Octopus: A Flexible, Efficient MPSoC Prototype for Non-linear Interference Cancellation

Luechao YUAN *, Chuan TANG, Cang LIU, Zuocheng XING

College of Computer, National University of Defense Technology, Changsha, 410073, China

Keywords: MPSoC; Tomlinson-Harashima; VLSI Design

Abstract. Many non-linear (NL) interference cancellation (IC) algorithms characterize vector based operations instead of matrix operations, and their performance can be further improved by a proper sorted preprocessing of the channel information, e.g., successive interference cancellation (SIC) and Tomlinson-Harashima precoding (THP) for multi-user multiple-input and multiple-output (MIMO) communication systems. However, on the one hand, application-specific architectures are efficient but not flexible in performing NL IC algorithms; on the other hand, existing single-core based flexible architectures aimed for intensive matrix operations are not efficient when mapping these algorithms. Alternatively, the multiprocessor system-on-chip (MPSoC) architecture can provide better diversity to implement demanding NL algorithms. This paper proposes an efficient and programmable MPSoC prototype to bridge the efficiency and flexibility gap, especially for versatile Gram-Schmidt process (GSP) aided NL IC algorithms. This prototype incorporates several slave processors and one master processor based on the division of computing and control of the mapped algorithms. The slave processor integrates a coarse-grained programmable element (CGPE) with special support for atomic vector operations, while the maser processor is responsible for task schedule and data transportation among slave processors and memories. For demonstration, a tightly coupled sorted QR decomposition (SQRD) aided THP is mapped to the proposed MPSoC, where a speculative dynamic runtime schedule (SDRS) strategy is applied to reduce the computational delay. The synthesis results are presented to show the efficiency and feasibility of our MPSoC prototype.

Introduction

The spatially multiplexed MIMO technology has been widely applied by current wireless communication systems, e.g., the 3GPP LTE and IEEE 802.11ac standards. However, due to the concurrent signal transmissions to multiple users/antennas, it also brings additional interference to the system. Therefore, the adopted interference cancellation (IC) algorithm [1] plays an important role in such systems, e.g., the detection and the precoding technologies at the receiver side and the transmitter side respectively. Moreover, to adapt to the varied application contexts and the fast evolving standards, future smart devices demand flexible platforms that are able to support various algorithms and soft update to achieve the best trade-offs between the cost and the satisfaction of the consumers.

The existing IC algorithms could be generally classified into two categories, i.e., the linear IC and the non-linear (NL) IC algorithms, which achieve various trade-offs between the performance and the complexity. Typical linear IC algorithms, e.g., zero forcing (ZF) and minimum mean square error (MMSE) methods, have been implemented on several flexible architectures [2,3] in the literature. As for the non-linear algorithms, although many high performance methods have been proposed for a decade, e.g., THPs [4,5] and SICs [1], few flexible VLSI implementations have been reported. Compared to the linear algorithms, the NL IC algorithms are quite different in two respects. On the one hand, in addition to matrix operations, they involve more vector-based iterative calculations. On the other hand, since the preprocessing of the channel information is a prerequisite, a well selected processing order significantly affects their performance [6].

Unfortunately, most of the existing flexible solutions[2,3] that are aimed for linear IC algorithms cannot well support these new features. The bottlenecks are the lack of parallelism and the implicit

branches caused by the NL IC algorithms. More specifically, the lack of parallelism is resulted in by the fact that the vector-based operations cannot provide enough parallel computational loads, which leads to poor utilization efficiency. The implicit branches induce unpredictable disturbance in the control flow, the data flow and the computation flow of these architectures. To the best of our knowledge, only a few ASIC implementations [7] that incorporate sorting operations have been reported. Most of the architectures [8-10] that calculate QRD didn't consider it.

To address the two bottlenecks, an efficient and programmable MPSoC prototype is proposed in this paper. Compared to the single-core based architectures, the multi-core architectures provide more diversity in terms of space and time to map demanding NL IC algorithms. For example, the control and the computing can be divided, so that the execution and mapping of the two parts can be scheduled flexibly in time and space. Moreover, each core in the MPSoC can be of partially autonomy to further release the interdependency between different tasks. We first design a coarse-grained programmable element (CGPE), which acts as the fundamental element of the slave processor to achieve high computational efficiency when performing GSP and THP. Afterwards, to support efficient branch control and memory access, a control-oriented two-way very long instruction word (VLIW) master processor is proposed. Finally, several customized multi-bank memory systems and FIFO buffers are utilized to serve data for the MPSoC system. As an exemplary case, we implement the demanding SQRD aided Tomlinson-Harashima precoding (SQRD_THP) algorithm on the proposed MPSoC. As a proof of concept, the VLSI synthesis results are presented to demonstrate the feasibility and efficiency of our prototype.

