Lightweight version of π-Cipher

Hristina Mihajloska, **Mohamed El Hadedy**, Danilo Gligoroski and **Kevin Skadron**

Department of Computer Science

University of Virginia

July 20, 2015

- Goal: Efficient and flexible crypto processing
  - ARX – simple operations and fast primitives
  - Crypto operations increasingly common
    - Power, performance benefits from specialization
- Outline
  - ARX Programmable Processing Element
  - Custom ARX for Pi-Cipher
  - Comparison
  - Conclusion

CAESAR competition

- Second round 29 candidates
  - mostly AES based
  - other sponge/permutation based
  - just 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, VU
  - Rune Erlend Jensen, NTNU

π-Cipher

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

π-Cipher

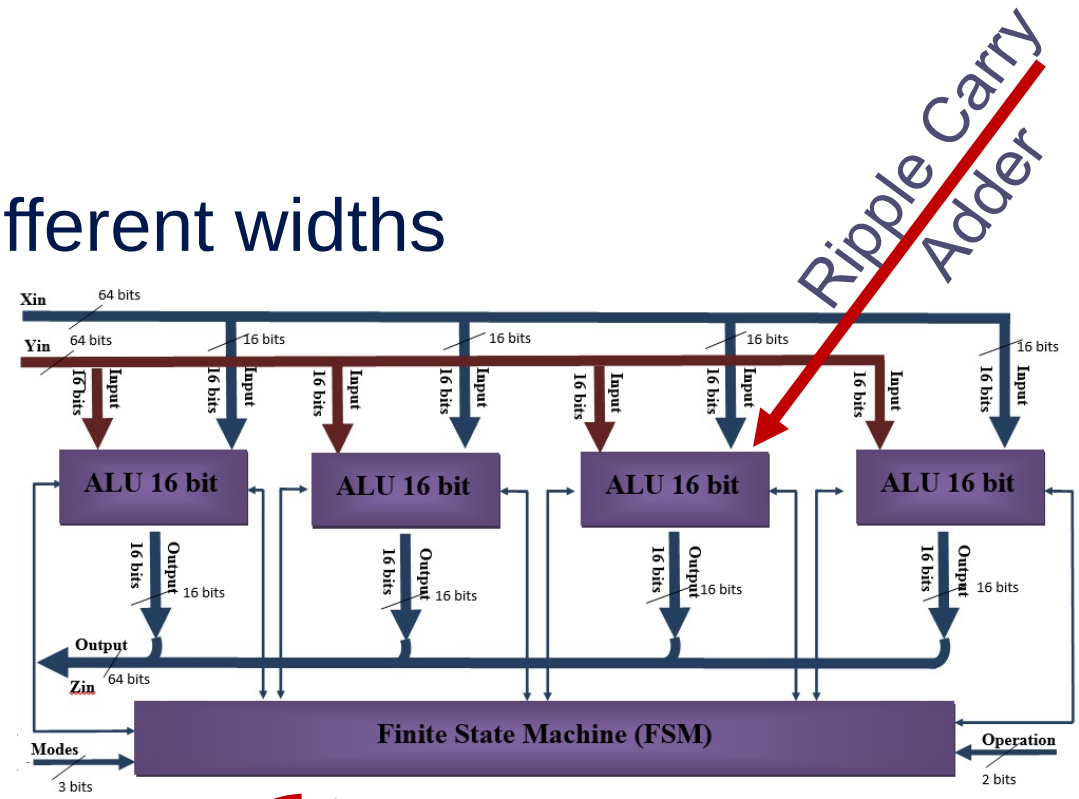
- The main role in the security and design has permutation function (π-function)
- Only uses Add, Rotate and XOR operations
- 4 rounds
  - each round has 8 ARX operations blocks (* op.)
  - every ARX block consists of 52 ARX operations
- Different word sizes
  - π16-Cipher – lightweight version

