A Library of Parameterized Hardware Modules for Floating-Point Arithmetic and Their Use

Pavle Belanović and Miriam Leeser
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA, 02115, USA
{pbelanov,mel}@ece.neu.edu

Many algorithms benefit from acceleration with reconfigurable hardware. This acceleration results from the exploitation of the fine-grained parallelism available in reconfigurable hardware. Custom circuits built for these applications process values in fixed or floating-point formats. Minimizing bitwidths of signals makes more parallelism in the implementations possible and reduces power dissipation of the circuit. However, this requires fine-grained control over the fixed and floating-point formats used. Arbitrary fixed-point formats are straight-forward to implement and are in common use. It is much more difficult to implement arbitrary floating-point formats due to the inherent complexity of the floating-point representation. We present a library of hardware modules that makes implementation of custom designs with arbitrary floating-point formats possible.

In order to operate on any custom floating-point format, all our hardware modules have been parameterized. Every module's VHDL description accepts two parameters: *exp_bits* and *man_bits*, representing exponent and mantissa bitwidths respectively. These two values are sufficient to describe any floating-point format when combined with the sign field, which always has a bitwidth of 1. The total bitwidth of the format is $1 + exp\_bits + man\_bits$. These parameters allow for the creation of a wide range of different modules through a single VHDL description. Values of the parameters help resolve the description at compile time and ensure synthesis of the correct amount of logic needed to perform the function of the module for the given format.

We provide a library of parameterized components that are pipelined and cascadable to form pipelines of floating-point operations. Each component has a ready and a done signal to allow them to be easily assembled into larger designs. Some error handling is supported, and detected errors are propagated to the end of the pipelined design. The hardware modules fall into three categories. Format control modules support denormalizing, rounding and normalizing operations. The arithmetic operators include addition, subtraction and multiplication. The format conversion modules convert between fixed-point and floating-point representations. The names, functions and latencies (in clock cycles) of all modules presented are shown in Table 1. All modules have a throughput of 1 clock cycle.

Table 1: Hardware modules

Module	Function	Latency
`denorm`	Introduction of implied integer digit	0
`rnd_norm`	Normalizing and rounding	2
`fp_add`	Addition	4
`fp_sub`	Subtraction	4
`fp_mul`	Multiplication	3
`fix2float`	Unsigned fixed point to floating-point conversion	4
	Signed fixed-point to floating-point conversion	5
`float2fix`	floating-point to unsigned fixed point conversion	4
	floating-point to signed fixed point conversion	5

Synthesis results for parameterized arithmetic operator modules are presented in Figure 1 for a set of floating-point formats ranging in total bitwidth from 8 to 32 bits. The quantities for the area of each instance are expressed in slices of the Xilinx XCV1000 FPGA. Results for the `fp_add` module also represent the `fp_sub` module, which has the same amount of logic. The number of operator cores per processing element is based on an XCV1000 FPGA with 85% area utilization, including an overhead allowance of approximately 1200 slices for necessary circuitry other than the operators themselves. Thus, the numbers shown correspond to a realistic design using the operator cores. The results in Figure 1 show growth in area with increasing total bitwidth for add and multiply modules. The inverse effect is the reduction in the number of cores that can realistically fit onto an XCV1000 processing element.

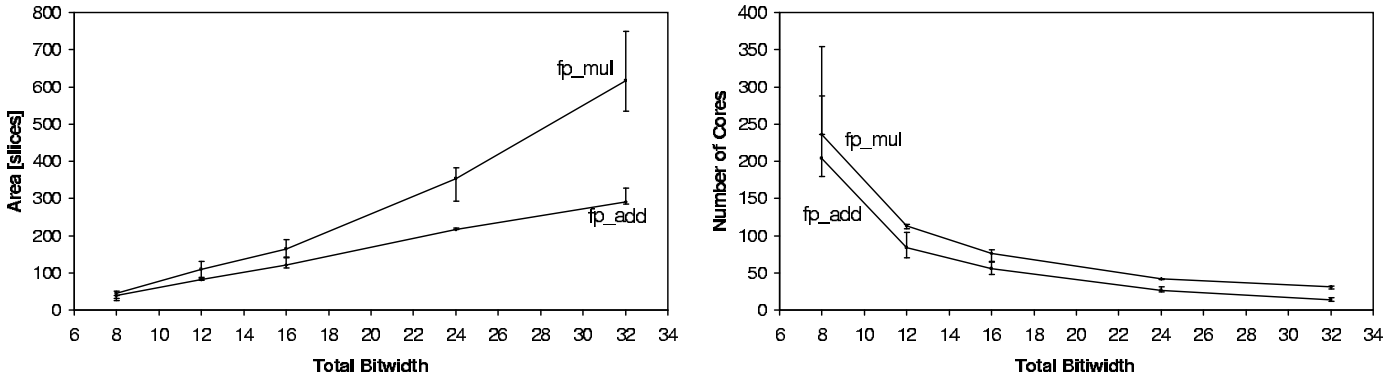

Figure 1: Growth of area and reduction in number of cores per processing element with increasing bitwidth

We have used the parameterized hardware modules to design and implement a hybrid fixed and floating-point implementation of the K-means clustering algorithm for multispectral and hyperspectral images. K-means is an unsupervised clustering algorithm that operates by assigning multidimensional pixels to one of K clusters. The aim of the algorithm is to minimize the variance of all pixels within each cluster. The algorithm is iterative: after each pixel is assigned to one of the clusters, the positions of cluster centers are re-calculated and the pixels are re-assigned until the algorithms converges. A previous implementation of the K-means algorithm [1] is based on a purely fixed-point datapath. The implementation presented here uses the hybrid fixed and floating-point datapath shown in Figure 2. Both implementations are designed to operate on 10-channel multispectral data, with 12 bits per channel. This data is segmented into 8 clusters. Comparison of the new, hybrid implementation to the existing, purely fixed-point

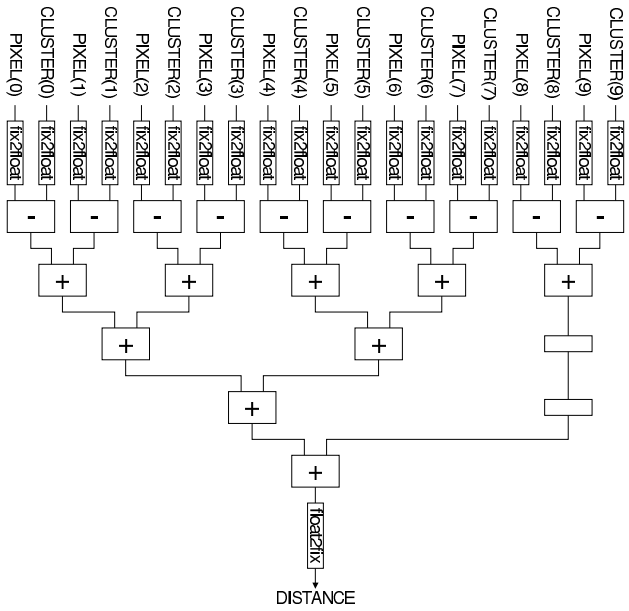

Figure 2: Floating-point section of the hybrid datapath

one shows that the library modules can be used to form fully pipelined circuits that perform a chain of arithmetic operations correctly in a fully custom floating-point format. Good quality K-means clustering is achieved by both implementations.

References

[1] M. Estlick, M. Leeser, J. Theiler, and J. Szymanski. Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware. In *International Symposium on Field-Programmable Gate Arrays*, pages 103–110. ACM, February 2001.