White Paper

Floating-Point Compiler
Increasing Performance With Fewer Resources

Showing new levels of high-performance, high-density, IEEE754-compliant floating-point applications in FPGAs is the focus of this white paper. A new tool is introduced that will allow 100 percent of the floating-point capability of the FPGA device to be used. Combined with the rich DSP resources and advanced routing fabrics of the most recent Altera® FPGAs, unprecedented performance numbers are shown, including 40 giga floating-point operations per second (GFLOP) double-precision and 90 GFLOP single-precision on 65-nm FPGAs. In addition, with higher system performance, the more optimal balancing of resources will leave sufficient logic and other resources to support full-featured applications based around these new floating- point datapaths.

Introduction

As floating-point applications become more prevalent, acceleration of floating-point operations become more important. New processors are being introduced that have increased floating-point performance, but as the interfaces and architectures are fixed, not all applications can be optimally supported.

FPGAs have always offered an almost unlimited flexibility in dataflow architectures, and are therefore an ideal solution to implement arithmetic functions or to accelerate a system by off-loading a datapath that cannot be optimally implemented in a processor. The complexity and precision of floating-point operations have previously been difficult to achieve for FPGAs, especially for double-precision applications.

A new floating-point compiler (FPC) has been developed to map floating-point datapaths to generic FPGA architectures efficiently. The efficiency of the FPC gains is achieved by fusing together large subsections of a datapath, by clustering similar operations together, and by optimizing the interface between clusters of dissimilar operators.

This also allows multiple precisions—integer, single, and double—to exist within a single datapath. This flexibility gives generic FPGAs a significant efficiency advantage over heterogeneous FPGA architectures. With typical logic savings in the datapath of 50 percent, combined with a similar reduction in latency, generic FPGAs can now easily support floating-point capability with the flexibility to implement a wider range of operator mixes (such as a larger ratio of adder/subtractors to multipliers), and still processing power to support an application using a datapath.

Floating-Point Datapath Compiler Overview

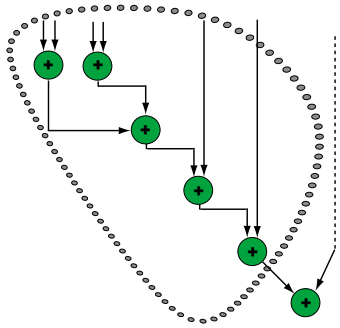
The FPC takes a C-language description of a function and converts it to a parallel datapath implementation. The C description contains no parallel information. It analyzes the expressions and variables, determines the inputs and outputs, and creates a dataflow graph of the internal operations. A C program defining a dot product (used by a matrix multiply application such as DGEMM) is examined here:

```
double x01, x02, x03, x04, x05, x06, x07, x08;
double x09, x10, x11, x12, x13, x14, x15, x16;
double x17, x18, x19, x20, x21, x22, x23, x24;
double x25, x26, x27, x28, x29, x30, x31, x32;
double x33, x34, x35, x36, x37, x38, x39, x40;

double c01, c02, c03, c04, c05, c06, c07, c08;
double c09, c10, c11, c12, c13, c14, c15, c16;
double c17, c18, c19, c20, c21, c22, c23, c24;
double c25, c26, c27, c28, c29, c30, c31, c32;
double c33, c34, c35, c36, c37, c38, c39, c40;

double result;
void main()
{
    double mid01, mid02, mid03, mid04, mid05, mid06, mid07, mid08;

    mid01 = (x01*c01 + x09*c09) + (x17*c17 + x25*c25) + (x33*c33);
    mid02 = (x02*c02 + x10*c10) + (x18*c18 + x26*c26) + (x34*c34);
    mid03 = (x03*c03 + x11*c11) + (x19*c19 + x27*c27) + (x35*c35);
    mid04 = (x04*c04 + x12*c12) + (x20*c20 + x28*c28) + (x36*c36);
    mid05 = (x05*c05 + x13*c13) + (x21*c21 + x29*c29) + (x37*c37);
    mid06 = (x06*c06 + x14*c14) + (x22*c22 + x30*c30) + (x38*c38);
    mid07 = (x07*c07 + x15*c15) + (x23*c23 + x31*c31) + (x39*c39);
    mid08 = (x08*c08 + x16*c16) + (x24*c24 + x32*c32) + (x40*c40);

    result = ((mid01 + mid02) + (mid03 + mid04)) +
        ((mid05 + mid06) + (mid07 + mid08));
}
```

The FPC creates a more efficient datapath implementation than can be achieved using individual cores. There two main principles employed are a change in the number representation and formats and the removal of redundant normalizations across a group of operators. The IEEE754 *(1)* format is used on the inputs and outputs of the datapath. Most C types are supported and float both double and integer values, which can be mixed within a function.

