FPGA-on-FPGA emulation

Pim van Leeuwen

Monday 3rd February, 2020

Abstract

FPGAs allow reconfiguration of its logic at any point after production. The result is that they are effective at prototyping application-specific integrated circuits, updating the internal logic while in the field and at low-cost low-quantity use cases. To optimise these processes, it is crucial to properly educate engineers in the implementation of FPGA programs and the FPGA compilation process. Traditional FPGA programming pipelines involve a computationally expensive (NP-hard) place & route process that slows down iterations of FPGA programs and hinders the educational process. We propose a virtual environment in which place & route is performed manually in which the student learns about the intricacies of place & route and in which compilation is linear. To this end, we require emulation of a virtual FPGA on a physical, concrete FPGA. In the proposed research, we find such an emulation using an algorithm that solves a variant of subgraph isomorphism. This algorithm aims to find emulations in as many cases as possible and to exploit the hierarchy of FPGAs to speed up computation. The expected result is a software package that computes emulators which each output a program for a concrete FPGA provided with a program for a virtual FPGA.

Contents

1 Introduction

2 Background

2.1 Field Programmable Gate Arrays

2.1.1 Lookup tables

2.1.2 Registers

2.1.3 Logic Cells

2.1.4 Routing

2.1.5 Compilation

2.2 Simulation versus Emulation

2.3 Path subgraph isomorphism

3 Models

3.1 FPGA

Our algorithm will generate an emulation using path subgraph isomorphism. To this end, we will model FPGA designs as graphs. For this purpose, let us define the type $hierarchygraph : (V, E, L, H)$ where V is a set of vertices, $E \subseteq (V \times V)$ is a set of undirected edges, L is a vertex labeling function $V \to \lambda$ where λ is a finite set of labels and H describes the hierarchy with a relation $V \to hierarchygraph$. We restrict ourselves to graphs without loops, i.e. $\nexists v \in V.(v, v) \in E$. Let $model(B)$ be the $hierarchygraph$ model of (a subset of) an FPGA layout B. Then:

- Whenever B has a connection to external hardware (e.g. a pin) with an optional identifier k where order is relevant, $model(B)$ has an edge to a distinct vertex v and $L(v) = \text{“}port\_out_k\text{”}$.
- Each component that is one step lower than B in the FPGA hierarchy (e.g. CLBs) corresponds to a distinct vertex v such that $L(v) = \text{“}component\text{”}$ and with a set of vertices S with optional identifiers such that $\forall s \in S.L(s) = \text{“}port\_in_{id(s)}\text{”} \wedge (s, v) \in E$. Furthermore, the contents of this component are recursively defined in $H(v)$.
- Whenever B has a routing switch, $model(B)$ has a distinct vertex v and $L(v) = \text{“}switch\text{”}$.
- Whenever B has a mux with n wires for each input and for its output, $model(B)$ has distinct vertices $\{v_{left,1} \dots v_{left,n}\} \cup \{v_{right,1} \dots v_{right,n}\} \cup \{v_{out,1} \dots v_{out,n}\} \cup \{v_{select}\}$ where:

  $\forall v_{left,i} \in \{v_{left,1} \dots v_{left,n}\} .L(v_{left,i}) = \text{“}mux\_left_i\text{”} \wedge (v_{left,i}, v_{select}) \in E,$
  $\forall v_{right,i} \in \{v_{right,1} \dots v_{right,n}\} .L(v_{right,i}) = \text{“}mux\_right_i\text{”} \wedge (v_{right,i}, v_{select}) \in E,$
  $\forall v_{out,i} \in \{v_{out,1} \dots v_{out,n}\} .L(v_{out,i}) = \text{“}mux\_out_i\text{”} \wedge (v_{out,i}, v_{select}) \in E$ and
  $L(v_{select}) = \text{“}mux\_select\text{”}$

  Whenever something is connected to input or output of the mux in B, $model(B)$ has an edge to the corresponding vertex $v \in \{v_{left,1} \dots v_{left,n}\} \cup \{v_{right,1} \dots v_{right,n}\} \cup \{v_{out,1} \dots v_{out,n}\}$. Furthermore, if B has a wire to the selection input, $model(B)$ has an edge to the vertex v_{select}.
- Each LUT that has n inputs and m outputs corresponds to unique vertices v_{in} and v_{out} such that $L(v_{in}) = \text{“}lut_{in}\text{”}$, $L(v_{out}) = \text{“}lut_{out}\text{”}$ and $(v_{in}, v_{out}) \in E$. Wires to the input of the LUT correspond with edges to v_{in} whilst wires from the output of the LUT correspond with edges from v_{out}.

  change to $model(B)$
- Each register that has n inputs corresponds to:

