PARBIT: A Tool to Transform Bitfiles to Implement Partial Reconfiguration of Field Programmable Gate Arrays (FPGAs)

Edson L. Horta, John W. Lockwood

WUCS-01-13

July 06, 2001

Department of Computer Science
Applied Research Lab
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130

Abstract

Field Programmable Gate Arrays (FPGAs) can be partially reconfigured to implement Dynamically loadable Hardware Plugin (DHP) modules. A tool called PARBIT has been developed that transforms FPGA configuration bitfiles to enable DHP modules. With this tool it is possible to define a partial reconfigurable area inside the FPGA and download it into a specified region of the FPGA device. One ore more DHPs, with different sizes can be implemented using PARBIT.

Supported by: NSF ANI-0096052, Xilinx Inc. and CNPq (Brazil)

Contents

1 Introduction **6**

2 VIRTEX FPGA **6**
2.1 Architecture . . . 6
2.2 Addressing . . . 7
2.3 Configuration . . . 10
2.3.1 Command Register (CMD) . . . 11
2.3.2 Configuration Option Register (COR) . . . 12
2.3.3 Control Register (CTL) . . . 12
2.3.4 Cyclic Redundancy Check Register (CRC) . . . 13
2.3.5 Frame Address Register (FAR) . . . 13
2.3.6 Frame Data Input Register (FDRI) . . . 13
2.3.7 Frame Length Register (FLR) . . . 13
2.3.8 Mask Register (MASK) . . . 14
2.4 Original Bitstream Format . . . 14

3 PARBIT **15**
3.1 Organization . . . 15
3.1.1 User Parameters . . . 18
3.1.2 Write Header . . . 18
3.1.3 Frame calculation . . . 18
3.1.4 Copy Slices(Slice Mode) . . . 19
3.1.5 Copy Block (Block Mode) . . . 21
3.1.6 Write Trailer . . . 23
3.1.7 Write CRC . . . 23
3.2 Partial Bitstream Format . . . 23
3.2.1 With Shutdown . . . 24
3.2.2 Without Shutdown . . . 24
3.3 Design Flow . . . 25
3.3.1 Slice Mode . . . 25
3.3.2 Block Mode . . . 25
3.4 Operation . . . 27

4 Using PARBIT (Block Mode) to implement the DHPs in the FPX Board **28**
4.1 Single-Size DHP (XCV1000E) . . . 28
4.1.1 Original Bitstream File . . . 28
4.1.2 Target Bitstream File . . . 29
4.1.3 Options Files . . . 30
4.1.4 Running PARBIT . . . 30
4.2 Double-Size DHP (XCV1000E) . . . 31
4.2.1 Original Bitstream File . . . 31
4.2.2 Target Bitstream File . . . 31
4.2.3 Options Files . . . 32
4.2.4 Running PARBIT . . . 32
4.3 Single-Size DHP (XCV2000E) . . . 33
4.3.1 Original Bitstream File . . . 33

4.3.2 Target Bitstream File 33
4.3.3 Options Files 34
4.3.4 Running PARBIT 35

5 Conclusions **35**

List of Figures

1 Virtex Architecture 7
2 Virtex Configuration Columns - XCV50 8
3 XCV50 Addressing Scheme 9
4 XCV1000E Addressing Scheme 9
5 Virtex Configuration Frames - CLB COLUMN 12 (XCV50) 10
6 Original Bitstream 15
7 PARBIT Flow Chart 17
8 Copy Slice Flow Chart 20
9 Copy Block Flow Chart 22
10 Write CRC Flow Chart 23
11 Partial Bitstream Example - With Shutdown 24
12 Partial Bitstream Example - Without Shutdown 24
13 Slice Mode - Original Bitstream Device 25
14 Block Mode - Original Bitstream Device 26
15 Block Mode - Target Device 26
16 XCV1000E - Single-Size DHP - Original Bitstream 29
17 XCV1000E - Single-Size DHP - Target Bitstream 29
18 XCV1000E - Double-Size DHP- Original Bitstream 31
19 XCV1000E - Double-Size DHP - Target Bitstream 32
20 XCV2000E - Single-Size DHP - Original Bitstream 33
21 XCV2000E - Single-Size DHP - Target Bitstream 34

List of Tables

1 Major Address Schemes 8
2 Number of Configuration Frames 10
3 Writing Frames - XCV50 11
4 CMD Register 11
5 Commands for CMD Register 12
6 COR Register 12
7 Options for COR Register 12
8 CTL Register 13
9 CRC Register 13
10 FAR Register 13
11 FDRI Register 13
12 FLR Register 14
13 MASK Register 14

1 Introduction

Field Programmable Gate Arrays (FPGAs) enable hardware circuits to be reconfigured an unlimited number of times. The implementation of a system that uses reconfigurability can be done in two ways: Compile-Time and Run-Time Reconfiguration [1]. For Compile-Time Reconfiguration (CTR) the FPGA does not change configuration during the application lifetime. Each application has specific functions that are loaded when the FPGA is started. Some examples of CTR systems are SPLASH [2] and PAM [3]. For Run-Time Reconfiguration (RTR), the FPGA changes configuration while it is operating. RTR can be total (all the device is reprogrammed) or partial (only part of the device is reprogrammed). Existing platforms have focused on reconfiguration of entire FPGA devices [4] [5] [6]. Some recent work has considered partial reconfiguration [7] [8].

Partial reconfiguration is a difficult task, especially in systems that require both partial reprogramming and run-time reconfiguration. In order to partially reconfigure a FPGA, it is necessary to isolate an specific area inside the FPGA and download the configuration bits related to that area. A tool called PARBIT (PARtial BItfile Transformer) has been developed to easily transform and restructure bitfiles to implement dynamically loadable hardware modules.

To restructure the configuration bitfile, the tool utilizes the original bitfile, a target bitfile (when needed) and parameters given by the user. These parameters include the block coordinates of the logic implemented on a source FPGA, the coordinates of the area for a partially programmed target FPGA, and the programming options.

This report reviews current FPGA technologies that support partial reconfiguration and presents a tool which transforms bitfiles to implement partial reconfiguration of an FPGA. Section 2 describes the architecture and configuration issues of the Xilinx FPGAs used in the Field Programmable Port Extender (FPX) [9], a module card of the Washington University Gigabit Switch (WUGS) [10]. Section 3 explains how the tool works and how to run PARBIT in its two operation modes: slice mode and block mode. The final section describes three examples of implementation of Dynamic Hardware Plugins (DHPs) [11] which are RTR modules on the FPX that implement user-defined functionality.

2 VIRTEX FPGA

The Xilinx VIRTEX [12] family of FPGA combines the features of partial reconfiguration with high density. A single Virtex device can hold more than 3 million system gates and permits a partial reconfiguration of frames, which are a fraction of the logic found on a column of the FPGA. The following sections describe in more detail the architecture of VIRTEX and how to address its configuration memory, to partially reconfigure the device.

2.1 Architecture

The VIRTEX architecture can be seen in Figure 1.

Each device has the following elements:

- CLBs (Configurable Logic Blocks);
- IOBs (Input Output Blocks);
- Block RAMs;
- Clock resources;

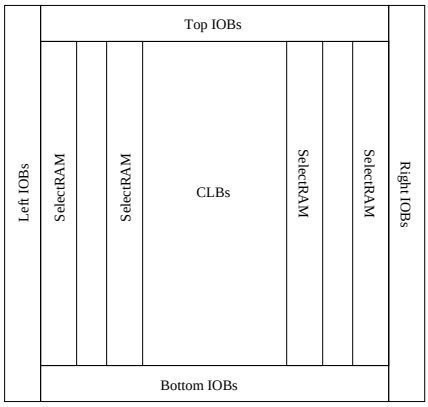

Figure 1: Virtex Architecture

- Programmable routing;
- Configuration circuitry.

To configure each resource on the Virtex, a series of bits, divided into fields of commands and data, are loaded into the device. The configuration file is called bitstream and it can be loaded into the device through three different ways: Master/Slave Serial, SelectMAP [13], and Boundary Scan [14]. The SelectMAP is an 8-bit parallel interface and the others are one-bit serial interface. One can write the configuration memory through one of these three interfaces, but only SelectMAP and Boundary Scan allows reading this memory.

To program each resource on the FPGA, the device is divided into columns. Each column corresponds to a vertical slice of the chip. There are five types of columns:

- Center: controls the global clock pins. There is one Center column per device;
- CLB: controls all the CLBs in that column, plus the two IOBs above and bellow them. The number of CLBs columns depends on the chip gate density;
- IOB: controls the configurations of all the IOBs on the right and left sides of the chip. There are two IOB columns per device;
- Block SelectRAM Interconnect: controls the connections inside each block RAM. The number of columns depends on the chip density;
- Block SelectRAM Content: controls the contents of each block RAM. The number of columns depends on the chip density.

Figure 2 shows the configuration columns for the XCV50 FPGA.