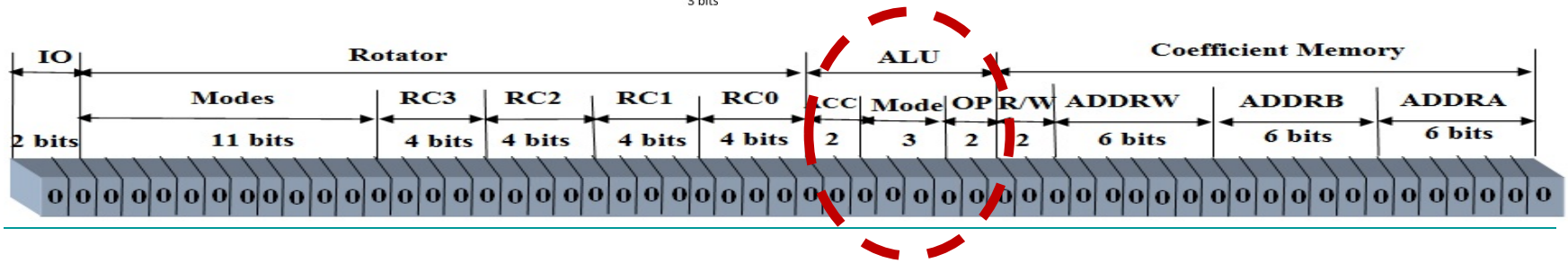
π16-Cipher

- In total:
  - 1664 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

ARX Programmable Processing Element

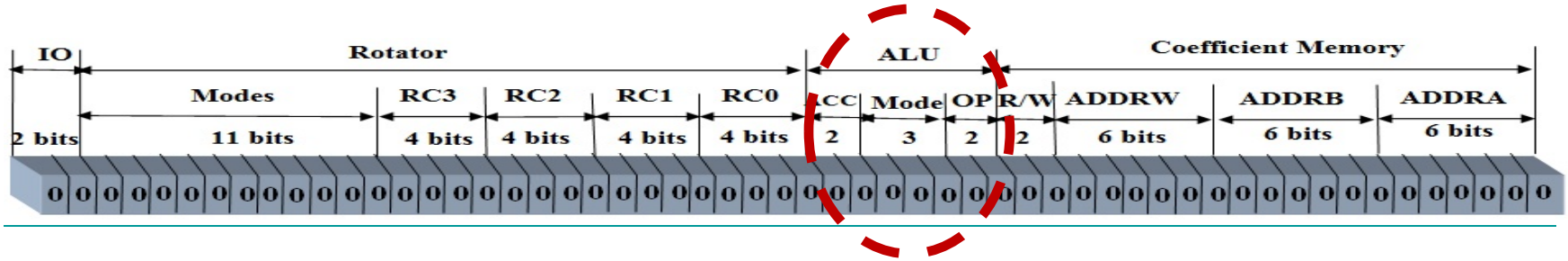
- Flexible & Simple
  - ALU – supports different widths
    - Four 16-bit ALU
    - Two 32-bit ALU
    - One 64-bit ALU





