KOOKMIN UNIVERSITY

Optimized Implementations of Keccak, Kyber, and Dilithium on the MSP430 Microcontroller

Kookmin University · Queen's University Belfast

DongHyun Shin, YoungBeom Kim, Ayesha Khalid,

Máire O'Neill, Seog Chung Seo

CSE CRYPTO & SECURITY ENGINEERING Lab

Motivation

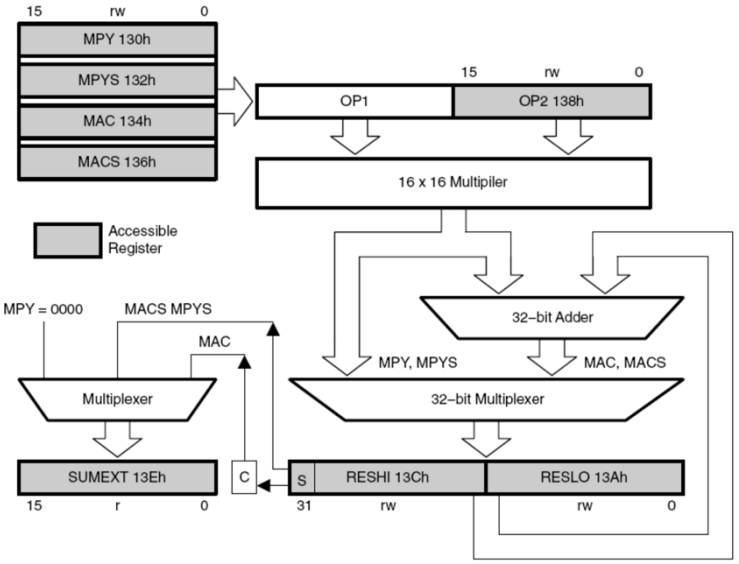
- Rapid advances in quantum computing
- Need to assess the possibility of PQC migration on all existing devices
- Limited PQC Software Implementations on the MSP430
- Existing PQC optimizations mainly focus on 32/64-bit ARM and RISC-V architectures
- Limited research on PQC implementation and optimization for the MSP430
- If these challenges remain unresolved, high migration costs may arise, including processor replacement and additional development costs

Contribution

- **Optimized Keccak implementation methodologies**
  - Review of existing Keccak optimization techniques
  - Proposed Twisting Technique to reduce memory access overhead
  - Proposed Zig-zag Technique for the 2-operand architecture
- **Optimized NTT implementation methodologies**
  - Review of state-of-the-art optimization methods for different architectures
  - Proposed 16-/32-bit modular multiplication and NTT techniques using the hardware multiplier
- **Kyber and Dilithium implementations on MSP430**
  - Kyber: 45.6–62.3% performance improvement over Reference C
  - Dilithium: 44.5–65.0% performance improvement over Reference C

Background

- **MSP430**
  - 16-bit MCU developed by Texas Instruments
  - 12 general-purpose registers (R4–R15)
  - No explicit multiplication instruction; uses a memory-mapped hardware multiplier
    - 16×16-bit multiplication: Typically requires 4 MOV instructions (gpr→multiplier (x2)→gpr (x2))
    - 32×32-bit multiplication: Typically requires 8 MOV instructions (gpr→multiplier (x4)→gpr (x4))
  - Typically equipped with 8–32 KB RAM and 128–512 KB ROM



Profiling on MSP430

- **Keccak (hashing+SHAKE+sampling)**
  - Kyber768: 60.1%-74.1%
  - Dilithium3: 67.1%-86.4%
- **NTT (NTT+iNTT+pointwise multiplication)**
  - Kyber768: 22.2-34.0%
  - Dilithium3: 12.1%-27.9%

Table 1: Profiling: Kyber768 (Ref) on MSP430

Kyber768	All	Keccak	NTT
KeyGen	5 480 k (100%)	4 057 k (74.1%)	1 217 k (22.2%)
Encaps	7 204 k (100%)	5 172 k (71.8%)	1 736 k (24.1%)
Decaps	7 070 k (100%)	4 250 k (60.1%)	2 400 k (34.0%)

