Configuration Bitstream Mapping with Programmable Resources on Spartan-3A FPGA using XDL and FAR

Pravin N. Matte [#1], Dr. Dilip D. Shah [*2]

[#] Department of Electronics and Telecommunication,
G.H. Raisoni College of Engineering and Management, Wagholi, Pune, India
[*]Department of Electronics and Telecommunication Engineering,
JSPMs Imperial College of Engineering and Research, Wagholi, Pune, India.
[1] pravin.matte@raisoni.net; [2]dilip.d.shah@gmail.com

***Abstract*— Configuration bitstream in FPGA is specific to vendor. It is difficult to understand bitstream format and eventually impossible to know how bits are placed in LUTs, flip flop, multiplexer, input and output interconnection network in FPGA. Normally user is not interested in knowing internals of FPGA, however it is believed that by opening the internals of FPGA, development would become very fast as it is achieved by open source community. In this paper we have proposed a simplified methodology to map configuration bitstream to its programmable resources targeting Spartan-3A FPGA. A simple AND gate design implemented in VHDL under different constraints. Correlation between different locations of AND gate is established with frame address register, XDL file report, FPGA editor and device view in PlanAhead.**

Keyword - Bitstream, Configuration memory, FPGA, FAR, NCD, XDL

I. INTRODUCTION

It is widely believed that the analysis of bitstream format is a tedious task. Increase in the complexity of digital circuit design demand for FPGA with more resources is increasing. FPGA has applications in wide variety of domains like space, telecommunications, networking, handheld devices etc. FPGA based industries demand CAD tools for design and development to meet time-to-market design metric. Like in open source communities, FPGAs development techniques are not open and certain things are vendor specific. Hence third party tools development has limited scope. If bitstream format would have been opened to users then possibility of cloning the device and IPs would have increased. Development on design tool side demands FPGAs internal details. Knowing FPGA bitstream definitely helps further development. Beside this, research communities and academicians are interested in knowing bitstream details. Limited work has been done in this direction.

A bitstream is a file that contains programming information for an FPGA. FPGAs bitstream is generated by CAD tools and configures (reconfigures) resources on FPGA fabrics to implement desired design. Fig. 1 depicts general structure of bitstream file during configuration. Bitstream file is viewed as file containing header for device name, architecture, and date information. After the start header it has synchronization word, device identification along with various device initialization commands. Most of the data is for actual configuration of FPGA. Last part of bitstream again contains command registers like CRC, DESYNC etc. and then FPGA goes into start sequence operation. These details are contains in bitstream file. Each FPGA device has its own bitstream format to configure itself.

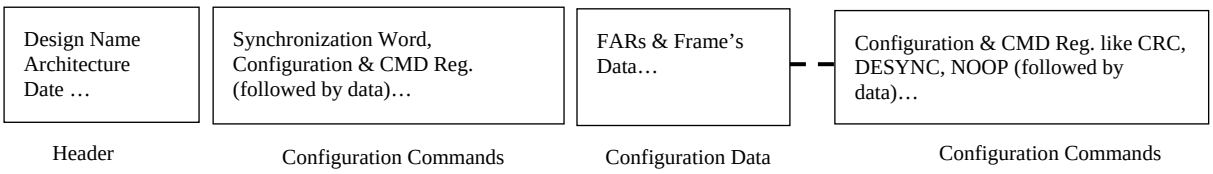


Fig. 1 General structure of bitstream file

Though fair amount of work is done to know bitstream format, this paper presents simple approach to understand FPGA bitstream and its mapping with resources on FPGA. Section II focused on related work done. FPGA generic architecture, design flow and bitstream format is described in section III. Preparation of background information is presented in section IV. A simple approach to analyze FPGA bitstream is elaborated in section V targeting Spartan-3A Xilinx FPGA. Results are discussed in section VI. At the end paper is concluded with conclusion in section VII.

II. RELATED WORK

FPGAs configuration bitstream is the final output of any VLSI CAD tool design methodology. In [1] database of mapping bitstream to their functional role is prepared using theoretical algorithm. This database is utilized to produce net list from any bitstream. The work shown in [1] is for Xilinx Virtex-II devices. Documentation on Xilinx Design Language (XDL) is scare. XDL provide access to almost all features of Xilinx devices. Various use cases and elaborated documentation provided in [2]. In [3], a method is presented to generate NCD from bitstream file.

Due to availability of official documents on internal architecture and configuration, it was possible for researchers to build development tools for FPGA. Alex and Clifford reversed engineered Lattice Semiconductor's iCE40 FPGA. This was first step towards the source tool chain for single FPGA [4]. "Bil" a reverse engineering tool presented in [5] retrieves netlist from bitstream for certain section of bitstream. VPR [6] performs packing, placement and routing. It maps technology mapped netlist to hypothetical FPGA specified by user. A low level bitstream manipulation tools BitMAT [7] and "BITMAN" [8] targeted Virtex- II FPGAs. A Java library named "abit" for direct manipulation of Atmel FPSLIC series bitstream and partial reconfiguration found in [9]. In [10] an open source tool based on library of micro-bitstream created for primitives and merged for creating larger design with simple merging operations. An open source tool set for reconfigurable computing presented in [11] is based on C++. Torc infrastructure can read, write, and manipulate EDIF and XDL netlist as well as Xilinx bitstream packets (but not configuration frame internals). Currently supported devices include all Virtex, VirtexE, Virtex2 pro, Virtex5, Virtex6, Virtex6L devices. PARBIT a tool in [12] extracts and relocates Virtex partial bitstream. The RapidSmith project [13] provide exclusive platform for implementing experimental idea, algorithms on Xilinx FPGAs.

Knowing bitstream format opens innovative approach for development in FPGA technology. Mostly, such type of work depends upon available documentation from vendor. But the vendor does open fair amount of information for end users and keeps certain information closed to avoid cloning of their devices. In this paper we have proposed a simple approach to map configuration bitstream to FPGA resources.

III. FPGA GENERIC ARCHITECTURE AND DESIGN FLOW

FPGA means field programmable gate array. Any arbitrary digital function is implemented in FPGA. It can be programmed to implement desired hardware on it. Basic building blocks of FPGAs are configuration logic blocks (CLBs), I/O blocks, switch blocks, routing resources, memory blocks, and other dedicated functional block. The Fig. 2 gives general architecture of FPGA and Spartan -3A FPGA layout as viewed in PlanAhead design tool.

