Design Space Exploration of 8-bit Approximate Prefix Adders

I. Introduction:

This project is to about to covering all the possible 8-bit approximate prefix adders from 4 different prefix trees, Brent Kung, Han Carlson, Sklansky and Kogge Stone Adders and extracting best approximate adders using pareto optimal graph between error matrices and hardware implementation matrices. The prefix tree of all the adders are shown in the figure 1.

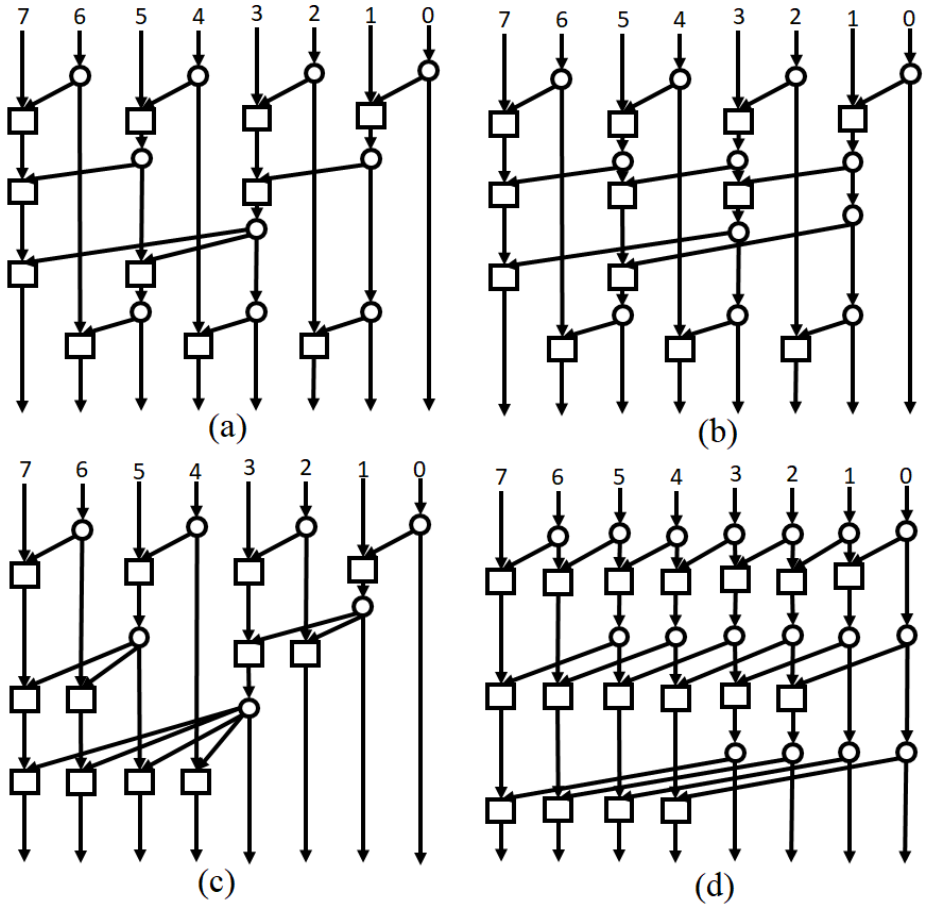

Figure 1. Prefix trees of 8-bit (a) Brent Kung Adder, (b) Han Carlson Adder, (c) Sklansky Adder and (d) Kogge Stone Adder

II. Methodology:

The project can be divided into four parts written below. In the prefix tree, all the straight lines are called bit lines, all the square blocks called generation block, circles are used branching where each branch gets all the data from above and cross lines connected between circles and square are called branches. These nomenclatures are limited only to this document.

a. Extracting all possible approximate prefix trees from the given adder:

This is first and the most important step of all. To extract all the possible adders, I have followed some rules which are mentioned in this section. All the generation blocks are either placed as it is in their place or removed completely from the tree which means either they are not in the approximate adder, or they are at the same place as in the original trees.

