ARCHITECTURE MODEL AND RESOURCE GRAPH BUILDING ALGORITHM FOR DETAILED FPGA ARCHITECTURE DESIGN[1]

Li Zhihua[* **] Yang Haigang[**] Yang Liqun[* **] Li Wei[**] Huang Juan[**]

[*](*University of Chinese Academy of Sciences, Beijing 100049, China*)

[**](*System on Programmable Chip Research Department, Institute of Electronics, Chinese Academy of Sciences, Beijing 100190, China*)

Abstract This paper addresses the issue of designing the detailed architectures of Field-Programmable Gate Arrays (FPGAs), which has a great impact on the overall performances of an FPGA in practice. Firstly, a novel FPGA architecture description model is proposed based on an easy-to-use file format known as YAML. This format permits the description of any detailed architecture of hard blocks and channels. Then a general algorithm of building FPGA resource graph is presented. The proposed model is scalable and capable of dealing with detailed architecture design and can be used in FPGA architecture evaluation system which is developed to enable detailed architecture design. Experimental results show that a maximum of 16.36% reduction in total wirelength and a maximum of 9.34% reduction in router effort can be obtained by making very little changes to detailed architectures, which verifies the necessity and effectiveness of the proposed model.

Key words Field-Programmable Gate Arrays (FPGAs) architecture model; Detailed architecture design; Architecture evaluation system

CLC index TN47

DOI 10.1007/s11767-014-4022-9

I. Introduction

Field-Programmable Gate Arrays (FPGAs) are widely used for implementing digital circuits because of their high levels of integration and rapid turnaround time. An FPGA consists of groups of heterogeneous hard blocks connected through programmable Connection Blocks (CB) and SWitch Blocks (SWB)[1]. In modern FPGAs[2–4], these hard blocks can be any types such as embedded memories and Digital Signal Processing (DSP) blocks. According to our design experiences, small discrepancies in detailed architecture design will make big difference in FPGA performances. The FPGA detailed architecture design mainly describes the connection of switch box and connection box, such as the population of each track at each location, the design of tracks at the corner of FPGA, the arrangement of CLB input and output pins and the mapping between CLB pins, and LUT pins. These factors impact the performance of FPGA. However, the current FPGA description models do not cover the detailed design above. Therefore, optimum design of detailed architectures is very important to the design of high performance FPGAs.

In order to develop a high-quality FPGA architecture, the FPGA architecture evaluation system is used to evaluate the utility of a huge number of architectural tradeoffs and decisions. Typically one “implements” (using a synthesis flow) a set of benchmark circuits in each FPGA architecture (or architecture variant) of interest, and determines the area required and speed achieved by these circuits in each of the architectures. The architecture which corresponds to the best combination of area, delay, and other metrics such as power, is evaluated as the best FPGA architecture, and is laid out and manufactured[5]. In this FPGA architecture evaluation system, an architecture model is first used to describe the detailed architecture information of FPGAs such as the hard block distribution, channel width and so on. Then the re-

[1] Manuscript received date: January 18, 2014; revised date: December 2, 2014.
Supported by National High Technology Research and Develop Program of China (No. 2012AA012301) and National Science and Technology Major Project of China (No. 2013ZX03006004).
Communication author: Yang Haigang, born in 1960, male, Doctor Degree. Institute of Electronics, Chinese Academy of Sciences, Beijing 100190, China.
Email: yanghg@mail.ie.ac.cn.

source graph of an FPGA is built based on the model. This resource graph is fed into the subsequent packing, placement, and routing procedures. Therefore, an FPGA architecture model and a general algorithm of building resource graph are the basis of FPGA detailed architecture design.

Several models that target FPGA architecture description have been developed, including those used in VPR 4.30[6], VPR 5.0[7], and VTR 1.0[8]. A general complex block model named CARCH is proposed in Ref. [9]. All of above models are high level models and are easy to create and understand, but require interpretation of the CAD tools. Besides, none of these models can deal with detailed FPGA architecture design efficiently. Considerable modifications to both models and CAD tools will be needed.

