Floating Point Synthesis From Model-Based Design

Mark Jervis, Martin Langhammer, Graham Griffiths
mjervis@altera.com, mlangham@altera.com, ggriffit@altera.com
Altera Europe
Holmers Farm Way
High Wycombe
Bucks HP12 4XF

Introduction

Model-based design tools such as DSP Builder [1] now enable you to follow a software-based design flow while targeting FPGAs. DSP Builder adds specialized Simulink libraries (‘block-sets’) to the MATLAB design and simulation environment that allow you to implement DSP designs quickly and easily. The block-set is based on a high level synthesis technology that optimizes the high level, untimed net-list entered as a schematic into low level, pipelined hardware targeted to the chosen Altera FPGA device, clock rate and data sample rate. The hardware is written out as plain text VHDL, along with scripts that integrate with the Quartus II software and the ModelSim simulator. The combination of these features allows you to create a design without needing detailed device knowledge, and generates a high quality implementation that runs on a variety of FPGA families with different hardware architectures. By specifying the desired clock frequency, the tool solves timing closure issues by generating register transfer level (RTL) code that is automatically pipelined. To date, this design flow has been restricted to fixed-point data representations.

Floating point system level design has traditionally been very difficult to realize on FPGAs, due to limitations in routing architectures. Recently, a new method of fused data path synthesis [2] has been introduced for FPGAs, allowing the implementation of data paths with hundreds of floating point operators. System speeds of 300 MHz for single precision, and 225 MHz for double precision are possible through a push-button flow. By combining the Floating Point Compiler with the above model-based design tool we enable high-level floating point synthesis from model-based design.

DSP Builder

The high-level model-based design approach adopted in DSP Builder allows the user to specify their algorithm without having to account explicitly for delays, or the particular architectural features of the underlying FPGA components. Figure 1 below shows such a schematic for an infinite impulse response (IIR) filter. A (valid, channel, data) protocol is used throughout, which allows synchronisation of such subsystems without cycle counting. This allows the tool to pipeline the system as necessary to reach the target clock frequency without altering the algorithm.

At the same time features like data-type propagation make the creation of such schematic designs quick and simple to modify.

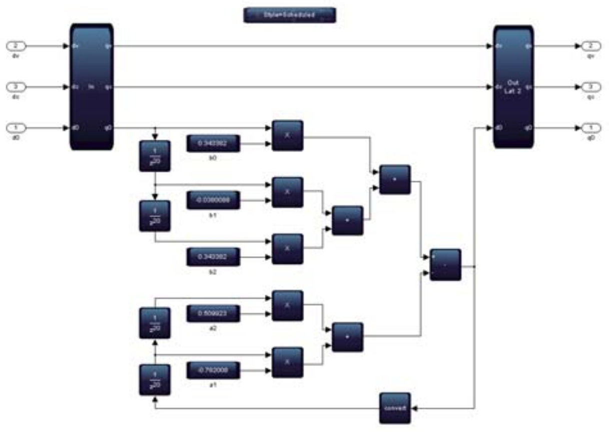

Figure 1: DSP Builder schematic design of an IIR filter.

Within the tool, the schematic is mapped to a data flow graph (DFG). The data flow graph is realised as a net-list of basic functional units. These functional units have a direct RTL implementation. One Simulink block may map to 0 (in the case of simple wiring), 1 or many functional units.

Since the representation has no set latency constraint, we are free to apply optimisation transforms both within and across groups of functional units that alter latency, provided it is done consistently. Maintaining a balance of latencies while pipelining to achieve a clock rate target can be framed as a linear programming problem (LPP) and solved with a standard LPP solver, such as SYMPHONY.

Optimizations across or on groups of functional units include reorganization of adder trees, bringing adder tree stages following multipliers into DSP blocks, splitting and pipelining adder chains, duplicate code removal, canonical signed digit representation of constant multipliers, and automated time-division multiplexing.

FPC Integration

FPC integration in DSP Builder brings a model-based design front end to the FPC technology outlined in [2]. It also brings other benefits of a shared environment with the existing tool; simulation with automated 3rd-party verification, integrated HDL with other DSP Builder blocks, memory-mapped interfaces, hardware-in-the-loop simulation and Quartus project and ModelSim script generation.

The FPC is initially integrated as a separate library of primitive blocks.

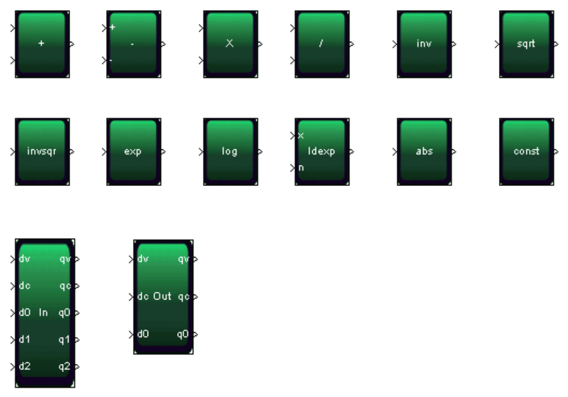

Figure 2: DSP Builder Floating Point Primitive Library.

