BitMan: A Tool and API for FPGA Bitstream Manipulations

Khoa Dang Pham, Edson Horta and Dirk Koch
School of Computer Science
The University of Manchester
Manchester, UK
Email: {khoa.pham, edson.horta, and dirk.koch}@manchester.ac.uk

***Abstract*—To fully support the partial reconfiguration capabilities of FPGAs, this paper introduces the tool and API BitMan for generating and manipulating configuration bitstreams. BitMan supports recent Xilinx FPGAs that can be used by the ISE and Vivado tool suites of the FPGA vendor Xilinx, including latest Virtex-6, 7 Series, UltraScale and UltraScale+ series FPGAs.**

The functionality includes high-level commands such as cutting out regions of a bitstream and placing or relocating modules on an FPGA as well as low-level commands for modifying primitives and for routing clock networks or rerouting signal connections at run-time. All this is possible without the vendor CAD tools for allowing BitMan to be used even with embedded CPUs. The paper describes the capabilities, API and performance evaluation of BitMan.

I. Introduction

FPGAs (Field Programmable Gate Array) have become more and more popular as this technology promises a massively parallel computing capability at relatively good power efficiency. For instance, Microsoft used FPGAs to accelerate their Bing search engine and demonstrated 95% throughput improvement at only 10% extra power [1].

However, compared to software development, FPGA development remains too complex. Given a user specification, a stack of transforming tool is executed for generating the bitstream binary for the FPGA. As illustrated in Figure 1a, this includes frontend design, logic synthesis, hardware implementation, and bitstream generation. All these processes take substantial amount of time, and large designs could easily take a day to complete.

High performance reconfigurable systems, such as proposed in the projects EXTRA [2], ECOSCALE [3], or OpenStack-enabled virtualized FPGA platform [4] require run-time allocation of hardware accelerators and heavily use partial run-time reconfiguration of the FPGA resources. However, fully flexible replacements are hard to achieve since a relocatable module requires inter-communications to other modules as well as fitting clock resources in order to work properly. This level of automation does not exist in current vendor design flows.

To enable flexible module placement, bitstream manipulation is essential. With a deep understanding of the bitstream format, we are able to generate and parameterize a new design without going back through the whole design flow. We can change the configuration of FPGA primitives (e.g., LUT values or memory (BRAM) contents), reroute wires, and reconfigure clock buffers. Another application is relocation and duplication of hardware modules. We can also support mapping overlay architectures to fabrics as well as routing through physical LUTs or FPGA resources [5]–[7]. The conventional flow is summarized in Figure 1a and an alternative using bitstream manipulation is suggested in Figure 1b.

In this work, we propose a generic methodology to analyze and manipulate the Xilinx FPGA bitstreams, including latest devices such as Zynq-7000 and Kintex UltraScale families. We provide a low level API providing access to FPGA fabric resources such as LUT/BRAM contents, routing and clock resources. Furthermore, high level functions such as module placement and relocation are fully supported.

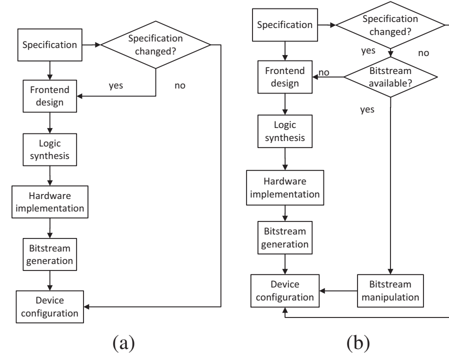

Fig. 1: New design flow using the bitstream manipulation tool. In (b), we are running through the full design cycle only in case we don't have the needed bitstream available for implementing system changes.

Besides a generic *X-Y* coordinate system abstraction for defining geometrical parameters, BitMan supports a coarse abstraction in resource column of a definable height (which in the case of Xilinx FPGAs will typically be the height of a clock region which is the smallest vertically atomically reconfigurable unit of these FPGAs). The module placement has to consider the primitive layout of the fabric and we adopted a string model approach as presented in [8].

