



ECIT | The Institute of Electronics,
Communications and
Information Technology



High Speed Whirlpool Hardware Architecture

Dr. Máire McLoone
*Royal Academy of Engineering
Research Fellow*





- **Necessity & Key Challenges behind Hardware Cryptography**
- **Whirlpool Algorithm**
- **Whirlpool Hardware Architecture**
- **General design techniques to achieve high speed hardware architectures**
- **Conclusion**



- **Encryption needs to be performed on data in real-time**
 - 100 Mbps networks, 1G Ethernet, 10G Ethernet
- **This holds the key to successful growth of applications such as WLANs, satellite communications, e-commerce ...**
- **Software architectures are too slow => Hardware solutions required**
- **Cryptographic SoC Architectures can be used to provide the security requirements of many applications**
- **FPGAs are well suited for crypto algorithms:**
 - allow algorithm agility
 - support alterable architecture parameters, scalable security (DES/ 3DES)
- **Clever mapping of complex math operations onto special purpose silicon architectures**



- **Selected for NESSIE**
- **Included in the ISO/IEC 10118-3 standard**
- **Designed by Barreto and Rijmen**
- **Operates on messages $< 2^{256}$ bits in length**
- **Produces a 512-bit hash output**
- **Utilises an underlying 512-bit block cipher, *W***

