Recryptor: A Reconfigurable In-Memory Cryptographic Cortex-M0 Processor for IoT

Yiqun Zhang, Li Xu, Kaiyuan Yang, Qing Dong, Supreet Jeloka, David Blaauw, Dennis Sylvester
University of Michigan, Ann Arbor, MI Email: zhyiqun@umich.edu

Abstract

This paper proposes Recryptor, an energy efficient and compact ARM Cortex-M0 based reconfigurable cryptographic processor using in-memory computing. Recryptor is capable of accelerating a wide range of cryptography algorithms and standards, including public/private key cryptography and hash functions, by augmenting the memory of a commercial general purpose IoT processor resulting in a highly compact implementation. The wide bit-width of memory is ideally suited for high bitwidth (64 – 512b) arithmetic operations common in cryptographic functions. Recryptor (28.8 MHz at 0.7 V) achieves 6.8× average speedup and 12.8× average energy improvements over state-of-the-art software and hardware-accelerated implementations with only 0.128 mm^2 area overhead in 40nm CMOS.

Introduction

Security is of utmost concern for Internet of Things (IoT) applications due to the potential pervasiveness of IoT devices. Different applications have different security demands, security algorithms and standards evolve over time, and limited computational resources on IoT platforms drive the need for a flexible and programmable cryptographic processor. Embedded processors tend to have 32-bit datapaths for energy/area reasons, but cryptographic functions can typically be made much more efficient with dedicated hardware support for high bit-width datapaths (64 – 512b). Previous work using ASICs achieve high throughput but are inherently inflexible [1], while cryptographic coprocessors typically have high area and power overhead since they implement an entire processor with fetch, decode, register file and local memory [2,6]. In this paper, we propose Recryptor, an IoT platform that accelerates primitive cryptographic operations by replacing a standard SRAM bank of a general purpose processor with a custom "Crypto-SRAM Bank" (CSB) with in-memory and near-memory computing. Recryptor is based on a 32-bit ARM Cortex M0 processor, which can directly program the CSB in software. We measure Recryptor's speed-up and energy gains on core functions for symmetric and asymmetric cryptography as well as hash functions. Compared with a Cortex-M0 baseline, we achieve energy gains of 9.1× for AES, >6.7× for elliptic curve cryptography (ECC) finite field multiplication and reduction (FFMR) and 4.9× for SHA-3 Keccak function, with energy gains of >4.1× across crypto algorithms relative to the literature.

Energy Efficient Crypto Processor

Recryptor (Fig. 1) is based on an ARM Cortex-M0 processor with 32KB memory. Each of the four memory banks is 8KB, with three implemented using a standard memory compiler while the final bank is the custom designed CSB. The CSB is comprised of sub-banks where a sub-bank of width N supports an N-bit wide single-cycle vectorized operation as well as normal 32-bit memory accesses. Size and placement of the sub-banks were optimized at design time to support a wide range of security operation primitives. Our implementation supports various ECC security levels (163 bits to 409 bits), SHA-3 (1600 bits) and AES (128 bits).

Fig. 2 shows the detailed CSB bitcell and near-memory datapath. Read-decoupled 10T bitcells are used to enable low voltage bitline computation. By selecting different sense-amp read out data, we can read 1 word, compute NOT of 1 word, or compute OR/AND/XOR on two words. Following the readout sens-amps is a compact, wiring-based shifter, which can left shift by 1/4/64 bits (LS1/4/64), right shift by 64 bits (RS64), right rotate 1/8 bits within 64bits (ROT1/8), and shift bytes as required in the ShiftRow and KeyGeneration steps of AES (SRow, KG). This output is one possible choice for writeback data (WData). The other three options are an arbitrary 64-bit rotator, DIN from the arbiter interfacing with the processor, and an AES SBox. The rotator uses 2 stages of 8-to-1 muxes (Fig. 3), where the 1st stage rotates 0~7 bits and 2nd stage rotates in multiples of 8 bits. In order to achieve low energy and stable operation at low voltage, we use transmission gates for the muxes and a negative clock-enabled latch between the two stages to reduce glitch power. By using wire meshes, compact layouts can be obtained for both 1st and 2nd stages. The Sbox is a key byte substitution module used in block ciphers, which uses a 2-stage glitch-free near-memory implementation [1] (Fig. 2). Table 1 shows the normalized area overhead of each custom module; note that the compiled SRAM uses push rules while for simplicity, the custom memory uses standard design rules allowing for future area reduction.