Algorithm analysis

The block diagram shown in Fig.1 consists of two processing stages: the channel matrix preprocessing and the interference cancellation. The preprocessing stage provides permutation filter (P), feedback filter (B) and feedforward filter (W) for the IC stage. The symbols waiting to be transmitted for different users/antennas are first reordered according to P. Subsequently, the interference caused by the up-layer users is canceled when precoding the low-layer users using B. Finally, the reverse interference from the low-layer users to the up-layer users is suppressed by the feedforward filter W. Consequently, it results in several clean parallel sub-channels for all the users/antennas [11-13].

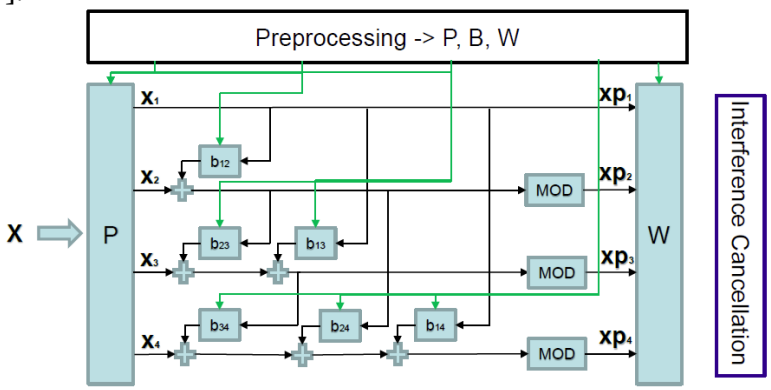


Fig.1. Tight coupled SQRD aided THP

Since the serial nature of the SQRD, the reorder information (P) and the triangular feedback matrix (B) are generated step by step. Meanwhile, the IC is also a serial procedure which mitigates the interference from the uppermost user/antenna to the lowest user/antenna one by one. Therefore, the THP process is able to start whenever necessary information is available before the end of the SQRD processing. Consequently, a proper overlap of the SQRD and the THP could be utilized to feed additional computing load to the empty processing resources. This, in effect, reduces the computational delay significantly.

With a proper decomposition, we found that four computation patterns are enough to accomplish the algorithm. They are vector inner production $h_i^H k_k$, which consists of a series of multiplications and accumulations (MAC); vector/scalar division (VSD), which can be equivalently represented by

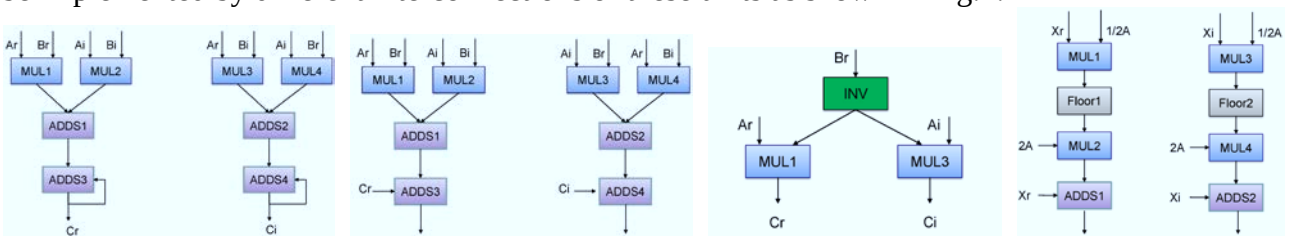
one reciprocal and one vector/scalar multiplication; and orthogonalization $q_k = h_k - r_{i,k} h_i$, which is composed of a series of multiplications and addition/subtractions (MAS); and modulo (MOD), which is defined as

$$MOD_\lambda(x) = x - \left\lfloor \frac{\mathrm{Re}(x)}{\lambda} + \frac{1}{2} \right\rfloor \lambda - j \left\lfloor \frac{\mathrm{Im}(x)}{\lambda} + \frac{1}{2} \right\rfloor \lambda \quad (1)$$

The Octopus MPSoC prototype

Slave-processor

Based on the analysis in the previous section, the design goal of the slave processor is quite straightforward, i.e., to support the four atomic operations efficiently. A closer observation of the four operations shows that the required primitive units are multiplier, adder/subtracter, reciprocal/square-root inverter and floor. To support complex-valued operations, it requires at least four real multipliers, four real adders, two real floors and one real inverter. The four operations can be implemented by different interconnections of these units as shown in Fig.2.



