Routability-Driven FPGA Placement Contest

Stephen Yang, Aman Gayasen, Chandra Mulpuri, Sainath Reddy, Rajat Aggarwal
Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124
stepheny,amang,chandim,sainath,rajata@xilinx.com

ABSTRACT

The advances of FPGA technology and increasing size of FPGA designs pose great challenges on FPGA design tools. Deep research on FPGA physical design problems is paramount to improve industrial tools. This contest is the first ISPD contest on FPGA CAD tools. Routability driven FPGA placement, in context of large designs modern FPGA architecture, is one of the best topics to start the effort.

KEYWORDS

FPGA; Placement; Routability; Congestion; Contest

1. INTRODUCTION

FPGA (Field Programmable Gate Array) placement is a classic problem in Electronic Design Automation field. It is one of the key steps in FPGA design flow, directly impacting the completion of the design flow and the performance of the resulting FPGA. With the advances of FPGA hardware technology and wide spreading applications, there are more and more challenges imposed on FPGA placement problem [1].

In year 2005, ISPD hold the first placement contest. 10 teams from worldwide universities research groups participated and competed for the prizes. Since then ISPD has hold 11 different contests, topics covering placement, global/detail routing, gate sizing and clock tree synthesis. These contests greatly stimulated academic research activities. A number of high-quality academic tools with novel ideas and sophisticated algorithms have presented in the past decade.

Yet ISPD has never done any CAD contest on FPGA related problems. Given the difficult and unique challenges, FPGA placement problem serves as a perfect topic for ISPD contest. This contest attracts both classical FPGA placement research groups, as well as standard-cell/mixed-size placement research groups. The former groups have deep understanding on FPGA architecture and FPGA specific algorithms like packing, timing-driven placement and graph routing. The latter groups were used to face large scale placement problem, dealing with hundreds of thousands or even millions of movable objects. Attacking FPGA problem from two different angles can greatly move academic research forward and lead to effective and efficient FPGA oriented algorithms.

As the world's leading provider of FPGAs, Xilinx Inc. took the responsibility to co-organize this FPGA placement contest. The contest benchmarks are based on industry leading 20nm Virtex UltraScale architecture. The size of the benchmarks reflects the typical modern high-end FPGA designs. The well-defined contest evaluation metrics have the key elements of FPGA design tool: wirelength, routability and runtime are all considered. We believe that this first FPGA placement contest will serve as the beginning of the prosperous research of FPGA physical design, attracting more young talents into the challenging and exciting EDA field.

2. BACKGROUND

There are many challenges in modern FPGA placement problem.

First, multiple objectives need to be considered during FPGA placement. Traditional FPGA placer took total wirelength as the main cost function. Nowadays, FPGA placer has to optimize the following objectives: wirelength, congestion, timing, power, utilization etc. Optimizing all the objectives at the same time is very hard, yet modern FPGA designs do need all of them to stay competitive.

Second, FPGA resource constraints have been the biggest challenge for FPGA placement. Various resources including LUT, flip flop, block RAM, DSP, distributed RAM need to be placed at different sites on the device. Large unit resources like block RAM and DSP are discrete --- their available sites are scarce and are often far away from each other. Placer needs to handle the multi-resource problem in a smooth fashion to be able to achieve good results.

The next challenge is clock. Modern FPGAs have complex and sophisticated clocking architecture. Designs with many clocks can fall into hard dilemma: the placement without clock consideration will eventually failed in clock rule checking, whereas posing the clock constraints early greatly hurt the placement quality. Since the clocking architecture is unique on each FPGA family, there is no generic solution that can fit all situations.

The last but not least challenge is on tool runtime. FPGAs in many applications replaced ASIC because their ease of design and fast turn-around time. This poses great tool runtime requirement. Unlike ASIC tools that can run overnight or days to get the results, FPGA tools will be abandoned if they cannot complete most jobs in a couple of hours. The runtime for placer often needs to be within an hour. This is a very challenging goal considering the ever grown FPGA design size. The above runtime number is only for traditional FPGA designs. As FPGAs get used more and more as software develop platform (e.g., SDAccel from Xilinx), the runtime target is even higher.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ISPD'16, April 3–6, 2016, Santa Rosa, California, USA.
© 2016 ACM. ISBN 978-1-4503-4039-7/16/04…$15.00.
DOI: http://dx.doi.org/10.1145/2872334.2886419

In addition, many FPGA specific rules/constraints pose more restrictions on FPGA placement, including I/O placement, physical synthesis compatibility, data path, large modules like carry chain or cascaded BRAMs/DSPs, and processor area in some FPGA device.

