A 16-bit Reconfigurable Encryption Processor for π-Cipher

Mohamed El Hadedy, Hristina Mihajloska, Danilo Gligoroski, Amit Kulkarni, Dirk Stroobandt and **Kevin Skadron**

Department of Computer Science

University of Virginia

May 23, 2016

CAESAR competition

- Second round 29 candidates
  - mostly AES based
  - other sponge/permutation based
  - Three of them are ARX based

- π-Cipher is one of the ARX based candidates

π-Cipher

- An authenticated encryption cipher with associated data
- Designers:
  - Danilo Gligoroski, NTNU
  - Hristina Mihajloska, FINKI
  - Simona Samardziska, FINKI
  - Haakon Jacobsen, NTNU
  - Mohamed El-Hadedy, UVA, UIUC
  - Rune Erlend Jensen, NTNU
  - Daniel Otte, RUB

π-Cipher

- Main design goals:
  - High security
  - Simplicity (ARX design)
  - Suitable for HW and SW
  - Parallel
  - Incremental
  - Tag second-preimage resistant
  - Intermediate tags

π-Cipher

- The main role in the security and design has permutation function (π-function)
- Only uses Add, Rotate and XOR operations
- 3 rounds
  - each round has 8 ARX operation blocks (* op.)
  - every ARX block consists of 52 ARX operations
- Different word sizes
  - π16-Cipher – lightweight version

π16-Cipher

- In total:
  - 1248 ARX operations (for one π-func)
  - 128 bits of memory for * operation
  - 512 bits of memory for round constants
  - Additional:
    - 64 bits for the counter
    - 256 bits for the state

π16-Cipher ARX Engine

$*$ operation for 64–bit words

Input: $\mathbf{X} = (X_0, X_1, X_2, X_3)$ and $\mathbf{Y} = (Y_0, Y_1, Y_2, Y_3)$ where X_i and Y_i are 64–bit variables.
Output: $\mathbf{Z} = (Z_0, Z_1, Z_2, Z_3)$ where Z_i are 64–bit variables.
Temporary 64–bit variables: $T_0, \ldots, T_{11}$.

μ–transformation for X:

1.
$$\begin{array}{lcl} T_0 & \leftarrow & ROTL^{7}(\texttt{0xF0E8E4E2E1D8D4D2} + X_0 + X_1 + X_2); \\ T_1 & \leftarrow & ROTL^{19}(\texttt{0xD1CCCAC9C6C5C3B8} + X_0 + X_1 + X_3); \\ T_2 & \leftarrow & ROTL^{31}(\texttt{0xB4B2B1ACAAA9A6A5} + X_0 + X_2 + X_3); \\ T_3 & \leftarrow & ROTL^{53}(\texttt{0xA39C9A999695938E} + X_1 + X_2 + X_3); \end{array}$$

2.
$$\begin{array}{lcl} T_4 & \leftarrow & T_0 \oplus T_1 \oplus T_3; \\ T_5 & \leftarrow & T_0 \oplus T_1 \oplus T_2; \\ T_6 & \leftarrow & T_1 \oplus T_2 \oplus T_3; \\ T_7 & \leftarrow & T_0 \oplus T_2 \oplus T_3; \end{array}$$

ν–transformation for Y:

1.
$$\begin{array}{lcl} T_0 & \leftarrow & ROTL^{11}(\texttt{0x8D8B87787472716C} + Y_0 + Y_2 + Y_3); \\ T_1 & \leftarrow & ROTL^{23}(\texttt{0x6A696665635C5A59} + Y_1 + Y_2 + Y_3); \\ T_2 & \leftarrow & ROTL^{37}(\texttt{0x5655534E4D4B473C} + Y_0 + Y_1 + Y_2); \\ T_3 & \leftarrow & ROTL^{59}(\texttt{0x3A393635332E2D2B} + Y_0 + Y_1 + Y_3); \end{array}$$

2.
$$\begin{array}{lcl} T_8 & \leftarrow & T_1 \oplus T_2 \oplus T_3; \\ T_9 & \leftarrow & T_0 \oplus T_2 \oplus T_3; \\ T_{10} & \leftarrow & T_0 \oplus T_1 \oplus T_3; \\ T_{11} & \leftarrow & T_0 \oplus T_1 \oplus T_2; \end{array}$$