The remaining of this paper is organized as follows. In Section II, we review the role of the FPGA bitstream and previous attempts to analyze and manipulate it. Section III discusses about how we have implemented the proposed methodology based on the bitstream format. Applications in dynamic partial reconfiguration and overlay architecture will be demonstrated and discussed in detailed in Section IV. Section V will summarize the work.

II. Background

A bitstream contains all information of a design which is mapped, placed and routed on a dedicated FPGA chip. However, bitstream manipulation needs to be done with care since a corrupted bitstream may damage the device physically and permanently [9]. Fortunately, bitstream manipulation also enables powerful features such as updating designs at run-time, fully flexible module replacements, or even composing overlay architectures on-the-fly. To do this, we need detailed information about the bitstream format.

Early efforts, such as JBit [10], JBG [11] and ParBit [12], provided means to dynamically link and assemble partial hardware modules into FPGA fabric. However, these approaches are not supporting latest devices as well as not easily able to reroute connections to modules and maintain clock resources.

Previously, Note et al. suggested to use the Xilinx Description Language and cross-correlation algorithm to analyze the Xilinx bitstream and reconstruct the netlist [13]. We are not using the cross-correlation algorithm since all bitstream

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

information can be derived precisely for CLB, DSP, BRAM, and the interconnection fabric.

RapidSimth [14] released by the Brigham Young University can parse, manipulate, and export bitstreams for Xilinx Virtex 4, Virtex 5 and Virtex 6. Moreover, in their latest attempts, Kulkarni et al. provided a similar API for bitstream manipulation to change the LUT contents and switch blocks configuration in Virtex 5 and 7 Series devices as part of their Dynamic Circuit Specialization system [15], [16]. Their works were significant and we are aiming at generalizing it for later devices since they do not support any newer FPGAs than the 7 Series. Additionally, [14]–[16] are based on the old Xilinx ISE design suite which is obsolete for latest devices which do not allow easily porting these tools to recent FPGAs. Instead of using only Xilinx ISE, we can support both ISE and Vivado design suites.

It is worth mentioning that since the UltraScale family, later Xilinx devices are only supported on the Vivado design tool. Our API provides a path to support latest Xilinx FPGA members using TCL scripts supported in Vivado. Moreover, we are not only targeting small bitstream manipulation but the replacement of large modules in a complex system.

III. Implementation

In this section, we are taking a closer look into a bitstream's structure, frame address, resource description and how they are being used in the BitMan tool.

A. Bitstream Format

The FPGA bitstream consists of configuration commands and configuration data. It has a header (including a bus width detection pattern, a SYNC word, and some configuration commands) and the actual configuration data (for all primitives and the routing), which is followed by a footer. We refer readers to configuration user guides from Xilinx vendor such as [17] and [18] for further information about header, bus width pattern and SYNC word. A footer may have CRC values, if any, and a DESYNC word to indicate the end of configuration data. In this work, we focus on the configuration frames, the device description and on how an FPGA device is reconfigured in order to help understanding how the bitstream manipulation tool works.

1) Configuration memory frames: The *configuration memory frames* are atomic, non-divided elements in FPGA configuration data. Each frame has its own address, which consists of a minor, major, and row address field as well as the block type of the resource (e.g., the routing of BRAMs and the actual BRAM content are stored in different sections of the bitstream, each belonging to a different block type). Consequently, the block type identifies if a resource is CLB (Configurable Logic Block), BRAM content or CFG_CLB [19]. Please note that the allocation of the FPGA resources into block type may vary across different device families of the vendor Xilinx and BitMan is designed very generic to take such family specific properties into account.

The row address shows which row of clock regions the resources belong to, while the major address specifies the resource column. The minor address, in turn, defines a specific configuration frame within a specific column of resources.

2) Device resource description: Xilinx FPGAs are organized as *resource columns* in terms of CLB, Block RAM, DSP, clock and other I/O. Multiple columns are gathered as a *resource row*. Figure 2 illustrates a Kintex UltraScale XCKU025 device layout with details of one resource row.

