International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 05 | May 2020 www.irjet.net p-ISSN: 2395-0072

# FPGA-based Sandbox Platform for Hardware Accelerators

**Vidhath BL[1]**
*Department of Electronics and Communications Engineering*
*RV College of Engineering*
*Bangalore, India*

**Harini V[2]**
*Department of Computer Science and Engineering*
*RV College of Engineering*
*Bangalore, India*

**Dr. Govinda Raju M[3]**
*Department of Electronics and Communications Engineering*
*RV College of Engineering*
*Bangalore, India*

**Sandhya S[4]**
*Department of Computer Science and Engineering*
*RV College of Engineering*
*Bangalore, India*

---***---

***Abstract:*** Modern software development often relies on source version control methodologies to realise a robust model for the development, testing and release of new features in production, where there is a clear isolation of testing environments from productive ones while also expediting integration of features from one environment into another. Using an early-stage meta-modelling approach, such a system may be explored in hardware development as well, enabling iterative hardware development in sandbox conditions, module reusability and easy collaborative development, thereby accelerating the hardware design process. This paper proposes to develop a Field Programmable Gate Array (FPGA) based platform that forms a flexible testing ground to assess and iterate over various design configurations as in sandbox conditions. An automated system of datasheet-like documentation and the user-rule-based generation of meaningful derivatives from the designs constructed on the platform is also proposed so as to enhance reusability and accelerate development.

**Keywords:** ***Intellectual Property (IP), Serial Peripheral Interface (SPI), Field Programmable Gate Array (FPGA), Metamodelling***

## 1. INTRODUCTION

Hardware acceleration is a term used to describe tasks being offloaded to devices which specialize in a particular set of operations [1]. While traditional designs are fine in most general use cases, there are others where it might be optimal to utilize the specific modules i.e. accelerators. This specificity results in their designs being highly complex, and any further development or redesign requires in-depth analysis of design, right from the simplest of logical entities. Consequently, the conventional design processes for hardware accelerator Intellectual Property (IPs) is elaborate in terms of both time and resources. Therefore, in the case of hardware accelerators, every iteration of redesign is very expensive. Hence it is evident that extensive testing of such IPs is to be performed before integrating them into the main design of a component. Thus effective testing methods and quicker redesign flows are needed to shorten the complex process of hardware accelerator design. [2]

Drawing from the concept of a sandbox which is a testing environment that isolates untested module changes and outright experimentation from the production environment, the paper proposes a platform wherein modules in production and modules under testing are isolated in hardware while being part of the same system in terms of design flow.

## 2. RELATED WORK

In today's embedded system growth, architecture efficiency remains a major issue. A positive step to boost the design process is domain-specific languages (DSLs). The incoherent syntax in various DSLs, however, negatively affects any gained productivity during system development and manual DSL development. In this report they propose to produce Python-embedded DSLs in a metamodel-based framework. By identifying basic blocks a target metamodel abstracts the models. Their framework generates an expressive DSL that automates model building and allows data flow programming with additional configuration. A "one-language ecosystem" was explored by the authors in [3], with the created DSLs describing RTL and firmware as well as formal properties by using the proposed framework on various meta-models. A chip-based device (SoC) consisting of an RTL code and a firmware stack is generated as proof of concept and formal properties are automatically checked. In order to construct the RTL DSL, this generative method results in a factor-of-six time reduction [4].

Metamodelling-based hardware generation methods are now exponentially relevant for heterogeneous hardware (HW) and software (SW) systems. The key design issues include the need for executable specifications, performance analysis and early design integration possibilities. Recent progress in hardware-software co-specification methodologies has been inspired by inherent characteristics of hardware definition languages; for example, VHDL implements abstract datasets and control mechanisms that may be paralleled in various programming languages using a robust object model. Using such a basis, various hardware generation frameworks have been developed over the past years in interests of maximizing the utility of the valuable pre-RTL architectural design exploration period by enabling early and swift RTL code generation. Three of these methods today are industrial, scholarly and potentially relevant: Bluespec, Genesis2 and Chisel. Chisel is a repository for the Scala language in general programming [5]. As Chisel is Scala-based, it does not use any current HDL framework or syntax, but instead the Scala compiler and runtime environment as well as its syntax and operators. Bluespec's BSV (Bluespec SystemVerilog) is another hardware generation framework, initially established as a Haskell library. It presents an object model corresponding to hardware modules, enforcing atomic actions to impart behaviour to the module in a manner that simplifies verification later on in the hardware development process [6]. Finally, Genesis2 is a framework that enables the generation of reconfigurable, architectural design templates utilizing Perl as the base compiler and runtime environment [7].

