Double Throughput Multiply-Accumulate Unit for FlexCore Processor Enhancements

Tung Thanh Hoang, Magnus Själander, Per Larsson-Edefors
Department of Computer Science and Engineering
Chalmers University of Technology
412 96 Gothenburg, Sweden
Email: {hoangt,hms,perla}@chalmers.se

***Abstract*—**
As a simple five-stage General-Purpose Processor (GPP), the *baseline* FlexCore processor has a limited set of datapath units. By utilizing a flexible datapath interconnect and a wide control word, a FlexCore processor is explicitly designed to support integration of special units that, on demand, can accelerate certain data-intensive applications. In this paper, we propose the integration of a novel Double Throughput Multiply-Accumulate (DTMAC) unit, whose different operating modes allow for on-the-fly optimization of computational precision. For the two EEMBC benchmarks considered, the FlexCore processor performance is significantly enhanced when one DTMAC accelerator is included, translating into reduced execution time and energy dissipation. In comparison to the GPP reference, the accelerated FlexCore processor shows a 4.37x improvement in execution time and a 3.92x reduction in energy dissipation, for a benchmark with many consecutive MAC operations.

I. Introduction

General-Purpose Processors (GPPs) have been commonly used at the core of System-on-Chip (SoC) designs, due to the high post-fabrication flexibility that GPP programmability offers. However, because of their generality, GPPs have problems to satisfy strict demands on throughput and energy dissipation in data-intensive computations.

From another aspect, a GPP configuration in an ASIC is not flexible at all, in the sense that its hardware units are hardcoded into the instruction set. Because of the need for instruction standardization in GPPs, functional units are selected and logically interconnected, before the implementation phase starts. This makes it next to impossible to add or subtract functional units in later development phases.

One of the primary goals of the FlexSoC scheme [1] is to provide a relatively simple GPP—the *baseline* FlexCore processor—that can be systematically and efficiently extended with accelerators. The baseline FlexCore processor has the datapath units of a simple five-stage GPP pipeline, but a *flexible* datapath interconnect. During implementation, the FlexCore interconnect can be designed to cover all communication paths—this is the *baseline* interconnect—or, better yet, customized to a number of often used communication paths (obtained from application profiling). Thanks to the instruction scheduling that can make use of the flexible interconnect, a baseline FlexCore datapath [1], [2] that uses a customized interconnect has shorter execution time and lower energy dissipation than more rigid architectures.

The Native-ISA (N-ISA) of the FlexSoC scheme makes it possible to collocate several accelerators with the baseline datapath. While our previous work assumed a baseline FlexCore datapath configuration that, except for a multiplier extension, lacked accelerators, this paper deals with *accelerator integration in a FlexCore processor*. The FlexCore N-ISA also enables fine-grain control over datapath units. In the FlexSoC scheme, the fine-grain control is used to support post-fabrication on-the-fly optimization, on top of the datapath and interconnect customization that can be done during implementation. In this paper, on-the-fly optimization refers to *optimization of computational precision*: We propose the Double Throughput Multiply-ACcumulate (DTMAC) accelerator, whose operating mode is under software control and can be changed during run-time.

Our contributions are summarized as follows:

- We propose the microarchitecture and circuitry for the DTMAC accelerator. With a limited overhead for flexibility, a DTMAC unit can operate on varying operand sizes, depending on the need of the currently executed application, increasing throughput and saving energy. For example, many embedded programs operate on 16-bit data, allowing the 32-bit FlexCore processor to optionally—and with limited overhead—execute two simultaneous 16-bit computations on one 32-bit datapath.
- We suggest a design flow to integrate accelerators with the baseline FlexCore processor to improve performance and energy efficiency for certain applications.
- We evaluate the performance and energy enhancement of the new DTMAC unit in the context of the FlexCore processor, for two different EEMBC benchmarks [3].

The remainder of this paper is organized as follows: Sec. II introduces the DTMAC microarchitecture and makes comparisons to a conventional MAC unit. Next, Sec. III discusses the FlexSoC scheme and illustrates a design flow for integration of any kind of accelerator into the FlexCore processor. In Sec. IV, we not only provide the power, delay, and area evaluation of the DTMAC unit itself, but use an EEMBC benchmark evaluation to show the execution time and energy dissipation advantages in the context of a FlexCore processor. Finally, we conclude our paper in Sec. V.

II. Double Throughput Multiply-Accumulate (DTMAC) Unit

A. Implementation

Designers of embedded integrated circuits today have to deal with strict requirements, especially in Digital Signal Processing (DSP) applications. A general rule-of-thumb says that about 10% of DSP-oriented program code belongs to inner loops and that these loops, when run cyclically, occupy some 90% of the total execution time. To select and integrate an appropriate set of accelerators to perform frequently executed instructions can, thus, significantly increase the overall performance of such embedded circuits. The impact of accelerators on performance of various DSP programs was reported in [4], and we incidently note that Multiply-ACcumulate (MAC) is described as a frequently used operation.