Resource columns consist of frames for the configuration of the corresponding primitives and for routing. Every column provides routing resources such as switch boxes with routing multiplexers. These routing resources are used for implementing the signal wiring inside the FPGA fabric. The configuration bits controlling the routing resources are encoded in the

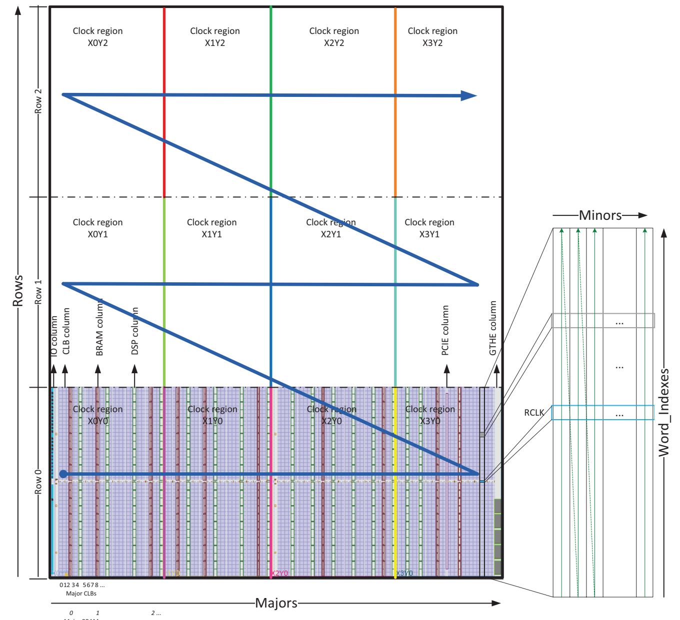

Fig. 2: Overview of Kintex UltraScale XCKU025's device layout. This device has in total 3 repetitive resource rows and 12 clock regions. Figure adopted from Vivado Design Tool.

TABLE I: Resource information in the Xilinx Virtex-6, 7 Series and UltraScale families.

Device family	Virtex-6		
Resource column	CLB	DSP	BRAM
Block Type	0x000	0x000	0x001
# of frames for interconnect	28	28	28
# of frames for content	8	0	128
Device family	7 Series		
Resource column	CLB	DSP	BRAM
Block Type	0x000	0x000	0x001
# of frames for interconnect	28	28	28
# of frames for content	8	0	128
Device family	UltraScale		
Resource column	CLB	DSP	BRAM
Block Type	0x000	0x000	0x001
# of frames for interconnect	58	4	(58 + 4)
# of frames for content	12	0	128

bitstream file together with the configuration of all other primitives of the FPGA.

The resource's architecture as well as the number of frames per column in one row stays the same in a family, but commonly differs from family to family. For example, a CLB on a 7 Series device has 8 frames for its content, but 12 frames on a UltraScale counterpart. The number of frames for routing is also different due to differences in the routing fabric. Table I gives a summary on the number of frames for a couple of device families that are all supported by BitMan.

Figure 3 shows how a connection in a switch box is encoded in the bitstream. We refer readers to see [20] for more details on the implementation of switch matrix multiplexers on modern FPGAs.

Figure 4 shows how a clock resource is encoded in the bitstream. By changing the configuration data, we could reroute clock signals. BitMan provides an API that allows reporting and manipulating clock tree and other routing resources by simply providing the resource column and the corresponding clock routing wires to be connected or deleted.

In BitMan, we can only manipulate the CLB, BRAM contents, routing, and clock resources because this is sufficient for

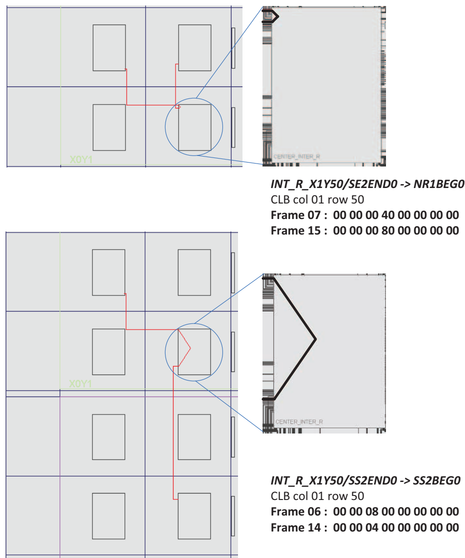

Fig. 3: Different routings resulted in different bitstream encodings.