(a) MAC (b) MAS (c) VSD (d) MOD

Fig.2. Atomic computational patterns with support for complex-valued arithmetics

The proposed CGPE consists of all the primitive units to perform atomic operations and several registers to exploit the data locality inside. To provide enough data to feed the CGPE, we design three input ports to support the MAS operation which requires the maximum inputs. The output port is set to be one since it can only produce at most one valid result each cycle. Moreover, we integrate a 16-entry register file (RF) and a four-stage pipeline to assist the CGPE as shown in Fig.3. According to the algorithm analysis, two input ports and one output ports are enough for the slave processor to communicate data with the master processor.

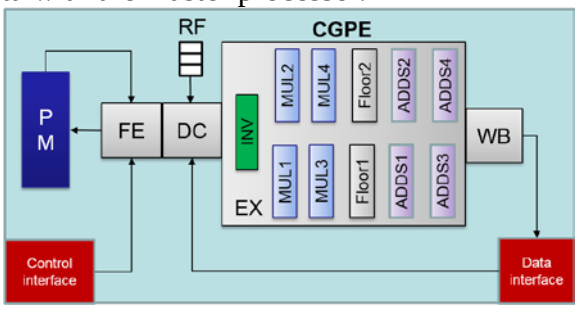


Fig.3. Block diagram of the slave processor

From the view of programming, the CGPE can be considered as a multi-way VLIW like processing element. Namely, all units inside the CGPE are able to work concurrently in the same cycle and the functionality and the interconnection of these units are controlled by instructions. Each atomic operation is completed by a pipeline that consists of several primitive units as shown in Fig. \ref{patterns}. Each primitive unit consumes one cycle to do the calculation and stores the intermediate result to the attached register, which will be the input of the next primitive unit in the pipeline. Since each atomic operation employs only part of the primitive units per cycle, different atomic operations can be overlapped to reduce the computational delay as long as no resource conflict.

Since the four atomic operations are able to cover the computational patterns of most linear and NL algorithms, the interconnection overhead of these primitive units could be much reduced compared to the complex bypassing and forwarding network in [3] without flexibility degradation.

The total number of bits to configure the functionality and interconnection of the CGPE is 36, which is much shorter than the 586 bit-width configuration bits of the CGRA reported in [2]. Since longer configuration bits implies higher programming and verifying effort to map algorithms onto the target architecture, the proposed CGPE could be easier to use for developers.

Primitive units

Considering the iterative nature of the NL algorithms, the primitive units are implemented using floating-point arithmetic to achieve higher numerical precision. To decide the exact numerical precision of these algorithms, we developed a customized floating-point C-library to test various IC algorithms under different modulation schemes. The simulation results demonstrate that 6 exponent bits and 17 mantissa bits are accurate enough to support 64-QAM modulation. For lower-order modulation schemes, less mantissa bits could be used to save power [3]. The exploration of the VLSI design for the reciprocal/square-root inverter shows that the iterative Newton-Raphson method results in the best trade-off among accuracy, circuit area and computational delay. Since the mantissa value lies in $[1,2)$, a proper selection of the initial guess with 3-bit precision could reduce the iteration times significantly. In our design, two Newton-Raphson iterations which consume seven clock cycles are sufficient for both inverters. Note that, the two inverters share the same multiplier and adder resources and the functionality could be configured by one mode bit.

Programming on the slave processor

Fig.4. Instruction structure of the slave processor

The instruction structure of the slave processor is illustrated in Fig.4. The control instructions are used to modify the processor state, including conditional/unconditional jumps, interrupt control and return, while the multi-slot instructions are responsible for the computing tasks. The first slot prepares the data to be consumed in the CGPE from the master processor or the RF; the second slot configures the functionality of the CGPE as explained previously; the third slot stores the results produced by the CGPE to the master processor or to the RF. In total, it requires 53 bits to represent the whole instruction set.

