SPEEDY implementation on cortex-m3,4

https://youtu.be/gOE7EHZy47E

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

SPEEDY

- Ultra low-latency block cipher

- 고성능 CPU의 내장 **하드웨어** 보안 솔루션을 위한 암호
- 면적, 에너지 제한은 부차적으로 고려
- 6bit-sbox 사용,192bit 평문 암호화

	Area [GE]					
	Commercial Foundry				NanGate OCL	
Cipher	90 nm LP	65 nm LP	40 nm LP	28 nm HPC	45 nm	15 nm
Gimli E-M	72644.00	82781.00	63100.50	144036.33	52038.67	57551.25
$\mathtt{MANTIS}_6$	21045.75	23264.50	20448.25	36073.33	12660.67	15954.00
$\mathtt{MANTIS}_7$	23229.25	26385.75	23192.50	43220.33	14225.67	17522.50
$\mathtt{MANTIS}_8$	26365.75	30316.75	25429.75	50793.00	15663.33	19707.50
Midori	18678.50	21964.00	17562.25	41450.67	10675.33	13927.25
Orthros	49639.75	61657.00	44715.75	74384.67	31317.33	39165.00
PRINCE	16244.25	19877.75	17177.00	38145.33	9873.33	13291.00
PRINCEv2	17661.25	18798.25	16556.50	33470.33	10332.00	13069.50
$\mathtt{QARMA}_5$-64-σ_0	19590.75	21706.75	20255.00	31703.00	11824.67	14880.75
$\mathtt{QARMA}_6$-64-σ_0	22624.25	25349.50	22689.00	38813.67	14165.67	17621.75
$\mathtt{QARMA}_7$-64-σ_0	25614.00	29323.00	24656.25	40494.33	15769.33	19770.25
$\mathtt{QARMA}_8$-64-σ_0	28813.75	32780.75	28262.75	47952.33	17908.00	22074.00
$\mathtt{QARMA}_5$-64-σ_1	20264.75	23753.00	20202.25	34302.00	12350.33	15588.75
$\mathtt{QARMA}_6$-64-σ_1	23162.25	26941.25	23333.75	45419.00	15066.00	18164.00
$\mathtt{QARMA}_7$-64-σ_1	26563.75	31495.00	27059.50	52108.00	16641.00	20670.25
$\mathtt{QARMA}_8$-64-σ_1	30534.50	35787.75	29116.50	54967.00	18963.67	22761.75
SPEEDY-5-192	47364.00	53856.00	47528.50	74467.00	27903.33	34649.00
SPEEDY-6-192	57322.00	64438.25	56816.00	88932.00	34085.00	41443.25
SPEEDY-7-192	68370.00	75273.00	65422.00	95235.67	39853.33	48727.75
SPEEDY-5-192 *	49902.00	58796.25	55846.75	80313.33	29839.00	38075.25
SPEEDY-6-192 *	59688.00	70653.00	66553.00	98950.00	36523.33	46266.50
SPEEDY-7-192 *	73397.75	84745.00	77519.75	111754.33	42813.33	54193.25

* = Optimized HDL code with direct instantiation of library cells based on Figures 2 and 3.