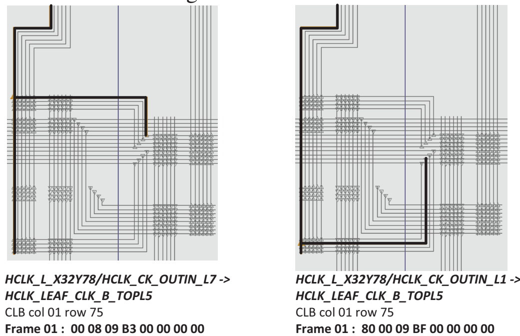

Fig. 4: Clock resource encodings in bitstream.

dynamically reconfiguring a partially reconfigurable module. Other resources, including I/O blocks, Gigabit transceiver, or hardened cores are commonly part of a static system that usually does not require run-time adaptation though means of reconfiguration.

B. Module Placement and Relocation

Module relocation is achieved by modifying address information fields inside the bitstream. BITMAN also checks the resource footprint of the FPGA primitives similar as proposed in [8].

C. Bitstream Manipulation Tool

BITMAN can be used as an independent tool or integrated to a controller as a software API. Table II shows BITMAN API examples exposed to higher level applications.

Figure 5 shows the operation of BITMAN. The whole input bitstream will be read and stored in a 2-D array *FrameBuffer*. BITMAN also receives commands from higher level applications. `(X, Y)` coordinate system refers to a grid at CLB granularity. Alternatively, a grid of the height of a clock region in vertical dimension can be used instead for convenience. All low level details of the bitstream are hidden from the user and BITMAN translates user-friendly commands into low level bitstream manipulation.

BITMAN is written in `ANSI C` and could be run on different platforms, from a desktop computer with Intel Core i7 to an embedded ARM Cortex-A9 or a softcore CPU. Its performances in various examples will be evaluated in the next Section.

TABLE II: BITMAN functions

High-level APIs	Functionality
`replace_FPGA_region(`X_0`,` Y_0`,` X_1`,` Y_1`,` X_2`,` Y_2`)` `duplicate_FPGA_region(`X_0`,` Y_0`,` X_1`,` Y_1`,` X_2`,` Y_2`)`	Replace/duplicate a rectangular FPGA region bounded by bottom-left (X_0, Y_0) and top-right (X_1, Y_1) to a new region which starts at (X_2, Y_2). Replace will clear configuration data in the old region.
`reroute_wire(X, Y, input, output)` `reroute_clock(X, Y, input, output)`	Change configuration data of switch box/clock multiplexer `(X, Y)` to connect `input` to `output`.
`change_LUT_content(X, Y, LUT, new_config)` `change_BRAM_content(X, Y, new_config)`	Change the content of LUT `(LUT)`/BRAM `(X, Y)` to the `new_config` data.

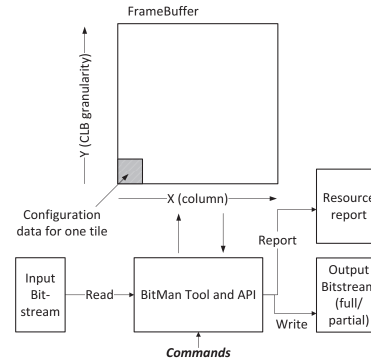

Fig. 5: BITMAN process

IV. APPLICATIONS AND EVALUATION

With the bitstream manipulation tool introduced in the previous section, we are going to discuss how it could bring benefits to applications using dynamic partial reconfiguration. Plain un-encrypted Xilinx FPGA bitstreams were used in below examples. BITMAN supports compressed bitstreams as generated by the Xilinx vendor tools, but does not support encryption. The later can be implemented easily by a system providing a secure storage mechanism.

