# Native Double Precision LINPACK Implementation on a Hybrid Reconfigurable CPU

Thang Viet Huynh
Graz University of Technology, Graz, Austria
Signal Processing and Speech Communication Laboratory
thang.huynhviet@tugraz.at

Manfred Mücke, Wilfried N. Gansterer
University of Vienna, Vienna, Austria
Research Lab Computational Technologies and Applications
{manfred.muecke|wilfried.gansterer}@univie.ac.at

***Abstract*—Applications requiring double precision (DP) arithmetic executed on embedded CPUs without native DP support suffer from prohibitively low performance and power efficiency. Hybrid reconfigurable CPUs, allowing for reconfiguration of the instruction set at runtime, appear as a viable computing platform for applications requiring instructions not supported by existing fixed architectures. Our experiments on a Stretch S6 as prototypical platform show that limited reconfigurable resources on such architectures are sufficient for providing native support of DP arithmetic. Our design using a DP fused multiply-accumulate (FMA) extension instruction achieves a peak performance of 200 MFlop/s and a sustained performance of 22.7 MFlop/s at a clock frequency of 100 MHz. It outperforms LINPACK using software-emulated DP floating-point arithmetic on the S6 by a factor of 5.7 while achieving slightly higher numerical accuracy. In single precision, multiple floating-point operators can be implemented in parallel on the S6.**

***Keywords*-Hybrid Reconfigurable CPU, Reconfigurable Hardware, Floating-Point, FMA, LINPACK, Power Efficiency.**

## I. Introduction

Scientific and advanced signal processing applications often require computations to be performed in IEEE double precision (DP) arithmetic. DP floating-point units (FPUs), however, occupy considerable chip area, have high latency and are used by a subset of applications only. CPU manufacturers are therefore reluctant to add static DP FPUs to their embedded CPUs and charge a premium for providing DP FPUs or do not include them in low-power devices. Numerical software executed on CPUs lacking support of DP arithmetic has to emulate DP arithmetic. This emulation increases both execution time and power consumption. It would therefore be desirable to combine low-power embedded CPUs with DP FPUs to provide a capable yet energy-efficient computing platform.

Hybrid reconfigurable CPUs represent an off-the-shelf architecture combining a conventional CPU with reconfigurable logic for extending the instruction set, thereby allowing each application to make best use of the available chip area. In contrast to classical accelerators (GPUs, FPGAs) connected to a CPU, hybrid reconfigurable CPUs integrate static and flexible fabrics into a single die or package, resulting in extremely low communication times (few clock cycles) between CPU and reconfigurable fabric. The available reconfigurable resources, however, are typically much more constrained compared to dedicated reconfigurable devices like FPGAs.

This work was funded by the Austrian Science Fund (FWF) under the project NFN-SISE and the Austrian Academic Exchange Service (ÖAD).

Hybrid reconfigurable CPUs combine high productivity (by relying on conventional CPU architecures and software tool chains) with the means to achieve high performance and power efficiency (through the extensible instruction set). They therefore perfectly match requirements of high-performance embedded applications where general-purpose CPUs can not be used due to high power consumption and/or too low performance and FPGAs require prohibitive development effort.

DP arithmetic is known to be especially challenging for implementation on reconfigurable logic as it combines wide (64 bit) operands, multiple wide and small arithmetic operators and complex control paths, leading to both high resource usage and low frequency [1]. While contemporary FPGAs feature sufficient resources to implement multiple DP floating-point operators [2], it is not evident that the same is true for the limited reconfigurable resources of a hybrid CPU. A second issue with DP arithmetic is efficiency of data transfer. On architectures featuring 16 or 32 bit wide data busses, as is the case for most embedded CPUs, handling 64-bit operands in an efficient manner is a considerable challenge. Even general-purpose CPUs can usually not satisfy the memory bandwidth required by a longer sequence of floating-point operations without data reuse.