Table 2: Profiling: Dilithium3 (Ref) on MSP430

Dilithium3	All	Keccak	NTT
KeyGen	24 638 k (100%)	21 274 k (86.4%)	2 986 k (12.1%)
Sign	35 363 k (100%)	23 729 k (67.1%)	9 853 k (27.9%)
Verify	24 480 k (100%)	19 665 k (80.3%)	4 279 k (17.5%)

KOOKMIN UNIVERSITY

Work 1 : Keccak-f1600

CSE CRYPTO & SECURITY ENGINEERING Lab

Work 1 : Keccak

- **Existing Keccak Optimization Techniques**
  - Applicability and effectiveness depend on architectural features:
    - Instruction set
    - Register size and number of registers
    - Availability of a barrel shifter

Methods[BDH+12]	Platforms	MSP430
Lane complementing	RISC-V†	✗
Lazy rotations	Cortex-M4, Armv8-A	✗
Bit-interleaving	RISC-V‡, Cortex-M4	△
In-place processing	RISC-V†‡, Cortex-M4, Armv8-A	✗

†: RV{32,64}I, ‡: RV{32,64}IB, ✗: Not applicable to MSP430, △: Feasible but suboptimal

Work 1 : Keccak

- **Existing Keccak Optimization Techniques**
  - Lane complementing
    - Reducing inversion operations by complementing selected lanes of the state

    →MSP430 supports the BIC instruction
  - Lazy rotation
    - Uses a barrel shifter to eliminate explicit rotation instructions and merge operations with the next round

    →Effective on architectures with a barrel shifter; MSP430 has no barrel shifter
  - In-place processing
    - Reusing state storage while restoring the original lane layout after multiple rounds

    →Register pressure

Work 1 : Keccak

- **Existing Keccak Optimization Techniques**
  - Bit-interleaving
    - 32-bit architectures: Convert 64-bit rotations into 32-bit rotations through state encoding
    - 16-bit architectures: Extend encoding to (mod 4) for 16-bit rotations
    - Additional encoding/decoding overhead

→Effective on architectures with a barrel shifter

→Optimized 64-bit rotations: Rotations by multiples of 16 can be performed for free

→Optimized 64-bit rotations outperform Bit Interleaving

Table 4: Rotation mapping for efficient ρ step implementation on MSP430. Rotations that are multiples of 16 are free and do not require actual execution. '+' denotes a left rotation, and '−' denotes a right rotation. A rotation of -8 costs 16 cycles, while rotations of ±1 each requires 5 cycles.

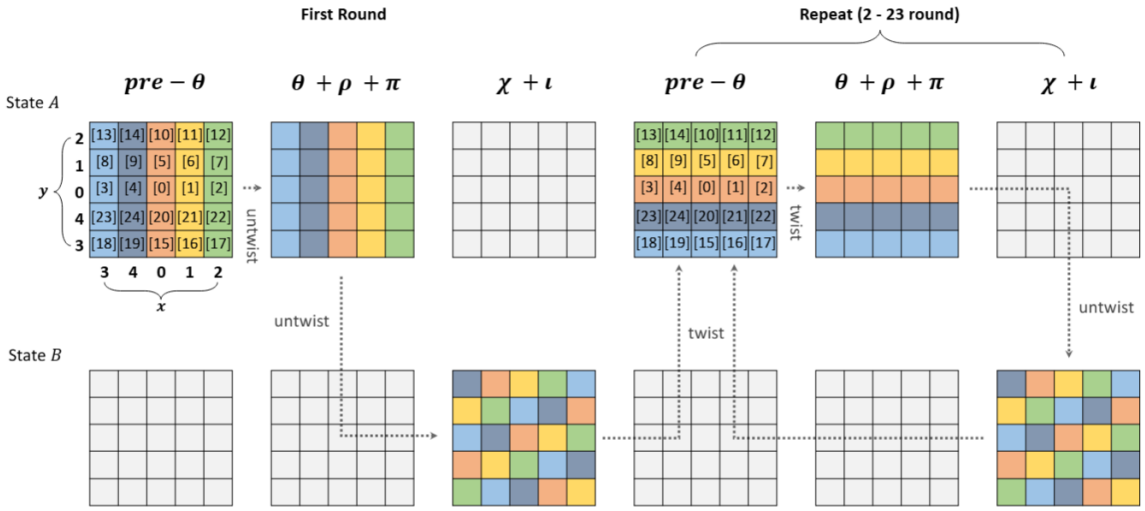
y \ x	3	4	0	1	2
2	$32-\mathbf{8}+\mathbf{1}$ (25)	$48-\mathbf{8}-\mathbf{1}$ (39)	$+\mathbf{3}$ (3)	$16-\mathbf{8}+\mathbf{2}$ (10)	$48-\mathbf{5}$ (43)
1	$64-\mathbf{8}-\mathbf{1}$ (55)	$16+\mathbf{4}$ (20)	$32+\mathbf{4}$ (36)	$48-\mathbf{4}$ (44)	$16-\mathbf{8}-\mathbf{2}$ (6)
0	$32-\mathbf{4}$ (28)	$32-\mathbf{8}+\mathbf{3}$ (27)	$+\mathbf{0}$ (0)	$+\mathbf{1}$ (1)	$64-\mathbf{2}$ (62)
4	$64-\mathbf{8}$ (56)	$16-\mathbf{2}$ (14)	$16+\mathbf{2}$ (18)	$+\mathbf{2}$ (2)	$64-\mathbf{3}$ (61)
3	$16+\mathbf{5}$ (21)	$16-\mathbf{8}$ (8)	$48-\mathbf{8}+\mathbf{1}$ (41)	$48-\mathbf{3}$ (45)	$16-\mathbf{1}$ (15)