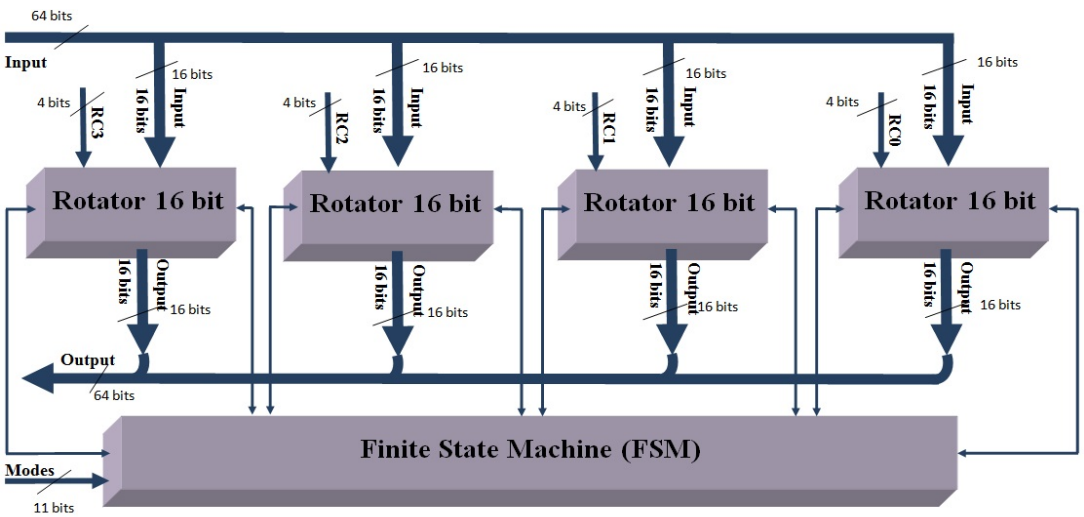
Arithmetic and Logic Unit

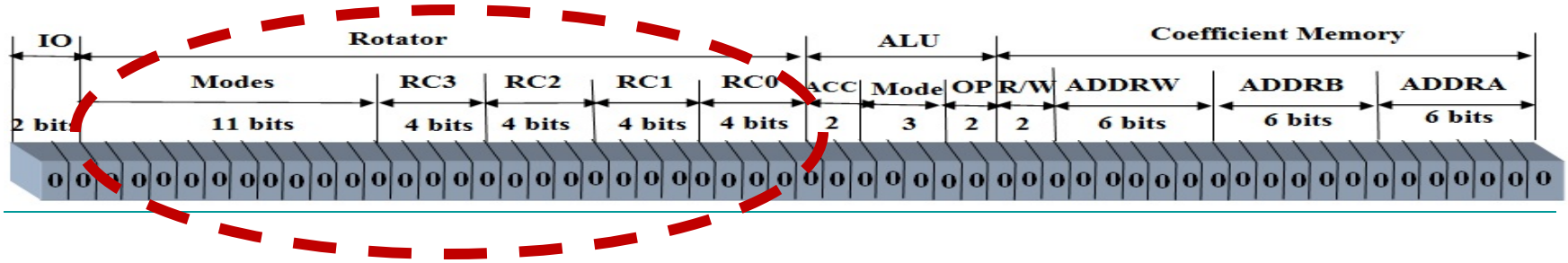
Arithmetic and Logic Unit Truth Table							Operation
Acc		Modes			Operations		
Inst (6)	Inst (5)	Inst (4)	Inst (3)	Inst (2)	Inst (1)	Inst (0)	
Reset the accumulator	Accumulate	1	1	1	0	0	XOR (ALU 64 bits)
		1	1	1	0	1	ADD (ALU 64 bits)
		1	1	1	1	0	INC (ALU 64 bits)
		1	1	1	1	1	SUB (ALU 64 bits)
		0	0	0	0	0	XOR (ALU 32 bits)
		0	0	0	0	1	ADD (ALU 32 bits)
		0	0	0	1	0	INC (ALU 32 bits)
		0	0	0	1	1	SUB (ALU 32 bits)
		1	0	1	0	0	XOR (ALU 16 bits)
		1	0	1	0	1	ADD (ALU 16 bits)
		1	0	1	1	0	INC (ALU 16 bits)
		1	0	1	1	1	SUB (ALU 16 bits)
		0	1	1	0	0	XOR (ALU 48 bits)
		0	1	1	0	1	ADD (ALU 48 bits)
		0	1	1	1	0	INC (ALU 48bits)
		0	1	1	1	1	SUB (ALU 48 bits)



ARX Programmable Processing Element

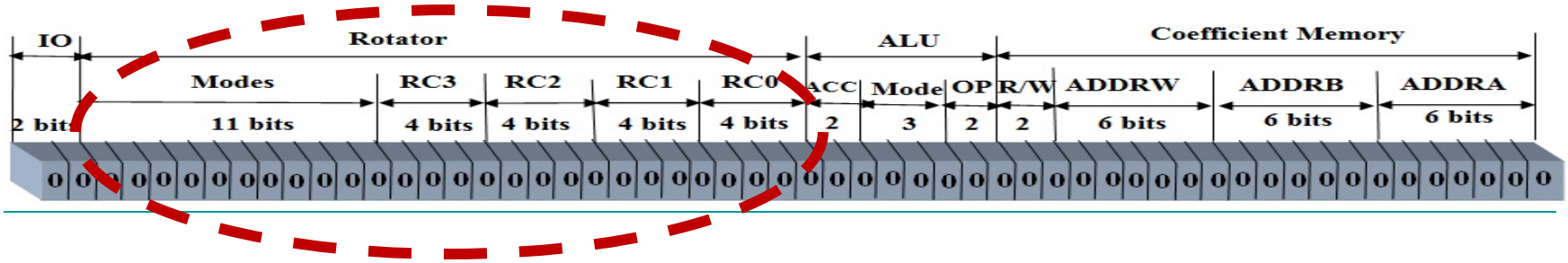
- Flexible & Simple
  - Rotator
    - Four 16-bit ROL
    - Two 32-bit ROL
    - One 64-bit ROL