Therefore, there is a significant amount of foundational literature and industrial precedent laying out frameworks and methodologies for hardware generation and the associated design flows. However, there is a present lack of a dedicated methodology that utilizes a hardware generation platform. Leveraging current trends in meta-modelling in the hardware domain, we propose a self-documenting sandbox platform to enable the quick testing and development of hardware accelerators to expedite architectural exploration, and introduce component modularity and reusability as with the development of software. It is to be noted that this paper focuses on the system methodology of the previously described sandbox platform rather than the underlying hardware generation framework that it will be implemented over.

## 3. SYSTEM DEVELOPMENT

For generating the RTL for SPI peripheral and register interface, the framework is based on two MoTs. Metamodeling enables easily adaptable SPI circuitry. Various configurations of SPI are needed for different applications. Large or minor changes in the HW components can occur depending upon the MoT. In the second MoT, the register interface hardware is created based on the frame format specification. The hardware designs discussed here are symbolic configurations only. This chapter discusses the SPI peripheral definition, the control and IP logic, and the configuration of these modules.

### *A. Methodology*

The proposed platform is generated using a meta-modelling based hardware design tool. This tool aims to bridge the gap between hardware and software design, taking advantage of trends in both areas to work with higher level description languages. Shifting toward a software-oriented approach supports the concept of abstraction and inheritance, which in turn enables the generic specification of modules [8]. With such a generic module in place, the addition of a component translates merely to the instantiation and configuration of the component in the platform's template as opposed to re-writing the Register Transfer Level (RTL) code. This facilitates rapid modification/redesign, hence drastically reducing development time.

The proposed platform consists of the following components on an FPGA: a Serial Peripheral Interface (SPI) slave peripheral for communication and the IPs under testing. These are represented in Fig,1.

The novelty of this platform is the configurable communication protocol to adapt to the requirements of the IPs under testing. From high level customisations like the frame format, to the very minute details like clock polarity and phase are made possible on this platform.

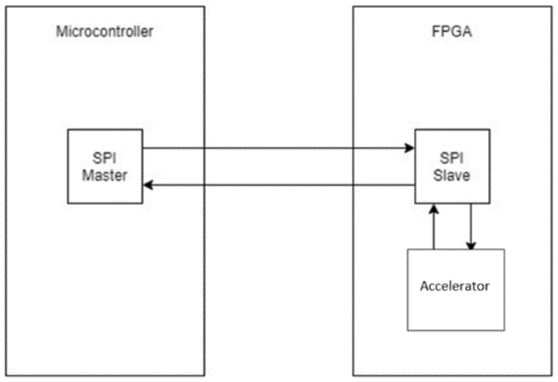


Fig 1: FPGA Testing Platform

With the platform in place, testing is done by interfacing it with an external master (e.g. microcontroller) on which the entire algorithm is written, which will potentially test the integration of the IP into the system. Flow of execution is transferred to the FPGA platform when the operation to be accelerated by the IP under testing is reached. Therefore with this arrangement it is made possible to test the functionality of an IP as part of a system even when the IP is not physically fabricated onto the master hardware.

## *B. Design Specification*

The platform is dependent on two templates for generating RTL one for the SPI Peripheral and another for the Register interface. Metamodeling enables the SPI hardware to be highly customizable. For different applications different SPIs are needed. Depending on the template, large or small changes can occur in the HW components. In the second template, based on the frame format specification, register interface hardware is generated. The hardware concepts which are presented in this section are only representative layouts. With this in mind the following chapter explains the concept of the SPI peripheral, the control and IP logic, and the interface between these modules.