- **MSP430 Memory Addressing Modes for Keccak State Operations**
  - Non-sequential memory access: Indexed Mode (3 cycles)
  - Sequential memory access: Indirect Auto-increment Mode (2 cycles)

Table 6-3. Source and Destination Addressing

As, Ad	Addressing Mode	Syntax	Description
00, 0	Register	Rn	Register contents are operand.
01, 1	Indexed	X(Rn)	(Rn + X) points to the operand. X is stored in the next word, or stored in combination of the preceding extension word and the next word.
01, 1	Symbolic	ADDR	(PC + X) points to the operand. X is stored in the next word, or stored in combination of the preceding extension word and the next word. Indexed mode X(PC) is used.
01, 1	Absolute	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word, or stored in combination of the preceding extension word and the next word. Indexed mode X(SR) is used.
10, –	Indirect Register	@Rn	Rn is used as a pointer to the operand.
11, –	Indirect Autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions, by 2 for .W instructions, and by 4 for .A instructions.
11, –	Immediate	#N	N is stored in the next word, or stored in combination of the preceding extension word and the next word. Indirect autoincrement mode @PC+ is used.

Work 1 : Keccak



Algorithm 8 First pre-θ step in twisted state

Input: ptrA

Output: r4, ..., r7 = $C[0]_0$, ..., $C[0]_3$, with C[0] = A[0,0]^ ...^ A[0,4]

```
1: mov @ptrA+, r4; mov @ptrA+, r5;
2: mov @ptrA+, r6; mov @ptrA+, r7;                    ▷ r4...r7 = A[0,0]0:3
3: xor @ptrA+, r4; xor @ptrA+, r5;
4: xor @ptrA+, r6; xor @ptrA+, r7;                    ▷ r4...r7 ^= A[0,1]0:3
5: xor @ptrA+, r4; xor @ptrA+, r5;
6: xor @ptrA+, r6; xor @ptrA+, r7;                    ▷ r4...r7 ^= A[0,2]0:3
7: xor @ptrA+, r4; xor @ptrA+, r5;
8: xor @ptrA+, r6; xor @ptrA+, r7;                    ▷ r4...r7 ^= A[0,3]0:3
9: xor @ptrA+, r4; xor @ptrA+, r5;
10: xor @ptrA+, r6; xor @ptrA+, r7;                   ▷ r4...r7 ^= A[0,4]0:3
```

- **Proposed Technique 1: Twisting Technique**
  - Original state layout requires Indexed Mode for steps with large offsets
  - Store the state based on the access pattern of the next step
  - Reduce memory access overhead by enabling Indirect Auto-increment Mode
  - Merged $\theta + \rho + \pi$ and $\chi + \iota$