A. Module relocation and duplication

A partial module might spread across a number of CLB and/or BRAM columns. It is worth mentioning that the reconfiguration of a module can be carried out without affecting surrounding modules or static system. In particular if some of the routing resources within a reconfigurable region implement static routing (e.g., for crossing signals of the surrounding system through a reconfigurable area), this is permitted and will have no side-effects due to a partial reconfigurable process. This requires routing constraints on the static routing through reconfigurable regions that can be generated with the GoAhead tool [9].

B. Rerouting

We are able to reroute clock signals by reconfiguring clock multiplexers in BUFG or BUFHCLK cells. By doing this, a relocatable module could be disabled/enabled or maintained

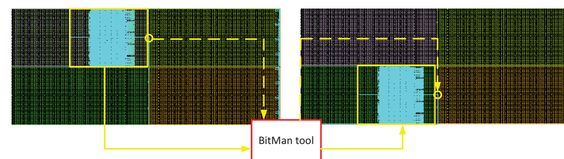

Fig. 6: An example of module relocation.

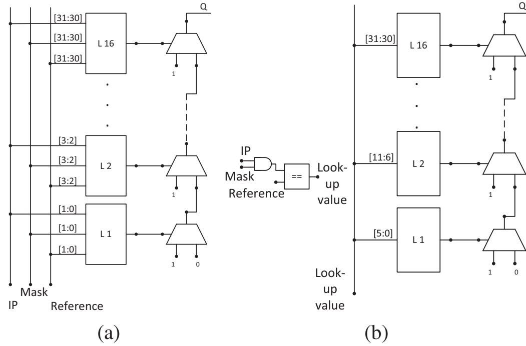

Fig. 7: Conventional CAM (a) vs. LUT-modifiable CAM (b).

its operation at a different frequency. This is also needed to keep the routing of the clock resources that belong to the static system untouched when partially reconfiguring a module.

Figure 6 shows an example of module relocation. There are 2 steps to achieve this moving: 1) relocate the whole module resource (fixed arrow), and 2) reroute clock signals for the relocatable module (dashed arrow). While simple systems may use one clock resources, BITMAN is designed for complete real-world systems that use a plurality of clock networks (e.g., for different memory controllers, NIC interfaces, PCIe, etc.). Any other routing resource including interconnection could be changed accordingly.

Table III show BITMAN performances on an ARM Cortex-A9 platform. In this experiment, a 2MB bitstream of the Zynq-7000 XC7Z010 device was used, and we have manipulated configuration data of an LUT, a CLB, a BRAM content, or a routing primitive, respectively.

C. LUT/BRAM content modification

BITMAN supports updating the content of LUTs and BRAMs in FPGA fabric on-the-fly. Application examples for this are changing coefficients in digital filters, updating keys in cryptography systems or swapping binaries stored in on-FPGA memory using the configuration interface rather than some extra user logic. An example with LUT update for FIR coefficients was mentioned in [15].

For demonstration the usefulness of LUT content modifications, we looked into an application where we compare an IP address with a masked reference IP. While the logic savings are less significant, it can be seen that the CAM approach in Figure 7b results in a carry chain that is only about a third as long as the conventional approach in Figure 7a.

D. Hardware mapping and linking for the overlay architecture

In [7], an approach for rapidly building overlay CGRA (Coarse Grained Reconfigurable Array) is presented where a small number of physically implemented PE modules were replicated for building large CGRAs with a hundred or more PEs. This approach tries to amortize CAD tool time for one PE to build large scale systems. With the help of stitching together fully placed and routed PE tiles, CAD tool times could be reduced by 9.3 times.

However, the stitching itself in [7] was carried out at the netlist level which requires a time-consuming netlist translation process that we circumvented by stitching PE tiles directly at the bitstream level. To demonstrate BITMAN, we repeated the same experiments but instead of stitching at the netlist level, the stitching was performed at the bitstream level. As shown in Table IV, this reduces the whole bitstream generation process into the range of seconds.