3. FPGA ARCHITECTURE

Xilinx FPGAs [2], an example of which is illustrated in Figure 1, consist of an array of programmable blocks of different types, including general logic (CLB), memory (BRAM) and multiplier (DSP) blocks, surrounded by a programmable routing fabric (interconnect) that allows these blocks to be connected via horizontal and vertical routing channels. This array is surrounded by programmable input/output blocks (IO) that interface the chip to the outside world.

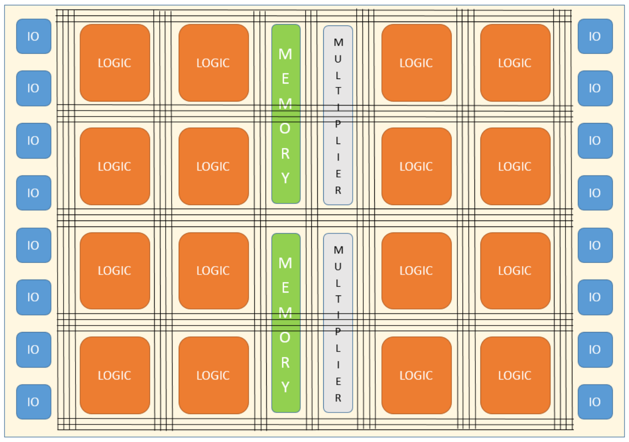


Figure 1. Example of Xilinx FPGA Architecture

This array has a configuration memory (SRAM) beneath it, which, when loaded with appropriate bits, programs the blocks and the interconnects to behave a certain way, as illustrated in Figure 2.

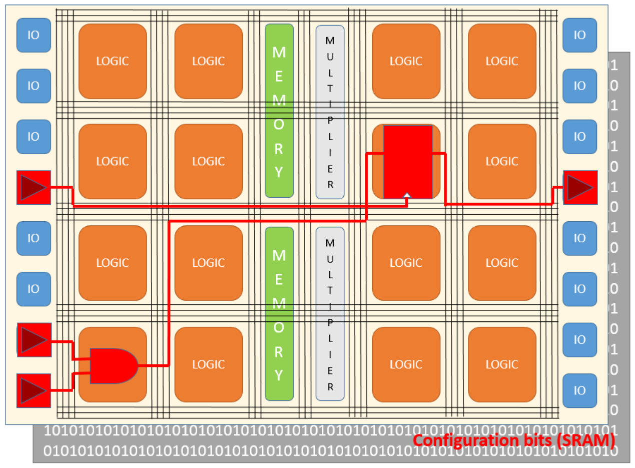


Figure 2. Example of Programming the Xilinx FPGA

Given a logic design that the user wants to implement on the FPGA, the Xilinx Implementation Tool flow (Vivado) converts the design into the appropriate set of configuration bits (Bitstream) which is loaded onto the SRAM to make the FPGA behave as the design. There are usually multiple steps involved in this tool flow, the main ones being, Synthesis, Placement, Routing, and Bitstream generation. Synthesis tool infers the design logic in terms of the logic blocks available within the FPGA. Placement tool places these inferred logic blocks on the various sites of physical logic blocks present in the FPGA. Routing tool connects up the pins of these physical logic blocks using the programmable interconnect routing structures in the FPGA. Bitstream generation tool then proceeds to generate the set of configuration bits that program these logic blocks and interconnect routing structures to behave as the design intended.

The general logic block (also referred to as the configurable logic block, or CLB), is the main resource for implementing general-purpose combinatorial and sequential circuits. The CLB is made up of the logic elements themselves, which are grouped together into a slice. These logic elements are of the type lookup tables (LUTs) or sequential elements (FFs). Each CLB contains one slice. Each slice provides sixteen LUTs and sixteen flip-flops. The slices and their CLBs are arranged in columns throughout the device. There are, however, certain restrictions pertaining to how these LUTs and FFs can be used within each slice. These are explained in detail in the “Placement Evaluation Flow” section under “Legalization Rules” subsection.

In the specific Xilinx FPGA we’re targeting for this contest, the XCVU095-ffva2104-es2 device, we have 67,200 CLB/SLICE locations, 880 usable IO locations, 770 DSP locations, and 1730 BRAM locations. More information on this device, and the architecture in general, can be obtained from [5].