		Minimum Latency [ns]					
		Commercial Foundry				NanGate OCL	
#ib	S-box	90 nm LP	65 nm LP	40 nm LP	28 nm HPC	45 nm	15 nm
4	Midori $\mathtt{Sb}_0$	0.089098	0.070579	0.055577	0.021051	0.111156	0.010619
4	Midori $\mathtt{Sb}_1$	0.132489	0.095724	0.080657	0.026898	0.119637	0.009058
4	Orthros	0.075344	0.051435	0.055908	0.021003	0.133932	0.008821
4	PRINCE	0.087938	0.066545	0.052826	0.031010	0.126588	0.010176
4	QARMA σ_0	0.090568	0.057602	0.051993	0.022180	0.128350	0.009409
4	QARMA σ_1	0.144465	0.101487	0.077186	0.031306	0.156462	0.011272
4	QARMA σ_2	0.100530	0.075846	0.081528	0.036485	0.154379	0.013354
5	ASCON	0.197794	0.151025	0.123356	0.057595	0.210599	0.019854
6	DES $\mathtt{S}_1$	0.260286	0.190725	0.153514	0.069299	0.309009	0.030846
6	OAIU8L24	0.138926	0.111734	0.088775	0.046295	0.215628	0.017971
6	Q2263	0.233256	0.171537	0.157194	0.068870	0.246198	0.028648
6	min(RU8L24)	0.220168	0.144777	0.126819	0.060535	0.240982	0.026696
6	SPEEDY	0.106872	0.081330	0.065966	0.029890	0.161653	0.016124
6	SPEEDY *	0.096468	0.073253	0.064215	0.029470	0.138825	0.012799
6	SPEEDY_INV	0.207746	0.152161	0.129433	0.071523	0.278395	0.025665
8	AES	0.407332	0.304098	0.248914	0.130490	0.491570	0.048258

* = Optimized HDL code with direct instantiation of library cells based on Figure 2.

	Minimum Latency [ns]					
	Commercial Foundry				NanGate OCL	
Cipher	90 nm LP	65 nm LP	40 nm LP	28 nm HPC	45 nm	15 nm
Gimli E-M	4.532467	3.330192	2.794736	1.178424	4.537304	0.435069
$\mathtt{MANTIS}_6$	4.625529	3.405490	2.891383	1.278725	4.479773	0.437595
$\mathtt{MANTIS}_7$	5.201681	3.722473	3.234409	1.421365	5.074452	0.492703
$\mathtt{MANTIS}_8$	5.823127	4.233543	3.631438	1.594997	5.739020	0.552384
Midori	5.061255	3.582221	3.142355	1.362237	4.934847	0.481522
Orthros	3.862139	2.678637	2.401275	1.087139	3.774836	0.369497
PRINCE	4.101177	2.866749	2.521302	1.108886	4.059997	0.389144
PRINCEv2	4.047311	2.944367	2.509131	1.103273	4.077636	0.387146
$\mathtt{QARMA}_5$-64-σ_0	4.075846	2.920377	2.498908	1.134901	4.014516	0.385281
$\mathtt{QARMA}_6$-64-σ_0	4.770325	3.418600	2.951308	1.308331	4.554445	0.448931
$\mathtt{QARMA}_7$-64-σ_0	5.449707	3.909138	3.389576	1.538606	5.336362	0.517093
$\mathtt{QARMA}_8$-64-σ_0	6.103768	4.396543	3.814078	1.697027	5.966323	0.575525
$\mathtt{QARMA}_5$-64-σ_1	4.515514	3.284252	2.815788	1.219624	4.367899	0.408580
$\mathtt{QARMA}_6$-64-σ_1	5.297867	3.808675	3.271455	1.388353	4.944635	0.472798
$\mathtt{QARMA}_7$-64-σ_1	6.014477	4.371963	3.745959	1.601572	5.800633	0.542712
$\mathtt{QARMA}_8$-64-σ_1	6.720944	4.904521	4.202632	1.797539	6.498429	0.608985
SPEEDY-5-192	2.994643	2.178075	1.867064	0.847761	3.187368	0.300466
SPEEDY-6-192	3.637978	2.639186	2.277422	1.032206	3.848132	0.366762
SPEEDY-7-192	4.261928	3.087257	2.663004	1.217946	4.515505	0.431032
SPEEDY-5-192 *	2.941130	2.121748	1.820950	0.826217	2.817971	0.290961
SPEEDY-6-192 *	3.559981	2.573561	2.223863	1.011173	3.382270	0.353391
SPEEDY-7-192 *	4.174183	3.029217	2.620612	1.186598	3.995325	0.413950

* = Optimized HDL code with direct instantiation of library cells based on Figures 2 and 3.