In a local cluster of operators, the maximum word growth is deterministic, and word shrinkage, due to bit cancellation, can be estimated. The fractional-format IEEE754 numbers are converted to real numbers with a larger precision to handle local word growth and shrinkage. Within a cluster, all results are kept un-normalized, if possible. Based on the type of operations, the datapath compiler has several options for inserting normalization operations, but in many cases, normalization will only have to be performed out of a local cluster (as shown in Figure 1) or out of the datapath.

Figure 1. DFG Example of a Local Cluster



When considering single precision, the alignment and addition/subtraction of signed numbers varies little in size and speed with increasing precision. As 36-x36-bit multiplies are supported by Stratix® series FPGAs natively and efficiently, a 36-bit mantissa is used, which allows precision to be maintained with un-normalized numbers.

A more aggressive normalization approach must be used to support double precision. Stratix series FPGAs support 54x54 multiplication efficiently, but larger multipliers are not easy to support. Normalization is required for a multiplier, but numerous special conditions in the relationship between datapath nodes can reduce these normalizations.

By allowing the mantissa and exponent values to increase or decrease outside their normal ranges within larger fields than specified for IEEE754, only the alignment need be performed in every operation. By sharing normalization between a cluster of operators, logic and latency are reduced. However, special conditions, particularly overflow and underflow, can also be combined across a cluster, although explicit overflow and underflow conditions input to a cluster must be forwarded through the cluster and their effect accounted for at the normalization point.

Deviation from the IEEE754 standard within a cluster will have a very minor impact on the differences between the datapath result and a sequential simulation of the function in software. The higher internal precision of the compiler-generated datapath improves accuracy. Also, because IEEE754 floating-point operations are non-associative, a datapath constructed from completely compliant cores will return a different result from the software simulation. Testing across many applications shows that it is typical to expect the same magnitude of error between the datapath and a sequential mode, as between a core-based datapath and the model, although the errors may be different.

The fast Fourier transform (FFT) “butterfly” shown below is another example of a compiler-optimized design. The multiplication of the radix 4 DFT matrix is handled outside the datapath, which sums the individual multiplies and multiplies the result by a twiddle factor. The individual radix 4 multiplies consists of ±1 and ±*j*, which can easily be performed using multiplexers and selectively inverting the sign bits of the IEEE754 inputs.

```
double realina, imagina;
double realinb, imaginb;
double realinc, imaginc;
double realind, imagind;
double realtwid, imagtwid;
double realout, imagout;

void main()
{
    double realdft, imagdft;

    realdft = (realina + realinb) + (realinc + realind);
    imagdft = (imagina + imaginb) + (imaginc + imagind);

    realout = (realdft * realtwid) - (imagdft * imagtwid);
    imagout = (realdft * imagtwid) + (imagdft * realtwid);
}
```

Floating-Point Compiler Design Considerations

Several of the potential problems associated with the use of standalone floating-point cores in system design follow. Each of these problems can be reduced greatly by using the FPC. Addressing these design considerations with the FPC enables designers to optimally use the advanced features and density found in Stratix FPGAs.

- *Balance:* Stratix devices are designed to support typical floating-point datapaths, where the ratio of multiplies to additions is approximately even. However, it is exacting to implement many types of algorithms where the additions outnumber the multipliers.
- *Latency:* This can be a design concern for large, complex datapaths. The complexity of each individual floating-point operation requires that FPGA implementations have deep pipelines to achieve good performance; if an application has multiple, deep floating-point operations, the latency through the datapaths may grow to hundreds of clock cycles.
- *Performance:* The logic resources required by the floating-point operators are very large. The 64-bit wide buses typically found in double-precision applications can be particularly demanding for the FPGA routing fabric to support. When a large datapath is fit into the device, performance can drop dramatically. The higher the device resource utilization, the harder it is to route the design and meet performance.
- *Development time:* For some applications, particularly financial modeling, it is important to design and implement the application very quickly. For larger FPGAs, a complex design with high resource utilization can take hours or even days to route. The smaller the design and the fewer the number of wide buses used, the quicker the routing and fitting of the design.

IEEE754 Cores

Generally, the FPC can generate a fused datapath that is much more efficient than the equivalent function constructed from individual cores. Occasionally, single cores (as seen in Table 1) can be better, especially when the datapath is very small.