  $\{v_{in,1} \dots v_{in,n}\} \cup \{v_{out,1} \dots v_{out,n}\} \cup \{v_{set}, v_{resetsync}, v_{resetasync}\}$ where
  $\forall v_{in,i} \in \{v_{in,1} \dots v_{in,n}\} .L(v_{in,i}) = \text{“}register\_in_i\text{”} \wedge (v_{in,i}, v_{set}) \in E,$
  $\forall v_{out,i} \in \{v_{out,1} \dots v_{out,n}\} .L(v_{out,i}) = \text{“}register\_out_i\text{”} \wedge (v_{out,i}, v_{set}) \in E,$

  change to $model(B)$

$L(v_{set}) =$ "$register\_set$",

$L(v_{resetsync}) =$ "$register\_reset\_sync$" $\wedge(v_{set}, v_{resetsync}) \in E$ and
$L(v_{resetasync}) =$ "$register\_resFet\_async$" $\wedge(v_{set}, v_{resetasync}) \in E$.

A connection to an input of the register at wire position x corresponds with an edge to $v_{in,x}$. Similarly, a connection to an output of the register at wire position x corresponds with an edge to $v_{out,x}$. Connections to the set-, synchronous reset and asynchronous reset gates in the FPGA correspond to edges to the respective vertex in $\{v_{set}, v_{resetsync}, v_{resetasync}\}$.

change to $model(B)$

change to $model(B)$

change to $model(B)$

change to $model(B)$

- Each connection by wire corresponds to an edge.