Bitslicing

- Cortex-m 에서 효율적인 구현이 가능할 것이라고 판단
- Bitslicing 구현과 cortex-m의 Barrel shifter 사용으로 최적화

- 비트슬라이스 표현 192bit -> 6*32 (6bit-sbox 고려)

	0	1	2	…	29	30	31
Register1	1st bit	1st bit	1st bit	…	1st bit	1st bit	1st bit
Register2	2st bit	2st bit	2st bit	…	2st bit	2st bit	2st bit
Register3	3st bit	3st bit	3st bit	…	3st bit	3st bit	3st bit
Register4	4st bit	4st bit	4st bit	…	4st bit	4st bit	4st bit
Register5	5st bit	5st bit	5st bit	…	5st bit	5st bit	5st bit
Register6	6st bit	6st bit	6st bit	…	6st bit	6st bit	6st bit

0	1	2	3	4	5	6	7	…
32	33	34	35	36	37	38	39	…
64	65	66	67	68	69	70	71	…
96	97	98	…	…	…	…	…	…
128	129	130	…	…	…	…	…	…
160	161	160	…	…	…	…	…	…

0	6	12	18	24	30	36	42	…
1	7	13	19	25	31	37	43	…
2	8	14	20	26	32	38	44	…
3	9	15	21	27	33	39	45	…
4	10	16	22	28	34	40	46	…
5	11	17	23	29	35	41	47	…

Packing

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

→

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

적은 사이클로 연산 가능

```
.macro swpmv out0, out1, in0, in1, m, n, tmp
    eor     \tmp, \in1, \in0, lsr \n
    and     \tmp, \m
    eor     \out1, \in1, \tmp
    eor     \out0, \in0, \tmp, lsl \n
.endm
```

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Swpmv 2번

→

0	4	2	6
1	5	3	7
8	12	10	14
9	13	11	15

Swpmv 2번

→

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

블록 정렬

- SPIDDY는 192비트 평문이 6bit비트 블록 32개가 연산....

32BIT

0	1	2	3	4	5
5	6	7	8	9	10
10	11	12	13	14	15
16	17	18	19	20	21
21	22	23	24	25	26
26	27	28	29	30	31

6 register

192비트

- 정렬이 어려움..

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

블록 정렬

- 단순하게 1비트 씩 이동 → 12 * 32 = 384 cycle + 6번 ldr
- 96 + 41 = 137 cycle + 6번 ldr 로 구현

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**⋯**
32	33	34	35	36	37	38	39	⋯
64	65	66	67	68	69	70	71	⋯
96	97	98	⋯	⋯	⋯	⋯	⋯	⋯
128	129	130	⋯	⋯	⋯	⋯	⋯	⋯
160	161	160	⋯	⋯	⋯	⋯	⋯	⋯

0	**1**	**2**	**3**	**4**	**5**	**36**	**37**	**⋯**
6	7	8	9	10	11	42	43	⋯
12	13	14	15	16	17	38	44	⋯
18	19	20	21	22	23	39	45	⋯
24	25	26	27	28	29	40	46	⋯
30	31	32	33	34	35	41	47	⋯

0	**6**	**12**	**18**	**24**	**30**	**36**	**42**	**⋯**
1	7	13	19	25	31	37	43	⋯
2	8	14	20	26	32	38	44	⋯
3	9	15	21	27	33	39	45	⋯
4	10	16	22	28	34	40	46	⋯
5	11	17	23	29	35	41	47	⋯

0	**6**	**12**	**18**	**24**	**30**	**31**
1	7	13	19	25	30	31
2	8	14	20	26	30	31
3	9	15	21	27	30	31
4	10	16	22	28	30	31
5	11	17	23	29	30	31