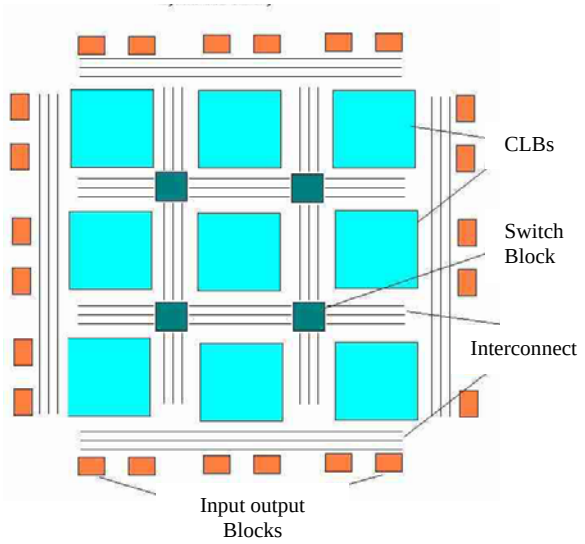


Fig. 2 (a) General architecture of FPGA

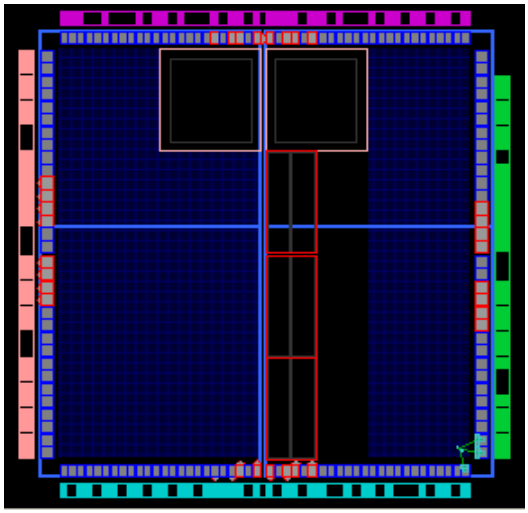


Fig. 2 (b) Spartan-3A layout in Plan Ahead

Fig. 2 FPGA architecture and PlanAhead view of Spartan-3A layout

A typical FPGA design flow consists of different stages like design entry, synthesis, mapping, place and route and bitstream generation. Fig. 3 depicts FPGA design flow. Three main stages of design are design entry and synthesis, design implementation and design verification [14]. Design entry is made either in schematic capture or in hardware design languages or with both. Synthesis is the process by which a given design is converted to logical design format (EDIF) or NGC file (by using Xilinx Synthesis Technology GUI). Design implementation converts the logical design format into Native Circuit Description (NCD) file. NCD file contains the physical

information of FPGAs. The bitstream file generated after this stage is used to program the FPGA. In the last stage designer verifies whether the circuit meets timing and functionality requirements

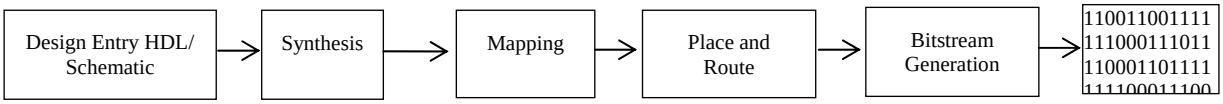


Fig. 3 Typical FPGA design flow

IV. BACKGROUND: HDL-NCD-XDL

Listing 1 shows VHDL codes for two input AND gate. DPI1 and DIP2 are the two input ports and LED1 is the output port. Fig. 4 represents RTL schematic and Fig. 5 shows schematic of AND gate generated after VHDL codes compiled on Xilinx ISE Design Suite 14.7. Device utilization summary is given in Table 1. Other relevant information is generated by the tool. Here we gathered information necessary for mapping of FPGA configuration bitstream to its resources. “andGate.ncd” file is located in project folder directory. It is Native Circuit Description file which contains how HDL codes are mapped to physical resources like LUTs, flip flops, I/Os, routing etc. on FPGA. But it is not in human readable format. Xilinx Design Language is used to convert NCD file into human readable format. A command like “xdl –ncd2xdl andGate.ncd” converts NCD to XDL file. and Gate.xdc contains human readable information. A brief introduction of XDL is given by illustrating “andGate” example.

Listing 1. VHDL code of AND gate

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity andGate is
4      Port ( DIP1 : in  STD_LOGIC;
5             DIP2 : in  STD_LOGIC;
6             LED1 : out  STD_LOGIC);
7  end andGate;
8  architecture Behavioral of andGate is
9  begin
10    LED1 <= DIP1 and DIP2;
11 end Behavioral;
```

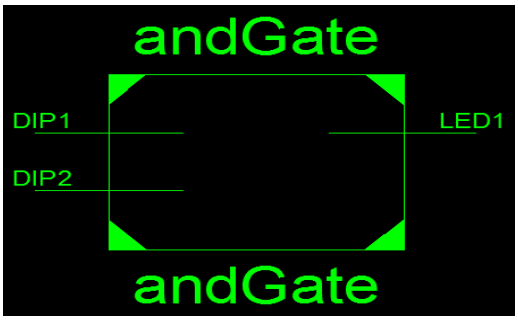


Fig.4 RTL schematic

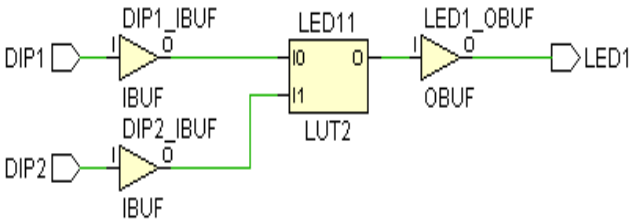


Fig. 5 Schematic of AND gate

Table 1. Device Utilization Summary

Logic Utilization	used	Available	Utilization
Number of 4 input LUTs	1	1,408	1%
Number of occupied Slices	1	704	1%
Number of Slices containing only related logic	1	1	100%
Number of Slices containing unrelated logic	0	1	0%
Total Number of 4 input LUTs	1	1,408	1%
Number of bonded IOBs	3	108	2%
Average Fanout of Non-Clock Nets	1.00		

Xilinx Design Languages (XDL)

The NCD file contains all necessary information to implement the design on FPGA. But the NCD file format is Xilinx specific and it is not possible to analyze it. However, Xilinx has provided Xilinx Design Language in Xilinx ISE Design suite. It converts NCD file into XDL file which is human readable. Less information is available on XDL documentation.

XDL File Format

An XDL file is a text file contains information about the name of the design, intended Xilinx FPGA device and cfg attribute which allows user to configure the certain aspects of the design. It also lists logical slices, nets for routing design. The keyword “inst” is used to begin the instance statement. Instance is the instance of FPGA primitive. After the instance keyword the name of the instance is mentioned. It is followed by instance type. Placement of the slice is described by the keyword “placed” or “unplaced” after the instance type. A string beginning with “cfg” keyword used to configure the LUT contents and other functionality. Instance statement for DIP1 in case of AND gate example is given below.

inst "DIP1" "IBUF", placed RIOIS_X17Y1 P78

```
cfg " DELAY_ADJ_ATTRBOX::FIXED GTSATTRBOX::#OFF IBUF_DELAY_VALUE::DLY0
          ICEINV::#OFF ICLK1INV::#OFF ICLK2INV::#OFF IDDRIN_MUX::#OFF IFD_DELAY_VALUE::DLY0
          IFF1::#OFF IFF1_INIT_ATTR::#OFF IFF1_SR_ATTR::#OFF IFF2::#OFF IFF2_INIT_ATTR::#OFF
          IFF2_SR_ATTR::#OFF IFFATTRBOX::#OFF IFFDMUX::#OFF IMUX::1 IOATTRBOX::LVCMOS25
          IREV_USED::#OFF ISR_USED::#OFF MISRATTRBOX::#OFF MISR_CLK_SELECT::#OFF
          O1INV::#OFF O1_DDRMUX::#OFF O2INV::#OFF O2_DDRMUX::#OFF OCEINV::#OFF
          ODDROUT1_MUX::#OFF ODDROUT2_MUX::#OFF OFF1::#OFF OFF1_INIT_ATTR::#OFF
          OFF1_SR_ATTR::#OFF OFF2::#OFF OFF2_INIT_ATTR::#OFF OFF2_SR_ATTR::#OFF
          OFFATTRBOX::#OFF OMUX::#OFF OREV_USED::#OFF OSR_USED::#OFF OTCLK1INV::#OFF
          OTCLK2INV::#OFF PCICE_MUX::#OFF PCIRDY_MUX::#OFF PULL::#OFF REVINV::#OFF
          SEL_MUX::0 SLEW::#OFF SRINV::#OFF T1INV::#OFF T2INV::#OFF TCEINV::#OFF
          TFF1::#OFF TFF1_INIT_ATTR::#OFF TFF1_SR_ATTR::#OFF TFF2::#OFF TFF2_INIT_ATTR::#OFF
          TFF2_SR_ATTR::#OFF TFFATTRBOX::#OFF TMUX::#OFF TREV_USED::#OFF TSMUX::#OFF
          TSR_USED::#OFF T_USED::#OFF DELAY_ADJ_BBOX:DIP1.DELAY_ADJ: INBUF:DIP1_IBUF:
          PAD:DIP1: "
  ;
```

The logic blocks are interconnected by the nets. FPGA fabric contains the static wires. The nets are used for routing purpose. A net has “outpin” as one starting point and “inpin” as endpoints (it can be multiple). The configuration in FPGAs routing structure is the connection between these nets. In Xilinx terminology these are called pips (programmable interconnect points). The inpin and outpin are connected by pip declaration. A set of pips declarations constitutes a single net. Pip is declared as “pip LOC S -> E” where pip is keyword, LOC is the location of pip, S is the starting point and E is the endpoint of the wire. Finally summary statements give statics of modules, instances and nets used in the design. Listing 2 summaries the description of XDL file format for “andGate” example compiled for XC3S50A Spartan-3A FPGA. Latter in the paper this design is named as “default” as it is compiled without any constraint.

Listing 2. XDL file from NCD file for AND gate default design example.

