Evaluation Of Compact FPGA Implementations For All SHA-3 Finalists

Bernhard Jungk

University of Applied Sciences Wiesbaden, Germany
Easycore GmbH, Erlangen, Germany

3rd SHA-3 Conference

Design Goals

- Comparison of all SHA-3 finalists for FPGAs (Xilinx)
- Overall goal: Low area usage, yet high throughput-area ratio
- No usage of BlockRAM, DSP units, ...
- Inclusion of the padding function
- Identical hardware interface

Design Goals

- Comparison of all SHA-3 finalists for FPGAs (Xilinx)
- Overall goal: Low area usage, yet high throughput-area ratio
- No usage of BlockRAM, DSP units, ...
- Inclusion of the padding function
- Identical hardware interface

Design Goals

- Comparison of all SHA-3 finalists for FPGAs (Xilinx)
- Overall goal: Low area usage, yet high throughput-area ratio
- No usage of BlockRAM, DSP units, ...
- Inclusion of the padding function
- Identical hardware interface

Design Goals

- Comparison of all SHA-3 finalists for FPGAs (Xilinx)
- Overall goal: Low area usage, yet high throughput-area ratio
- No usage of BlockRAM, DSP units, ...
- Inclusion of the padding function
- Identical hardware interface

Design Goals

- Comparison of all SHA-3 finalists for FPGAs (Xilinx)
- Overall goal: Low area usage, yet high throughput-area ratio
- No usage of BlockRAM, DSP units, ...
- Inclusion of the padding function
- Identical hardware interface

Design Strategy

- Main idea #1: Folding of algorithmb
- Benefit #1: Usage of RAM instead of registers
- Benefit #2: Reusing redundant parts of the compression functions
- Drawbacks: Reduced throughput, additional multiplexers, larger control logic

Design Strategy

- Main idea #1: Folding of algorithmb
- Benefit #1: Usage of RAM instead of registers
- Benefit #2: Reusing redundant parts of the compression functions
- Drawbacks: Reduced throughput, additional multiplexers, larger control logic

Design Strategy

- Main idea #1: Folding of algorithmb
- Benefit #1: Usage of RAM instead of registers
- Benefit #2: Reusing redundant parts of the compression functions
- Drawbacks: Reduced throughput, additional multiplexers, larger control logic

Design Strategy

- Main idea #1: Folding of algorithmb
- Benefit #1: Usage of RAM instead of registers
- Benefit #2: Reusing redundant parts of the compression functions
- Drawbacks: Reduced throughput, additional multiplexers, larger control logic

Design Strategy

- Main idea #2: Pipelining of folded architectures
- Benefit #1: Improved clock frequency
- Benefit #2: No (or few) additional clock cycles, if the pipeline is designed carefully.
- Drawback: Often more area is required
- But: For FPGAs a register can often be placed in the same slice as a LUT
- Control logic sometimes more difficult, sometimes easier to implement

Design Strategy

- Main idea #2: Pipelining of folded architectures
- Benefit #1: Improved clock frequency
- Benefit #2: No (or few) additional clock cycles, if the pipeline is designed carefully.
- Drawback: Often more area is required
- But: For FPGAs a register can often be placed in the same slice as a LUT
- Control logic sometimes more difficult, sometimes easier to implement

Design Strategy

- Main idea #2: Pipelining of folded architectures
- Benefit #1: Improved clock frequency
- Benefit #2: No (or few) additional clock cycles, if the pipeline is designed carefully.
- Drawback: Often more area is required
- But: For FPGAs a register can often be placed in the same slice as a LUT
- Control logic sometimes more difficult, sometimes easier to implement