In this paper, we investigate if the reconfigurable resources typically available on hybrid reconfigurable CPUs are sufficient for providing native support of DP arithmetic. As a prototypical hybrid reconfigurable embedded CPU we use the Stretch S6 [3], [4]. The focus on embedded devices is dictated by the fact that currently no hybrid reconfigurable general-purpose CPU is available. We extend the instruction set of the Stretch S6 with one extension instruction implementing a IEEE-754 [5] compliant DP fused multiply-accumulate (FMA) operation. For performance evaluation, the popular LINPACK benchmark [6] is used. Our aim is to compare the performance of DP LINPACK using the DP FMA extension instruction to the performance of SP LINPACK using the native SP FPU on the Stretch S6, evaluating the hybrid architecture's ability to compete with fully optimised functional hardware blocks of considerable size like a SP FPU while retaining the high accuracy of DP arithmetic.

The scientific contributions of this paper are: ($i$) Demonstra-

tion that DP operations can be implemented within the *limited reconfigurable resources* of a Stretch S6 hybrid reconfigurable CPU; ($ii$) Presentation of DP LINPACK results using a DP FMA extension instruction on S6 showing a performance comparable to the one achieved by using the native SP FPU and ($iii$) presentation of SP LINPACK results using a short-vector SIMD SP FMA extension instruction on S6.

The remainder of this paper is organized as follows. Section II summaries some related work implementing floating-point operations for CPUs and FPGAs. The Stretch S6's architecture is described in Section III. Section IV presents the implementation of the IEEE-754 compliant DP FMA extension instruction on the Stretch S6. Section V presents the performance evaluation of the extension instruction with the LINPACK benchmark. Finally, Section VI summarises the work presented and details future work.

## II. Related Work

While the floating-point performance of selected kernels on dedicated reconfigurable hardware (FPGAs) can be impressive, more control-oriented parts of the application typically require a large development effort in custom logic and are therefore off-loaded to sequential processing units. Available solutions differ mostly in the way reconfigurable logic and sequential processing units are coupled. Hybrid reconfigurable processors represent tightly coupled solutions (see [7] for an overview). Current FPGAs allow for implementation of complex floating-point arithmetic rivalling and outperforming current high-end CPUs [2]. See [1], [8], [9], [10] for an extensive discussion of implementation, performance and trends of floating-point arithmetic on FPGAs.

Chong et al. [11] proposed a flexible multi-mode embedded FPU on a single FPGA, which can be configured to perform various floating-point operations over a variety of applications. Jin et al. [12] added an IEEE-754 compliant single precision FPU to the eMIPS ("extensible MIPS") microprocessor (the eMIPS exists as a soft-core design and was synthesized on a Xilinx FPGA). A single precision (SP) fused operation that integrates 25 basic operations was implemented and evaluated by executing single precision LINPACK. The work of Jin et al. is complementary to our work as they use a soft-core CPU implemented on an FPGA. While this allows for the changes not only in the extension instructions but also in the CPU architecture itself, our approach allows for more realistic absolute performance and power measurements.

In contrast to existing work, we implement native support for DP floating-point arithmetic on a hybrid reconfigurable CPU.

## III. Prototypical Hybrid Reconfigurable CPU

The Stretch S6 [3] is a reconfigurable embedded CPU combining a fixed Tensilica Xtensa LX instruction set architecture with a dynamically reconfigurable Stretch extension unit (see Fig. 1). The 32-bit Xtensa LX core (blue) can run at a clock frequency of up to 300 MHz and the programmable Instruction Set Extension Fabric (ISEF, yellow) can run at

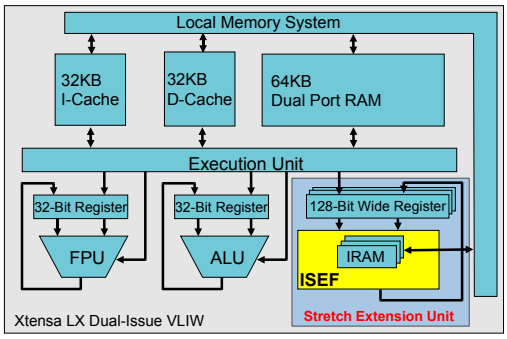


Fig. 1. Stretch S6 Architecture

clock frequencies identical, 1/2 or 1/3 of the Xtensa clock frequency. The Xtensa core is equipped with an IEEE single precision (SP) FPU. DP arithmetic is emulated in software based on the gcc soft float routines.