```
# ========================================
# XDL NCD CONVERSION MODE $Revision: 1.01$
# time: Sat Jan 27 00:55:22 2018
# ========================================
# The syntax for the design statement is:
# ========================================
design "andGate" xc3s50atq144-5 v3.2 ,
cfg "
_DESIGN_PROP::PK_NGMTIMESTAMP:1516987046"
;
# ========================================
inst "DIP1" "IBUF",placed RIOIS_X17Y1 P78
inst "DIP2" "IBUF",placed BIOIS_X16Y0 P72
inst "LED1" "IOB",placed RIOIS_X17Y1 P76
inst "LED1_OBUF" "SLICEL",placed CLB_X16Y1
SLICE_X23Y0
# ========================================
net "DIP1_IBUF" ,
  outpin "DIP1" I ,
  inpin "LED1_OBUF" G1 ,
  pip CLB_X16Y1 G1_B1 -> G1_B_PINWIRE1 ,
  pip CLB_X16Y1 OMUX_W1 -> G1_B1 ,
  pip RIOIS_X17Y1 I1_PINWIRE -> IOIS_Y1 ,
  pip RIOIS_X17Y1 IOIS_Y1 -> OMUX1 ,
  ;
net "DIP2" , cfg " _BELSIG:PAD,PAD,DIP2:DIP2",
  ;
net "DIP2_IBUF" ,
  outpin "DIP2" I ,
  inpin "LED1_OBUF" G4 ,
  pip BIOIS_X16Y0 I1_PINWIRE -> IOIS_Y1 ,
  pip BIOIS_X16Y0 IOIS_Y1 -> OMUX15 ,
  pip CLB_X16Y1 G4_B1 -> G4_B_PINWIRE1 ,
  pip CLB_X16Y1 OMUX_N15 -> G4_B1 ,
  ;
net "LED1" , cfg " _BELSIG:PAD,PAD,LED1:LED1",
  ;
net "LED1_OBUF" ,
  outpin "LED1_OBUF" Y ,
  inpin "LED1" O1 ,
  pip CLB_X16Y1 Y1 -> E2BEG4 ,
pip RIOIS_X17Y1 E2MID4 -> IOIS_G3_B0 , ;
# ====================================
# SUMMARY
# Number of Module Defs: 0
# Number of Module Insts: 0
# Number of Primitive Insts: 4
# Number of Nets: 6
```

Note: *In the above listing "cfg" keyword and its listing is not included.*

V. METHODOLOGY AND IMPLEMENTATION

Initially an AND gate was designed and implemented in VHDL hardware description language without any constraints. Latter same VHDL codes were used to implement AND gate design with different constraints so that AND gate will map to different location. The pblock is used for restricting AND gate on FPGA layout along with defined I/O pins location is mentioned in the Table 2. In each case, design is compiled on Xilinx ISE 14.7 Design Suite [15]. Ensure that generate debug is enabled in process properties dialog box. This is shown in Fig. 6. This will generate FAR address for each configuration frame.

Fig. 7 shows location of CLBs on FPGA layout selected for experimentation for default, const_1 and const_2. After compiling same AND gate design with different constraints of slices and I/O pins assignment, a bit file database is prepared. Each "andGate.bit" file is analysed separately. Configuration bitstream file is opened in Hex Editor Neo. A frame containing all zero in bitstream file is selected. After that with the help of hex editor all such frames containing zeros are found out. An observation of starting portion and ending portion of the frame is noted. The important part of bitstream is the configuration data and its addresses. For all the designs, FAR addresses containing configuration information are tabulated. Frame addresses which points to zero configuration information are neglected. This process will separate out bitstream information into four parts: starting, ending, design information and no information of design. The design information portion is pointed by FAR register addresses. PlanAhead tool is used to observe device view for mapped resources on Spartan-3A in FPGA layout. An FPGA editor [16] is used to view how resources are placed and routed.