In this paper, we propose a flexible architecture model which can describe the detailed architectures of all types of heterogeneous hard blocks in an FPGA, and an efficient algorithm of building FPGA resource graph. Based on these, we offer a feasible way to design detailed architectures of an FPGA. The rest of the paper proceeds as follows. Section II introduces the proposed architecture model. Section III presents the proposed building algorithm. Section IV reports on experimental results. Section V concludes this paper.

II. FPGA Architecture Description Model

In our FPGA architecture evaluation system, we use YAML Ain't Markup Language (YAML) file format to describe the architectural information of the FPGA. As XML is a pioneer in many domains, YAML is the result of lessons learned from XML and other technologies. But YAML has more advantages as follows:

(1) YAML is easily readable by humans.

(2) YAML is portable between programming languages.

(3) YAML matches the native data structures of agile languages.

(4) YAML has a consistent model to support generic tools.

(5) YAML supports one-pass processing.

(6) YAML is expressive and extensible.

(7) YAML is easy to implement and use.

In short, YAML tries to use a more agile way to complete the tasks expressed in XML, and always results in a more compact structure[10]. Fig. 1 is a YAML file example. It describes the array size of the FPGA architecture. YAML uses spaces to reflect the level of indentation. The key-value is delimited by a colon and space. More YAML syntax is discussed in YAML spec document[10].

```
FPG architecture:
  array size:
    nx: 48
    ny: 48
```

Fig. 1 A YAML file example

Our FPGA architecture description model is mainly comsisted of three parts:

(1) The first part describes the meta information of the FPGA, including FPGA array size, IO number, heterogeneous hard block distribution and so on.

(2) The second part describes the major information of different types of hard blocks, including IOs, Configurable Logic Blocks (CLBs), clock network resources, DSPs, memories, and other macrocells.

(3) The last part models the detailed switch patterns of the FPGA routing channels, including vertical and horizontal channels.

Our FPGA architecture description model is illustrated in Fig. 2. The details of each part are discussed below.

```
FPGA architecture:
  meta information:
    ···
  hard block:
    IO: ···
    CLB: ···
    clktree: ···
    #Other macro cells.
    ···
  channel:
    channel x: ···
    channel y: ···
```

Fig. 2 Overview of FPGA architecture description model

1. Meta information

The description model of the hard blocks deals with IOs, CLBs, clock network resources, DSPs, memories, and other macro cells. It is time-consuming and redundant to build a special model for each of these blocks. So we establish a general model to describe all kinds of hard blocks.

2. Hard block model

This part declares global FPGA architecture information, including FPGA array size, IO number (the number of I/Os that are contained per IO pad), heterogeneous hard block (such as memories) distribution, routing architecture type(single-driver or multi-driver). Fig. 3 shows an example.

```
meta information:
  array size:
    nx: 4
    ny: 4
  hard block distribution:
    memory:[{x: 1, y: 1~4}]
    IO number:
      top: 3
      bottom: 3
      left: 3
      right: 3
    driver type:
      opin to track: single driver
      track to track: single driver
```

Fig. 3 FPGA meta information description model

In order to simplify the process of building resource graph, we treat the hard block as a black box with certain inputs and outputs, as illustrated in Fig. 4. Inputs are connected from routing channels in switch blocks and outputs are connected to routing channels in switch blocks. In this paper, we use a general flow-based method to model different types of hard blocks. Inputs and outputs of hard block and switch block are modeled as nodes in the flow. Connections between them are modeled as edges in the flow. An example is illustrated in Fig. 5.

The greatest advantage of our architecture description model lies in that it can customize the detailed architectures, including the connection patterns of which the routing channels connected to the inputs and the outputs connected to the routing channels. Benefiting from this, we can also handle the case in which the inputs of LUTs in CLB are not logically equivalent. The only thing needs to do here is to model different LUT inputs as different edges of inputs.