Conventionally a MAC unit is made up of a chain of a multiplier and an accumulate adder, with a pipeline register in between, and an accumulate register for data feedback. Thus, the output of the multiplier is stored in registers in each cycle and is accumulatively added in next cycles. The MAC unit has been widely used not only in modern DSP processors, but also in GPPs to support efficient execution of some demanding applications.

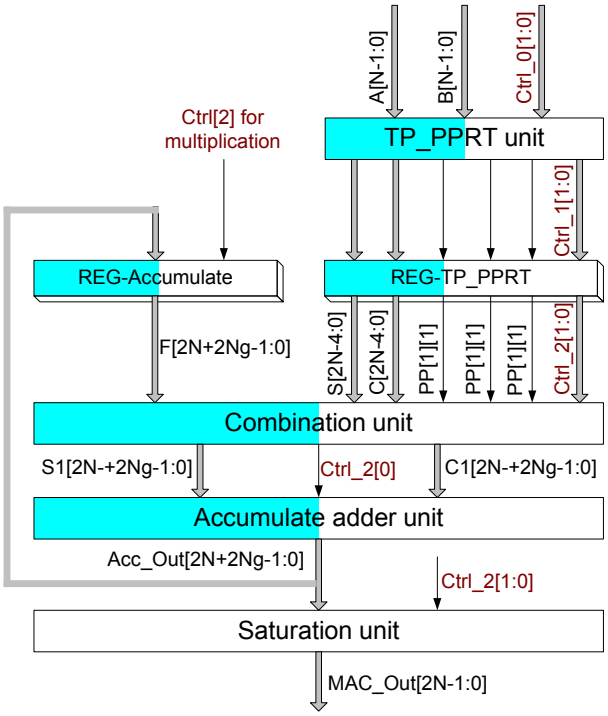


Fig. 1. The general architecture of the proposed Double Throughput MAC for a 32-bit datapath. In this paper, N=32 and Ng=8 denotes the input width and the number of guarding bits, respectively.)

Many embedded applications are based on a 16-bit dynamic range, while the FlexCore processor along with many other embedded processors has a 32-bit datapath. Potentially a 32-bit datapath could accommodate the execution of two simultaneous 16-bit operations. A MAC unit that can optionally switch between N-bit operation and $2 \times N/2$-bit operation is here referred to as a Double Throughput MAC (DTMAC). This feature would be useful in many DSP-oriented applications, e.g. when the dynamic range is lower or when there is a need to simultaneously calculate real and imaginary values.

Obviously, a double throughput 32-bit MAC can be (logically) implemented by tying together two separate, single 16-bit MACs [5]. In this case, two separate 16-bit multipliers are required to support two parallel MAC operations but it comes with a design complexity cost in that the full 32-bit operation is difficult to implement, due to complex connections of the individual multipliers' outputs. FPGA technology offers reconfigurability that can support double throughput MAC [6], but FPGAs are still inefficient in terms of speed and power as compared to ASIC solutions.

Our DTMAC unit in Fig. 1 is designed to support the efficient execution of several operating modes in a 32-bit datapath. The unit employs the twin-precision (TP) technique [7], in terms of a modified 32-bit TP multiplier[1] that contains a Twin-Precision Partial-Product Reduction Tree (TP-PPRT) to generate the partial product outputs, which in conventional schemes are fed to a final adder[2]. Instead we insert a level of adder cells (Combination unit) that combine the outputs of the TP-PPRT with the result of the accumulate adder. In the guarding bit positions of the combination unit, the half adder cells add '1's with the accumulated result, to obtain the correct logical function. The combination unit can be placed after or before the pipeline registers depending on whether the TP-PPRT or the accumulate adder block represents the dominant delay of the DTMAC unit. The use of the combination unit makes it possible to build a high-speed, but still flexible DTMAC unit using only two pipeline stages, which limits the clock load and makes for a power-efficient design.

The accumulate adder is a twin-precision version of a Ladner-Fisher parallel-prefix adder [8] that contains 80 bits, divided in two sections (high and low) each containing 32 data and eight (8) extra guarding bits. Because each of the two sections has eight guarding bits, this DTMAC unit supports loops with 256 iterations without requiring any right shifting of the output to avoid overflow.

