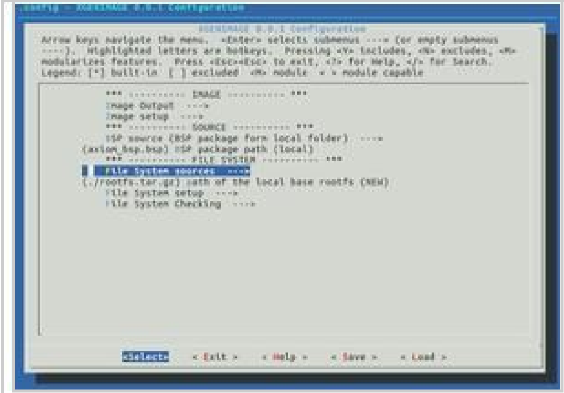
center local BSP source</td><td rowspan="3">

center remote BSP source</td></tr>
<tr><td>BSP package path</td><td>local source for the BSP package (default: ./axiom_bsp.bsp</td></tr>
<tr><td>release</td><td>release to use of the BSP present into the server (default: release 1.0</td></tr>
</table>

File System sources

Item	Function Description
Base File System from local archive	use local archivie as base for the File System building (default: *disable*)
Base File System from remote archive	use remote archivie as base for the File System building (default: *enable*)
Base File System from debootstrap procedure	use full debootstrap (firsr and second stage) for the File System building (default: *disable*)


center File System sources


center File System local sources

File System setup

Item	Function Description
Board's hostname	system hostname (default: *axiom*)
Account user name	name of the dafault user (default: *ubuntu*)
Account user password	passowrd of the default user (default: *ubuntu*)
Root password access	password of the system administrator (default: *root*)


center File System setup

Image creation

This procedure is divided in tasks:

- Downlaod 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 c reated

Command:

```
# ./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:

```
##DESCRIPTION##
<brief descriptin>
##DESCRIPTION##

##FILE##
<file, object of the patch>
##FILE##
```

```
##CMD##
<command>
##CMD##
```

With:

- <brief descriptin>: 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
        ├── apt.conf
        ├── 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
```

PMT - Power Monitoring Tool

Contents

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:

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

To install the newest version:

```
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 create eight file (one for 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 fi le Each file has a standar 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
```

```
2723            1594            1280
2998            1672            1380
3292            1686            1380
3559            1654            1340
3829            1608            1320
4116            1684            1380
4400            1649            1340
4700            1701            1400
4993            1642            1340
5258            1661            1380
5554            1654            1340
5816            1661            1380
6067            1573            1320
6325            1557            1280
6568            1436            1240
6812            1415            1280
7059            1638            1340
7304            1431            1220
7551            1449            1220
7798            1517            1380
8064            1645            1340
8341            1689            1380
8634            1638            1340
8902            1665            1380
9196            1666            1380
9515            1679            1380
9795            1680            1380
```

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

	SPI	eMMC	uSD	USB	TFTP	NFS
image.ub	ToDo	ToDo	Yes	Yes	Yes	
FileSystem		Yes	Yes	Yes		ToDo

KERNEL/PowerMonitor

Contents

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

Chip Part Number: INA219 (http://www.ti.com/lit/ds/symlink/ina219.pdf)
The following rail are monitored:

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

(*) 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).

Bus/SYSFS combination	
Bus	SYSFS Folder
PM_VCC_INTFP	/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0040/hwmon/hwmon0
PM_VCC_INT	/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0041/hwmon/hwmon1
PM_VIN	/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0042/hwmon/hwmon2
PM_INTFP_DDR	/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0043/hwmon/hwmon3
PM_1V2_DDR_PS	/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0044/hwmon/hwmon4
PM_1V2_DDR_PL	/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0045/hwmon/hwmon5
PM_MGTAVCC	/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0046/hwmon/hwmon6
PM_MGTAVTT	/sys/devices/platform/amba/ff020000.i2c/i2c-0/0-0047/hwmon/hwmon7

In each SYSFS folder there are the following file descriptor:

SYSFS User Interface		
File Name	Access	Description
bus_name	r	Name of the bus of the board under measurement
shunt_resistor	r	Value in mOhm of the shunt resistor
in0_input	r	Shunt voltage measurement data (in mV)
in1_input	r	Bus voltage measurement data (in mV)
curr1_input	r	Current (in mA) flowing into the shunt resistor
power1_input	r	Power (in mW) supplied to the load over the bus
compensation	r/w	Compensation value (in ‰) to delete the load of the filter
dump_regs	r	Raw value list of all chip registers