Table 2. Floor planning of AND gate and constraints

Const. No.	Pblock Range	CLB	DIP1	DIP2	LED1
Default	NA	CLB_X16Y1	RIOSIS_X17Y1	BIOIS_X16Y0	RIOIS_X17Y1
			P78	P72	P76
Const_1	SLICE_X20Y28:SLICE_X21Y29	CLB_X15Y15	RIOSIS_X17Y15	RIOSIS_X17Y15	RIOSIS_X17Y14
			P101	P103	P104
Const_2	SLICE_X0Y30:SLICE_X1Y31	CLB_X1Y16	TOIOIB_X1Y17	TOIOIB_X1Y17	TOIOIB_X1Y17
			P141	P140	P143
Const_3	SLICE_X0Y0:SLICE_X1Y1	CLB_X1Y1	BIOIB_X1Y0	BIOIB_X1Y0	BIOIS_X2Y0 P39
			P37	P38	P39
Const_4	SLICE_X10Y20:SLICE_X11Y21	CLB_X6Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11
			P12	P13	P10
Const_5	SLICE_X2Y20:SLICE_X3Y21	CLB_X2Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11
			P12	P13	P10
Const_6	SLICE_X4Y20:SLICE_X5Y21	CLB_X3Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11
			P12	P13	P10
Const_7	SLICE_X6Y20:SLICE_X7Y21	CLB_X4Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11
			P12	P13	P10

Const. No.	Pblock Range	CLB	DIP1	DIP2	LED1
Const_8	SLICE_X8Y20:SLICE_X9Y21	CLB_X5Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11
			P12	P13	P10
Const_9	SLICE_X2Y22:SLICE_X3Y23	CLB_X2Y12	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11
			P12	P13	P10
Const_10	SLICE_X2Y26:SLICE_X3Y27	CLB_X2Y14	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11	LIOIS_PCIX0Y11
			P12	P13	P10

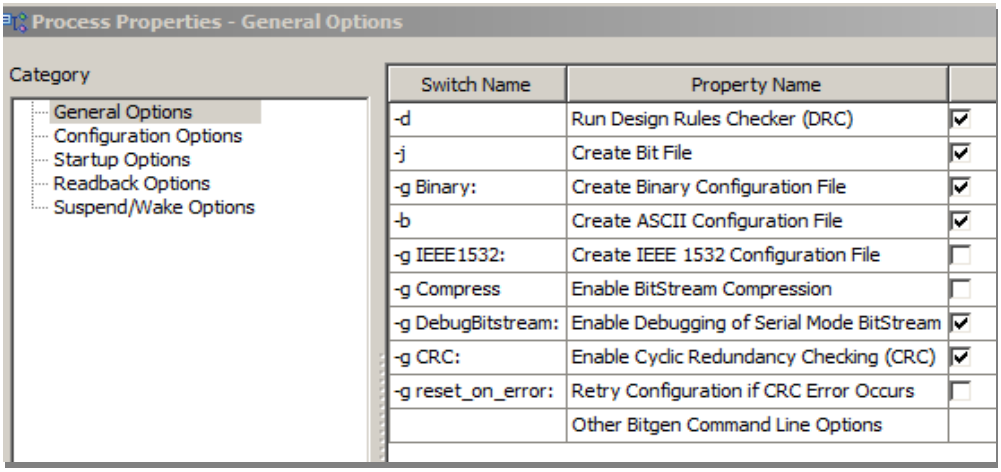
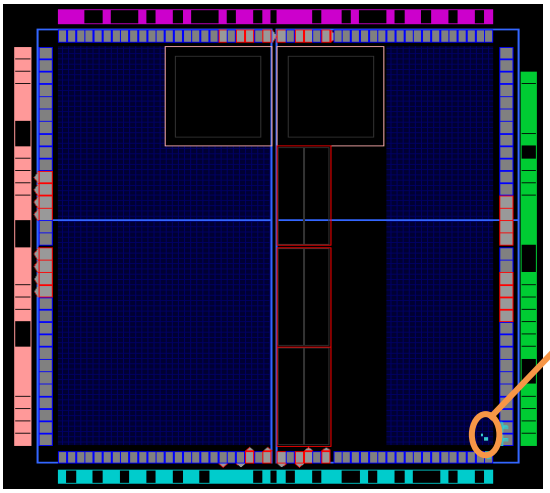
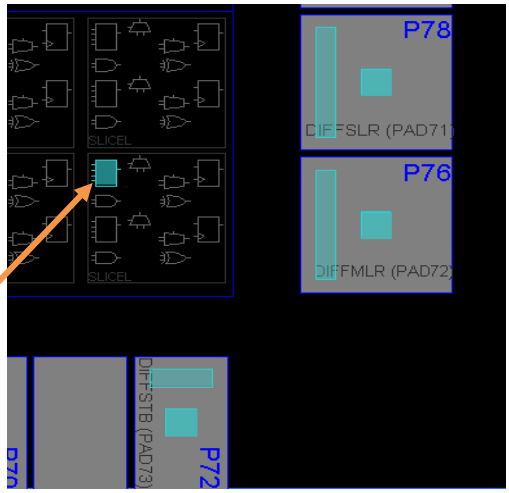


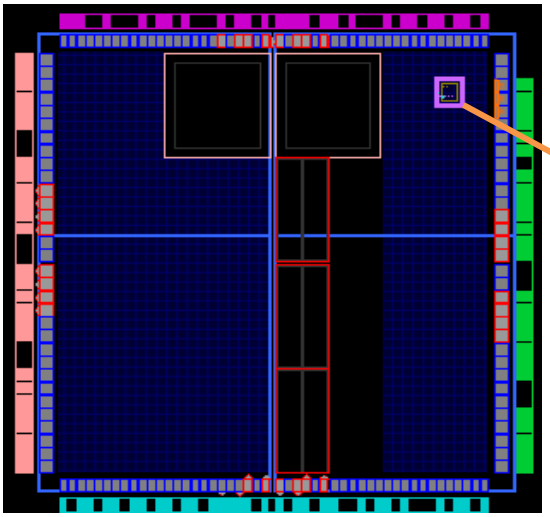
Fig. 6 Enable debugging of bitstream in process properties of generate Programming



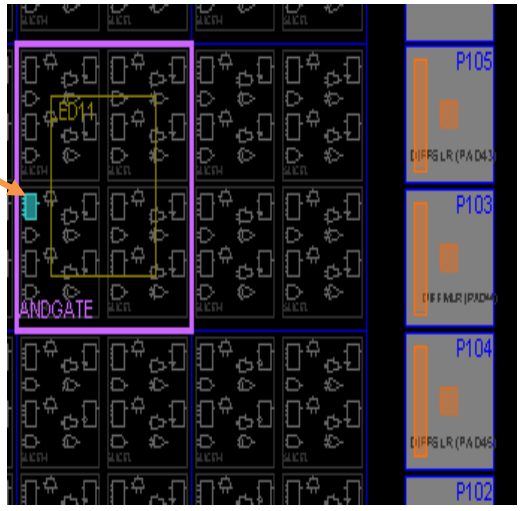
A. Default on device layout



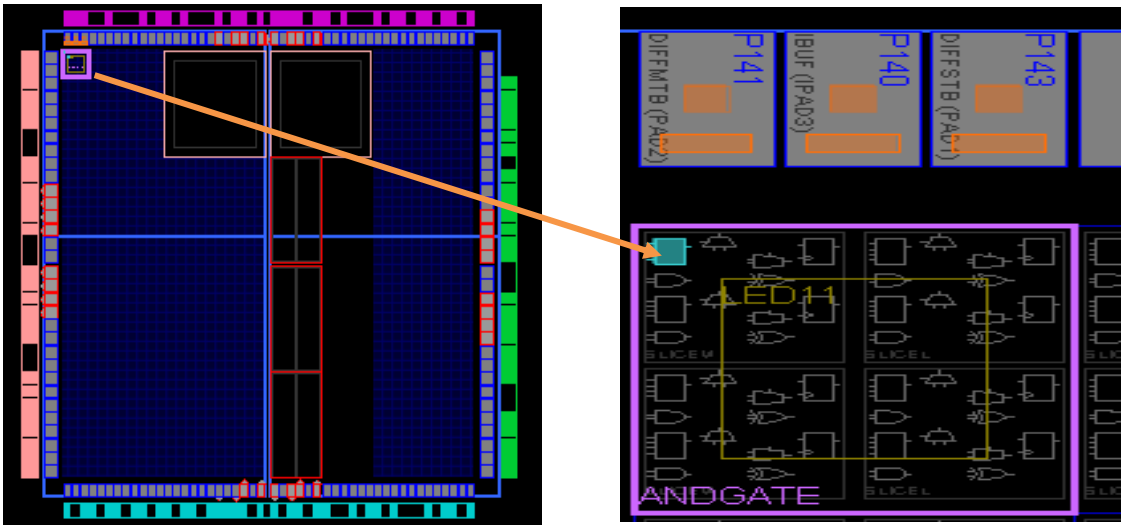
A. Default zoom view



B. Const_1 on device layout



B. Const_1 zoom view



C. Const_2 on device layout C. Const_2 zoom view

Fig. 7 Different CLB location selections with constraints

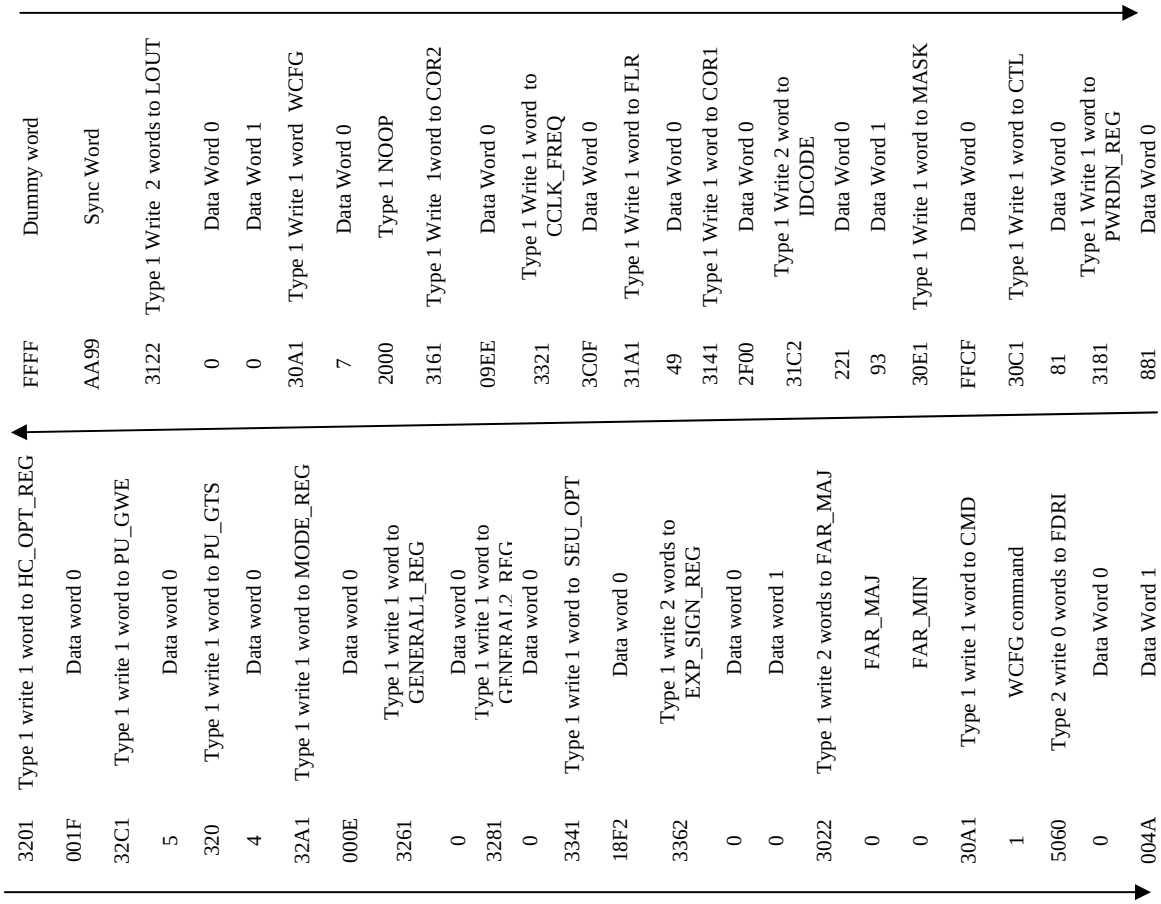
VI. RESULTS AND DISCUSSION

Analysis of Configuration Bitstream

Bitstream files consist of a start header (Fig. 8), configuration data, bitstream final command (Fig.10) and start-up sequence. The start header contains information like design name, target device, file modification time, dummy word, synchronization word, reset CRC, device ID code, etc. The packets contain configuration commands and configuration data. The small state machine in FPGA controls configuration of programmable resources on FPGA. The command register in FPGA is either read or written by configuration information in data frames in bitstream. Always action taken by state machines is as per the command registers data. The data frames are loaded with configuration data as pointed by the frame address register (FAR). Configuration data frames can be, single frame, set of frames or full configuration space. During