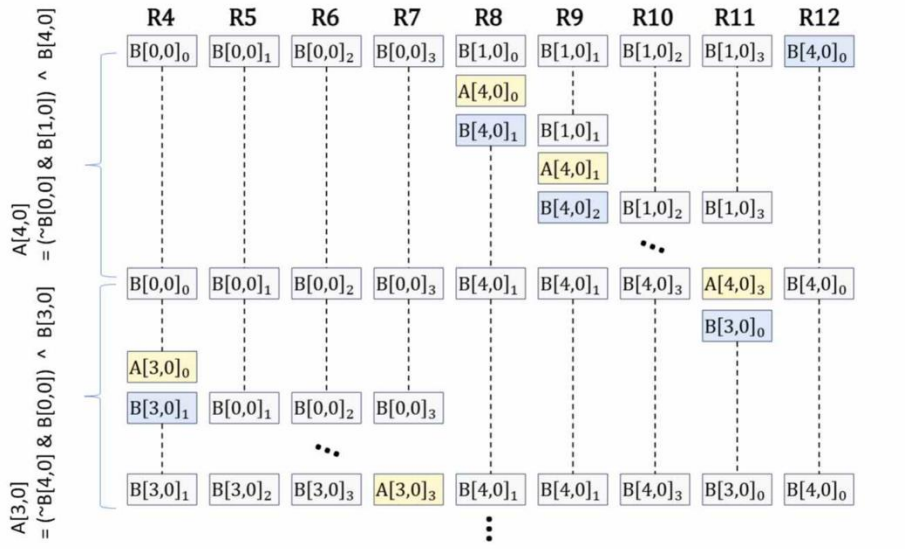
- **MSP430 Uses a 2-Operand Instruction Format**
  - 3-operand instructions: Preserve both source registers
  - 2-operand instructions: Overwrite one source operand
    - Careful selection of source operands is required

2-operand ⇨ xor r4, r5 // r4 = r4 XOR r5

3-operand ⇨ eor r4, r5, r6 // r4 = r5 XOR r6

Work 1 : Keccak

- **Proposed Technique 2: Zig-zag Technique**
  - Optimize the operation order in θ and χ steps to preserve source operands as much as possible in the 2-operand architecture
    - θ : D[0] → D[3] → D[1] → D[4] → D[2]
    - χ : A[4,y] → A[3,y] → A[2,y] → A[1,y] → A[0,y]
  - Minimize memory access and recomputation overhead



Algorithm 7 Keccak-f1600

Input: 1600-bit A[x,y], all indexes are calculated mod 5
Output: 1600-bit A[x,y]

```
1: //r[x,y], RC[i] are constants fixed in the specification
2: for i in 0 to 23:
3:   //pre-θ step, for x in 0 to 4
4:    C[x] = A[x,0] ⊕ A[x,1] ⊕ A[x,2] ⊕ A[x,3] ⊕ A[x,4]
5:    D[x] = C[x-1] ⊕ rot(C[x+1],1)
6:   //θ step, for x,y in 0 to 4
7:    A[x,y] = A[x,y] ⊕ D[x]
8:   //ρ+π steps, for x,y in 0 to 4
9:   B[y,2*x+3*y] = rot(A[x,y], r[x,y])
10:  //χ step, for x,y in 0 to 4
11:  A[x,y] = ((~B[x+1,y]) & B[x+2,y]) ⊕ B[x,y]
12:  //ι step
13:   A[0,0] = A[0,0] ⊕ RC[i]
14: end for
```

Work 1 : Keccak

- **Kecak Results**
  - 57% faster than Reference C
  - Significant performance gap compared to 32-bit architectures
    - More instructions for 64-bit operations on a 16-bit architecture
    - Lack of a barrel shifter

Work	Platform (Board)	Keccak-f1600
Asm[HAZ+24]	Cortex-M4 (STM32F4)	9 218
Asm[HAZ+24]	Cortex-M3 (ATSAM3X8E)	9 789
C[ZYHK25]	RISC-V (K230)	15 579
Asm[ZYHK25]		7 808
C (Ref)	MSP430 (MSP430F67791)	86 599
Asm (This work)		55 316

KOOKMIN UNIVERSITY

Work 2 :
NTT-based polynomial multiplication

CSE CRYPTO & SECURITY ENGINEERING Lab

- **State-of-the-Art Modular Multiplication and Reduction Methods**
  - Review of state-of-the-art methods
  - Adopt Montgomery Multiplication and Barrett Reduction for both Kyber and Dilithium, considering constant loading overhead (e.g., twiddle factors) and register pressure