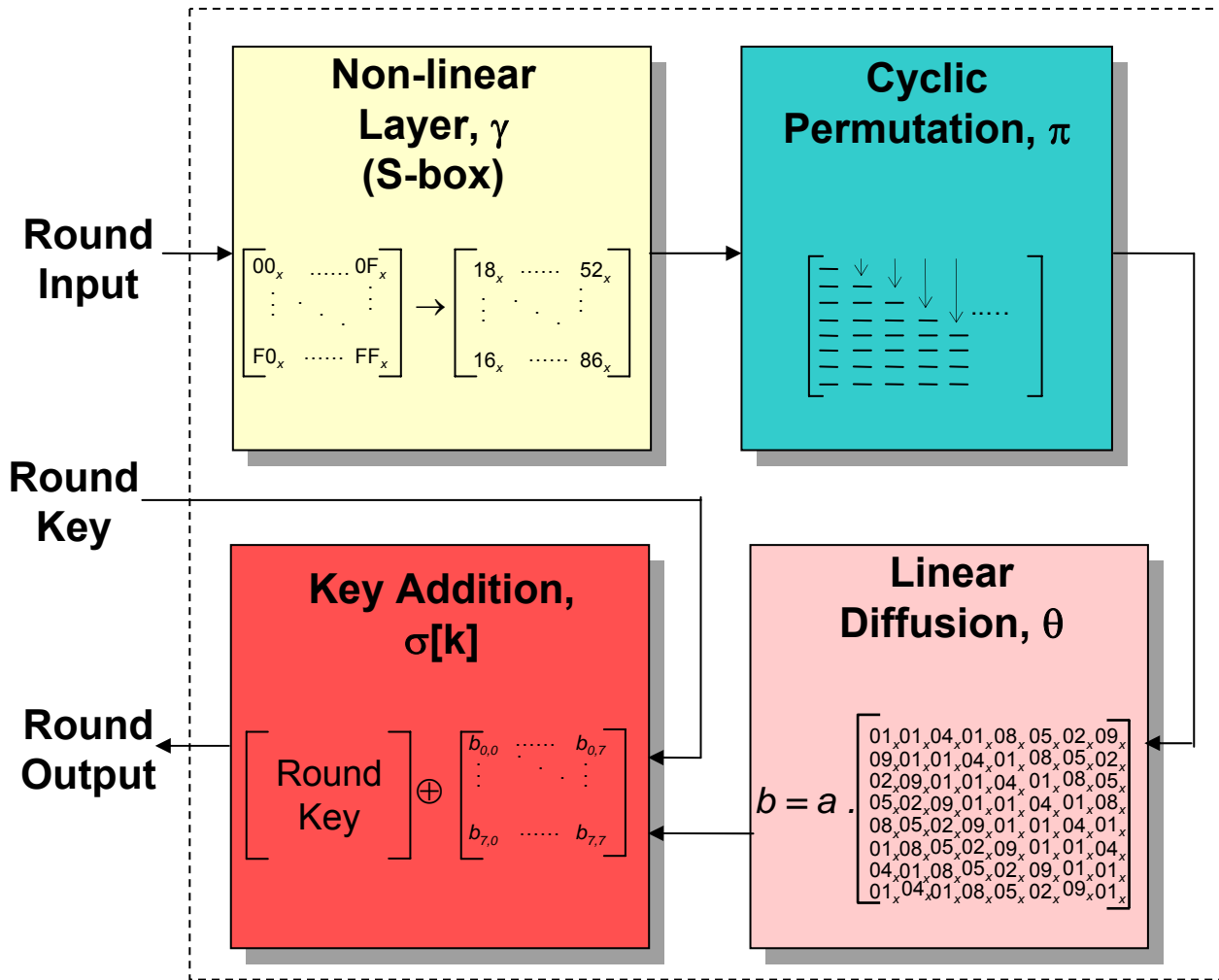


- **Message Padding**

- Append '1' to end of message, M
- Then as few '0' bits to achieve a length which is an odd multiple of 256
- Then 256-bit right-justified binary representation of message length
- Split message into n x 512-bit blocks, $m_1, m_2, m_3, \dots, m_n$

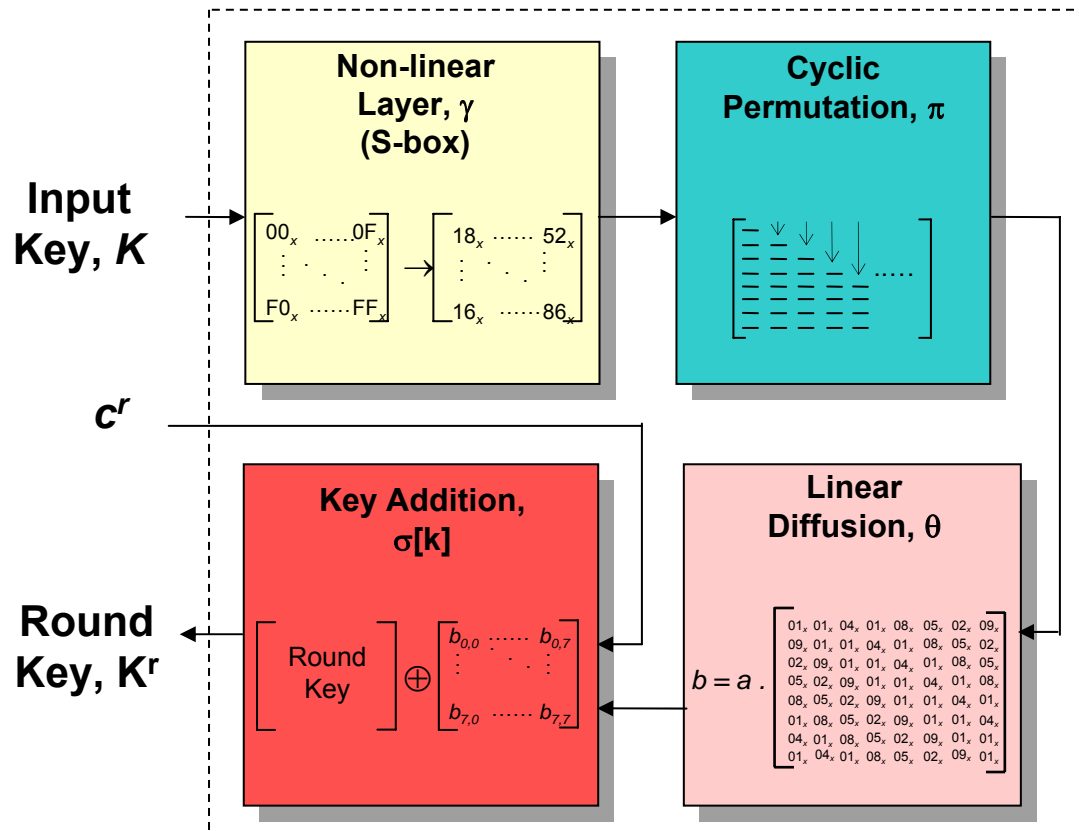
- ***W Round, $\rho[k]$***

- Non-linear layer, γ (S-box)
- Cyclic Permutation, π
- Linear Diffusion, θ
- Key Addition, $\sigma[k]$