TABLE III: BITMAN performances on various platforms

System configuration	Processing time (μs)			
	LUT	CLB	BRAM	Routing
Dual-core ARM Cortex-A9 @ 866MHz and 512MB RAM - Linux 3.18	45	94	229	54

TABLE IV: BITMAN performance on overlay architecture's support

	BITMAN	Rapid Overlay Builder [7]
Numbers of PEs	101	101
Time (seconds)	2.24	2259

V. CONCLUSION

In this paper, we introduced the tool and API BITMAN that permits complete bitstream manipulation tasks to be carried out at run-time for all latest FPGAs of the vendor Xilinx without the need of running complex and time-consuming CAD tools. Various use cases were demonstrated and discussed to show BITMAN tool's advantages. This includes module relocation and duplication, modifying switch matrix and LUT settings as well as stitching together CGRAs from a PE library.

The results of BITMAN are configuration bitstreams that can be directly sent to the FPGA through any available configuration port (e.g., ICAP, or PCAP). BITMAN is available as a command line tool on Windows for x86 machines as well as a shared library on Linux for ARM (as provided on Zynq-7000 and UltraScale+ FPGAs) under [21].

ACKNOWLEDGEMENT

This work is supported by the European Commission under the H2020 Programme and the ECOSCALE project (grant agreement 671632).

Thanks Xi Yue from University of British Columbia for kindly providing us his experimental setup and results of the Rapid Overlay Builder.

REFERENCES

[1] A. Putnam et al., "A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services," in *ISCA*, 2014.
[2] D. Stroobandt et al., "EXTRA: Towards the Exploitation of eXascale Technology for Reconfigurable Architectures," in *ReCoSoC*, 2016.
[3] I. Mavroidis et al., "ECOSCALE: Reconfigurable Computing and Runtime System for Future Exascale Systems," in *DATE*, 2016.
[4] S. Byma et al., "FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack," in *FCCM*, 2014.
[5] D. Koch et al., "An Efficient FPGA Overlay for Portable Custom Instruction Set Extensions," in *FPL*, 2013.
[6] E. Hung and S. J. Wilton, "Towards Simulator-like Observability for FPGAs: A Virtual Overlay Network for Trace-buffers," in *FPGA*, 2013.
[7] X. Yue, "Rapid Overlay Builder for Xilinx FPGAs," Master's thesis, University of British Columbia, 2014.
[8] N. B. Grigore and D. Koch, "Placing Partially Reconfigurable Stream Processing Applications on FPGAs," in *FPL*, 2015.
[9] D. Koch, *Partial Reconfiguration on FPGAs: Architecture, Tools, and Applications*, 2012.
[10] S. Guccione et al., "JBits: Java based Interface for Reconfigurable Computing," in *MAPLD*, 1999.
[11] A. K. Raghavan and P. Sutton, "JPG - A Partial Bitstream Generation Tool to Support Partial Reconfiguration in Virtex FPGAs," in *IPDPS*, 2002.
[12] E. L. Horta et al., "Using PARBIT to Implement Partial Run-Time Reconfigurable Systems," in *FPL*, 2002.
[13] J.-B. Note and É. Rannaud, "From the Bitstream to the Netlist," in *FPGA*, 2008.
[14] C. Lavin et al., "RAPIDSMITH - A Library for Low-level Manipulation of Partially Placed-and-Routed FPGA Designs," NSF Center for High Performance Reconfigurable Computing (CHREC), Department of Electrical and Computer Engineering, Brigham Young University, Tech. Rep., 2014.
[15] A. Kulkarni and D. Stroobandt, "How to Efficiently Reconfigure Tunable Lookup Tables for Dynamic Circuit Specialization," *International Journal of Reconfigurable Computing*, 2016.
[16] A. Kulkarni et al., "A Fully Parameterized Virtual Coarse Grained Reconfigurable Array for High Performance Computing Applications," in *IPDPS*, 2016.
[17] Xilinx, *UG470 - 7 Series FPGAs Configuration User Guide*, 2015.
[18] Xilinx, *UG570 - UltraScale Architecture Configuration User Guide*, 2015.
[19] Xilinx, *UG621 - Virtex 5 Libraries Guide for HDL Designs*, 2009.
[20] D. Lewis et al., "The Stratix II Logic and Routing Architecture," in *FPGA*, 2005.
[21] K. Pham, "BitMan GitHub," git@github.com:khoapham/bitman.git, 2016.