2.2 Addressing

The configuration memory is divided in two block types: RAM and CLB. The first one contains only the block SelectRAM content columns. The CLB type contains the Center, CLB, IOB and Block RAM interconnect columns. Each one of these blocks are divided in major and minor addresses. The major address (MJA) is related to the configuration column position in the memory and the minor address (MNA) is related to the frame (see next section) position inside a column .

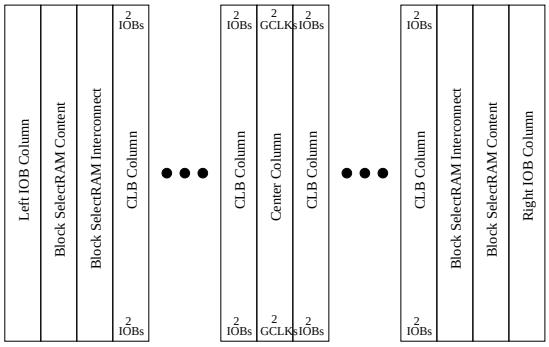

Figure 2: Virtex Configuration Columns - XCV50

Column Type	Block Type	Virtex	Virtex-E	Virtex-EM
First MJA	CLB	0	0	0
	RAM	0	1	1
MJA Order	CLB	1: Center 2: CLB 3: IOB 4: BRAM Interconnect	1: Center 2: CLB / BRAM Interconnect 3: IOB	1: Center 2: CLB 3: IOB 4: BRAM Interconnect
	RAM	BRAM Content	BRAM Content	BRAM Content

Table 1: Major Address Schemes

For the minor address and block types, the numbering schemes are the same, for all the Virtex devices. The major address numbering scheme is different for each family and it is shown on Table 1.

The Virtex and Virtex-EM families have the same numbering scheme for the CLB space:

1. Begins with zero (for the Center column);
2. Begins with 2 on the left side (first column after Center column), continues only with even numbers, until the last CLB on the far left of the device;
3. Continues with the left IOB column;
4. Continues with the left BRAM Interconnect column;
5. Begins with 1 on the right side (first column after Center column), continues only with odd numbers, until the last CLB on the far right of the device;
6. Continues with the right IOB column;
7. Ends with the right BRAM Interconnect column;

The left RAM address space is 0 for Virtex and 1 for Virtex-EM. The right RAM address space is 1 for Virtex and 2 for Virtex-EM.

Figure 3 shows the numbering scheme for the XCV50 FPGA.

The Virtex-E family has a different scheme, because there are some Block RAMs interspersed betweem the CLB columns:

1. Begins with zero (for the Center column);

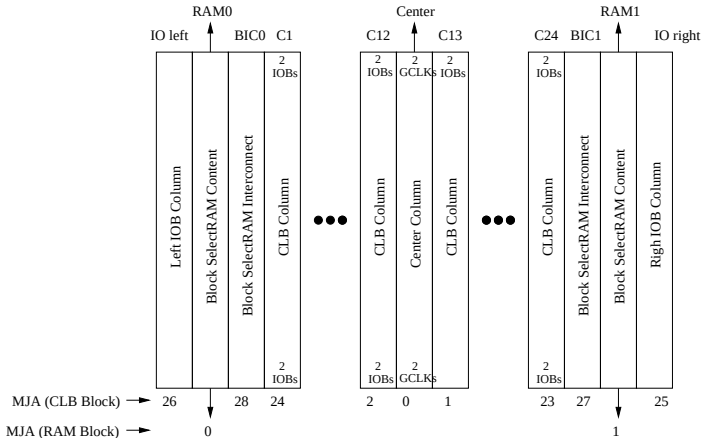

Figure 3: XCV50 Addressing Scheme

2. Begins with 2 on the left side (first Column after Center column), continues only with even numbers, until the last CLB on the far left of the device, including the left BRAM Interconnect columns between the CLB columns;

3. Begins with the left IOB column;

4. Begins with 1 on the right side(first column after Center column), continues only with odd numbers, until the last CLB on the far right of the device, including the right BRAM Interconnect columns between the CLB columns;

5. Ends with the right IOB column;

The Virtex XCV1000E numbering scheme is shown in Figure 4.

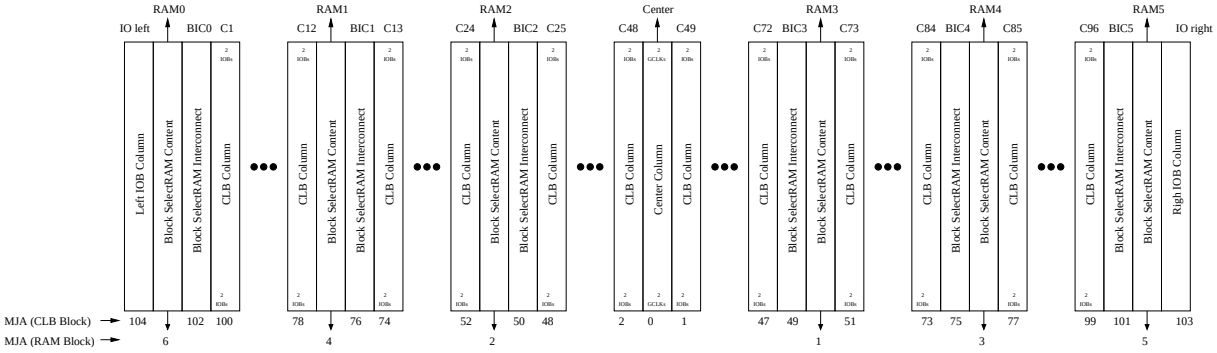

Figure 4: XCV1000E Addressing Scheme

2.3 Configuration

Each one of the configuration columns are divided in smallest slices, called frames. A frame is the smallest part of the configuration memory that can be written or read, and it is one-bit wide. The number of frames in each configuration column is fixed (see Table 2).

Column Type	# of Frames	# per Device
Center	8	1
CLB	48	# of CLB columns
IOB	54	2
Block SelectRAM Interconnect	27	# of Block SelectRAM columns
Block SelectRAM Content	64	# of Block SelectRAM columns

Table 2: Number of Configuration Frames

The number of bits in a frame is proportional to the device density. The frame is divided according to the column type. For the CLB columns, for example, it is divided in 18-bit quantities: the first group controls the Top 2 IOBs, the next ones control each CLB row and the last group controls the Bottom 2 IOBs.

For example, for the XCV50 (16 rows and 12 words/frame), there will be 18+16x18+18= 324 valid bits and 60 zeroes at the end. Figure 5 shows the configuration columns for the XCV50 FPGA.

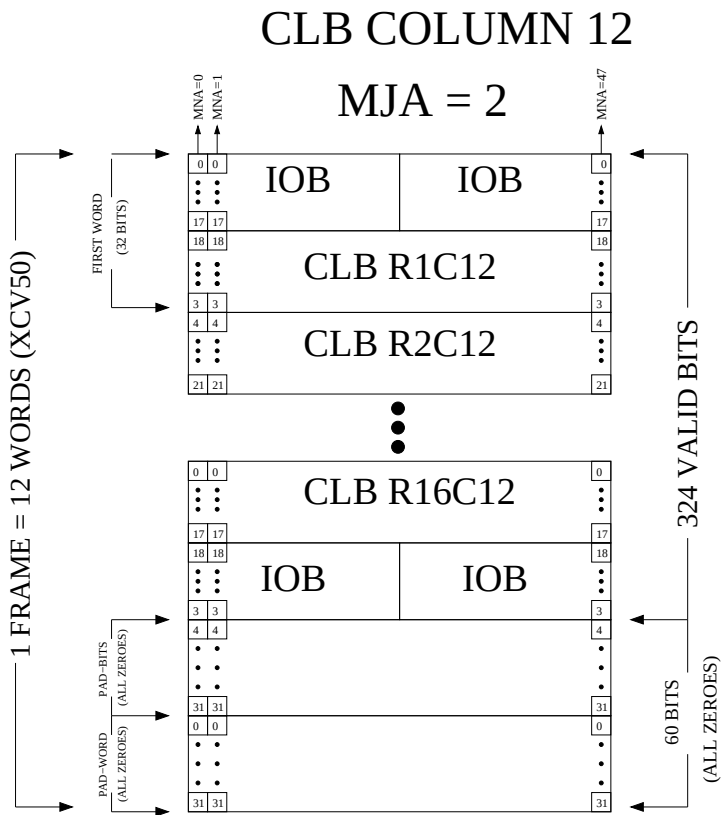

Figure 5: Virtex Configuration Frames - CLB COLUMN 12 (XCV50)

To read or write the frames into the configuration memory, the frames are grouped in 32-bit words. For each device, the number of 32-bit words in a frame is a constant value. If the number of valid bits is less than

the frame length, it is filled with zeroes on the right side. There is also a pad frame that must be considered, when reading or writing the configuration memory. This pad frame is a complete frame with all the words equal to zero.