16-bit Modular Multiplication	Work	Target platform	Cycles on the MSP430
LookUp-Table	[KS25]	8-bit AVR	37
Barrett (Approximate)	[HKS25]	32-bit Cortex-M3	35
Barrett	[BHK+21]	64-bit Cortex-A	35
Plantard	[HZZ+22]	32-bit Cortex-M4	28
	[HZZ+24]	32-bit RISC-V	
Montgomery	[Sei18]	AVX2/AVX512	30

Work 2 : NTT

Algorithm 11 16-bit CT Algorithm on MSP430

Input: two 16-bit integers a, b, twiddle factor ζ
Output: $a = a + b\zeta, b = a - b\zeta$

```
1: mov ζ, &MPYS; mov b, &OP2;
2: mov &RESHI, tmp;
3: mov &RESLO, &MPYS; mov #q^-1, &OP2;
4: mov &RESLO, &MPYS; mov #q, &OP2;
5: sub &RESHI, tmp; mov a, b;
6: add tmp, a; sub tmp, b;
```

Montgomery reduction

Algorithm 12 32-bit CT Algorithm on MSP430

Input: two 32-bit integers $a(= a_1||a_0), b(= b_1||b_0)$, twiddle factor $\zeta = (\zeta_1||\zeta_0)$
Output: $a = a + b\zeta, b = a - b\zeta$

```
1: mov ζ0, &MPYS32L; mov ζ1, &MPY32SH;
2: mov b0, &OP2L; mov b1, &OP2H;
3: mov &RES2, tmp0; mov &RES3, tmp1;
4: mov &RES0, &MPYS32L; mov &RES1, &MPY32H;
5: mov #q0^-1, &OP2L; mov #q1^-1, &OP2H;
6: mov &RES0, &MPYS32L; mov &RES1, &MPYS32H;
7: mov #q0, &OP2L; mov #q1, &OP2H;
8: sub &RES2, tmp0; subc &RES3, tmp1;
9: mov a0, b0; mov a1, b1;
10: add tmp0, a0; addc tmp1, a1;
11: sub tmp0, b0; subc tmp1, b1;
```

Montgomery reduction

- **Montgomery multiplication on MSP430**
  - Perform shifts by multiples of 16 for free
  - Directly load operands into the hardware multiplier when their values do not need to be preserved
  - Directly use the multiplier output as the input for Montgomery Reduction
    - Multiplier latency when using the output as the next input
    - Avoid stalls by loading the next butterfly inputs or performing addition/subtraction for the previous butterfly

- **Fermat prime multiplication on MSP430**
  - Dilithium: Small coefficients in cs_1 and cs_2 allow the use of a smaller prime based on the maximum coefficient bound
    - Dilithium2 and Dilithium5: Use $q' = 257$
      - Perform modular reduction without costly hardware multiplication using a prime-specific reduction method

Algorithm 10 Optimized modular multiplication based on a Fermat prime ($q = 257$)

Input: two 16-bit integers a, b, with $a \cdot b \in (-q \cdot 2^{15}, q \cdot 2^{15})$
Output: 16-bit c_0 , with $c_1 \in (-q, q)$

```
1: //c(= c1||c0) = a · b
2: mov a, &MPYS; mov b, &OP2;
3: mov &RESLO, c0; mov &RESHI, c1;
4: mov.b c0, tmp; swpb c0;          ▷ Fermat prime number reduction begins here
5: and.b c0, c0; sub c0, tmp;
6: mov.b c1, c0; add tmp, c0;
7: swpb c1; sxt c1;
8: sub c1, c0;
```

    - Dilithium3: Use $q' = 769$
      - Apply the same method as Kyber

Work 2 : NTT

- **NTT / Matrix-Vector Multiplication Optimization**
  - MSP430 typically has limited RAM (8–32 KB)
  - Avoid methods that require extra memory for better performance
    - Our proposed methods are independent of and compatible with these techniques
  - Layer Merging:
    - Kyber: Merge up to 3 layers
    - Dilithium: Merge up to 2 layers

