SIMON_SPECK

2020.05.17

https://youtu.be/33inn9wMAHA

한성대학교 HANSUNG UNIVERSITY

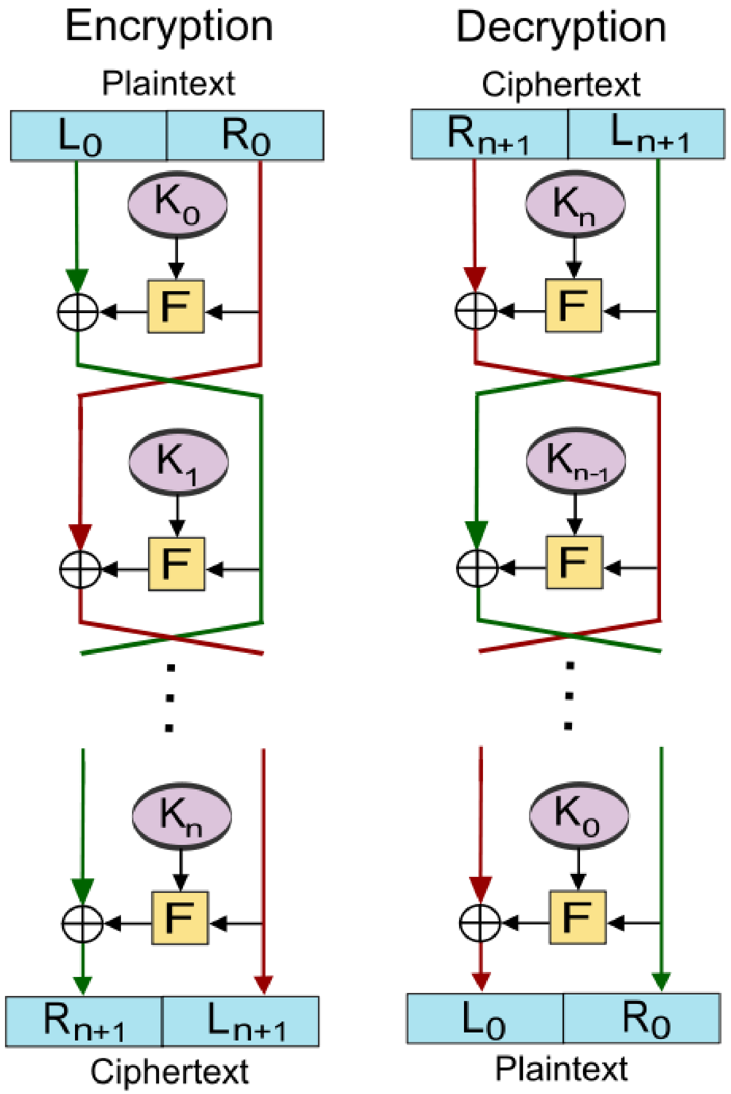
CryptoCraft LAB
https://crypto.modoo.at

SIMON SPECK

- NSA(National Security Agency)에서 제작한 경량 암호 – 2013
- SIMON: 하드웨어에 최적화
- SPECK: 소프트웨어에 최적화
- Feistel cipher

Feistel Cipher

- 블록 암호를 생성하는 대칭 구조
- 암호화, 복호화가 매우 유사
  - 키의 순서만 다름



SIMON

Block size (bits)	Key size (bits)	Rounds
32	64	32
48	72	36
	96	36
64	96	42
	128	44
96	96	52
	144	54
128	128	68
	192	69
	256	72

SIMON

- Feistel → n bit word
- block 길이: 2n
- Key multiplier: m (2,3,4)
- Simon64/128 → 64bit plaintext block(n=32) + 128 bit key

- Simon32/64 → 32bit plaintext block(n=16) + 64 bit key

SIMON

- #define ROTL32(x,r) (((x)<>(32-(r))))
- #define ROTR32(x,r) (((x)>>(r)) | ((x)<<(32-(r))))

- define f32(x) ((ROTL32(x,1) & ROTL32(x,8)) ^ ROTL32(x,2))
- define R32x2(x,y,k1,k2) (y^=f32(x), y^=k1, x^=f32(y), x^=k2)