When writing, the pad frame must be the last one to be written, as shown in Table 3.

Data Frame 0 (15 words)	Pad Word
...	...
Data Frame n (15 words)	Pad Word
Pad Frame 16 words	

Table 3: Writing Frames - XCV50

There are eleven configuration registers in Virtex devices, used to program the configuration memory, read the data from this memory and control the chip operation. Each register is accessed through the register field in a 32-bit command word. All the data and command words have 32 bits. The command word is followed by one or more data words, according to the Word Count field (last eleven bits). If there is more then 2,047 words following the command word, the word count must be zero and, before the data words, a special command word has to be used, allowing 1,048,575 words to be read or written from configuration memory.

The following sections describes each register normally used in the original bitstream generated by Xilinx tools.

2.3.1 Command Register (CMD)

The Command Register (CMD) is used to control the operation of the configuration state machine, the Frame Address Register (FAR) and some global signals. The format of the command word for the CMD is shown bellow (32-bit word in hexadecimal format):

3	0	0	0	8	0	0	1	Write CMD
0	0	0	0	0	0	0	X	Command X

Table 4: CMD Register

The commands and their functions are shown in Table 5.

Cmd	X	Description
WCFG	1	Write Configuration Data. Used prior to writing configuration data to the FDRI. It takes the internal configuration state machine through a sequence of states that control the shifting of the FDR and the writing of the configuration memory.
LFRM	3	Last Frame. This command is loaded prior to writing the last (pad) data frame if the GHIGH_B signal was asserted. This command is not necessary if the GHIGH_B signal was not asserted. This allows overlap of the last frame write with the release of the GHIGH_B signal.
START	5	Begin Startup Sequence. Starts the startup sequence. This command is also used to start a shutdown sequence prior to partial reconfiguration. The Startup Sequence begins with the next successful CRC check.
RCRC	7	Reset CRC. Used to reset CRC register.
AGHIGH	8	Assert GHIGH_B Signal. Used prior to reconfiguration to prevent contention while writing new configuration data. All CLB outputs and signals are forced to a one.
SWITCH	9	Switch CCLK Frequency. Used to change (increase) the frequency of the Master CCLK.

Table 5: Commands for CMD Register

2.3.2 Configuration Option Register (COR)

The Configuration Option Register (COR) is used to control the configuration options for the device. The format of the command word for the COR is shown bellow (32-bit word in hexadecimal format):

3	0	0	1	2	0	0	1	Write COR
X	X	X	X	X	X	X	X	Option Word

Table 6: COR Register

The options used by PARBIT are shown in Table 7.

Option Word	Description
00803F2D	No pipeline stage for DONEIN; DONE pin is open drain; readback capture is one-shot; CCLK=4.3 MHz; Startup sequence clock = CCLK; no waiting for DLL lock signal; startup sequence; DONE active in fourth cycle of startup sequence; GTS inactive in fifth cycle of startup sequence; GWE active in sixth cycle of startup sequence; GSR inactive in sixth cycle of startup sequence. These are the default options.
00903F2D	Default, except:CCLK=User Clock.
0090FF2D	Default, except:CCLK=User Clock; No Shutdown; DONE=Keep State.
00A03F2D	Default, except:CCLK=JTAG Clock.
00A0FF2D	Default, except:CCLK=JTAG Clock; No Shutdown; DONE=Keep State.

Table 7: Options for COR Register

2.3.3 Control Register (CTL)

The Control Register (CTL) defines some device properties, after its configuration: security (read/write permissions), configuration interface and I/O pins. The format of the command word for the CTL is shown in Table 8, along with the option word used to keep the configuration interface after device programming.

3	0	0	0	A	0	0	1	Write CTL
0	0	0	0	0	0	4	0	Option Word

Table 8: CTL Register

2.3.4 Cyclic Redundancy Check Register (CRC)

The Cyclic Redundancy Check Register (CRC) is used to check the data written to any configuration register. It is calculated according to an specific algorithm [15], each time a configuration register is written. When the write CRC command is executed, the value of this command is checked against the internal value. If this is an incorrect value, the device is put in an ERROR condition, blocking its functioning. The format of the command word for the CRC is shown bellow (32-bit word in hexadecimal format):

3	0	0	0	0	0	0	1	Write CRC
0	0	0	0	X	X	X	X	CRC Value

Table 9: CRC Register

2.3.5 Frame Address Register (FAR)

The Frame Address Register (FAR) is used to indicate the block type (bits 26,25), major address (MJA) (bits 24 to 17) and minor address (MNA) (bits 16 to 9). The format of the command word for the FAR is shown bellow in Table 10, with type=CLB; MJA=4 and MNA=0.

3	0	0	2	0	0	0	1	Write FAR
0	0	0	8	0	0	0	0	FAR Value

Table 10: FAR Register

2.3.6 Frame Data Input Register (FDRI)

The Frame Data Input Register (FDRI) is used to indicate how many words will be written to the configuration memory. The number of words will depend on the device frame lenght and the column that is being programmed. The format of the command word for the FDRI is shown in Table 11. If the number of words is more than 2047, it will be necessary to use the special command word, after the write command, with a word count equals to zero.

3	0	0	0	4	X	X	X	Write FDRI $(X < 2048)$
3	0	0	0	4	0	0	0	Write FDRI $(WordCount = 0)$
5	0	0	X	X	X	X	X	$(X \geq 2048)$

Table 11: FDRI Register

2.3.7 Frame Length Register (FLR)

The Frame Length Register (FLR) shows how many 32-bit words are necessary to form a frame (without the pad word). The format of the command word for the FLR is shown in Table 12.

3	0	0	1	6	0	0	1	Write FLR
X	X	X	X	X	X	X	X	FLR Value

Table 12: FLR Register

2.3.8 Mask Register (MASK)

The Mask Register (MASK) is used to enable writing to the CTL Register. A “1” in bit N of this register allows that bit position to be written. The format of the command word for the MASK is shown in Table 13.

3	0	0	0	C	0	0	1	Write MASK
X	X	X	X	X	X	X	X	MASK Value

Table 13: MASK Register

2.4 Original Bitstream Format

The format shown in Figure 6 is an example of a configuration bitstream file. This file is used by PARBIT to read the partial reconfigurable area defined by the user. This is also the format utilized in the target bitstream file, when the tool operates in block mode (see next section).

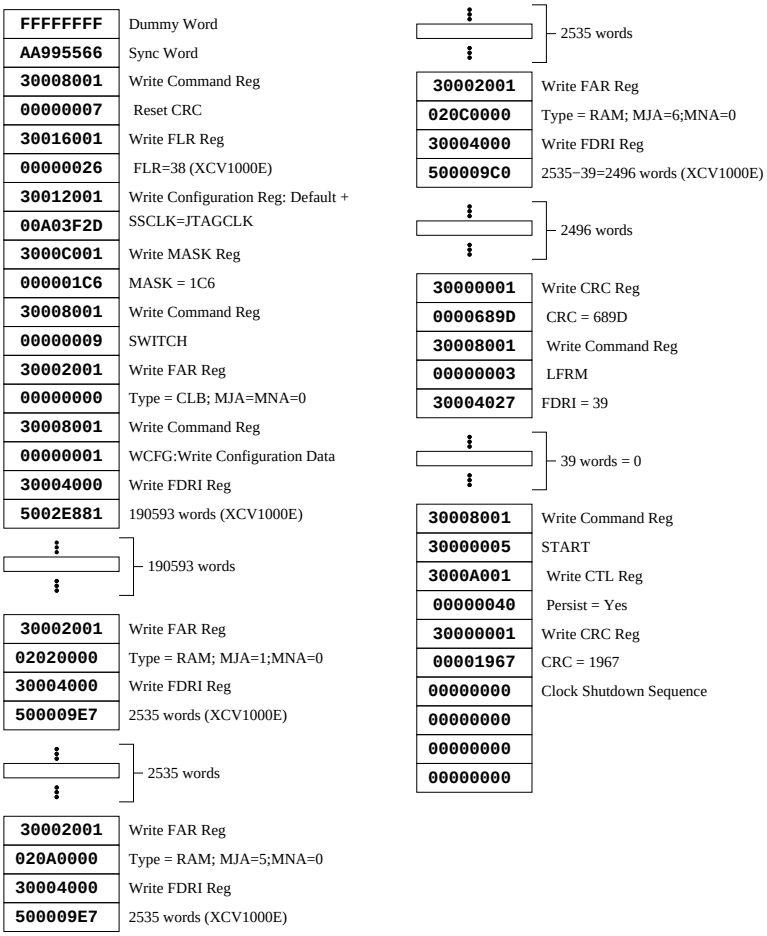

Figure 6: Original Bitstream

3 PARBIT

In order to generate the partial bitstream file, PARBIT reads the configuration frames from the original bitstream and copies to the partial bitstream only the configuration bits related to the area defined by the user. It then generates new values to the configuration address registers, according to the partial reconfigurable area.