The slave processor is controlled by the master processor through the control interface. If the slave processor is in idle mode, the master processor could send an interrupt vector and a trigger pulse to the control interface. The interrupt vector specifies the functionality that the slave processor will perform and the trigger pulse determines the occasion when the computing should start. Immediately after the interrupt is triggered, the slave processor starts to fetch the program memory and push instructions to the pipeline according to the interrupt vector. During the interrupt service period, the slave processor will signal the master processor as busy and indicate whether the output result is valid or not.

Master-processor

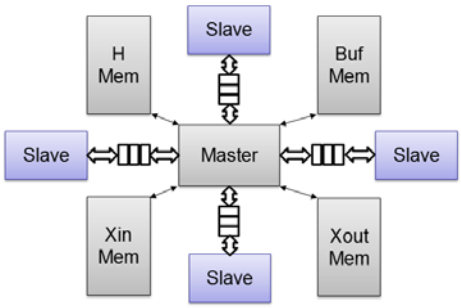


Fig.5. Block diagram of the Octopus

As illustrated in Fig.5, the top-level design of the MPSoC prototype for a 4x4 MIMO system contains two input memories to provide the channel information H and the signals X to be transmitted, one buffer memory to save intermediate results and one result memory to store the final results. The master processor exchanges data with slave processors via several FIFO buffers to alleviate the data pressure and synchronization problems. Since our design is aimed for embedded systems, the H_Mem, Xin_Mem and Xout_Mem should be provided by the external system. For the

completeness consideration, we integrate all the memories to our architecture to obtain a reasonable performance estimation.

Since the computing tasks performed at the slave processors can be easily controlled by sending the corresponding interrupt vectors to each slave processors, the master processor can focus on determining the triggering occasion and providing (collecting) data for (from) slave processors. Therefore, we design two subsystems for the master processor, namely, the memory control system and branch control system, which will be described in following subsections.

Memory system

The THP algorithm requires various memory access patterns. For H_Mem, the master processor has to read 4 elements in one row of the matrix H within one cycle to support 4 slave processors to run the norm and feedforward calculation simultaneously; while there is at most one element will be stored to H_Mem per cycle since the MGS updates only one column in each iteration. Therefore, the H_Mem is designed as a four-bank memory which supports 4-word read and single-word write.

For Xin_Mem and Buf_Mem, assuming that x_i^t nominates the data will be sent by the i-th antenna at the t-th instant. Different IC stages of THP require single-, double- and triple-word access per cycle from $x_{1\ldots3}^t$ to feed slave processors; while at the feed-forward stage of THP, to perform 2D matrix multiplication on 4 slave processors, it requires 4 elements from $x_{1\ldots3}^{t_*}$ each cycle. However, conventional multi-bank memory like H_Mem cannot support both access patterns of $x_{1\ldots3}^{t_*}$ and $x_i^{(1\ldots4)+t_*}$ without bank conflicts. Therefore, we design a customized layout for the Buf_Mem and Xin_Mem as shown in Fig. 6. The entries of each bank are equally-spaced occupied by all $x_{1\ldots4}$ vectors. More specifically, the entry location l of x_i^t in the p-th bank can be calculated by $l = MOD_4\left(p-i+1\right)+4\left\lfloor \frac{t}{4} \right\rfloor$. With this padding design, the layout is able to avoid bank conflicts, no matter whether it accesses multiple elements from the same or different x_i vector.

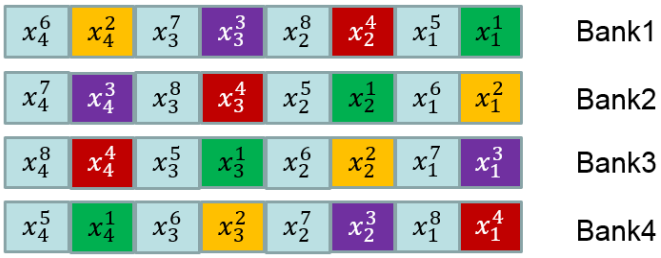


Fig.6. Elements layout of the conflict-free multi-bank memory

To tackle the implicit branches, we design a customized branch control system to fast indicate the master processor where the program counter (PC) should go when norm results are available. Most of the units in this subsystem are untouchable by instruction sets. Firstly, a guard unit will keep monitoring the states of the slave processors. When the norm results are available, the sorting unit will find the minimum norm in two cycles (assuming there are at most four norm inputs). In the third cycle, according to the sorting result, the branch unit will read the address registers, where stores all the possible branch addresses under current situation, and write it to the branch register which will be read by the branch instruction to modify the next PC. Consequently, as soon as the fourth cycle after the norm calculation finish, the master processor is able to schedule the slave processors to do the next computing. Compared to the soft loop to check the states and then do the comparison and finally jump to the target address, the hardware supported branch control saves running cycles and program size.