SIMON

```
void Simon6496KeySchedule(u32 K[],u32 rk[])
{
  u32 i,c=0xfffffffc;
  u64 z=0x7369f885192c0ef5LL;

  rk[0]=K[0]; rk[1]=K[1]; rk[2]=K[2];

  for(i=3;i<42;i++){
    rk[i]=c^(z&1)^rk[i-3]^ROTR32(rk[i-1],3)^ROTR32(rk[i-1],4);
    z>>=1;
  }
}
```

- rk[0-41] 생성

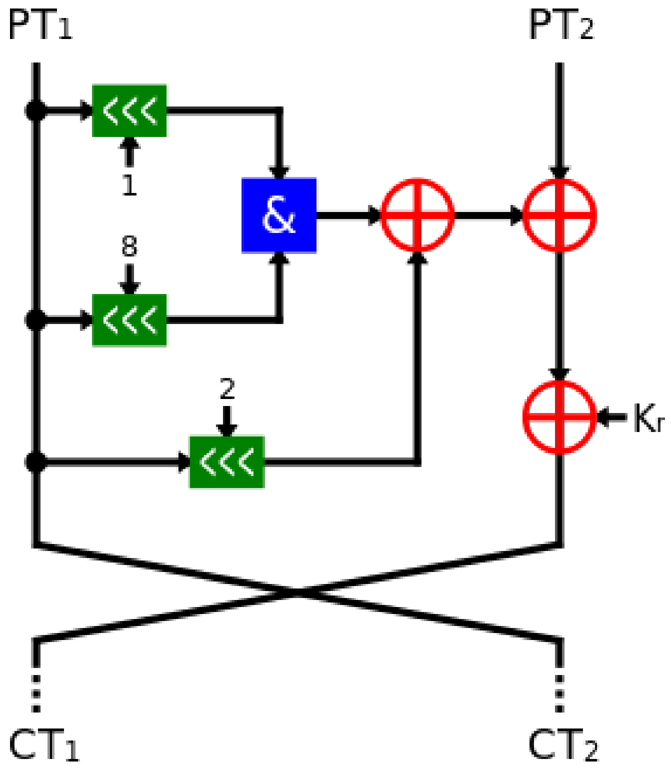
SIMON

```
void Simon6496Encrypt(u32 Pt[],u32 Ct[],u32 rk[])
{
  u32 i;

  Ct[1]=Pt[1]; Ct[0]=Pt[0];
  for(i=0;i<42;) R32x2(Ct[1],Ct[0],rk[i++],rk[i++]);
}
```

- define f32(x) ((ROTL32(x,1) & ROTL32(x,8)) ^ ROTL32(x,2))
- define R32x2(x,y,k1,k2) (y^=f32(x), y^=k1, x^=f32(y), x^=k2)

