FPGA-on-FPGA emulation

Pim van Leeuwen

Thursday 6th 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 $hierarchygraph$.

Definition 3.1. A $hierarchygraph$ is a structure $\subseteq (V, E, L, H, C)$ where:

- V is a set of vertices.
- $E \subseteq (V \times V)$ is a set of undirected edges without loops, i.e. $\nexists v \in V.(v, v) \in E$.
- L is a vertex multilabeling function $V \rightarrow P(\lambda)$ where $P(\lambda)$ is the power set of a finite set of labels λ.
- $H \subseteq (V \times G)$ where G is the set of all $hierarchygraphs$. This describes the hierarchy of an FPGA.
- $C \subseteq (V \times V_H)$ where V_H is the union of the vertex sets of $hierarchygraphs$ in H. This allows us to separate different wires coming out of a hierarchical component of the FPGA by linking them with vertices in the $hierarchygraph$ definition of that component.

Definition 3.2. Let $model(B)$ be a $hierarchygraph$ model of (a subset of) an FPGA layout B such that:

- For each pin in B, $model(B)$ has a vertex v such that $L(v) = \{\text{“}pin\text{”}\}$.
- For each routing switch in B, $model(B)$ has a vertex v such that $L(v) = \{\text{“}switch\text{”}\}$.
- For each component in B that is one step lower than B in the FPGA hierarchy (e.g. CLBs), $model(B)$ has a distinct vertex v such that $L(v) = \{\text{“}component\text{”}\}$ and a set of vertices S such that $\forall s \in S.L(s) = \{\text{“}port\text{”}\} \wedge (s, v) \in E$. Furthermore, the contents of this component are recursively defined in $H(v)$ such that ports are linked:

  $\forall s \in S.\exists w \in (H(v).V).(s, w) \in C$.

  Furthermore, this link is unique, i.e.:

  $\forall w_1, w_2 \in (H(w).V).\forall s \in S.((s, w_1) \in C \wedge (s, w_2) \in C) \rightarrow w_1 = w_2$
- For each mux with n wires for each input and for its output in B, $model(B)$ has a distinct vertex v such that $L(v) = \{\text{“}component\text{”}\}$ and a set of vertices S such that $|S| = 3n + 1$.

  $\forall s_i \in S.L(s_i) = \{\text{“}port\text{”}\} \wedge (v, s_i) \in E$

  $H(v) = (V_{mux}, E_{mux}, L_{mux}, \emptyset, \emptyset)$ where:

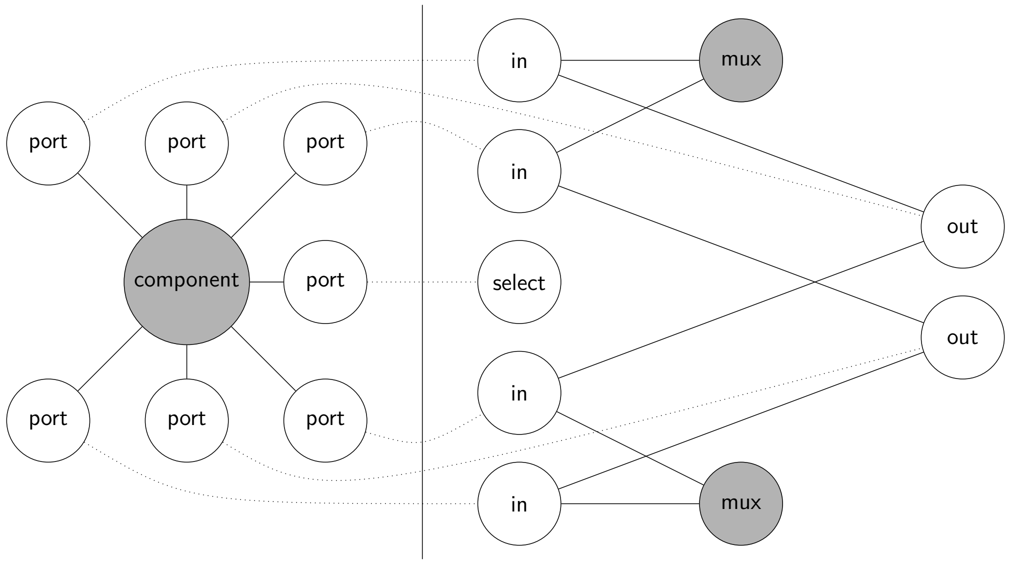


Figure 1: How a 2-wire mux is represented in $model(B)$ (left) in terms of vertex v with $L(v) =$ "$component$", port vertices and edges, and the $hierarchygraph$ $H(v)$ (right) storing the internal details of the mux. Dashed lines indicate the relation C that links the vertices representing outgoing wires. Vertex labels are shown as text.

