Volume: 8 Issue: VII Month of publication: July 2020
DOI: https://doi.org/10.22214/ijraset.2020.30262

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue VII July 2020- Available at www.ijraset.com

Design and Synthesis of Custom IP in VLSI

Bejagam Divya[1], P. Venkatapathi[2], R. Madhuri Muddapu [3]
[1]M.Tech Scholar, [2]Associate Professor, Department of Electronics and Communication Engineering,
Malla Reddy College Of Engineering
[3]Scientist 'c', ECE Department, DRDO, RCI, Hyderabad

Abstract: The top module or sub-module of a project in the Vivado Design Suite can be packaged as an IP and it is available in the IP repository of the Vivado IP catalog for reuse.
This paper exhibit how chosen modules in an RTL project can be packaged as IP for reuse. IP packaging is IP-XACT-compliant, where IP-XACT is a XML format that defines and describes electronic components and their designs.
IP packaging also allows for different file sets to be part of the package. These file sets include simulation, test bench, example design, XDC constraints, and documents.
Keywords: Vivado, CustomIP, Xilinx, fifo, RTL.

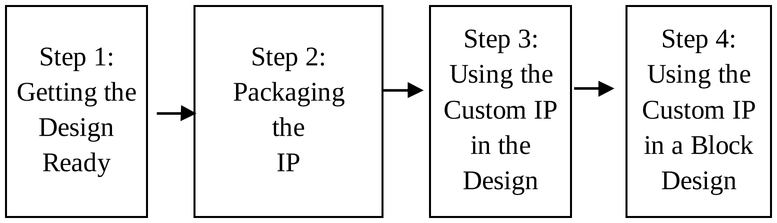
Objectives
- Select a sub module in an RTL project and begin IP packaging
- Select a repository location to create the new IP designation
- State basic options for IP packaging
- Define IP interfaces from the ports
- Specify IP customization parameters
- Add or remove files to/from many of the possible file groups throughout during packaging

I. INTRODUCTION

The Vivado Design Suite provides an IP-centric design flow that helps you to quickly turn the designs and algorithms into reusable IP. The Vivado IP Catalog is a united IP repository that provides the framework for the IP-centric design flow. This catalog consolidates IP's from all sources including Xilinx® IP, thirdparty IP, and end-user designs targeted for reuse as IP into a single environment. The Vivado IP packager tool is a unique design reuse feature, which is based upon the IP-XACT standard. The IP packager tool provides you with the facility to package a design at any phase of the design flow and deploy the core as system-level IP. In this paper, I define a new custom IP from an active Vivado project, using the Create and Package IP wizard. Start with an existing design project in the Vivado IDE, define identification information for the new IP, add documentation to support its use, and add the IP to the IP Catalog. After packaging, you verify the new IP through synthesis in a separate design project. The project contains Verilog source files for a simple UART interface.
The existing design includes timing constraints defined in an XDC file (uart_top.xdc). These constraints were defined for the UART design as a standalone design. However, when packaged as an IP, the design inherits some of the needed constraints from the parent design. In this case, you must modify the XDC file to separate constraints the IP requires when used in the context of a parent design, and the constraints the IP requires when used out-of-context (OOC) in a standalone capacity. This requires splitting the current XDC file. We should prepare the design constraints prior to packaging the design for inclusion in the IP catalog; however, we can also perform these steps after packaging the IP.

II. PROPOSED WORK

General Flow


Step 1: Getting the Design Ready
Step 2: Packaging the IP
Step 3: Using the Custom IP in the Design
Step 4: Using the Custom IP in a Block Design

A. FIFO

"FIFO" stands for first-in, first-out

FIFOs are usually used in electronic circuits for buffering and flow manages between hardware and software. In its hardware form, a FIFO mainly consists of a set of read and write pointers, storage and control logic. Storage may be static random access memory (SRAM), flip-flops, latches or any other suitable form of storage. For FIFOs of non-trivial size, a dual-port SRAM is regularly used, where one port is dedicated to writing and the other to reading. A synchronous FIFO is a FIFO where the clock is used for both reading and writing is same. An asynchronous FIFO uses clocks for reading and writing are different. Asynchronous FIFOs introduce metastability issues. A general operation of an asynchronous FIFO uses a Gray code (or any unit distance code) for the read and write pointers to ensure reliable flag generation. One further note concerning flag generation is that one must essentially use pointer arithmetic to generate flags for asynchronous FIFO implementations. Conversely, one may use either a leaky bucket approach or pointer arithmetic to generate flags in synchronous FIFO implementations.