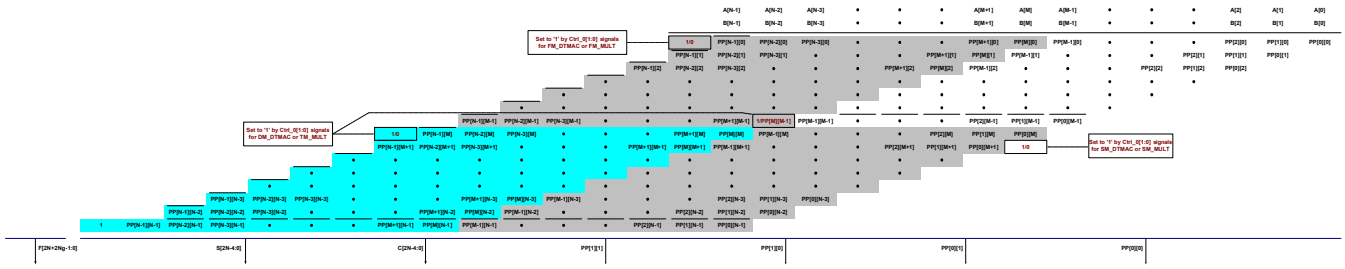
The AND gate is inserted and controlled by one control bit (CTRL2[0]) to set the XOR's input at position 40, to either zero or to the carry signal of the 32-bit data part of the low section of the accumulate adder, depending on operation modes.

The DTMAC unit operates on two's complement data and supports six operating modes (three for MAC operations and three for multiplications), as determined by the value of the 3-bit control signal (CTRL):

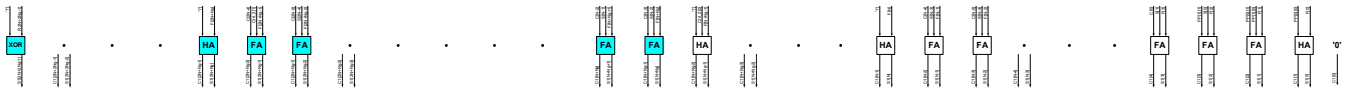
- **000** : Full throughput 32-bit MAC (FM_DTMAC).
- **001** : Single throughput 16-bit MAC (SM_DTMAC).

[1]Here, the term "modified" is used to describe a multiplier whose multiplication is not complete until carry propagation is carried out in the accumulate adder.

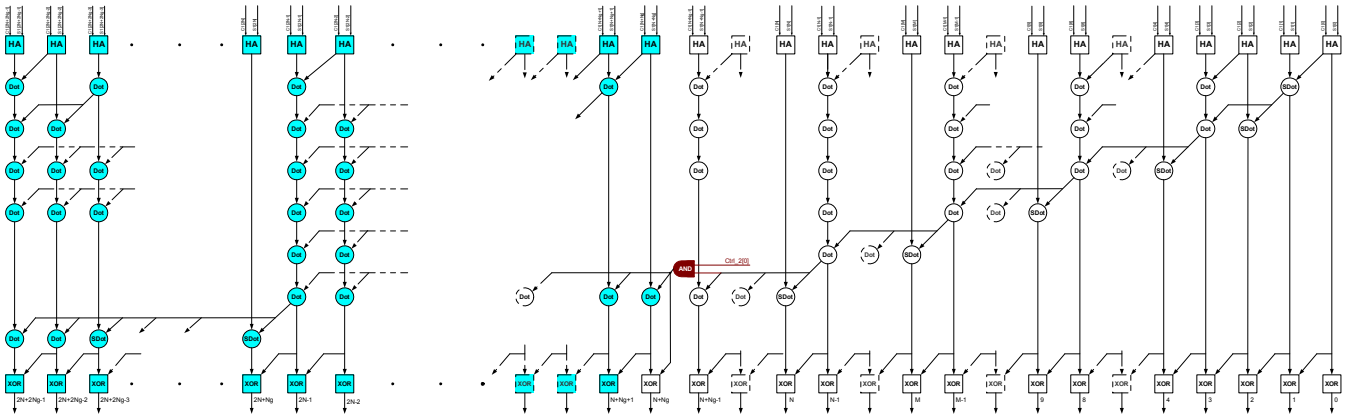
[2]A final adder is based on a high-speed carry propagate architecture to sum the outputs of a PPRT and to generate the final product.



(a) TP-PPRT using Baugh algorithm. (Here, M= N/2.)



(b) Combination unit.



(c) Accumulate adder.

Fig. 2. Structure of the DTMAC's components: (a) TP-PPRT, (b) Combination unit and (c) Accumulate adder. (The cyan and white box denotes computation in most-significant bits (MSB) and least-significant bits (LSB) part, respectively, while grey denotes full-precision computation.)

- **010** : Double throughput 2x16-bit MAC (DM_DTMAC).
- **100** : Single-precision 16-bit multiplication (SM_MULT).
- **101** : Twin-precision 2x16-bit multiplication (TM_MULT).
- **110** : Full-precision 32-bit multiplication (FM_MULT).

B. Evaluation

For the purpose of evaluation, we developed an RTL model of a 32-bit DTMAC as well as conventional 2-cycle 16-bit and 32-bit MACs, for reference. All designs were simulated and synthesized using the Design Compiler tool, for a commercial 1.1-V 65-nm standard-V_T cell library and the same timing constraints. PrimeTime was used to find the critical path delay.

