International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 7, Issue 7, July 2017)

String Matching on Multicontext FPGA using Dynamic Partial Reconfiguration

J. Armstrong Joseph[1], Reeba korah[2], Salivahanan[3]

[1]*Centre for Research, Anna University, Chennai, 600025, India.*
[2]*Alliance College of Engineering and Design, Alliance University, Bangaluru, 562106, India.*
[3]*SSN College of Engineering, Kalavakkam, India.*

Abstract— **If logic be optimized for each problem instance, FPGAs do better than ASICs. CAD tools to generate problem instance dependent logic and time required configuring the FPGAs. In this paper, a novel approach for mapping and reconfiguration proposed that uses dynamic partial reconfiguration of FPGAs to do speed-up over existing approaches. Main idea is to design and map problem instance dependent logic on FPGA that maps problem instance dependent logic on other contexts of the same FPGA. As a result, CAD tools needed to use just once for each problem and not once for every problem instance as is usually done. To prove this approach, a detailed implementation of snort rule set based IDS using Boyer-Moore (BM) string matching algorithm presented. It implemented to get correct estimates of performance on FPGA device. Speedups in mapping time of approx. 10^6 over CAD tools and dynamic reconfiguration time 0.016 ms for 8 patterns obtained. Significant speedups obtained in overall mapping time as well including a speed up ranging from 3 to 16 times over a software implementation of snort rule set based IDS using BM algorithm running on a Pentium @ 2.8GHz workstation.**

Keywords— **partial reconfiguration, context switching, Dynamic reconfiguration, Field programmable gate arrays**

I. INTRODUCTION

Significant performance has obtained by considering reconfigurability of FPGA over modes of computation for several applications with mapping time and reconfiguration time. Mapping time refers to the time to be taken for compile, place and route the logic used on FPGA. The time needed to load the configuration data into the FPGA called reconfiguration time. Time has required to reconfigure a traditional FPGA is very high. Muticontext would be implemented to reduce the reconfiguration time; a device has more configuration context called as muticontext [9]. A single on-chip context has activated at a given time and other context could be activated in as fast as one clock cycle. This activation process is known as context switching [17]. The effectiveness of reconfigurable computing is better achievement by building hardware solutions for each single instance of a given problem. That is logic optimized for that particular instance, and loads them into a reconfigurable device to solve the instance problem.

Applications which produce instance independent logic will be loaded on a reconfigurable device is simply not exploiting the power of reconfiguration. Partial Reconfiguration (PR), is able to enable the reconfiguration process on a particular reconfigurable section of an FPGA design while the remaining part is still operating dynamically. Architectural enhancement to Xilinx FPGAs provides better support for dynamically reconfigurable designs. They would be augmented by a new design methodology. It uses pre-routed IP cores to communicate between static and dynamic modules. They permit static designs to route through regions otherwise reserved for dynamic modules. The CAD tool flow to automate this partial reconfiguration method is based on Xilinx Vivado 2014.2's [16] commercial tools for FPGA design has to do following steps are netlist import; Floor planning the design for partial reconfiguration checking design rules, exporting netlist, Implementation of flow management and bitstream size estimation. The configuration data made for programming the FPGA is called bitstream. In order to create the partial bitstreams, make logic function that controls logical elements. Also, make 8 logical elements of equal size partially reconfigurable modules. Reconfigurable modules would be described by changing the values in the LUTs of contiguous columns of chip and starting from the left side column and moving to the right side column of the chip. Thus 8 logical elements were initially created. This resulted in an overall of 8 partially reconfigurable modules that stored as partial bitstreams in the compact flash for the application snort [11, 12] rule set based IDS using BM string matching algorithm.

II. RELATED WORK

Reconfigurability in hardware systems that swap configurations within programmable device during runtime are achieved in various approaches towards minimum latency through partial reconfiguration and context switching. Coarse-grained reconfigurability can be accomplished by multiplexing various functional units. Fine-grained reconfigurability is less efficient because of a huge routing area required.