The FPGA platform consists of a Serial Peripheral Interface (SPI) slave peripheral, register interface logic and a state machine to control data flow between IPs under testing, as depicted in Fig.2. The synthesizable Register Transfer Level (RTL) is generated by specifying configuration in the template created for the FPGA platform and SPI communication interface in the metamodeling hardware generation tool. In order to cater to the various IPs being tested, the communication protocol is also widely configurable. From high level customisations like the frame format, to very minute details like clock polarity and phase. Finally, the IPs under testing are integrated into a top level Finite State Machine (FSM) which controls the flow of execution.

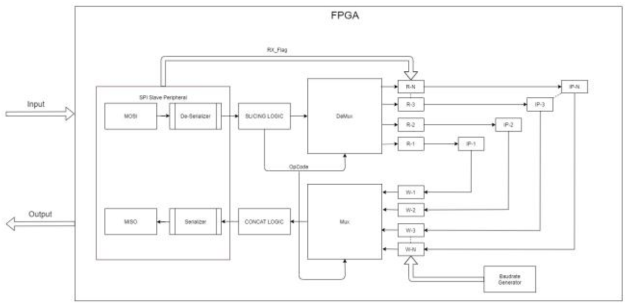


Fig 2: System Block Diagram

## *C. Design Extensions*

Tests take effect on the device by interfacing it with an external master (e.g. microcontroller), which includes the full algorithm to test functionality of the IP.

### *(a) Interface with XMC*

The above generated RTL is synthesized and burnt onto an Arty A7-35T FPGA. This FPGA is used as the system clock frequency is much superior to that of the XMC Microcontroller on which the main algorithm is executed. To see best improvement in results, the accelerator IPs are run on this board. The firmware and driver code for the XMC Controller is also auto-generated using the same metamodelling based hardware tool [9] which generates the RTL of the platform. Therefore the system diagram now consists of an intermediate layer between the FPGA and XMC Controller that fills in the firmware and driver code for the XMC (as shown in Fig.3)

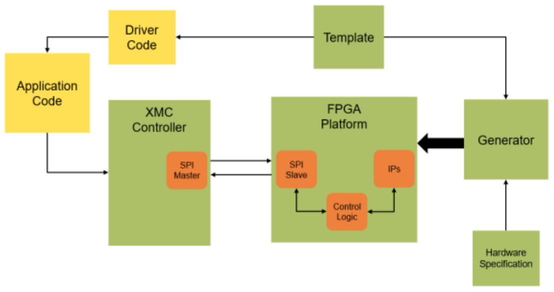


Fig 3: Accelerator Testing Setup

### *(b) Automatic datasheet and derivative generation*

Since the metamodelling-based hardware generation tool enables numerous variations in hardware specification, it is critical to support the same with meaningful documentation to provide the users of the platform the ease of referring to a datasheet to generate the specific hardware required for their application. The basic internal flow of the documentation generation process is shown in Fig 4.

