Systematic Approach To Realize RTL From Design Specification

Vardhana M[1], Oswald Sunil Mendonca[2], Srinivas R[3]
Dept of Electronics and Communication Engineering
[1] NMAM Institute of Technology Nitte

Abstract- *Electronic Design Automation is gaining more and more scope in the design industry, where it is required to automate the design, which is modular in approach, and is confined to minor change. To reduce the design time, it is necessary to have design automation. In this paper we propose a systematic approach, to convert the given design specification to Register Transfer Logic (RTL).*

Keywords- RTL, EDA, Script

I. INTRODUCTION

Thisarticle discusses the need for automation and various methods that can be used to realize the design specification to RTL. Requirement and methodology for realizing the design RTL from specification is discussed.

II. DESIGN SPECIFICATION

Methodology used to capture the design specification and design description is described in this chapter.

A.IP-XACT

IP-XACT is a Extensible Markup Language (XML) that adheres to standards set by the SPIRIT consortium. It describes a understandable way, hardware components and the hardware designs. IP-XACT was created by the SPIRIT Consortium as a standard to enable automated configuration and integration [1].

The standard was established in 2003 by the SPIRIT consortium. It was submitted to the Institute of Electrical and Electronics Engineers Standards Association and received approval in June 2009 [2].

It became available as the IEEE 1685 IP-XACT standard in June 2010. The IP-XACT forms that are standardized include, components, systems, bus interfaces and connections, abstractions of those buses, details of the components including address maps, register and field descriptions, file set descriptions for use in automating design, verification, documentation and use flows for electronic systems.

IP-XACT is a simple XML document that contains a description of electronic components and their design. It is like every other XML document, set of tags their attributes and information contained within those tags. These tags should represent correctly a component that is synthesizable, for example consider a register that has been described using the IP-XACT format in fig. 1 [2], taken from online source.

```
<?xml version="1.0" encoding="UTF-8" ?>
<spirit:design
xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4"
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4
   http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4/index.xsd">
  <spirit:vendor>spiritconsortium.org</spirit:vendor>
  <spirit:library>simple_lib</spirit:library>
  <spirit:name>simple_design</spirit:name>
  <spirit:version>1.0</spirit:version>
  <spirit:componentInstances>
<spirit:interconnections>
</spirit:design>
```

Fig 1. IP-XACT XML

B.Excel Based

Apart from the existing tools or methods to capture the design specification, excel based design specification is widely used in industry. Systematic entries in excel, with pre-specified format, could greatly help in RTL generation. Well defined excel will have all the information with respect to the design. Once such excel used for top level integration is specified in fig 2.

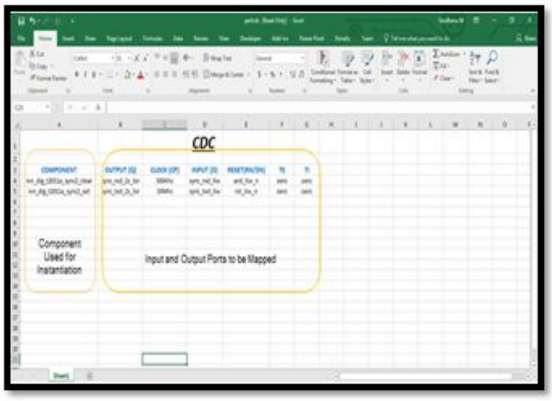

Fig 2. Top Level Integration Specification Captured in Excel

III. PROPOSED APPROACH

The proposed approach involves the various steps as depicted in the figure. The design specification captured using XML or excel, is segregated into several sections, few of which are depicted in the figure. These sections may involve the functionality of the design, the interfaces and ports of the design, such as intermediate connections and inter module connections, via which the design can interact with other external modules. The specification can be further classified into instances, which is called by the design or instantiated. These instances may be basic primitive modules, or design specific modules.

Once the specification is divided into several class, Perl scripting can be used to generate the required RTL for the design. The Perl script takes all the segregated class and builds up the basic RTL skeleton. The script can use base modules if required to build the RTL. General block diagram of the proposed approach is presented in fig 3.

