PandABlocks   
FPGA Development Framework

|  |  |
| --- | --- |
| **Author:** | Isa Uzun | Senior FPGA Development Engineer Diamond Light Source |

|  |  |  |
| --- | --- | --- |
| **Revision History** | | |
| **Author** | **Date** | **Description** |
| Isa Uzun | 27/03/2017 | Initial release |
|  |  |  |

Contents

[1. Introduction 4](#_Toc478477026)

[2. System Architecture 4](#_Toc478477027)

[3. PandaBox Hardware Architecture 5](#_Toc478477028)

[4. PandaBlock Firmware Framework 5](#_Toc478477029)

[4.1 Functional Block VHDL Module Structure 7](#_Toc478477030)

[5. Common CONFIG fileset 8](#_Toc478477031)

[5.1 config 9](#_Toc478477032)

[5.2 registers 10](#_Toc478477033)

[5.3 description 11](#_Toc478477034)

[5.4 meta 11](#_Toc478477035)

[6. Project Directory Structure 12](#_Toc478477036)

[7. Building PandABlocks Firmware 13](#_Toc478477038)

[Step-1: Generate Application CONFIG Fileset 13](#_Toc478477039)

[Step-2: Auto-Generate Memory Interface VHDL Modules 14](#_Toc478477040)

[Step-3: Generate Xilinx IPs 14](#_Toc478477041)

[Step-4: Generate Zynq Arm Core 15](#_Toc478477042)

[Step-5: Build Top-Level Design 15](#_Toc478477043)

[Step-6: Generate panda-fpga\_<APP>.zpg 15](#_Toc478477044)

[8. FMC or SFP Blocks 16](#_Toc478477045)

[9. Adding Custom Functional Blocks 17](#_Toc478477046)

[10. REFERENCES 17](#_Toc478477047)

[APPENDIX-A: FB\_TOP VHDL TEMPLATE 19](#_Toc478477048)

[APPENDIX-B: FMC\_TOP VHDL TEMPLATE 20](#_Toc478477049)

[APPENDIX-C: SFP\_TOP VHDL TEMPLATE 21](#_Toc478477050)

# Introduction

This document describes the PandABlocks FPGA development framework targeting the Zynq-based PandABox hardware platform. It provides steps to follow for 3rd-parties to develop and integrate custom functionalities into the framework and run on target hardware.

# System Architecture

FPGA firmware and TCP-Server comprise the core of the PandABlocks system architecture implemented on the Xilinx Zynq FPGA device as shown in Figure 1. These two layers are tightly coupled through a common set of configuration files as described in Section 5. Successful operation of PandABlocks on target hardware requires proper management of these fileset.

**Figure 1 Firmware and TCP-Server on Zynq 7030 FPGA.**

# 

# PandaBox Hardware Architecture

Figure 2 shows a high-level block diagram of the PandaBox hardware [1]. There are two FPGA devices on its carrier board:



**Figure 2 PandA hardware peripherals**

***Picozed Zynq 7030***: implements all the main capabilities at the firmware level. It interfaces to main I/O ports, implements logical blocks and data capture functionality.

***Spantan-6***: serves for housekeeping purposes and interfaces to slow I/O ports such as encoder control, status LEDs and temperature monitoring.

These 2 FPGAs are connected via 4-wire SPI-like interface using a custom protocol where data flows in both directions.

***NOTE:*** SlowFPGA firmware is out-of-scope of this document.

# PandaBlock Firmware Framework

PandABlocks is the firmware framework that can target any Zynq-based hardware platform. The framework is composed of Functional Blocks (FB). A Functional Block is a VHDL module performing a specific task such as Look-Up Table, Pulse Generation, Divider and Counter. A FB is controlled and interrogated through a number of registers.

User-defined number of instances of functional blocks are compiled and built together for an application-specific FPGA firmware to run Zynq device.

A functional block has [2]:

* Discrete bit-type input and output ports,
* 32-bits wide position-type input and output ports,
* Configuration and status registers.

A simple block diagram of a generic FB is shown in Figure 3.



**Figure 3 FB block diagram.**

Wiring of FBs is achieved via a common system bus and position bus routed internally across the design. System bus feeds functional block’s bit-type inputs while position bus feeds the position-type inputs. Configuration and monitoring of FBs are achieved via a registers mapped to processor memory address space.