continuous writing of configuration data into FPGA, the FAR is automatically incremented after every FDRI write [17]. In this case frame length register (FLR) contains the length of data frame. It is written before the initiation of command to load block of data. Finally START, CRC checksum DESYNCH command are executed towards the end of bitstream file as shown in Figure 10 and Figure 11. Figure 8 and Fig. 10 shows starting and ending part of the bitstream file of AND gate example. The interpretation of each word in the frame is explained in Fig. 9 and Fig.11.

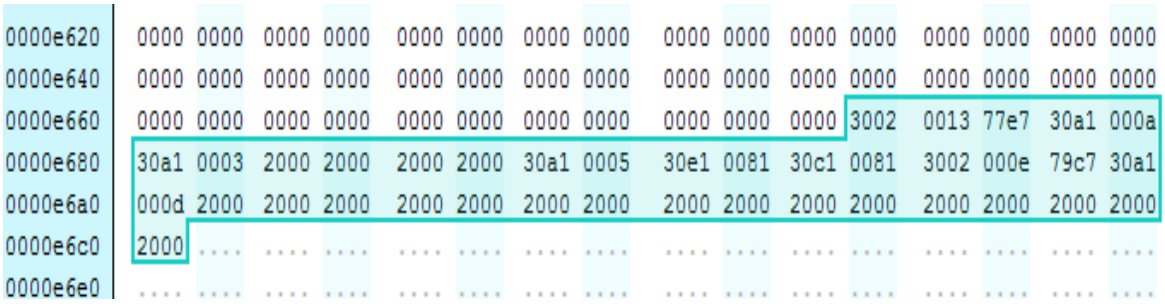
00000000	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00000000	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff
00000020	aa99	3122	0000	0000	30a1	0007	2000	3161	09ee	3321	3c0f	31a1	0049	3141	2f00	31c2
00000040	0221	0093	30e1	ffcf	30c1	0081	3181	0881	3201	001f	32c1	0005	32e1	0004	32a1	000e
00000060	3261	0000	3281	0000	3341	18f2	3362	0000	0000	3022	0000	0000	30a1	0001	5060	0000
00000080	004a	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
000000a0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
000000c0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Fig 8 Start portion of bitstream file (AND Gate)



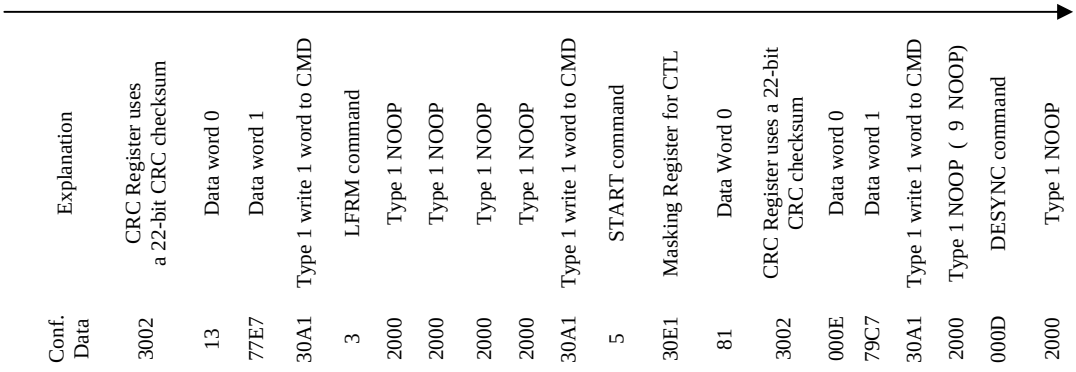
3201	Type 1 write 1 word to HC_OPT_REG	FFFF	Dummy word
001F	Data word 0	AA99	Sync Word
32C1	Type 1 write 1 word to PU_GWE	3122	Type 1 Write 2 words to LOUT
5	Data word 0	0	Data Word 0
320	Type 1 write 1 word to PU_GTS	0	Data Word 1
4	Data word 0	30A1	Type 1 Write 1 word WCFG
32A1	Type 1 write 1 word to MODE_REG	7	Data Word 0
000E	Data word 0	2000	Type 1 NOOP
3261	Type 1 write 1 word to GENERAL1_REG	3161	Type 1 Write 1word to COR2
0	Data word 0	09EE	Data Word 0
3281	Type 1 write 1 word to GENERAL2_REG	3321	Type 1 Write 1 word to CCLK_FREQ
0	Data word 0	3C0F	Data Word 0
3341	Type 1 write 1 word to SEU_OPT	31A1	Type 1 Write 1 word to FLR
18F2	Data word 0	49	Data Word 0
3362	Type 1 write 2 words to EXP_SIGN_REG	3141	Type 1 Write 1 word to COR1
0	Data word 0	2F00	Data Word 0
0	Data word 1	31C2	Type 1 Write 2 word to IDCODE
3022	Type 1 write 2 words to FAR_MAJ	221	Data Word 0
0	FAR_MAJ	93	Data Word 1
0	FAR_MIN	30E1	Type 1 Write 1 word to MASK
30A1	Type 1 write 1 word to CMD	FFCF	Data Word 0
1	WCFG command	30C1	Type 1 Write 1 word to CTL
5060	Type 2 write 0 words to FDRI	81	Data Word 0
0	Data Word 0	3181	Type 1 Write 1 word to PWRDN_REG
004A	Data Word 1	881	Data Word 0

Fig.9 Decoding of start portion of bitstream file



0000e620	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000e640	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000e660	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	3002	0013	77e7	30a1	000a
0000e680	30a1	0003	2000	2000	2000	2000	30a1	0005	30e1	0081	30c1	0081	3002	000e	79c7	30a1
0000e6a0	000d	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000
0000e6c0	2000	....	....	....	....	....	....	....	....	....	....	....	....	....	....	....
0000e6e0	....	....	....	....	....	....	....	....	....	....	....	....	....	....	....	....