The ISEF is an array of reconfigurable computational resources, memories, registers and respective interconnect, which can be used to implement user-defined extension instructions. The ISEF's computational resources comprise 4096 arithmetic units (AUs) for addition/logic operations, and 8192 multiply units (MUs) for multiply/shift operations. Between Xtensa core and ISEF, data is transferred via 128-bit *Wide Registers* (WRs). The ISEF supports full pipelining of extension instructions with up to 27 pipeline stages. ISEF fundamental sources of potential performance gains [3] include instruction specialization, spatial- and/or temporal parallelism, and low-latency embedded memory (IRAM).

Application development for the Stretch CPU [3] typically starts with a new or existing C/C++ program running on a sequential CPU platform. The code is profiled to identify the code segments which consume most of the execution time. These identified code segments will then be replaced by user-defined extension instructions which are defined via Stretch C [3], implemented in the ISEF, and invoked from the main program as C intrinsics.

## IV. Double Precision Floating-Point Fused Multiply-Accumulate Extension Instruction

The Xtensa core is equipped with an IEEE single precision FPU only. We want to exploit the ISEF to execute double precision floating-point operations and use the LINPACK as a standard benchmark. The performance of LINPACK relies heavily on the execution of `DAXPY` [6], performing $\mathbf{y} = \alpha \cdot \mathbf{x} + \mathbf{y}$. We chose to improve performance by replacing the DP floating-point operations within `DAXPY` with calls to a fused multiply-accumulate (FMA) extension instruction, thereby circumventing gcc's emulation of DP arithmetic. FMA combines a multiplication followed by an addition (i.e., $Z = X_1 X_2 + X_3$) into an atomic operation with one single rounding only, thereby potentially providing more accurate results and higher performance compared to a sequence of floating-point multiplication and addition. Many algorithms to implement floating-point FMA on digital logic have been presented. In this work, we follow the textbook implementation in [1].

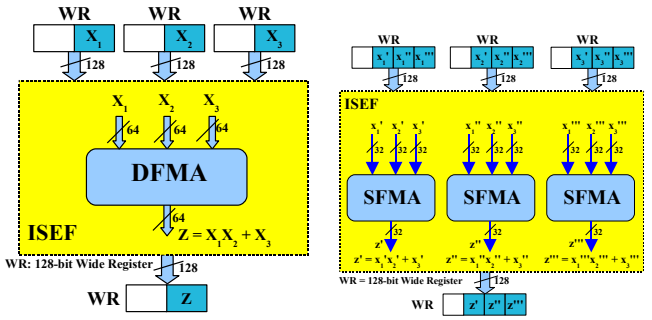


Fig. 2. DFMA & SFMA extension instructions implemented on ISEF

TABLE I
RESOURCES REQUIRED TO IMPLEMENT DFMA AND MULTIPLE PARALLEL SFMA EXTENSION INSTRUCTIONS ($f_{\text{XTENSA}} = f_{\text{ISEF}} = 100MHz$)

| | Available | DFMA | 1 SFMA | 2 SFMAs | 3 SFMAs |
|---|---|---|---|---|---|
| AUs | 4096 | 55% | 19.3% | 42.2% | 62.3% |
| MUs | 8192 | 39% | 7.8% | 15.7% | 23.4% |
| Stages | 27 | 25 | 14 | 16 | 17 |

*DFMA on S6 ISEF:* Using Stretch C, we specified an IEEE-754 compliant (supporting round-to-nearest and normalised numbers) DP floating-point fused multiply-accumulate extension instruction DFMA. The Stretch C compiler (SCC) was able to synthesize the design targeting the S6 ISEF only if both, Xtensa and ISEF were forced to run at 100 MHz, resulting in a DFMA peak performance of 200 MFlop/s. Table I reports the DFMA ISEF resource usage as reported by SCC. Fig. 2 depicts how the DFMA extension instruction is implemented in the ISEF.