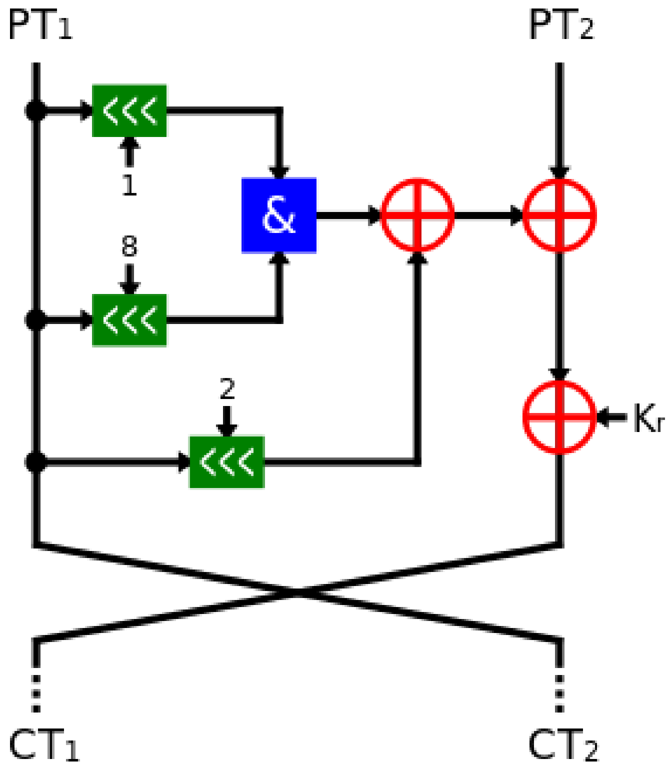
SIMON

- define f32(x) ((ROTL32(x,1) & ROTL32(x,8)) ^ ROTL32(x,2))
- define R32x2(x,y,k1,k2) (y^=f32(x), y^=k1, x^=f32(y), x^=k2)



SIMON

- define f32(x) ((ROTL32(x,1) & ROTL32(x,8)) ^ ROTL32(x,2))
- define R32x2(x,y,k1,k2) (y^=f32(x), y^=k1, x^=f32(y), x^=k2)