Users can program the Cortex-M0 to use the CSB to accelerate various security algorithms. Two algorithms, López-Dahab (LD) finite-field multiplication and reduction (ECC), and the Keccak function (SHA-3), are shown with their vectorized CSB-based implementation. For LD, we pre-compute reduction related polynomials ($T'(u)$) for better performance, which reduces overflow bits by shifting immediately after multiplication. Table 2 shows the comparison among standard base-line LD code, fixed register implementation [3] and the proposed CSB method; a 9.1× improvement is achieved in terms of number of basic operations. For Keccak, the proposed π' step modifies the intermediate results of each iteration to avoid the matrix transpose in the original π step [4], which would normally require a large number of memory operations. This allows us to exploit the CSB's row-wise vector capabilities for better performance and efficiency. Table 3 shows the operation comparison of baseline code and CSB, which offers a 5.2× improvement.

Programming the CSB requires additional configuration instructions, which add overhead. To reduce this overhead and further improve efficiency, we implement a set of optional FSMs that directly control the CSB through customized control logic. These FSMs incur just 3.6k μm^2 area overhead, and for example, on FFMR-233b, the FSM reduces cycle count from 2336 to 826, providing 2× energy gain. However, to maintain full flexibility, all functions are directly accessible to the Cortex-M0 processor as well. Fig. 4 shows the simulated power breakdown of the custom blocks when performing different security functions. The utilization of the added blocks differs across applications, but power overhead remains low across all. In addition, the M0 is clock-gated during CSB operations, saving up to 6% of total system power.

Measurements & Conclusion

Recryptor is implemented in 40nm CMOS along with a separate baseline Cortex-M0 with four standard memory banks. Fig. 5 shows that the measured maximum frequencies of baseline and Recryptor are comparable across a range of supply voltages. It also compares the energy of running three different functions between Recryptor and the baseline. Table 4 compares the optimal energy and time required for different unit functions on Recryptor, the baseline and other state-of-the-art implementations. Reference [5] is an ASIC design for SHA-3, while [6, 7] are coprocessor designs with limited applications. [3] uses hand-optimized assembly running on a standard Cortex-M0+, but only the 233-bit ECC implementation is provided. Compared to the baseline, Recryptor obtains 8.3×–18.6× runtime improvements and achieves 4.9×–11.8× energy gains for a variety of crypto functions. Compared to state-of-the-art, energy gains are at least 4.1×. Overall, Recryptor achieves 6.8× geometric average speedup and 12.8× energy improvements over baseline and the state-of-the-art. Therefore, Recryptor offers a compelling option for IoT platforms due to its performance, flexibility and efficiency. Fig. 6 shows the die photo.

Acknowledgement

We thank the TSMC University Shuttle Program for chip fabrication.

References

[1] Y. Zhang, et al, VLSI 2016.
[2] J. W. Lee, et al, ISSCC 2013.
[3] R. de Clercq, et al. DAC 2014.
[4] Y. Wang, et al, EDSSC 2015.
[5] P. Pessl, M. Hutter, CHES 2013.
[6] M. Hutter, et al, WISTP 2011.
[7] G. Sayilar, et al, ICCAD 14.

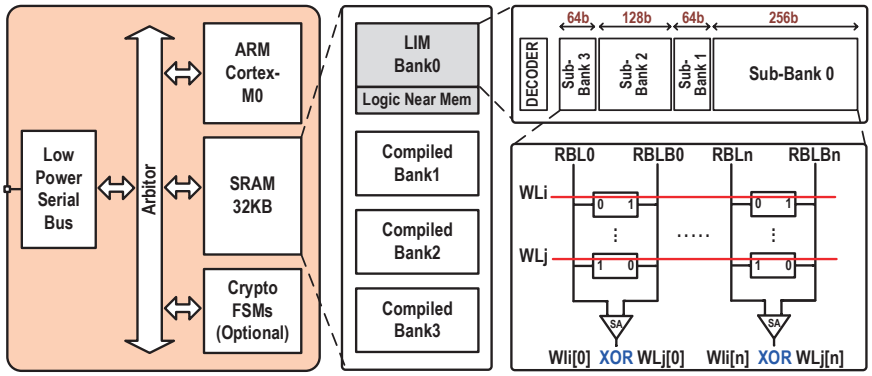

Fig.1. Proposed Recryptor architecture, with ARM Cortex-M0 and 32 KB Memory.

