Session 2: The loop unrolling architecture

Antonio de la Piedra

Institute for Computing and Information Sciences – Digital Security
Radboud University Nijmegen

February 25, 2015

Session 2

- Introduce the loop unrolling architecture
- We rely on NOEKEON
- The implementation that is used in this session can be download from `https://github.com/adelapie/noekeon_loop/archive/master.zip`

Overview

1. Area
2. The loop unrolling architecture
3. A loop unrolling architecture for NOEKEON
    - K = 2
    - K = 4
4. Area/Throughput ratio

How this is related to your projects

1. You can see this architecture between the iterative and the pipeline one in terms of area and throughput.
2. Once you have implemented an iterative architecture of the block cipher you have chosen, feel free to implement a loop unrolling one, analyze the speed up in throughput.

Important

- There is a new tutorial about creating and testing combinational circuits at `https://rucryptoengineering.files.wordpress.com/2015/02/tutorial2.pdf`
    - Follow it before start implementing the functions of the round of the block cipher you have chosen.
    - It considers you have succeeded with the first one.
- The theory behind this chapter is based on
    - *Cryptographic Engineering, Koc, Cetin Kaya (Ed.) 2009*

Part I: Design parameters

Area

Area is related to:

- Cost: typically proportional to the circuit area.
- Maximum area limit: related to cost, fabrication technology, power consumption.

Area

Units:

- Application-Specific Integrated Circuit (ASIC): μm^2, transistor count, logic gate count
- Field-Programmable Gate Array (FPGA): number of configurable logic blocks (CLB), also related to embedded blocks utilization e.g. BRAM, DSPs, etc.
  - **More with Pedro later**.

Part II: The loop unrolling architecture

Loop unrolling

- The combinational part of the circuit implements K rounds of the cipher.
- The number of cycles required for encrypting data block is decreased by a factor K.

Loop unrolling

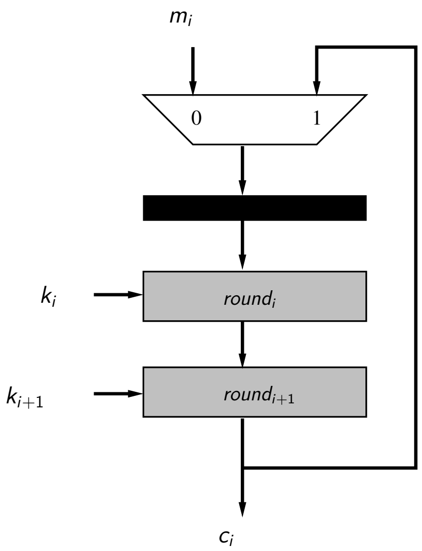

Figure : Loop unrolling for $K = 2$

Loop unrolling

- The number of required clock cycles is reduced by a K factor.

Throughput

$$Throughput = \frac{blocksize}{\frac{rounds}{K} \cdot T_{CLK}}$$

- The minimum clock period increases by a factor smaller by K but throughput is increased, latency decreased.

Latency

$$Latency = \frac{rounds}{K} \cdot T_{CLK}$$

Loop unrolling

Disadvantages:

- The area increases proportional to the number of unrolled rounds.
- The number of subkeys k_i that are used by cycle must be stored, which increases the area according to the factor K. For instance, in the case of AES-128 (ten rounds, 128 bit keys):
  - For $K = 2$, 5 cycles, 256 bits
  - For $K = 5$, 2 cycles, 640 bits

Part III: A loop unrolling architecture for NOEKEON

Methodology

- Replicate the round by a factor of K
- Adapt key schedule and round constant generation according to K

Requirements

- We rely on direct mode: no key schedule
  - We don't need to store the subkeys in this case
- We need to generate more than one round constant:
  - $K = 2$
  - $K = 4$

Round constants $K = 2$

- In each cycle, two rounds of the cipher are applied
- We need two constants per cycle
- The round constant generator that was used in the iterative design cannot be used

Round constants $K = 2$

- Create two shift registers
- Initialize them with the correspondent round constants

Round constants $K = 2$

- e.g. for encryption:
  - $[80, 1b, 36, 6c, d8, ab, 4d, 9a, 2f, 5e, bc, 63, c6, 97, 35, 6a]$
- With two shift registers:
  - $RC_0 = [80, 36, d8, 4d, 2f, bc, c6, 35]$
  - $RC_1 = [1b, 6c, ab, 9a, 5e, 63, 97, 6a]$

Round constants $K = 4$

- e.g. for encryption:
  - $[80, 1b, 36, 6c, d8, ab, 4d, 9a, 2f, 5e, bc, 63, c6, 97, 35, 6a]$
- With four shift registers:
  - $RC_0 = [80, d8, 2f, c6]$
  - $RC_1 = [1b, ab, 5e, 97]$
  - $RC_2 = [36, 4d, bc, 35]$
  - $RC_3 = [6c, 9a, 63, 6a]$