In the proposed DTMAC unit, there exists no final adder. This makes the critical path delay of our 2-cycle DTMAC dominated by the delay of the TP-PPRT part. Our DTMAC actually has the same critical delay as that of a conventional 3-cycle single 32-bit MAC, in which a pipeline register is inserted between the PPRT block and the final adder to sever the critical path of the multiplication. The result is that the DTMAC unit, despite the operating mode flexibility, has small area, low power dissipation and short critical path delays.

When the DTMAC unit operates in SP_MULT mode, half of the respective registers are de-activated to isolate the inputs of half of the accumulate adder and the MSB input bits of the multiplier are set to zero, to reduce switching activity and dynamic power dissipation.

Fig. 3 presents both the delay of individual blocks and the critical path delay for conventional 2-cycle 16-bit (MAC16) and 32-bit MACs (MAC32) as well as a 32-bit DTMAC in different operation modes (SM, DM, and FM). The short critical path delay is an advantage of our DTMAC, as it achieves around 28% shorter delay than MAC32 and around 10.4% shorter than MAC16. If high speed is a design target, the DTMAC unit can be synthesized at 485 MHz, which is 10.2% and 25% faster than MAC16 and MAC32, respectively.

The power dissipation was obtained using 10K random test vectors at the respective top operating frequency and results are shown in Table I.

In terms of energy per cycle, the DTMAC unit operating in single 16-bit MAC mode dissipates only around 2% more than MAC16; the conventional, fixed-function 16-bit MAC. When the DTMAC is performing full 32-bit MAC operations, it has an energy dissipation that is 15.4% lower than MAC32. These

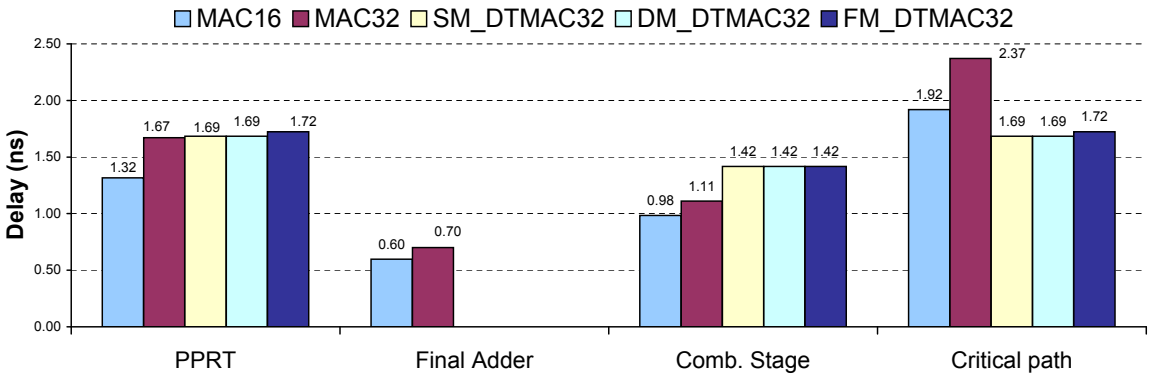


Fig. 3. Delay for different parts of i) a conventional 2-cycle 16-bit MAC, ii) a conventional 2-cycle 32-bit MAC and a 32-bit DTMAC running in iii) single mode (SM), iv) double throughput mode (DM), and v) full mode (FM). Here, the PPRT is of TP type for the DTMAC cases, there is no final adder in the DTMAC cases, and, finally, the Comb. Stage of the DTMAC cases contains the accumulate adder and the combination unit, while for the conventional MAC designs the Comb. Stage contains only an accumulate adder.

TABLE I
DELAY, POWER AND ENERGY DISSIPATION OF 2-CYCLE SINGLE 16-BIT MAC, 2-CYCLE SINGLE 32-BIT MAC AND 32-BIT DTMAC

Design	Timing* (ns)	Power (mW)	Energy (pJ)	Cell area (μm2)
MAC16	2.294(80.7%)	9.50(27.0%)	21.79(25.4%)	12256(29.4%)
MAC32	2.749(100.0%)	31.18(88.6%)	85.71(100.0%)	40594(97.5%)
SM_DTMAC32	2.060(74.9%)	21.31(32.1%)	23.30(27.2%)	41624(100.0%)
DM_DTMAC32	2.060(74.9%)	20.58(58.5%)	42.42(49.5%)	41624(100.0%)
FM_DTMAC32	2.060(74.9%)	35.19(100.0%)	72.49(84.6%)	41624(100.0%)

***: Here, timing is summation of critical path delay and propagation, setup and hold time of in/out registers.**

two comparisons reveal that the implementation of flexibility in the DTMAC unit comes at a limited cost. In fact, energy is saved for the larger operands, thanks to the new micro-architecture and circuits.