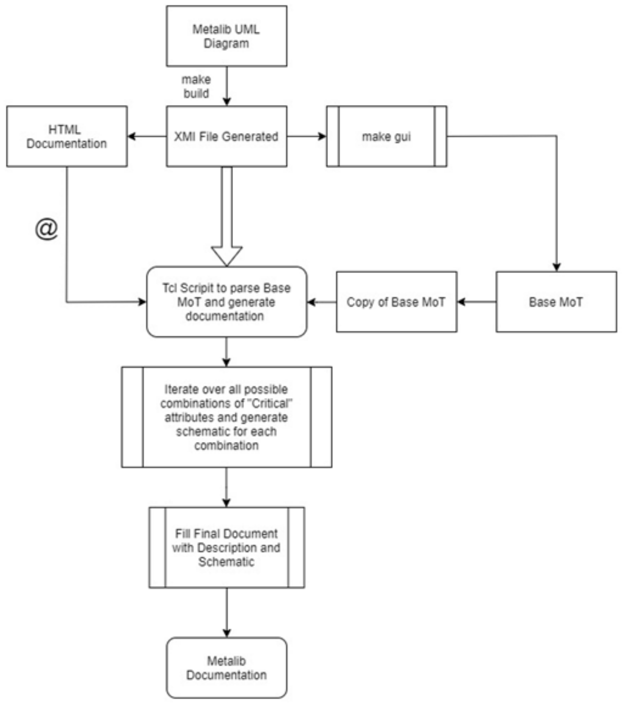
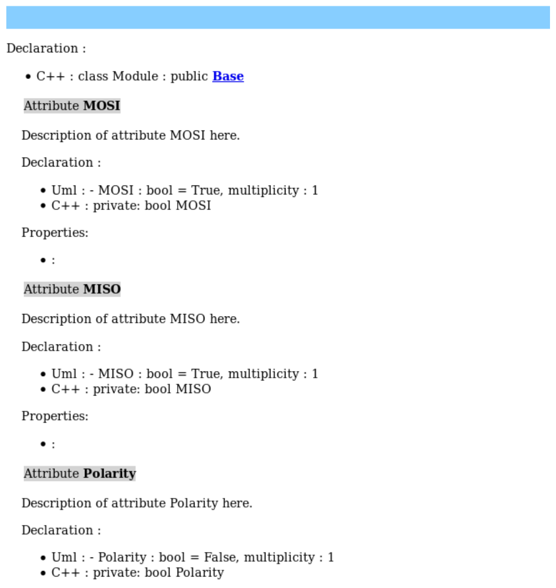

Metalib UML Diagram
make build
HTML Documentation
XMI File Generated
make gui
@
Tcl Scripit to parse Base MoT and generate documentation
Copy of Base MoT
Base MoT
Iterate over all possible combinations of "Critical" attributes and generate schematic for each combination
Fill Final Document with Description and Schematic
Metalib Documentation

Fig 4: Flow of Datasheet Generator

There are two parts to a datasheet: the port description and the schematic. The port description is derived from attribute and class description from the template fed to the metamodelling framework.

The schematic is generated using Nlview (an automatic schematic diagram generation engine), based on an intermediate model file used by the RTL generators. An example of the documentation and schematic is shown in Fig.5.

The generated documentation consists of attribute descriptions and schematics that guide the users on attaining their requirements using the template. A sample document with schematic is shown in Fig 5.



Declaration :

- C++ : class Module : public **Base**

Attribute **MOSI**

Description of attribute MOSI here.

Declaration :

- Uml : - MOSI : bool = True, multiplicity : 1
- C++ : private: bool MOSI

Properties:

- :

Attribute **MISO**

Description of attribute MISO here.

Declaration :

- Uml : - MISO : bool = True, multiplicity : 1
- C++ : private: bool MISO

Properties:

- :

Attribute **Polarity**

Description of attribute Polarity here.

Declaration :

- Uml : - Polarity : bool = False, multiplicity : 1
- C++ : private: bool Polarity

Fig 5(a): Attributes/components of the hardware specification are automatically documented in a HTML page.

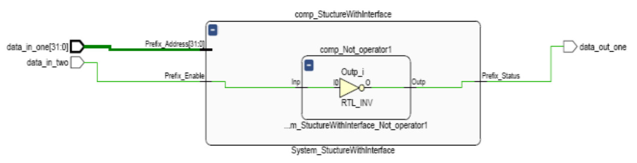
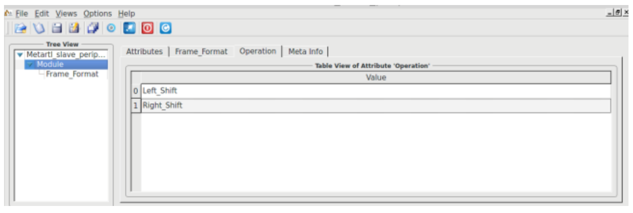