Programming on the master processor

To deliver enough instruction throughput to serve the slave processors, the master processor is designed using a two-slot VLIW architecture. Besides the control instructions that include branches and returns, all instructions can be combined issued as long as there is no resource conflicts.

Before writing the program code at the master processor side to serve the slave processors, it is worthwhile to check what disturbance will be caused by the sorting operation. Our analysis shows

that the unpredictable sorting operations have three kinds of negative effect on the mapping of SQRD aided THP. Firstly, we cannot determine which tasks should be performed on the slave processors before the end of the sorting operation. As the consequence of the previous uncertainty, we cannot determine the data flow among the slave processors and memories. For instance, the slave processor which corresponds the minimal column norm should export its column to other slave processors to do the next GSP, while this slave processor itself will perform THP or update W. However, we do not know which slave processor should be the selected one. Thirdly, the occasion to trigger the slave processor is also unknown. If we simply wait until the sorting result is available and then schedule the following tasks, it will result in extremely unbalanced workload before and after the available moment of the results. This, in effect, will decrease the utilization of the hardware resources and increase the computational delay significantly.

To address this problem, we propose a speculative dynamic runtime schedule (SDRS) strategy to serve the slave processors in advance of the end of sorting operation according to the possibility. For example, before the first sorting operation in line 5, although we cannot predict which column norm of the H matrix will be the minimal one, we indeed know the other three columns have to perform GSP after this sorting. This fact implies that each slave processor has 75\% possibility to perform the GSP. Therefore, it is rewarding to prepare the necessary data for all slave processors to perform the GSP in advance of the sorting operation rather than just wait. After the sorting result is available, the master processor can trigger three slave processors immediately and then focus on serving the slave processor which corresponds to the chosen minimal column.

This strategy can be further applied to the following sorting operations to save running cycles by overlapping the computation on slave processors and the control on the master processor. Moreover, since the columns that have been chosen by the previous sorting operations will not be changed by the following sorting operation, the slave processors corresponding to these chosen columns have 100\% certainty to determine what to do in the next step even without the knowledge of the sorting results. Nevertheless, the disadvantage of this strategy is that we have to emulate all the possible sorting sequences in the program code, which results in significant increase of the program size.

Implementation results

Table 1 Area and power breakdown of the proposed architecture

Module		Sub module	A.∑ (kGE)	Area (kGE)	P.∑ (mW)	Power (mW)
Octopus	MEMs	-	169.86	-	90.3	-
	FIFOs	-	54.34		36.8	-
	Slave_A	ADDs	51.72	14.21	46.6	17.9
		REGs		13.47		8.5
		MULs		10.24		14.6
		INV		8.50		1.1
		MISC		4.23		4.35
		FLOOR		1.07		0.15
	Slave_B	-	52.52	-	47.3	-
	Slave_C	-	52.09	-	47.1	-
	Slave_D	-	52.11	-	49.8	-
	Master	Mem_Ctr	36.90	9.79	19.9	5.2
		REGs		9.17		6.5
		MUX		8.62		2.7
		FE&DC		4.09		2.4
		MISC		3.69		2.5
		Branch		1.54		0.6

As a proof of concept, the gate-level synthesis results of our MPSoC prototype for a 4x4 MIMO system are presented in this section. The design is synthesized using a 90 nm standard performance CMOS technology under typical conditions with a core voltage of 1V. The frequency is set to be

500 MHz.

Table 1 lists a detailed area and power breakdown of the MPSoC including 4 slave processors, one master processor, FIFOs and necessary memories. As it can be observed, memories and FIFOs occupy 47.6% of the total area; the average core area of the slave processor is 52 kGE, in which the primitive units and registers take up 91.8\%; the master processor consumes 36.9 kGE, in which 73.7% of the area consist of units dedicated to serving data including the Mem_Ctr, Regs and MUX. The MUX is a collection of multiplexers to route datum for/from slave processors from/to memories through the mater-processor. Additionally, the word length of the memories, registers and primitive units are listed in the last column. The registers in the slave processor consist of one 16-entry 48 bit-width RF to save complex-valued local datum and several 24 bit-width registers to save real-valued intermediate results of the primitive units. For the master processor, one 16-entry ordinary 16 bit-width RF and one 16-entry complex 48 bit-width RF are integrated.