The final point of comparison is when the DTMAC is in the DM mode and performs two operations in one cycle. Here, the DTMAC unit dissipates less energy per 16-bit MAC operation than MAC16 does. As we will see in Sec. IV, in the context of a processor, the execution time reduction offered by the double throughput mode allows us to save substantial amounts of energy at the processor level.

Our work is not only concerned with the performance of a stand-alone DTMAC unit, but with the scalability of the FlexCore processor. The flexible interconnect will have to cater for more ports when an accelerator is added, which may degrade performance. Because of the low delay of the DTMAC accelerator, the critical delay of the FlexCore processor mainly is defined by the extension of interconnect ports, not longer from the accelerator delay. In Sec. IV, we will show that the DTMAC integration does not cause any degradation of the FlexCore processor speed, which is in contrast to using for example a conventional 2-cycle 32-bit MAC. But first, in the next section, we will review the FlexCore processor and discuss accelerator integration.

III. FLEXSOC ACCELERATOR INTEGRATION

With the exception of the flexible datapath interconnect, a 32-bit FlexCore processor derives its datapath from a conventional single-issue five-stage MIPS [9]: The baseline FlexCore processor, thus, has a Load/Store (LS), a Register File (RF), a Program Counter (PC) and an ALU unit. Instead of using pipelined control signals, the horizontal Native-ISA (N-ISA) is issued each cycle to control the units and the interconnect of the datapath. The baseline FlexCore architecture as well as some possible accelerators are shown in Fig. 4.

The FlexCore datapath interconnect is based on switch-boxes, whose sizes depend on how many communication paths are supported—a design decision taken during the interconnect customization phase. In this paper, we will refer to two different interconnect configurations resulting from the customization phase:

- **GPP**: The communication paths found in a conventional, rigid MIPS interconnect are used as template for customization of the FlexCore interconnect. This has the effect that the MIPS communication paths are—assuming an ASIC implementation—hardcoded into the switchbox connections and that instructions are scheduled and executed in the same way as in the conventional MIPS datapath.
- **BASELINE**: At the expense of growing hardware, the implementation of all communications paths allows each datapath unit to receive inputs from the output of any unit, including itself.

The two configurations above are defined for the purpose to reason about how to optimize the communication paths in a pipelined GPP architecture relating to, for example, bypass and forwarding techniques in a conventional MIPS. The BASELINE and the GPP interconnect configurations are considered to be the nominal-case and worst-case configurations, respectively, from an execution time point of view. From a power and energy point of view, the best interconnect configuration has fewer paths than BASELINE, but more paths than the GPP configuration. This tailored interconnect configuration would have higher computational efficiency than GPP, but less hardware than BASELINE. As presented in [2], a very energy-efficient tailored interconnect can be designed based

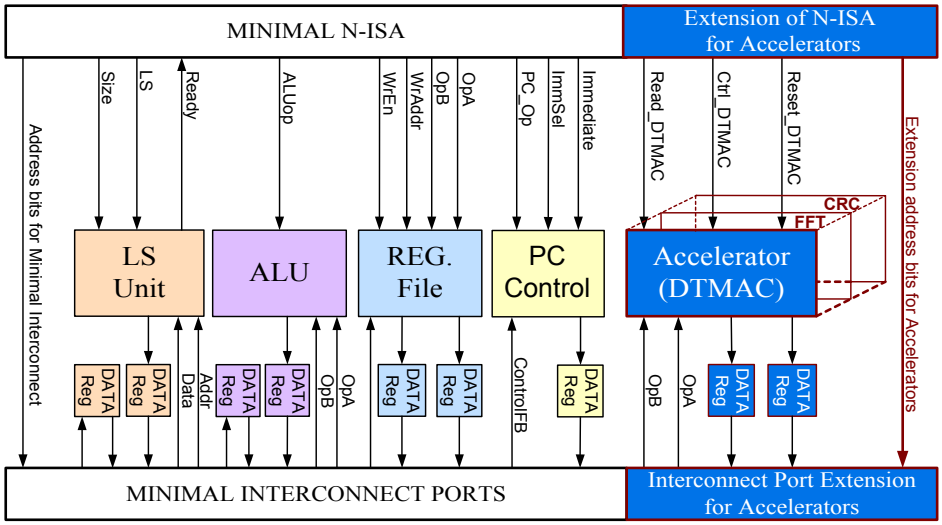


Fig. 4. The baseline FlexCore processor with accelerators.

on statistics of which paths are frequently used, assuming an application or an application domain. This work, however, is *not* considering any tailored interconnect configurations at all.

To support a new accelerator, we need to extend the N-ISA by extra bits. In the case of the DTMAC accelerator, the extra bits are interconnect ports, DTMAC reset, DTMAC stall and DTMAC operating mode bits, see Fig. 5.