Shift register for round constants

```
entity rc_shr is
        port(clk : in std_logic;
             rst : in std_logic;
             rc_in : in std_logic_vector(63 downto 0);
             rc_out : out std_logic_vector(7 downto 0));
end rc_shr;

architecture Behavioral of rc_shr is
signal sh_reg : std_logic_vector(63 downto 0);
begin
pr_shr: process(clk, rst)
        begin
          if rising_edge(clk) then
            if rst = '1' then
              sh_reg <= rc_in;
            else
              sh_reg <= sh_reg(55 downto 0) & sh_reg(63 downto 56);
            end if;
          end if;
        end process;

        rc_out <= sh_reg(63 downto 56);

end Behavioral;
```

Shift register for round constants

```vhdl
        signal init_val_shr_0 : std_logic_vector(63 downto 0);
        signal init_val_shr_1 : std_logic_vector(63 downto 0);
begin
        init_val_shr_0 <= X"8036d84d2fbcc635";
        init_val_shr_1 <= X"1b6cab9a5e63976a";

        RC_SHR_0 : rc_shr port map (clk, rst, init_val_shr_0, rc_s);
        RC_SHR_1 : rc_shr port map (clk, rst, init_val_shr_1, rc_2_s);

        rc_ext_s <= X"000000" & rc_s;
        rc_2_ext_s <= X"000000" & rc_2_s;
```

Rounds for $K = 2$

```
signal stage_0_a_0_out_s : std_logic_vector(31 downto 0);
signal stage_0_a_1_out_s : std_logic_vector(31 downto 0);
signal stage_0_a_2_out_s : std_logic_vector(31 downto 0);
signal stage_0_a_3_out_s : std_logic_vector(31 downto 0);
```

Rounds for $K = 2$

```
ROUND_F_0 : round_f port map (enc,
                              rc_ext_s,
                              a_0_in_s,
                              a_1_in_s,
                              a_2_in_s,
                              a_3_in_s,
                              k_0_mux_s,
                              k_1_mux_s,
                              k_2_mux_s,
                              k_3_mux_s,
                              stage_0_a_0_out_s,
                              stage_0_a_1_out_s,
                              stage_0_a_2_out_s,
                              stage_0_a_3_out_s);

ROUND_F_1 : round_f port map (enc,
                              rc_2_ext_s,
                              stage_0_a_0_out_s,
                              stage_0_a_1_out_s,
                              stage_0_a_2_out_s,
                              stage_0_a_3_out_s,
                              k_0_mux_s,
                              k_1_mux_s,
                              k_2_mux_s,
                              k_3_mux_s,
                              a_0_out_s,
                              a_1_out_s,
                              a_2_out_s,
                              a_3_out_s);
```

For $K = 4$

- Four shift registers
- Four rounds interconnected
- *noekeon_loop_k_4.vhd*

Part IV: Area and throughput analysis

Number of cycles per round

- For $K = 2$, $16/2 = 8$ cycles
- For $K = 4$, $16/4 = 4$ cycles

Synthesis results

- Iterative (Session 1):
  - Area: 264 FFs, 966 LUTs, $N = 16$
  - Max. Freq: 144.211 MHz
- Loop unrolling $K = 2$:
  - Area: 355 FFs, 1,496 LUTs, $N = 8$
  - Max. Freq: 97.404 MHz
- Loop unrolling $K = 4$:
  - Area: 367 FFs, 2,829 LUTs, $N = 4$
  - Max. Freq: 56.461 MHz

Throughput

Throughput iterative

$$Throughput = \frac{blocksize}{rounds \cdot T_{CLK}}$$

Throughput loop unrolling

$$Throughput = \frac{blocksize}{\frac{rounds}{K} \cdot T_{CLK}}$$

Throughput

- Iterative (Session 1)
  - $\frac{128}{16 \cdot \frac{1}{144.211 \cdot 10^6}} = 1.153,7$ Gbits/s
- Loop unrolling $K = 2$
  - $\frac{128}{\frac{16}{2} \cdot \frac{1}{97.404 \cdot 10^6}} = 1.558,5$ Gbits/s
- Loop unrolling $K = 4$:
  - $\frac{128}{\frac{16}{4} \cdot \frac{1}{56.461 \cdot 10^6}} = 1.806,8$ Gbits/s

Conclusions

- Rely on the iterative architecture when the area is the only parameter you care about.
- Rely on pipelining when you only care about throughput (next week).
- Consider the loop unrolling architecture as an intermediate step between both.
    - You should start with a full unrolled architecture for pipelining.

Q/A

Questions?