Currently supported functions are multiply, add, subtract, divide, inverse (1/x), square root, inverse square root, exponent (e^x), natural logarithm, (ln(x)), load exponent, $x2^n$ and absolute value. These are effectively C MATH.H library elements, which map to multiplier based architectures where appropriate.

In addition, two other blocks mark the separation of the fixed- and floating point domains within the system. These follow the same (valid, channel, data) protocol used throughout DSP Builder. They generate floating point conversion RTL code and in simulation correct for latencies added in hardware, such that at a subsystem level the design simulates with cycle accuracy.

Designs for the floating point part of the design are created schematically with blocks from the floating point library. The boundary blocks provide a conversion between the fixed and floating point domains where necessary, such that a mixed system can be constructed seamlessly.

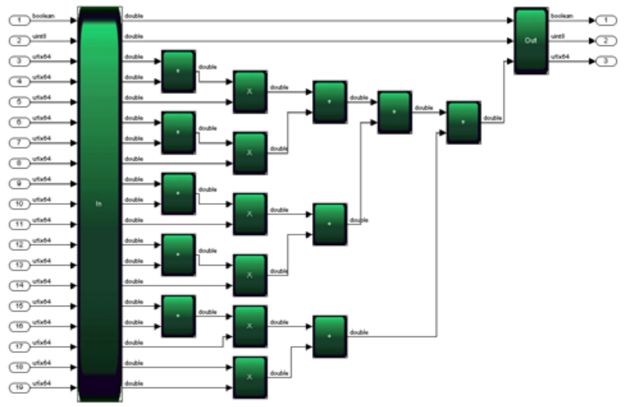

Figure 3: Simple Floating Point Subsystem.

The following data path was designed for a LU decomposition function using Crout's algorithm. The functional blocks in DSPB have been mapped to FPC blocks. The FPC has restructured the data path to avoid overflows and balanced it locally, which is done independently of the DSPB optimizations.

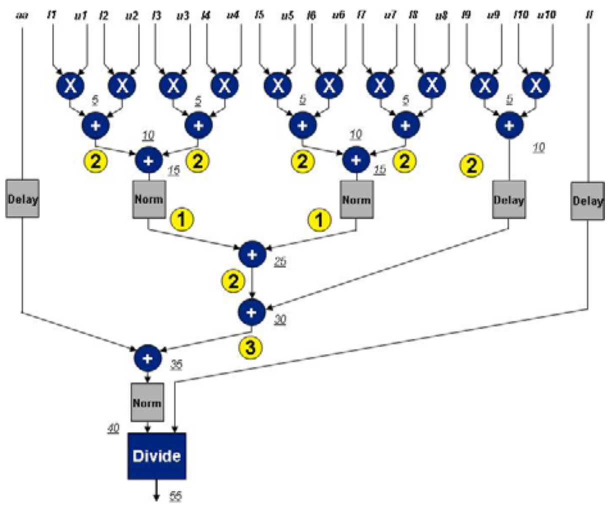

Figure 4: LU Decomposition Data path.

The estimated core logic figure is based on a standalone single precision multiplier and adder/subtractor pair (requiring approximately 650 LUTs and 750 registers [3]) implementation of the dot product portion of the data path. The vector logic also includes the divider; savings in excess of 50% logic area are realized.

Table 1: LU Data path Resource Requirements

Matrix Size	Logic	DSP	Vector Logic	Core Logic
12x12	5197/8652	75	4587/7855	7800/9000
64x64	21457/27346	283	20681/26004	41600/48000

Roadmap

Currently the set of optimizations employed by the FPC to obtain the sustained system performance while minimising the hardware resources used is applied separately to those optimizations applied by DSP Builder. Future versions could integrate these optimizations: first applying the DFG transformations to build and apply floating-point specific transforms; then on this transformed DFG, applying further DSP Builder optimizations transforms, such as time-division multiplexing, and duplicate code removal. Blocks could then be mixed within subsystems, and in fact the block libraries largely combined wit the type of generated hardware just dependent on the data type chosen in the schematic.

References

[1] Altera Corporation, "DSP Builder User Guide, 2009"

[2] M. Langhammer, "Floating point data path synthesis for FPGAs", *Proc. Field Programmable Logic Conf.*, September 2008, pp. 355-360

[3] Altera Corporation, "*Altera Floating Point Megafunctions*", www. altera.com/products/ip/dsp/arithmetic/m-alt-float-point.html