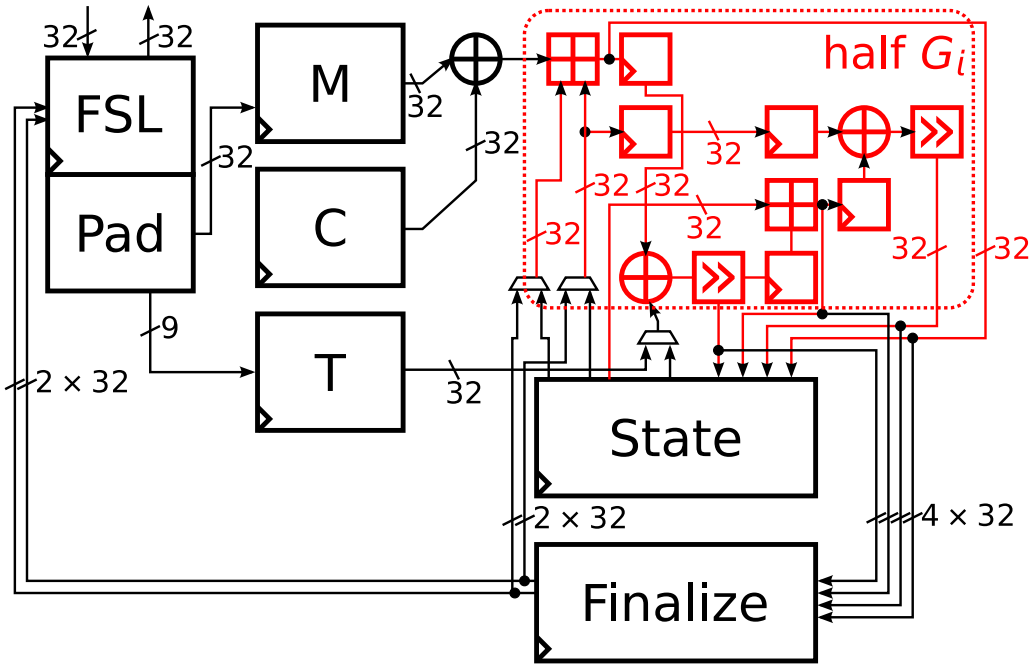
Design Strategy

- Main idea #2: Pipelining of folded architectures
- Benefit #1: Improved clock frequency
- Benefit #2: No (or few) additional clock cycles, if the pipeline is designed carefully.
- Drawback: Often more area is required
- But: For FPGAs a register can often be placed in the same slice as a LUT
- Control logic sometimes more difficult, sometimes easier to implement

Design Strategy

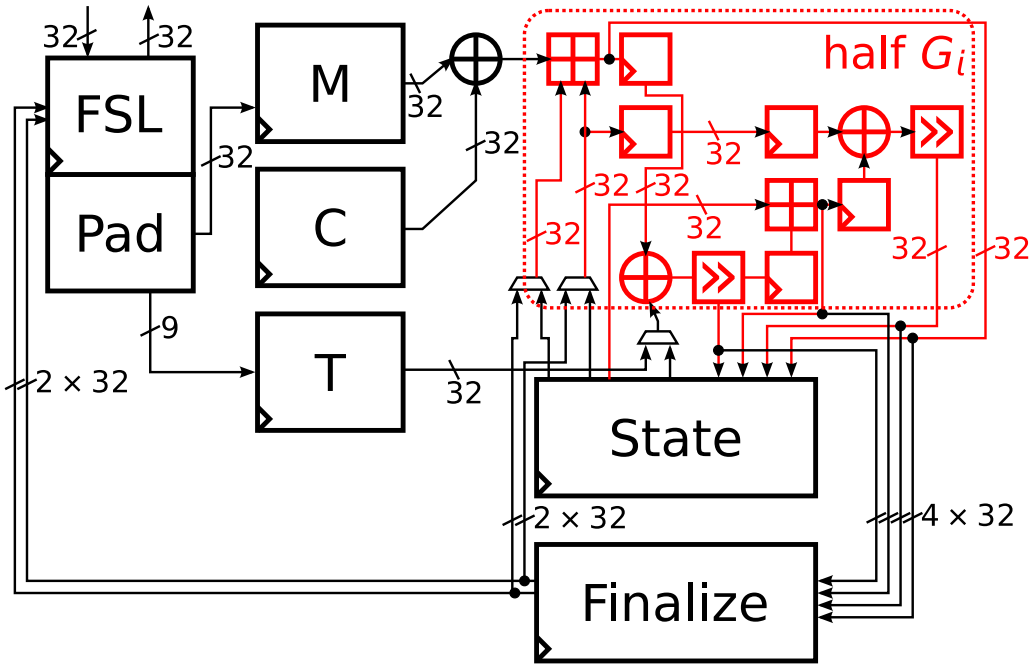
- Main idea #2: Pipelining of folded architectures
- Benefit #1: Improved clock frequency
- Benefit #2: No (or few) additional clock cycles, if the pipeline is designed carefully.
- Drawback: Often more area is required
- But: For FPGAs a register can often be placed in the same slice as a LUT
- Control logic sometimes more difficult, sometimes easier to implement