*SFMA on S6 ISEF:* Reducing the number format's bit width allows for implementation of multiple parallel floating-point operators in the ISEF. We chose to implement FMA for single precision (SFMA). Table I reports the ISEF resource usage when one, two and three SFMAs are implemented in parallel corresponding to a SFMA peak performance of 200, 400, 600 MFlop/s. The routing resources (not reported in detail by SCC) are exhausted with the implementation of three SFMAs while computational resources for a fourth SFMA are theoretically still available.

## V. EXPERIMENTS

After having shown that DP FMA can be implemented on the ISEF, we are concerned with the question how well a complex extension instruction integrates into an existing application. In the following, we present performance figures for execution of LINPACK on the Stretch S6 hybrid reconfigurable CPU using SP and DP arithmetic and with and without use of extension instructions DFMA and SFMA.

*LINPACK:* The LINPACK benchmark [6] measures how fast a computer solves a dense $N \times N$ system of linear equations $Ax = b$. The LINPACK implementation used in this work is based on a C code available on netlib[1] using Level-1 BLAS routines [6]. We replace the original sequence of DP floating-point multiplication and addition in DAXPY by a single DFMA extension instruction. For SP LINPACK, the SP FPU or the SFMA extension instruction is used.

[1]http://www.netlib.org/benchmark/linpackc.new

TABLE II
LINPACK PERFORMANCE ON STRETCH S6 & DESKTOP CPUS [MFLOP/S]

| System size N | | 100 | 200 | 300 | 400 | 500 |
|---|---|---|---|---|---|---|
| DP emulated S6 | 0.3 GHz | 3.5 | 3.8 | 3.9 | 3.9 | 3.9 |
| **DP DFMA ISEF** | **0.1 GHz** | **18.1** | **20.3** | **21.7** | **22.3** | **22.7** |
| SP on S6 FPU | 0.3 GHz | 67.7 | 63.7 | 64.4 | 64.9 | 65.2 |
| SP 1SFMA ISEF | 0.1 GHz | 28.6 | 27.1 | 28.0 | 28.6 | 29.1 |
| **SP 2SFMAs ISEF** | **0.1 GHz** | **36.3** | **37.3** | **40.3** | **42.3** | **43.7** |
| SP 3SFMAs ISEF | 0.1 GHz | 16.3 | 16.0 | 16.3 | 16.6 | 16.7 |
| DP AMD Opteron | 2.8 GHz | 1654 | 1670 | 1660 | 1580 | 1570 |
| DP Intel Core i7 | 3.2 GHz | 1830 | 1720 | 1570 | 1553 | 1560 |

*Stretch S6 Setup:* Four experiments are set up to profile the LINPACK performance on the Stretch S6. For compiling the code on Stretch S6, we use SCC version 2010.01 (built Feb, 5th 2010) with flag `-stretch-effort10` and `-O3`.

*Desktop CPU Setup:* For comparison, we repeat the measurements on two current desktop CPUs: *1) AMD Opteron 2439*, 2.8GHz 512KB cache, released in 2009; and *2) Intel Core i7 970*, 3.2GHz 256KB cache, released in 2010. The LINPACK code was compiled using gcc version 4.3.2 under Debian 4.3.2-1.1 with optimization flag `-O3`.