Examples of FIFO status flags include: full, empty etc.

The first known FIFO implemented in electronics was done by Peter Alfke in 1969 at Fairchild Semiconductors. Peter Alfke was later a director at Xilinx.

B. FIFO full-empty

A hardware FIFO is used for synchronization purposes. It is often implemented as a circular queue, and thus has two pointers:

1) Read pointer / read address register
2) Write pointer / write address register

Read and write addresses are originally both at the first memory location and the FIFO queue is *empty*.

C. FIFO Empty

When the read address register reaches the write address register, the FIFO triggers the *empty* signal.

D. FIFO Full

When the write address register reaches the read address register, the FIFO triggers the *full* signal.

In both cases, the read and write addresses end up being equal. To distinguish between the two situations, a simple and robust solution is to add one extra bit for each read and write address which is inverted each time the address wraps. With this set up, the disambiguation conditions are:

1) When the read address register equals the write address register, the FIFO is empty.
2) When the read address LSBs equal the write address LSBs and the extra MSBs are different, the FIFO is full.

III. RESULTS

A. Synthesis Report

1) Starting Synthesize: Time (s): cpu = 00:00:03 ; elapsed = 00:00:07 . Memory (MB): peak = 575.859 ; gain = 187.633
2) Finished Synthesize: Time (s): cpu = 00:00:03 ; elapsed = 00:00:08 . Memory (MB): peak = 639.816 ; gain = 251.590
3) Finished Constraint Validation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:08 . Memory (MB): peak = 639.816 ; gain = 251.590

```
Start Loading Part and Timing Information
Loading part: xc7z020clg484-1
Finished Loading Part and Timing Information : Time (s): cpu = 00:00:04 ; elapsed = 00:00:08 . Memory (MB): peak = 639.816 ;
gain = 251.
Finished RTL Optimization Phase 2 : Time (s): cpu = 00:00:04 ; elapsed = 00:00:08 . Memory (MB): peak = 639.816 ; gain =
251.590
```

a) Detailed RTL Component Info

```
+---Adders :
         2 Input    3 Bit     Adders := 2
         3 Input    3 Bit     Adders := 2
+---Registers :
               32 Bit    Registers := 1
                3 Bit    Registers := 1
```

```
+---RAMs :
                    256 Bit        RAMs := 1
+---Muxes :
            2 Input      3 Bit        Muxes := 21
            2 Input      1 Bit        Muxes := 2
Finished RTL Component Statistics
Start RTL Hierarchical Component Statistics
Hierarchical RTL Component report
Module myfifoIP_test
```

b) Detailed RTL Component Info

```
+---Adders :
            2 Input      3 Bit       Adders := 2
            3 Input      3 Bit       Adders := 2
+---Registers :
                     32 Bit    Registers := 1
                      3 Bit    Registers := 1
+---RAMs :
                    256 Bit        RAMs := 1
+---Muxes :
            2 Input      3 Bit        Muxes := 21
            2 Input      1 Bit        Muxes := 2
Finished RTL Hierarchical Component Statistics
Start Part Resource Summary
```

c) Part Resources

```
DSPs: 220 (col length:60)
BRAMs: 280 (col length: RAMB18 60 RAMB36 30)
Finished Part Resource Summary
No constraint files found.
Start Cross Boundary and Area Optimization
Finished Cross Boundary and Area Optimization : Time (s): cpu = 00:00:07 ; elapsed = 00:00:17 . Memory (MB): peak = 806.379 ; gain = 418.152
Start ROM, RAM, DSP and Shift Register Reporting
Distributed RAM: Preliminary Mapping  Report (see note below)
+--------------+------------+-----------+----------------------+--------------+
|Module Name   | RTL Object | Inference | Size (Depth x Width) | Primitives   |
+--------------+------------+-----------+----------------------+--------------+
|myfifoIP_test | FIFO_reg   | Implied   | 8 x 32               | RAM32M x 6   |
+--------------+------------+-----------+----------------------+--------------+
Note: The table above is a preliminary report that shows the Distributed RAMs at the current stage of the synthesis flow. Some Distributed RAMs may be reimplemented as non Distributed RAM primitives later in the synthesis flow. Multiple instantiated RAMs are reported only once.
Finished ROM, RAM, DSP and Shift Register Reporting
```