To map the SQRD aided THP on our MPSoC architecture, the whole computation is divided into 7 subtasks. Since we implement a MMSE extended SQRD, the number of elements in each column processed by GSP is 8. In order to fill the vacant slave processor and do not cause additional delay, the number of element processed in each THP stage is chosen to be 12, 6 and 4. The total cycle counts of the SQRD without and with THP are 165 and 257 respectively. The percentages of cycles cannot be overlapped due the sorting operations are 22.4\% and 25\% respectively.

The power consumption results are derived based on averaged power analysis of the topographical gate-level models of the processors. Besides the program memories of the processors and the Buf_Mem, the power dissipation of the H_Mem, Xin_Mem and Xout_Mem are also included to get a realistic estimation. The results illustrate that the four slave processors consume 56.5\% of the total power dissipation, in which the power consumed by adders, regs and multipliers take up 88\%. This result is reasonable since the MAC and MAS operations are the most recurring calculations the SQRD aided THP and the registers are used frequently to save intermediate results. The low power consumption of the Inv unit results from its low activity, while the reason for the Floor unit is its low complexity. The power consumed by the master processor takes up 5.9\% of the total power, which is relatively low compared to its area percentage. The power dissipation of Mem_Ctr, Regs and Mux accounts for 72.3\% of the power consumption of the master processor, which matches their area consumption.

Comparison with state-of-art ASICs

Table 2 lists the comparison of the MPSoC prototype with two ASIC designs. To make a reasonable comparison, we exclude the area and the power consumed by memories of our design in this table. The area consumption of the Ref. [8] consists both the ASIP core and the ASIC for THP, and the cycle counts that include the THP processing are calculated by adding up the cycles used to perform QRD and the scaled cycles to process 12 vectors by THP.

As it is shown, the throughput performance of our MPSoC design approaches the performance achieved by ASIC [7], and outperforms the combined ASIP+ASIC architecture [8]. However, the area and the power efficiencies of our design are inferior to the other two designs. The inefficiency is partially caused by the following two facts. Firstly, the floating-point arithmetic consumes more power and area than the fixed-point counterpart when performing the same calculation. Secondly, in order to achieve flexibility and programmability, the integrated master processor and FIFOs consume 25\% cycles, 30\% area and 23\% power in our current design, which can be avoided in the ASIC implementations. Although it consumes more power and area, considering the floating-point units can provide better numerical precision and adaptability for various algorithms, it is worthwhile to use floating-point arithmetic on a flexible architecture. Moreover, as proposed in [3], the numerical aware floating-point technology is able to reduce the power consumption significantly, which can be implemented in our further designs. As an initial implementation of our concept, the flexibility overheads could be further reduced in future iterations. For example, the cycle count could be further reduced by asynchronous triggering of the slave processors instead of synchronous triggering. Additionally, low-power technologies, which has not been adopted yet,

could improve the power efficiency of our design significantly in the future.

Table 2 Comparison with ASIC-based implementations

	ASIC[7]	ASIP + ASIC[8]	This work
Sorting	Yes	No	Yes
Programmable	No	Yes/No[a]	Yes
System settin	4x4 complex	4x4 complex	4x4 complex
MMSE extended	Yes	No	Yes
QRD Method	MGS	CORDIC	MGS
Arithmetic	fixed-point	fixed-point	floating-point
CMOS technology [nm]	180	90	90
Clock frequency [MHz]	162	400/160	500
Area [kGE]	61.8	221.1(92.7+128.4)	299.68
Cycles	104[b]	233[b] /273[c]	165[b] /257[c]
Throughput [M SQRD/s]	3.12[d]	1.72	3.03
Power [mW]	N/A	39.4/9.09	247.5

[a] Ref. [8] require additional ASIC implementation of THP so that the THP part is not programmable;
[b] Cycles per matrix; [c] Cycles per matrix plus THP processing of 12 data vectors;
[d] This value is scaled to 90nm technology.