4. BENCHMARKS

The benchmarks for ISPD 2016 placement contest have been generated using an internal netlist-generation tool based on Generate NetList (Gnl). The tool allows us to create netlists of different placement and routing complexities by varying the number of components and their interconnection. Additionally, it provides control over the type of components (primitives) used in the netlist. For ISPD benchmarks, we have restricted the primitives to be Look-Up-Tables (LUTs), Flip-Flops (FFs), DSP blocks (DSPs), and Block RAMs (BRAMs). The target device is xcvu095, part of the Virtex UltraScale [3] family.

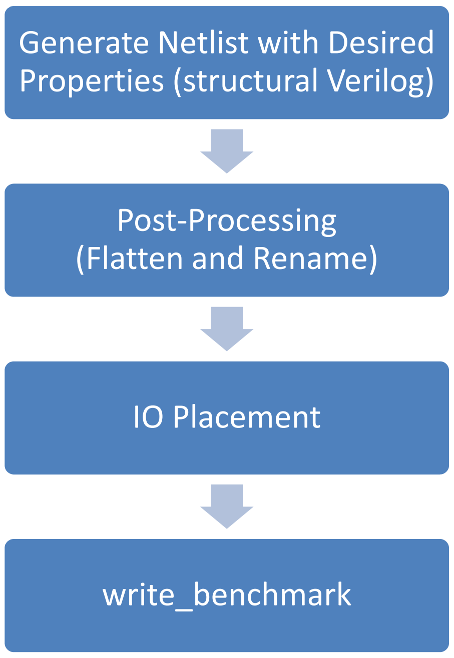
The following properties of the netlist were varied among the ISPD benchmarks.

1) Number of instances. We have created benchmarks that utilize 55% to 83% of the LUTs available. We have also varied the number of DSPs, BRAMs, and FFs to create medium to highly utilized designs.

2) Rent exponent. Interconnection complexity has been varied by creating netlists of different Rent exponents. This is important to test the routability aspect of the placement solution.

3) Number of resets. FPGA architecture limits the number of unique reset nets per Slice. Hence, by varying the number of resets we test how well the placer can support such restrictions.

Table 1 captures the characteristics of the benchmarks.



Generate Netlist with Desired Properties (structural Verilog)
Post-Processing (Flatten and Rename)
IO Placement
write_benchmark

Figure 3. Benchmark Generation Flow

Figure 3 explains the flow used for generating these benchmarks. First, we generate structural Verilog using our netlist-generation tool. The input to this tool is a configuration file, which specifies the desired parameters in the netlist. The structural Verilog file is post-processed to create a flattened design, without any hierarchies. Along with dissolving hierarchies, we also rename the instances and nets in this step. Next, we run Vivado placer to place IO ports of the design. Finally, we write the benchmark in Bookshelf format. The Bookshelf format list the instances in the design in a ".nodes" file and their interconnection in a ".nets" file. It also writes IO placement in a ".pl" file. Library cells are separately listed in a "*.lib" file.

5. PLACEMENT EVALUATION

5.1 Placement Interface

Contestants are expected to write the output of their placement tool in a specific (.pl) file format. Placer's output placement file should contain locations of all the instances in the design. The location of an instance has three fields: x-coord, y-coord (to determine the SITE) and BEL (index within the SITE). Figure 4 shows the BEL number for LUTs/FFs placed inside a SLICE SITE.

For BRAM and DSP instances, since there are no BELs within a SITE, the BEL index remains 0.

The following is a snippet of a placement file:

```
inst_1000 165 161 3        # (this instance is a LUT)
inst_1003 165 161 12       # (this instance is a FF)
inst_1100 29 0 0           # (this instance is a DSP)
inst_1200 34 0 0           # (this instance is a BRAM)
```

The placement output (.pl) file, will be given as an input to Xilinx Vivado tool using the flow.tcl file, which is available as part of each benchmarks archive. Vivado Placer will then read these instance placements, and check for legal placement on every

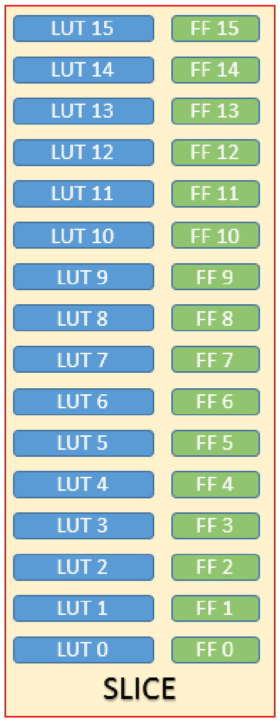


Figure 4. BEL offsets within a SLICE

instance. In case of illegal placement, Vivado Placer will error out with a reason behind the illegality for each instance. If the placement is legal, Vivado router starts and completes routing, or report unroutable design. If routing completes successfully, the following message indicates total routed wirelength: "Total Routed Wirelength: xxxxx (Vertical xxxx, Horizontal xxxx)". In case of unroutable placement, the following message shows up: "CRITICAL WARNING: [Route 35-162] xxxx signals failed to route due to routing congestion."