6비트 블록표현

S-box

- C코드

```c
void SB(StateChar input) {
    for(int i = 0; i < 32; i++)
        input[i] = S[(int)input[i]];
}
```

```c
const uint8_t S[64] = { 0x08,0x00,0x09,0x03,0x38,0x10,0x29,0x13,0x0c,0x0d,0x04,0x07,0x30,0x01,0x20,0x23,
                        0x1a,0x12,0x18,0x32,0x3e,0x16,0x2c,0x36,0x1c,0x1d,0x14,0x37,0x34,0x05,0x24,0x27,
                        0x02,0x06,0x0b,0x0f,0x33,0x17,0x21,0x15,0x0a,0x1b,0x0e,0x1f,0x31,0x11,0x25,0x35,
                        0x22,0x26,0x2a,0x2e,0x3a,0x1e,0x28,0x3c,0x2b,0x3b,0x2f,0x3f,0x39,0x19,0x2d,0x3d };
```

- 테이블 연산 대신 32블록 병렬 연산

$$
\begin{aligned}
y_0 &= (x_3 \wedge \neg x_5) \vee (x_3 \wedge x_4 \wedge x_2) \vee (\neg x_3 \wedge x_1 \wedge x_0) \vee (x_5 \wedge x_4 \wedge x_1),\\
y_1 &= (x_5 \wedge x_3 \wedge \neg x_2) \vee (\neg x_5 \wedge x_3 \wedge \neg x_4) \vee (x_5 \wedge x_2 \wedge x_0) \vee (\neg x_3 \wedge \neg x_0 \wedge x_1),\\
y_2 &= (\neg x_3 \wedge x_0 \wedge x_4) \vee (x_3 \wedge x_0 \wedge x_1) \vee (\neg x_3 \wedge \neg x_4 \wedge x_2) \vee (\neg x_0 \wedge \neg x_2 \wedge \neg x_5),\\
y_3 &= (\neg x_0 \wedge x_2 \wedge \neg x_3) \vee (x_0 \wedge x_2 \wedge x_4) \vee (x_0 \wedge \neg x_2 \wedge x_5) \vee (\neg x_0 \wedge x_3 \wedge x_1),\\
y_4 &= (x_0 \wedge \neg x_3) \vee (x_0 \wedge \neg x_4 \wedge \neg x_2) \vee (\neg x_0 \wedge x_4 \wedge x_5) \vee (\neg x_4 \wedge \neg x_2 \wedge x_1),\\
y_5 &= (x_2 \wedge x_5) \vee (\neg x_2 \wedge \neg x_1 \wedge x_4) \vee (x_2 \wedge x_1 \wedge x_0) \vee (\neg x_1 \wedge x_0 \wedge x_3).
\end{aligned}
$$

0	6	12	18	24	30	36	42	…
1	7	13	19	25	31	37	43	…
2	8	14	20	26	32	38	44	…
3	9	15	21	27	33	39	45	…
4	10	16	22	28	34	40	46	…
5	11	17	23	29	35	41	47	…

- 분배 법칙

$(a \wedge b) \vee (a \wedge c) = a \vee (b \wedge c)$: 1개의 $\wedge$연산 제거

$$y_0 = (\boxed{x_3} \wedge \neg x_5) \vee (\boxed{x_3} \wedge x_4 \wedge x_2) \vee (\neg x_3 \wedge \boxed{x_1} \wedge x_0) \vee (x_5 \wedge x_4 \wedge \boxed{x_1}),$$

ShiftColumns

- 수식 $$y_{[i,j]} = x_{[i+j,j]}\,, \quad \forall\, i, j\,.$$

- C 코드

```c
void SC(StateChar input) {
    bool temp[32][6];
    for(int i = 0; i < 32; i++)
        for(int j = 0; j < 6; j++)
            temp[i][j] = ((input[(i + j) % 32] >> (5 - j)) & 1);

    for(int i = 0; i < 32; i++) {
        input[i] = 0;
        for(int j = 0; j < 6; j++) {
            input[i] <<= 1;
            input[i] ^= temp[i][j];
        }
    }
}
```