3.2 FPGA program

4 Preliminaries

5 Algorithm

5.1 State space storage

5.2 State space exploration

5.3 Hierarchy usage

5.4 Defining f

6 Performance experiments

7 Software design

7.1 Architecture

7.2 Manual

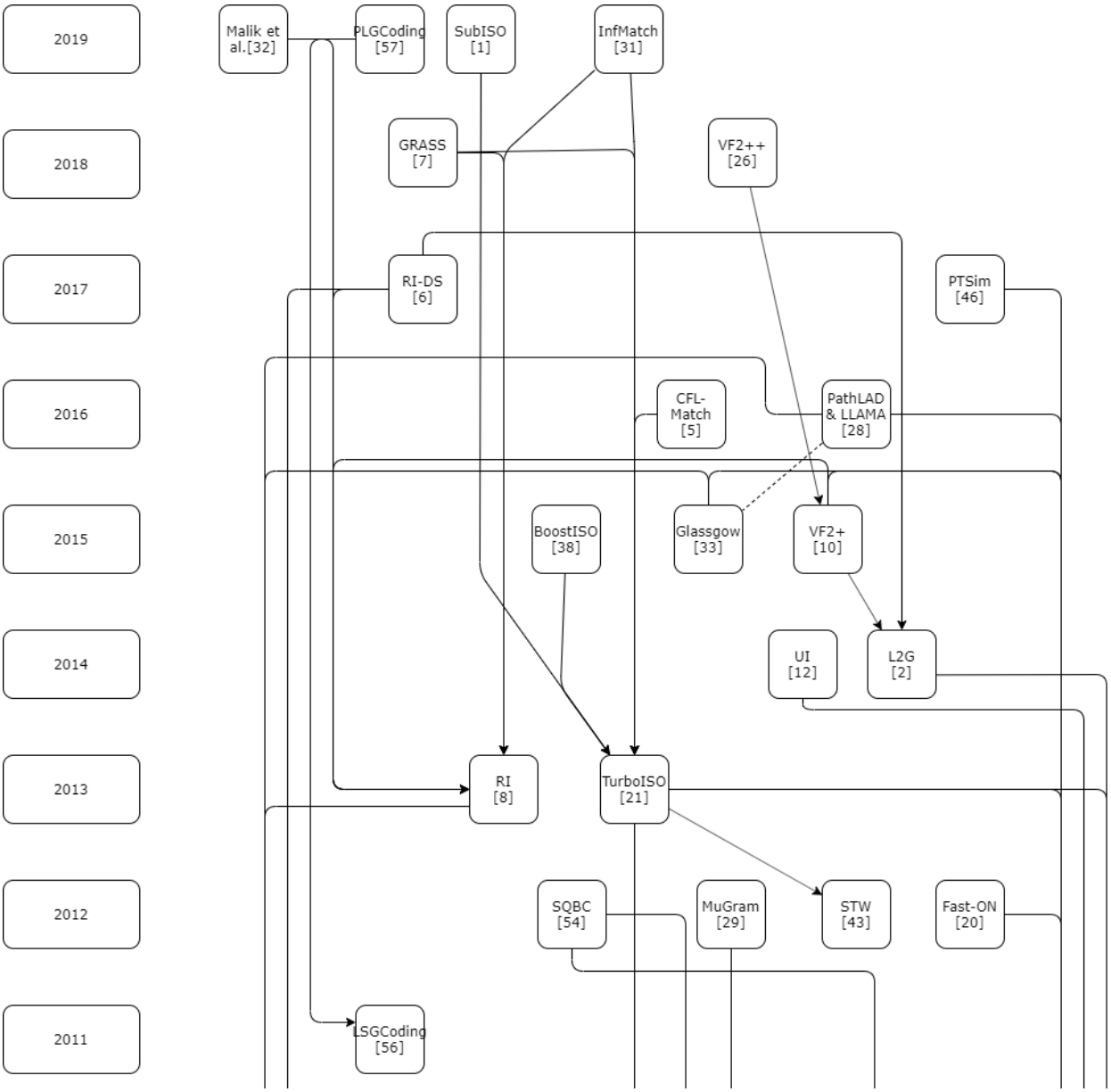
8 Conclusion

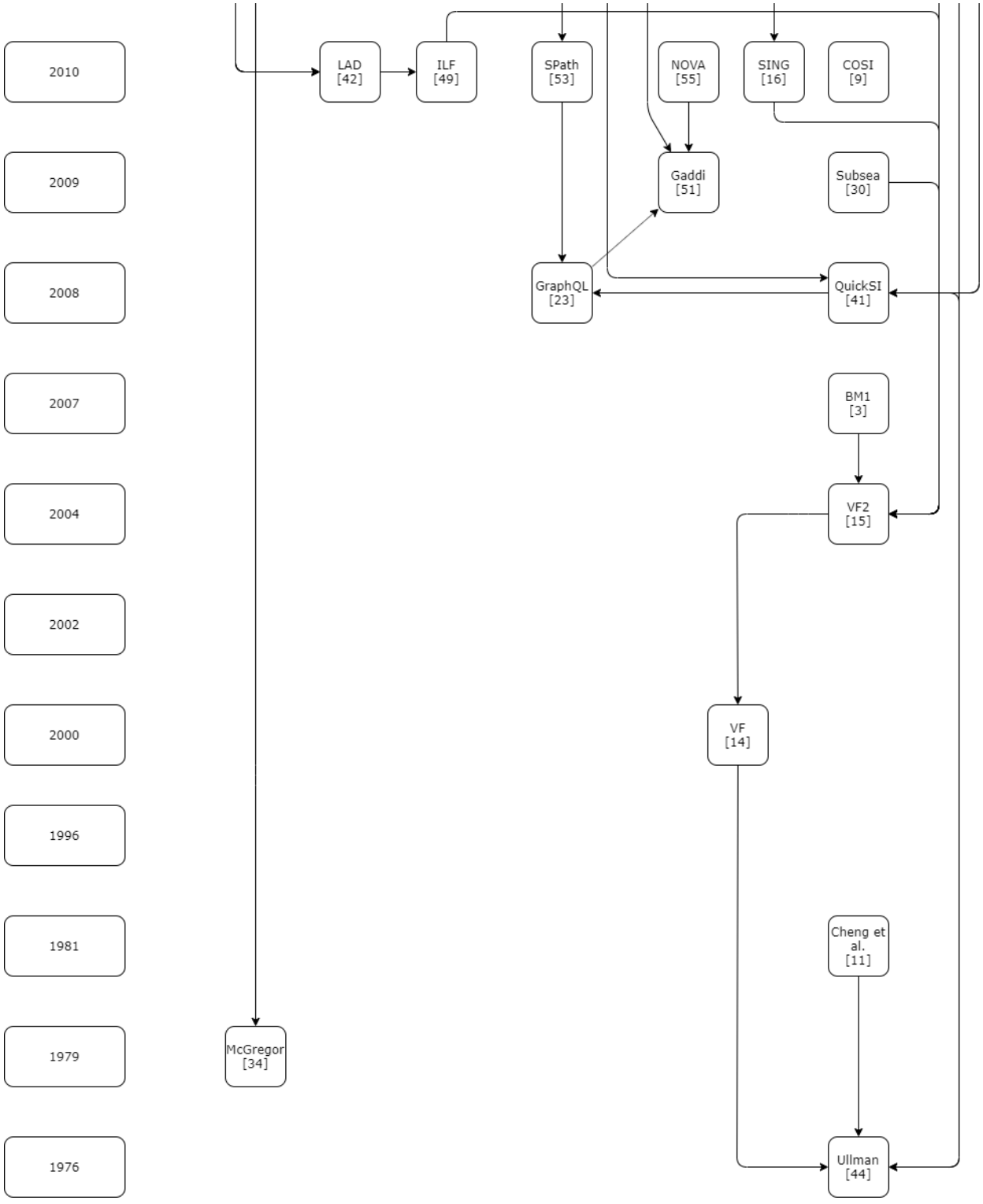
9 Discussion

10 Future Research

Appendices

A History of subgraph isomorphism algorithms




2010
2009
2008
2007
2004
2002
2000
1996
1981
1979
1976
LAD [42]
ILF [49]
SPath [53]
NOVA [55]
SING [16]
COSI [9]
Gaddi [51]
Subsea [30]
GraphQL [23]
QuickSI [41]
BM1 [3]
VF2 [15]
VF [14]
Cheng et al. [11]
McGregor [34]
Ullman [44]