Due to increased space complexity, logical blocks [1] inside an FPGA has more than one LUT, flip-flop and mix of arithmetic, combinatorial, and multiplexing logic. Reiner Hartenstein et al [2] observed different architectures namely primarily mesh-based, linear arrays, crossbar-based architectures, etc. for coarse grain reconfigurable hardware. Dynamically configurable gate array (DPGA) concept [8, 9, 10, 13] was implementing for context switching in a run-time reconfigurable system and proposed by Bolotski et al. [14] who also discussed context swapping within an FPGA. RaPiD (Reconfigurable Pipelined Data path) [4] was proposed to speed-up regular computation demanding tasks by pipelines. Several parallel segmented 16 bit buses constitute routing and configuration. This architecture is based on the idea of providing a number of computing resources like ALUs, RAMs, registers and Multiplexers. PipeRench is a dynamically reconfigurable architecture allows reconfiguration of processing elements in each execution cycle. This architecture provides a global bus for data transfer. Kilocore KC256 is a commercial version of PipeRench [5] has specialized structure for the pipelined execution. Cryptographic applications have proposed partially reconfigurable application specific instruction set processor [3]. Fast SRAM based scratch pad had added to reconfigurable block to speed up the cryptographic application execution. Partially reconfigurable modules followed the difference based flow [6], [7] by making small changes on design; Then generating the bitstreams based on the differences made.

III. METHODOLOGY

Implementing a dynamical partial reconfigurable system on a FPGA introduces several other steps in the design process. In design process, look-up tables used as distributed RAM (DRAM) or shift registers (SRL16s) configured as dynamic part for reconfiguring. Partially reconfigurable region (PRR) used to describe the area of the FPGA that would be reserved for implementing all the tasks in one dynamically reconfigurable subset. The term partially reconfigurable module (PRM) used to describe a single dynamic task that would be mapped into a PRR. Each task in the subset has described as a partially reconfigurable module (PRM) that would be swapped in and out of its corresponding PRR at run-time. Design is being organized into a series of HDL files with a pre-determined file structure. The first phase of design is the top-level file; Top file has all global logic such as clock primitive's e.g. digital clock managers and global clock buffers; signal declarations; IO port, base design and partially reconfigurable (PR) module instantiations.

All communication (with a few exceptions for global signals such as clocks) would be explicitly declared using 8-bit bus macros provided by Xilinx. The goal of the budgeting phase is to find the size and place of the PRRs and to lock down the bus macros. Physical block (PBlocks) partition could be design physically and composed hierarchically for block nesting. PBlocks has placed and sized on a floorplan of the target FPGA. The static design components have mapped to a single PBlock. After the top level HDL file and customized constraints file (.UCF) for each module has created and synthesized for its necessary to translate this information into a native Xilinx format. The top level context includes IO placement, clock resources, bus macros, static and PRM module placement. This file used to pass context information between static implementation and PRM implementation phases. In static implementation phase, place and route the static subset of the design. This phase required for synthesized netlists. Netlists have include the static subset of the design be copied into a reserved directory. Finally MAP and PAR would be run, resulting in a placed and routed file for the static part of the design. In PRM implementation phase, builds PR modules that will go into the PR regions. Each PRM has its own directory. The PRM directories contain the synthesized netlist for the PRM and the user constraints file containing information about the particular PRM and associated context logic. The final implementation phase is the merge phase. In the merge phase, a complete design would be built from the static design and each PRM. The PR_verifydesign script would be generated for a merged full bitstream and partial bitstream for each PRM. The resulting bitstream used for the initial full bitstream with the desired PRMs already in place. Each PR regions have "blanking" bitstreams generated by PR_assemble script. A "blanking" bitstream is a default configuration. It has no PRM logic. The "blanking" bitstreams has the static route-throughs. Blanking Blanking bitstreams would be loaded when a PRM is not required which can reduce power consumption. Once the full and partial bitstreams generated, they have to be tested.

IV. PROPOSED IMPLEMENTATION ON A MULTICONTEXT FPGA

Before computation begins, the pattern P, pattern length M, text T and text length n have stored in external memory that would be accessed by the multi context FPGA. The following logic has configured eleven contexts of the FPGA.