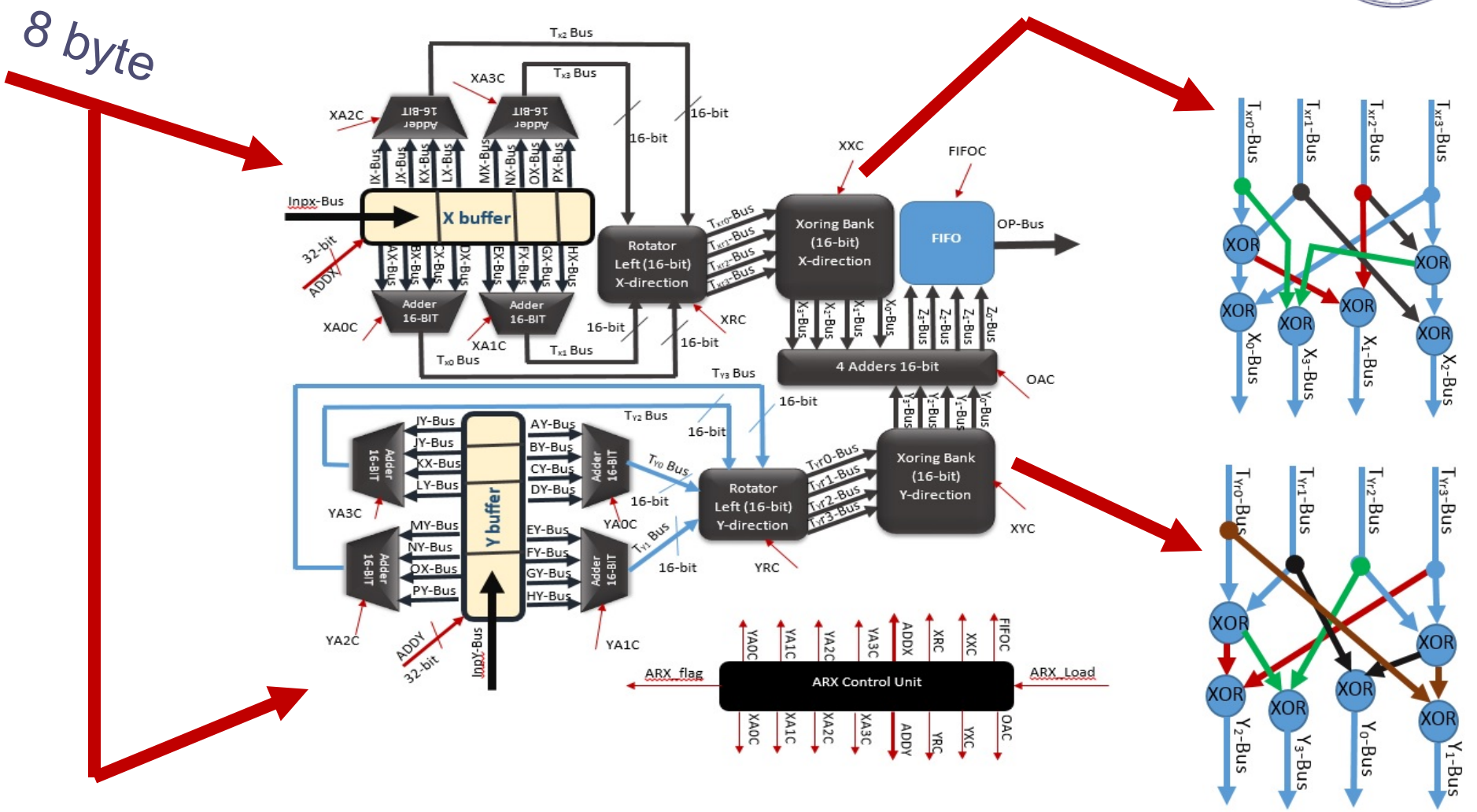
σ–transformation for both $\mu(X)$ and $\nu(Y)$:

1.
$$\begin{array}{lcl} Z_3 & \leftarrow & T_4 + T_8; \\ Z_0 & \leftarrow & T_5 + T_9; \\ Z_1 & \leftarrow & T_6 + T_{10}; \\ Z_2 & \leftarrow & T_7 + T_{11}; \end{array}$$