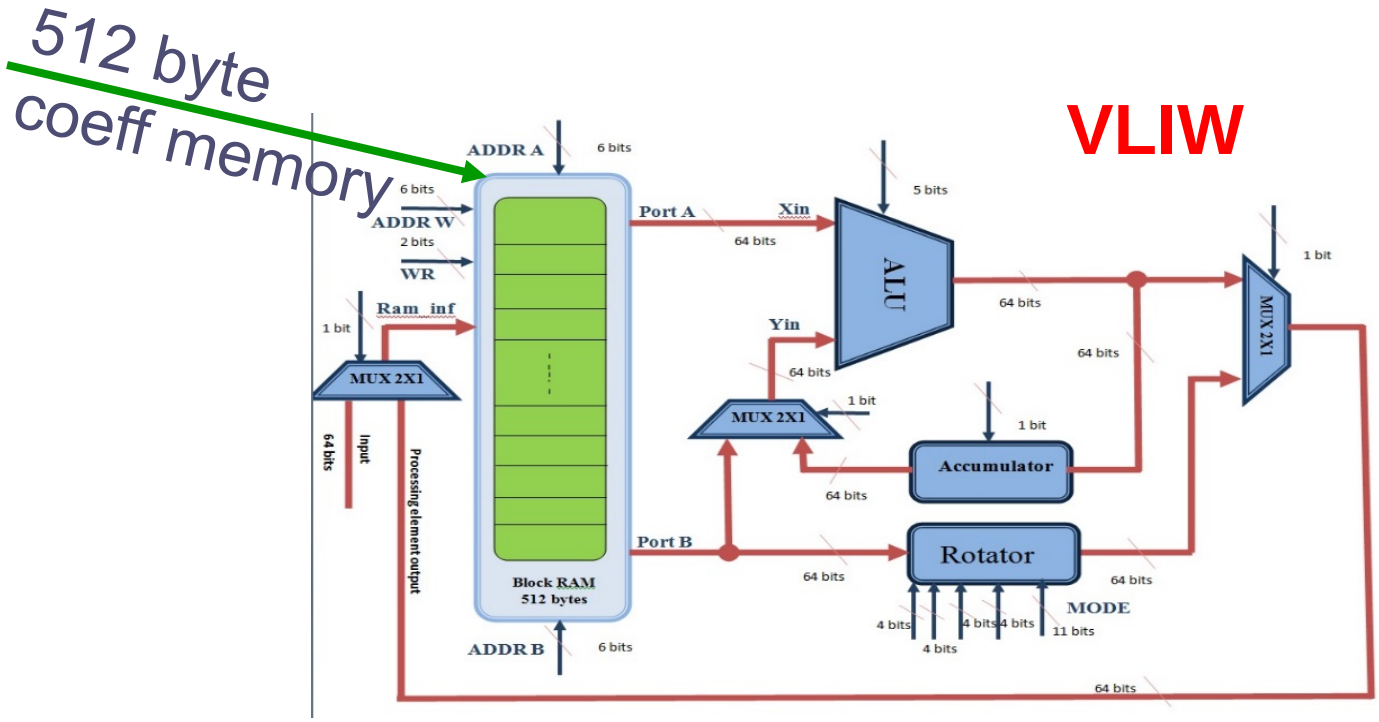


Rotator

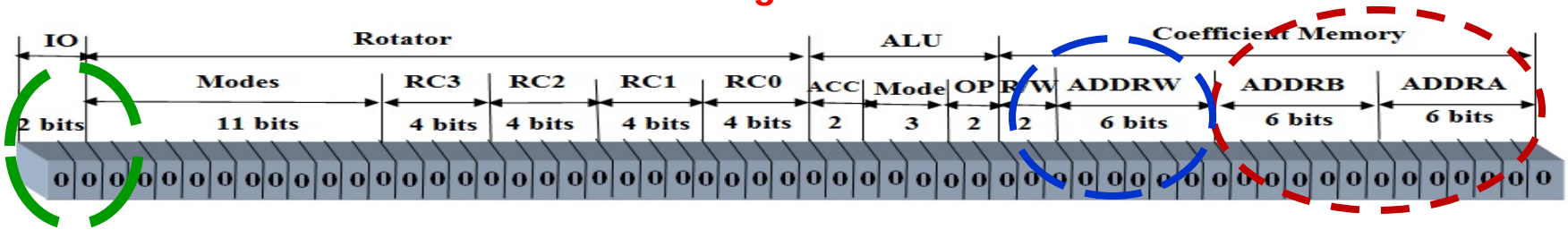
Operation	Mode	RC3	RC2	RC1	RC0	Data width
ROR (1 → 15)	00001010101	1→F	1→F	1→F	1→F	64 Bits Mode
ROR (16 → 31)	01101010101	1→F	1→F	1→F	1→F	
ROR (16 → 31)	01101010101	1→F	1→F	1→F	1→F	
ROR (32 → 47)	10001010101	1→F	1→F	1→F	1→F	
ROR (48 → 63)	11101010101	1→F	1→F	1→F	1→F	
ROR (1 → 15) ROR (1 → 15)	00011011101	1→F	1→F	1→F	1→F	32 Bits Mode
ROR (16 → 31) ROR (16 → 31)	01111011101	1→F	1→F	1→F	1→F	
ROR (1 → 15) ROR (1 → 15) ROR (1 → 15) ROR (1 → 15)	00000000000	1→F	1→F	1→F	1→F	16 Bits Mode