Techniques	Speed	Stack	MSP430
Layer merging [GOPS13]	↑	-	✓
Lazy reduction for pw-mul[ABCG20]	↑	-	✓
Streaming Public matrix **A** [BKS19, HKS25]	↓	⇓	✓
Asymetric multiplication[BHK+21]	↑	↑	✗
Better accumulation[AHKS22]	↑	↑	✗

Work 2 : NTT

- **NTT Results**
  - 16-bit NTT: 134–210% performance improvement over Reference C
  - 32-bit NTT: 56–96% improvement over Reference C and up to 10% over [SKS25]
  - Fermat Prime (257): Up to 94% improvement over our original 16-bit NTT by reducing hardware multiplier usage in modular reduction

Work	Platform (Board)	16-bit (NTT/NTT^{-1}/PWM)	16-bit($q = 257$) NTT/NTT^{-1}/PWM	32-bit NTT/NTT^{-1}/PWM
Asm[AHKS22]	Cortex-M4 (STM32F4)	5 992/6 282/1 613	5 524/5 563/1 225	8 093/8 415/1 955
Asm[HZZ+24]	Cortex-M3 (ATSAM3X8E)	8 026/8 779/4 311	–	–
Asm[HKS25]	Cortex-M3 (NUCLEO-F207ZG)	–	–	21 876/26 524/–
Asm[HZZ+24]	RISC-V	13 218/11 427/2 891	–	–
Asm[ZYHK25]	(K230)	5 714/6 005/1 673	–	7 054/7 561/2 026
Asm[LWW23]	Cortex-M0 (–)	20 972/20 303/12 101	–	–
Asm[KS25]	AVR (ATmega1280p)	58 364/67 792/–	–	–
Asm[SKS25]	MSP430 (MSP430F6779†)	–	30 399/57 804/18 869	90 736/114 026/21 076
Asm[LAKS17]	MSP430 (MSP430F67791)	178 100/–/–	–	–
C (Ref)		102 505/175 840/51 939	–	158 038/206 692/32 546
Asm (This work)		43 732/50 378/16 765	22 545/45 982/13 429	82 635/105 409/20 826

Results

Results

- **Kyber: 46.1–51.3% improvement in key generation, 45.6–60.0% in encapsulation, and 46.2–62.3% in decapsulation over Reference C**

Table 9: Cycle counts (cc), and stack usage (B) for KeyGen, Encaps, and Decaps of all security levels of Kyber. 1,000 cc is denoted by k. *: **A** generated on-the-fly. § : Asymmetric multiplication and Better accumulation applied.

Work	Platform Board	variant		Kyber512		Kyber768		Kyber1024	
				cc	stack	cc	stack	cc	stack
Asm [AHKS22]	32-bit Cortex-M4 (STM32F4)	speed*§	K	443 k	4 272	718 k	5 312	1 138 k	6 336
			E	536 k	5 376	870 k	6 416	1 324 k	7 432
			D	487 k	5 384	796 k	6 432	1 227 k	7 448
		stack*	K	444 k	2 220	724 k	2 736	1 149 k	3 256
			E	540 k	2 308	879 k	2 808	1 341 k	3 328
			D	492 k	2 324	807 k	2 824	1 246 k	3 352
Asm [HZZ+24]	32-bit Cortex-M3 (ATSAM3X8E)	speed*§	K	518 k	3 268	842 k	4 044	1 333 k	4 812
			E	626 k	3 860	1 017 k	4 636	1 548 k	5 404
			D	587 k	3 860	958 k	4 636	1 471 k	5 404
		stack*	K	519 k	2 212	844 k	3 084	1 342 k	3 596
			E	628 k	2 300	1 025 k	2 772	1 563 k	3 284
			D	590 k	2 308	967 k	2 788	1 486 k	3 300
Asm [HZZ+24]	32-bit RISC-V (K230)	speed*§	K	–	–	1 052 k	–	–	–
			E	–	–	1 261 k	–	–	–
			D	–	–	1 179 k	–	–	–
C(Ref) [ZYHK25]		–	K	–	–	1 222 k	–	–	–
			E	–	–	1 602 k	–	–	–
			D	–	–	1 691 k	–	–	–
Asm [ZYHK25]		speed§	K	–	–	496 k	–	–	–
			E	–	–	606 k	–	–	–
			D	–	–	578 k	–	–	–
Asm [LWW23]	32-bit Cortex-M0 (–)	stack*	K	942 k	3 136	1 492 k	3 648	2 292 k	4 160
			E	1 489 k	2 720	2 259 k	3 232	3 246 k	3 744
			D	1 594 k	2 720	2 359 k	3 232	3 347 k	3 744
Asm [KS25]	8-bit AVR (ATmega1280p)	stack*	K	5 290 k	2 220	8 608 k	2 736	13 640 k	3 256
			E	7 385 k	2 308	11 169 k	2 808	16 946 k	3 328
			D	6 763 k	2 324	10 250 k	2 824	15 773 k	3 352
C(Ref) [KSSW22]	16-bit MSP430 (MSP430F67791)	–	K	3 244 k	6 146	5 480 k	10 176	8 620 k	15 296
			E	4 288 k	8 804	7 204 k	13 344	10 760 k	18 976
			D	4 170 k	9 576	7 070 k	14 436	10 559 k	20 548
Asm (This work)		stack*	K	2 221 k	2 740	3 622 k	3 190	5 710 k	3 704
			E	2 946 k	2 816	4 503 k	3 266	6 820 k	3 780
			D	2 852 k	2 824	4 357 k	3 274	6 640 k	3 794