$$\begin{array}{lllll}
V_{mux} = & \{v_{left,1} \dots v_{left,n}\} \cup \{v_{right,1} \dots v_{right,n}\} \cup \{v_{out,1} \dots v_{out,n}\} \cup \{v_{left}, v_{right}\} \\
E_{mux} = & \{(v_{left,i}, v_{left}) & : v_{left,i} & \in \{v_{left,1} \dots v_{left,n}\}\} & \cup \\
& \{(v_{right,i}, v_{right}) & : v_{right,i} & \in \{v_{right,1} \dots v_{right,n}\}\} & \cup \\
& \{(v_{left,i}, v_{out,i}) & : v_{left,i} & \in \{v_{left,1} \dots v_{left,n}\}\} & \cup \\
& \{(v_{right,i}, v_{out,i}) & : v_{right,i} & \in \{v_{right,1} \dots v_{right,n}\}\}
\end{array}$$

$$\begin{array}{lllll}
L_{mux} = & \{(v_{left,i} \ , \{\text{“}in\text{”}\}) & : v_{left,i} & \in \{v_{left,1} \dots v_{left,n}\}\} & \cup \\
& \{(v_{right,i} \ , \{\text{“}in\text{”}\}) & : v_{right,i} & \in \{v_{right,1} \dots v_{right,n}\}\} & \cup \\
& \{(v_{out,i} \ , \{\text{“}out\text{”}\}) & : v_{out,i} & \in \{v_{out,1} \dots v_{out,n}\}\} & \cup \\
& \{(v_{left} \ , \{\text{“}mux\text{”}\}), (v_{right}, \{\text{“}mux\text{”}\})\}
\end{array}$$

Lastly, the relation C contains a link from nodes in the FPGA $hierarchygraph$ to nodes in the mux $hierarchygraph$, i.e. $\forall s \in S.\exists w \in V_{mux}.(s, w) \in C$. The model of a mux is illustrated in Figure 1.

- For each LUT in B with n inputs and m outputs, $model(B)$ has a distinct vertex v such that $L(v) = \{\text{“}component\text{”}\}$ and a set of vertices S such that $|S| = n + m$.

  $\forall s \in S.L(s) = \{\text{“}port\text{”} : i \in \{1 \dots n\}\} \wedge (v, s) \in E$

  $H(v) = (V_{lut}, E_{lut}, L_{lut}, \emptyset, \emptyset)$ where:

  $V_{lut} = \{v_{in,1} \dots v_{in,n}\} \cup \{v_{out,1} \dots v_{out,m}\} \cup \{v_{left}, v_{right}\}$