Context 0 has control logic that governs overall execution of the snort ruleset IDS Application. Context 1 has logic for customizing the FSM for given m. and searching pattern using BM algorithm. Context 2 has data path for Arithmetic and logical operations of the BM algorithm as well as logic for runtime FSM construction. Hardwired into this logic is configuration bits for the OR-gate. Fig 1 and 2 Shows Dynamic partial reconfiguration and screenshot for IDS application architecture and computation performed on each context.

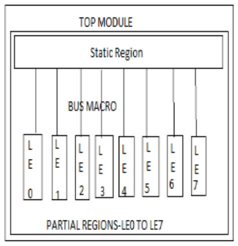

Fig 1. Dynamic partial reconfiguration and screen shot for IDS application architecture

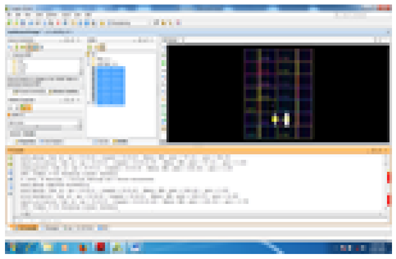

Fig 2. Snort ruleset based IDS implementation on a multi context FPGA.

```
context 0 (top )
/ *    Stage 1 of Instantiations of IO port, bus macro;
/*Stage 2 Declarations of digital clock managers (DCMs)
and BUFG global clock buffers) and signal;
 * / switch context 1;
if   AND datapath;*/ switch context 2;
else if OR datapath;*/ switch context 3;
else if NAND datapath;*/ switch context 4;
else if NOR datapath;*/ switch context 5;
else if XOR datapath;*/ switch context 6;
else if ADD datapath;*/ switch context 7;
else if SUBTRACT datapath;*/ switch context 8;
else if COMPARE EQUAL datapath;*/ switch context 9;

context 1 (Static)
/*Stage 1 of FSM construction based on Snort rule set
/*Stage 2 Searching pattern based on BM algorithm
/*Store pattern characters in pattern registers
*/ read em    ; Write cm; */switch context 0;
```

```
context 2 (Partial reconfigure Le0  )
read bits from SRAM for AND data path        ;*/ switch
context 1
context 3 (Partial reconfigure Le1  )
read bits from SRAM for OR data path         ;*/ switch
context 1
context 4 (Partial reconfigure Le2  )
read bits from SRAM for NAND data path      ;*/ switch
context 1
context 5 (Partial reconfigure Le3  )
read bits from SRAM for NOR data path        ;*/ switch
context 1
context 6 (Partial reconfigure Le4  )
read bits from SRAM for XOR data path        ;*/ switch
context 1
context 7 (Partial reconfigure Le5  )
read bits from SRAM for ADD data path        ;*/ switch
context 1
context 8 (Partial reconfigure Le6  )
read bits from SRAM for SUBTRACT data path         ;*/
switch context 1
context 9 (Partial reconfigure Le7  )
```

The context switching [17] is similar to context switching of processes on a uniprocessor. At a time only one of the FPGA contexts executes and switching to a context resumes its execution from where it had stopped earlier due to a context switch. This is possible because the state of the active context (bits stored in all the flip-flops) have saved before switching to a different context. Multicontext FPGA can very quickly switch between stored configurations. Now compare the mapping time (T_M + T_{ME}) of the proposed multicontext FPGA for Snort ruleset based IDS approach with other approaches. Table 1. Shows Speedup in mapping time (pattern size m=8) with other approaches. CAD tools need would be used just once for each problem and build logic with template logic. Software based snort IDS with BM algorithm for searching pattern has implemented on Pentium @ 2.8GHz workstation.

Table 1.
Speedup in mapping time (pattern size m=8)

Approach	Clock Time T_{clk}	Time to perform T_M	Time to reconfigure T_{ME}	Mapping time T= (T_M+ T_{ME})	Speedup
Multicontext FPGA	97.6 ns	3.7 µs	9 µs	2.1 µs	1.0
CAD tool	--	76 s	1 ms	76 s	Approx. $6x10^6$
Software IDS	--	20 ms	1 ms	20 ms	1892

Comparison of the implementation with other FPGA based string matching implementations Observed, that in [18] TM=0.16s 0 and TME=3.05s.