(A)	INTERCONNECT (40)		DTMAC (5)			PC (37)	BUF (2)	LS (5)	REGFILE (18)	ALU (5)
DTMAC Add. (8)	Add. bits of BUF(8), PC(4), LS(8), RF(4), and ALU (8)		Stall (1)	CTRL (3)	Enb. (1)					

(B)	INTERCONNECT (15)		DTMAC (5)			PC (37)	BUF (2)	LS (5)	REGFILE (18)	ALU (5)
DTMAC Add. (2)	Add. bits of BUF(2), PC(3), RF(2), and ALU (6)		Stall (1)	CTRL (3)	Enb. (1)					

Fig. 5. The FlexCore Native-ISA that supports a DTMAC accelerator. (a) To support the BASELINE interconnect 112 bits are required, (b) while the GPP interconnect calls for 87 bits.

The N-ISA extension is straightforward: The application code is analyzed to determine which accelerators are useful [4]. The FlexSoC scheme stores compressed instructions in memory, with a subsequent on-the-fly decompression in an instruction decoder that is based on partitioned look-up tables [10]. Thus, a wider N-ISA does not increase memory footprint and instruction bandwidth linearly. Still, extending the N-ISA does not come for free, so accelerators must be critically selected and possible sharing of control signals should be studied.

The evaluation platform that we use for integration of the DTMAC or any suitable accelerator is presented in Fig. 6. The FlexSoC VHDL generator and the cycle-accurate FlexSoC simulator are core functionalities of our platform: After updating the IP library of the FlexSoC generator with a new accelerator and its basic building blocks, such as the DTMAC, it is possible to generate new FlexCore configurations simply by selecting the datapath units and accelerators to be included for this particular configuration and defining the available communication paths. The FlexCore RTL code is then generated given that each accelerator is available as RTL code. Finally, scripts for simulation, synthesis, and (optionally) place and route for the specified CMOS process technology are produced for the back-end phase.

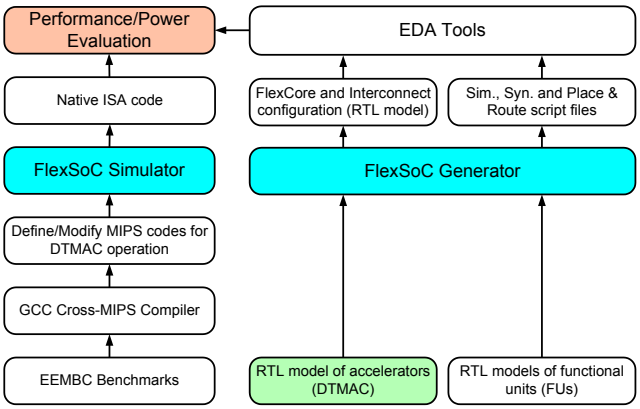


Fig. 6. Evaluation platform of FlexCore accelerator integration system.

IV. Evaluation Using EEMBC Benchmark

A. The Compared Designs

Before discussing evaluation results, we will describe the three designs that we are comparing:

- GPP-MULT : This case is a MIPS-like GPP processor. Since the FlexCore ALU unit only supports the elementary operations (comparison, logic, add/sub), this processor is equipped also with a conventional 32-bit multiplier pipelined in two stages, to execute multiplications. This datapath uses the GPP interconnect configuration of Sec. III.
- BASELINE-MULT : This is a baseline FlexCore datapath extended, as above, with a 2-cycle 32-bit conventional

multiplier, using the BASELINE interconnect configuration.

- BASELINE-DTMAC : This case is a baseline FlexCore datapath extended with the DTMAC accelerator. Since the DTMAC unit has a 32-bit multiplication mode, we can remove the conventional multiplier of the two designs above. This case uses the BASELINE interconnect configuration of Sec. III.

B. Implementation

We synthesized the three different designs by using Design Compiler with a commercial 1.1-V 65-nm standard-V_T cell library and one and the same timing constraint. The synthesis results are reported in Table II.

As outlined in Sec. III, the interconnect configuration can be customized to the communication path utilization. Since BASELINE-MULT and BASELINE-DTMAC utilize the full BASELINE interconnect configuration, the corresponding timing, power and area results that we present in Tables III and IV thus are pessimistic. The execution time, however, is not affected by interconnect customization, as the whole idea of customization is to keep the communications paths that enable efficient computing.

At this point, we can verify that BASELINE-DTMAC can operate at a cycle time of 2.34 ns, which is shorter than the cycle time of a stand alone 2-cycle single 32-bit MAC (2.75 ns). When comparing to GPP-MULT and its cycle time of 2.20 ns, we see that the integration of a flexible DTMAC accelerator into a FlexCore processor does not significantly degrade processor performance.

A 3-cycle solution of the MAC operation—possibly leading to an even shorter critical path—is not considered in this work, because that requires a different MAC operation schedule for the comparison and because that leads to a large power overhead (in the clock load of the extra pipeline registers).