- Barrel shifter 사용하면 비용x

	0	1	2	3	4	5	…
1	1	1	1	1	1	1	…
2	2	2	2	2	2	2	…
3	3	3	3	3	3	3	…
4	4	4	4	4	4	4	…
5	5	5	5	5	5	5	…
6	6	6	6	6	6	6	…

<<<1

<<<2

<<<3

<<<4

<<<5

	0	1	2	3	4	5	…
1	1	1					…
2	2	2					…
3	3	3					…
4	4	4					…
5	5	5					…
6	6	6					…

MixColumns

- 수식 $$y_{[i,j]} = x_{i,j} \oplus x_{[i+\alpha_1,j]} \oplus x_{[i+\alpha_2,j]} \oplus x_{[i+\alpha_3,j]} \oplus x_{[i+\alpha_4,j]} \oplus x_{[i+\alpha_5,j]} \oplus x_{[i+\alpha_6,j]} \,, \quad \forall i,j\,.$$

- C 코드

```c
void MC(StateChar input) {
    const int alphas[] = {1, 5, 9, 15, 21, 26};

    StateChar temp;
    for(int i = 0; i < 32; i++)
        temp[i] = input[i];

    for(int a = 0; a < (sizeof(alphas) / sizeof(alphas[0])); a++)
    {
        for(int i = 0; i < 32; i++)
            input[i] ^= temp[(i + alphas[a]) % 32];
    }
}
```

$0 \oplus 1 \oplus 5 \oplus 9 \oplus 15 \oplus 21 \oplus 26$
$1 \oplus 2 \oplus 6 \oplus 10 \oplus 16 \oplus 22 \oplus 27$
$2 \oplus 3 \oplus 7 \oplus 11 \oplus 17 \oplus 23 \oplus 28$
$\vdots$
$30 \oplus 31 \oplus 3 \oplus 7 \oplus 13 \oplus 19 \oplus 24$
$31 \oplus 0 \oplus 4 \oplus 8 \oplus 14 \oplus 20 \oplus 25$

MixColumns

- 수식 $$y_{[i,j]} = x_{i,j} \oplus x_{[i+\alpha_1,j]} \oplus x_{[i+\alpha_2,j]} \oplus x_{[i+\alpha_3,j]} \oplus x_{[i+\alpha_4,j]} \oplus x_{[i+\alpha_5,j]} \oplus x_{[i+\alpha_6,j]}\,, \quad \forall i,j\,.$$

- C 코드

```c
void MC(StateChar input) {
    const int alphas[] = {1, 5, 9, 15, 21, 26};

    StateChar temp;
    for(int i = 0; i < 32; i++)
        temp[i] = input[i];

    for(int a = 0; a < (sizeof(alphas) / sizeof(alphas[0])); a++)
    {
        for(int i = 0; i < 32; i++)
            input[i] ^= temp[(i + alphas[a]) % 32];
    }
}
```

$$
\begin{array}{l}
\boxed{\begin{array}{c} 0 \\ 1 \\ 2 \\ \\ 30 \\ 31 \end{array}}
\begin{array}{l}
\oplus 1 \oplus 5 \oplus 9 \oplus 15 \oplus 21 \oplus 26 \\
\oplus 2 \oplus 6 \oplus 10 \oplus 16 \oplus 22 \oplus 27 \\
\oplus 3 \oplus 7 \oplus 11 \oplus 17 \oplus 23 \oplus 28 \\
\qquad\qquad \vdots \\
\oplus 31 \oplus 3 \oplus 7 \oplus 13 \oplus 19 \oplus 24 \\
\oplus 0 \oplus 4 \oplus 8 \oplus 14 \oplus 20 \oplus 25
\end{array}
\end{array}
$$

	0	1	2	3	4	5	…
1	1	1	1	1	1	1	…
2	2	2	2	2	2	2	…
3	3	3	3	3	3	3	…
4	4	4	4	4	4	4	…
5	5	5	5	5	5	5	…
6	6	6	6	6	6	6	…