These times are for a naive string matching implementation on 16 CAL1024 FPGAs that runs at 20 MHz Thus, in [18], speedups will be obtained only for very large problem sizes due to high $T_M + T_{ME}$

Reconfiguration time estimated by the total loading time of a dynamic reconfiguration architecture by Load Time Total = #Patterns×(#Reg+#BL)/Freq (sec).

From the result of Table 1, the Multicontext FPGA took 2.1 μsec per PMM to load an input pattern; took the total loading times 0.016 msec to load 8 patterns.

Table 2.
Summary parameters of proposed pattern matching hardware

Class of Patterns	# Op	# Add	# Reg	# BL	# Slice	Freq	Throughput	Load Time Total	#Patterns	#Chars Total
Multicontext FPGA	2	1	89	32	327	363 MHz	11.6 Gbps	0.016 ms	8	256 letters
Exact string pattern [22]	5	0	3	256	54	363 MHz	2.9 Gbps	0.182 msec	256	8,192 letters

In Table 2, we show the summary of proposed pattern matching hardware, dynamic reconfigurable hardwares described as where #Op, #Add, #Reg, #BL, and #Slice are the numbers of 32-bit operations, 32-bit integer additions, registers, block RAM lines per PMM, respectively. The number of block RAM lines (#BL) is given by the number of block RAMs times $|\Sigma| = 256$. #Patterns and #Total chars are the number and the total size of input patterns, respectively.

Table3.
Results comparisons of regular expression matching hardware based on various dynamic reconfigurable architectures

Design	Class	Device	Throughput	bRAM/ char	Logic Cells/ char	#Chars Total	Remark
Multicontext FPGA	Snort	Virtex-7 X485T	11.6 Gbps	4 bytes/ char	0.78	256	327 slices
Dynamic BP-NFA for STR [22]	STR	Virtex-5 LX300	2.9 Gbps	4 bytes/ char	10.8	8192	54 slices
KMP-based hardware [19]	STR	Virtex-II Pro	1.8 Gbps	4 bytes /char	3.2	3200	NA
Bitsplit-based hardware [20]	STR	Virtex-4	1.6 Gbps	46 bytes/ char	1.4	16715	NA
RegExp Controller hardware [21]	REG	Virtex-4 FX100	1.4 Gbps	46 bytes/ char	2.56	16715	NA

In Table. 3. Shows comparisons of regular expression matching hardware based on various dynamic reconfigurable architectures. Multicontext FPGA is designed by a device Virtex-7 X485T and it has configured by dynamic partial reconfiguration using Xilinx Vivado v2014.2 [16]. The result is as high speedups in mapping time and reasonable reconfiguration time over various existing approaches using multicontext FPGA.

V. Conclusion

Speedups in mapping and reconfiguration time required to map logic at runtime FPGAs. It has achieved by the novel approach of developing logic that maps problem instance of application. The selected application for this logic is snort ruleset based IDS. CAD tool used just once for each problem through which build logic has done by template logic. The reduction in mapping time achieved is extremely important because FPGAs can do better than ASICs only if the mapping is problem instance dependent, which means that the runtime mapping time is a part of the overall execution time. Dynamic partial reconfiguration has performed using multicontext FPGAs and how to efficiently realize the above approach through the application snort ruleset based IDS with BM string matching algorithm. In this application, FSM construction has done by using snort ruleset. Such a fine-grained interleaving of mapping logic and using it would not be possible with software in the loop. Results show that high speedups in mapping time and reconfiguration time over various existing approaches.

REFERENCES

[1] Anupam Chattopadhyay, “Ingredients of Adaptability: A Survey of Reconfigurable Processors” VLSI Design Volume 2013 http://dx.doi.org/10.1155/2013/683615

[2] Reiner.H, “A Decade of Reconfigurable Computing: a Visionary Retrospective”, DOI: 10.1109 Source: IEEE Xplore Conference: Design, Automation and Test 2001

[3] K. Karuri, S. Kraemer, “A Tool Flow for Design Space Exploration of Partially reconfigurable Processors”, Integrated Signal Processing Systems RWTH Aachen University.