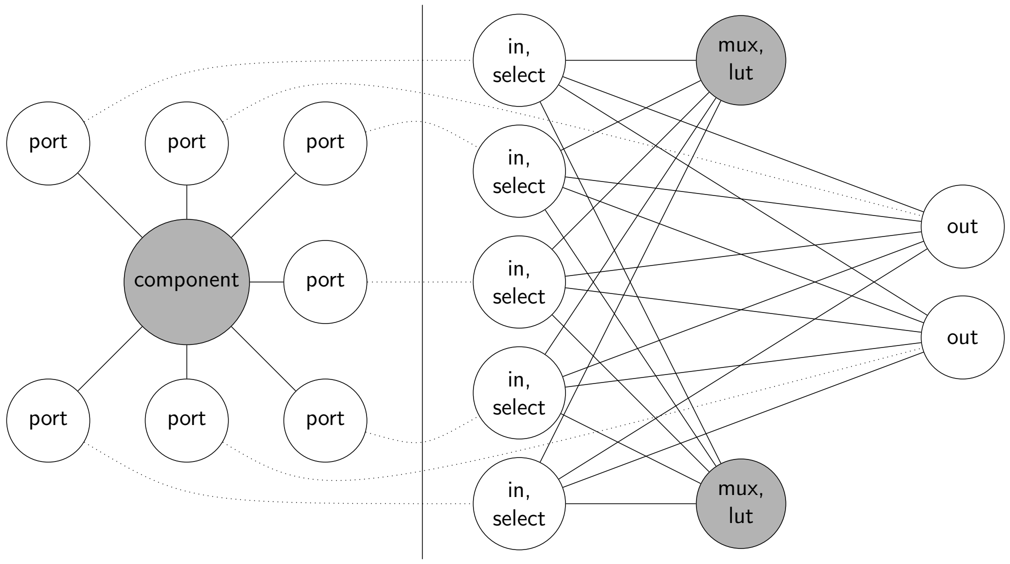


Figure 2: How a 5-input-2-output lut is represented in $model(B)$ (left) in terms of vertex v with $L(v) =$ "$component$", port vertices and edges, and the $hierarchygraph$ $H(v)$ (right) storing the internal details of the lut. This $hierarchygraph$ representation contains subgraphs of muxes that this lut can emulate. Dashed lines indicate the relation C that links the vertices representing outgoing wires. Vertex labels are shown as text.

$$
\begin{aligned}
E_{lut} = \quad & \{(v_{in,i}, v_{left}) \quad : v_{in,i} \in \{v_{in,1} \dots v_{in,n}\}\} \cup \\
& \{(v_{in,i}, v_{right}) \quad : v_{in,i} \in \{v_{in,1} \dots v_{in,n}\}\} \cup \\
& \{(v_{in,i}, v_{out,j}) \quad : v_{in,i} \in \{v_{in,1} \dots v_{in,n}\}, v_{out,j} \in \{v_{out,1} \dots v_{out,m}\}\}
\end{aligned}
$$

$$
\begin{aligned}
L_{lut} = \quad & \{(v_{in,i}, \{\text{“}in\text{”}, \text{“}select\text{”}\}) \quad : v_{in,i} \in \{v_{in,1} \dots v_{in,n}\}\} \quad \cup \\
& \{(v_{out,i}, \{\text{“}out\text{”}\}) \quad : v_{out,i} \in \{v_{out,1} \dots v_{out,m}\}\} \quad \cup \\
& \{(v_{left}, \{\text{“}in\text{”}, \text{“}select\text{”}\})\} \quad \cup \\
& \{(v_{right}, \{\text{“}in\text{”}, \text{“}select\text{”}\})\}
\end{aligned}
$$

Lastly, the relation C contains a link from nodes in the FPGA $hierarchygraph$ to nodes in the lut $hierarchygraph$, i.e. $\forall s \in S. \exists w \in V_{lut}.(s, w) \in C$. The model of a lut is illustrated in Figure 2.

- For each register in B with n inputs, $model(B)$ has a distinct vertex v such that $L(v) = \{\text{“}component\text{”}\}$ and a set of vertices S such that $|S| = 2n + 2$ if the register has synchronous- or asynchronous reset capabilities, $|S| = 2n+3$ if it has both or $|S| = 2n+1$ if it has neither.

  $\forall s \in S. L(s) = \{\text{“}port\text{”}\} \wedge (v, s) \in E$

  $H(v) = (V_{reg}, E_{reg}, L_{reg}, \emptyset, \emptyset)$ where:

  $V_{reg} = \{v_{in,1} \dots v_{in,n}\} \cup \{v_{out,1} \dots v_{out,n}\} \cup \{v_{set}, v_{sync}{}^{1}, v_{async}{}^{1}\}$

[1] These vertices, their edges, their labels and mapping in C are omitted if the register is incapable of synchronous- or asynchronous reset, respectively.