BLAKE Compact Design Overview



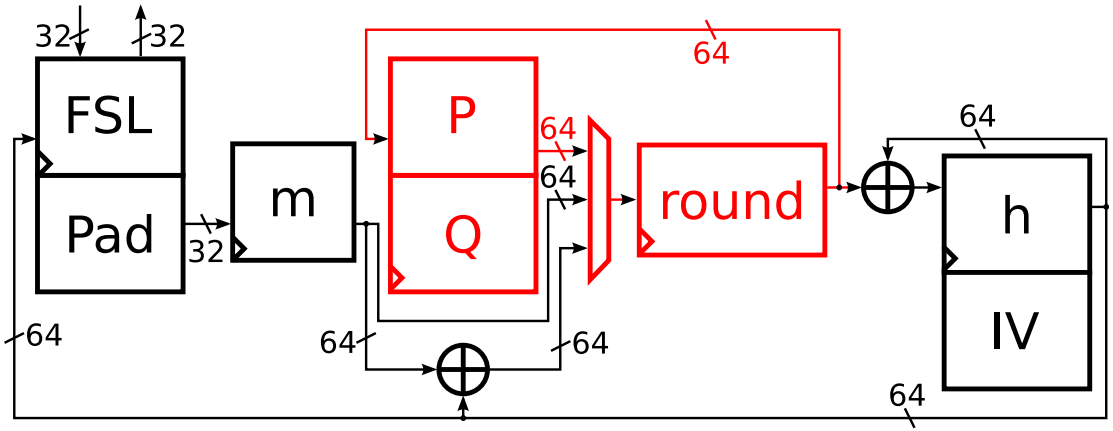
- Two implementations, 1 or 2 halves of a G_i function
- Quasi-Pipeline:
  - Depth 4 for 1 G_i half
  - Depth 2 for 2 G_i halves

BLAKE Compact Design Overview



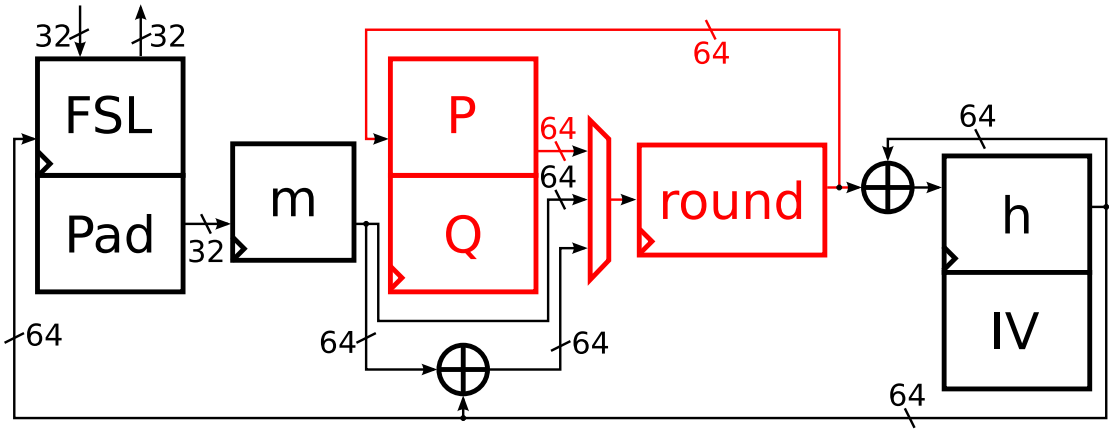
- Two implementations, 1 or 2 halves of a G_i function
- Quasi-Pipeline:
  - Depth 4 for 1 G_i half
  - Depth 2 for 2 G_i halves