Fig.10 End portion of bitstream file



Conf. Data	Explanation
3002	CRC Register uses a 22-bit CRC checksum
13	Data word 0
77E7	Data word 1
30A1	Type 1 write 1 word to CMD
3	LFRM command
2000	Type 1 NOOP
2000	Type 1 NOOP
2000	Type 1 NOOP
2000	Type 1 NOOP
30A1	Type 1 write 1 word to CMD
5	START command
30E1	Masking Register for CTL
81	Data Word 0
3002	CRC Register uses a 22-bit CRC checksum
000E	Data word 0
79C7	Data word 1
30A1	Type 1 write 1 word to CMD
2000	Type 1 NOOP (9 NOOP)
000D	DESYNC command
2000	Type 1 NOOP

Fig. 11 Decoding of end portion of bitstream file

Decoding of Frame Address Register (FAR)

Frame addresses decoded for each of the design is shown in Table 3. All designs belongs to block type 0 containing CLBs, IOBs information. Major address selects major column and minor address selects memory-cell address line within a major column. Column labelled with "M" indicates major address whereas "m" indicates minor address of frames. The addressing scheme gives the idea of the columns where the design lies vertically. Frame addresses and total number of frames required for implementation of design varies according to design type.

Table 3. Decoding of frame address register

	default		Const_1		Const_2		Const_3		Const_4		Const_5		Const_6		Const_7		Const_8		Const_9		Const_10	
SN	M	m	M	m	M	m	M	m	M	m	M	M	M	m	M	m	M	m	M	m	M	m
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	14	3	13	0	3	0	3	3	2	0	2	0	2	0	2	0	2	0	2	0	2	0
4	14	4	13	1	3	1	3	5	2	1	2	1	2	1	2	1	2	1	2	1	2	1
5	14	5	13	7	3	4	3	6	2	10	2	9	2	10	2	9	2	9	2	10	2	6
6	14	6	13	9	3	5	3	7	2	14	2	13	2	13	2	11	2	17	2	14	2	7
7	14	7	13	11	3	6	3	8	2	15	2	15	2	14	2	12	2	18	2	15	2	9
8	14	8	14	14	3	7	3	9	2	17	3	13	2	15	3	13	4	13	2	17	3	13
9	14	10	14	15	3	8	3	10	2	18	3	14	4	13	3	14	4	14	2	18	4	0
10	14	11	15	0	3	9	3	11	8	0	4	0	4	15	5	13	6	13	8	0	4	1
11	14	12	15	1	3	10	3	12	8	1	4	1	5	0	5	14	6	14	8	1	4	6
12	14	13	15	9	3	11	3	13	8	8	4	6	5	1	6	0	7	0	8	8	4	8
13	15	0	15	14	3	12	3	14	8	9	4	8	5	8	6	1	7	1	8	9	4	9
14	15	1	16	0	3	13	4	0	8	11	4	9	5	9	6	6	7	6	8	11	4	11
15	15	7	16	1	3	15	4	1	8	12	4	11	5	11	6	9	7	9	8	12	4	12
16	15	11			3	16	4	8	8	13	4	12	5	12	6	12	7	10	8	13		
17	15	12			3	17	4	10	8	17			5	17			7	11	8	17		
18	16	0			3	18	4	11									7	12				
19	16	1			4	18	4	12									8	14				
20							4	14														
21							4	15														

Note: *"M" indicates Major Address and "m" indicates minor address.*

Matching and Mismatching Patterns in Configuration Bitstream

Extended Spartan-3A device, XC3S50A has 367 device frames. Result of match and mismatch pattern in configuration bitstream shown in Fig. 12.

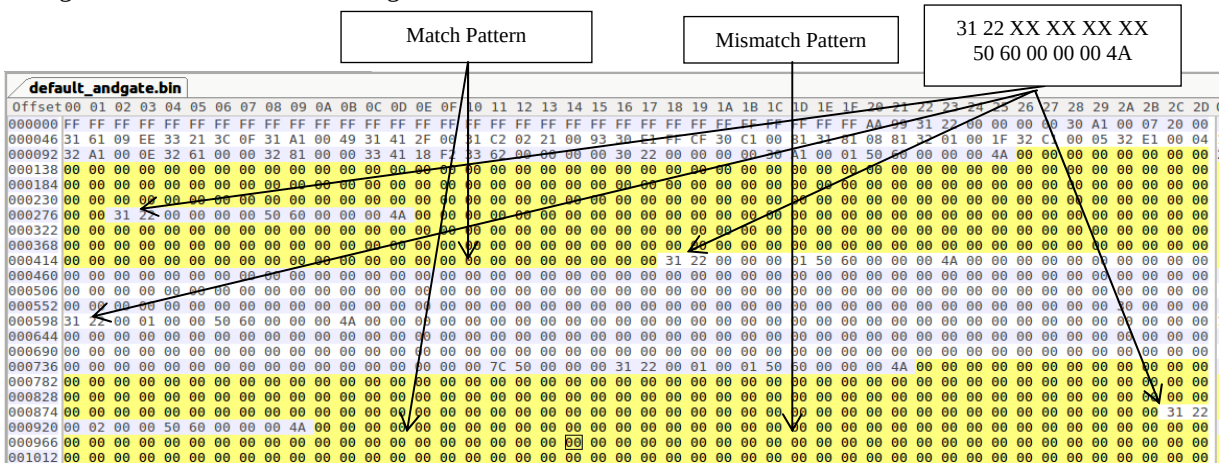


Fig.12 Match and mismatch patterns in configuration bitstream

Table 4 indicated the count of match and mismatch patterns as observed in each bitstream file. Also, it gives total count of combined pattern. The frame address for which mismatch pattern were observed are mentioned in Table 3. The mismatch pattern in bitstream file contains information about the implemented designs.

Table 4. Match and Mismatch Patterns in Configuration Bitstream

