From C to Fault-Tolerant FPGA-based Systems

Dimitris Agiakatsikas*, Ganghee Lee*, Thomas Mitchell*, Ediz Cetin†, and Oliver Diessel*
*School of Computer Science and Engineering, UNSW Sydney
†School of Engineering, Macquarie University Australia
{d.agiakatsikas, ganghee.lee, thomas.mitchell, o.diessel}@unsw.edu.au, ediz.cetin@mq.edu.au

***Abstract*—This work presents an automated flow for producing fault-tolerant Field Programmable Gate Array (FPGA) systems. The flow uses the TLegUp High Level Synthesis (HLS) tool to generate triplicated register-transfer level designs for algorithms expressed in the C language and Vivado design suite for their implementation on Xilinx 7-series FPGAs. TLegUp has been extended to partition the design into a number of Triple Modular Redundant (TMR) components, which can be optionally floorplanned during their implementation. Partitioning the TMR design into a network of smaller TMR components and isolating their modules through flooplanning increases system reliability. We implemented a fine- and a coarse grain approach to partition the design, whereby the former approach uses a network flow algorithm to partition the application's Data Flow Graph (DFG) at the instruction level, while the latter uses the same algorithm to partition the design at the C function level. Results reveal that both approaches provide similar reliability enhancement to the system, but function-level partitioned designs are smaller and faster.**

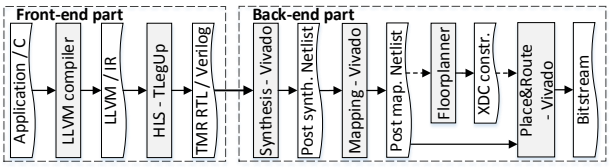

Figure 1. Architecture of the proposed flow.

Fig .1 illustrates the proposed flow, which consists of two parts, a front- and a back-end part. The front-end uses TLegUp [1] to generate Verilog code of TMR-based designs for algorithms described in the C language. This part involves: (i) high-level synthesis (HLS) and partitioning of the design, (ii) voter insertion as described in [2] and (iii) triplication of the design as guided in [3]. The flow uses a modified version of the network flow algorithm presented in [4] to partition the design into a user defined number of TMR components by applying either, (i) function-level partitioning (FLP), whereby the applications C functions are clustered into partitions, or (ii) by applying instruction-level partitioning (ILP) on the DFG of the entire computation of the C algorithm which is obtained from the LegUp flow [5] once all C functions of the C algorithm are inlined and translated into LLVM intermediate representation. Both approaches aim to balance resource utilization between partitions, while minimizing the total bit-width of signals used between them. The Verilog is then implemented on an FPGA with the back-end. The back-end involves netlist synthesis, technology mapping, placement, routing, and bitstream generation, while the design can optionally be floorplanned by the academic tool [6].

We used our flow to implement non-floorplanned and floorplanned TMR designs of 17 CHstone, DWARV and Bambu HLS bechnarks on a Xilinx Artix-7 200T FPGA and compared them against simplex (non-triplicated) design versions. The TMR designs were partitioned with ILP and FLP into k = 1, 2, 4 and 8 TMR components (i.e. partitions). The results show that both the FLP and ILP circuits utilize approximately $3 - 4\times$ more resources than the simplex circuits when k = 1. However, the ILP circuits suffer an exponential utilization increase as k increases. LegUp generates a Finite State Machine (FSM) for each C function during HLS. Thereby, the ILP designs in which all C functions are inlined into the main function have only one centralized complex FSM to control the entire circuit. Therefore, more partitions result in more wires and as a consequence in more voters to interconnect the centralized FSM with the partitions of an ILP circuit. This has negative effects on the operating frequency (FM) and the resource balance between the partitions of the ILP circuits. On the other hand, results of FLP circuits are consistently more balanced than ILP across all k for all metrics we considered. Finally, fault-injection experiments (conducted in the same way as in [1]) showed that both ILP and FLP circuits with k = 1 and 2 are approximately $500\times$ less sensitive to configuration memory soft-errors, which can be improved by a factor of $1.3\times - 3.4\times$, on average, when the circuits are flooplanned.

ACKNOWLEDGEMENT

This research was supported in part by the Australian Research Council's Linkage (LP140100328) and Discovery (DP150103866) Projects funding schemes. We also thank Dr. Antonio Miele for providing the floorplanner [6].

REFERENCES

[1] G. Lee *et al.*, "TLegUp: A TMR Code Generation Tool for SRAM-Based FPGA Applications Using HLS," in *IEEE Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 1–4.

[2] J. M. Johnson *et al.*, "Voter insertion algorithms for FPGA designs using triple modular redundancy," in *ACM/SIGDA Int. Symp. on Field-programmable Gate Arrays (FPGA)*, 2010.

[3] C. Carmichael, *Triple module redundancy design techniques for Virtex FPGAs*, Xilinx Application Note XAPP197, 2001.

[4] H. Liu *et al.*, "Network-flow-based multiway partitioning with area and pin constraints," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 1, pp. 50–59, 1998.

[5] A. Canis *et al.*, "LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems," *ACM Trans. on Embedded Computing Systems (TECS)*, vol. 13, no. 2, p. 24, 2013.

[6] M. Rabozzi *et al.*, "Floorplanning Automation for Partial-Reconfigurable FPGAs via Feasible Placements Generation," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 1, pp. 151–164, Jan 2017.