Fig 5(b): Schematic diagrams are created using the RTL generated for the hardware specification

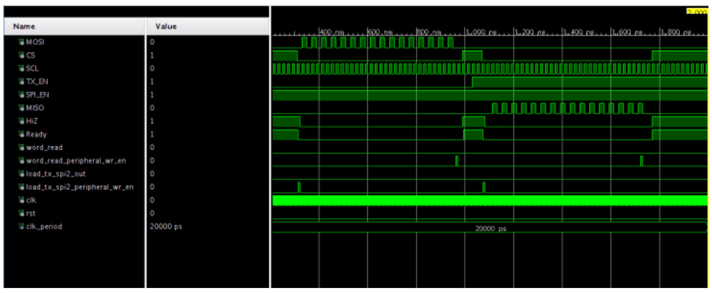
## 4. RESULTS

Simulation tests (as shown in Fig 7) were performed on the FPGA platform to detect register and line delays to synchronise the SPI communication with the XMC. To improve accuracy, register enables are controlled using a baud rate generator which precisely gives a pulse when data is available at the input of the registers [10]. Additionally the interface to synthesize the RTL was tested with various combinations of IPs as shown in Fig 6.



| | Value |
|---|---|
| 0 | Left_Shift |
| 1 | Right_Shift |

Fig 6: GUI to generate specific hardware using template



| Name | Value |
|---|---|
| MOSI | 0 |
| CS | 1 |
| SCL | 0 |
| TX_EN | 1 |
| SPI_EN | 1 |
| MISO | 0 |
| HiZ | 1 |
| Ready | 1 |
| word_read | 0 |
| word_read_peripheral_wr_en | 0 |
| load_tx_spi2_out | 0 |
| load_tx_spi2_peripheral_wr_en | 0 |
| clk | 0 |
| rst | 0 |
| clk_period | 20000 ps |

Fig 7: Validating synthesized hardware in Vivado

An FPGA-based sandbox testing platform as described has been successfully designed, developed and tested, with the RTL generated by the platform synthesized and tested on an Arty A7-35T FPGA. An accompanying framework of automated documentation of hardware modules through the development process has been integrated and tested as well.

Future developments would chiefly involve testing the viability of the platform in production architectural design workflows and augment the versatility of the system for more complex hardware generation problems.

## REFERENCES

[1] Tom Dillon, Jeremy Paatela, Guenter Dannoritzer, Scott Hussong, "Accelerating Algorithm Implementation in FPGA/ASIC Using Python", Dillon Engineering, Inc.
[2] Jianwen Zhu, "MetaRTL: Raising the Abstraction Level of RTL Design", Technical Report ECE-00-xx, Electrical and Computer Engineering, University of Toronto
[3] G. Z. Derek Lockhart and C. Batten, "PyMTL A Unified Framework for Vertically Integrated Computer Architecture Research", International Symposium onMicroarchitecture (MICRO47), 2014.
[4] Z. Han et al, "Towards a PythonBased One Language Ecosystem for Embedded Systems Automation", IEEE Nordic Circuits and Systems Conference, 2019
[5] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, Krste Asanovic, "Chisel: Constructing Hardware in a Scala Embedded Language", EECS Department, UC Berkeley, 2015
[6] R. S. Nikhil and K. R. Czeck, "BSV by Example", Bluespec, 2010
[7] O. Shacham et al., "Avoiding Game Over: Bringing Design to the Next Level", Design Automation Conference (DAC), 2012
[8] Frank P. Coyle and Mitchell A. Thornton, "From UML to HDL: A Model Driven Architectural Approach to Hardware-Software Co-Design", Technical Report, Computer Science and Engineering Dept. Southern Methodist University
[9] Drechsler et al, "Efficient Automatic Visualization of SystemC Designs", Forum on specification and Design Languages, FDL 2003, September 23-26, 2003, Frankfurt
[10] T Liu and Y Wang, "IP design of universal multiple devices SPI interface," 2011 IEEE International Conference on Anti-Counterfeiting, Security and Identification