Grøstl Compact Design Overview



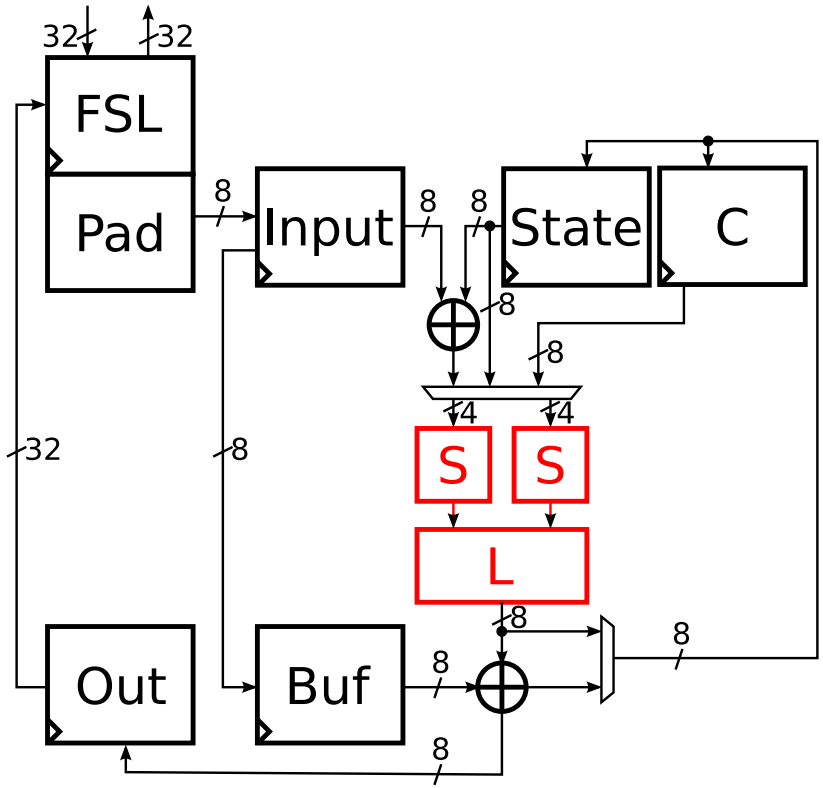
- Hardware for P and Q shared
- Composite field S-box implementation

Grøstl Compact Design Overview



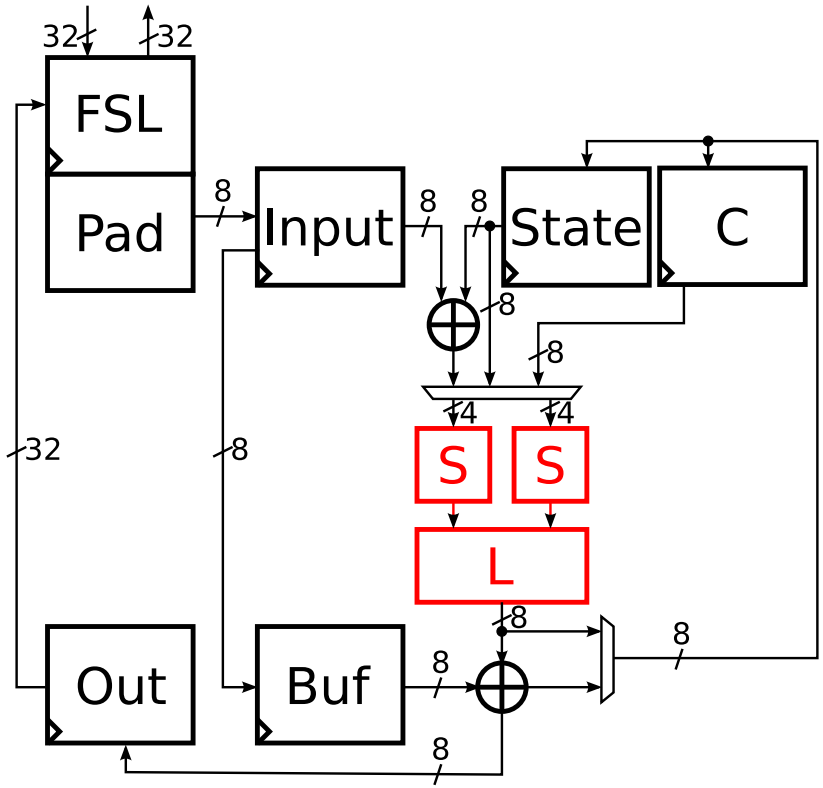
- Hardware for P and Q shared
- Composite field S-box implementation