If the block mode is used, the tool also reads the target bitstream file to copy the configuration bits that are inside a column specified by the user, but outside the partial reconfigurable area. This happens due to the fact that one frame occupies all the rows of a column and, in block mode, the partial reconfigurable area is smaller than a whole column. Another task performed by PARBIT is the reallocation of the partial reconfigurable area: the tool calculates new frame addresses according to the new coordinates chosen by the user.

The next sections describe how PARBIT implements these tasks and how the user operates the tool.

3.1 Organization

This section describes the main modules of PARBIT, used to read the original bitstream, read the target bitstream (when needed), get the user parameters and write the partial bitstream file. There are two main

operation modes defined by the user parameters:

- Slice: in this mode the user specifies a slice containing one or more full CLB columns. The tool generates the partial bitstream with these columns, in the same position they were in the original bitstream file.
- Block: in this mode it is possible to define an area inside the CLB columns of the chip, without the top and bottom IOB frame control bits. Then, the user defines where to put this block, in a bitstream of the same type as the original one (target bitstream). The tool generates the partial bitstream file containing the area selected by the user (from the original bitstream) and this file will be used to reconfigure the target device.

Besides the main operation modes, there are other options available to the user:

- Shutdown: it is possible to choose if the chip will be turned off prior the reconfiguration or not, with the latter being the most usual;
- Configuration port: the user can choose if the configuration port used to load the bitstream file will be the JTAG or the SelectMAP;
- Side: it is possible to generate a partial bitstream to program only the left part or the right part of the chip (slice mode only).

The PARBIT flow chart is shown in Figure 7.

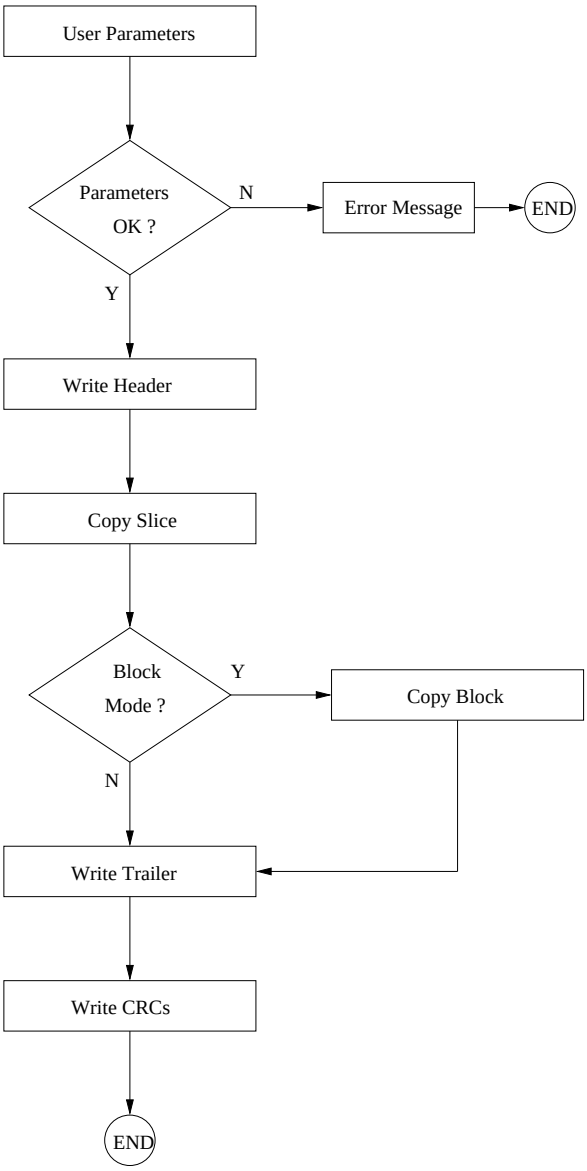

Figure 7: PARBIT Flow Chart

The next sections describe each one of these processes.

3.1.1 User Parameters

In this process, the tool parses all the options chosen by the user and verifies that there are no inconsistencies. It then reads the original bitstream and checks if the user options are compatible with the device. If there are errors, PARBIT issues a message and stops running.

3.1.2 Write Header

This process copies the header from the original bitstream and modifies some parts, in order to create the header for the partial bitstream. Before the dummy word (FFFFFFFF), it writes the value (11111111H), that will indicate how many bytes there are in the configuration bitstream. This value will be updated when the tool writes the trailer. It also programs the configuration registers, according to the user parameters.

3.1.3 Frame calculation

This process calculates the MJA for each CLB column of the partial bitstream. This MJA address will be used to write the FAR configuration register in the partial bitstream file. It is used also to generate the address of the first word of the first frame for that CLB column (fm_st_wd). This index is used to copy the frames from the original bitstream to the partial one. If it is a VIRTEX-E, there will be also an adjustment in these values, due to the existence of the BlockRAMs between the CLB columns.

The following formulas are used to calculate those values:

Left Side
$CLB\_Col \leq Chip\_Cols/2$
$MJA = Chip\_Cols - CLB\_Col * 2 + 2$
$fm\_st\_wd = FL * (8 + (MJA - 1) * 48)$
$RAM\_Bound = (Chip\_Rams/2 - 1) * RAM\_Space$
$MJA\_adj = 2 * ceiling((RAM\_Bound - CLB\_Col + 1)/RAM\_Space)$
$fm\_st\_wd\_adj = FL * (27 * MJA\_adj)$

Right Side
$CLB\_Col > Chip\_Cols/2$
$MJA = 2 * CLB\_Col - Chip\_Cols - 1$
$fm\_st\_wd = FL * (8 + (MJA - 1) * 48)$
$RAM\_Bound = Chip\_Cols - (Chip\_Rams/2 - 1) * RAM\_Space + 1$
$MJA\_adj = 2 * ceiling((CLB\_Col - RAM\_Bound + 1)/RAM\_Space)$
$fm\_st\_wd\_adj = FL * (27 * MJA\_adj)$

$MJA\_final = MJA + MJA\_adj$
$fm\_st\_wd\_final = fm\_st\_wd + fm\_st\_wd\_adj$

where:

Variable	Definition
MJA	Frame Major Address
fm_st_wd	Frame start word
MJA_final	Frame Major Address (VIRTEX-E)
fm_st_wd_final	Frame start word (VIRTEX-E)
CLB_Col	Column number of the desired CLB
Chip_Cols	Number of CLB columns on the Virtex device
Chip_Rams	Number of block SelectRAM columns on the Virtex device
RAM_Space	Spacing of block SelectRAM columns (in terms of CLB columns)
FL	Number of 32-bit words in the frame

More details can be found at [15]

3.1.4 Copy Slices(Slice Mode)

This process copies the CLB column frames from the original bitstream into the partial bitstream. Each CLB column chosen by the user has 48 frames. Before each block of configuration frames, the tool calculates the MJA and writes the FAR register with this value. After each block (except the last), the tool writes the Pad Frame. The flow-chart of this process can be seen in Figure 8.

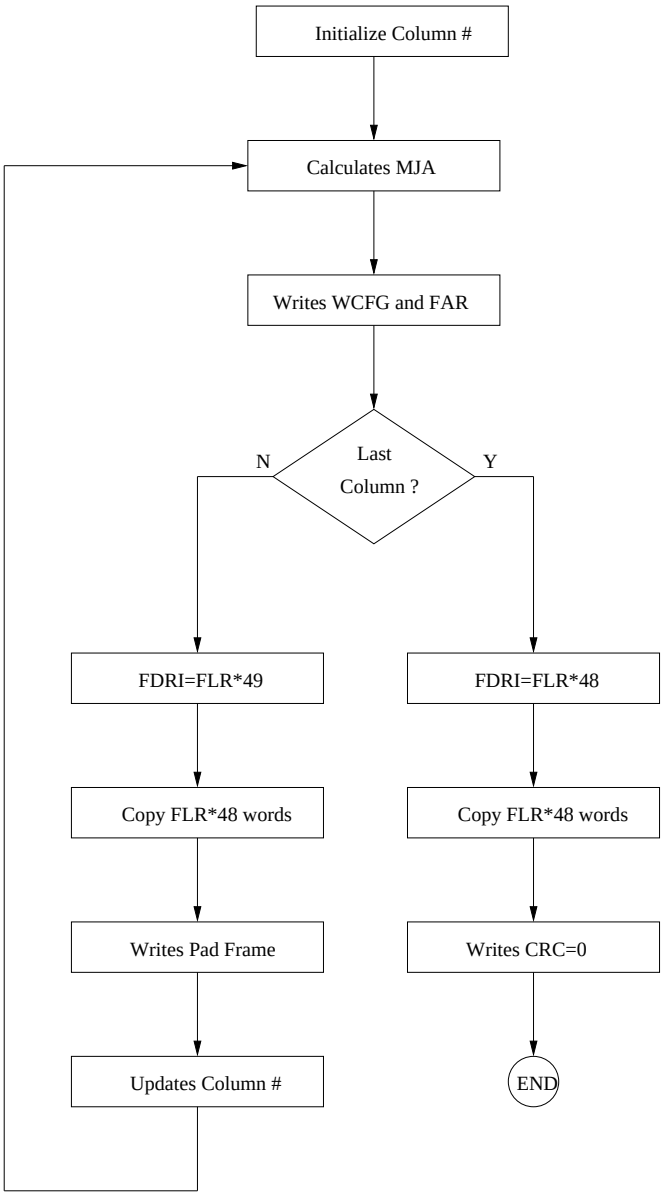

Figure 8: Copy Slice Flow Chart

3.1.5 Copy Block (Block Mode)

This process runs only if the user parameters define the block mode. It will run after the Copy Slice process and it has to open a second bitstream file, called “target”. The target bitstream file must have an empty area where the reconfigurable block will be placed. The “Copy Block” process opens the target bitstream and compares what parts of it are outside the block defined by the user. After this, it copies the bits that control these parts to the partial bitstream file. The flow for this process is shown in Figure 9.