- Expands the input key, K , into 10 x 512-bit round keys
- Expansion process uses 10 iterations of the W Round





- Different constant matrix, c^r used in every iteration
- Generated using s-box, where

$$\text{Row } 0 = \text{Sbox} [8(r-1) + j] \quad 0 \leq j \leq 7$$

$$\text{Row } 1 \text{ to } 7 = [00, 00, 00, 00, 00, 00, 00, 00]$$

- Example:

$$c^1 = \begin{bmatrix} 18 & 23 & C6 & E8 & 87 & B8 & 01 & 4F \\ 00 & 00 & 00 & \dots & \dots & \dots & & 00 \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ 00 & 00 & 00 & \dots & \dots & \dots & & 00 \end{bmatrix}$$



$$H_i = W [H_{i-1}](m_i) \oplus H_{i-1} \oplus m_i \quad 1 \leq i \leq n$$

- 512-bit message, $m_1, m_2 \dots m_n$ as 8 byte x 8 byte matrix
- H_{i-1} is the input key, K
- Initial Key, $H_0 =$ string of 512 '0' bits
- Output hash value = H_n



- **Objective : To ascertain how fast Whirlpool hash function can operate on modern hardware device**
- **Target Device : Xilinx Virtex-4 FPGA LX device**
- **W Round used in both message processing & key schedule**

Minimise Area : Re-use Hardware Round arch for both

Max Speed : Use two Hardware Round archs in parallel

- **Can also unroll Hardware Rounds to improve speed**

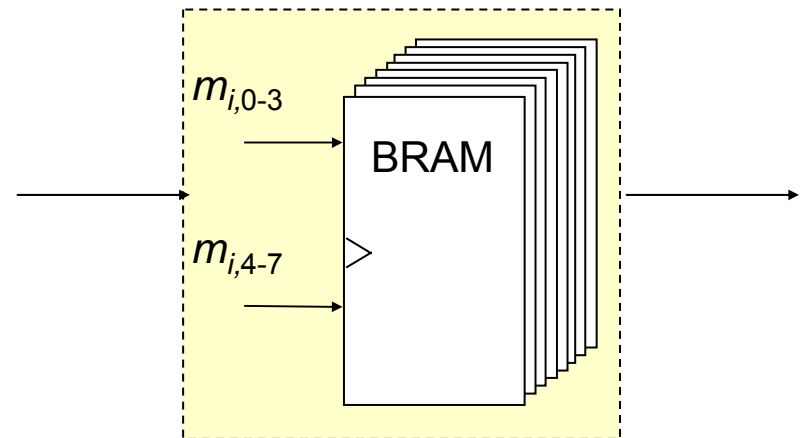


- 1. LUT-based design of s-box**
- 2. Use Small LUTs**
- 3. Use Boolean Expressions**