Constraint No	Pattern Match Found	Pattern Mismatch	Total Pattern
Default	349	19	368
Const_1	353	15	368
Const_2	349	19	368
Const_3	347	21	368
Const_4	351	17	368
Const_5	352	16	368
Const_6	351	17	368
Const_7	351	16	368
Const_8	349	19	368
Const_9	353	15	368
Const_10	353	15	368

Though the HDL codes are same for each design, the location of each design is different on FPGA tiles. For each of the design the mismatch patterns count is different. This is because the routing structure is different for each design. Routing of designs viewed in FPGA editor. Fig. 13 (a) depicts routing structure for default design. Three switch boxes are used for routing. Default design is located by placement and routing tool at bottom right corner of the layout. Fig. 13(b) depicts routing structure for design with const_4. This design resides in middle portion of the layout.

For the default design LUT belongs to SLICE_X23Y0 of CLB_X16Y1. This is also included in statement, “inst "LED1_OBUF" "SLICEL", placed CLB_X16Y1 SLICE_X23Y0” in XDL file generated from NCD file. Also the instance statement for instances DIP1, DIP2 and LED1 are : inst "DIP1" "IBUF", placed RIOIS_X17Y1 P78 , inst "DIP2" "IBUF", placed BIOIS_X16Y0 P72, inst "LED1" "IOB", placed RIOIS_X17Y1 P76 respectively. The instance view shown in Fig. 14

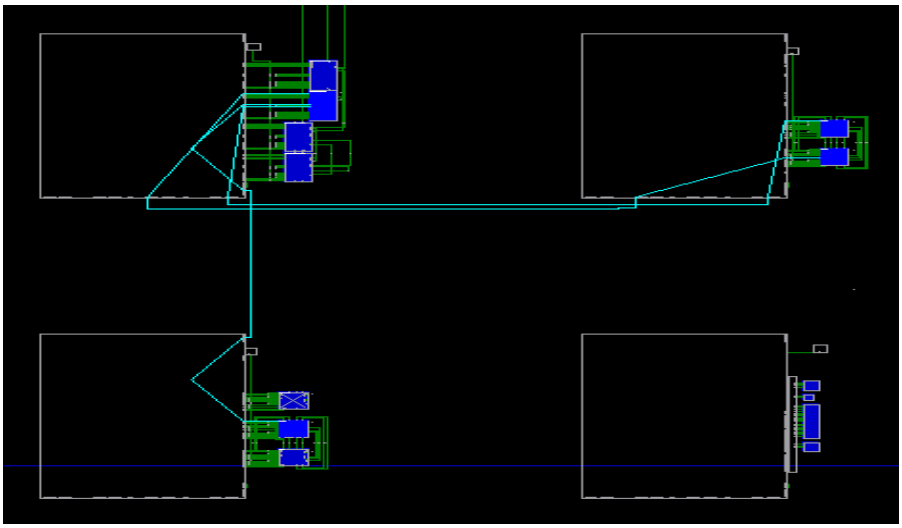


Fig. 13 (a) Default Design

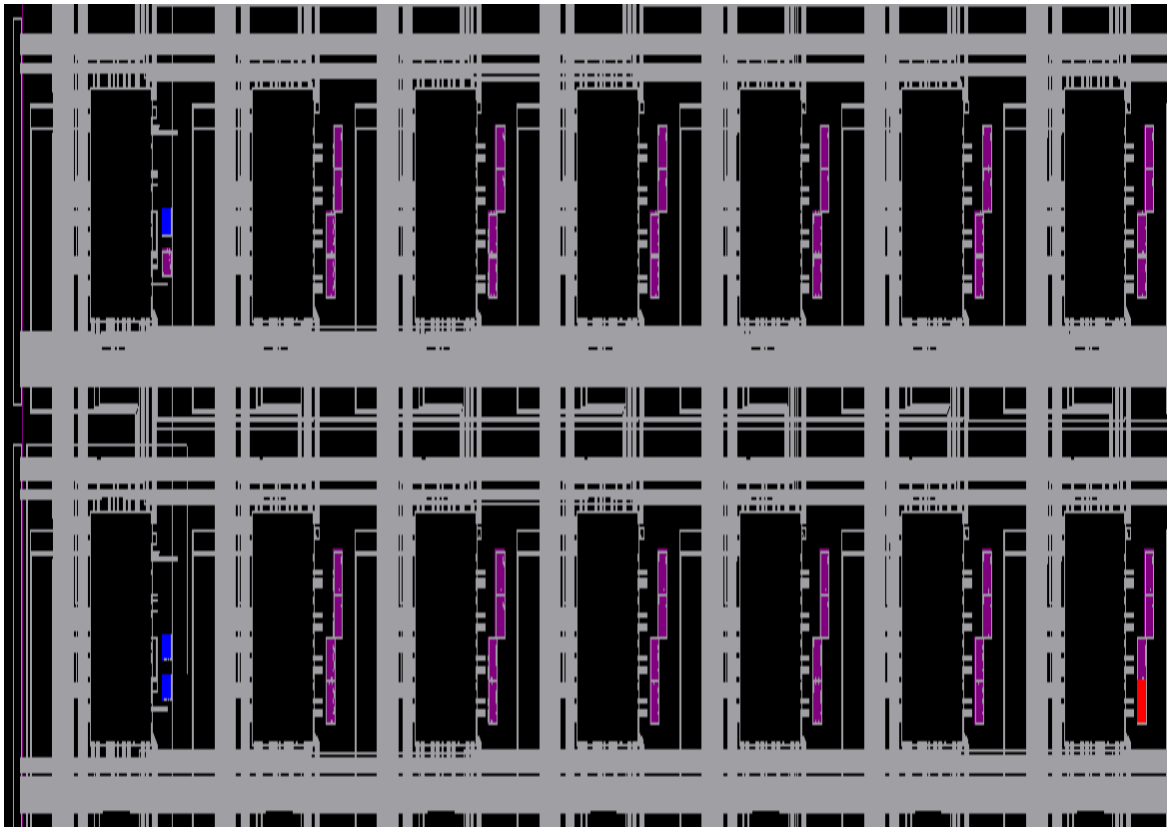


Fig. 13 (b) Design with Const_4

Fig. 13 The routing structure

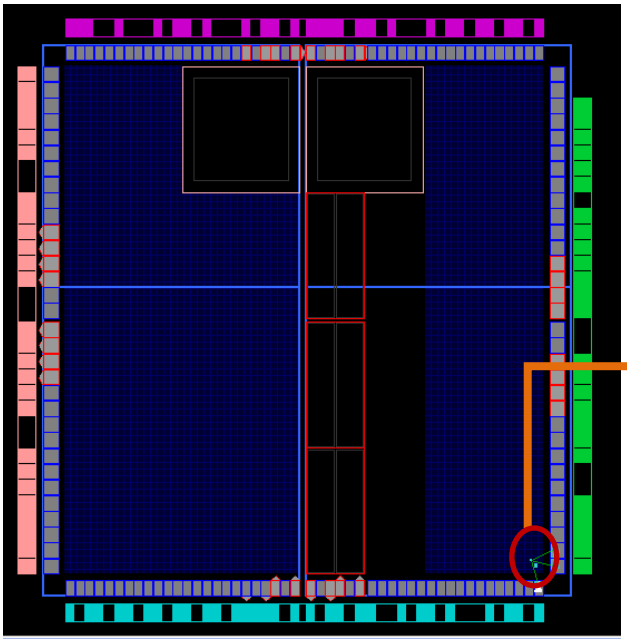


Fig. 14 (a) Location on FPGA layout

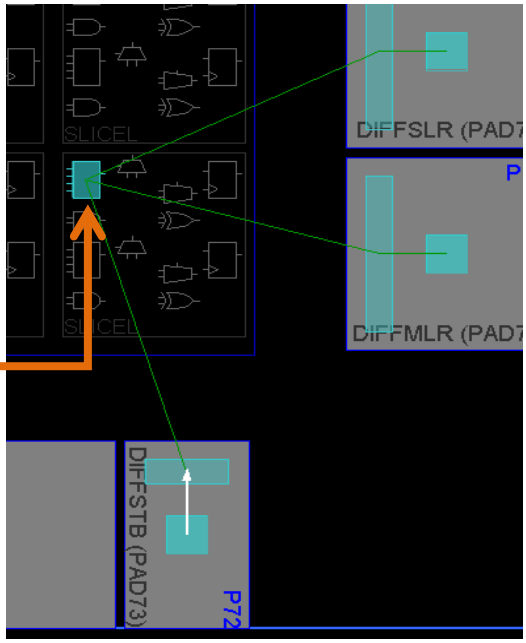


Fig. 14 (b) Zoom view of design

Fig. 14 Location of default design on FPGA

The detail view of SLICE_X23Y0 as viewed in FPGA editor is depicted Fig. 15. Likewise view for other instances generated and routing of nets related with XDL file obtained in listing 2.

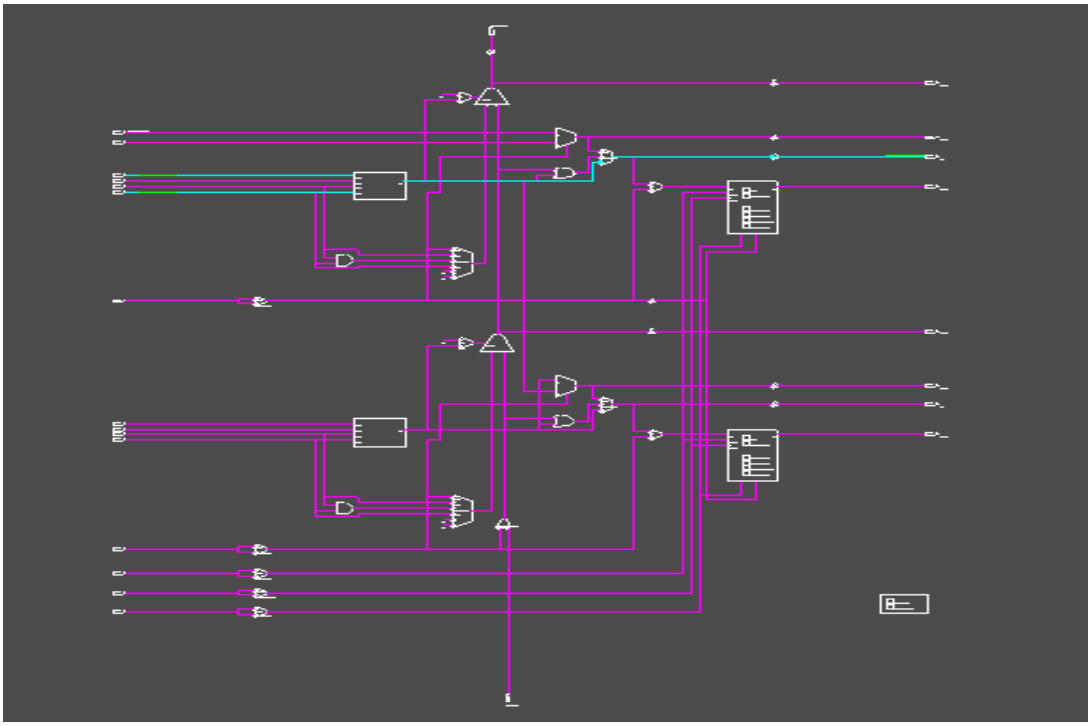


Fig. 15 LUT mapped on slice

VII. CONCLUSION

A simplified straight approach to map FPGAs resources to its HDL description is demonstrated in this paper. We found that information in XDL file derived from NCD file is in human readable form and helps to develop the understanding of FPGA design layout. Xilinx ISE Design suite and PlanAhead tool played vital role in analysis. Though we are able to locate FPGA design location vertically with addresses information in FAR_MAJ register, configuration information pointed by FAR_MIN address register is yet not decoded for Spartan-3A FPGA by our approach. We observed that as we move any design from bottom-left position to top-left position to top-right position to top-bottom position, the configuration information in bitstream goes away and away form starting position of bitstream file. The addition of “Match Pattern” and the “Mismatch Pattern” in configuration bitstream file equals 368 for all design under study.

REFERENCES

[1] J.-B. Note and Eric Rannaud, “From the bitstream to the netlist", in Proceedings of the 2008 ACM/SIGDA 16th Annual International Symposium on Field-Programmable Gate Arrays, FPGA 2008 (Monterey, California), pp. 264-264, February 24-26, 2008,