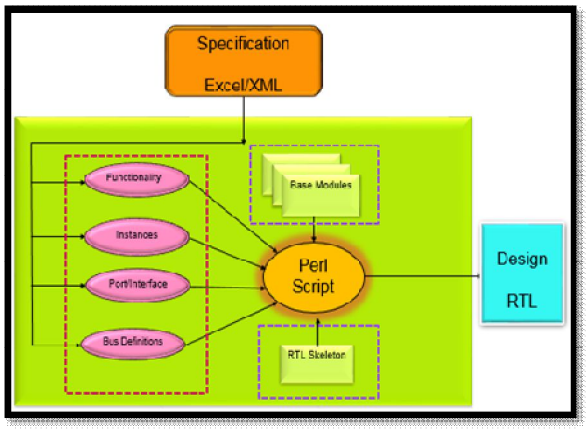

Fig 3. Proposed Schema for RTL Generation from Specification

IV. DESIGN EXAMPLE

This section describes some of the examples, to automate the process of manual RTL coding of the design.

Let us consider a design, whose behavior is defined in the form of Finite State Machine (FSM). The FSM definition can be captured in the excel sheet, which can be further used generate the RTL. the column heading may include name of the state, output signals and its value at each state, state transition conditions etc. simple example is shown in the fig 4.

	A	B	C	D	E
1	PRESENT STATE NAME	STATE OUTPUT	NEXT STATE NAME	TRANSITION CONDITION	SIGNAL EVALUATION
2					
3	STATE_0	COUNT	STATE_1	CLK	COUNT+1
4	STATE_1	COUNT	STATE_2	CLK	COUNT+1
5	STATE_2	COUNT	STATE_3	CLK	COUNT+1
6	STATE_3	COUNT	STATE_4	CLK	COUNT+1
7	STATE_4	COUNT	STATE_5	CLK	COUNT+1
8	STATE_5	COUNT	STATE_6	CLK	COUNT+1
9	STATE_5	COUNT	STATE_7	CLK	COUNT+1
10	STATE_7	COUNT	STATE_0	CLK	COUNT+1

Fig 4. Design Specification for Counter

Fig shows the simple excel, where the specification of the design is captured in the excel sheet. The above design is an example of 3-bit counter.

From the excel, it is clear that, state transition occurs from one state to other when there is a CLK signal. Again, it can be also specified at what CLK edge the transition should occur., example, pos_edge (clk) or neg_edge (clk). At every CLK signal, the state transition occurs.

Evaluation at each state is specified in column SIGNAL EVALUATION. At each state, the register named COUNT is incremented by one. Typically, for a 3-bit counter will have the size of the register as [2:0]. This can also be specified in the separate column. The column STATE OUTPUT specifies the value of the signal that has to be outputted.The generated code may look as in fig 5.

```
module counter(clk,rst,count);
input clk;
input rst;
output reg [2:0]count;
typedef enum {state_0,state_1,state_2,state_3,state_4,state_5,state_6,state_7} state;
state present_state,next_state;
always@(posedge(clk))
begin
if(rst)
begin
present_state=state_0;
next_state=state_1;
count=0;
end
else
begin
present_state=next_state;
end
end
always@(present_state)
begin
case (present_state)
state_0:begin count=0; next_state = state_1; end
state_1:begin count+=1;next_state = state_2; end
state_2:begin count+=1; next_state = state_3; end
state_3:begin count+=1; next_state = state_4; end
state_4:begin count+=1; next_state = state_5; end
state_5:begin count+=1; next_state = state_6; end
state_6:begin count+=1; next_state = state_7; end
state_7:begin count+=1; next_state = state_0; end
endcase
end
endmodule
```

Fig. 5. RTL for 3Bit Counter

V. CONCLUSION

In this paper, systematic, way to capture design specification, which enable the automation of RTL coding of the design has been discussed. Simple design of a counter has been captured and the RTL for the same is presented.

VI. ACKNOWLEDGEMENT

Author(s) would like to thank each one who helped in writing the paper.

REFERENCES

[1] http://www.accellera.org/activities/working-groups/ip-xact

[2] “IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows”, IEEE Computer Society and the IEEE Standards Association Corporate Advisory Group