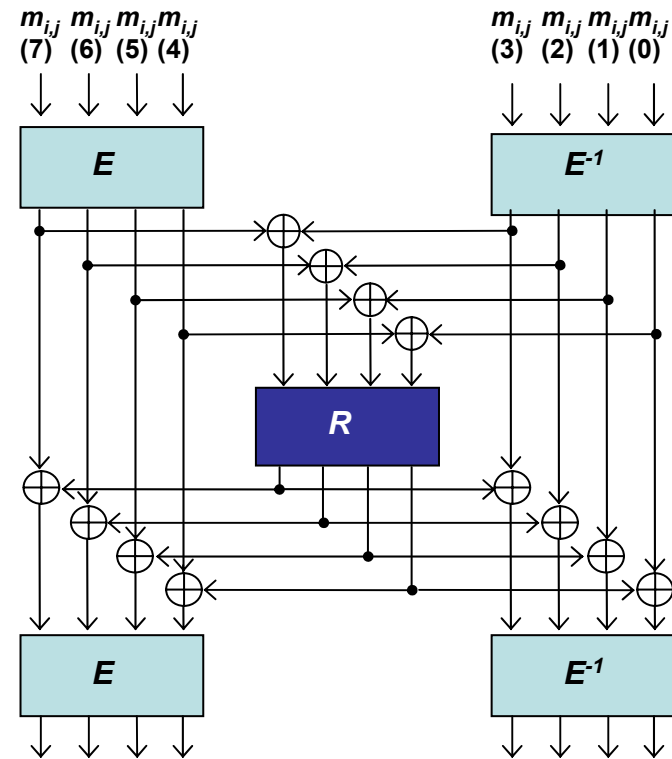
1. LUT-based design of s-box

- 256 pre-specified values
- Need 64 LUTs
- Mapped onto BRAM
- Two LUTs per BRAM





2. Use Small LUTs



- Small LUTs contain 16 byte values
- Mapped to async distributed ROM found within CLB slice



3. Use Boolean Expressions

Eg: $z = R[u]$

$$t_0 \leftarrow \overline{u_0}$$

$$t_1 \leftarrow u_2 \wedge u_3$$

$$t_2 \leftarrow u_0 \oplus t_1$$

$$t_2 \leftarrow t_2 \vee u_1$$

$$t_3 \leftarrow u_3 \vee t_0$$

$$z_2 \leftarrow t_2 \oplus t_3$$

$$t_2 \leftarrow \overline{u_2}$$

$$t_2 \leftarrow t_2 \oplus t_3$$

$$t_3 \leftarrow u_1 \vee t_2$$

$$z_3 \leftarrow t_1 \oplus t_3$$

$$t_3 \leftarrow t_3 \oplus t_0$$

$$t_0 \leftarrow u_0 \vee z_3$$

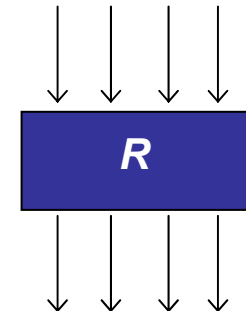
$$t_1 \leftarrow \overline{u_1}$$

$$t_1 \leftarrow t_1 \oplus u_3$$

$$z_0 \leftarrow t_0 \oplus t_1$$

$$t_3 \leftarrow t_3 \vee z_0$$

$$z_1 \leftarrow t_2 \oplus t_3$$





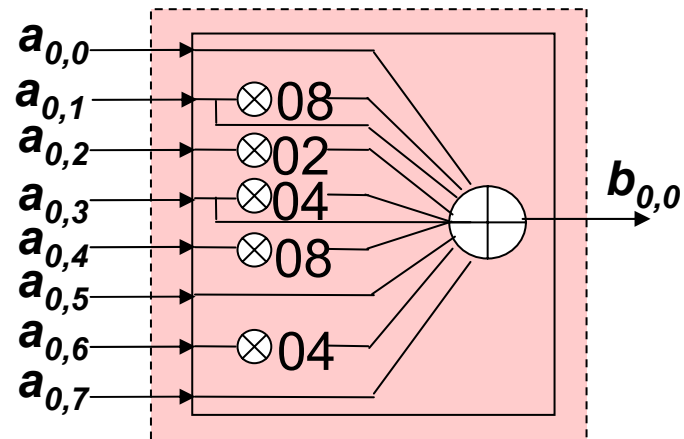
- Involves 64 multiplications by pre-specified constants
- $GF(2^8)$ mult = shifting and possible XOR with value $11d_x$

Eg: $b_{0,0} =$

$$a_{0,0} \oplus [a_{0,1} \otimes 09_x] \oplus [a_{0,2} \otimes 02_x] \oplus [a_{0,3} \otimes 05_x] \oplus [a_{0,4} \otimes 08_x] \\ \oplus a_{0,5} \oplus [a_{0,6} \otimes 04_x] \oplus a_{0,7}$$

$\Rightarrow b_{0,0} =$

$$a_{0,0} \oplus a_{0,1} \oplus [a_{0,1} \otimes 08_x] \oplus [a_{0,2} \otimes 02_x] \\ \oplus a_{0,3} \oplus [a_{0,3} \otimes 04_x] \oplus [a_{0,4} \otimes 08_x] \\ \oplus a_{0,5} \oplus [a_{0,6} \otimes 04_x] \oplus a_{0,7}$$