**Figure 4 PandABlocks framework architecture**

Figure-4 shows the overall architecture of PandABlocks framework with following components.

* Zynq Core Block Design: is hardware specific implementation of Zynq ARM core and AXI interconnect logic.
* Base Functional Block: is hardware specific block in order to provide physical IO interfacing available on the carrier board such as TTL, LVDS and Encoders.
* Position Capture Functional Block: implements dedicated DMA-based data capture logic for all bit and position type data fields in the system. This block is tightly coupled with Linux kernel driver.
* Generic Functional Blocks: are VHDL modules implementing user-defined configurable functionalities. There can be multiple instances of these FBs.

## Functional Block VHDL Module Structure

PandABlocks employs a standard module structure for the design of its functional blocks following bottom-to-top hierarchy as shown in Figure-5. 3rd party developers wishing to add own FBs into the framework should obey this structure for easy integration. A FB is composed of 3 VHDL modules as below.

* <FB>.vhd: This file implements the “function core” such as pulse generation, counter etc. The entity has direct port declarations for physical input and outputs, configuration parameters and status information. It doesn’t implement a memory bus interface.
* <FB>\_block.vhd: The block file wraps the function core (*<FB>.vhd*) and auto-generated memory interface module (*<FB>\_ctrl.vhd*) for parameter and status register access. It also implements bus multiplexers for input ports.
* <FB>\_top.vhd: The top level file instantiates required number of functional blocks and implements a thin layer of address decoding logic to target each block individually on the address space.



**Figure 5 PandABlocks Functional Blocks implementation structure.**

The development framework provides VHDL templates for the *<FB>\_top.vhd* and *<FB>\_block.vhd* modules as shown in the Appendix-A so that developers can focus their efforts to implementing the core FB.

Finally, top-level VHDL design module (*top.vhd*) is composed by instantiating *<FB>\_top.vhd* components for target application in-mind.

# Common CONFIG fileset

CONFIG filesets are used to capture Functional Block’s structure. As an example; SRGATE block has 2 bit-type inputs, 1 bit-type output and 4 configuration registers as depicted in Figure-6.



**Figure 6 SRGATE block diagram**

The CONFIG fileset for a functional block consists of 4 individual files as demonstrated in the following subsections for the SRGATE.

## config

This file describes a functional block’s fields including:

* Inputs and Outputs.
* Configuration and status registers.
* Number of instances in the top-level design.

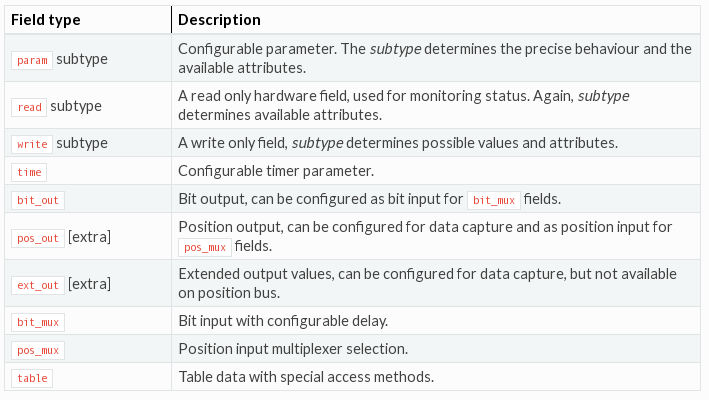
Example below shows *config* file for SRGATE functional block.



**Figure 7 config file for SRGATE block**

The second column in the file above defines the field type for each field. The types of fields PandABlocks framework supports are listed in Table 1. For detailed information regarding field types and their attributes please see reference [2].

Table Supported Field types for Functional Blocks



## registers

This file defines following information regarding a functional block;

* Register addresses for:
  + Configuration registers.
  + Status registers.
  + Control registers for *bit\_mux* type inputs.
* System bus bit index for each output port.

Figure-8 shows an example registers file with MACRO definitions for SRGATE.