Conclusion

In this paper, a flexible and efficient MPSoC prototype for non-linear interference cancellation is proposed and evaluated. The slave processor integrates a coarse-grained programmable element (CGPE) with special support for four atomic vector operations, including MAC, MAS, INV and MOD, which achieves both high efficiency and high flexibility. Moreover, the master processor that served as the data and control center for the slave processors provides additional diversity to implement demanding algorithm on the MPSoC. To further reduce the computational delay, the speculative dynamic runtime schedule (SDRS) strategy is applied when mapping the tightly coupled SQRD aided THP. As a proof of concept, our initial VLSI implementation achieves comparable throughput performance to ASICs. Although the flexibility overhead is still inevitable, further improvement could be expected in the future to shrink the efficiency gap between the MPSoC prototype and ASICs.

Acknowledgement

This work was supported in part by the NSF of China (Grant No. 61170083), Specialized Research Fund for the Doctoral Program of Higher Education (Grant No. 20114307110001).

References

[1] N. Miridakis and D. D. Vergados, “A survey on the successive interference cancellation performance for single-antenna and multiple-antenna OFDM systems,” IEEE Communications Surveys & Tutorials, vol. 15, no. 1, pp. 312–335, First 2013.

[2] X. Chen, A. Minwegen, S. Hussain, A. Chattopadhyay, G. Ascheid, and R. Leupers, “Flexible, efficient multimode MIMO detection by using reconfigurable ASIP,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, pp. 2173–2186, Oct 2015.

[3] D. Guenther, R. Leupers, and G. Ascheid, “Efficiency enablers of lightweight SDR for MIMO baseband processing,” IEEE Transactions on Very Large Scale Integration Systems, vol. 24, no. 2, pp. 567–577, Feb 2015.

[4] M. Tomlinson, “New automatic equaliser employing modulo arithmetic,” Electronics letters, vol. 7, no. 5, pp. 138–139, 1971.

[5] H. Harashima and H. Miyakawa, “Matched-transmission technique for channels with intersymbol interference,” IEEE Transactions on Communications, vol. 20, no. 4, pp. 774–780, 1972.

[6] J. Liu and W. A. Krzymien, “Improved Tomlinson-Harashima precoding for the downlink of multiple antenna multi-user systems [mobile radio applications],” in IEEE Wireless communications and Networking Conference (WCNC), vol. 1, 2005, pp. 466–472.

[7] P. Luethi, C. Studer, S. Duetsch, E. Zgraggen, H. Kaeslin, N. Felber, and W. Fichtner, “Gram-Schmidt-based QR decomposition for MIMO detection: VLSI implementation and comparison,” in IEEE Asia Pacific Conference on Circuits and Systems(APCCAS), 2008, pp. 830–833.

[8] K. Shimazaki, S. Yoshizawa, Y. Hatakawa, T. Matsumoto, S. Konishi, and Y. Miyanaga, “A VLSI design of a tomlinson-harashima precoder for MU-MIMO systems using arrayed pipelined processing,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. 96, no. 11, pp. 2114–2119, 2013.

[9] R. C.-H. Chang, C.-H. Lin, K.-H. Lin, C.-L. Huang, and F.-C. Chen, “Iterative decomposition architecture using the modified Gram–Schmidt algorithm for MIMO systems,” IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 57, no. 5, pp. 1095–1102, 2010.

[10] C. Liu, Z. Xing, L. Yuan, C. Tang, and Y. Zhang, “A novel architecture to eliminate bottlenecks in parallel tiled QRD algorithm for future MIMO systems,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. PP, no. 99, pp. 1–1, 2016.

[11] C. Windpassinger, Detection and precoding for multiple input multiple output channels. Shaker, 2004.

[12] R. F. Fischer, “Complexity-performance trade-off of algorithms for combined lattice reduction and QR decomposition,” International Journal of Electronics and Communications (AEU¨), vol. 66, no. 11, pp. 871–879, 2012.

[13] R. F. Fischer, “From gram–schmidt orthogonalization via sorting and quantization to lattice reduction,” in Proc. of the 6th Joint Workshop on Coding and Communications (JWCC), 2010, pp. 13–17.