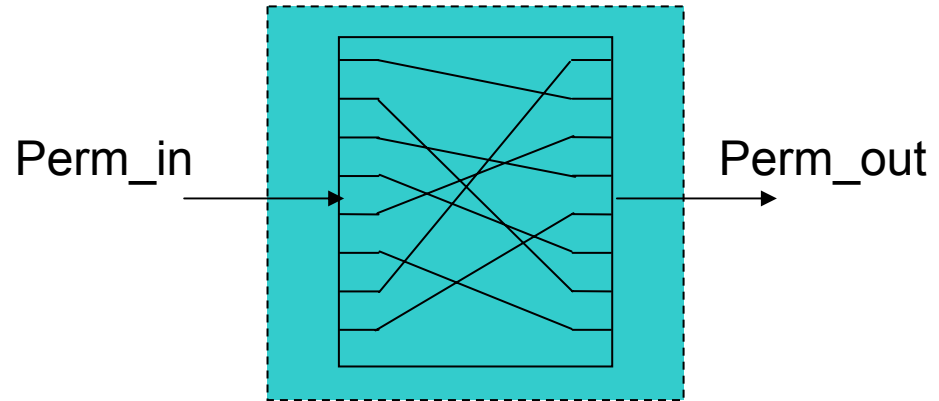




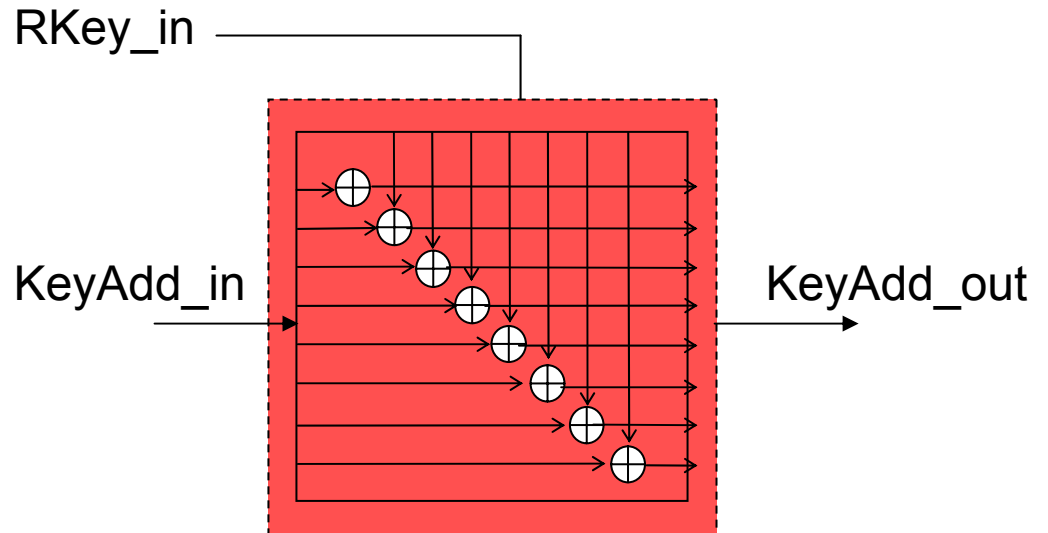
- **GF(2⁸) mult can be performed using composite fields**
- **Results in lower area circuit**
- **Polynomial for GF(2⁴) : $Q(y) = y^4 + y + 1$**
- **Polynomial for GF(2⁴)² : $P(x) = x^2 + x + \lambda$, $\lambda \in \text{GF}(2^4)$**
- **Map a in GF(2⁸) to $(a_h x + a_l)$ in GF(2⁴)² using transf, T**
 $(a_h x + a_l) (b_h x + b_l) \bmod x^2 + x + \lambda$
 $= [a_h b_h + a_h b_l + a_l b_h]x + [a_l b_l + a_h b_h \lambda]$
 $= (c_h x + c_l)$
- **Map back to c in GF(2⁸) using inverse transformation, T⁻¹**