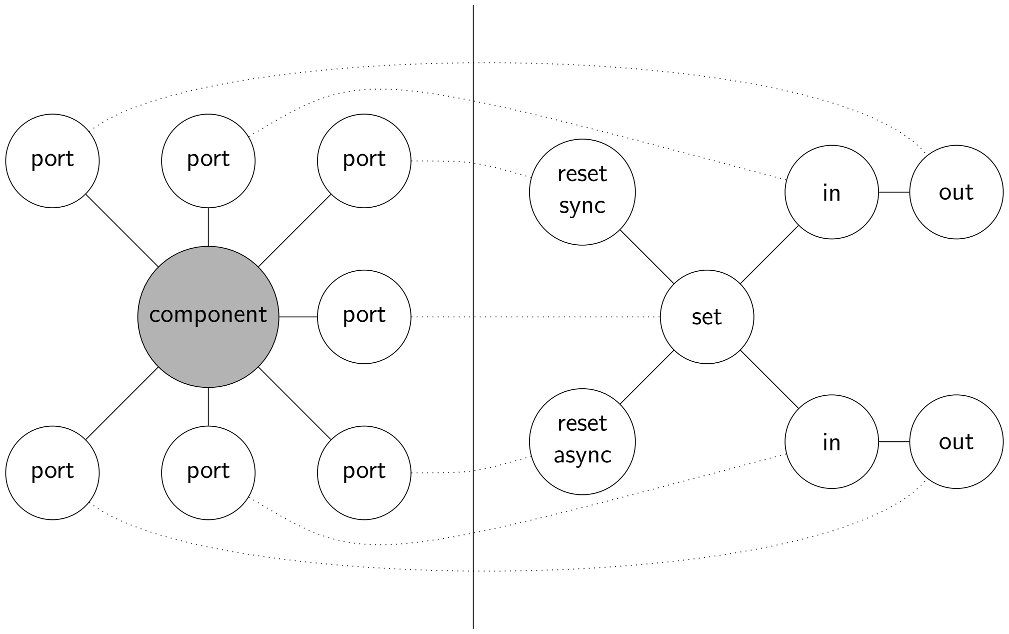


Figure 3: How a 2-input register is represented in $model(B)$ (left) in terms of vertex v with $L(v) =$ "$component$", port vertices and edges, and the $hierarchygraph$ $H(v)$ (right) storing the internal details of the register. Dashed lines indicate the relation C that links the vertices representing outgoing wires. Vertex labels are shown as text.

$$
\begin{aligned}
E_{reg} = \quad & \{(v_{in,i}, v_{out,i}) && : v_{in,i} \in \{v_{in,1} \dots v_{in,n}\}\}\cup \\
& \{(v_{in,i}, v_{set}) && : v_{in,i} \in \{v_{in,1} \dots v_{in,n}\}\}\cup \\
& \{(v_{sync}{}^{1}, v_{set}), (v_{async}{}^{1}, v_{set})\}
\end{aligned}
$$

$$
\begin{aligned}
L_{reg} = \quad & \{(v_{in,i}, \text{“}in\text{”}) && : v_{in,i} \in \{v_{in,1} \dots v_{in,n}\}\}\cup \\
& \{(v_{out,i}, \text{“}out\text{”}) && : v_{out,i} \in \{v_{out,1} \dots v_{out,n}\}\}\cup \\
& \{(v_{set}, \text{“}set\text{”}), (v_{sync}{}^{1}, \text{“}sync\text{”}), (v_{async}{}^{1}, \text{“}async\text{”})\}
\end{aligned}
$$

Lastly, the relation C contains a link from nodes in the FPGA $hierarchygraph$ to nodes in the register $hierarchygraph$, i.e. $\forall s \in S.\exists w \in V_{reg}.(s, w) \in C$. The model of a mux is illustrated in Figure 1.

- Each connection by wire in B corresponds to an edge in $model(B)$. For muxes, luts, registers and hierarchical components this means an edge to the "port"-vertex v for which $C(v)$ represents the appropriate input/output of the component.
- $model(B)$ contains nothing else.

3.2 FPGA program

Let $model(B)$ be the hierarchygraph of an FPGA layout B. We then define the interpretation of $model(B)$ to