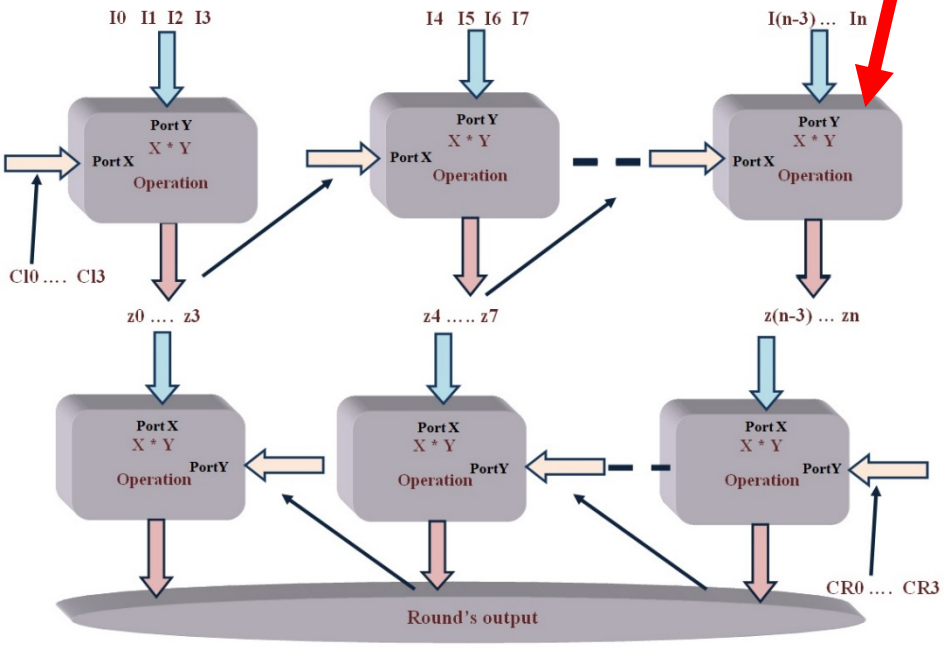
ARX Engine Architecture:

- ✓ Dual core processor with the cores running in parallel
- ✓ 64-bit buffer and 16 reading ports for each of the cores
- ✓ 8 adders (working on 16-bit words) (Ripple-Carry adders)
- ✓ 2 rotators
- ✓ 2 XOR-ing banks
- ✓ ARX Control Unit

Quad ARX Engine



π16-Cipher ARX Engine



*** operation for 64–bit words**

Input: $\mathbf{X} = (X_0, X_1, X_2, X_3)$ and $\mathbf{Y} = (Y_0, Y_1, Y_2, Y_3)$ where X_i and Y_i are 64–bit variables.
Output: $\mathbf{Z} = (Z_0, Z_1, Z_2, Z_3)$ where Z_i are 64–bit variables.
Temporary 64–bit variables: $T_0, \ldots, T_{11}$.

μ–transformation for X:

1.
$$\begin{array}{lll} T_0 & \leftarrow & ROTL^{7}(\texttt{0xF0E8E4E2E1D8D4D2} + X_0 + X_1 + X_2); \\ T_1 & \leftarrow & ROTL^{19}(\texttt{0xD1CCCAC9C6C5C3B8} + X_0 + X_1 + X_3); \\ T_2 & \leftarrow & ROTL^{31}(\texttt{0xB4B2B1ACAAA9A6A5} + X_0 + X_2 + X_3); \\ T_3 & \leftarrow & ROTL^{53}(\texttt{0xA39C9A999695938E} + X_1 + X_2 + X_3); \end{array}$$

2.
$$\begin{array}{lll} T_4 & \leftarrow & T_0 \oplus T_1 \oplus T_3; \\ T_5 & \leftarrow & T_0 \oplus T_1 \oplus T_2; \\ T_6 & \leftarrow & T_1 \oplus T_2 \oplus T_3; \\ T_7 & \leftarrow & T_0 \oplus T_2 \oplus T_3; \end{array}$$

ν–transformation for Y:

1.
$$\begin{array}{lll} T_0 & \leftarrow & ROTL^{11}(\texttt{0x8D8B87787472716C} + Y_0 + Y_2 + Y_3); \\ T_1 & \leftarrow & ROTL^{23}(\texttt{0x6A696665635C5A59} + Y_1 + Y_2 + Y_3); \\ T_2 & \leftarrow & ROTL^{37}(\texttt{0x5655534E4D4B473C} + Y_0 + Y_1 + Y_2); \\ T_3 & \leftarrow & ROTL^{59}(\texttt{0x3A393635332E2D2B} + Y_0 + Y_1 + Y_3); \end{array}$$

2.
$$\begin{array}{lll} T_8 & \leftarrow & T_1 \oplus T_2 \oplus T_3; \\ T_9 & \leftarrow & T_0 \oplus T_2 \oplus T_3; \\ T_{10} & \leftarrow & T_0 \oplus T_1 \oplus T_3; \\ T_{11} & \leftarrow & T_0 \oplus T_1 \oplus T_2; \end{array}$$