All the bit lines follow the same path in both the case. And the branches are connected to same bit lines, but their connection can be changed according to the generation block in the same bit line. Which means if the branch is connected to some line in the routing side, it can be originated from all the generation block of previous level or it can also be originated from staring of the bit line. Any configuration following all these rules are included into the design space of the approximate 8-bit prefix tree adders and exported into pickle file for the future uses. The nomenclature to define the approximate prefix tree is as follows, [(generation blocks of 1st stage), (generation blocks of 2nd stages), [level of the previous blocks connected via branches], (generation blocks of 3rd stages), [level of the previous blocks connected via branches]…], in the 1st stage the branches are always connected to the origin of previous bit line so there is no need to specify the level.

b. Automatically generating the Verilog RTL and collecting performance results:

The basic flow of the Verilog module is the same, it has module name in the starting followed by I/O name declaration and I/O size. The logic flow of all the prefix tree is also same for all the configurations, which is described in the figure 2. The prefix tree only describes the stage 2 of the logic, pre processing and post processing are same for all the adders. Based on the exported adders from the previous section, the automation script generates all the RTL code which is followed by hardware synthesizing in *abc* tool and exporting area and delay results for all the adders. (For better results we should synthesise all the configuration in Synopsys DC using nangate open source library before making pareto optimal curves).

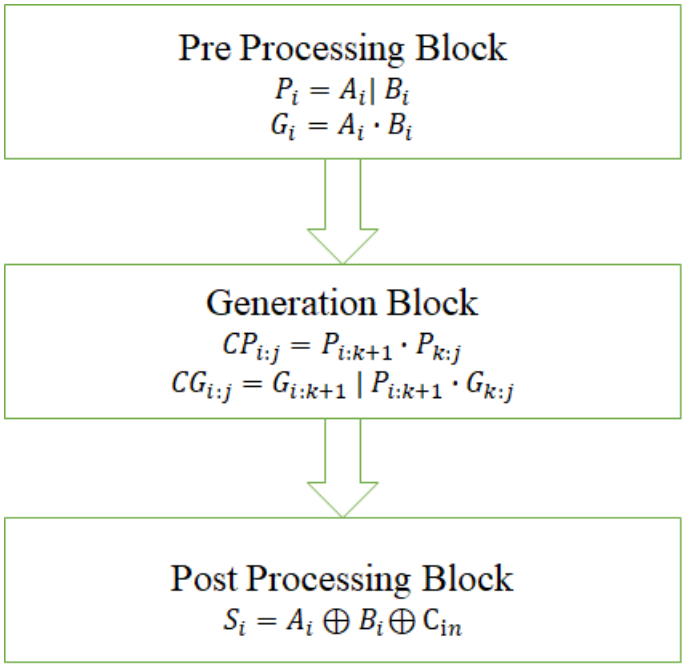

Figure 2. Logic flow of prefix trees

c. Creating error matrices:

For this section, same logic is written into python script which describes all the configuration using the exported adders in section a. All the possible inputs (256*257/2) are given and the approximate sums are calculated for all the inputs. Bases on the approximate sums and accurate sums seven different error matrices are calculated which

are as follows, ER - Error Rate, MAE - Mean Absolute Error, WCAE - Worst Case Absolute Error, MSE - Mean Square Error, WCSE - Worst Case Square Error, MRE - Mean Relative Error and WCRE - Worst Case Relative Error.

d. Assembling all the pareto optimal adders:

From hardware implementation matrices and error matrices, various pareto optimal curves needs to be generated and all the shortlisted adders should be extracted for different curves. The area, delay and power results for all the shortlisted adder should be generated using Synopsys DC and nangate library (this can also be done for all the configurations). The library containing all the matrices, Verilog codes and python logic needs to be created which was the aim of the project.

III. Work done:

In the methodology, section a, b and c are completed, the work for last section and assembling the library is remaining.