### A. Measurements

The estimated number of floating-point operations at system size $N$ is $(2/3N^3 + 2N^2)$ [6]. For every experiment, we measure the total execution time in cycles. The LINPACK performance in floating-point operations per second (Flop/s) is calculated by dividing the number of estimated floating-point operations by the respective LINPACK execution time [6].

Table II reports the performance of DP and SP LINPACK benchmarks achieved on Stretch S6 and on desktop CPUs. DP LINPACK on S6 using software-emulated DP arithmetic at 300 MHz achieves only 3.9 MFlop/s ($N$=500). By providing native DP floating-point arithmetic through the DFMA extension instruction on ISEF running at 100 MHz, the performance of DP LINPACK on S6 is improved significantly to 22.7 MFlop/s ($N$=500), corresponding to a speed-up of 5.8. The desktop CPUs operating at much higher clock frequencies significantly outperform the Stretch S6 hybrid CPU in raw performance (Flop/s).

### B. Discussion

*Cycles per FP Operation (CPF):* To better characterize the efficiencies of different LINPACK implementations running at different clock frequencies, we estimate the average issue rate of floating-point operations, or *cycles per FP operation, CPF*. For system size $N$=500, the DAXPY routine performs about $83833333 \approx 83 \cdot 10^6$ Flops, requiring about $370 \cdot 10^6$ clock cycles. This results in $(370 \cdot 10^6)/(83 \cdot 10^6) \approx 4.4$ clock cycles per floating-point operation. Similarly, we repeat the calculation of *CPF* corresponding to system order $N$=500 for other LINPACK implementations (see Table III).

In terms of clock cycles per floating-point operation, DP LINPACK with DFMA is *comparable* to SP LINPACK with

TABLE III
LINPACK EXECUTION CYCLES & CYCLES PER FP OPERATION (CPF)
(N=500, #FLOP = 83833333)

| | | Exec. cycles [$\times 10^6$] | *CPF* |
|---|---|---|---|
| DP emulated S6 | 0.3 GHz | 6382 | 76.4 |
| **DP DFMA ISEF** | **0.1 GHz** | **370** | **4.4** |
| SP on S6 FPU | 0.3 GHz | 386 | 4.6 |
| SP 1SFMA ISEF | 0.1 GHz | 289 | 3.4 |
| **SP 2SFMAs ISEF** | **0.1 GHz** | **192** | **2.3** |
| SP 3SFMAs ISEF | 0.1 GHz | 502 | 6.0 |
| DP AMD Opteron | 2.8 GHz | 148 | 1.8 |
| DP Intel Core i7 | 3.2 GHz | 170 | 2.0 |

TABLE IV
RESIDUALS FOR LINPACK SOLUTION (N=500)

| | $\kappa(A)$ | $\|\|Ax-b\|\|_\infty$ | $r_N$ | $r_1$ | $r_\infty$ |
|---|---|---|---|---|---|
| SP on S6 Xtensa FPU | 4.6e+2 | 1.42e-03 | 0.04 | 0.04 | 17.34 |
| **SP SFMA on S6 ISEF** | 4.6e+2 | **1.28e-03** | **0.03** | **0.03** | **16.39** |
| DP emulated on S6 | 4.6e+2 | 3.58e-12 | 0.05 | 0.05 | 23.56 |
| **DP DFMA on S6 ISEF** | 4.6e+2 | **2.87e-12** | **0.04** | **0.04** | **18.90** |
| DP on AMD | 4.6e+2 | 3.58e-12 | 0.05 | 0.05 | 23.56 |
| DP on Intel | 4.6e+2 | 3.58e-12 | 0.05 | 0.05 | 23.56 |

native SP S6 FPU (4.4 vs 4.6 *CPF*). SP LINPACK using two SFMAs in parallel in ISEF is *nearly two times faster* than SP LINPACK using native SP S6 FPU (2.3 vs 4.6 *CPF*)

The desktop CPUs are about *twice as efficient* as the Stretch S6 (1.8/2.0 vs. 4.4 CPF). This is most probably due to wider memory buses and larger caches present on desktop CPUs.