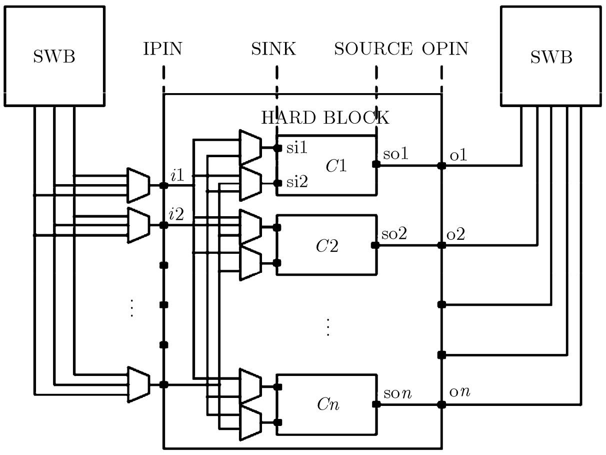

Fig. 4 FPGA hard block

```
route graph node:
  source:
    - name: so1
          edges: [o1]
  opin:
        - name: o1
          edges:
                eo: #tracks to east
                  - position: {deltaX: 0, deltaY: 0}
                    length: 4
                    track index: [0,2,5,7]
                wo: #tracks to west
                no: #tracks to north
                so: #tracks to south
  ipin:
        - name: i1
          edges: [si1,si2]
          track2ipin:
                ei: #tracks from east
                  - position: {deltaX: 0, deltaY: 0}
                    track index: [0,6,13,20]
                wi: #tracks from west
                si: #tracks from south
                ni: #tracks from north
  sink:
    - name: si1
```

Fig. 5 Hard blocks model

Through the model mentioned above, we are capable of modeling different heterogeneous hard blocks in a general way. And there is no need to modify the corresponding resource graph building algorithm.

3. Channel model

There are horizontal and vertical routing channels across the FPGA arrays. Connections between them are made through switch blocks. For every track in the channel with length L, there are L+1 possible locations to switch, as illustrated in Fig. 6.

We found in practice that even small differences in detailed routing architectures could result in discrepant performances; therefore, we model detailed architectures of switch pattern at every location for every track in this paper. Fig. 7 illustrates an example how we model routing channels. As illustrated in Fig. 7, a unique track index is assigned to each track in the channel. For tracks with same length and same switch patterns, we model their switch location, switch pattern, and the length of the tracks that they switch to.

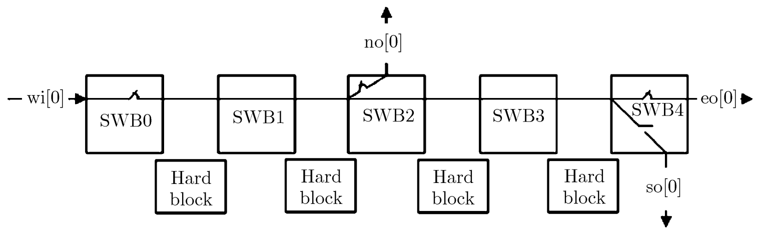

Fig. 6 Switch blocks distribution along channels

It is important to note that the channels can also connect to the input pins of hard blocks, or from the output pins of hard blocks, we have already described these information in previous hard block models. So far we have modeled all the FPGA resources, making it convenient to customize detailed architecture during FPGA architecture evaluation.

III. Resource Graph Building Algorithm

In this section, we present an algorithm that takes the architecture model as input and generates resource graph as output.

In our building algorithm, hard block pins and tracks in the channels are abstracted as nodes, and source and sink nodes are added for each logically equivalent group of pins. Building resource graph is a process of linking these nodes together.

The building process is divided into the following 3 steps.

```
channel x:
  width: 40
  track type: #tracks with different length
            - length: 4
              amount: 40
              connection:
                        wi: #connection of tracks from west
                          - name: wi0
                            edges:
                                     eo:
                                       - position: {deltaX: 4,deltaY: 0}
                                         track2track:
                                           - length: 4
                                             trackindex: [0]
                                     no:
                                       - position: {fdeltaX: 2,deltaY: 0}
                                         track2track:
                                           - length: 4
                                             trackindex: [0]
                                     so:
                                       - position: {deltaX: 4,deltaY: 0}
                                         track2track:
                                           - length:4
                                             track index: [0]
                        ei: #connection of tracks from east
```

Fig. 7 Channel model

1. Extract FPGA architecture information

This step mainly "translates" the FPGA architecture model into a structure that the computer can recognize and deal with.

Firstly, the syntax correctness of the YAML description file is verified. Then the FPGA architecture extraction work is performed, including parsing the YAML file (there are some open source YAML parsing libraries on the internet), and extracting relevant information from the YAML nodes tree. This information is put together into the final FPGA model. Finally, the correctness of the generated model is checked under FPGA architecture design rules.