C. Evaluation

Two EEMBC benchmarks, AutoCorrelation (AutoBench) and FFT (FFTBench) [3], are selected to evaluate performance and energy dissipation of the three processor designs, according to the evaluation platform that was previously mentioned in Sec. III. The two benchmarks represent two extremes of computational length of MAC operations—in AutoBench there are many consecutive MAC operations, while FFTBench has short sequences of MAC operations. The evaluation phase is performed as follows:

- **Step 1**: We re-organize the C benchmark code to support double throughput MAC operations: We first locate the sub-routines, where the MAC operations are executed, then we pack two 16-bit operands into one 32-bit operand by using an extra PACK[3] instruction. Finally, we break long loops into two equal and shorter loops to avoid overflow effect when accumulate additions are executed inside the DTMAC unit.

- **Step 2**: The modified C code is compiled into MIPS assembly code by a MIPS cross-compiler based on GNU-GCC [11]. 2x16-bit operand packing instructions (PACK) could be replaced by one load 32-bit instruction for the DTMAC inputs, if two 16-bit values are consecutively stored in memory. This feature can be efficiently exploited in many DSP applications, in which the input values are accessed sequentially.

- **Step 3**: We identify MAC operations in the sub-routines of the MIPS assembly code, and manually perform instruction scheduling to adapt to the FlexCore instruction set.

- **Step 4**: We invoke the cycle-accurate FlexSoC simulator using the modified MIPS assembly code as input. When the benchmarks have been verified to be correct, cycle count values are extracted for both the benchmark core and the instructions inside the inner loop.

We evaluated AutoBench for the three different designs. The results are reported in Table III. In comparison to GPP-MULT, the execution time—the product of cycle count and timing—of BASELINE-DTMAC is reduced 77.1% while BASELINE-MULT "only" achieves a reduction of 33.8%. Regarding energy—the product of execution time and minimum clock cycle time—BASELINE-DTMAC and BASELINE-MULT dissipates 74.5% and 36.2%, respectively, less energy than GPP-MULT. These improvements are very significant, and can be explained by the fact that AutoBench is a highly parallel application in which the 2x16-bit inputs of the DTMAC unit are loaded without needing PACK instructions. With a customized interconnect, the energy reduction would be even more significant.

The results from the FFTBench evaluation are reported in Table IV. Compared to BASELINE-MULT, our BASELINE-DTMAC does not provide any significant reduction in cycle count, execution time and energy. The reason for this is found in the extra PACK instructions that are required for loading FFT coefficients into the DTMAC unit. The computational length of the MAC operations in FFTBench is quite short, so the computational efficiency of a dedicated MAC accelerator is limited. However, we should notice that BASELINE-DTMAC, thanks to the flexible interconnect, still has *an execution time that is 45.1% shorter and an energy dissipation that is 38.9% lower than that of GPP-MULT*.

Fig. 7 summarizes the evaluation results, regarding speed up of execution time and reduction of energy dissipation for the two benchmarks.

V. Conclusion

In the framework of enhancing performance and reducing energy dissipation of embedded processors, we have presented the architectural integration of a new, low-complexity Double-

[3]The PACK instructions are represented by instruction groups of SHR, AND, and OR.

TABLE II
TIMING, POWER AND AREA OF GPP-MULT, BASELINE-MULT AND BASELINE-DTMAC ARCHITECTURES

Datapath configuration	Timing (ns)	Power (mW)	Cell area (μm2)
GPP-MULT	2.20(94.0%)	6.53(89.8%)	54916(86.5%)
BASELINE-MULT	2.34(100%)	6.29(86.5%)	62371(98.3%)
BASELINE-DTMAC	2.34(100%)	7.27(100%)	63458(100%)

TABLE III
CYCLE COUNT, EXECUTION TIME AND ENERGY DISSIPATION OF EEMBC - AUTOBENCH

Datapath configuration	**EEMBC Benchmark**						
	AutoBench Evaluation						
	Cycle count (Core)	Cycle count (Loop)	Execution time (μs- Core)	Execution time (μs - Loop)	Energy diss. (nJ-Core)	Energy diss. (nJ-Loop)	Cell area (μm2)
GPP-MULT	1534(100%)	1372(100%)	3.51(100%)	3.02(100%)	22.89(100%)	19.70(100%)	54916(86.5%)
BASELINE-MULT	992(62.2%)	784(57.1%)	2.32(66.2%)	1.83(60.8%)	14.60(63.8%)	11.54(58.6%)	62371(98.3%)
BASELINE-DTMAC	343(21.5%)	259(18.9%)	0.8(22.9%)	0.61(20.1%)	5.84(25.5%)	4.41(22.4%)	63458(100%)