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 root  4096 Feb 11 16:54 bus_name
-rw-r--r-- 1 root root  4096 Feb 11 16:54 compensation
-r--r--r-- 1 root root  4096 Feb 11 16:54 curr1_input
lrwxrwxrwx 1 root root    0 Feb 11 16:54 device -> ../../../0-0042/
-r--r--r-- 1 root root  4096 Feb 11 16:54 dump_regs
-r--r--r-- 1 root root  4096 Feb 11 16:54 in0_input
-r--r--r-- 1 root root  4096 Feb 11 16:54 in1_input
-r--r--r-- 1 root root  4096 Feb 11 16:54 name
lrwxrwxrwx 1 root root    0 Feb 11 16:54 of_node -> ../../../../../../../../firmware/devicetree/base/amba/i2c@ff020000/ina219@42/
-r--r--r-- 1 root root  4096 Feb 11 16:54 power1_input
-rw-r--r-- 1 root root  4096 Feb 11 16:54 shunt_resistor
lrwxrwxrwx 1 root root    0 Feb 11 16:28 subsystem -> ../../../../../../../../class/hwmon/
-rw-r--r-- 1 root root  4096 Feb 11 16:28 uevent
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 istance of the driver relative to the INA219 chip, a device file descriptor is createrd 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:

IOCTL Programming Interface			
Flag	Access type	Description	Parameter
INA2XX_BUS_NAME_GET	r	Returns the name of the bus of the board under measurement	*char
INA2XX_BUS_VOLTAGE_GET	r	Returns the bus voltage measurement data (in mV)	*int
INA2XX_SHUNT_VOLTAGE_GET	r	REturns the shunt voltage measurement data (in mV)	*int
INA2XX_CURRENT_GET	r	Returns the current (in mA) flowing into the shunt resistor	*int
INA2XX_POWER_GET	r	Returns the power (in mW) supplied to the load over the bus	*int
INA2XX_CALIBRATION_GET	r	Returns the calibration value used by the chip to evaluate the curret/power	*int

Example of IOCTL access to read the bus name of the driver with ID=2:

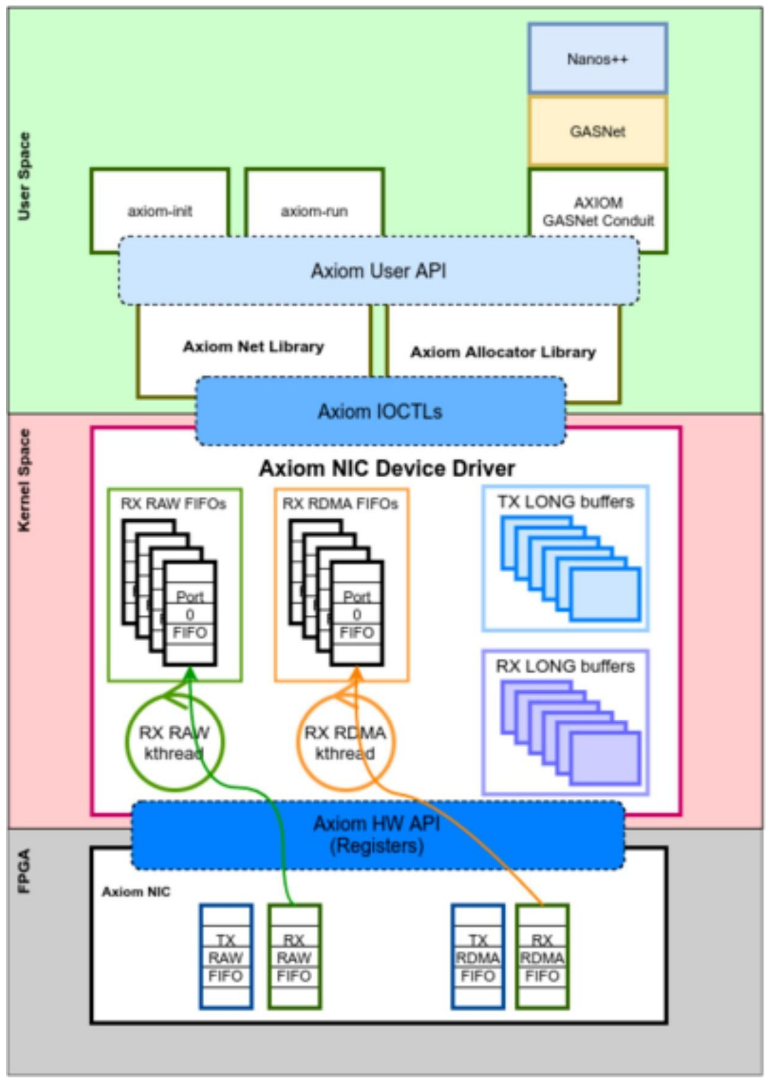
```
#define PM_BUS_NAME_LEN              128

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_compiles_deb_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

```
# 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

```
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

```
************ Master computed Topology ******* ************
Node          IF0    IF1    IF2    IF3    IF4
0             255    255    255    255    255
1             255    255      2    255    255
2             255      1    255    255    255
Node 1 ROUTING TABLE
Node          IF0    IF1    IF2    IF3    IF4
0               0      0      0      0      0
1               1      0      0      0      0
2               0      0      1      0      0
 IPoverAXIOM set to subnet 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)]