d) Report RTL Partitions

```
+-+--------------+------------+----------+
| |RTL Partition |Replication |Instances |
+-+--------------+------------+----------+
+-+--------------+------------+----------+
No constraint files found.
Start Timing Optimization
Finished Timing Optimization : Time (s): cpu = 00:00:07 ; elapsed = 00:00:17 . Memory (MB): peak = 811.871 ; gain = 423.645
Start ROM, RAM, DSP and Shift Register Reporting
Distributed RAM: Final Mapping  Report
+--------------+------------+-----------+----------------------+--------------+
|Module Name   | RTL Object | Inference | Size (Depth x Width) | Primitives   |
+--------------+------------+-----------+----------------------+--------------+
|myfifoIP_test | FIFO_reg   | Implied   | 8 x 32               | RAM32M x 6   |
+--------------+------------+-----------+----------------------+--------------+
Finished ROM, RAM, DSP and Shift Register Reporting
```

e) Report RTL Partitions

```
+-+--------------+------------+----------+
| |RTL Partition |Replication |Instances |
+-+--------------+------------+----------+
+-+--------------+------------+----------+
Start Technology Mapping
Finished Technology Mapping : Time (s): cpu = 00:00:07 ; elapsed = 00:00:17 . Memory (MB): peak = 811.871 ; gain = 423.645
Report RTL Partitions:
+-+--------------+------------+----------+
| |RTL Partition |Replication |Instances |
+-+--------------+------------+----------+
+-+--------------+------------+----------+
Start IO Insertion
Finished IO Insertion : Time (s): cpu = 00:00:09 ; elapsed = 00:00:19 . Memory (MB): peak = 811.871 ; gain = 423.645
```

f) Report Check Netlist

```
+------+------------------+-------+---------+-------+------------------+
|      |Item              |Errors |Warnings |Status |Description       |
+------+------------------+-------+---------+-------+------------------+
|1     |multi_driven_nets |      0|        0|Passed |Multi driven nets |
+------+------------------+-------+---------+-------+------------------+
Start Renaming Generated Instances
Finished Renaming Generated Instances : Time (s): cpu = 00:00:09 ; elapsed = 00:00:19 . Memory (MB): peak = 811.871 ; gain =
423.645
```

g) Report RTL Partitions

```
+-+--------------+------------+----------+
| |RTL Partition |Replication |Instances |
+-+--------------+------------+----------+
+-+--------------+------------+----------+
Start Rebuilding User Hierarchy
Finished Rebuilding User Hierarchy : Time (s): cpu = 00:00:09 ; elapsed = 00:00:19 . Memory (MB): peak = 811.871 ; gain =
423.645
Start Renaming Generated Ports
Finished Renaming Generated Ports : Time (s): cpu = 00:00:09 ; elapsed = 00:00:19 . Memory (MB): peak = 811.871 ; gain =
423.645
Start Handling Custom Attributes
```

```
Finished Handling Custom Attributes : Time (s): cpu = 00:00:09 ; elapsed = 00:00:19 . Memory (MB): peak = 811.871 ; gain =
423.645
Start Renaming Generated Nets
Finished Renaming Generated Nets : Time (s): cpu = 00:00:09 ; elapsed = 00:00:19 . Memory (MB): peak = 811.871 ; gain =
423.645
Start Writing Synthesis Report
```

h) Report BlackBoxes

```
+-+--------------+----------+
| |BlackBox name |Instances |
+-+--------------+----------+
+-+--------------+----------+
```