**Figure 8 registers file for SRGATE block**

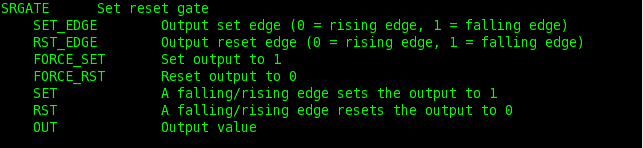
This file is then processed within the project build flow and populated with the actual register mapping assignments. The processed registers file is shown in Figure-9.



**Figure 9 registers file after processing for SRGATE**

## description

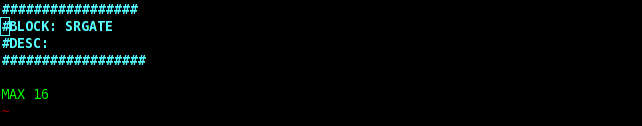
This file contains short description of each field in the config file.



**Figure 10 description file for SRGATE block**

## meta

This file defines the maximum number of instances for the block in the firmware



**Figure 11 meta file for SRGATE which states that maximum 16 SRGATE block can be instantiated in the top-level design**

These set of configuration files for each block are used to auto-generate:

* A global VHDL package files to define register names and addresses for all FBs.
* VHDL Memory Interface modules for individual FBs.
* A VHDL module to combine all bit-type and pos-type FB outputs into two global busses namely system bus and position bus.

The same config fileset is also used as an argument to tcp-server. This way tcp-server is informed about the functional blocks and their attributes available on the FPGA firmware.

# Project Directory Structure

PandABlocks repository contains following sub-directories.

|  |  |
| --- | --- |
| **Sub-directory Name** | **Description** |
| common/  |---python/  |---vhdl/ | Common source and build files   * Python scripts used in the build process * VHDL source files shared between FBs |
| modules/  |--- <FunctionalBlock>  | |---config/  | |---sim/  | |---vhdl/  |  |--- <FunctionalBlock>  | |---config/  | |---sim/  | |---vhdl/  ... | It contains all sources per Functional Block   * Functional Block sub-directory   + Config fileset per FB (config, registers and description)   + Python and FPGA simulation files   + VHDL source files |
| apps/ | Firmware application configuration files |
| targets/  |---PandABox/  |---bd/  |---config/  |---constr/  |---scripts/ | Target hardware platform specific files   * Target hardware   + Zynq ARM core block design source   + Vivado BSP   + FPGA constraints   + Build scripts |
| tools/ | Utilities |
| doc/ | HTML documentation |
| SlowFPGA/\* | FPGA project files for the Xilinx Spartan-6 Slow Control |
| build/  |---PandABox/  | |---config\_d/  | |---ip\_repo/  | |---panda\_ps/  | |---panda\_top/  |---SlowFPGA/ | Build directory   * Zynq-based target hardware   + Assembled application specific config fileset   + Xilinx IP netlists   + Zynq ARM core netlist   + Top-level Vivado project output files * SlowFPGA output files |

\* This directory is temporarily kept in this repository and will be moved to PandABox repository on OHW.



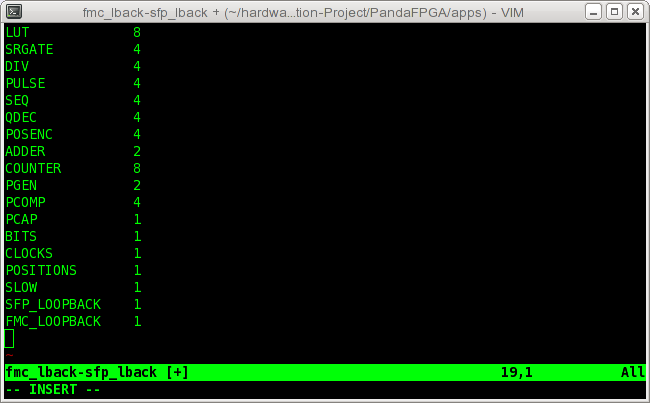
# Building PandABlocks Firmware

FPGA firmware built process is Makefile driven. It uses multiple python and tcl scripts as explained in following sections. PandaBlocks build flow requires Vivado 2015.1.

Output projects and built files are stored in *<BUILD\_DIR>/<TARGET>* directory. *<BUILD\_DIR>* and *<TARGET>* variables are defined in the CONFIG file. By default TARGET is set to PandABox.