[4] C. Ebeling,”RaPiD: Reconfigurable Pipelined Datapath Architecture” 6th International Workshop on Field-Programmable Logic and Applications, FPL '96 Darmstadt, Germany, September 23–25, 1996 Proceedings.

[5] S. C. Goldstein et al.: “PipeRench: A Coprocessor for Streaming Multimedia Acceleration”; Proc. ISCA‘99, Atlanta, May 2-4, 1999.

[6] Xilinx Inc., “Two Flows for Partial Reconfiguration: Module Based or Difference Based. XAPP290 (v1.2) September 9, 2004,"

[7] “Difference Based Partial Reconfiguration. XAPP290(v2.0) December 3, 2007," 2007.

[8] Andre DeHon, “Dynamically programmable gate array with multiple contexts” https://www.google.com/patents/US5742180

[9] T. Fujii, K. Furuta, M. Motomura, M. Nomura, M. Mizuno, K. Anjo, K. Wakabayashi, Y. Hirota, Y. Nakazawa, H. Ito, and M. Yamashina, “A dynamically reconfigurable logic engine with a multi-context/ multi-mode unified-cell architecture,” Proc. Intl. Solid-State Circuits Conf., pp.360–361, 1999.

[10] T. Kitaoka, H. Amano, and K. Anjo, “Reducing the configuration loading time of a coarse grain multicontext reconfigurable device,” Proc. FPL (LNCS2778), pp.171–180, 2003

[11] Sourcefire, “Snort: The Open Source Network Intrusion Detection System” http://www.snort.org, 2016`.

[12] Haoyu Song, T. Sproull, M. Attig, J. Lockwood, “Snort offloader: a reconfigurable hardware NIDS filter”, Field Programmable Logic and Applications, 2005. International Conference on, 24-26 Aug. 2005

[13] A. DeHon. DPGA-coupled microprocessors: Commodity ICs for the early 21st century. In Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, pp. 31–39, 1994.

[14] M. Bolotski, A. DeHon, and T. Knight. Unifying FPGAs and SIMD Arrays. In Proceedings of the 1994 IEEE Workshop on Field Programmable Gate Arrays, IEEE Computer Society Press, Napa California, February 1994.

[15] The Xilinx Corporation, Virtex 7 Series FPGA Devices," 2016, http://www.xilinx.com

[16] Vivado Design Suite User Guide: Partial Reconfiguration, http://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_1/ug909-vivado-partial-reconfiguration.pdf

[17] SCALERA, S. M., AND V´AZQUEZ, J. R. The design and implementation of a context switching FPGA. In IEEE Symposium on Field-Programmable Custom Computing Machines (Apr. 1998), pp. 495–498.

[18] GUNTHER, B., MILNE, G., AND NARASIMHAN, L.Assessing document relevance with run-time reconfigurable machines. In Proceedings of the 4th IEEE Sym- posium on FPGAs for Custom Computing Machines (Napa, California, Apr 1996).

[19] Z. K. Baker and V. K. Prasanna, “Time and area efficient pattern matching on FPGAs,” in Proc. FPGA’04, pp. 223–232, 2004.

[20] H. J. Jung, Z. K. Baker and V. K. Prasanna, “Performance of FPGA implementation of bit-split architecture for intrusion detection systems,” in Proc. RAW’06, 2006.

[21] Yusaku Kaneta, Shingo Yoshizawa, Shin-ichi Minato, Hiroki Arimura, and Yoshikazu Miyanaga, “Efficient multiple regular expression matching on FPGAs based on extended SHIFT-AND method,” in Proc. SASIMI’10, Oct. 2010.

[22] Z. K. Baker, H. Jung, and V. K. Prasanna, “Regular expression software deceleration for intrusion detection systems,” in Proc. FPL’06, pp. 1–8, 2006.