JH 1st Compact Design Overview



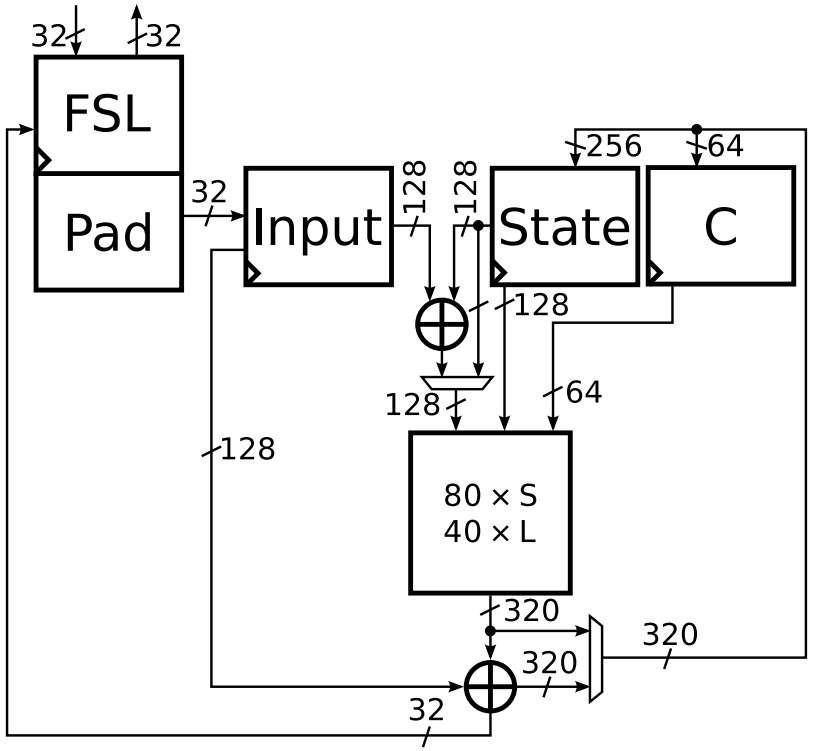
- Very compact implementation (8 bit datapath)
- Shared core for constants computation and round function

JH 1st Compact Design Overview



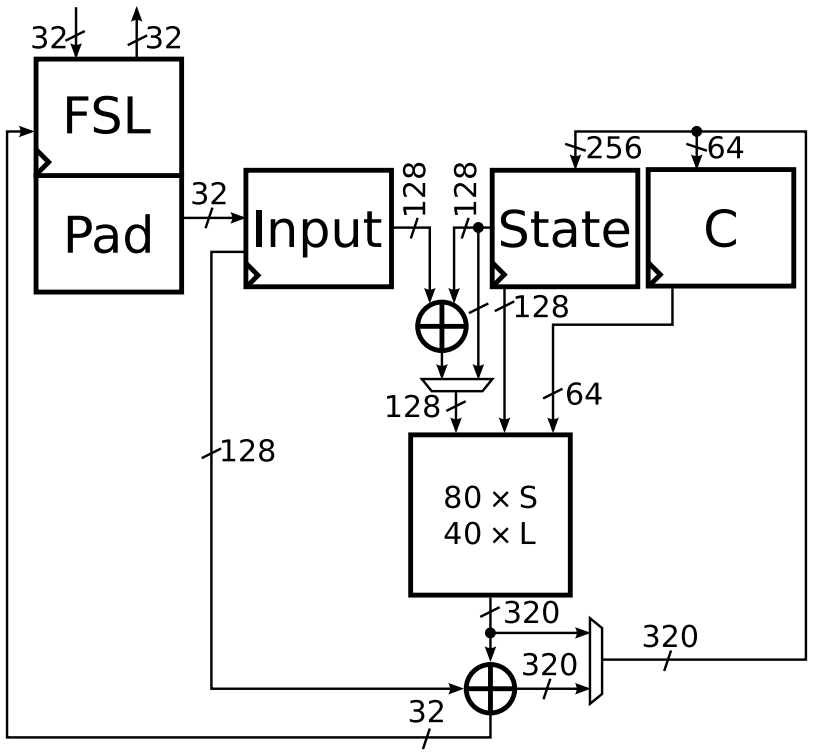
- Very compact implementation (8 bit datapath)
- Shared core for constants computation and round function