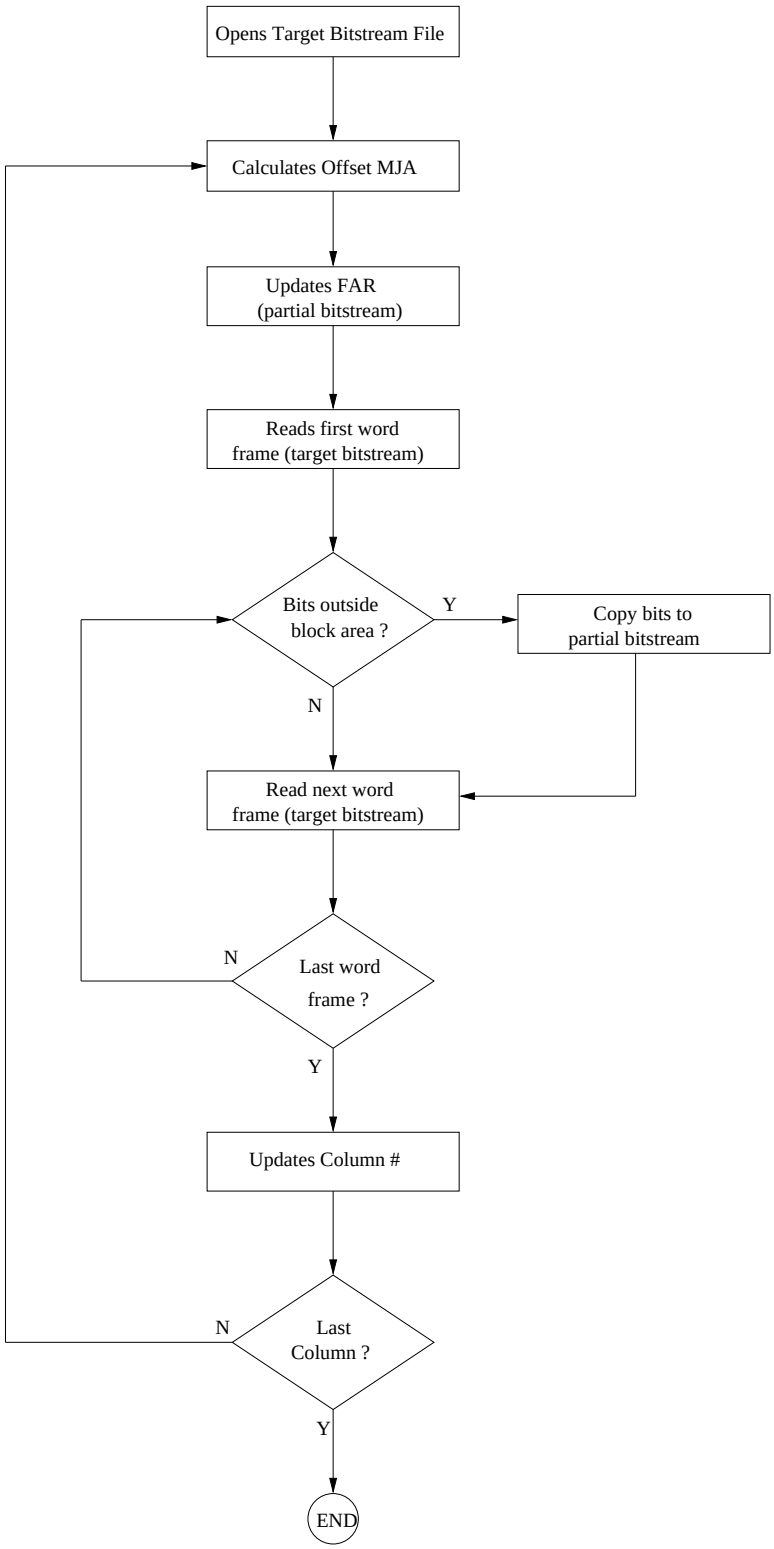

Figure 9: Copy Block Flow Chart

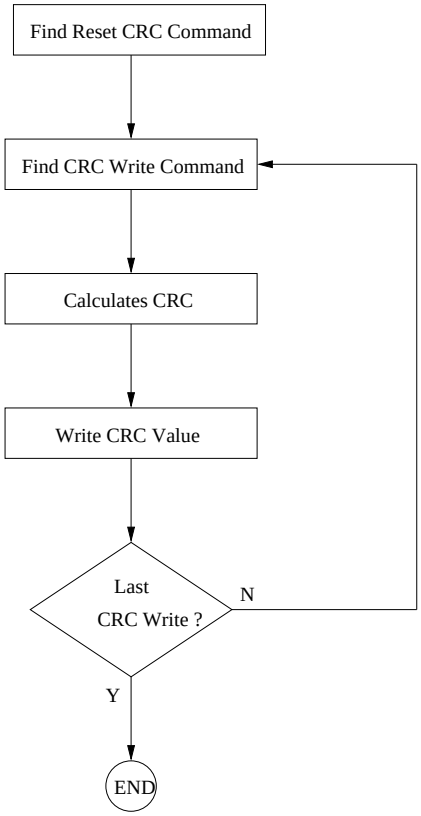

Figure 10: Write CRC Flow Chart

3.1.6 Write Trailer

After all the CLB column configuration frames are written in the partial bitstream, this process writes the last frame, programs all the remaining configuration registers, updates the configuration file size in the bitstream beginning and writes some information regarding the project that generated the bitstream.

3.1.7 Write CRC

This process parses the partial bitstream from the moment it resets the CRC value until the last CRC write command, calculating and updating each one of these values (see Figure 10).

To calculate the CRC value, the tool has to consider only the data written to some of the configuration registers. These data are mixed with the register address and transformed through a series of shifting and XOR logical operations. More details about this algorithm can be seen on [15].

3.2 Partial Bitstream Format

The partial bitstream generated by PARBIT has basically two formats: with and without shutdown. The options related to the configuration port and the slice/block modes produces similar bitstreams, differing only in the option words written in the configuration registers.

3.2.1 With Shutdown

A partial bitstream file, containing the configuration bits to reconfigure one CLB column, and generated with the option “Shutdown”, is shown in Figure 11.

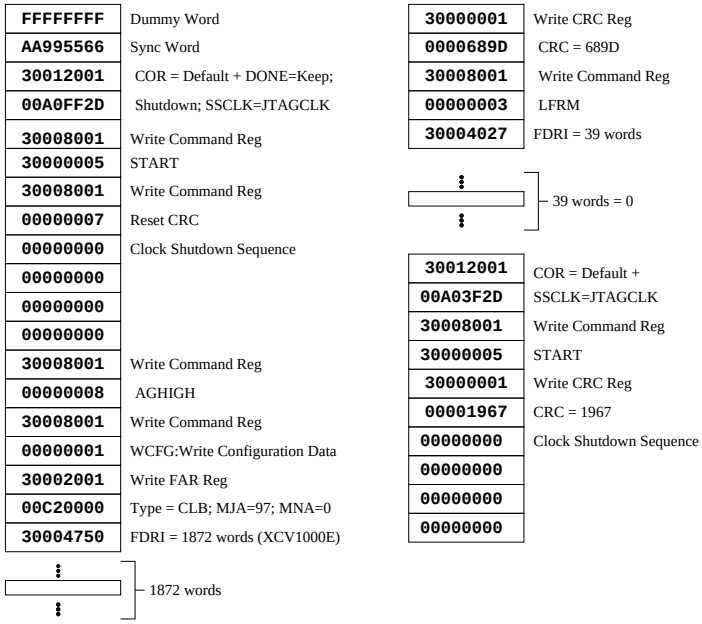

Figure 11: Partial Bitstream Example - With Shutdown

3.2.2 Without Shutdown

A partial bitstream file, containing the configuration bits to reconfigure one CLB column, and generated with the option “No Shutdown”, is shown in Figure 12.

