Discrete Fourier Transform IP Generator[*]

Grace Nordin James C. Hoe Markus Püeschel
Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA

Abstract

Intellectual Property (IP) libraries are commonly used by hardware designers to increase productivity and reduce the time-to-market. These static IP libraries do not allow the designers flexibility in customizing trade-offs. We propose a parameterized DSP IP generator that allows designers to specify the cost/performance tradeoff. We present a prototype implementation of a parameterized DFT generator and compare our generated DFT with Xilinx Logicore's DFT IP Core. Our results show that we generate high-quality DFT blocks that match the performance and cost of Xilinx LogiCore DFT implementations. More importantly, we show that our parameterized design generation yields customized DFT blocks over a range of different performance/cost tradeoff points.

Introduction. We propose a parameterized IP generator as an alternative to static IP blocks. The generator is tailored for application-specific tradeoffs, such as area, performance, numerical accuracy and power consumption. Our approach preserves the advantage of using static IP blocks, while allowing the designers more control over the design. This generator can be used together with a search engine to find the best possible implementation for a given set of constraints. Here we present our experience in developing a parameterized generator for discrete Fourier transform (DFT). A full description of this work has been submitted to a conference.

Generation of Discrete Fourier Transform. Our DFT generator is based on the Pease algorithm for the DFT, which we express in a formula notation as

$$\mathrm{F}_{2^n} = \{\prod_{i=0}^{n-1} \mathrm{L}_2^{2^n} (\mathrm{I}_{2^{n-1}} \otimes F_2)\, \mathrm{T}_{n-i}\}\, \mathrm{R}_{2^n}, \quad \mathrm{F}_2 = \left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right) \tag{1}$$

where ‘I’ denotes an identity matrix, ‘$\otimes$’ the Kronecker product of matrices, ’T’ denotes the Twiddle factors, ‘R’ denotes the bit reversal, and ‘L’ a stride permutation. Figure 1 shows a dataflow representation of (1) for $n = 3$. This formula-derived dataflow graph can be directly mapped to a combinational circuit where the implementation cost is approximately $n \log(n)/2$ C blocks, plus the routing cost of realizing the L_2^n wire permutations. The cost of a combinational implementation is usually very large and unrealistic to implement for large n. A common practical DFT implementation requires a sequential implementation where the logic resources, e.g., C, are reused multiple times by *horizontal folding* or *vertical folding*. Figure 1 shows, for $n = 3$, block diagrams of a horizontally folded DFT (middle) and a horizontally and vertically folded DFT (right).

Our DFT generator accepts as input parameters the DFT size, the data format (i.e., fixed-point number range and precision), and a design parameter p that controls the degree of parallelism in the generated implementation. This freedom allows the designer to select a custom tradeoff between minimizing cost (i.e., area and power) and maximizing performance (i.e., latency and throughput). Our DFT generator can also accept target-specific parameters to reflect the designer's preference for different classes of resources. A parameter that our DFT generator allows is a relative value for a *Block Select-RAM* (BRAM, a specialized memory

[*]This work was supported by NSF through award ITR/NGS-0325687

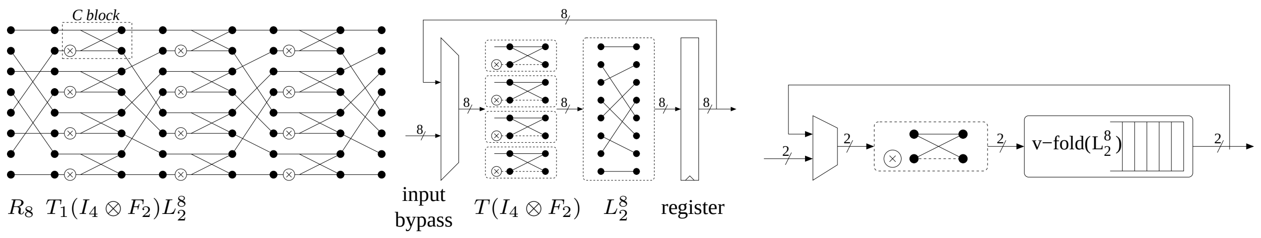

Figure 1: Pease DFT algorithm. From left to right: completely flattened, horizontally folded, fully horizontally and vertically folded.

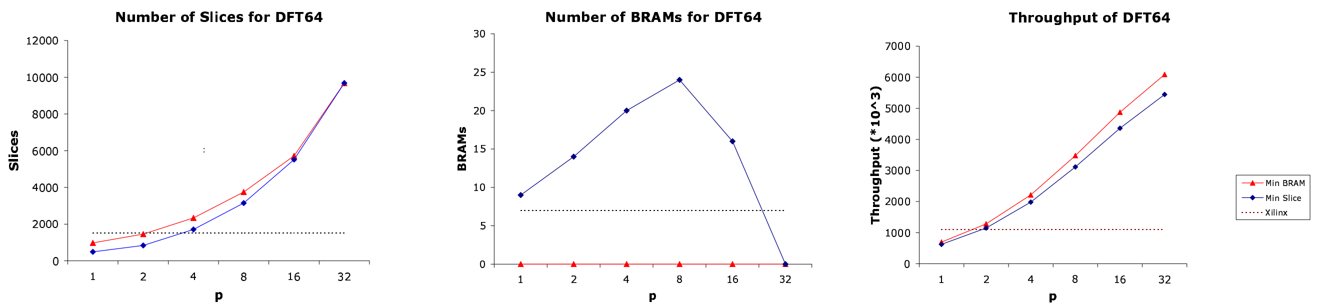

Figure 2: Synthesis results for F_{64}: slice utilization, BRAM utilization, and transform throughput (transform/second, overlapping loading and unloading).

Table 1: Parameters for DFT IP Generator and the corresponding effects on logic slices, BRAM and throughput as each parameter increases

Parameter	Logic Slices	BRAM	Throughput
n (transform size)	⇑	⇑	⇓
p (parallelism)	⇑	⇑	⇑
fixed-point number range and precision	⇑	⇑	–
relative value of BRAM cost	⇓	⇑	–

primitive) in terms of *slices* (generic logic building blocks). The DFT generator takes this preference into account to balance resource minimization across BRAM and logic slice utilization. Table 1 lists some current parameters that our DFT generator currently supports, and the corresponding effects on BRAM, logic slice utilization and throughput. The output of our generator is an RTL-level Verilog description of the desired DFT implementation.

Sample Result. For $n = 6$, our DFT generator produces 6 implementations representing different tradeoffs between the different design goals and constraints. Figure 2 shows the resource utilization, in terms of slices and BRAM, and throughput over these 6 design choices, compared against the latest Xilinx LogiCore DFT implementations. The generated DFT implementations are synthesized for the Xilinx Virtex2-Pro XC2VP100-6FF1696 FPGA using Xilinx ISE version 6.1.03i. To show the effects of our Xilinx Virtex2-Pro-specific parameters, each graph reports two separate results corresponding to the extreme tradeoff points of slices and BRAM utilization. They are 1) minimize the use of slices; and 2) minimize the use of BRAMs. As the graphs show, our minimum resource design points (i.e., $p = 1$ or $p = 2$) occupy a similar tradeoff space as the Xilinx LogiCore DFT implementations. By varying p and the relative value of BRAM, the designer can customize the tradeoff function between performance, slice, and BRAM usage. For larger p values, our generated DFT implementations can offer a higher throughput at the cost of an increased resource requirements.