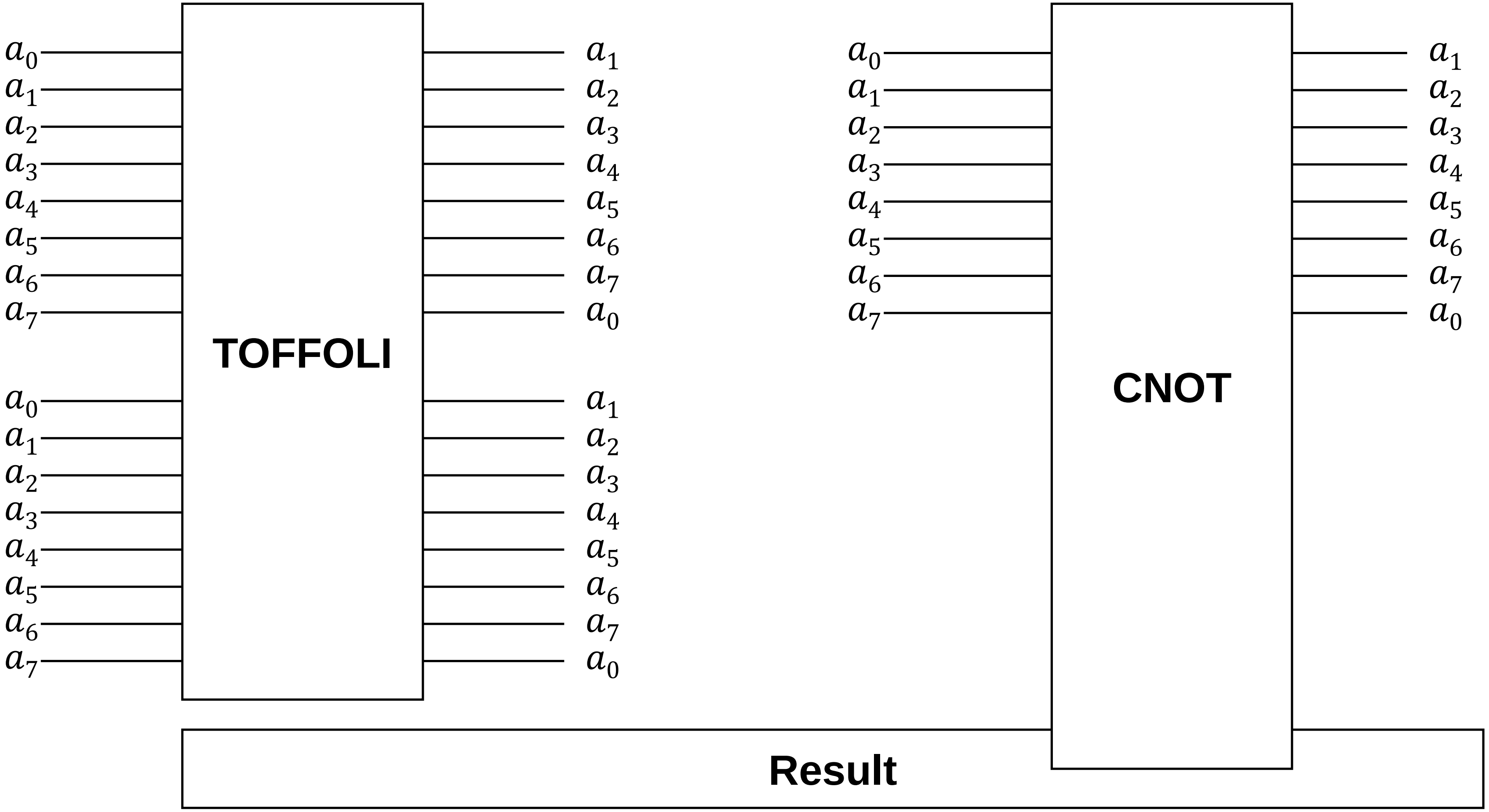
SIMON

- define f32(x) ((ROTL32(x,1) & ROTL32(x,8)) ^ ROTL32(x,2))

Left Shift by 1

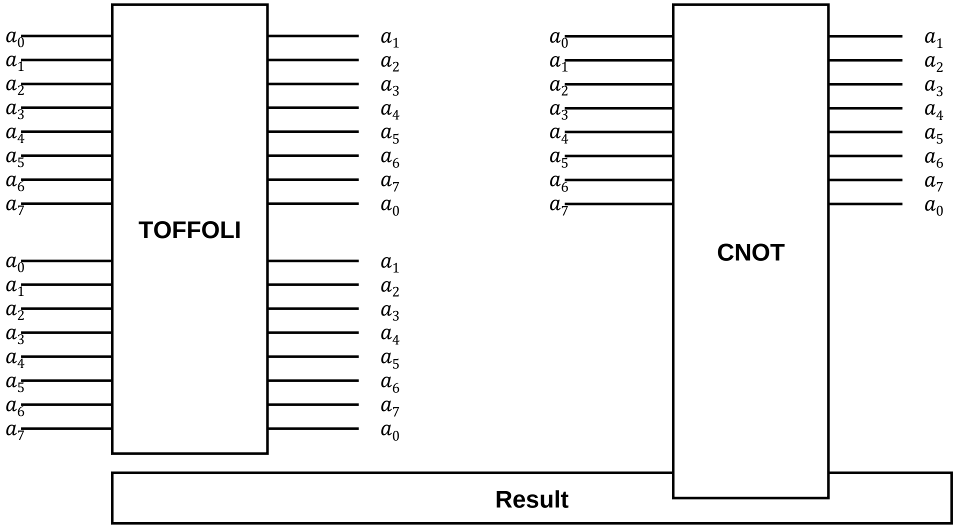
SIMON

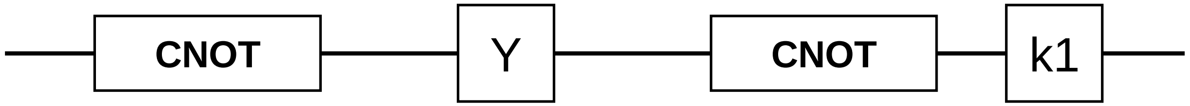
- define f32(x) ((ROTL32(x,1) & ROTL32(x,8)) ^ ROTL32(x,2))



SIMON

- define f32(x) ((ROTL32(x,1) & ROTL32(x,8)) ^ ROTL32(x,2))
- define R32x2(x,y,k1,k2) (y^=f32(x), y^=k1, x^=f32(y), x^=k2)





SPECK

Block size (bits)	Key size (bits)	Rounds
2×16 = 32	4×16 = 64	22
2×24 = 48	3×24 = 72	22
	4×24 = 96	23
2×32 = 64	3×32 = 96	26
	4×32 = 128	27
2×48 = 96	2×48 = 96	28
	3×48 = 144	29
2×64 = 128	2×64 = 128	32
	3×64 = 192	33
	4×64 = 256	34

SPECK

- Feistel → n bit word
- block 길이: 2n
- Key multiplier: m (2,3,4)
- SPECK64/128 → 64bit plaintext block(n=32) + 128 bit key
- SPECK32/64 → 32bit plaintext block(n=16) + 64 bit key