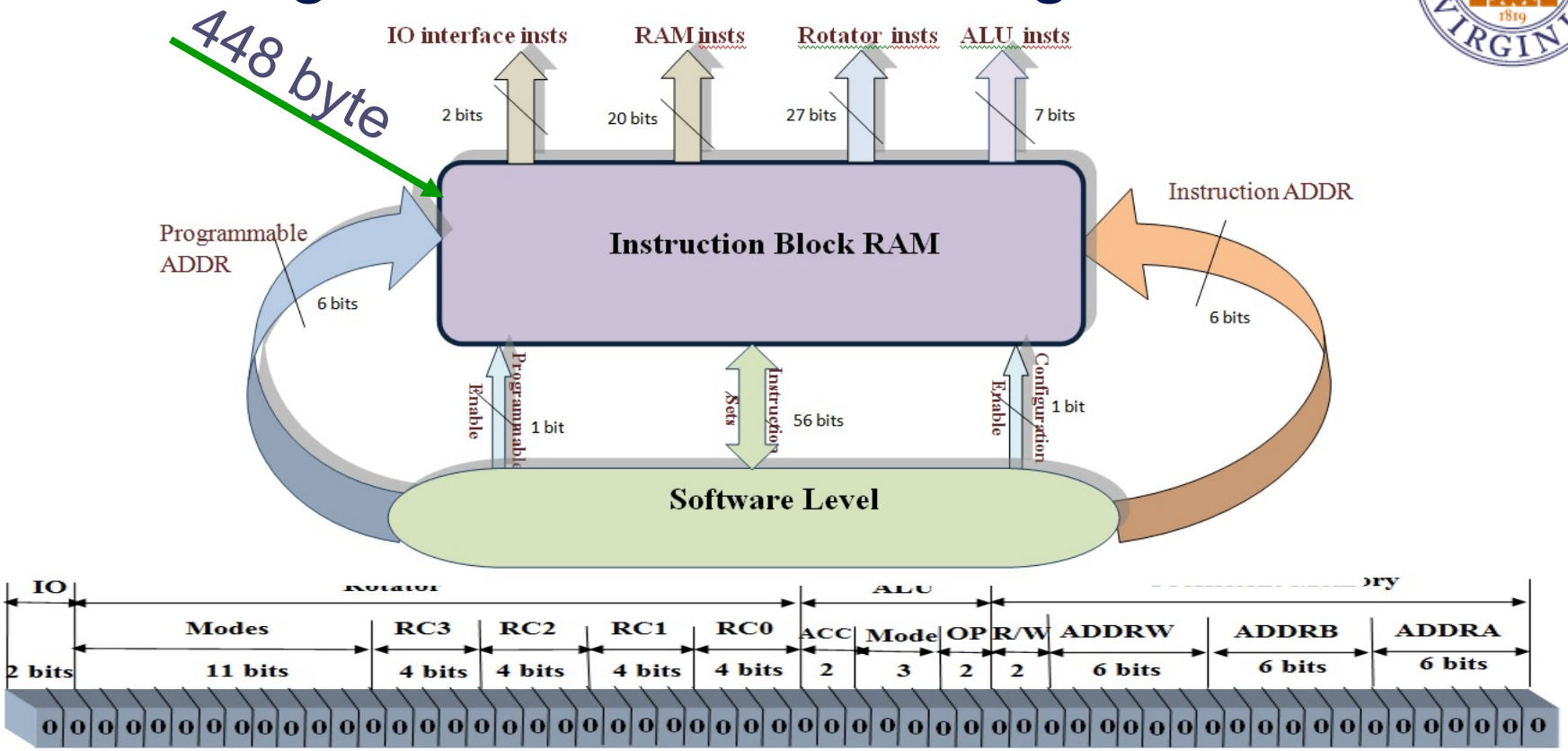
ARX Programmable Processing Element



Processing Element

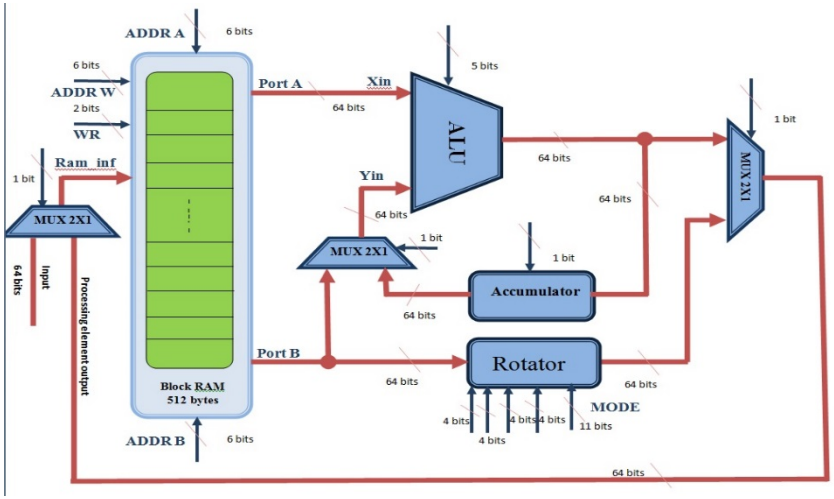


ARX Programmable Processing Element

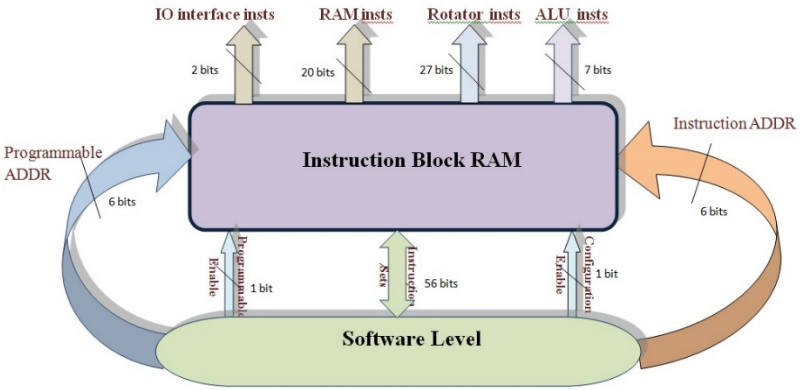


- Dual-ported instruction memory allows:
  - A new program to be loaded while the current program progresses
  - Or an early start on a new program while the rest of the program is still loading Or seamless processing of a program that cannot entirely fit into the on-chip memory.