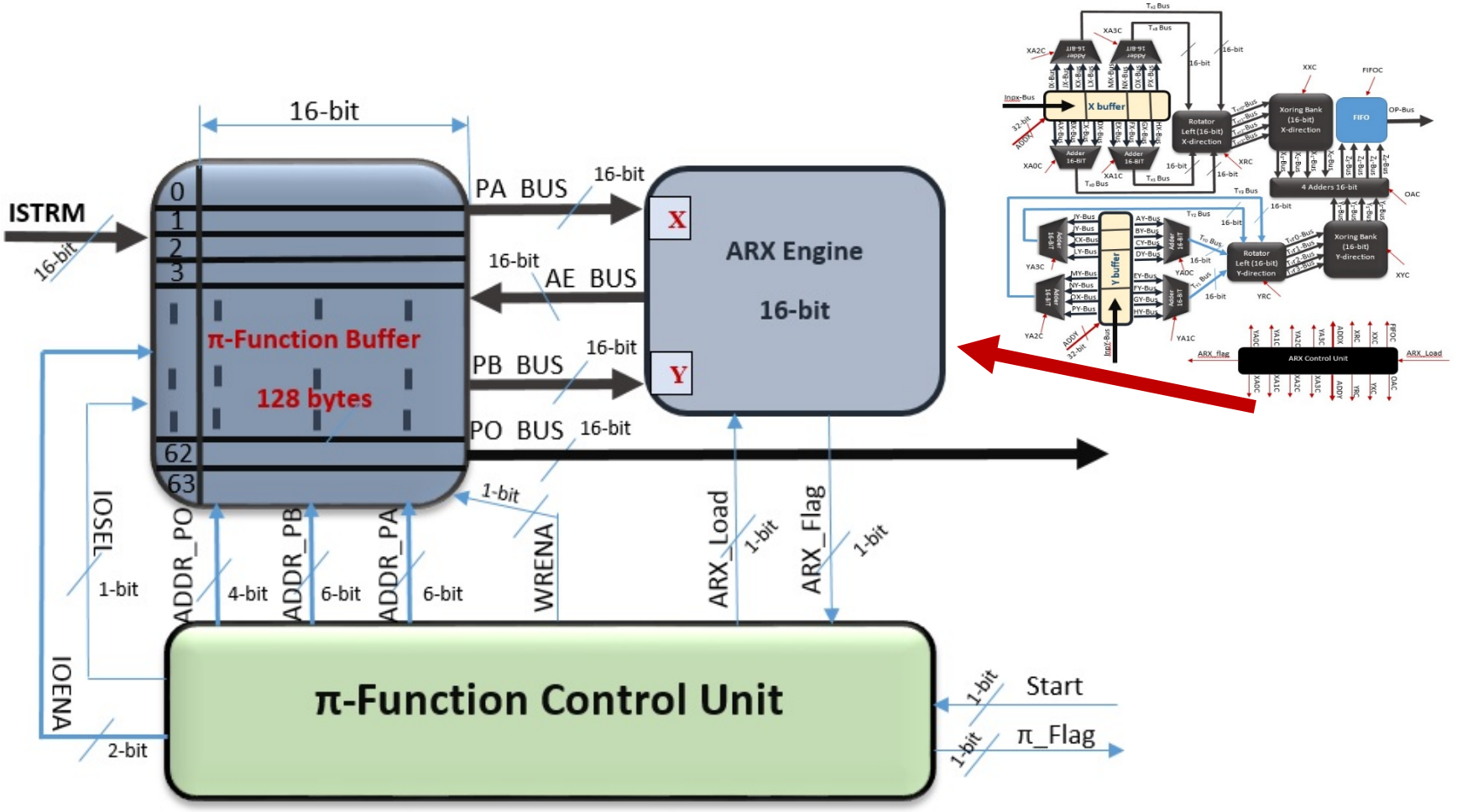
σ–transformation for both $\mu(X)$ and $\nu(Y)$:

1.
$$\begin{array}{lll} Z_3 & \leftarrow & T_4 + T_8; \\ Z_0 & \leftarrow & T_5 + T_9; \\ Z_1 & \leftarrow & T_6 + T_{10}; \\ Z_2 & \leftarrow & T_7 + T_{11}; \end{array}$$

π-function core

- ✓ 128 bytes memory buffer
- ✓ π-func. Control Unit relies on Moore-style state machine with 32 states
- ✓ Takes 675 cycles to complete
- ✓ Can run at 250 MHz and implemented in 971 slices (Virtex-7)