Table 1. Single-Precision Cores

Operator	ALUTs	Registers	DSP	Latency	Performance
Add/Sub	533	496	n/a	10	355 MHz
Multiplier	128	249	4 (18x18)	8	395 MHz
Divider	659	1289	n/a	28	288 MHz
Square Root	407	908	n/a	28	310 MHz

The resource requirements of the cores provide a good understanding of the balance between digital signal processing (DSP) and soft logic resources. Based on a typical 1:1 ratio of multiplies to addition/subtractions, a double-precision functional pair (Table 2) will require about 2000 look-up tables (LUTs) and slightly more than 2000 registers.

Table 2. Double-Precision Cores

Operator	ALUTs	Registers	DSP	Latency	Performance
Add/Sub	1249	1459	n/a	14	310 MHz
Multiplier	703	897	9 (18x18)	11	340 MHz
Divider	2775	6323	n/a	57	212 MHz
Square Root	1881	3579	n/a	57	201 MHz

For both single- and double-precision applications, multipliers and adder/subtractors can be clocked at over 300 MHz, but be derated for a datapath comprising multiple operators. The larger the function, the greater the performance degradation. The performance advantage gained by using the FPC is a 50 percent reduction in logic use for many of the datapaths. The smaller datapaths will not only ease the fitting of a large system containing the datapath, but the datapath will be much easier to fit to a higher performance level.

Not all available cores are shown. Slightly slower versions of the core or those not truly IEEE754-compliant (such as 1 *ulp* error) can require up to 20 percent less logic resources. However, a large system designed with the lower performance cores may not be slower than using the faster cores.

Devices

Altera Stratix II and Stratix III devices are designed to provide a mix of DSP (hard-logic) and soft-logic resources that are well balanced for floating-point implementations. For the 90-nm Stratix II family, the most common choice for implementing large floating-point datapaths is the EP2S180 (Table 3).*(2)*

Table 3. Stratix II EP2S180 Device Resources

ALUTs/Registers	Multipliers (18x18)	Multipliers (54x54)	Memory
143,520	384	42	9.3 Mbytes

Up to 42 double-precision multipliers can be implemented with the hard multiplier resources in the device. With a typical datapath requiring a 1:1 ratio of multipliers to adder/subtractors (implemented in ALUTs), the EP2S180 can support the entire design of 84 double-precision floating-point functions, including required interfaces and control logic. In practice, the 70 percent device utilization of double-precision operations will only allow a very simple application to use the datapath, and even then the system performance may be greatly reduced. Redesigning the datapath with the FPC will shrink the design, so that the entire system fits easily. In 90-nm devices, fully utilized system speeds of 225 MHz are possible with a compiled datapath, giving 19 GFLOPs. Alternately, 192 single-precision operations can be supported, or 43 GFLOPs.

For the 65-nm Stratix III FPGA, using the datapath example, the EP3SE260 (Table 4) can theoretically support an entire design with up to 152 double-precision floating-point operators. However, the 90 percent device utilization will likely cause an unsuccessful fit, making system design impossible. Again, using the FPC allows all of the capabilities of the device being used.

Table 4. Stratix III EP3SE260 Device Resources

ALUTs/Registers	Multipliers (18x18)	Multipliers (54x54)	Memory
203,520	768	76 (*1*)	14.6 Mbytes

Note:
(1) 64 54x54 multipliers can be supported using the dedicated 54x54 mode.

In 65-nm devices, fully utilized system speeds of 250 MHz are possible with a compiled datapath. The maximum 152 double-precision operators give 38 GFLOPs, and 96 GFLOPs via the 384 single-precision operators.