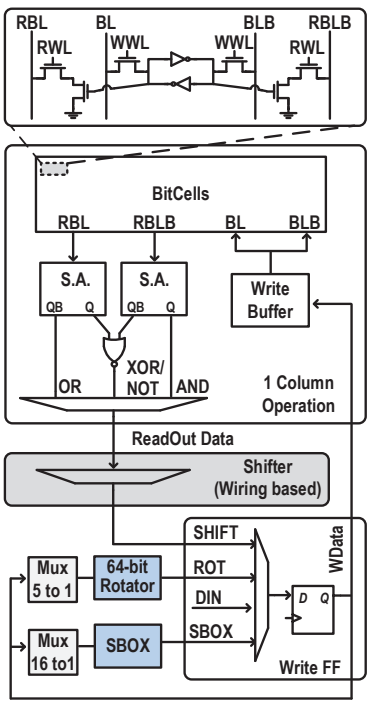

Fig.2. Proposed Crypto-SRAM Bank (CSB).

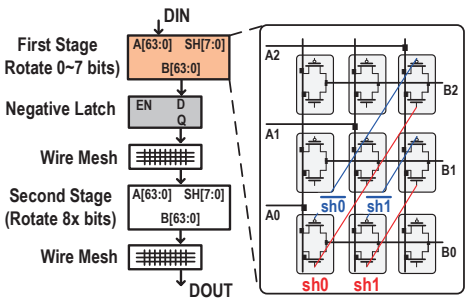

Fig.3. 2-stage 64-bit Rotator

Table.1. Area comparison

8KB SRAM	Norm. Area
Compiled	1x
Custom CSB:	3.26x
Bank	2.63x
Shifter	0.55x
Rotator	0.06x
Sbox	0.02x

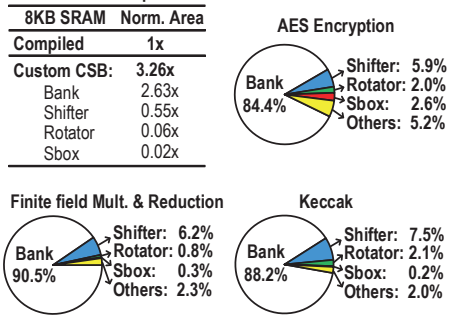

Fig.4. Simulated power breakdown of different security functions

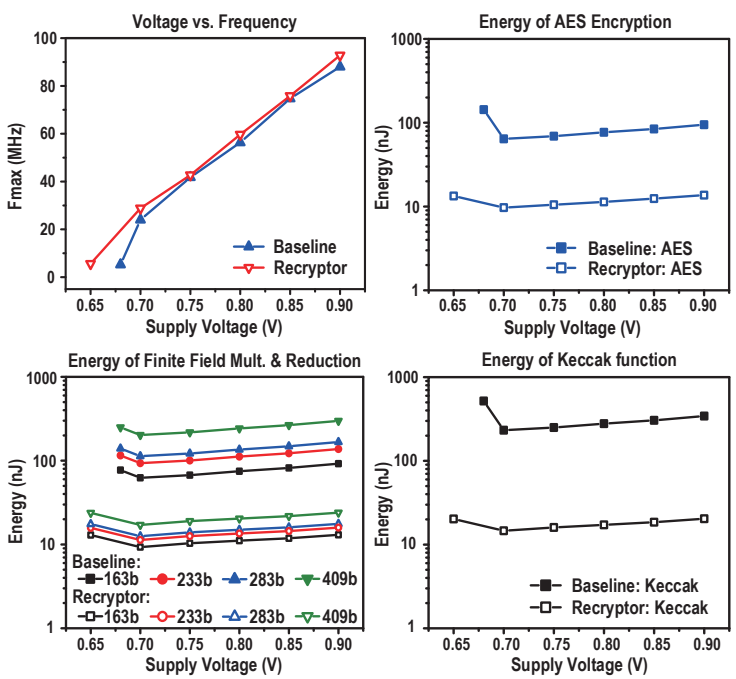

Fig.5. Frequency and energy measurement of Baseline & Recryptor of different applications

Algorithm 1: López-Dahab multiplication & reduction in F_{2^m}

Input: $x = (x_{m-1} \ldots x_0)$, $y = (y_{m-1} \ldots y_0)$, $r = (r_{m-1} \ldots r_0)$
Output: $c = (c_{m-1} \ldots c_0) = xy \bmod r$
[Note: x / y/ r / c is at 1 physical line in CSB]
1: Compute $T(u) \leftarrow uy \bmod r$ for all polynomials u of degree lower than ω
2: Compute $T'(u) \leftarrow ur$ for all polynomials u of degree lower than ω
3: $c \leftarrow 0$
4: **for** $j \leftarrow \lceil m/\omega - 1 \rceil$ **do**
5: $u = (x \gg j \cdot \omega)$ & 0xF
6: $c \leftarrow \text{SHIFT}(c \oplus T(u), \text{LS4})$
7: $u' = (c \gg m)$ & 0xF
8: $c \leftarrow c \oplus T'(u')$ ***[Note: reduction step, c stays to be m bits]***
9: **end for**
10: **return** c

