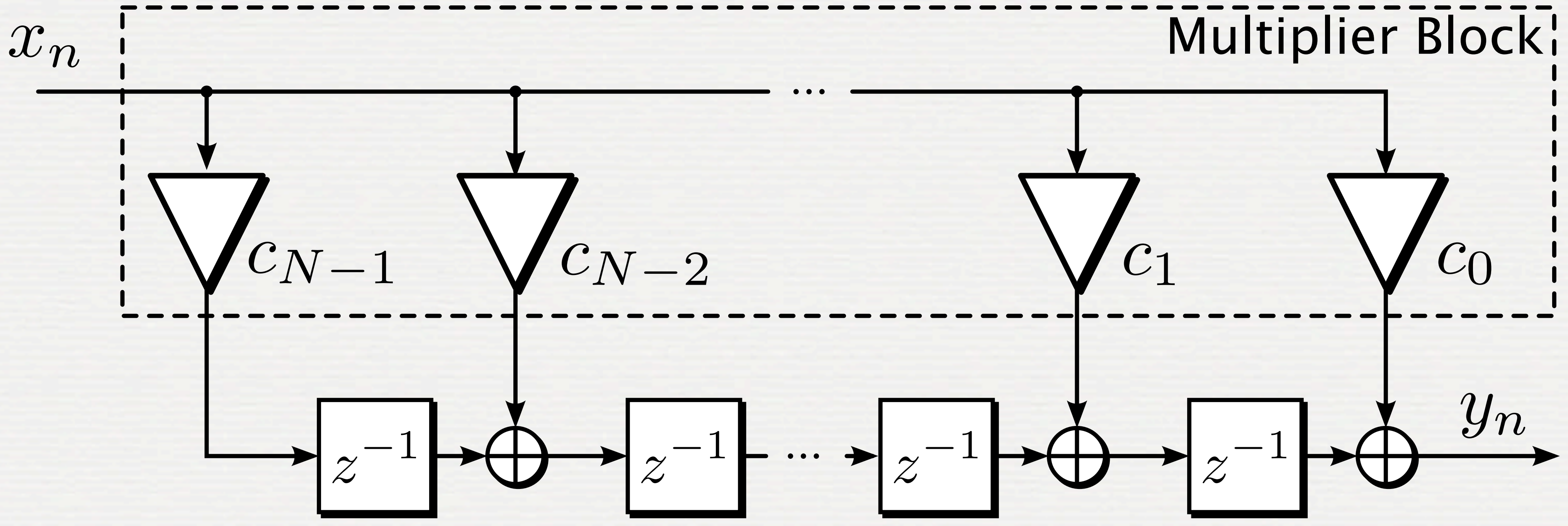
Dynamically Reconfigurable FIR Filter Architectures with Fast Reconfiguration

Martin Kumm, Konrad Möller and Peter Zipf

University of Kassel, Germany

FIR FILTER

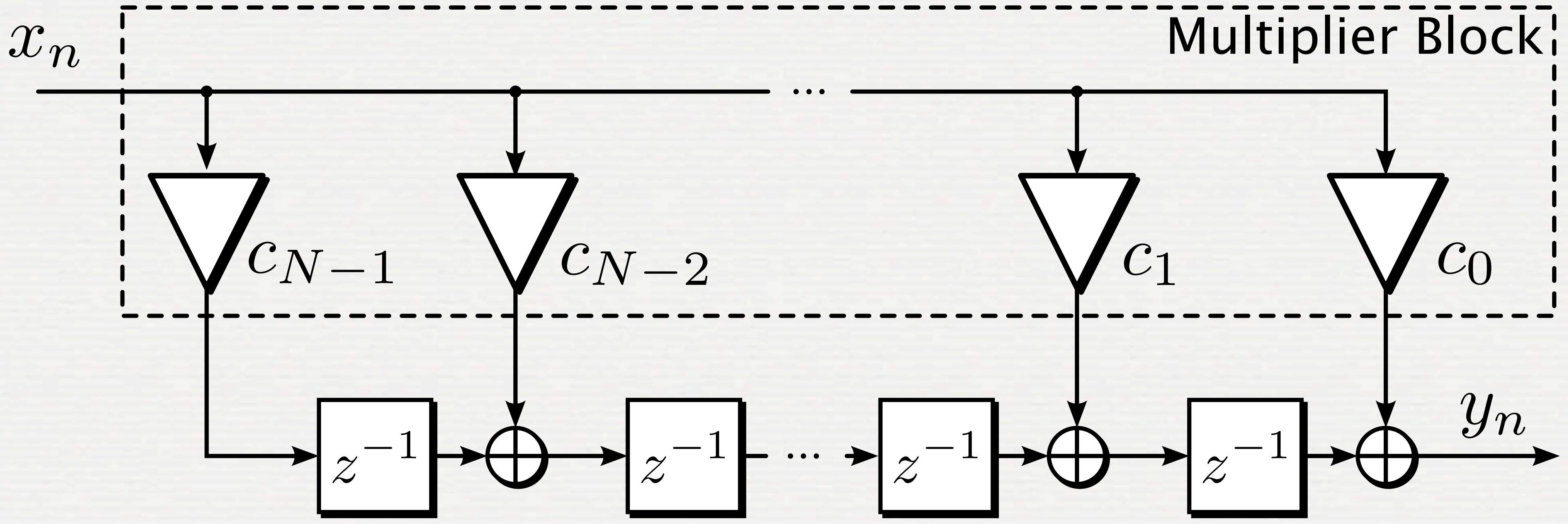
- Fundamental component in digital signal processing
- Computationally complex due to numerous multiply/accumulate operations



WHY RECONFIGURATION?