2. Building resource graph

There are two steps in the graph building process: generating the nodes of resource graph, and establishing the connection pattern between these nodes. Pseudo-code of generating the resource nodes of FPGA is shown in Algorithm 1.

Algorithm 1 Generate resource graph nodes

```
Require: ($nx$, $ny$) is FPGA array size and hard_block($i$, $j$) is the
  type of hard blocks on location ($i$, $j$).
Ensure: rr_nodes stores resource graph nodes.
  for row = 0 to $nx$ + 1 do
    for col = 0 to $ny$ + 1 do
      if hard_block(row, col) is legal hard block then
        /* For each hard block, add source, opin, ipin, sink
        nodes to rr_nodes. */
        AddSourceNodes( );
        AddOpinNodes( );
        AddIpinNodes( );
        AddSinkNodes( );
      end if
      if hard_block(row, col) has switch block then
        /* Add tracks of vertical and horizontal channel
        which is adjacent to the hard block into rr_nodes. */
        AddTracksNodes( );
      end if
    end for
  end for
```

Next, connection pattern between each node are added as edges in the resource graph. The pseudocode is shown in Algorithm 2.

In our system, the obtained critical path delay is characterized by the routine of the routing segments. To perform timing-driven routing and timing analysis, timing parameters, such as the delay between tracks with different lengths, the delay from IO to registers and the delay matrix of LUT, *etc.*, are needed and should be added into our architecture description model to obtain timing performance.

Algorithm 2 Build the connection pattern of nodes

```
Require: rr_nodes stores all resource graph nodes.
Ensure: All nodes of rr_nodes have connection edges.
  for row = 0 to $nx$ + 1 do
    for col = 0 to $ny$ + 1 do
      if hard_block(row, col) is legal hard block then
        /* To each hard block, building the nodes connection
        of source, opin, ipin, sink. */
        /* Source nodes merely connect to their opins. */
        BuildSourceNodesConnection( );
        /* Opin nodes connect to tracks and input pins of
         adjacent hard blocks. */
        BuildOpinNodesConnection( );
        /* Ipin nodes connect to sink opins. */
        BuildIpinNodesConnection( );
        /* Sink nodes have no edges. */
        BuildSinkNodesConnection( );
      end if
      if hard_block(row, col) has switch block then
        /* Tracks connect to other tracks through switch
         blocks. Tracks also connect input pins of hard block. */
        BuildTrackToIpin( );
        BuildTrackToTrack( );
      end if
    end for
  end for
```

3. Legality checking of resource graph

Finally, to verify whether the resource graph is correct, we check the generated resource graph according to FPGA design rules.

IV. Experimental Results

To illustrate the flexibility of our detailed FPGA architecture description model and the efficiency of our algorithm of building resource graph, a set of evaluation experiments have been carried on in the FPGA architecture evaluation system that we developed.

Several detailed architectures are evaluated through our system. In all these architectures, CLBs contain ten 4-input LUTs, the routing architecture employs length 4 tracks with single-driver manner, and switch block flexibility (Fs)[1] is set to 3. A base architecture is defined with a uniform distribution as connection block pattern and Wilton type [11,12] as switch block pattern exists at location 0, 1, and 4. We map the 20 largest MCNC benchmark circuits into the architectures. Push count, pop count, and total wirelength are selected as the evaluation metric. Push count and pop count are measurements of router effort[13] and total wirelength is a reflection of timing performance.

Detailed architectures in switch blocks and connection blocks are modified and evaluated against the base architecture. Through the experimental results, we validate the effectiveness of our architecture description model and resource graph building algorithm. The experiments will be described in details below.

1. Impact of switch pattern and location of switch blocks

In this section, we observe how the switch pattern and location of switch blocks affect FPGA performances. In VPR, switch pattern must be the same at every switch location, and once there is a switch, one track must switch to all the other three directions. In our architecture description model, switch pattern and switch locations can be customized. To verify whether these detailed architectures in switch blocks will affect FPGA performances, we use the following three different architectures:

(1) arch 1 is an architecture with Wilton switch pattern exists at location 1 and Disjoint[11, 12] switch exists at locations 0 and 4.