Results

- **Dilithium: 38.4–42.3% improvement in key generation, 30.1–40.0% in signing, and 36.5–41.1% in verification over [SKS25]**

Table 11: Cycle counts (cc), and stack usage (B) for KeyGen, Sign, and Verify of all security levels of Dilithium. 1,000 cc is denoted by k. For Dilithium, because the signing procedure includes aborts, benchmarks are typically reported as averages over 10,000 runs. Since execution on the MSP430 is time-consuming, our MSP430 implementation benchmarks use seeds that yield the average number of aborts per security level–4, 5, and 4, respectively. *: **A**, **y** generated on-the-fly. † : simulator.

Work	Platform Board	variant		Dilithium2		Dilithium3		Dilithium5	
				cc	stack	cc	stack	cc	stack
Asm [AHKS22]	32-bit Cortex-M4 (STM32F4)	stack*	K	1 596 k	8 508	2 827 k	9 540	4 829 k	11 696
			S	4 093 k	49 k	6 623 k	69 k	8 803 k	116 k
			V	1 572 k	36 k	2 692 k	58 k	4 707 k	93 k
C(Ref) [ZYHK25]	32-bit RISC-V (K230)	–	K	–	–	4 123 k	–	–	–
			S	–	–	13 671 k	–	–	–
			V	–	–	4 145 k	–	–	–
Asm [ZYHK25]		speed	K	–	–	1 934 k	–	–	–
			S	–	–	5 069 k	–	–	–
			V	–	–	1 889 k	–	–	–
Asm [HKS25]	8-bit AVR (ATmega1280p)	stack*	K	44 181 k	9 282	78 786 k	11 330	135 525 k	13 378
			S	58 600 k	12 059	181 787 k	14 107	457 611 k	16 155
			V	44 081 k	12 751	76 267 k	13 775	134 076 k	16 079
Asm [SKS25]	16-bit MSP430 (MSP430F6779†)	stack*	K	12 776 k	13 236	23 593 k	17 397	38 985 k	21 005
			S	83 892 k	18 172	168 330 k	22 564	234 416 k	26 834
			V	12 749 k	16 622	22 870 k	19 158	38 580 k	23 458
C(Ref) [HKS25]	16-bit MSP430 (MSP430F67791)	–*	K	13 505 k	13 236	24 638 k	17 397	40 631 k	21 005
			S	105 419 k	17 148	203 764 k	21 540	271 094 k	25 810
			V	13 869 k	16 622	24 480 k	19 158	41 012 k	23 458
Asm (This work)		stack*	K	9 142 k	13 236	17 053 k	17 397	27 399 k	21 005
			S	63 892 k	18 172	129 404 k	22 564	167 436 k	26 834
			V	9 251 k	16 622	16 758 k	19 158	27 345 k	23 458

Thank you!

dkldkl13@kookmin.ac.kr



- https://github.com/KMU-CSE/PJT_MSP430_KECCAK_KYBER_DILITHIUM
- https://sites.google.com/kookmin.ac.kr/cselab