Cyclic Permutation
≡ **Hardwiring**

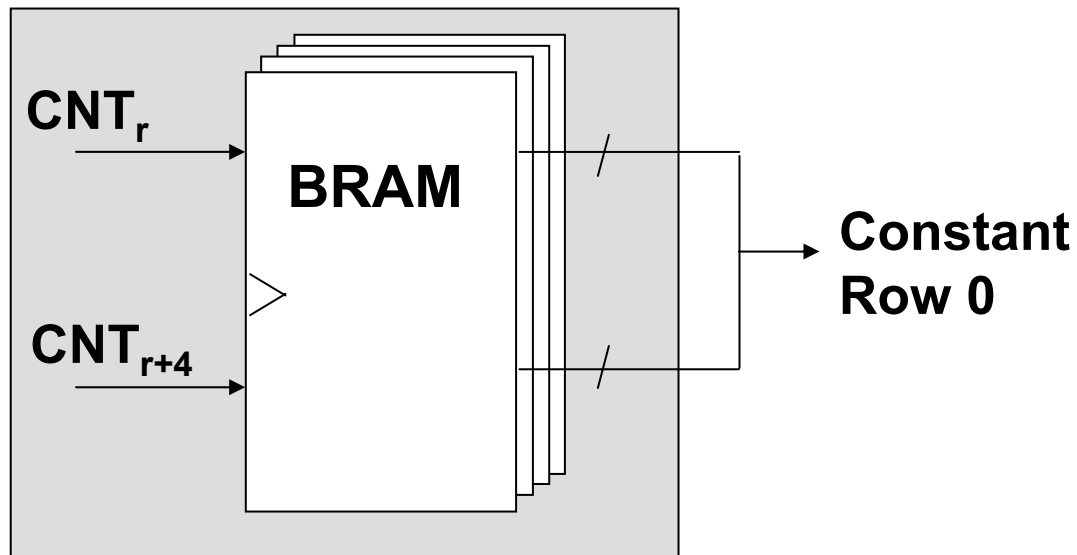


Key Addition
≡ **64 byte-wise XORs**



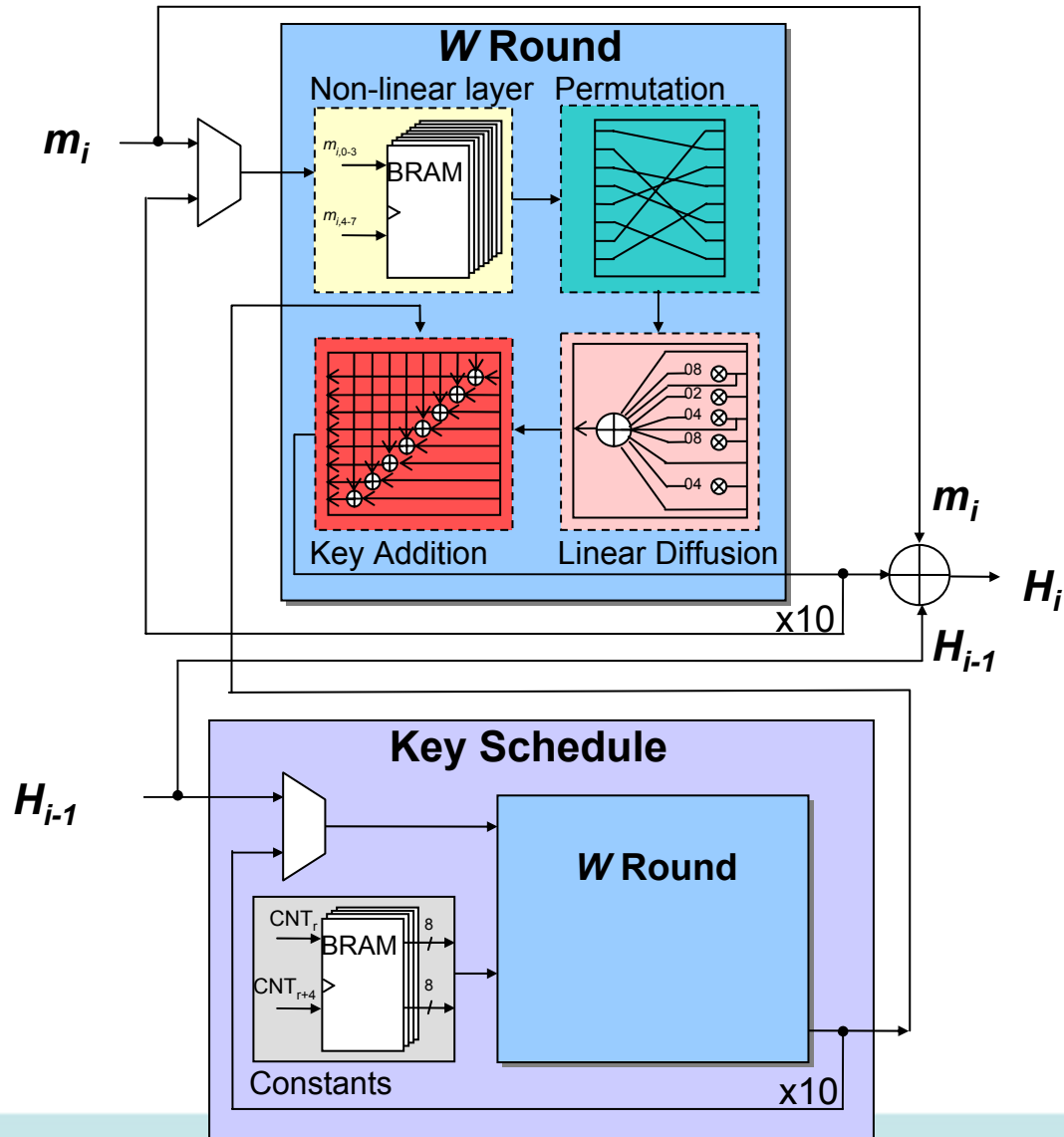