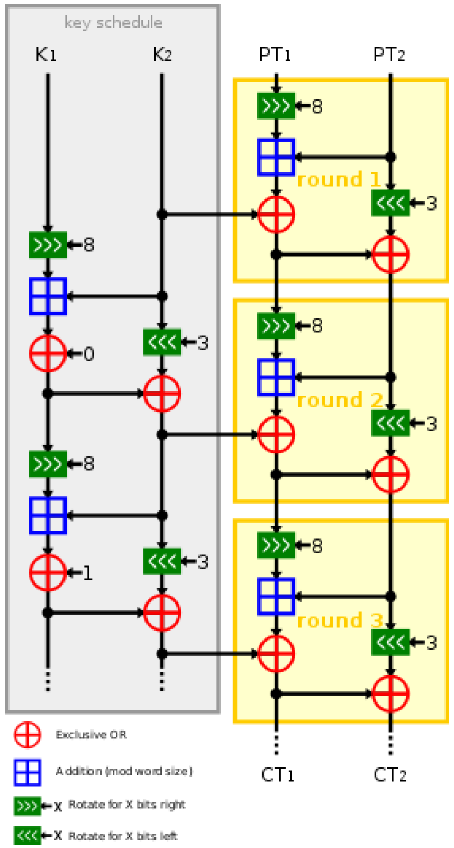
SPECK

- define ER32(x,y,k) (x=ROTR32(x,8), x+=y, x^=k, y=ROTL32(y,3), y^=x)
- define DR32(x,y,k) (y^=x, y=ROTR32(y,3), x^=k, x-=y, x=ROTL32(x,8))

SPECK

- define ER32(x,y,k) (x=ROTR32(x,8), x+=y, x^=k, y=ROTL32(y,3), y^=x)

```
void Speck6496KeySchedule(u32 K[],u32 rk[])
{
  u32 i,C=K[2],B=K[1],A=K[0];

  for(i=0;i<26;){
    rk[i]=A; ER32(B,A,i++);
    rk[i]=A; ER32(C,A,i++);
  }
}
```



- rk[0] = A

SPECK

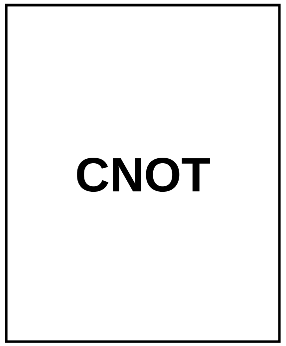
- define ER32(x,y,k) (x=ROTR32(x,8), x+=y, x^=k, y=ROTL32(y,3), y^=x)

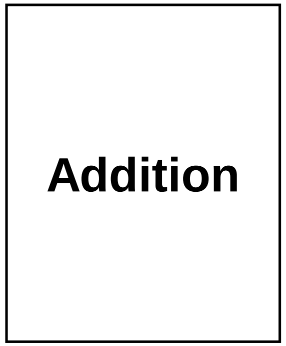
```
void Speck6496Encrypt(u32 Pt[],u32 Ct[],u32 rk[])
{
  u32 i;

  Ct[0]=Pt[0]; Ct[1]=Pt[1];
  for(i=0;i<26;) ER32(Ct[1],Ct[0],rk[i++]);
}
```

SPECK

- define ER32(x,y,k) (x=ROTR32(x,8), x+=y, x^=k, y=ROTL32(y,3), y^=x)