*Bottleneck Identification:* Profiling the routine DAXPY for both SP and DP implementations (with and without using extension instruction) on Stretch S6, shows a significant amount of data cache misses due to the fact that the memory required for storing $A$ and $b$ (about 40KB at $N = 100$ for SP LINPACK) is larger than available data cache size on S6 (32KB). Since Level-1 BLAS routines are used for current LINPACK implementation, the algorithm obtains no data reuse [6]. These facts explain the high CPF for Stretch S6. Profiling also shows a big amount of pipeline interlocks when running SP LINPACK using three SFMAs, explaining its low performance compared to other SP benchmarks.

Overall LINPACK performance on Stretch S6 could significantly be improved by making use of the zero-latency DualPort RAM to optimize the data movement from main memory to ISEF, which will be investigated in future work.

### C. Residuals

In accordance with [6], we report three residuals of the LINPACK solutions computed (see Eqn. (1) and Table IV; $N$ is the system size, $\epsilon$ is the relative machine precision). The matrix $A$ is a random matrix generated by the LINPACK code. For reference, we report the condition number $\kappa$ of $A$ using the 2-norm: $\kappa(A) = ||A||_2||A^{-1}||_2$.

$$r_N = \frac{||Ax-b||_\infty}{||A||_1 N\epsilon}; r_1 = r_N \frac{N}{||x||_1}; r_\infty = \frac{||Ax-b||_\infty}{||A||_\infty ||x||_\infty \epsilon} \quad (1)$$

## VI. CONCLUSION AND OUTLOOK

We have shown that the reconfigurable fabric of a Stretch S6 hybrid reconfigurable CPU is able to provide native support for double precision floating-point arithmetic. For single precision, even multiple operators can be implemented in parallel. The main limitation is the requirement to run Xtensa and ISEF at 100MHz in order to implement FMA extension instructions. Our results demonstrate that hybrid reconfigurable CPUs are a viable computing platform for scientific computing and advanced signal processing applications.

Future work will focus on investigating efficient data-transfer mechanisms between memory and ISEF to further improve performance, and on exploiting custom precision floating-point operations to trade accuracy for performance.

## REFERENCES


[1] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.

[2] M. Langhammer and T. VanCourt, "Accelerating floating point DGEMM on FPGAs," in *Proceedings of the 12th Workshop on High Performance Embedded Computing (HPEC 2008)*, 2008.

[3] R. E. Gonzalez, "A software-configurable processor architecture," *IEEE Micro*, vol. 26, no. 5, pp. 42–51, September 2006.

[4] Stretch Inc., *Stretch SCP Programmer's Reference - Version 1.0*, 2007.

[5] "IEEE standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1 –58, 29 2008.

[6] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: past, present and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003.

[7] H. P. Huynh and T. Mitra, "Runtime adaptive extensible embedded processors – a survey," in *Proceedings of the 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. SAMOS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 215–225.

[8] K. Underwood, "FPGAs vs. CPUs: Trends in peak floating-point performance," in *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, 2004.

[9] F. de Dinechin, J. Detrey, O. Cret, and R. Tudoran, "When FPGAs are better at floating-point than microprocessors," in *FPGA '08: Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2008.

[10] D. Strenski, P. Sundararajan, and R. Wittig, "The expanding floating-point performance gap between FPGAs and microprocessors," November 2010.

[11] Y. J. Chong and S. Parameswaran, "Flexible multi-mode embedded floating-point unit for field programmable gate arrays," in *FPGA '09: Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, New York, NY, USA, 2009, pp. 171–180.

[12] Z. Jin, R. N. Pittman, and A. Forin, "Reconfigurable Custom floating-point instructions," in *Microsoft Research Technical Report, MSR-TR-2009-157*, 2009.