## Step-1: Generate Application CONFIG Fileset

Application files, stored in <*apps*> directory, control which functional blocks are included in the firmware built, targeting a particular application. The base PandABox application file is shown in Figure-12 as an example.



**Figure 12 Application file example. The first column declares the required functional blocks in the firmware and the second column defines the number of instantiations per FB.**

The first step in the build process pulls all module-based CONFIG fileset from <*modules/<FB>/config*> directories and creates a global application-specific config fileset which includes config, registers and description files. The generated config files are stored in *<BUILD\_DIR>/<TARGET>/config\_d* directory.

*<TOP\_DIR>/common/python/config\_generator.py* script is called in the Makefile to carry out this step.

## Step-2: Auto-Generate Memory Interface VHDL Modules

*<TOP\_DIR>/common/python/vhdl\_generator.py* script is used in the Makefile to auto-generate following VHDL files from the information available in the config fileset.

1. *address\_defines.vhd* : VHDL package file which defines register names and address for all FBs in the design,
2. *<FB\_Name>\_ctrl.vhd* : VHDL modules implementing memory interface logic for each FB.
3. *panda\_busses.vhd* :VHDL module to combine all bit-type FB outputs into “system bus” and pos-type outputs into “position bus”.

All resulting files are stored in *<BUILD\_DIR>/<TARGET>/autogen* directory.

## Step-3: Generate Xilinx IPs

All Xilinx IPs used in the design are generated in out-of-context mode using *<scripts/build\_ips.tcl>* script. The script includes configuration of all IPs in the design. An example for configuring the “pulse\_queue” FIFO IP is shown in Table-2. Any new user IP configuration must be added in this file as required.

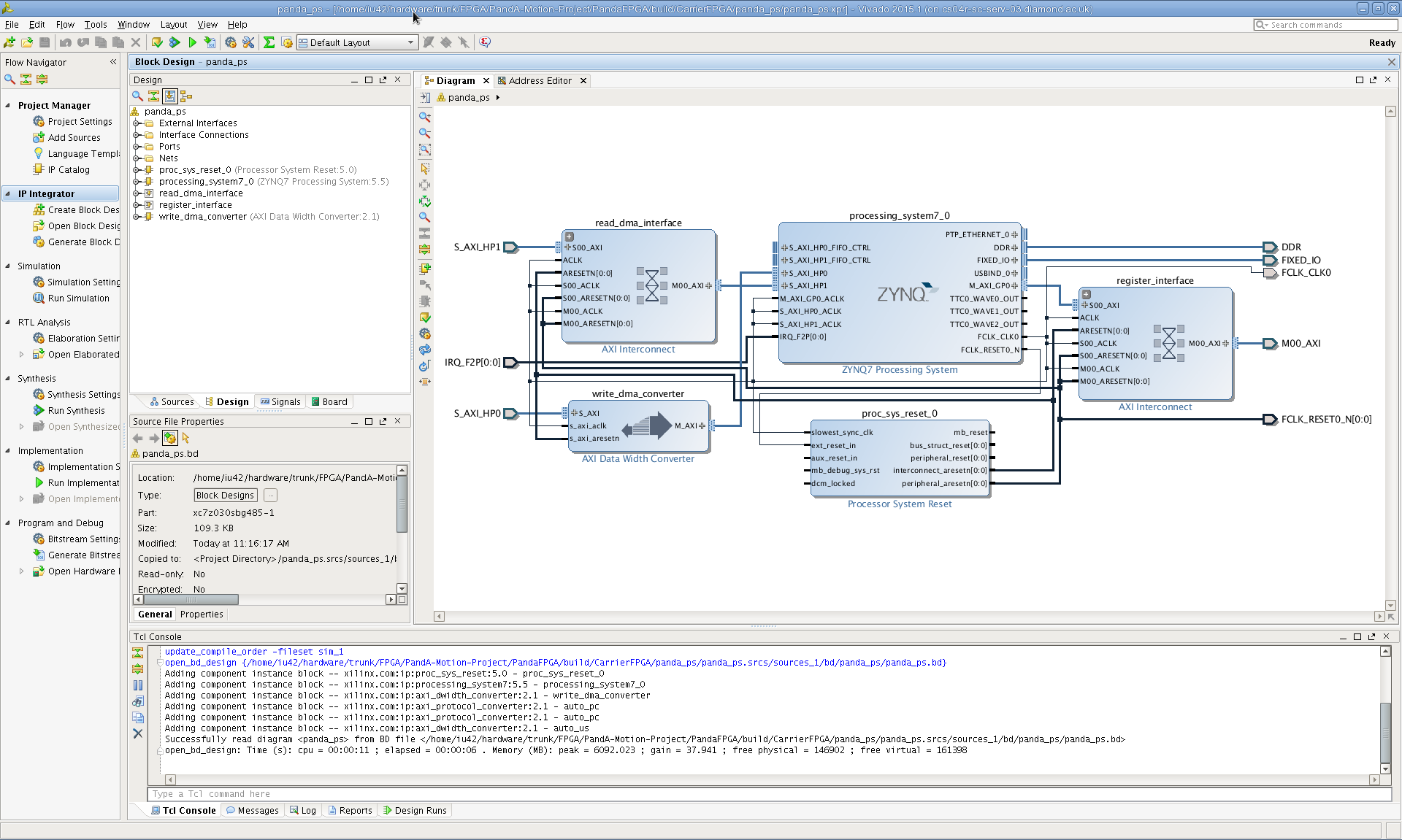
This step places all generated IP netlists in the *<BUILD\_DIR>/PandABox /ip\_repo* directory.