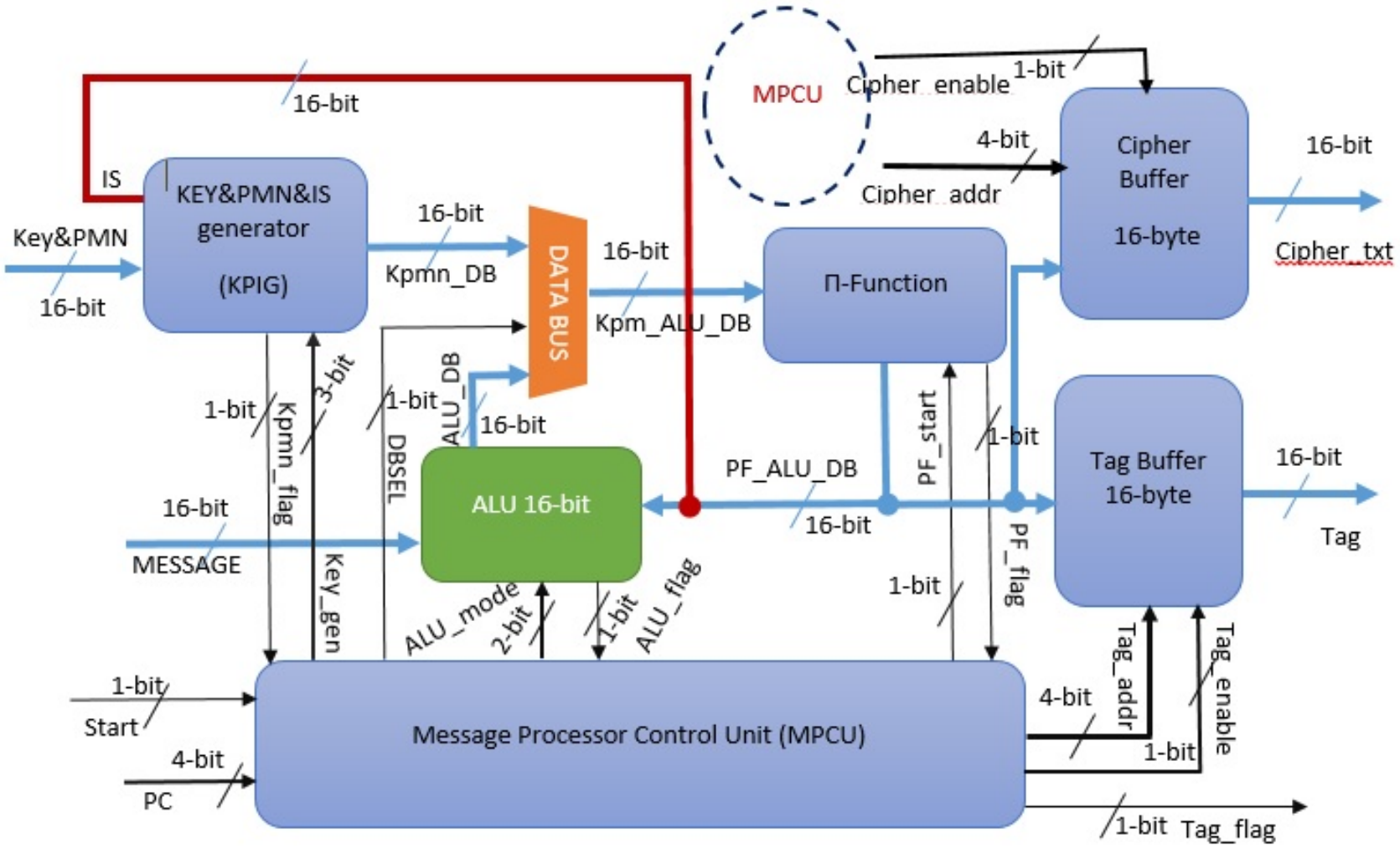
π-Function


ISTRM
16-bit
16-bit
0
1
2
3
62
63
π-Function Buffer
128 bytes
PA BUS
16-bit
AE BUS
16-bit
PB BUS
16-bit
PO BUS
16-bit
X
Y
ARX Engine
16-bit
1-bit
WRENA
ADDR_PO
4-bit
ADDR_PB
6-bit
ADDR_PA
6-bit
IOSEL
1-bit
IOENA
2-bit
ARX_Load
1-bit
ARX_Flag
1-bit
π-Function Control Unit
1-bit
Start
1-bit
π_Flag
X buffer
Y buffer
Rotator Left (16-bit) X-direction
Rotator Left (16-bit) Y-direction
Xoring Bank (16-bit) X-direction
Xoring Bank (16-bit) Y-direction
FIFO
4 Adders 16-bit
ARX Control Unit
ARX_flag
ARX_Load

