IPXACT-Based RTL Generation Tool

Ahmad El-Shiekh[1], Ahmad El-Alfy[1], Ahmad Ammar[1], Mohamed Gamal[1], Mohammed Dessouky[1],
Khaled Salah[2], Hassan Mostafa[3]
[1]Department of electronics and communications, faculty of engineering, Ain Shams University, Egypt.
[2]Mentor Graphics, Cairo, Egypt.
[3]Electronics and Communications Engineering Department, Cairo University, Giza.
Email: [2]Khaled_mohamed@mentor.com, [3]hmostafa@uwaterloo.ca

***Abstract*— This paper proposes a new CAD tool that automates the RTL code generation based on the IPXACT standard (develop RTL code using XML files). Many related work generates RTL design using C language. In this work, the generation is based on XML descriptions. The tool is developed using Python. The generated RTL code can be synthesized by the synthesis tool like Design Compiler. Several commercial tools like MATLAB have this capability, but the proposed tool is faster and more configurable.**

Keywords—Register Transfer Level (RTL), IPXACT, Verilog, Python, XML.

I. INTRODUCTION

In a recent study by Harry Foster, Chief Scientist at Mentor Graphics, he accentuated that about 20% of the design projects in the industry have 70% of the project time. As today's system on chip (SoC) designs incorporate more IP components and time-to-market pressures rise, designers are looking for a way to build and update designs easily [1].

The IPXACT is a powerful standard developed by the SPIRIT consortium [2], that gives the ability to write the electronic components and designs in a standard data exchange format (XML) file, which is both human readable and machine processable. The saved data in the XML file can later be used to generate the RTL design. This method eliminates the complexity of generating an RTL design using C language. It also gives the ability to provide a single description of the components to all customers, regardless of the design language or tools that they use and enable developers to transfer designs between environments that use different design languages.

The aim of this work is to move the RTL design to a higher level by generating RTL code directly from an IPXACT xml file. This methodology eliminates the need to manually write the RTL code. Moreover, it can be used to automatically generate RTL testcases. IPXACT is a simple XML file that adheres to standards set by the SPIRIT consortium. It describes in an understandable way, hardware components and the hardware designs. The proposed methodology will be applied to many SoC bus protocols to verify it [3].

The rest of this paper is organized as follows. In section II, traditional flow versus IPXACT flow is discussed. The detailed implementation of the proposed approach is discussed in section III. The experimental results are shown in section IV. Finally, section V concludes the paper.

II. BACKGROUND: TRADITIONAL FLOW VERSUS IPXACT FLOW

IPXACT is a standard that specifies how to describe different types of electronic IPs in the form of an XML document. It gives the ability to have a standard way to describe the main features of an IP such that the uses of the IP, both humans and tools can access the information in a professional automated fashion [4].

An IP component has several attributes that can be mapped directly to XML. Memory maps, registers, bus interfaces, ports, views, parameters, generators and file sets are some of the attributes that the component contains and can be mapped to XML. When multiple components are connected together, it becomes an IPXACT design file.

In the traditional design flow, there is no solid relation between the design process and the verification process. This wastes a lot of time and may delay the time to market. However, the IPXACT flow solves this problem as the design and the verification processes can be done in one task instead of two tasks.

III. THE PROPOSED TOOL

A. The Proposed Tool: Methodology

The flowchart of the proposed tool is shown in Fig. 1, where we have a huge library of the main digital design building blocks that described in XML files. The user can configure the specifications of each block such as the initial value of a register, the type of the FSM either Mealy or Moore and many more of specs.

Supported blocks of the tool are computational blocks, communication blocks, storage blocks, arithmetic and logic unit, finite state machines, multiplexers, counters, registers, memories.

After the selection of the blocks, the user can connect these blocks in the block diagram window, so he can have a complete system of his own design. Directly after this step the user generates the IPXACT XML file for the system. The stored data in the IPXACT XML contains the blocks of the systems, specifications of these blocks and how these blocks are connected together.

To generate Verilog code, each block in the tool has a Verilog template which is generated depending on the XML specifications which had been chosen by the user.

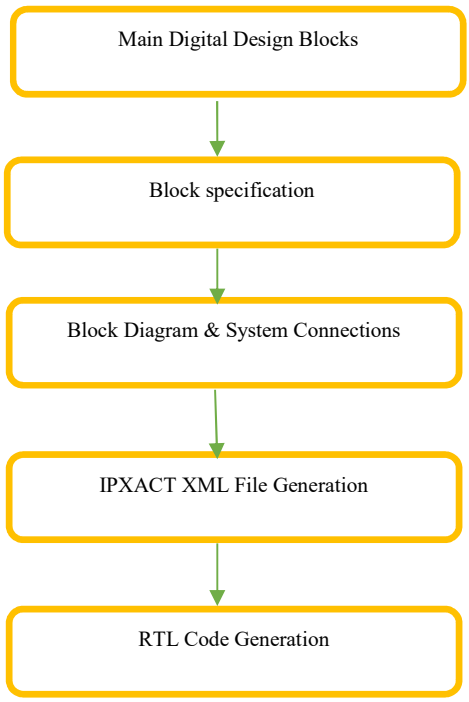

Fig.1 Flowchart of the proposed tool

B. *The Proposed Tool: Implementation*

The proposed tool is implemented using python due to its ease of use when manipulating strings and outputs files, and when dealing with GUI. It is an open source language; its libraries are available and can be run on all environments with no problems.