MixColumns

- 수식 $$y_{[i,j]} = x_{i,j} \oplus x_{[i+\alpha_1,j]} \oplus x_{[i+\alpha_2,j]} \oplus x_{[i+\alpha_3,j]} \oplus x_{[i+\alpha_4,j]} \oplus x_{[i+\alpha_5,j]} \oplus x_{[i+\alpha_6,j]}\,, \quad \forall i,j\,.$$

- C 코드

```c
void MC(StateChar input) {
    const int alphas[] = {1, 5, 9, 15, 21, 26};

    StateChar temp;
    for(int i = 0; i < 32; i++)
        temp[i] = input[i];

    for(int a = 0; a < (sizeof(alphas) / sizeof(alphas[0])); a++)
    {
        for(int i = 0; i < 32; i++)
            input[i] ^= temp[(i + alphas[a]) % 32];
    }
}
```

$0 \oplus \boxed{1} \oplus 5 \oplus 9 \oplus 15 \oplus 21 \oplus 26$

$1 \oplus \boxed{2} \oplus 6 \oplus 10 \oplus 16 \oplus 22 \oplus 27$

$2 \oplus \boxed{3} \oplus 7 \oplus 11 \oplus 17 \oplus 23 \oplus 28$

$\vdots$

$30 \oplus \boxed{31} \oplus 3 \oplus 7 \oplus 13 \oplus 19 \oplus 24$

$31 \oplus \boxed{0} \oplus 4 \oplus 8 \oplus 14 \oplus 20 \oplus 25$

- Barrel shifter <<<1

	0	1	2	3	4	5	…
1	1	1	1	1	1	1	…
2	2	2	2	2	2	2	…
3	3	3	3	3	3	3	…
4	4	4	4	4	4	4	…
5	5	5	5	5	5	5	…
6	6	6	6	6	6	6	…

<<<1

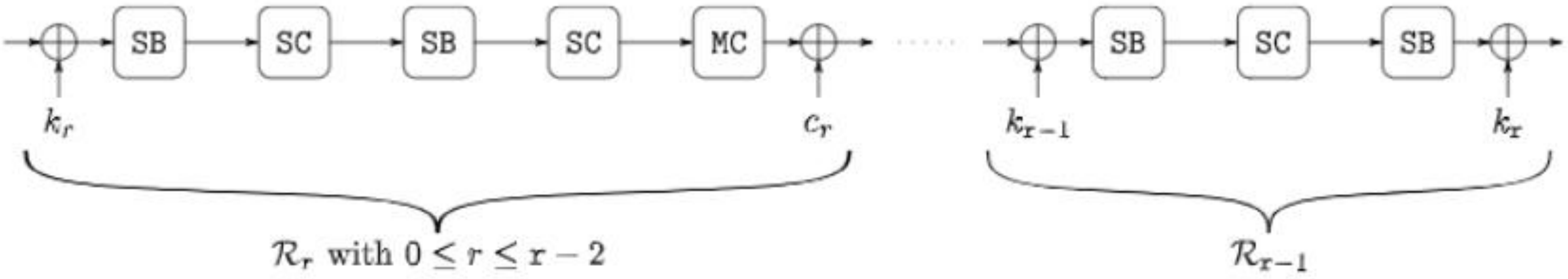
	1	2	3	4	5	…
1	1	1	1	1	1	…
2	2	2	2	2	2	…
3	3	3	3	3	3	…
4	4	4	4	4	4	…
5	5	5	5	5	5	…
6	6	6	6	6	6	…

AddRoundKey & AddRoundConstant

$$y_{[i,j]} = x_{[i,j]} \oplus k_{r\,[i,j]}\,, \quad \forall i,j\,.$$

$$y_{[i,j]} = x_{[i,j]} \oplus c_{r\,[i,j]}\,, \quad \forall i,j\,.$$

- 반올림 상수는 숫자 π−3 = 0.1415....의 이진수로 선택



Q & A