Message processor

- ✓ Initialization generator KPIG (it is implemented in just 53 slices at 460 MHz)
- ✓ Data Bus 2x1 multiplexer
- ✓ 16-bit ALU (it has 118 slices as a standalone unit and can run at 408 MHz) (Virtex-7)
- ✓ π-function core
- ✓ 16 bytes cipher-text buffer and tag buffer
- ✓ Message processor control unit MPCU

Message Processor



KPIG

Consist of two buffers :

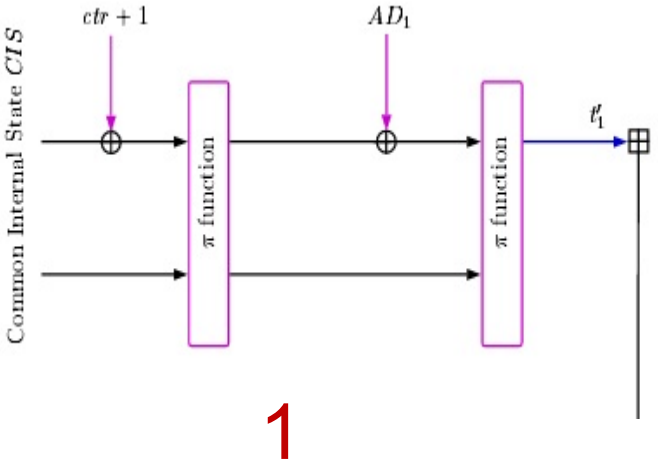
- ✓ Store the Key, and PMN(Public Message Number)
- ✓ Store the Common Internal State (CIS)
- ✓ Each Buffer is presented with 32-bit
- ✓ Can run at 460MHz and implemented in just 53 slices (Vitex-7)
- ✓ Eight states

key_gen	State	Function
000	Initial	Clear the buffers and internal control signals
001	GENKEY_ISRD	Store the key, kpmn_DB data bus ← ((Key ‖ PMN ‖ 10*) ⊕ IS)
010	GENKEY_ISGENRD	Store the key and IS. kpmn_DB data bus ← ((Key ‖ PMN ‖ 10*) ⊕ IS)
011	GENPMN_ISGENRD	Store the key, PMN, and IS. kpmn_DB data bus ← ((Key ‖ PMN ‖ 10*) ⊕ IS)
100	GENPMN_ISRD	Store the PMN. kpmn_DB data bus ← ((Key ‖ PMN ‖ 10*) ⊕ IS)
101	GENKEYPMN_GENISRD	Store the PMN, Key, and IS. kpmn_DB data bus ← ((Key ‖ PMN ‖ 10*) ⊕ IS)
110	GENKEYPMN_ISRD	Store the PMN and Key. kpmn_DB data bus ← ((Key ‖ PMN ‖ 10*) ⊕ IS)
111	GENIS	Store IS

MP ALU

- The rule of the ALU is to XOR the message with the selected data from the output of the π-function or just pass the π-function's output without any changes based on the ALU_mode value.
- The ALU as a standalone unit has been implemented in just 118 slices and can run at 408 MHz.