JH 2nd Compact Design Overview



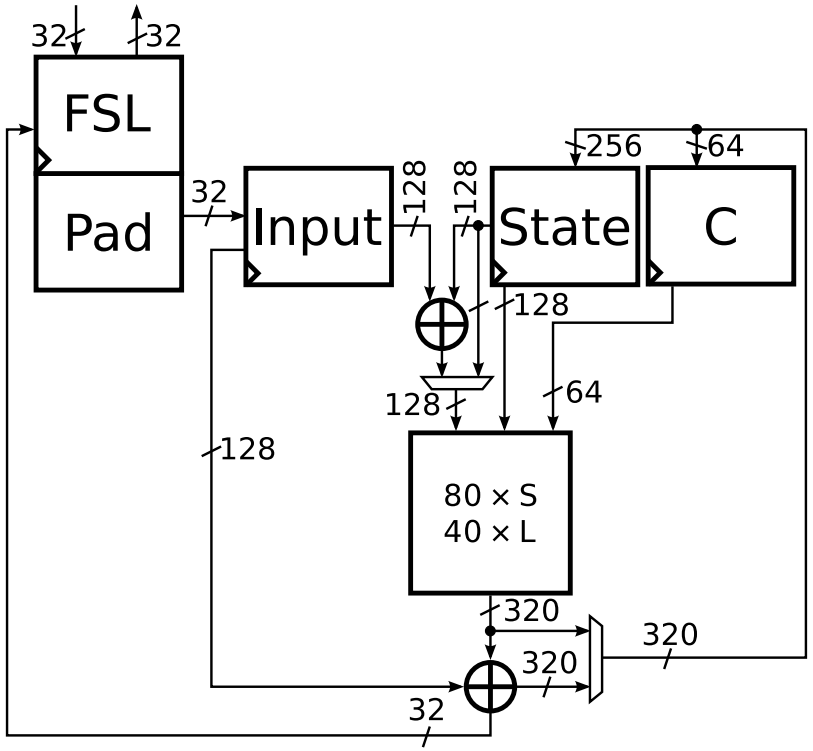
- Very broad datapath (320 bit), but still compact
- Makes output RAM unnecessary for degrouping
- Explicit `LUT6_2` instances for S-boxes, linear transformation