TABLE IV
CYCLE COUNT, EXECUTION TIME AND ENERGY DISSIPATION OF EEMBC - FFTBENCH

Datapath configuration	**EEMBC Benchmark**						
	FFTBench Evaluation						
	Cycle count (Core)	Cycle count (Loop)	Execution time (μs- Core)	Execution time (μs - Loop)	Energy diss. (nJ-Core)	Energy diss. (nJ-Loop)	Cell area (μm2)
GPP-MULT	58149(100%)	44032(100%)	127.93(100%)	96.87(100%)	835.11(100%)	632.37(100%)	54916(86.5%)
BASELINE-MULT	36071(62.0%)	21504(48.8%)	84.41(66.0%)	50.32(51.9%)	530.96(63.6%)	316.51(50.1%)	62371(98.3%)
BASELINE-DTMAC	30000(51.6%)	18368(41.7%)	70.20(54.9%)	42.98(44.4%)	510.35(61.1%)	312.47(49.4%)	63458(100%)

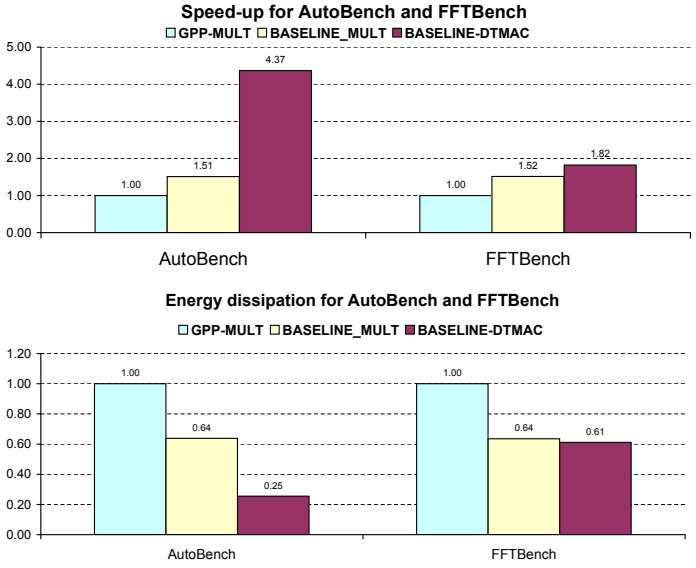


Fig. 7. Overview of speed improvement and reduction in energy dissipation.

Throughput MAC (DTMAC) unit in the FlexCore processor. We showed that we can insert a highly flexible circuit such as the DTMAC accelerator to enable run-time processor optimizations, without any detrimental effect on the performance of the FlexCore processor. Rather, for the applications that can make use of MAC acceleration and run-time optimization of computational precision, the execution time and energy dissipation can be significantly reduced.

REFERENCES

[1] M. Thuresson, M. Själander, M. Björk, L. Svensson, P. Larsson-Edefors, and P. Stenstrom, "FlexCore: Utilizing Exposed Datapath Control for Efficient Computing," in *Proc. of IEEE Intl Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation (IC-SAMOS)*, 2007.

[2] M. Själander, P. Larsson-Edefors, and M. Björk, "A Flexible Datapath Interconnect for Embedded Applications," in *Proc. of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, May 2007, pp. 16–20.

[3] "Embedded Microprocessor Benchmark Consortium." [Online]. Available: http://www.eembc.org

[4] D. Liu, E. Tell, and A. Nilsson, "Implementation of Programmable Baseband Processors," in *Radio Design in Nanometer Technologies, Springer Publisher*, 2006, pp. 83–100.

[5] R. K. Kolagotla, J. Fridman, B. C. Aldrich, M. M. Hoffman, W. C. Anderson, M. S. Allen, D. B. Witt, R. R. Dunton, and L. A. Booth, "High Performance Dual-MAC DSP Architecture," *IEEE Signal Processing Magazine*, pp. 42–53, July 2002.

[6] S. Hong and S.-S. Chin, "Reconfigurable Embbedded MAC Core Design For Low-Power Coarse-Grain FPGA," *Electronic Letter*, vol. 39, no. 7, pp. 606–608, April 2003.

[7] M. Själander, H. Eriksson, and P. Larsson-Edefors, "An Efficient Twin-Precision Multiplier," in *Proc. of IEEE Intl Conf. on Computer Design*, October 2004, pp. 30–33.

[8] P. Kogge and H. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans. on Computers*, vol. 22, no. 8, pp. 786–193, August 1973.

[9] J. L. Hennessy and D. A. Patterson, *Computer architecture - A quantitative approach*. Elsevier Publisher, 2007.

[10] B. Gorjiara and D. Gajski, "FPGA-friendly Code compression for Horizontal Microcoded Custom IPs," in *Proc. of ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays*, February 2007, pp. 108–115.

[11] "GNU Compiler Collection." [Online]. Available: http://www.gnu.org/software/gcc/