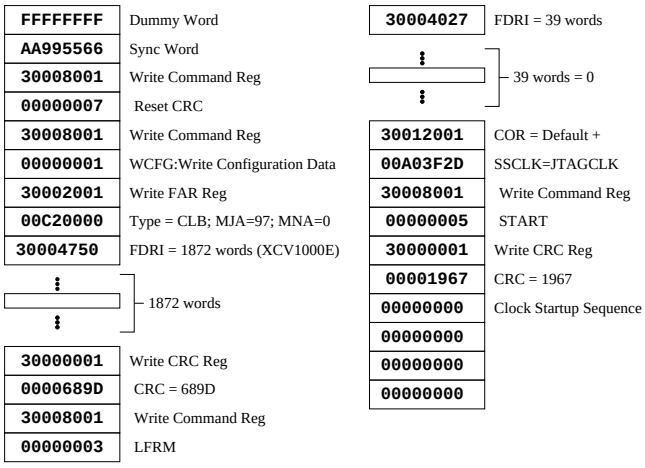

Figure 12: Partial Bitstream Example - Without Shutdown

3.3 Design Flow

The design flow with PARBIT depends on the configuration mode chosen by the user.

3.3.1 Slice Mode

In the “Slice Mode”, the user has to define only the start and end columns of the design to be reconfigured in the chip. The location of this design is always the same. See Figure 13.

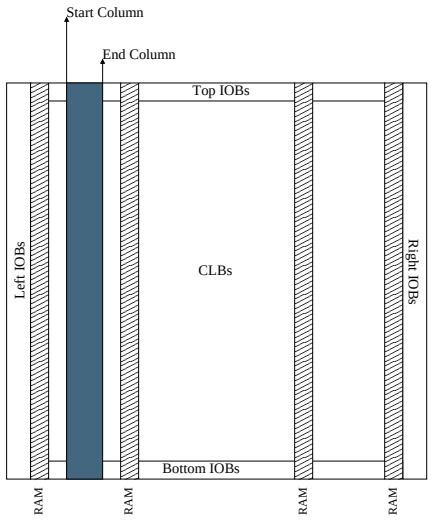

Figure 13: Slice Mode - Original Bitstream Device

The partial bitstream can be loaded into the device using the configuration port chosen by the user.

In this mode, all the I/O pins controlled by the CLB columns of the partial bitstream change according to the new design. It is important to keep these pins in the same location, when generating a new partial design.

3.3.2 Block Mode

In the “Block Mode”, the user has to define the following variables:

- Start/end columns: width of the partial reconfigurable area;
- Start/end rows: height of the partial reconfigurable area;
- Target row/column: the new location of the partial reconfigurable area, in the target project;

These values can be seen on Figure 14.

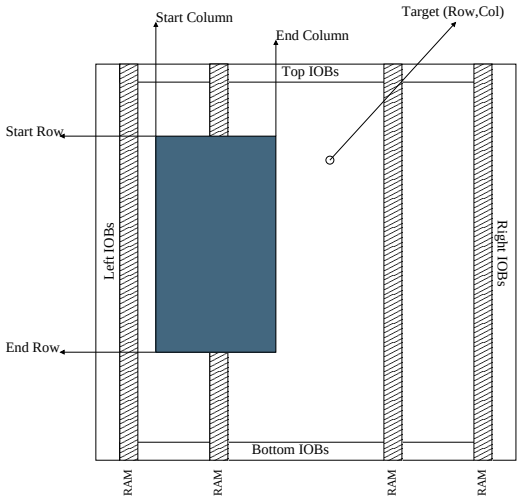

Figure 14: Block Mode - Original Bitstream Device

The tool needs the original bitstream and the target bitstream to generate the partial bitstream. After loading the new design into the target device, it will be like the one shown on Figure 15.

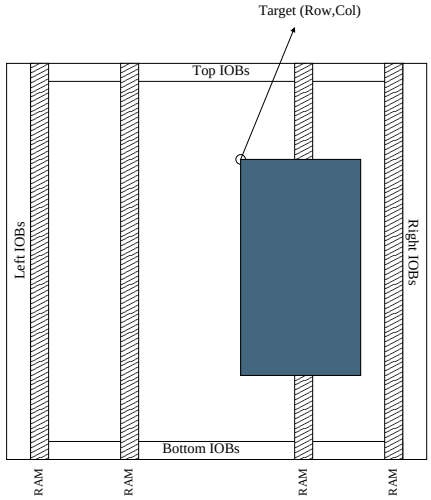

Figure 15: Block Mode - Target Device

The I/O pins from the target device did not change after the partial bitstream is loaded. The user has to define some boundary connection points to communicate with the new module.

3.4 Operation

PARBIT is a command-line based tool. Before running it, the user has to choose some parameters. These parameters have to be passed to the tool through an options file.

The command line is shown bellow, with all the options explained in detail:

parbit option original partial [target]

where:

- **Original** Bitstream file generated by Xilinx tools. This file contains the reconfigurable area that is extracted by PARBIT and transformed in a partial bitstream file. In the Slice Mode the reconfigurable area is placed in the same location it was set in the original design. In the Block Mode this area is the same, but it is placed in a different location.
- **Partial** Bitstream file generated by PARBIT, with the reconfigurable area defined by the user and its designated location. This location is the same one as in the original file (Slice Mode) or a new one (Block Mode), defined by user parameters and the target file.
- **Target** Bitstream file generated by Xilinx tools. It is necessary only in Block Mode. This file contains the fixed configuration for the FPGA, plus an empty area reserved to receive the reconfigurable area generated by PARBIT. The location of this empty area has to be passed to the tool (TargetRow, TargetColumn) and must have the same size as the partial reconfigurable area, defined in the option file.
- **Option** Text file containing user options. Each line of this file has the following format:

  OPTION:VALUE

The options available are the following:

- **FPGA** FPGA type. The available values are: XCVXCV50E, XCV100E, XCV200E, XCV300E, XCV400E, XCV405E, XCV600E, XCV812E, XCV1000E, XCV1600E, XCV2000E,XCV2600E, XCV3200E.
- **StartColumn** Start Column of partial reconfigurable area. This value ranges from 1 to the maximum available CLB columns in the chosen FPGA. It is used in both modes: Slice and Block.
- **EndColumn** End Column of partial reconfigurable area. This value ranges from 1 to the maximum available CLB columns in the chosen FPGA. It is used in both modes: Slice and Block.
- **StartRow** Start Row of partial reconfigurable area. This value is used only in Block mode.It defines, in conjunction with EndRow, the height of the partial reconfigurable area. In Slice Mode this height is equal to the chip height. This value ranges from 1 to the maximum rows in the chosen FPGA.
- **EndRow** End Row of partial reconfigurable area. This value is used only in Block Mode. It defines, in conjunction with StartRow, the height of the partial reconfigurable area. In Slice Mode this height is equal to the chip height. This value ranges from 1 to the maximum rows in the chosen FPGA.
- **TargetRow** Row of the new position of partial reconfigurable area. This option defines, with TargetColumn, the upper left corner of the new location of partial reconfigurable area inside the FPGA. It ranges from 1 to the maximum rows available in the FPGA, provided that it does not override the FPGA area. This option is valid only in Block Mode.

- **TargetColumn** CLB Column of the new position of partial reconfigurable area. This option defines, with TargetRow, the upper left corner of the new location of partial reconfigurable area inside the FPGA. It ranges from 1 to the maximum CLB columns available in the FPGA, provided that it does not override the FPGA area. This option is valid only in Block Mode.
- **Port** Programming port used to reconfigure the FPGA. The values are: JTAG (JTAG serial port) or SelectMAP (parallel). The default value is SelectMAP.
- **Shutdown** Start up sequence. Defines how the FPGA works, during reconfiguration. The values are: Yes (performs a shutdown before resuming new configuration) or No (the device continues working, during reconfiguration). The default value is No.
- **Side** Defines one reconfigurable area equal to half FPGA. The values are: Right (the right side of FPGA) and Left (the left side of FPGA). The tool generates the corresponding values for StartColumn and EndColumn automatically. This option is valid only in Slice Mode.
- **Verbose** Defines what kind of information appears on the screen, during PARBIT running. The value range is from 0 (none) to 4 (maximum).

4 Using PARBIT (Block Mode) to implement the DHPs in the FPX Board

One of the applications for PARBIT is the generation of the DHP [11] partial bitstream that is loaded in the FPX [16] board. Three examples of implementations are shown:

- Single-size DHP on XCV1000E;
- Double-size DHP on XCV1000E;
- Single-size DHP on XCV2000E;

The design methodology for build DHP modules on the FPX is to use standard tools to compile, place, and route logic into a fixed region of an XCV1000E or XCV2000E FPGA. After generating the source bitstream, PARBIT is run to transform the source file into a partial bitstream file.

