Session 3: Pipeline architectures

Antonio de la Piedra

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

March 4, 2015

Session 3

- Introduce all the variants of the pipeline architecture
- We rely on NOEKEON
- The implementations that are used in this session can be download from:
  - Inner-round examples: `https://github.com/adelapie/noekeon_inner_round/archive/master.zip`

Overview

1. The pipeline architecture
2. Inner-round pipelining
3. Outer-round pipelining
4. Mixed inner-round outer-round pipelining
5. Summary

How this is related to your projects

1. Pipelining constructs on top of the iterative and the loop unrolling architecture.
2. Sessions 1 and 2 are considered the main building blocks in this case.
3. Theory behind this session: Gaj et al. *FPGA and ASIC Implementations of AES* in Cryptographic Engineering, Kaya (Editor), 2009.

Part I: The pipeline architecture

Introduction

- High-performance implementations
  - Cryptographic accelerators e.g. high-speed network equipment such as routers, etc.
  - Cryptanalysis
- Area is not a design constraint

Introduction

- We rely on the iterative and the loop unrolling architecture.
- We insert registers in the round (inner-round pipelining) and between rounds (outer-round pipelining) in order to reduce the minimum clock delay and improve the throughput.

Part II: Inner-round pipelining

Inner-round pipelining

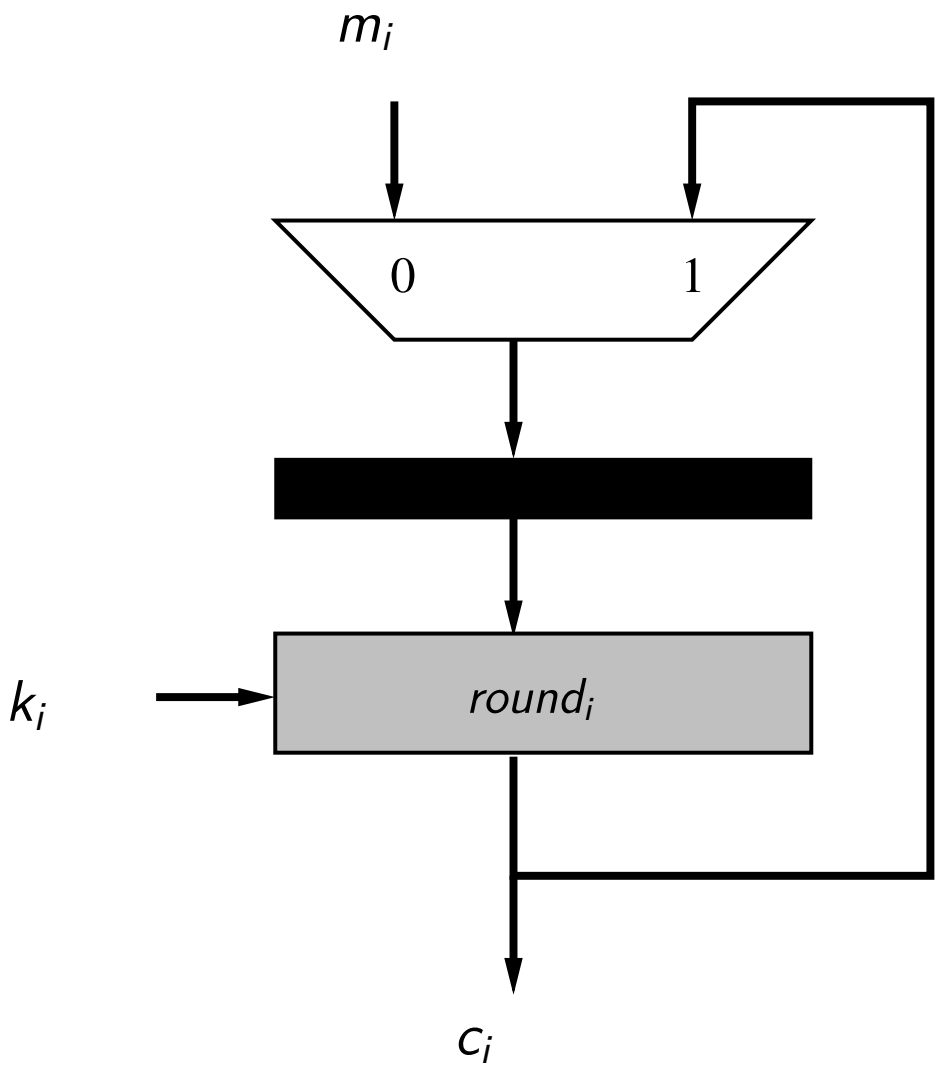

Figure : Iterative architecture of a block cipher

Inner-round pipelining

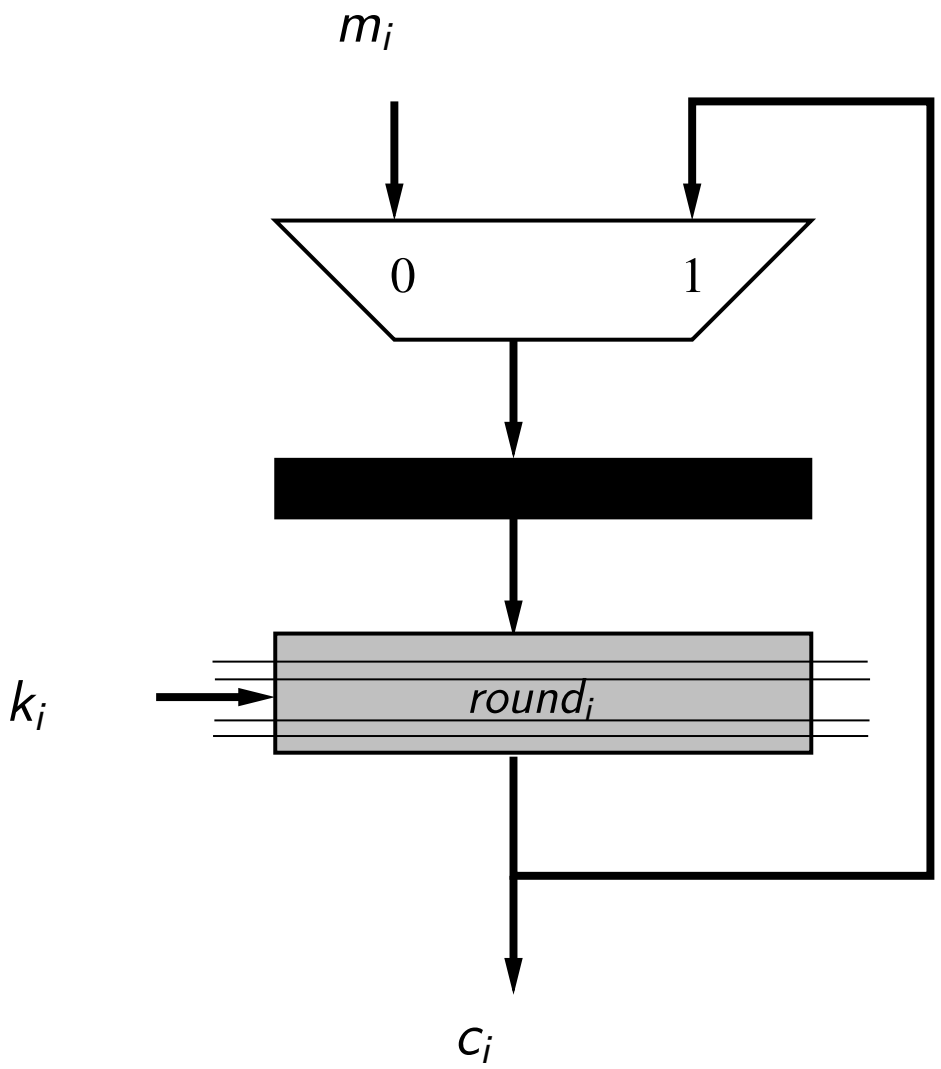

Figure : Inner-round pipelining for $k = 4$

Inner-round pipelining

- Divide the round in independent operations e.g. the round functions
- Insert k registers for every operation
- Find the optimal k that balances throughput and area

