

HLS Final Project Presentation

Hardware Design of Ciphers via HLS : The Case of PRESENT

Team #3

Github : https://github.com/ANIIIIII/HLS_final_project.git

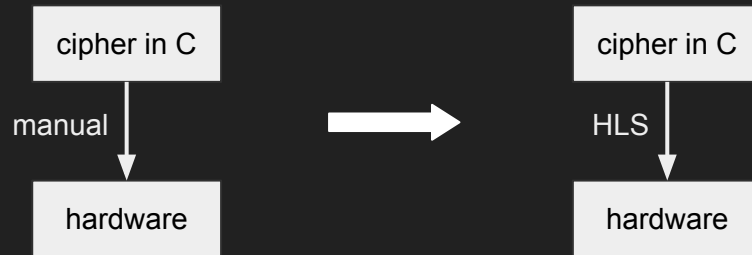
Outline

- Abstract
- PRESENT
- Cipher Design via HLS
 - sBox Layer
 - Permutation Layer
 - Optimizations
- Experiment Results
- Conclusion

Abstract

Our goal is to demonstrate the feasibility of using HLS in optimizing hardware design of ciphers by implementing several architectures in [2].

The point is how we replace conventional method by HLS design. So we will focus on the general method to design a cipher with HLS.

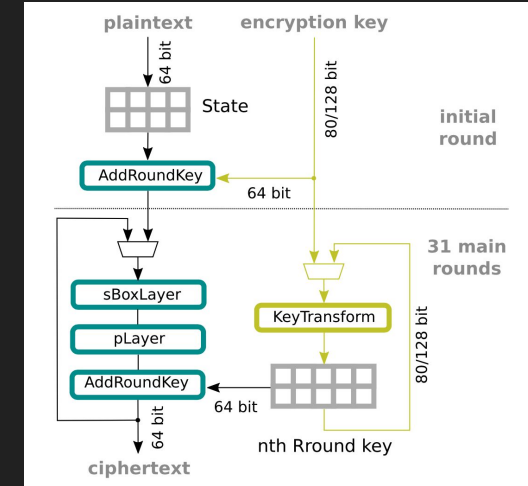


PRESENT

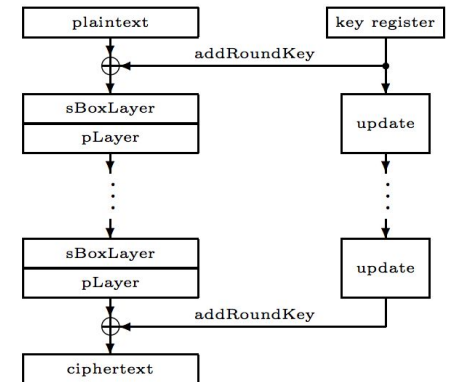
lightweight block cipher

64 bits block size

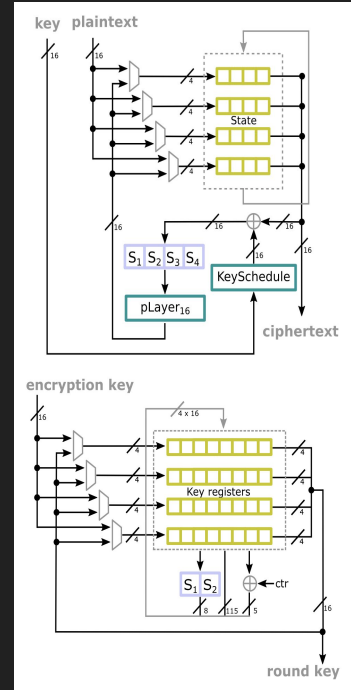
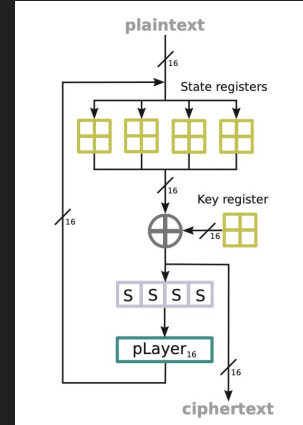
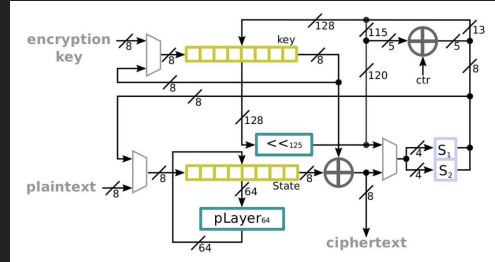
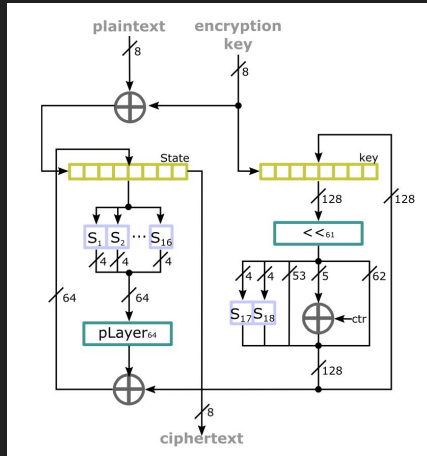
80/128 bits key size



```
generateRoundKeys()  
for  $i = 1$  to 31 do  
    addRoundKey( $STATE, K_i$ )  
    sBoxLayer( $STATE$ )  
    pLayer( $STATE$ )  
end for  
addRoundKey( $STATE, K_{32}$ )
```