Shift

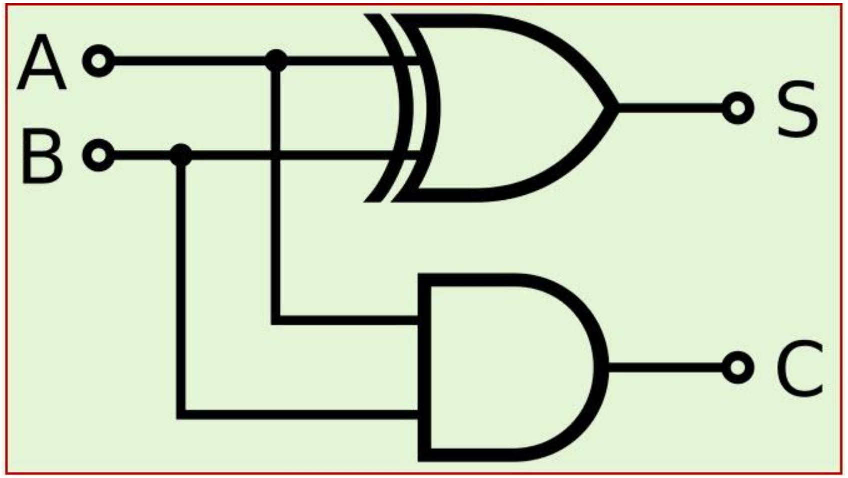




SPECK

- Half Adder

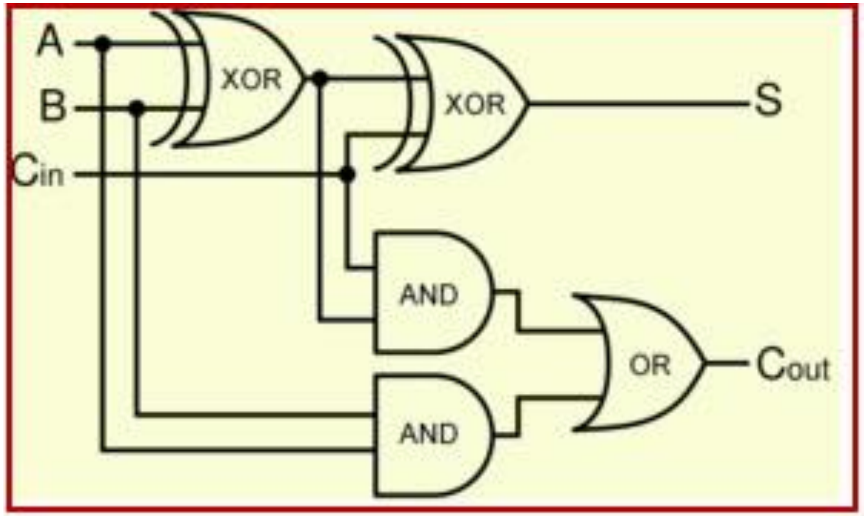
INPUTS		OUTPUTS	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



SPECK

- Full Adder

INPUTS			OUTPUT	
A	B	C-IN	C-OUT	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



SPECK

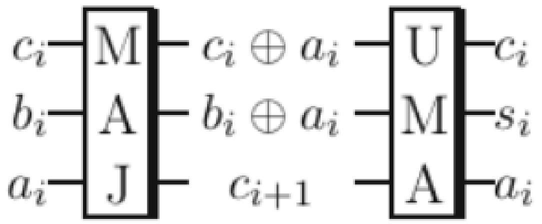
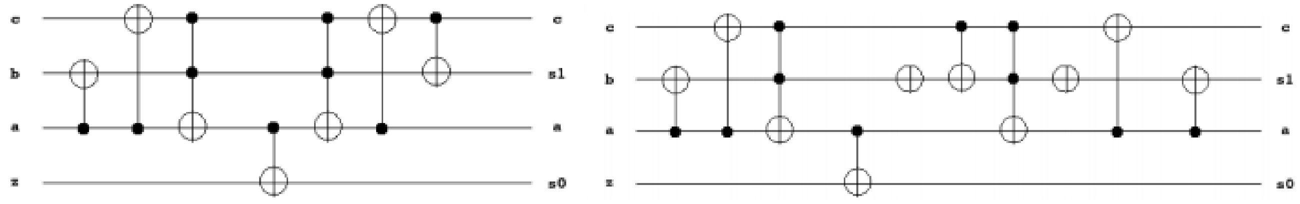


Figure 20. Basic addition circuit (Half adder). s_i is the modular sum of the inputs a_i and b_i, and $c_i$is the incoming carry bit. [Picture from [1]]



(a) (b)

Figure 21. Extension of Fig. 20 to full addition (a), and its High parallelism version (b).

Q & A