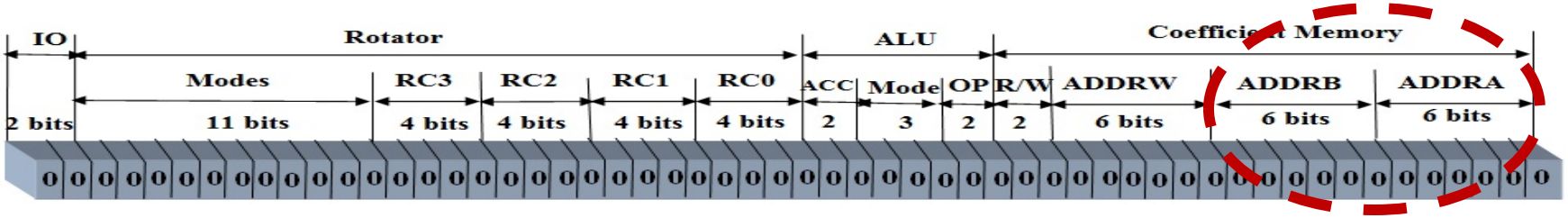
ARX Programmable Processing Element



Processing Element

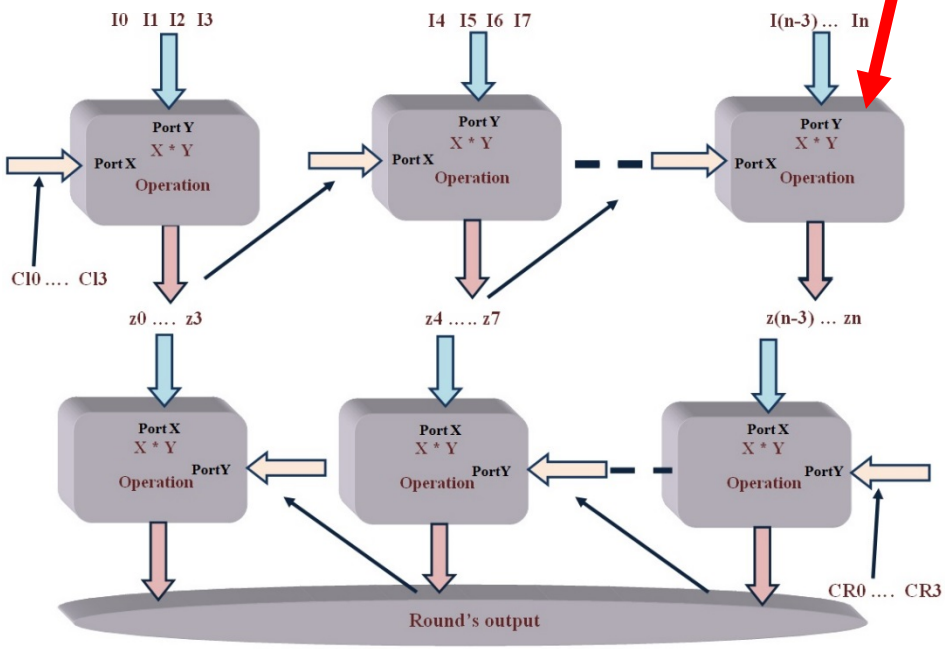


Instruction RAM



Custom ARX Implementation of Pi-Cipher

- Single Width ARX
- Double-Width ARX
- Quad-width ARX



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

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

ν–transformation for Y:

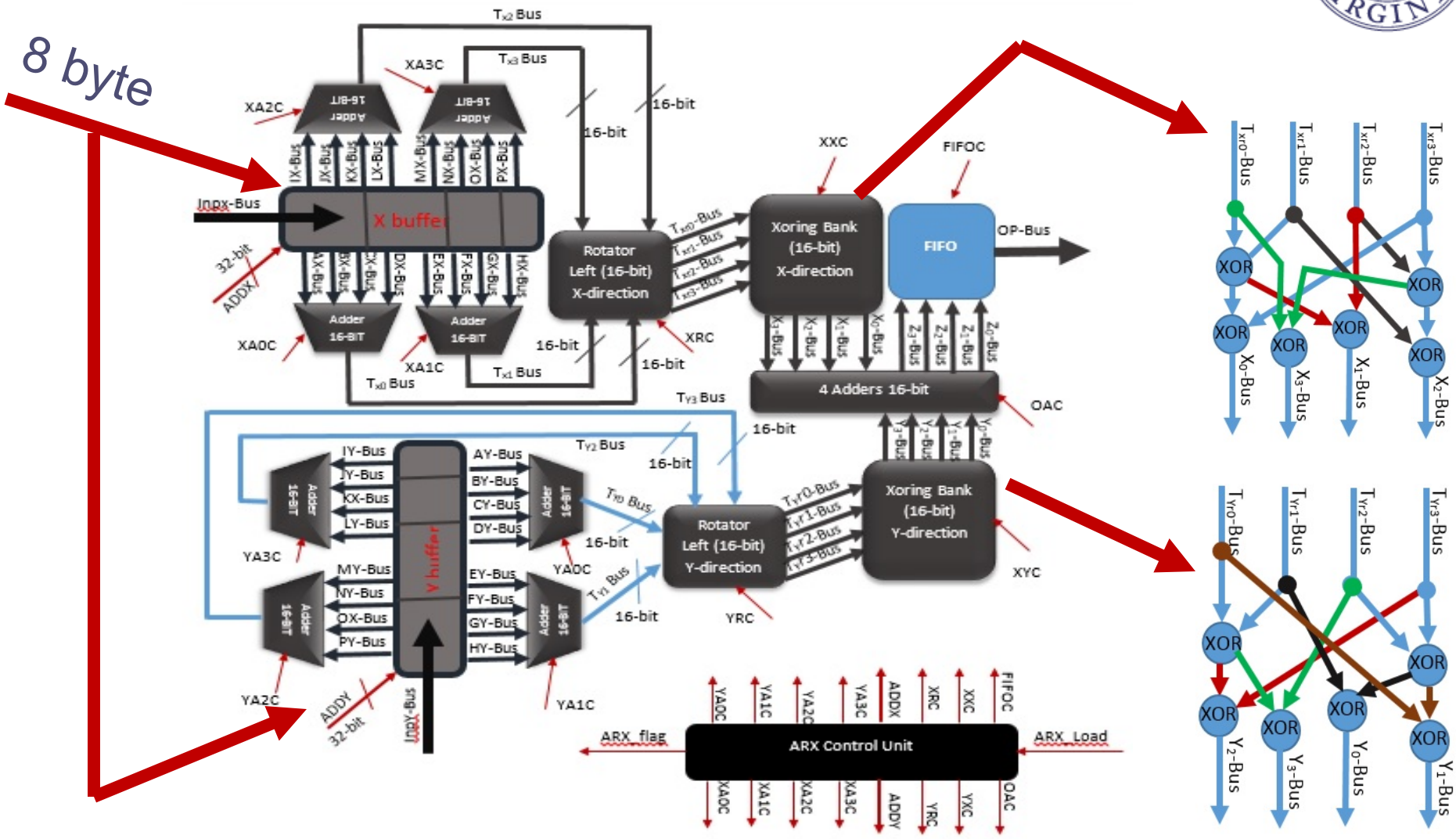
1.
$$\begin{aligned}
T_0 &\leftarrow ROTL^{11}(\texttt{0x8D8B87787472716C} + Y_0 + Y_2 + Y_3);\\
T_1 &\leftarrow ROTL^{23}(\texttt{0x6A696665635C5A59} + Y_1 + Y_2 + Y_3);\\
T_2 &\leftarrow ROTL^{37}(\texttt{0x5655534E4D4B473C} + Y_0 + Y_1 + Y_2);\\
T_3 &\leftarrow ROTL^{59}(\texttt{0x3A393635332E2D2B}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

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

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

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

Custom, Quad ARX Implementation



Implementations & Comparison

	PPE	Single Width	Double Width	Quad Width
Throughput	1.20 Gbps	3.57 Gbps	3.68 Gbps	4.34 Gbps
Area (Slices)	227	132	154	266
Frequency (MHz)	250	371	324	347
Throughput/Area (Mbps/slice)	5.4	27.7	24.5	16.7

ARX Performance (16-Wide Pi-Cipher)

Xilinx Virtex 7 FPGA

Implementations & Comparison

	PPE	Single Width	Double Width	Quad Width
Throughput	1.17 Gbps	3.09 Gbps	3.68 Gbps	4.22Gbps
Area (Slices)	227	445	447	634
Frequency (MHz)	250	254	243	245
Throughput/Area (Mbps/slice)	5.3	7.1	8.4	6.8

ARX Performance (64-Wide Pi-Cipher)

Xilinx Virtex 7 FPGA

Performance

- PPE
  - **Pros**: It is a **programmable** element which can be used to implement different algorithms based on the ARX paradigm
  - **Pros**: It can support different word sizes
  - **Pros**: Duplicating the PPE can achieve 75% of the 64-bit custom core's throughput, but with greater flexibility

Performance

- PPE
  - **Cons**: The custom hardware implementations are much more efficient than the PPE (Area & Performance)
  - **Cons**: The PPE design we have so far is handicapped because it uses a native 64-bit ALU, which we suspect lowers the achievable frequency by increasing the critical path

Future Work

- Further improve efficiency of the PPE
- Implement more algorithms with the PPE & benchmark against custom hardware

Thank you

Q&A