A snapshot of the implemented tool is shown in Fig. 2. The main building blocks window is shown in Fig. 3.

Fig. 2 GUI Interface of the proposed tool.

Fig. 3 Main building blocks window. The user adds from this library the blocks he needs in order to design his system.

C. *The Proposed Tool: An Illustrative Example*

An illustrative example for descripting 7 bits register is shown in Fig. 4. Firs, wet set the name of the register then we start to choose how we want to implement our register. So, we name a sub from the register called the ID and we set its size to 3 bits. Next, we choose the attribute to be read only and we can set the initial value to binary, decimal or hexadecimal. The first three bits represents the ID of the register, the 4^{th} to the 7^{th} bits represents data value. XML generated file for the proposed register is shown in listing 1.

In this generated IPXACT XML, the user can read the description of his register. It's a must to generate the XML file first because the python script will use it to generate the Verilog code for the block. It first takes the name chosen by the user to rename the verilog module, then it checks the attributes so that he can generates the inputs and the outputs inside the verilog module. After this it sets the initial value of each sub with the chosen type as well.

Finally it writes a set of if conditions that expresses the attribute of the subs whether it's a read only, write only or read/write sub register. This method shows that the implementation of any RTL code almost takes no time which we will lead to fast time-to-market and eliminates the necessity to write a complete code for the digital design blocks. The generated verilog code for the proposed register is shown in listing 2.

Fig. 4 Register specifications window.

Listing 1: XML generated file for the proposed register

```
<Register1>
    <reg_Register>
        <sub>
            <name>ID</name>
            <size>3</size>
            <attribute>Read only</attribute>
            <initial_value>0</initial_value>
            <init_val_type>Binary</init_val_type>
            <start>0</start>
            <end>2</end>
        </sub>
        <sub>
            <name>Data</name>
            <size>4</size>
            <attribute>Read/Write</attribute>
            <initial_value>0</initial_value>
            <init_val_type>Decimal</init_val_type>
            <start>3</start>
            <end>6</end>
        </sub>
    </reg_Register>
</Register1>
```

Listing 2: Verilog generated file for the proposed register

```
module reg_Register1 (
    input clk,
    input wr,
    input rst,
    input [3:0] Data,
    output reg [6:0] data_out
        );
always @(posedge clk or posedge rst)
begin
    data_out[2:0] <= 3'b0;
    data_out[6:3] <= 4'h0;
    if (rst)
    begin
        data_out[6:3] <= 0;
    end
    else if (wr)
    begin
        data_out[6:3] <= Data[3:0];
    end
end
endmodule
```

IV. EVALUATION OF THE PROPOSED TOOL: I2C AND DDR4 AS A CASE STUDY

A. I²C Protocol Overview

I²C is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. It's very suitable for applications that need occasional communication over a short distance between many devices. The I²C standard is a true multi-master bus including collision detection and arbitration that prevents data corruption if two or more masters tries to control the bus simultaneously [7].

I²C has the following features:

- Multi Master Operation.
- Software programmable clock frequency.
- Clock Stretching and Wait state generation.
- Software programmable acknowledge bit.
- Bus busy detection.
- Supports 7 and 10bit addressing mode.
- Static synchronous design.
- Fully synthesizable

B. DDR4 Protocol Overview

The DDR4 SDRAM is a high-speed dynamic random-access memory internally configured as sixteen-banks, 4 bank group with 4 banks for each bank group for x4/x8 and eight-banks, 2 bank group with 4 banks for each bank-group for x16 DRAM [5].

The DDR4 protocol has many operations, however the most important are activation, read and write. The **ACTIVE** command is used to open (or activate) a row in a particular bank for a subsequent access,

The **READ** command is used to initiate a burst read access to an active row, and The **WRITE** command is used to initiate a burst write access to an active row [6].

C. DDR4 and I^2C Implmentation and Verification

We implement them by using the finite state machine block as depicted in Fig. 5. From the GUI, we determine the number of states, the type of encoding, the machine type either it's mealy or Moore, the reset state number, the number of both input bits and output bits. After generating the Verilog code, we create a testbench to verify it as shown in Fig. 6 and Fig. 7 for I2C and DDR4 respectively.