PRESENT : iterative / serial / 16-bit / proposed



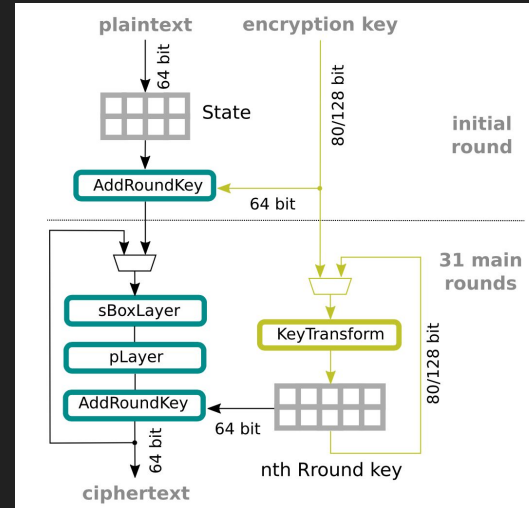
Cipher Design via HLS

A cipher is a loop where each round performs several operations.

Typically each round executes a single cycle.

Then we illustrate design perspectives to some common components in ciphers :

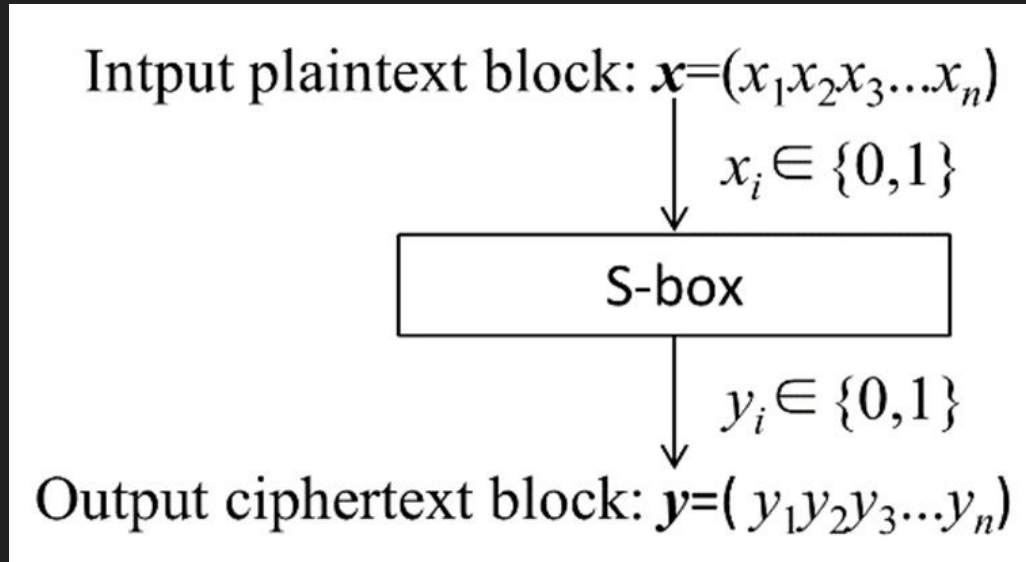
- sBox Layer
- Permutation Layer
- Optimization



sBox Layer

A function mapping input to the output $f : X \rightarrow Y$

sBox is typically lookup table in software and hardware implementation.