**Table 2 Snippet from build\_ips.tcl script showing configuration of fifo\_generator IP.**

|  |
| --- |
| #  # Create PULSE\_QUEUE IP  #  create\_ip -name fifo\_generator -vendor xilinx.com -library ip -version 12.0 \  -module\_name pulse\_queue -dir $origin\_dir/  set\_property -dict [list \  CONFIG.Performance\_Options {First\_Word\_Fall\_Through} \  CONFIG.Input\_Data\_Width {49} \  CONFIG.Data\_Count {true} \  CONFIG.Output\_Data\_Width {49} \  CONFIG.Reset\_Type {Synchronous\_Reset} \  ] [get\_ips pulse\_queue]  generate\_target all [get\_files $origin\_dir/pulse\_queue/pulse\_queue.xci]  synth\_ip [get\_ips pulse\_queue] |

## Step-4: Generate Zynq Arm Core

At the heart of the design lies the Zyqn ARM core which is captured as a Vivado Block Design. Block design source file (*panda\_ps.tcl*) is located in *PandABox/src/bd/* directory.



**Figure 13 PandABox Zyqn ARM core block design**

<*scripts/build\_ps.tcl*>script creates *<BUILD\_DIR>/PandABox /panda\_ps* project and builds Zynq ARM core netlist.

***NOTE:*** This project can be opened using Vivado in order to modify the Zynq block design. Modified block design must be exported back into the *PandABox/src/bd/* directory.

## Step-5: Build Top-Level Design

In the final step, <*scripts/build\_top.tcl*> script read all IP netlists, Zynq ARM core netlist and all VHDL design sources in order to create the firmware image. At the end of this step, output firmware image *panda\_top.bit*is stored in *<BUILD\_DIR>/PandABox*.

## Step-6: Generate panda-fpga\_<APP>.zpg

Firmware installation onto the target PandABox platform is managed through zpkg files. In this last step, firmware image and configuration files are packaged into *panda-fpga\_<APP>.zpg* file. The directory structure of the package file is defined in *<TOP\_DIR>/etc/ panda-fpga.list* file.

The \*.zpkg file must be transferred onto the target platform and installed as defined in the PandABox Setup and Installation Manual [5].

# FMC or SFP Blocks

FMC and SFP-based functional blocks are special blocks because:

* They have direct connectivity to physical I/O pins, and
* Their instantiation is limited to 1 FMC FB and 3 SFP FBs on PandABox hardware platform.

PandABlocks already supports three different FMC applications through functional blocks below:

* fmc\_loopback: This FB implements loopback on FMC connector for unit testing.
* fmc\_24vio: A FB to provide configurable 24V I/O capability through FMC.
* fmc\_acq420: 3-rd party FB to interface D-Tacq ACQ420 FMC product.