JH 2nd Compact Design Overview



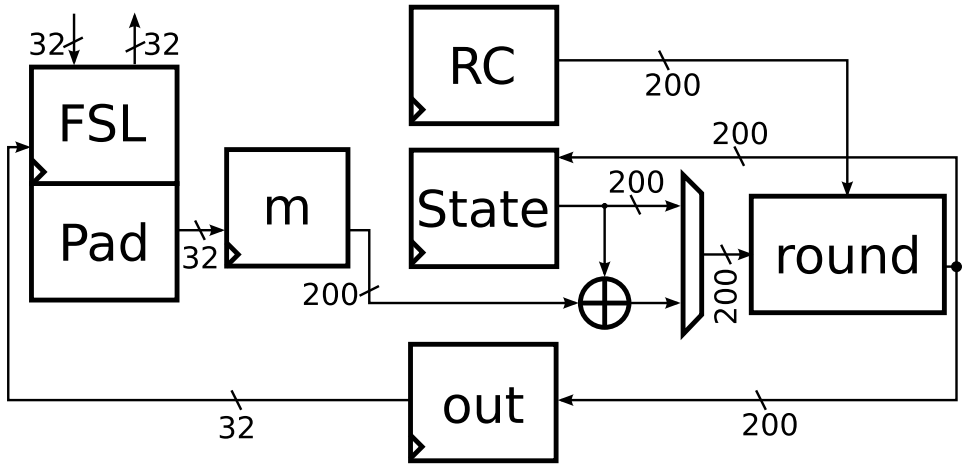
- Very broad datapath (320 bit), but still compact
- Makes output RAM unnecessary for degrouping
- Explicit `LUT6_2` instances for S-boxes, linear transformation

JH 2nd Compact Design Overview



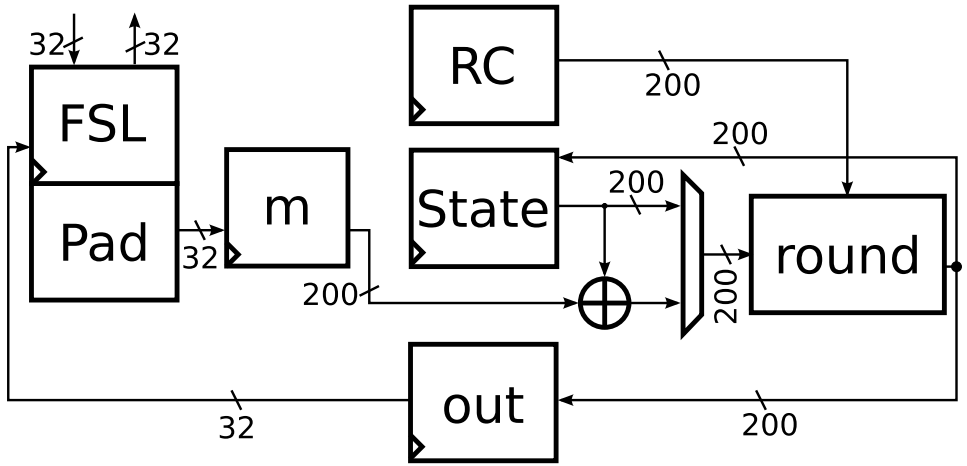
- Very broad datapath (320 bit), but still compact
- Makes output RAM unnecessary for degrouping
- Explicit `LUT6_2` instances for S-boxes, linear transformation

Keccak Compact Design Overview



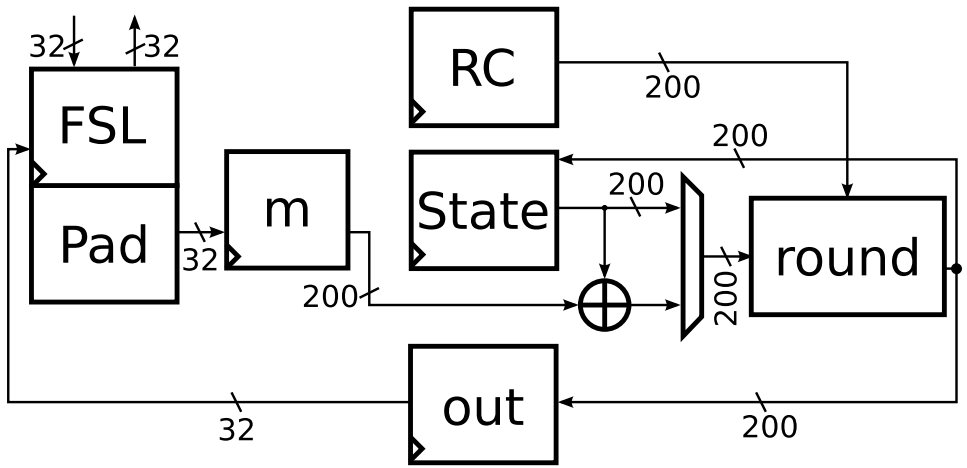
- $\frac{1}{8}$ round function
- Folding with Keccak-'slices' instead of -'lanes'
- Rescheduled round function, permutation at the end