MPC



1

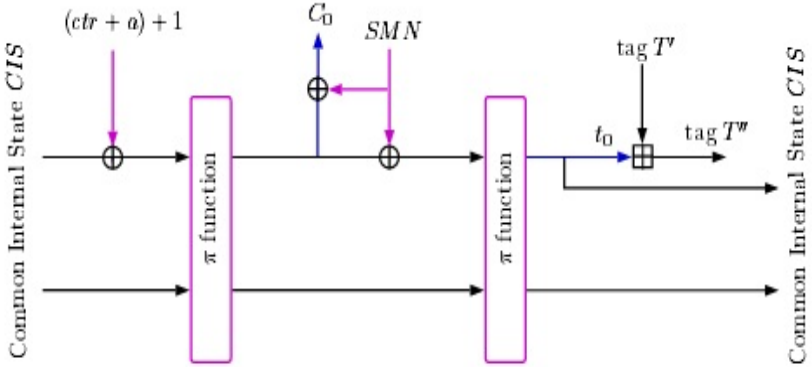


Figure 1.5: Processing the secret message number SMN

2

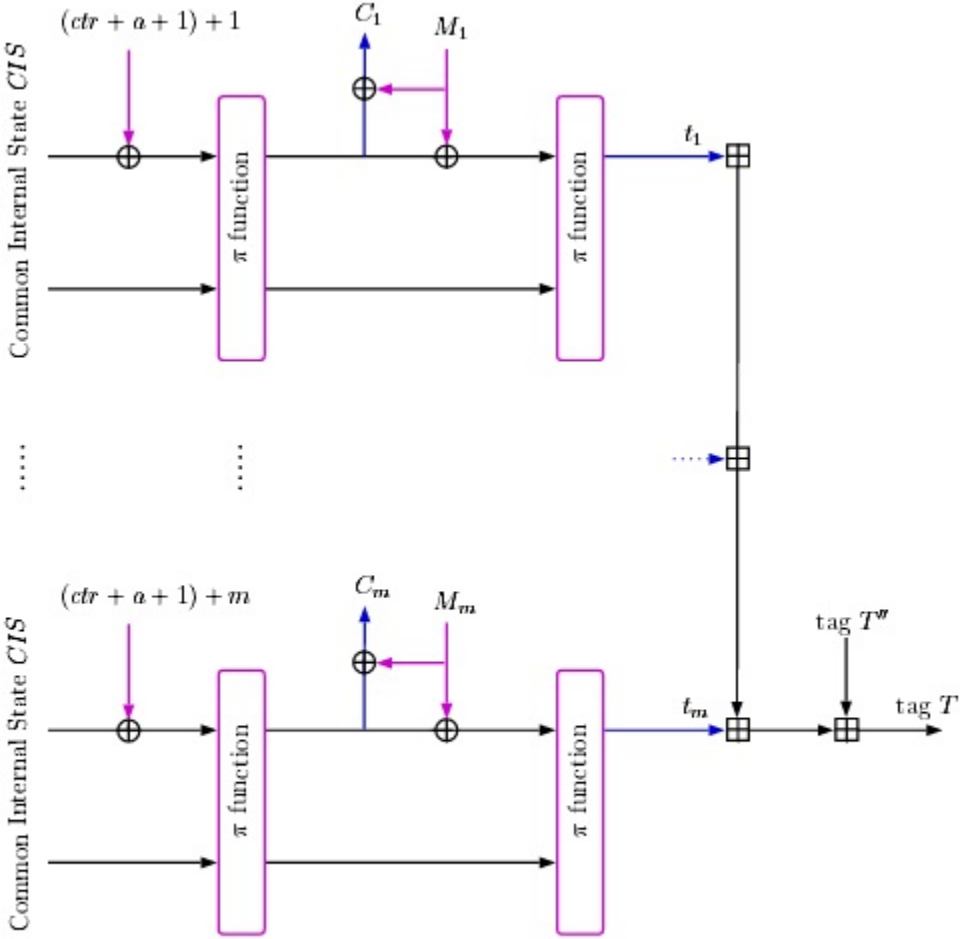


Figure 1.6: Processing the message M with m blocks in parallel

3

Hardware Implementation

- The ARX engine, π-func core and message processor of π16-Cipher096 were synthesized for and verified on Xilinx Virtex-7 architecture, specifically XC7VX485T-2FFG1761