Each FMC (and SFP) application is managed under its own subdirectory in modules/. For FMC and SFP functional blocks, there is in additional folder (*const*) to manage FMC/SFP application specific constraints.

Table FMC and SFP Functional Blocks Directory Structure

|  |  |
| --- | --- |
| modules/  |--- fmc\_*<app#1>*  | |---config/  | |---sim/  | |---vhdl/  | |---const/  |  |--- fmc\_*<app#2>*  | |---config/  | |---sim/  | |---vhdl/  | |---const/ | It contains all sources per Functional Block   * Functional Block sub-directory   + Config fileset per FB (config, registers and description)   + Python and FPGA simulation files   + VHDL source files   + FMC/SFP specific constraint files |

To facilitate the integration of FMC and SFP Functional Blocks into the top-level design, VHDL templates for FMC and SFP components (*sfp\_top.vhd* and *fmc\_top.vhd*) are provided by the framework as shown in Appendix-B and C.

# Adding Custom Functional Blocks

PandABlock FPGA Development Kit provides following VHDL templates to help developing and integrating custom FBs. All VHDL template files are located in *<TOP\_DIR>/common/vhdl/templates* directory.

* *FB\_top.vhd* : Top-level Functional Block entity template,
* *FB\_block.vhd* : FB block entity template
* *fmc\_top.vhd* : FMC-based FB entity template
* *sfp\_top,vhd* : SFP-based FB entity template

The flow diagram in Table-4 shows the steps to follow in order to add a new functional block into the development framework.

# REFERENCES

[1] PandABox Hardware User Manual

[2] Panda-Server Documentation

[3] PandABox Firmware Design Documentation

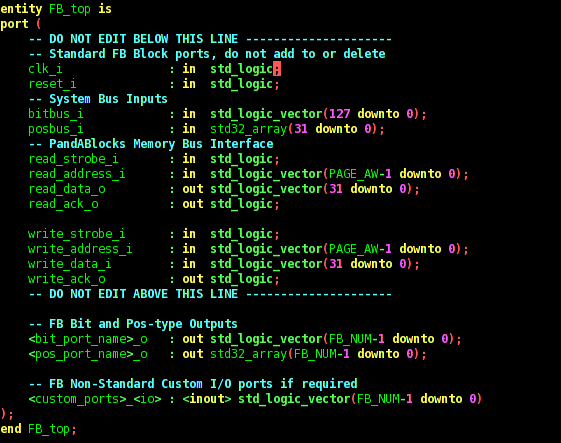
[4] PandaBox Register Address Map Documentation

[5] PandABox Setup and Installation Manual

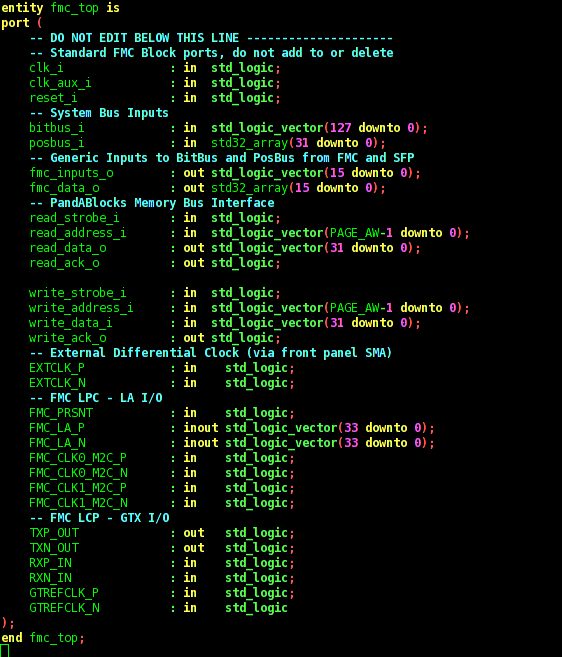
Table Flow for adding new Functional Block into the framework



# APPENDIX-A: FB\_TOP VHDL TEMPLATE



# APPENDIX-B: FMC\_TOP VHDL TEMPLATE



# APPENDIX-C: SFP\_TOP VHDL TEMPLATE