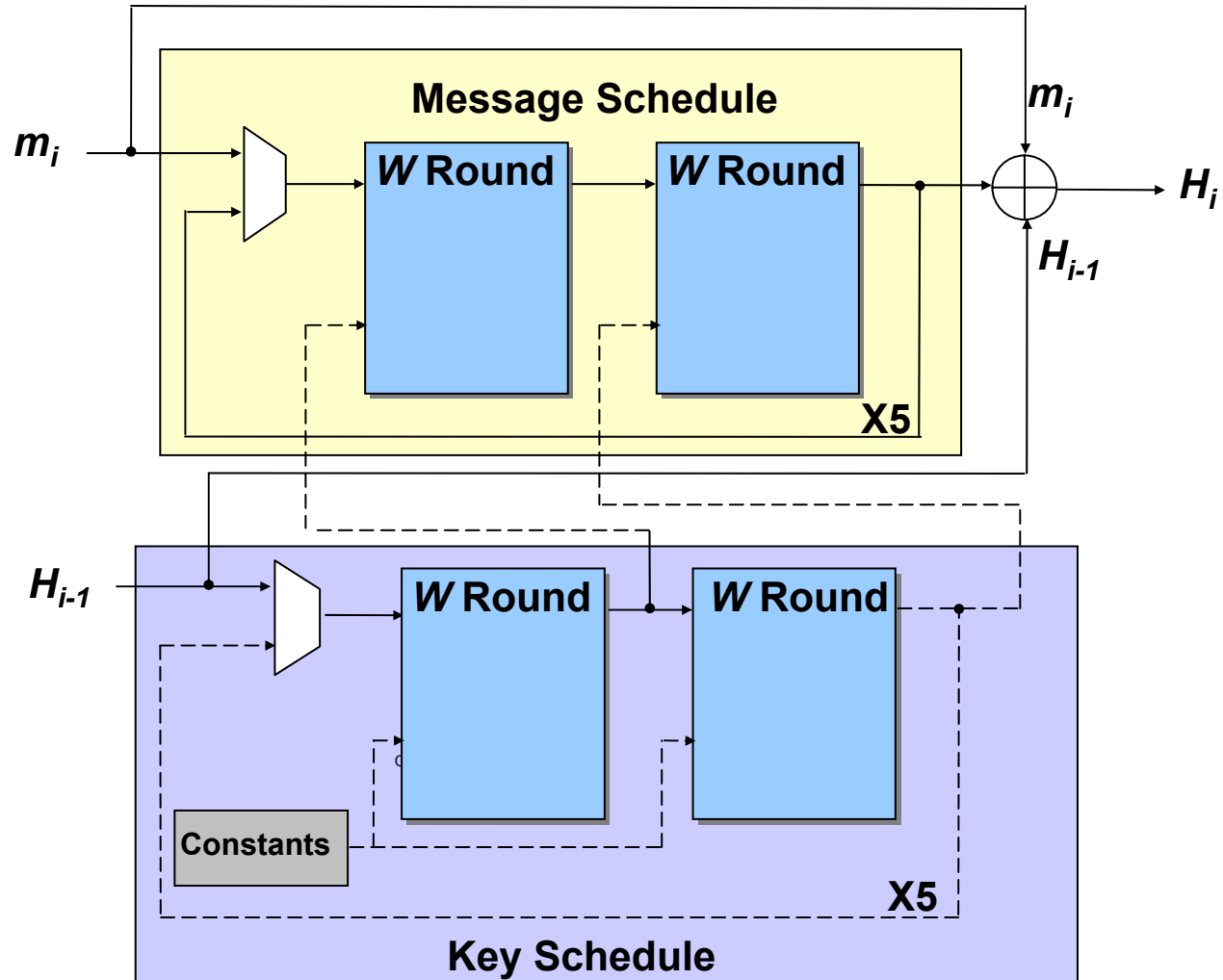
- LUT required to store 10 constants, c^r used in Rounds
- Mapped to either distributed ROM or BRAM
- Full-LUT Method: Implemented using 4 BRAM