Design Example: DGEMM

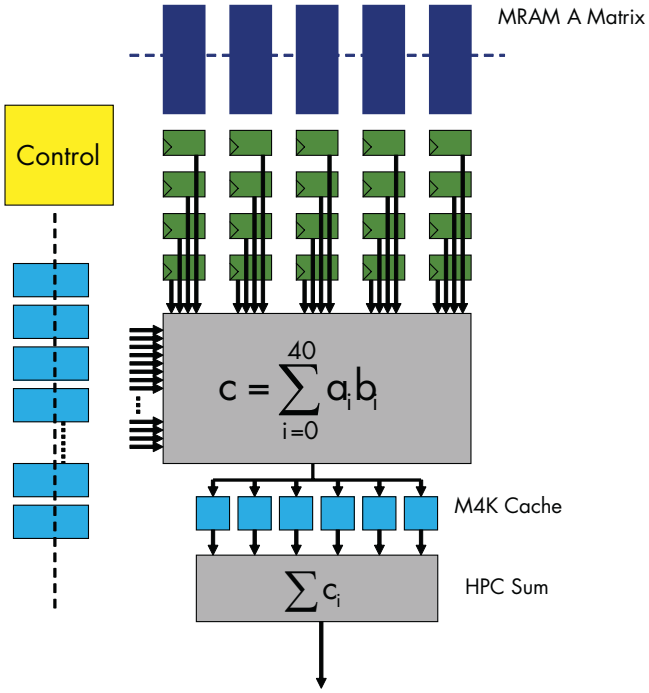
DGEMM is a common building block for many algorithms and is the most important component of the scientific LINPACK benchmark commonly used on CPUs. The Basic Linear Algebra Subprograms (BLAS) include DGEMM in the Level 3 group. The DGEMM routine calculates the new value of matrix *C* based on the product of matrix *A* and matrix *B* and the previous value of matrix *C* (α, β are scalar coefficients): $C = \alpha AB + \beta C$. For this analysis, $\alpha = \beta = 1$ is used, though any scalar value can be used as it can be applied during the data transfer in and out. This operation results in a 1:1 ratio of adders and multipliers.

This FPGA benchmark focuses on the performance from a Stratix II EP2S180 implementation of the AB matrix multiplication with data from a locally attached SRAM. The effort to extend this core to include the accumulator to add the old value of *C* is a relatively minor effort.

In the matrix multiply core, a total of 40 double-precision multipliers are used, with 46 floating-point adders. From the core resource table, the datapath calculated requires 86K ALUTs and 103K registers, if constructed from individual cores. If the datapath is generated using the FPC, then the logic resources drop to 35K ALUTs and 81K registers. The large reduction in logic allows the design to compile in less than an hour, at over 200-MHz system speed, and with no timing constraints set.

As 100 percent of the floating-point capability of a device is used, the internal bottleneck now shifts to the memory bandwidth. A greater memory bandwidth (shown in Figure 2) can be supported by the device, but it requires that a larger percentage of the internal memory is used, and that the *A* matrix is stored in heterogeneous memory architecture.

Figure 2. Block Diagram of the Matrix Multiply Design



The device is theoretically capable of supporting the datapath if constructed from individual cores, but this would be a very difficult design to place and route in the device. Likewise, reducing the memory utilization will create a smaller, easier design to fit in the FPGA. Effective memory bandwidth can be increased by using staging registers to store a portion of the *A* matrix at the inputs of the datapath. The sequence of operations in computing a matrix multiply is:

1. Load *A* matrix
2. Load *B* matrix
3. Stage 4 slices (40 numbers) out of MRAMs (row in *A*)
4. For each column in *B*, calculate a 40-element dot product
5. Store each sum in cache
6. Stage next 40 numbers out of *A* row
7. Multiply by next 40 numbers in *B* columns (repeat until *A* row complete)
8. Sum caches while next *A* row processing starts

With a single HyperTransport™ 2.0 interface, blocking must be used to efficiently compute large matrices. Taking into account the maximum bandwidth of the interface and the staging time for the *A* matrix sections, 88 percent utilization can be achieved in the datapath. A system speed of 200MHz—limited by the HyperTransport interface, not the floating-point datapath—is therefore equivalent to 15.1 GFLOPs. (0.2 GHz * 86 operators * 0.88 load factor = 15.1 GFLOPs). Enough DSP and logic resources remain to implement the scalar multiplies required for the DGEMM calculation.

Conclusion

The floating-point datapath compiler greatly reduces logic requirements and latency by constructing the datapath as a single entity, rather than a collection of individual operators. Many functions within a typical floating-point operation are shared between multiple operators, while accuracy is maintained when a higher precision number representation is used.

Altera's Stratix II and Stratix III devices can efficiently support double-precision arithmetic functions. They are general-purpose devices, but feature some optimizations for high-precision arithmetic calculations. The FPC makes it possible to effectively use those FPGA features to support the most demanding floating-point applications.

Further Information

1. IEEE 754: Standard for Binary Floating-Point Arithmetic
   http://standards.ieee.org/reading/ieee/interp/754-1985.html
2. *Designing and Using FPGAs for Double-Precision Floating-Point Math*:
   www.altera.com/literature/wp/wp-01028.pdf

Acknowledgements

- Martin Langhammer, Principal Scientist, Altera Corporation
- Danny Kreindler, Sr. Strategic Marketing Manager, Applications Business Group, Altera Corporation

101 Innovation Drive
San Jose, CA 95134
www.altera.com