✓ ARX engine performance

- 266 slices running at 347 MHz and achieving 4.34 Gbps

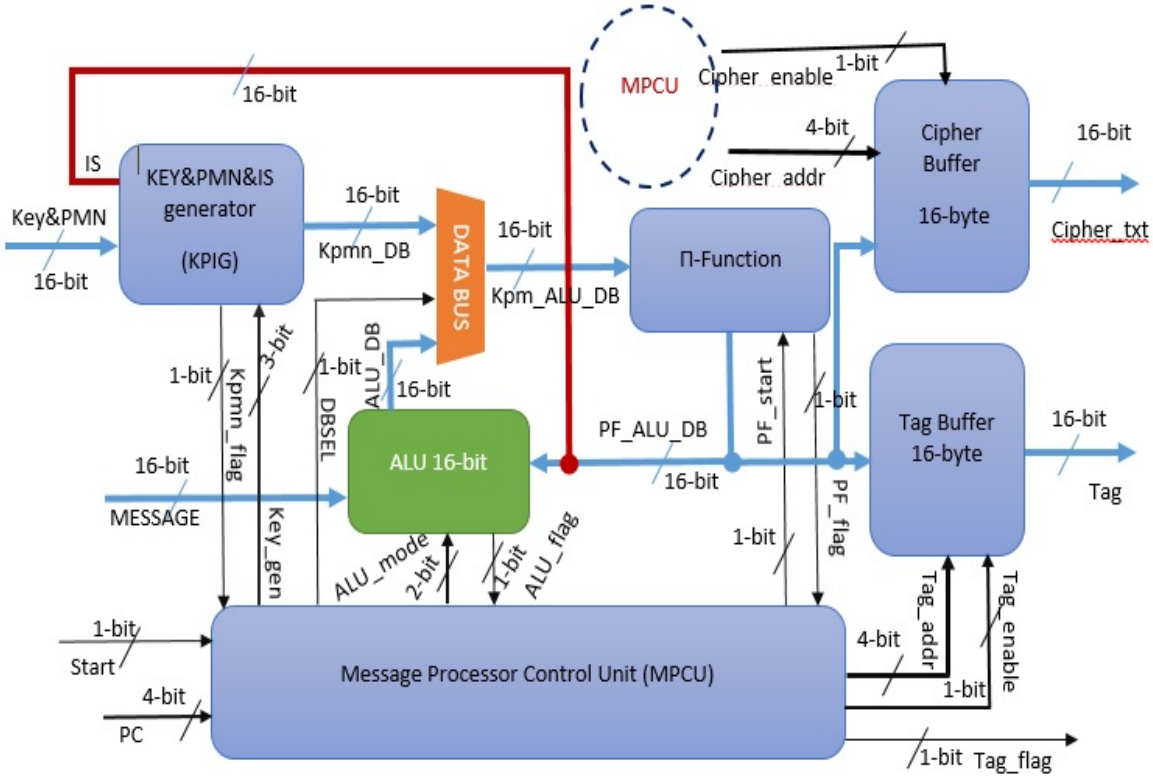
✓ The π-function performance

- 971 slices running at 250 MHz and achieving 95 Mbps

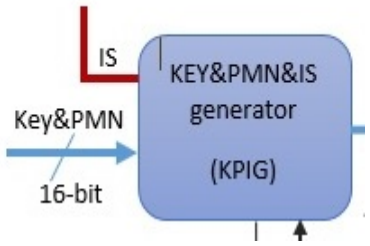
Performance

- The message processor performance
  - 1114 slices running at 250 MHz and achieving 15 Mbps
  - It is 1% of the whole FPGA area
  - Running in parallel (100 message processors) will achieve 1.5 Gbps

Parameterized Key generator



Reconfigurable module



Parameter input (Key&PMN)

Parameterized Key generator

- The KPIS is a parameter input and the value of the a parameter input changes infrequently
- The bit-streams of key generator are expressed as Boolean function of the parameter input KPIG
- (Dynamic Circuit Specialization) For every change in parameter value, a specialized configuration is generated by evaluating the Boolean functions for specific parameter value and the key generator module is reconfigured to replace stale frames with specialized frames

Results of Parameterized Key generator

Implementation	LUTs (TLUTs)	Reconfiguration (ms)
Conventional	9052 (0)	0
Parameterized configuration	8942 (256)	600

Reduction in FPGA LUT resources (27 slices)
at the cost of 600 ms (reconfiguration time)

Conclusion

- ARX Engine:
  - 266 slices (Area), achieving 4.34 Gbps at 347 MHz
  - Four times faster than previous work
- π-func:
  - Security core, most expensive part of the cipher, 971 slices(Area), achieving 95 Mpbs at 250 MHz

Conclusion (cont)

- Message Processor (The whole encryption process)
  - 1114 slices (Area), can be clocked at 250 MHz, achieving 15 Mpbs
  - It uses 1% of the FPGA resources
  - Dynamic Circuit Specialization used and decreased around 27 slices from the total number, which is running at the same clock
- FPGA platform : Virtex-7 (VC707)

Conclusion (cont)

- Using one percent of the FPGA resources for implementing π-Cipher, shows how light the algorithm can be
- Different computation modules in the encryption processor (MP) can be run in different clock
- Adding more flexibility to the processor by increasing the parameterized factor(instead of just using the key as a parameter, could have the processor serve different data-width (16-bit, 32-bit, and 64-bit) versions of the Cipher

Future Work

- Allow the processor to benefit from the different clock domain, which the FPGA can introduce by using PLL
- Add more flexibility to the processor, instead of running the 16-bit version; we will make the processor serves the two other versions of the Cipher, which will increase the rule of the Dynamic Circuit Specialization on optimizing the total area of the processor and decreasing the critical path (Increase the Max.Freq)
- Apply the processor on different IoT applications, which will benefit from the small area with reasonable speed the processor introduced.

Thank you

Q&A