5.2 Legalization Rules

Each SLICE site provides sixteen LUTs and sixteen FFs. There are, however, certain restrictions pertaining to how these LUTs and FFs can be used within each SLICE.

Using LUTs in a SLICE:

- The 16 LUTs within SLICE are conceptual LUTs that can only be fully used under certain conditions:
- When implementing a 6-input LUT with one output, one can only use LUT 1 (leaving LUT 0 unused) or LUT 3 (leaving LUT 2 unused) or ... or LUT 15 (leaving LUT 14 unused)
- When implementing two 5-input LUTs with separate outputs but common inputs, one can use {LUT 0, LUT 1} or {LUT 2, LUT 3} or ... or {LUT 14, LUT 15}
- The above rule of coming LUTs with separate outputs but common inputs, holds good for 5-input LUTs (as mentioned above) or fewer input LUTs as well
- When implementing two 3-input (or fewer input) LUTs together (irrespective of common inputs), one can use {LUT 0, LUT 1} or {LUT 2, LUT 3} or ... or {LUT 14, LUT 15}

Using FFs in a SLICE:

- There are 16 FFs per SLICE (two per LUT pair), and all can be used fully under certain conditions:
- All FFs can take independent inputs from outside the SLICE, or outputs of their corresponding LUT pair (FF 0 can take LUT 0 or LUT 1 output as input, ..., FF 15 can take LUT 14 or LUT 15 output as input)
- All can be configured as either edge-triggered D-type flip-flops or level-sensitive latches. The latch option is by top or bottom half of the SLICE (0 to 7, and 8 to 15). If the latch option is selected on a FF, all eight FFs in that half must be either used as latches or left unused. When configured as a latch, the latch is transparent when the clock input (CLK) is high.
- There are two clock inputs (CLK) and two set/reset inputs (SR) to every SLICE for the FFs. Each clock or set/reset input is dedicated to eight of the sixteen FFs, split by top and bottom halves (0 to 7, and 8 to 15). FF pairs ({0,1} or {2,3} or ... or {14,15}) share the same clock and set/reset signals. The clock and set/reset signals have programmable polarity at their slice inputs, allowing any inversion to be automatically absorbed into the CLB.
- There are four clock enables (CE) per SLICE. The clock enables are split both by top and bottom halves, and by the two FFs per LUT-pair. Thus, the CEs are independent for: {FF 0, FF 2, FF 4, FF 6}, {FF 1, FF 3, FF 5, FF 7}, {FF 8, FF 10, FF 12, FF 14}, {FF 9, FF 11, FF 13, FF 15}. When one storage element has CE enabled, the other three storage elements in the group must also have CE enabled. The CE is always active High at the slice, but can be inverted in the source logic.
- The two SR set/reset inputs to a SLICE can be programmed to be synchronous or asynchronous. The set/reset signal can be programmed to be a set or reset, but not both, for any individual FF. The configuration options for the SR set and reset functionality of a register or latch are: No set or reset, Synchronous set (FDSE primitive), Synchronous reset (FDRE primitive), Asynchronous set (preset) (FDPE primitive), Asynchronous reset (clear) (FDCE primitive). The SR set/reset input can be ignored for groups of four flip-flops (the same groups as controlled by the CE inputs). When one FF has SR enabled, the other three FFs in the group must also have SR enabled.
- The choice of set or reset can be controlled individually for each FF in a SLICE. The choice of synchronous (SYNC) or asynchronous (ASYNC) set/reset (SYNC_ATTR) is controlled in groups of eight FFs, individually for the two separate SR inputs.

Some of these FF Packing rules are illustrated in Figure 5.

More information on the CLB composition can be obtained from [4].

5.3 Evaluation Metrics

- For each design in the benchmark suite, the placers will be ranked based on the contest evaluation metric. The final rank for a placer will be the sum of the individual ranks on all the circuits. The placer with the smallest total rank wins the contest.
- The placement runtime must be 12 hours or shorter.
- The placement must be legal (legalization rules are described in the previous section).