* Syntax SHIFT(***x***, ***y***): apply ***y*** shifts to ***x*** vector using ***Shifter***

Table.2. Estimated required operations for finite field multiplication and reduction in $F_{2^{233}}$

Multiplication					
Method	MEM READ	XOR	SHIFT	MEM WRITE	Total Operations [cycles]*
Baseline	1208	745	315	752	4980
[7]	705	745	315	249	2968
CSB	58	72	62	77	327
Reduction					
CSB	58	72	4	76	268

* Memory operations are assume to require 2 cycles/operation, but for CSB, it is 1 cycle/operation due to direct write-back after read

Algorithm 2: KECCAK-f function

Input: KECCAK[b](S), where $S' = S[0{:}4, y]$ is at 1 physical line, $\forall y \in [0, 4]$
Output: S
1: **for** $i \leftarrow 0$ to $nr - 1$ **do**
2: θ step: $C = S'[0] \oplus S'[1] \oplus S'[2] \oplus S'[3] \oplus S'[4]$
3: $D = \text{SHIFT}(C, \text{LS64}) \oplus \text{SHIFT}(\text{SHIFT}(C, \text{RS64}), \text{ROT1})$
4: $S'[y] = S'[y] \oplus D, \forall y \in [0, 4]$
5: ρ step: read $S'[y]$ in 1 cycle, then $S[x, y] = \text{ROT}(S[x, y], r[x, y])$
6: π' step: $S'[y]$ = **do** SHIFT($S'[y]$, LS64) **for** y iterations
7: ***[Note: π' step result is the transpose of π step in odd iterations]***
8: χ step: $E[y] = \text{SHIFT}(S'[y], \text{LS64})$
9: $S'[y] = S'[y] \oplus (\text{NOT } E[y])$ AND SHIFT($E[y]$, LS64)
10: ι step: $S[0, 0] = S[0, 0] \oplus RC[i]$
11: **end for**
12: **return** S

Table.3. Estimated required operations for Keccak-f (1 iteration)

	MEM READ	XOR/ NOT/AND	SHIFT	MEM WRITE	Rotate	Total [cycles]*
Baseline	172	72	0	60	30	506
CSB	1	14	19	38	25	98

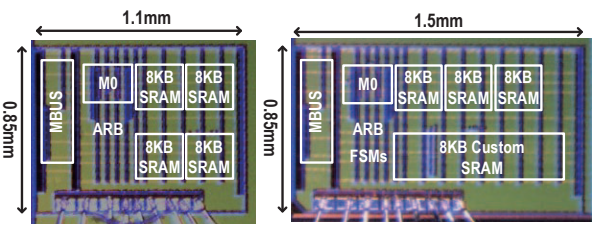

Fig.6. Die photo of Baseline and Recryptor in TSMC 40nm

Table.4. Comparison table of different crypto algorithms & designs

Applications		Designs	#cycles	Freq (MHz)	Time (Norm.) (us)	Energy (Norm.) (nJ)
AES		Baseline	6358	24	265 (1x)	64.2 (1x)
		[6]	5429	0.847	6410 (24x)	10259 (160x)
		[7]	20	1000	0.02 (7.5E-5x)	124 (1.93x)
		Recryptor	726	28.8	25.2 **(0.1x)**	7.05 **(0.11x)**
Finite Field Multiplication + Reduction	163 bits	Baseline	5966	24	249 (1x)	62.4 (1x)
		Recryptor	678	28.8	23.5 **(0.09x)**	9.30 **(0.15x)**
	233 bits	Baseline	8921	24	372 (1x)	93.4 (1x)
		[3]	3672	48	76.5 (0.21x)	45.9 (0.49x)
		Recryptor	826	28.8	28.7 **(0.08x)**	11.3 **(0.12x)**
	283 bits	Baseline	10809	24	450 (1x)	113 (1x)
		Recryptor	916	28.8	31.8 **(0.07x)**	12.6 **(0.11x)**
	409 bits	Baseline	19319	24	805 (1x)	202 (1x)
		Recryptor	1246	28.8	43.3 **(0.05x)**	17.1 **(0.08x)**
Keccak		Baseline	23015	24	959 (1x)	238 (1x)
		[5]	15427	1	15427 (16x)	211 (0.89x)
		Recryptor	3329	28.8	116 **(0.12x)**	48.7 **(0.2x)**

[5,6,7]: Simulation only, no silicon implementation. [6]: 350nm; [7]: 45nm; [5]: 130nm; [3]: No technology given, Cortex M0+ processor; only include mult., no reduction.