(2) arch 2 is an architecture with Disjoint switch pattern exists at location 1 and Wilton switch exists at locations 0 and 4.

(3) arch 3 is an architecture with Wilton switch block pattern exists at locations 0, 1, and 4 while Fs is set to 4.

It is obvious that there are very little differences in detailed architectures between these three architectures but big difference from the base architecture. Figs. 8~10 show the normalized experimental results for these three architectures compared to the basic architecture. From Figs. 8~10, we can find that all these three architectures are superior to the base architecture in router effort and total wirelength. Block pins are 4.23%~ 8.08% reductions on average in push count and 7.73%~ 9.34% reductions on average in pop count, indicating better routability over the base architecture. And there are 13.88%~16.36% reductions on average in wirelength, indicating better timing performances over the base architecture. The experimental results are in line with Ref. [14], which show that the disjoint switch block lead to faster circuits and better area-efficiency than the Wilton switch block. Furthermore, these results show us how great the detailed architecture affects the overall performances, which verifies the necessity and effectiveness of our FPGA architecture model.

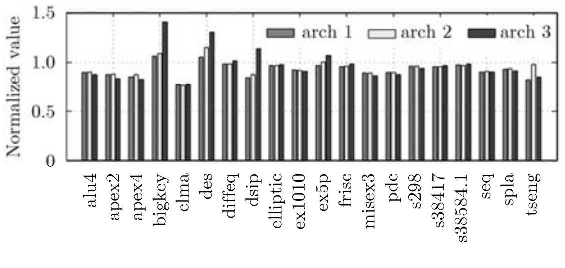

Fig. 8 Normalized push count

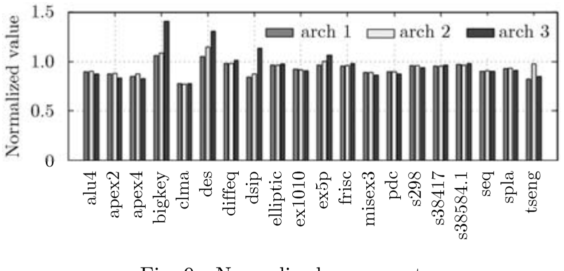

Fig. 9 Normalized pop count

2. Impact of connection pattern of connection blocks

In VPR, input pins and the output pins of hard blocks are connected to the channel through a uniform connection pattern as default. The Fc parameter sets the number of tracks to which each logic block input pin connects in each channel bordering the pin[1]. In our architecture model, one can define the detailed connection pattern. To check how these detailed architectures in connection blocks affect FPGA performances, Fc is set to 15% of the channel width (the number of tracks in channel) in our experiments and we modify the connection pattern into a Gaussian distribution (arch 4) and random distribution (arch 5) separately. The normalized experimental results compared to the base architecture are shown in Figs. 11~13.