Throughput

$$Throughput(k) = \frac{blocksize}{\#\ rounds \cdot T_{CLK\_inner\_round}(k)}$$

Inner-round pipelining: NOEKEON

```
Round(k, a, c1, c2, enc) {
  if (enc)
    a ^= c1;

  theta(k, a);

  if (!enc)
    a ^= c2;

  pi1(a);
  gamma(a);
  pi2(a);
}
```

Inner-round pipelining: NOEKEON

- There are three possibilities ($k = 1, 2, 3$) for inserting registers:
  - Before Π_1
  - Before γ
  - Before Π_2
- There is already a register before θ
- Other options: pipeline the internal operations of γ and θ.

Inner-round pipelining: NOEKEON

- For each possibility of k we have to deal with:
  - Round constant generation
  - Subkey generation:
    - NOEKEON direct mode

Inner-round pipelining: NOEKEON

- We use 128-bit registers (same length as the NOEKEON state):

```
entity reg_128 is
  port(clk : in std_logic;
       rst : in std_logic;
       data_in_0 : in std_logic_vector(31 downto 0);
       data_in_1 : in std_logic_vector(31 downto 0);
       data_in_2 : in std_logic_vector(31 downto 0);
       data_in_3 : in std_logic_vector(31 downto 0);
       data_out_0 : out std_logic_vector(31 downto 0);
       data_out_1 : out std_logic_vector(31 downto 0);
       data_out_2 : out std_logic_vector(31 downto 0);
       data_out_3 : out std_logic_vector(31 downto 0));
end reg_128;
```

Inner-round pipelining: NOEKEON

```
architecture Behavioral of reg_128 is
    signal reg_32_0_s : std_logic_vector(31 downto 0);
    signal reg_32_1_s : std_logic_vector(31 downto 0);
    signal reg_32_2_s : std_logic_vector(31 downto 0);
    signal reg_32_3_s : std_logic_vector(31 downto 0);
begin
    pr_reg: process(clk, rst)
    begin
        if rising_edge(clk) then
            if rst = '1' then
                reg_32_0_s <= (others => '0');
                reg_32_1_s <= (others => '0');
                reg_32_2_s <= (others => '0');
                reg_32_3_s <= (others => '0');
            else
                reg_32_0_s <= data_in_0;
                reg_32_1_s <= data_in_1;
                reg_32_2_s <= data_in_2;
                reg_32_3_s <= data_in_3;
            end if;
        end if;
end process;

data_out_0 <= reg_32_0_s;
data_out_1 <= reg_32_1_s;
data_out_2 <= reg_32_2_s;
data_out_3 <= reg_32_3_s;
```

Inner-round pipelining: NOEKEON

```
THETA_0 : theta port map (a_0_in_s,
                          a_1_in,
                          a_2_in,
                          a_3_in,
                          k_0_in,
                          k_1_in,
                          k_2_in,
                          k_3_in,
                          theta_0_s,
                          theta_1_s,
                          theta_2_s,
                          theta_3_s);
REG_STAGE_0: reg_128 port map (clk,
                               rst,
                               theta_0_s,
                               theta_1_s,
                               theta_2_s,
                               theta_3_s,
                               stage_0_out_0_s,
                               stage_0_out_1_s,
                               stage_0_out_2_s,
                               stage_0_out_3_s);
PI_1_0 : pi_1 port map (stage_0_out_1_s,
                        stage_0_out_2_s,
                        stage_0_out_3_s,
                        pi_1_1_s,
                        pi_1_2_s,
                        pi_1_3_s);
```

Inner-round pipelining: NOEKEON

- Round constant generation
  - We should maintain the constant during all the internal stages of the round for $k = 1, 2, 3$ stages.
  - Straightforward option: a shift register initialized with:
    - 80801b1b36366c6cd8d8abab4d4d etc.
    - 8080801b1b1b3636366c6c6cd8d8 etc.
    - 808080801b1b1b1b363636366c6c etc.