It's obvious that using the proposed tool we save a lot of time. TABLE I provides a comparison between the hand-crafted method and the proposed tool in terms of run-time, where results show that the proposed tool is faster than the handcrafted one [8]–[10].

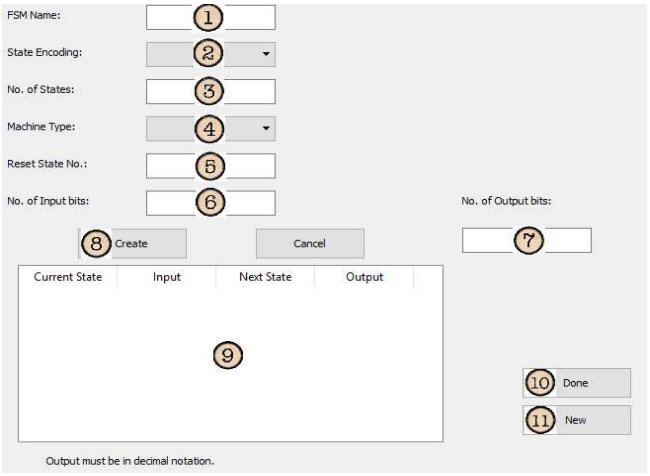

Fig. 5 FSM specifications window.

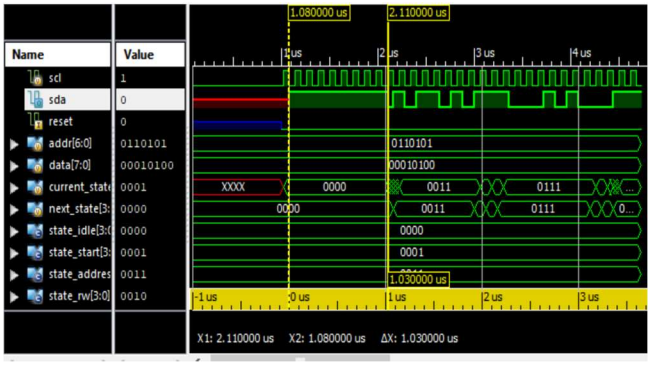

Fig. 6 I2C simulation results.

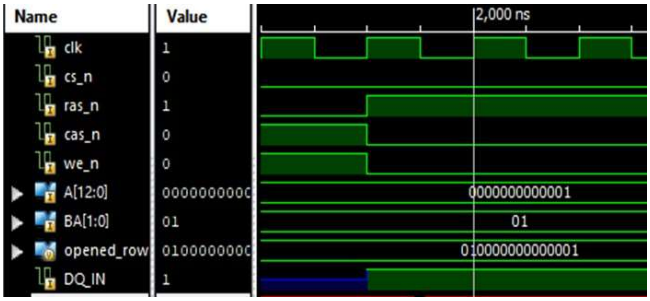

Fig. 7 DDR4 simulation results: write operation.

TABLE I
COMPARISON OF THE PROPOSED TOOL AND HAND-CRAFTED IMPLEMENTATION

Protocol	Run Time		
	Hand-Crafted	This Work (Secs)	[11]
I2C	2 days	5	8
DDR4	20 days	3	5

V. CONCLUSIONS

Over the past decade, the increasing hardware design complexity uncovered the necessity for moving the RTL design to a higher level by generating RTL code directly from an IPXACT XML file. The manual implementation of RTL is a challenging and time-consuming process. Accordingly, automation of RTL would lead to reduce the pressure of the time-to-market problem. This paper proposes a new CAD tool that automates the RTL code generation based on the IPXACT standard. The tool offers generating the RTL code to the most important of digital design blocks through a GUI interface and also can connect the blocks together to generate an RTL code for a complete system. Moreover, the generated code is synthesizable.

REFRENCES

[1] Harry Foster, "Trends in functional verification: a 2014 industry study", DAC, 2015.
[2] IEEE. (2014). IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows.
[3] Timo D. Hämäläinen, & Esko Pekkarinen. (2015). Kactus2: Open Source IP-XACT tool.
[4] Accellera. (2018). IP-XACT User Guide.
[5] JEDEC. (2012). DDR4 SDRAM.
[6] JEDEC. (2003). Double Data Rate (DDR).
[7] I2C-Master Core Specifications, Richard Herveille, July 3, 2003.
[8] K. Salah, M. AbdelSalam. "IP cores design from specifications to production: Modeling, verification, optimization, and protection." 25th International Conference on Microelectronics (ICM), 2013.
[9] K. Salah "A UVM-based smart functional verification platform: Concepts, pros, cons, and opportunities." 9th International Design & Test Symposium (IDT), 2014.
[10] K. Salah. "A Unified UVM Architecture for Flash-Based Memory." 18th International Workshop on Microprocessor and SOC Test and Verification (MTV), 2017.
[11] https://www.nutaq.com/matlab-hdl-coder-xilinx-system-generator.