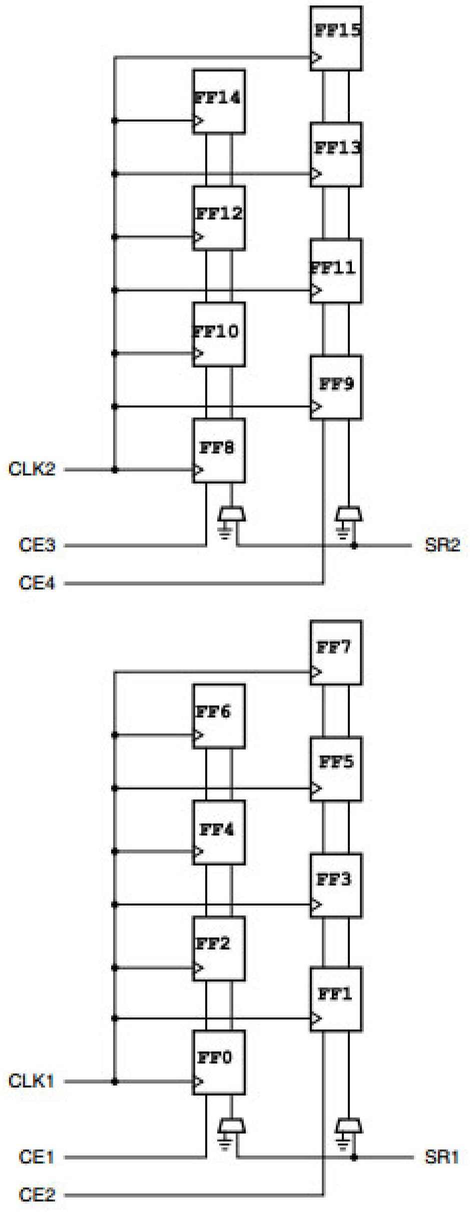


Figure 5. Flip Flop control signals connectivity within a SLICE

- The placement has to be routed by Vivado router, and the router has to complete the job within 12 hours. Routing is regarded as failed if it takes more than 12 hours to complete.
- PlacementScore=RoutedWirelength*(1 + Runtime_Factor)
  - Vivado router reports total routed wirelength. This is the base of the score.
  - Total placement and routing runtime will be used in computing P&R_Runtime_Factor;
  - Runtime_Factor= -(Runtime - Median_Runtime) / 10.0
  - There is 1% scaling factor for every 10% runtime reduction/addition against the median runtime of all place+route solutions;
  - Runtime factor is between -10% and +10%

  - Although runtime is a part of the contest metric, the "Total Routed Wirelength" will be the dominant component. In other words, a placer will not get a significant advantage if it is extremely fast compared to the median runtime of all the placers participating in the contest.
- The failed place/route job will get the lowest rank on this design. In the presence of multiple failures, the break-tie factors are: placer failure or router failure, router runtime, number of unrouted nets, number of illegal placements.

6. ACKNOWLEDGMENTS

The authors would like to thank Dr. Ismail Bustany and Dr. Steven Li for their valuable suggestions on the contest details.

7. REFERENCES

[1] R. Aggarwal, 2014. *FPGA Place and Route Challenges*. In Proc. of International Symposium on Physical Design.

[2] Xilinx, "UltraScale Architecture", http://www.xilinx.com/products/technology/ultrascale.html

[3] Xilinx, "Virtex UltraScale FPGAs", http://www.xilinx.com/publications/prod_mktg/ultrascalevirtex-product-table

[4] Xilinx, "UltraScale Architecture Configurable Logic Block User Guide", http://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf

[5] Xilinx, "UltraScale Architecture and Product Overview", http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf

[6] GNL: http://users.elis.ugent.be/~dstrooba/gnl/

Design	#LUTs	#FFs	#BRAMs	#DSPs	#I/O	#Control Sets	Rent exponent
Design1	300K (55%)	241K (22%)	400 (23%)	200 (26%)	453 (54%)	651	0.5
Design2	300K (55%)	241K (22%)	400 (23%)	200 (26%)	453 (54%)	651	0.6
Design3	350K (65%)	259K (24%)	800 (46%)	300 (39%)	533 (64%)	1271	0.7
Design4	400K (74%)	304K (28%)	800 (46%)	500 (65%)	533 (64%)	1271	0.6
Design5	400K (74%)	292K (27%)	800 (46%)	500 (65%)	533 (64%)	1271	0.7
Design6	450K (83%)	338K (31%)	1000 (58%)	400 (52%)	603 (72%)	2091	0.55
Design7	450K (83%)	339K (31%)	1000 (58%)	400 (52%)	603 (72%)	2091	0.65

Table 1. Benchmark statistics

*Number in parenthesis indicates the utilization as percentage of available resources in the FPGA [6]