Part III: Outer-round pipelining

Outer-round pipelining

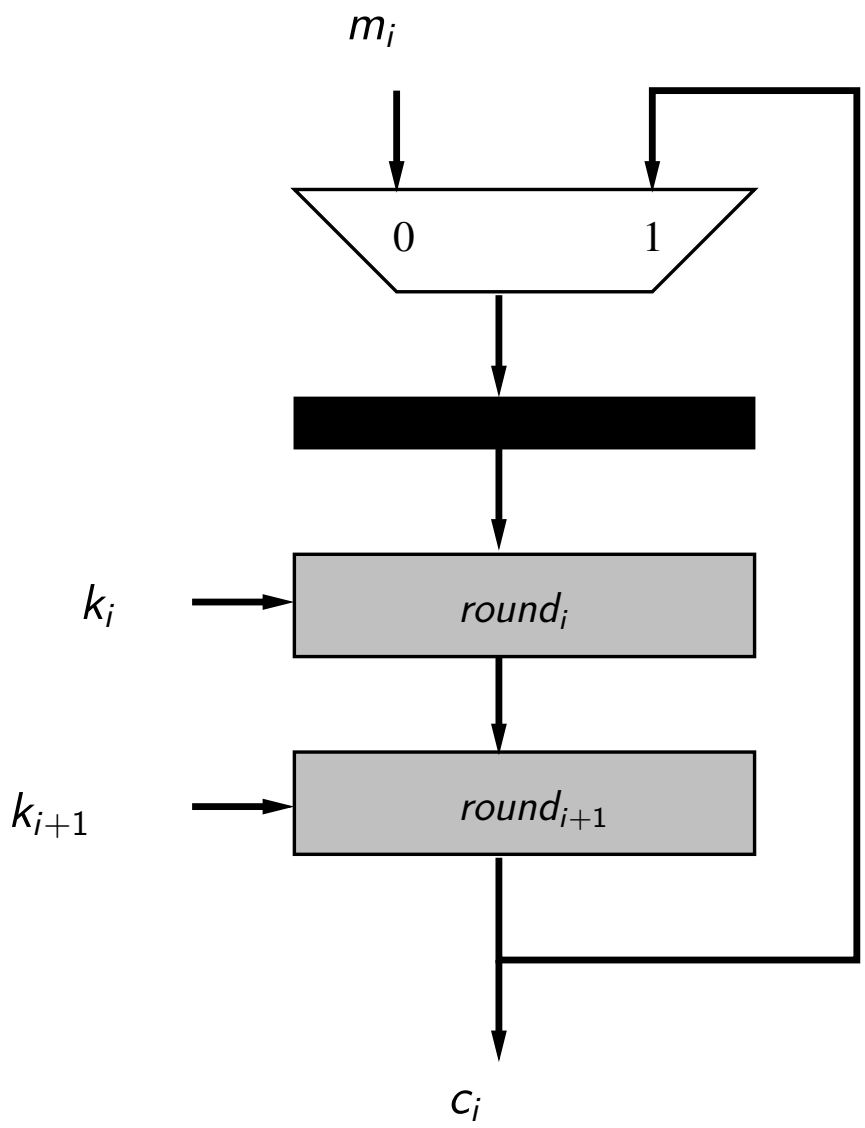

Figure : Loop unrolling architecture of a block cipher for $K = 2$

Outer-round pipelining

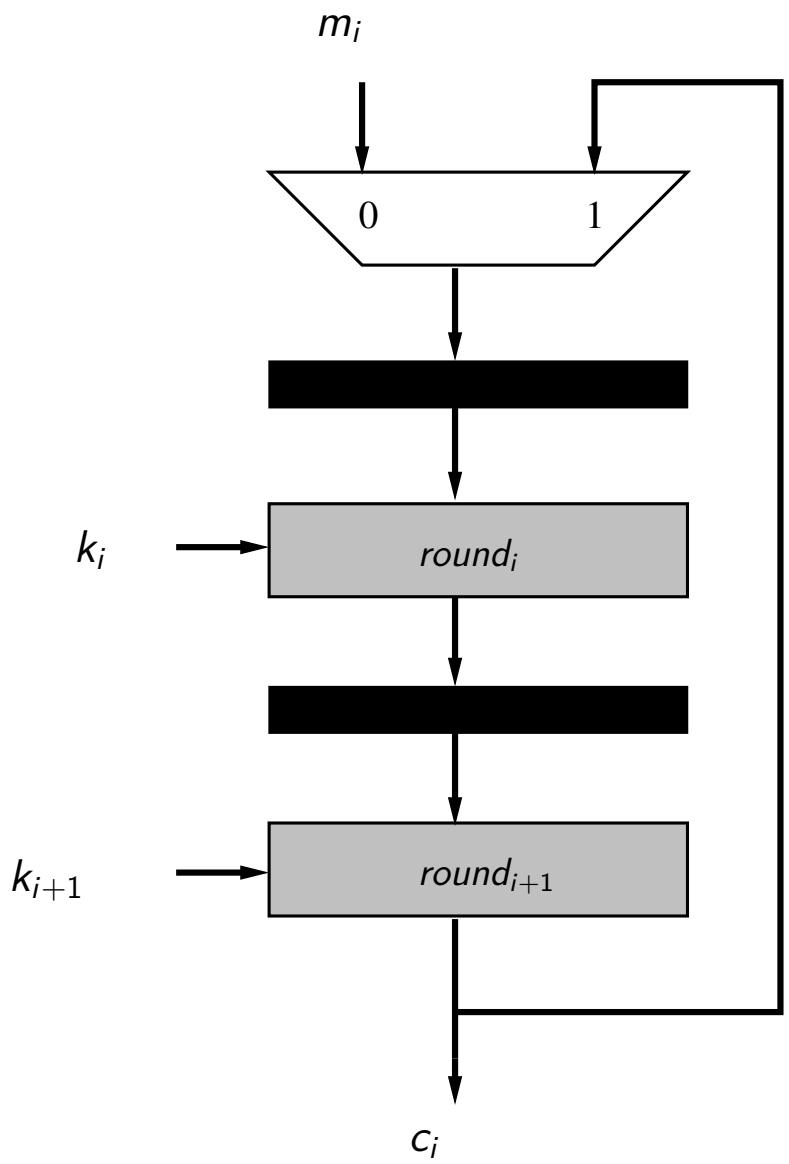

Figure : Outer-round pipelining for $K = 2$

Outer-round pipelining

- Starting from a loop unrolling architecture of factor K
- Insert K registers between every round

Throughput

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

Outer-round pipelining: NOEKEON

- Round constant generation
  - This time connect each round constant to each stage.
- Subkeys
  - Direct mode in NOEKEON.
  - Otherwise, directly connect the subkeys to the correspondent stage

Outer-round pipelining: NOEKEON

- We use 128-bit registers (same length as the NOEKEON state):

```
ROUND_F_1 : round_f port map (enc,
                              rc_1_s,
                              stage_1_a_0_out_s,
                              stage_1_a_1_out_s,
                              stage_1_a_2_out_s,
                              stage_1_a_3_out_s,
                              k_0_mux_s,
                              k_1_mux_s,
                              k_2_mux_s,
                              k_3_mux_s,
                              stage_2_a_0_out_s,
                              stage_2_a_1_out_s,
                              stage_2_a_2_out_s,
                              stage_2_a_3_out_s);

STAGE_1 : reg_128 port map (clk,
                            rst,
                            stage_2_a_0_out_s,
                            stage_2_a_1_out_s,
                            stage_2_a_2_out_s,
                            stage_2_a_3_out_s,
                            stage_3_a_0_out_s,
                            stage_3_a_1_out_s,
                            stage_3_a_2_out_s,
                            stage_3_a_3_out_s);
```

Outer-round pipelining: NOEKEON

```
signal rc_0_s : std_logic_vector(31 downto 0);
signal rc_1_s : std_logic_vector(31 downto 0);
signal rc_2_s : std_logic_vector(31 downto 0);
signal rc_3_s : std_logic_vector(31 downto 0);
signal rc_4_s : std_logic_vector(31 downto 0);

rc_0_s <= X"00000080";
rc_1_s <= X"0000001b";
rc_2_s <= X"00000036";
rc_3_s <= X"0000006c";
rc_4_s <= X"000000d8";
```