Overall W Round Architecture







Full-LUT Method **v** Small LUT Method **v** Boolean Method

- **Kitsos *et al* showed that using small LUTs achieved higher performance over Boolean method**
- **Small LUT Method**
 - **7993 CLB slices, 4280 Mbps**
- **Full LUT Method**
 - **4908 CLB slices, 68 BRAM, 4710 Mbps**
- **Therefore, full-LUT method used in Iterative Design**



Iterative Design **v** Unrolled Design

- Iterative design uses Full-LUT method which involves synchronous BRAM
- Unrolled design uses Small-LUT method which are implemented as asynchronous distributed ROM
- Iterative design:
 - 4908 CLB slices, 68 BRAM, 91 MHz, 4710 Mbps
- Unrolled x2 design:
 - 13210 slices, 47.8 MHz, 4896 Mbps



	Hash Function	Device	Area Slices/BRAM	Speed Mbps
McLoone, Mclvor	Whirlpool (Unroll x 2)	Virtex-4 X4VLX100	13210 / 0	4896
McLoone, Mclvor	Whirlpool (Iterative)	Virtex-4 X4VLX100	4908 / 68	4710
Barreto, Rijmen	Whirlpool	550 MHz Pentium III	-	33
Lien et al	SHA-512 (Unroll x 5)	Virtex	- / 0	1034
McLoone, Mclvor	SHA-512 (Iterative)	Virtex-4 X4VLX100	2734 / 2	854
Grembowski et al	SHA-512 (optimised)	Virtex XCV1000	3441 / 2	676
Crowe et al	SHA-512 (Unroll x 4)	Virtex-E XCV2000E	3506 / 0	533
Helion Tech.	MD5	Virtex-II	613 / 1	744
Järvinen et al	MD5 (Full unroll)	Virtex-II XC2V4000	7997 / 0	725
Deepakumara et al	MD5 (Full unroll)	Virtex XCV1000	4763 / 0	354
Helion Tech.	SHA-256	Virtex-II	849 / 1	685
Sklavos et al	SHA-256 (Iterative)	Virtex XCV200	1060 / 0	326

- **Design Techniques**

- Unrolling / Pipelining / Sub-pipelining
- Shift-register design

- **Computer Arithmetic Techniques**

- Use of Carry-Save adders for multiple additions
- Use of Montgomery multiplication for high-speed modular multiplication

- **Data manipulation**

- Use of Composite Fields

- **Optimisation to the hardware device**

- Use of on-chip memory – BRAMs
- Use of embedded multipliers



- **Hardware Whirlpool design can run at up to 5 Gbps**
- **Unrolled architecture achieves highest throughput where optimal number of unrolled rounds is two**
- **150 times faster than software implementation**
- **Whirlpool up to 4.5 times faster than previous hash functions**
- **Can be further improved by implementing a number of independent Whirlpool architectures in parallel on device**