sBox Layer - Lookup Table Implementation

register style

```
ap_uint<4> S[] = {0xc, 0x5, 0x6, 0xb, 0x9, 0x0,  
0xa, 0xd, 0x3, 0xe, 0xf, 0x8, 0x4, 0x7, 0x1,  
0x2};
```

```
ap_uint<4> sbox(ap_uint<4> in){  
    ap_uint<4> out = S[in];  
    return out;  
}
```



if style

```
ap_uint<4> sbox(ap_uint<4> in){  
    ap_uint<4> out  
    switch(in) {  
        case 0: out = 0xc; break;  
        case 1: out = 0x5; break;  
        case 2: out = 0x6; break;  
        case 3: out = 0xb; break;  
        case 4: out = 0x9; break;  
        case 5: out = 0x0; break;  
        case 6: out = 0xa; break;  
        case 7: out = 0xd; break;  
        case 8: out = 0x3; break;  
        case 9: out = 0xe; break;  
        case 10: out = 0xf; break;  
        case 11: out = 0x8; break;  
        case 12: out = 0x4; break;  
        case 13: out = 0x7; break;  
        case 14: out = 0x1; break;  
        case 15: out = 0x2; break;}  
    return out;  
}
```

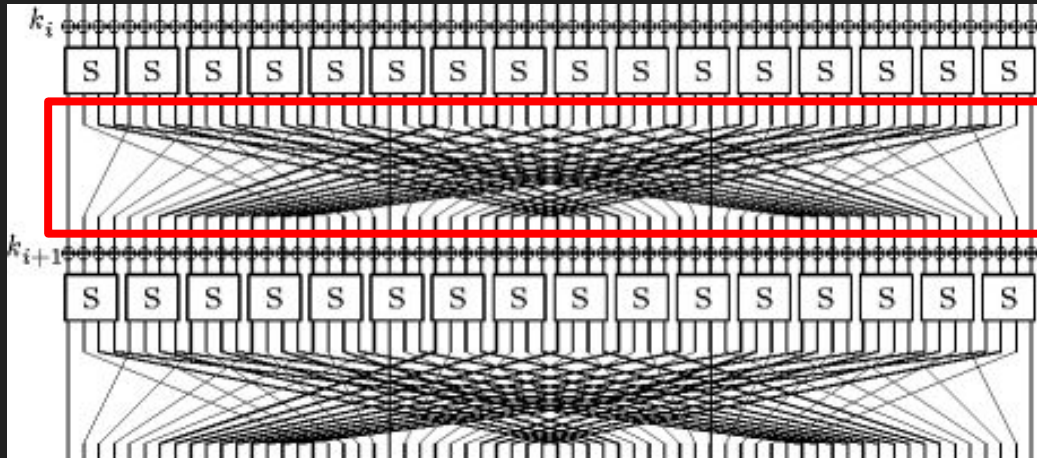
software style

but it costs 1 cycle in reading register value

Permutation Layer

Permute the input bits.

Permutation is simply wire connection in the hardware implementation.

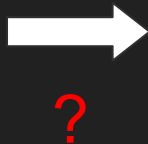


Permutation

register style

```
ap_uint<2> P[] = {0x3, 0x2, 0x1, 0x0};

ap_uint<4> permutation(ap_uint<4> in){
    ap_uint<4> out;
    out[P[0]] = in[0];
    out[P[1]] = in[1];
    out[P[2]] = in[2];
    out[P[3]] = in[3];
    return out;
}
```



if style

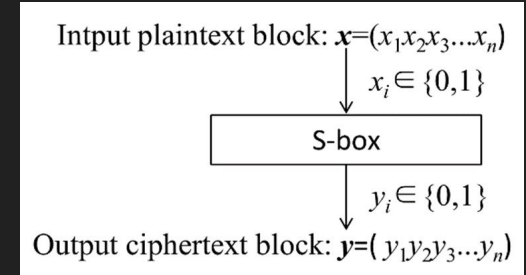
```
ap_uint<4> permutation(ap_uint<4> in){
    ap_uint<4> out;
    for(int i = 0; i < 4; i++){
#pragma HLS unroll factor=4
        switch(in[i]){
            case 0: out[3] = in[i]; break;
            case 1: out[2] = in[i]; break;
            case 2: out[1] = in[i]; break;
            case 3: out[0] = in[i]; break;}
    }
    return out;
}
```

compiler automatically generate correct hardware

directly using software style

Optimizations

- Loop Unrolling / Loop Pipelining ...
 - Implemented by simply adding #pragma .
- Cipher Optimizations
 - There are several implementation of sBox mapping function :
 - LookupTable
 - Bitslicing
 - Fixslicing
 - Each of them have its own advantages.