Part IV: Mixed Outer-round Inner-round pipelining

Mixed Outer-round Inner-round pipelining

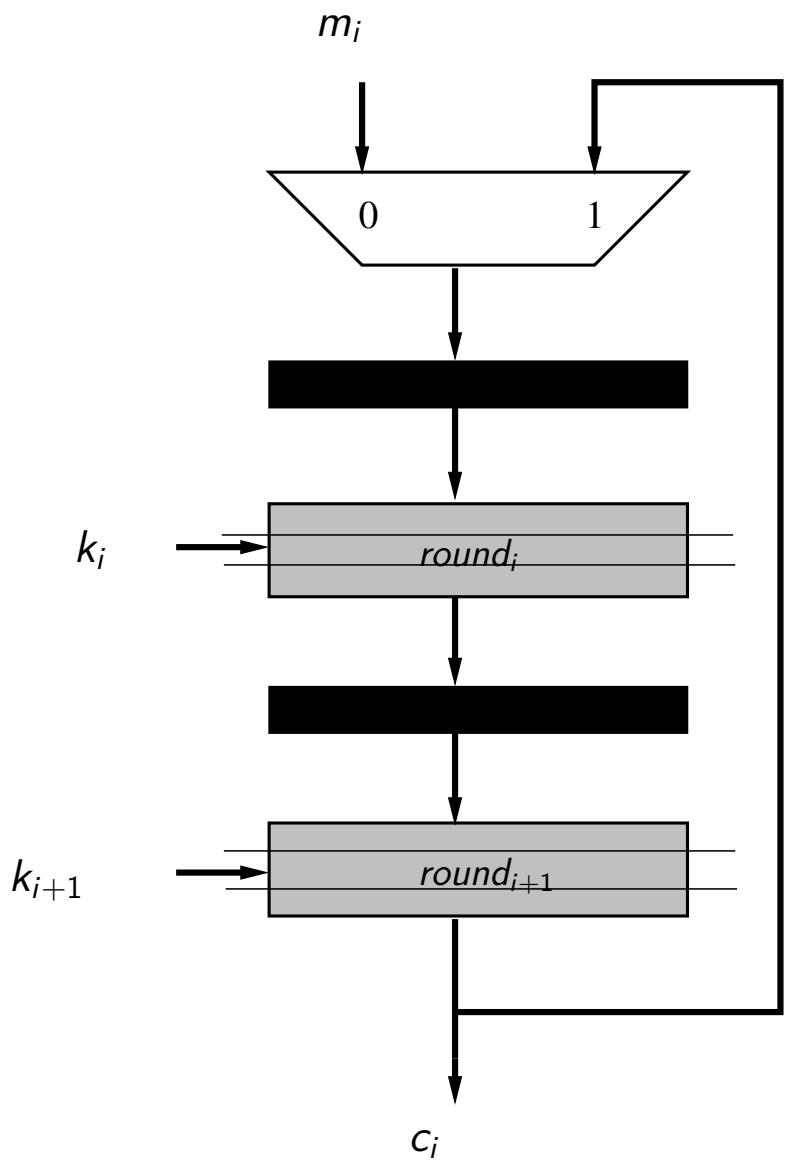

Figure : Inner-round Outer-round mixed pipelining for $k = 2, K = 2$

Mixed Outer-round Inner-round pipelining

- Start from an outer-round pipelining architecture
- Replace the round by the optimal case found in the inner-round architecture

Throughput

$$Throughput(K,k) = \frac{K \cdot blocksize}{\#\ rounds \cdot T_{CLK}(k)}$$

Mixed Outer-round Inner-round pipelining: NOEKEON

- The round constant is fixed by round
- Also the subkeys (not needed in NOEKEON)

Summary

- Inner-round pipelining: best throughput/area balance for an optimal k:
  - the area related to the registers is generally smaller than adding more rounds
- Outer-round pipelining: starting point from mixed outer-round inner-round pipelining.
- Inner-round outer-round pipelining:
  - Best throughput
  - Greater area when fully unrolled

Q/A

Questions?