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

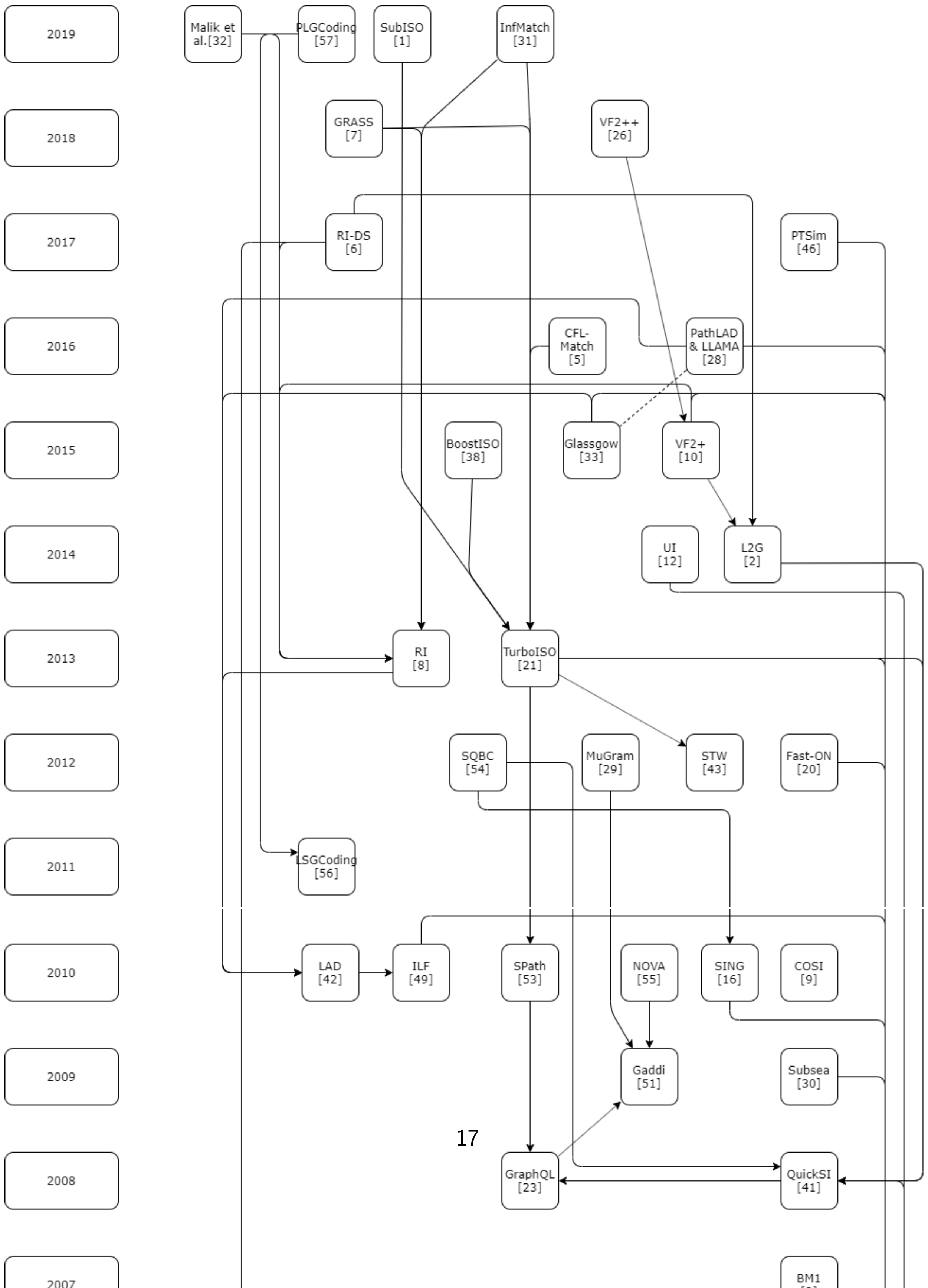
8 Conclusion

9 Discussion

10 Future Research

Appendices

A History of subgraph isomorphism algorithms



2019
Malik et al.[32]
PLGCoding [57]
SubISO [1]
InfMatch [31]
2018
GRASS [7]
VF2++ [26]
2017
RI-DS [6]
PTSim [46]
2016
CFL-Match [5]
PathLAD & LLAMA [28]
2015
BoostISO [38]
Glassgow [33]
VF2+ [10]
2014
UI [12]
L2G [2]
2013
RI [8]
TurboISO [21]
2012
SQBC [54]
MuGram [29]
STW [43]
Fast-ON [20]
2011
LSGCoding [56]
2010
LAD [42]
ILF [49]
SPath [53]
NOVA [55]
SING [16]
COSI [9]
2009
Gaddi [51]
Subsea [30]
2008
GraphQL [23]
QuickSI [41]
2007
BM1