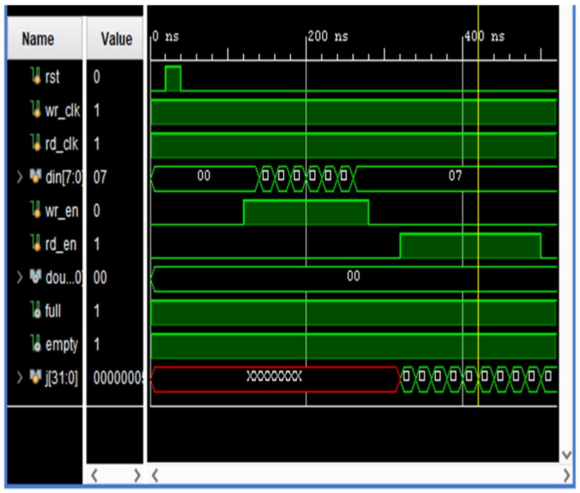
i) Report Cell Usage

```
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |LUT2   |     1|
|3     |LUT3   |     3|
|4     |LUT4   |     4|
|5     |LUT5   |     5|
|6     |LUT6   |    11|
|7     |RAM32M |     6|
|8     |FDRE   |    41|
|9     |IBUF   |    37|
|10    |OBUF   |    34|
+------+-------+------+
```

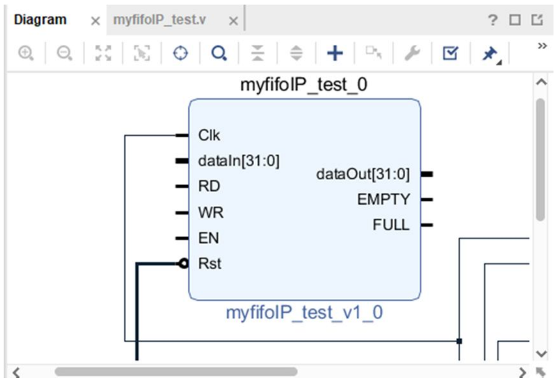
j) Report Instance Areas

```
+------+---------+-------+------+
|      |Instance |Module |Cells |
+------+---------+-------+------+
|1     |top      |       |   143|
+------+---------+-------+------+
Finished Writing Synthesis Report : Time (s): cpu = 00:00:09 ; elapsed = 00:00:19 . Memory (MB): peak = 811.871 ; gain =
423.645
Synthesis finished with 0 errors, 0 critical warnings and 2 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:00:09 ; elapsed = 00:00:19 . Memory (MB): peak = 811.871 ; gain = 423.645
Synthesis Optimization Complete : Time (s): cpu = 00:00:09 ; elapsed = 00:00:19 . Memory (MB): peak = 811.871 ; gain = 423.645
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 927.316 ; gain = 0.000
A total of 6 instances were transformed.
RAM32M => RAM32M (RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMS32, RAMS32): 6 instances
16 Infos, 2 Warnings, 0 Critical Warnings and 0 Errors encountered.
synth_design completed successfully
synth_design: Time (s): cpu = 00:00:14 ; elapsed = 00:00:29 . Memory (MB): peak = 927.316 ; gain = 543.074
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 927.316 ; gain = 0.000
```

B. Simulation Result



C. Custom IP



IV. CONCLUSION

The proposed design of custom IP (FIFO) is simulated and synthesized in Xilinx Vivado 19.1. The source code is written in verilog. This paper is concentrating on creating and packaging custom IP. By using IP's in the project we can reach our requirements fastly. For that, we need the package the sub module as an IP and then add that IP to the IP repository by that we can reuse it and we can also add that IP to the IP catalog.

REFERENCES

[1] L. E. Carlson and J. F. Sullivan, "Hands-on Engineering: Learning by Doing in the Integrated Teaching and Learning Program," International Journal of Engineering Education, Vol. 15, No. 1, pp. 20-31, 1999
[2] Synopsys University program, Synopsys Inc., Mountain View, CA: http://www.synopsys.com
[3] Nano-electronics and Computing Research Center, School of Engineering, San Francisco State University, San Francisco, CA: http://userwww.sfsu.edu/~necrc/
[4] Jan Rabaey, A. Chandrakasan, and B. Nikolic, Digital Integrated Circuits (2nd Edition) , Prentice Hall, 2003
[5] M Alioto, G. Di. Cataldo, G. Palumbo, "Mixed Full Adder topologies for high performance low power arithmetic circuits", Elsevier Microelectronics Journal 38 (2007) 130 – 139.