Keccak Compact Design Overview



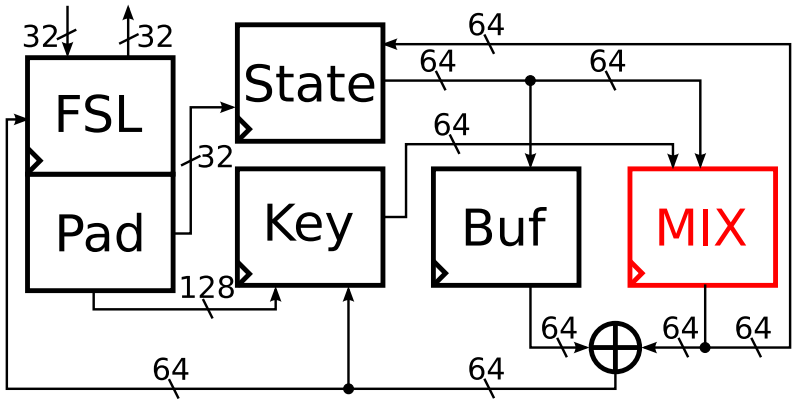
- $\frac{1}{8}$ round function
- Folding with Keccak-'slices' instead of -'lanes'
- Rescheduled round function, permutation at the end

Keccak Compact Design Overview



- $\frac{1}{8}$ round function
- Folding with Keccak-'slices' instead of -'lanes'
- Rescheduled round function, permutation at the end

Skein Compact Design Overview



- One pipelined MIX function, includes the key injection

Analysis Clock Cycles

Algorithm	Per round	Overhead	Complete compression function
BLAKE-1	16	4	228
BLAKE-2	8	5	115
Grøstl	16	0	160
JH-1	160	0	6720
JH-2	4	0	168
Keccak	8	8	200
Skein	8	8	584

Table: Clock Cycles for compression function

Results Virtex-5

Algorithm	Slices	MHz	MBit/s	$\frac{\text{MBit/s}}{\text{Slice}}$
BLAKE-1	251	211	477	1.90 (5)
BLAKE-2	374	163	725	1.94 (4)
Grøstl	368	305	975	2.64 (1)
JH-1	193	283	22	0.11 (7)
JH-2	377	278	847	2.24 (2)
Keccak	393	159	864	2.19 (3)
Skein	519	299	262	0.50 (6)

Results Virtex-6

Algorithm	Slices	MHz	MBit/s	$\frac{\text{MBit/s}}{\text{Slice}}$
BLAKE-1	260	263	590	2.26 (4)
BLAKE-2	419	204	908	2.18 (5)
Grøstl	328	365	1168	3.56 (1)
JH-1	221	442	33	0.14 (7)
JH-2	352	344	1048	2.97 (2)
Keccak	397	197	1071	2.69 (3)
Skein	406	316	277	0.68 (6)

Results Spartan-3

Algorithm	Slices	MHz	MBit/s	$\frac{\text{MBit/s}}{\text{Slice}}$
BLAKE-1	948	88.6	198	0.20 (3)
BLAKE-2	1716	71.6	318	0.18 (4)
Grøstl	1220	148	473	0.38 (1)
JH-1	807	124	9.4	0.011 (7)
JH-2	2060	113	344	0.16 (5)
Keccak	1665	71.2	387	0.23 (2)
Skein	1347	128	112	0.083 (6)

Results Spartan-6

Algorithm	Slices	MHz	MBit/s	$\frac{\text{MBit/s}}{\text{Slice}}$
BLAKE-1	257	155	477	1.85 (4)
BLAKE-2	413	113	725	1.75 (5)
Grøstl	344	236	975	2.83 (1)
JH-1	171	241	22	0.12 (7)
JH-2	372	185	847	2.27 (2)
Keccak	420	122	864	2.05 (3)
Skein	418	210	262	0.62 (6)

Thank you for your attention.