4.1 Single-Size DHP (XCV1000E)

4.1.1 Original Bitstream File

The first step is the generation of the original bitstream file. This file contains the partial reconfigurable area, that will be loaded into the device. It is called 1S-DHP-org.bit. This file must have always the same location for the partial reconfigurable area. The user must confine the block between these coordinates:

- start column=8, end column=17;
- start row=7, end row=58.

The original bitstream has the format shown in Figure 16.

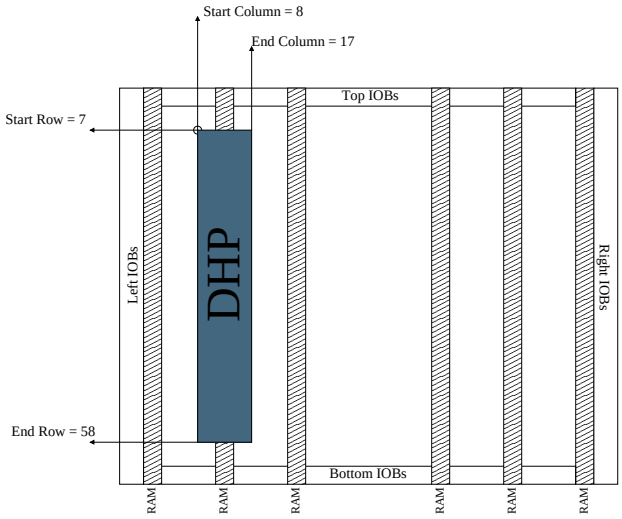

Figure 16: XCV1000E - Single-Size DHP - Original Bitstream

4.1.2 Target Bitstream File

The target design has to provide some specific empty areas to load the block design generated in the previous step. Each one of these areas is defined by two coordinates (Row, Col), as shown in Figure 17.

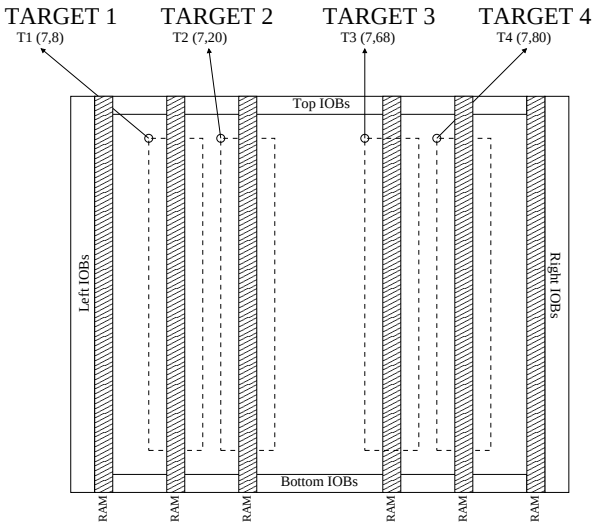

Figure 17: XCV1000E - Single-Size DHP - Target Bitstream

The target file is called 1S-DHP-target.bit.

4.1.3 Options Files

The options file (1S-DHP-T1.opt) for the Target 1 location must have the following lines:

```
FPGA:XCV1000E
StartColumn:8
EndColumn:17
StartRow:7
EndRow:58
TargetRow:7
TargetColumn:8
```

The options files related to the other possible locations of hardware plugins only have to change the value in the last line:

- Target 2 (1S-DHP-T2.opt):

  ```
  TargetColumn:20
  ```

- Target 3 (1S-DHP-T3.opt):

  ```
  TargetColumn:68
  ```

- Target 4 (1S-DHP-T4.opt):

  ```
  TargetColumn:80
  ```

4.1.4 Running PARBIT

With the files above, the next step is to run PARBIT:

parbit 1S-DHP-T1.opt 1S-DHP-org.bit 1S-DHP-T1-partial.bit 1S-DHP-target.bit

parbit 1S-DHP-T2.opt 1S-DHP-org.bit 1S-DHP-T2-partial.bit 1S-DHP-target.bit

parbit 1S-DHP-T3.opt 1S-DHP-org.bit 1S-DHP-T3-partial.bit 1S-DHP-target.bit

parbit 1S-DHP-T4.opt 1S-DHP-org.bit 1S-DHP-T4-partial.bit 1S-DHP-target.bit

The files 1S-DHP-T1-partial.bit, 1S-DHP-T2-partial.bit, 1S-DHP-T3-partial.bit, 1S-DHP-T4-partial.bit contain the partial bitstream necessary to load the DHP module onto locations T1, T2, T3 and T4, respectively.

4.2 Double-Size DHP (XCV1000E)

4.2.1 Original Bitstream File

The first step is the generation of the original bitstream file. This file contains the partial reconfigurable area, that will be loaded into the device. It is called 1D-DHP-org.bit. This file must have always the same location for the partial reconfigurable area. The user must confine the block between these coordinates:

- start column=8, end column=29;
- start row=7, end row=58.

The original bitstream has the format shown in Figure 18.

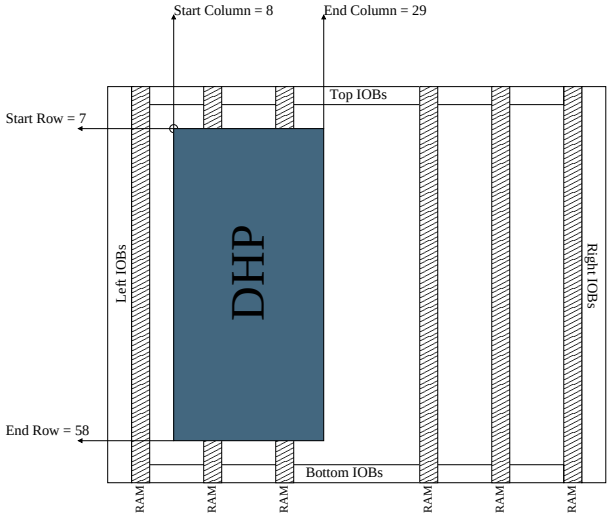

Figure 18: XCV1000E - Double-Size DHP- Original Bitstream

4.2.2 Target Bitstream File

The target design has to provide some specific empty areas to load the block design generated in the previous step. Each one of these areas is defined by two coordinates (Row, Col), as shown in Figure 19.

The target file is called 1D-DHP-target.bit.

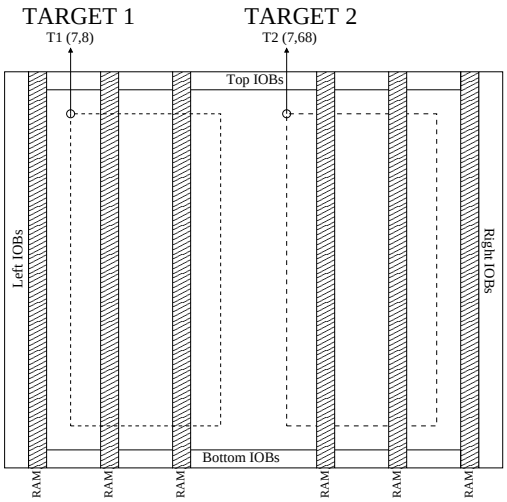

Figure 19: XCV1000E - Double-Size DHP - Target Bitstream

4.2.3 Options Files

The options file (1D-DHP-T1.opt) for the Target 1 location must have the following lines:

```
FPGA:XCV1000E
StartColumn:8
EndColumn:29
StartRow:7
EndRow:58
TargetRow:7
TargetColumn:8
```

The options files related to the other possible location of hardware plugin only have to change the value in the last line:

- Target 2 (1S-DHP-T2.opt):

  ```
  TargetColumn:68
  ```

4.2.4 Running PARBIT

With the files above, the next step is to run PARBIT:

parbit 1D-DHP-T1.opt 1D-DHP-org.bit 1D-DHP-T1-partial.bit 1D-DHP-target.bit

parbit 1D-DHP-T2.opt 1D-DHP-org.bit 1D-DHP-T2-partial.bit 1D-DHP-target.bit

The files 1D-DHP-T1-partial.bit, 1D-DHP-T2-partial.bit contain the partial bitstream necessary to load the DHP module onto locations T1 and T2, respectively.

4.3 Single-Size DHP (XCV2000E)

4.3.1 Original Bitstream File

The first step is the generation of the original bitstream file. This file contains the partial reconfigurable area, that will be loaded into the device. It is called 2S-DHP-org.bit. This file must have always the same location for the partial reconfigurable area. The user must confine the block between these coordinates:

- start column=8, end column=17;
- start row=7, end row=74.

The original bitstream has the format shown in Figure 20.

