SPORT Lab SFQ Logic Circuit Benchmark Suite

Naveen Katam, Soheil Nazar Shahsavani, Ting-Ru Lin,
Ghasem Pasandi, Alireza Shafaei, and Massoud Pedram

Computer Engineering Technical Report Number CENG-2017-01

Ming Hsieh Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, California 90089-2562

April 2017

SPORT Lab SFQ Logic Circuit Benchmark Suite

Naveen Katam, Soheil Nazar Shahsavani, Ting-Ru Lin,
Ghasem Pasandi, Alireza Shafaei, and Massoud Pedram

Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089

{nkatam, nazarsha, tingruli, pasandi, shafaeib, pedram}@usc.edu

1 Introduction

Single flux quantum (RSFQ) technology uses Josephson junctions (JJs) as nonlinear active elements, instead of transistors in the CMOS technology, in order to develop an extremely high speed and low power logic circuit family. This is because JJs switch quickly (~ 1 ps) and dissipate little switching energy ($\sim 10^{-18}$ J) at low temperatures. However, many problems, including architectures and design automation methodologies and tools, must be solved before SFQ can become a realistic option for realizing large-scale, high-performance, and energy-efficient computing systems in the future. More specifically, the electronic design automation (EDA) tool-chain for SFQ technologies is incomplete or limited both in capability and scale. Hence, the purpose of this report is to provide benchmarks for interested readers to advance the state-of-the-art in logic synthesis and physical design problems of large SFQ circuits.

For each benchmark, the following files are provided:

1. A Verilog HDL file, which is the input to the logic synthesis tool. Since we are using ABC as our main logic synthesis tool, which does not support multi-bits in the Verilog file, all signals and buses are defined as bit-level variables.
2. A set of files specified in Bookshelf format, which are inputs to the placement tool. We assume this input netlist is path-balanced and all fanouts are implemented with splitters. More details on this will follow next.

2 Path-Balancing

If the zero-skew clocking scheme (H-tree network) is used for the SFQ circuit, then the circuit should be path-balanced to make sure that each cell samples the correct timing instance of each signal. In a path-balanced circuit, all paths from any primary input to any primary output have the same logical depth. Any netlist can be path-balanced by inserting D flip-flops (DFFs) to paths with a smaller depth.

Our algorithm for path-balancing a circuit is as follows. If a cell with logic level i has a fanout cell with logic level j, then by inserting $j - i - 1$ DFFs between these two cells, we can balance the corresponding path. Therefore, for each fanout of each cell, we insert the required number of DFFs. We then run the retiming algorithm in ABC to reduce the number of path-balancing DFFs. Our algorithm does not result in the minimum number of path-balancing DFFs. In fact, designing an algorithm that inserts the minimum number of DFFs for path-balancing a netlist is an open problem in the SFQ logic.

3 Fanout Implementation

SFQ logic cells are limited to driving only one fanout cell, because of the difficulty in distributing the pulse to multiple fanouts. To take multiple fanouts from a cell, it is necessary to insert splitter cells. A splitter cell typically has one input and two outputs (as input pulse is regenerated at each output as a pulse). As a result, to realize a fanout of n, we need $n-1$ splitter cells, which are implemented as a complete binary tree to reduce the overall splitting delay.

4 List of Benchmarks

The benchmark suite currently includes the following circuits:

- Carry look-ahead adder (CLA): A 4-bit CLA, and a 16-bit adder where four blocks of 4-bit CLAs are cascaded together.
- Kogge-Stone adder (KSA): 4-, 8-, 16-, and 32-bit KSAs.
- Integer divider: 4-, 8-, and 16-bit integer dividers based on the restoring division. The basic version of fully-combinational implementation of the 4-bit integer divider is shown in Figure 1. This structure is optimized to get low gate-count.

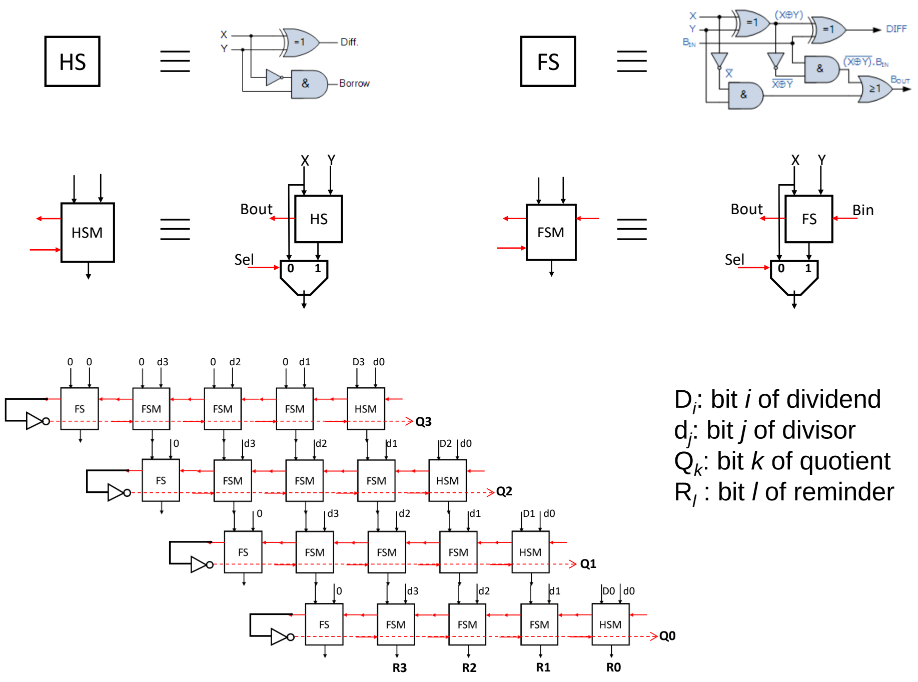

Figure 1. Integer divider.

Acknowledgments

This work is supported by the Safe and Secure Operations Office of the Intelligence Advanced Research Projects Activity (IARPA) under contract no. FA8750-15-C-0203-IARPA-BAA-14-03.