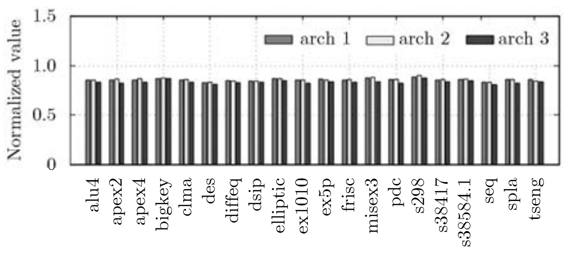

Fig. 10 Normalized wirelength

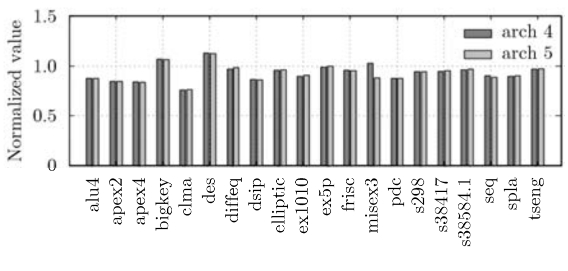

Fig. 11 Normalized push count

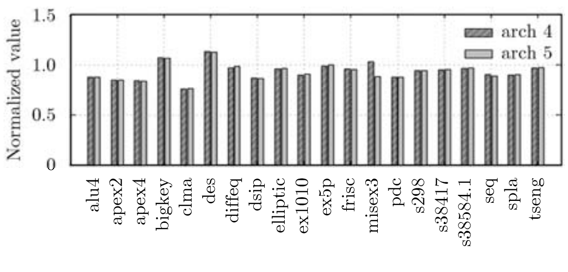

Fig. 12 Normalized pop count

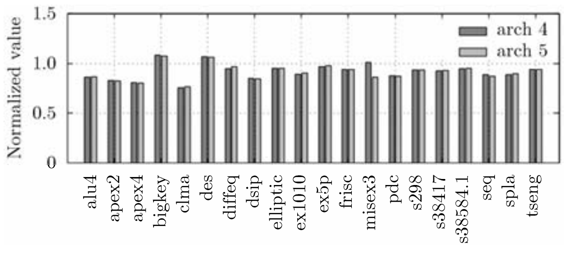

Fig. 13 Normalized wirelength

From Figs. 11~13, we can find that random connection pattern performs better than Gaussian connection pattern in push count and pop count, indicating better routability. Gaussian connection pattern has shorter wirelength than random connection pattern, indicating better timing performance. Both architectures are superior to the base architecture, with 7.04%~7.64% reductions on average in push count, 8.51%~9.08% reductions on average in pop count, and 15.46%~15.52% reductions on average in wirelength. These results show us again why it is important to model detailed architectures.

V. Conclusions and Future Work

In this paper, we propose a flexible FPGA architecture model, which can describe detailed architecture information of heterogeneous hard blocks and routing channels. Based on this model, we present a resource graph building algorithm. The contributions of our work are as follows:

(1) The proposed FPGA architecture description model gives a consistent description of different heterogeneous hard block, making it easy to process through the unified algorithm and facilitate the addition of new heterogeneous hard block without modifying the software source code.

(2) The proposed model can be used to describe detailed architectures, which has been proved to have great significance in the optimum design of FPGA architecture.

There is much more work remaining to support more features, such as critical path delay computing, area modeling, power modeling and so on. And we believe that this paper will provide an alternative for the optimum design of FPGA architecture, and offer a feasible way for FPGA architecture design to advance from theory to practice.

References

[1] V. Betz, J. Rose, and A. Marquardt. Architecture and CAD for Deep-submicron FPGAs. Kluwer Academic Publishers, 1999.

[2] J. Luu, J. H. Anderson, and J. S. Rose. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 2011,

227–236.

[3] Altera. Stratix v device family overview. http://www.altera.com/literature/hb/stratix-iv/stx5 siv51001.pdf, 2014.

[4] Xilinx. Xilinx ds180 7 series fpgas overview. http://www.xilinx.com/support/documentation/data sheets/ds180 7Series Overview.pdf, 2014.

[5] V. Betz and J. Rose. Automatic generation of FPGA routing architectures from high-level descriptions. Proceedings of the Eighth ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 2000, 175–184.

[6] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. Field-Programmable Logic and Applications, Springer, 1997, 213–222.

[7] V. Betz. VPR and t-vpack users manual. http://www, eeeg. toronto, edu/-janders/ ece1387/e1/manual 430. pdf, 2000.

[8] J. Rose, J. Luu, C. W. Yu, *et al.*. The vtr project: architecture and CAD for FPGAs from verilog to routing. Proceedings of 2012 International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 2012, 77–86.

[9] D. G. Paladino. Academic clustering and placement tools for modern Field-Programmable Gate Array architectures. [Ph.D. Dissertation], University of Toronto, 2008.

[10] O. Ben-Kiki, C. Evans, and B. Ingerson. Yaml Aint Markup Language version 1.2, YAML.org, Technical Report, September 2009.

[11] M. I. Masud and S. J. Wilton. A new switch block for segmented FPGAs. Field Programmable Logic and Applications, Springer, 1999, 274–281.

[12] Guy G. Lemieux and D. M. Lewis. Analytical framework for switch block design. LNCS 2438, 2002, 122–131.

[13] N. Shah and J. Rose. On the difficulty of pin-to-wire routing in FPGAs. 22nd International Conference on Field Programmable Logic and Applications (FPL). Oslo, Norway, 2012, 83–90.

[14] V. Betz. Architecture and CAD for speed and area optimizations of FPGAs. [Ph.D. Dissertation], University of Toronto, 1998.