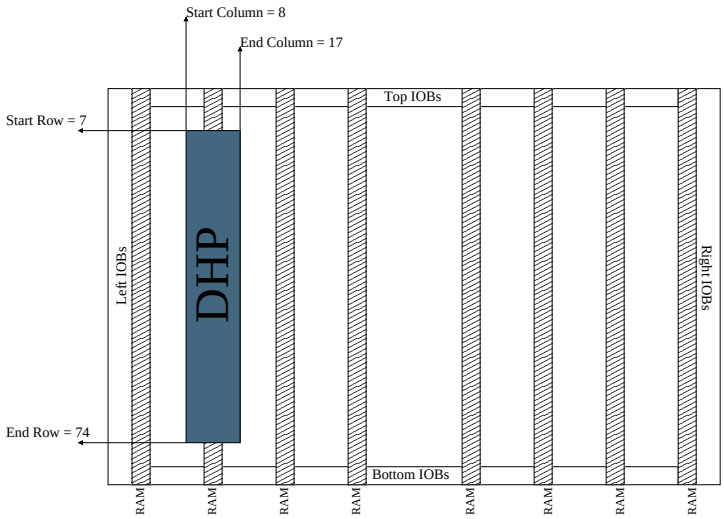

Figure 20: XCV2000E - Single-Size DHP - Original Bitstream

4.3.2 Target Bitstream File

The target design has to provide some specific empty areas to load the block design generated in the previous step. Each one of these areas is defined by two coordinates (Row, Col), as shown in Figure 21.

The target file is called 2S-DHP-target.bit.

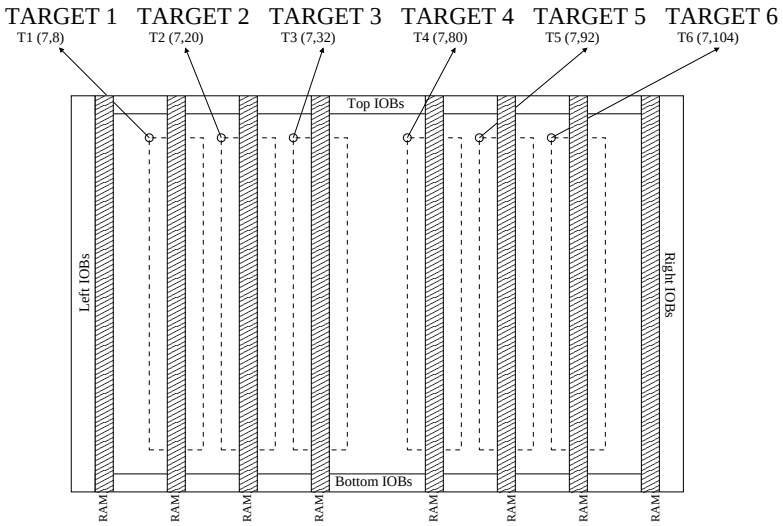

Figure 21: XCV2000E - Single-Size DHP - Target Bitstream

4.3.3 Options Files

The options file (2S-DHP-T1.opt) for the Target 1 location must have the following lines:

```
FPGA:XCV2000E
StartColumn:8
EndColumn:17
StartRow:7
EndRow:74
TargetRow:7
TargetColumn:8
```

The options files related to the other possible locations of hardware plugins only have to change the value in the last line:

- Target 2 (2S-DHP-T2.opt):

  ```
  TargetColumn:20
  ```

- Target 3 (2S-DHP-T3.opt):

  ```
  TargetColumn:32
  ```

- Target 4 (2S-DHP-T4.opt):

  ```
  TargetColumn:80
  ```

- Target 5 (2S-DHP-T5.opt):

  ```
  TargetColumn:92
  ```

- Target 6 (2S-DHP-T6.opt):

  ```
  TargetColumn:104
  ```

4.3.4 Running PARBIT

With the files above, the next step is to run PARBIT:

parbit 2S-DHP-T1.opt 2S-DHP-org.bit 2S-DHP-T1-partial.bit 2S-DHP-target.bit

parbit 2S-DHP-T2.opt 2S-DHP-org.bit 2S-DHP-T2-partial.bit 2S-DHP-target.bit

parbit 2S-DHP-T3.opt 2S-DHP-org.bit 2S-DHP-T3-partial.bit 2S-DHP-target.bit

parbit 2S-DHP-T4.opt 2S-DHP-org.bit 2S-DHP-T4-partial.bit 2S-DHP-target.bit

parbit 2S-DHP-T5.opt 2S-DHP-org.bit 2S-DHP-T5-partial.bit 2S-DHP-target.bit

parbit 2S-DHP-T6.opt 2S-DHP-org.bit 2S-DHP-T6-partial.bit 2S-DHP-target.bit

The files 2S-DHP-T1-partial.bit, 2S-DHP-T2-partial.bit, 2S-DHP-T3-partial.bit, 2S-DHP-T4-partial.bit, 2S-DHP-T5-partial.bit, 2S-DHP-T6-partial.bit contain the partial bitstream necessary to load the DHP module onto locations T1, T2, T3, T4, T5 and T6, respectively.

5 Conclusions

A tool called PARBIT has been developed that transforms and combines multiple bitstreams into files that can be used for partial run-time reconfiguration (P-RTR). It allows regions of logic to be extracted from an FPGA bitstream. It enables this logic to be relocated to another region of the device. Finally, it allows a partial reconfiguration block to reconfigure an FPGA even when the target block shares common frames with the original bitstream.

PARBIT enables DHP modules to be implemented on the FPX. The program accepts options to generate a bitstream which can load a DHP module into any region of the Reprogrammable Application Device on the FPX. PARBIT generates DHP modules that are single-size or double-size and can be used for RAD devices implemented with either an XCV1000E or XCV2000E.

References

[1] B. L. Hutchings and M. J. Wirthlin, "Implementation approaches for reconfigurable logic applications," in *Field-Programmable Logic and Applications (FPL'1995)* (W. Moore and W. Luk, eds.), (Oxford, England), pp. 419–428, Springer-Verlag, Berlin, Aug. 1995.

[2] D. T. Hoang, "Searching genetic databases on splash 2," in *IEEE Workshop on FPGAs for Custom Computing Machines* (D. A. Buell and K. L. Pocek, eds.), (Los Alamitos, CA), pp. 185–191, IEEE Computer Society Press, 1993.

[3] P. Bertin, H. Touati, and E. Lagnese, "PAM programming environments: Practice and experience," in *IEEE Workshop on FPGAs for Custom Computing Machines* (D. A. Buell and K. L. Pocek, eds.), (Los Alamitos, CA), pp. 133–138, IEEE Computer Society Press, 1994.

[4] J. M. Ditmar, "A Dynamically Reconfigurable FPGA-based Content Addressable Memory for IP Characterization," Master's thesis, KTH- Royal Institute of Technology, Stockholm, Sweden, 2000.

[5] J. D. Hadley and B. L. Hutchings, "Designing a partially reconfigured system," in *Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing, Proc. SPIE 2607* (J. Schewel, ed.), (Bellingham, WA), pp. 210–220, SPIE – The International Society for Optical Engineering, 1995.

[6] D. Ross, O. Vellacott, and M. Turner, "An FPGA-based Hardware Accelerator for Image Processing," in *More FPGAs: Proceedings of the 1993 International workshop on field-programmable logic and applications* (W. Moore and W. Luk, eds.), (Oxford, England), pp. 299–306, 1993.

[7] S. McMillan and S. Guccione, "Partial run-time reconfiguration using JRTR," in *Field-Programmable Logic and Applications / The Roadmap to Reconfigurable Computing (FPL'2000)*, (Villach, Austria), pp. 352–360, Aug. 2000.

[8] E. L. Horta and S. T. Kofuji, "The architecture of a reconfigurable ATM switch (RECATS)," in *Workshop de Computação Reconfigurável*, (Marília, SP, Brazil), Aug. 2000.

[9] J. W. Lockwood, J. S. Turner, and D. E. Taylor, "Field programmable port extender (FPX) for distributed routing and queuing," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000)*, (Monterey, CA, USA), pp. 137–144, Feb. 2000.

[10] J. S. Turner, T. Chaney, A. Fingerhut, and M. Flucke, "Design of a Gigabit ATM switch," in *INFOCOM'97*, 1997.

[11] D. E. Taylor, J. S. Turner, and J. W. Lockwood, "Dynamic hardware plugins (DHP): Exploiting reconfigurable hardware for high-performance programmable routers," in *IEEE OPENARCH 2001: 4th IEEE Conference on Open Architectures and Network Programming*, (Anchorage, AK), Apr. 2001.

[12] X. Inc., "Virtex-E 1.8 v field programmable gate arrays." Xilinx DS022, 2001.

[13] C. Carmichael, "Virtex configuration and readback." Xilinx XAPP138, Mar. 1999.

[14] X. Inc., "Configuration and readback of Virtex FPGAs using (JTAG) boundary scan." Xilinx XAPP139, Feb. 2000.

[15] S. Kelem, "Virtex configuration architecture advanced user's guide." Xilinx XAPP151, Sept. 1999.

[16] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, “Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX),” in *ACM International Symposium on Field Programmable Gate Arrays (FPGA’2001)*, (Monterey, CA, USA), pp. 87–93, Feb. 2001.