[2] C. Beckhoff, D. Koch, and J. Torresen, “The xilinx design language (xdl): Tutorial and use cases," in Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 6th International Workshop, pp. 1-8, 2011.
[3] Z. Ding, Q.Wu, Y. Zhang, and L. Zhu, “Deriving an NCD file from an FPGA bitstream: Methodology, architecture and evaluation”, Microprocessors and Microsystems, vol 37, pp. 299-312, May 2013,
[4] Joushua Vasquez, “Reverse Engineering Lattice’s ICE40 FPGA Bitstream,” https://hackaday.com/2015/03/29/reverse-engineering-lattices-ice40-fpga-bitstream/
[5] F. Benz, A. Seffrin, and S. Huss, “Bil: A tool-chain for bitstream reverse-engineering,", in Field Programmable Logic and Applications (FPL), 22nd International Conference, 2012, pp. 735-738.
[6] Vaughn Betz and Jonathan Rose, “VPR: A new packing, placement and routing tool for FPGA research,” International Workshop on Field Programmable Logic and Applications, 1997.
[7] Casey J Morford, “BitMaT - Bitstream Manipulation Tool for Xilinx FPGAs,” Master thesis. December 15th, 2005 Bradley Department of Electrical and Computer Engineering Blacksburg, Virginia.
[8] Khoa Dang Pham, Edson Horta and Dirk Koch, “BITMAN: A Tool and API for FPGA Bitstream Manipulations,” Design, Automation & Test in Europe Conference & Exhibition (DATE), 27-31 March 2017.
[9] Adam Megacz, “A Library and Platform for FPGA Bitstream Manipulation,” 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007), 23-25 April 2006, PP 45-54.
[10] R. Soni, N. Steiner, and M. French, “Open-source bitstream generation," in Field-Programmable Custom Computing Machines”(FCCM), IEEE 21st Annual International Symposium , 2013, pp. 105-112.
[11] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French, “Torc: Towards an open-source tool flow", 19th International Symposium on Field-Programmable Gate Arrays (FPGA 2011), February 27-March 1, 2011.

[12] Edson L. Horta, John W. Lockwood, "PARBIT: A Tool to transform bitfiles to implement partial reconfiguration of Field Programmable Gate Arrays (FPGAs)," Report Number: WUCS-01-13, Computer Science and Engineering, Washington University in St. Louis, 2001.
[13] C. Lavin, M. Padilla, P. Lundrigan, B. Nelson, and B. Hutchings, "Rapid prototyping tools for FPGA designs: RapidSmith," International Conference on Field-Programmable Technology (FPT'10), December 2010.
[14] Xilinx, "Command Line Tool User Guide, UG628 (v 14.7), October 2, 2013," https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/devref.pdf
[15] Xilinx, "ISE Design Suite 14: Release Notes, Installation, and Licensing.UG631 (v14.7) October 2, 2013," https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/irn.pdf
[16] Xilinx, "FPGA Editor Guide – 2.1i," http://ebook.pldworld.com/_semiconductors/Xilinx/Foundation/ISE/3.3i/Documentation/fpedit.pdf
[17] Xilinx, "Spartan-3 Generation Configuration User Guide (UG332 (v1.7) January 27, 2015 2.1)," https://www.xilinx.com/support/documentation/user_guides/ug332.pdf.