$$\begin{aligned}S_1 &\leftarrow S_1 \oplus (S_0 \wedge S_2) \\S_0 &\leftarrow S_0 \oplus (S_1 \wedge S_3) \\S_2 &\leftarrow S_2 \oplus (S_0 \vee S_1) \\S_3 &\leftarrow S_3 \oplus S_2 \\S_1 &\leftarrow S_1 \oplus S_3 \\S_3 &\leftarrow \neg S_3 \\S_2 &\leftarrow S_2 \oplus (S_0 \wedge S_1) \\\{S_0, S_1, S_2, S_3\} &\leftarrow \{S_3, S_1, S_2, S_0\},\end{aligned}$$

Experiment Results

RESOURCE USAGE AND PERFORMANCE RESULTS FOR THE FIVE ARCHITECTURES UNDER EVALUATION

	Work	Design	STATE (bit)	KEY (bit)	FF	LUT	SLC	FMAX (MHz)	LAT (cycles)	Thr (Mbps)	Thr* (Mbps)	Thr*/SLC (Kbps/Slice)
					xc6slx16-3cs324							
iterative	[18]	C1	64	128	200	202	58	160.21	55	186.42	15.78	272.05
serial	[18]	C2	64	128	203	157	44	164.23	303	34.69	2.86	65.09
16bit	[20]	C3	64	128	73	147	48	206.40	132	100.07	6.57	136.97
proposed	This work.	C5	64	128	201	220	61	210.66	136	99.13	6.38	104.61
					xc3s200-5ft256							
iterative	[18]	C1	64	128	200	381	191	179.95	55	209.40	15.78	82.61
serial	[18]	C2	64	128	203	258	131	177.34	303	37.46	2.86	21.86
16bit	[20]	C3	64	128	73	280	153	120.71	132	58.53	6.57	42.97
proposed	This work.	C5	64	128	201	264	151	194.63	136	91.59	6.38	42.26
					xc5vlx50t-3ft1136							
iterative	[18]	C1	64	128	200	283	88	271.67	55	316.12	15.78	179.31
serial	[18]	C2	64	128	203	237	72	289.69	303	61.19	2.86	39.78
16bit	[20]	C3	64	128	73	182	75	321.96	132	156.10	6.57	87.66
proposed	This work.	C5	64	128	201	239	73	431.78	136	203.19	6.38	87.41
					xc4vlx25-12ft668							
iterative	[18]	C1	64	128	200	382	192	284.33	55	330.86	15.78	82.18
serial	[18]	C2	64	128	203	258	131	288.52	303	60.94	2.86	21.86
16bit	[20]	C3	64	128	73	279	151	223.51	132	108.37	6.57	43.54
proposed	This work	C5	64	128	201	265	152	364.56	136	171.56	6.38	41.98

* Using a frequency of 13.56 MHz.

Our Results

Platform : Vivado HLS 2019.2 / PYNQ-Z2

	key bits	FF	LUT	SLC
iterative	128	210	316	87
serial	128	220	267	76
16-bit	128	74	288	80
proposed	128	202	314	87

Results are comparable with conventional method !

Conclusion

This project is actually a proof-of-concept to the feasibility of HLS :

HLS is able to generate comparable results compared with conventional method.

- It is good for people with software background to design hardware.

HLS has high scalability in optimizations.

- It simplifies the process to reach more dedicated design.

Reference

- [1] S. Feizi, A. Nemati, A. Ahmadi and V. A. Makki, "A high-speed FPGA implementation of a Bit-slice Ultra-Lightweight block cipher, RECTANGLE," 2015 5th International Conference on Computer and Knowledge Engineering (ICCKE), 2015, pp. 206-211, doi: 10.1109/ICCKE.2015.7365828.
- [2] C. A. Lara-Nino, A. Diaz-Perez and M. Morales-Sandoval, "Lightweight Hardware Architectures for the Present Cipher in FPGA," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 64, no. 9, pp. 2544-2555, Sept. 2017, doi: 10.1109/TCSI.2017.2686783.