- Many applications require the change of coefficients...
- ...but only from time to time

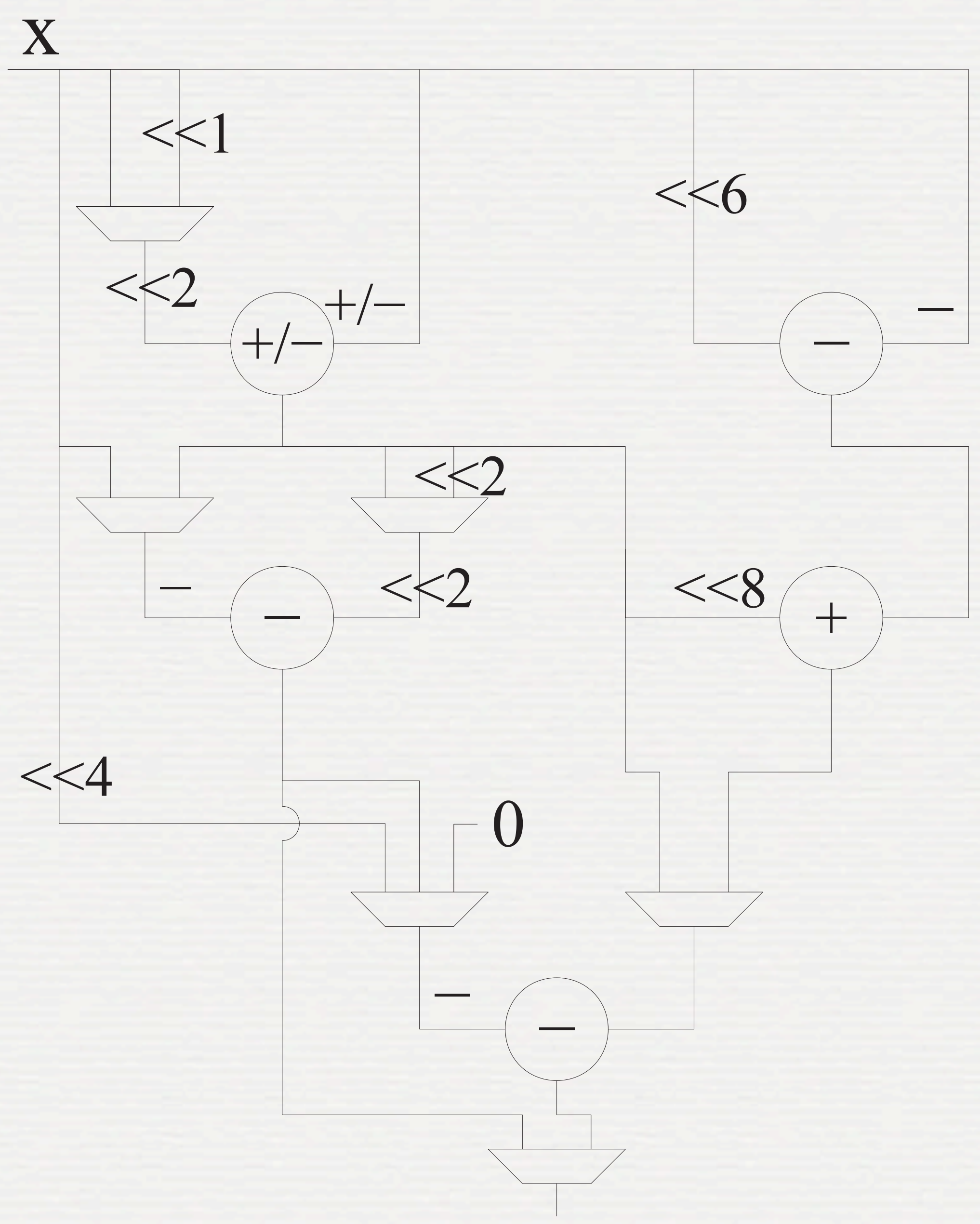
⇨ Possibility to reduce complexity



METHODS OF RECONFIGURATION

1. Integrating multiplexers into the design
2. Partial reconfiguration (e.g., using ICAP)
3. Reconfigurable LUTs

MULTIPLEXER BASED RECONFIGURATION



x·{815,621,831,105}

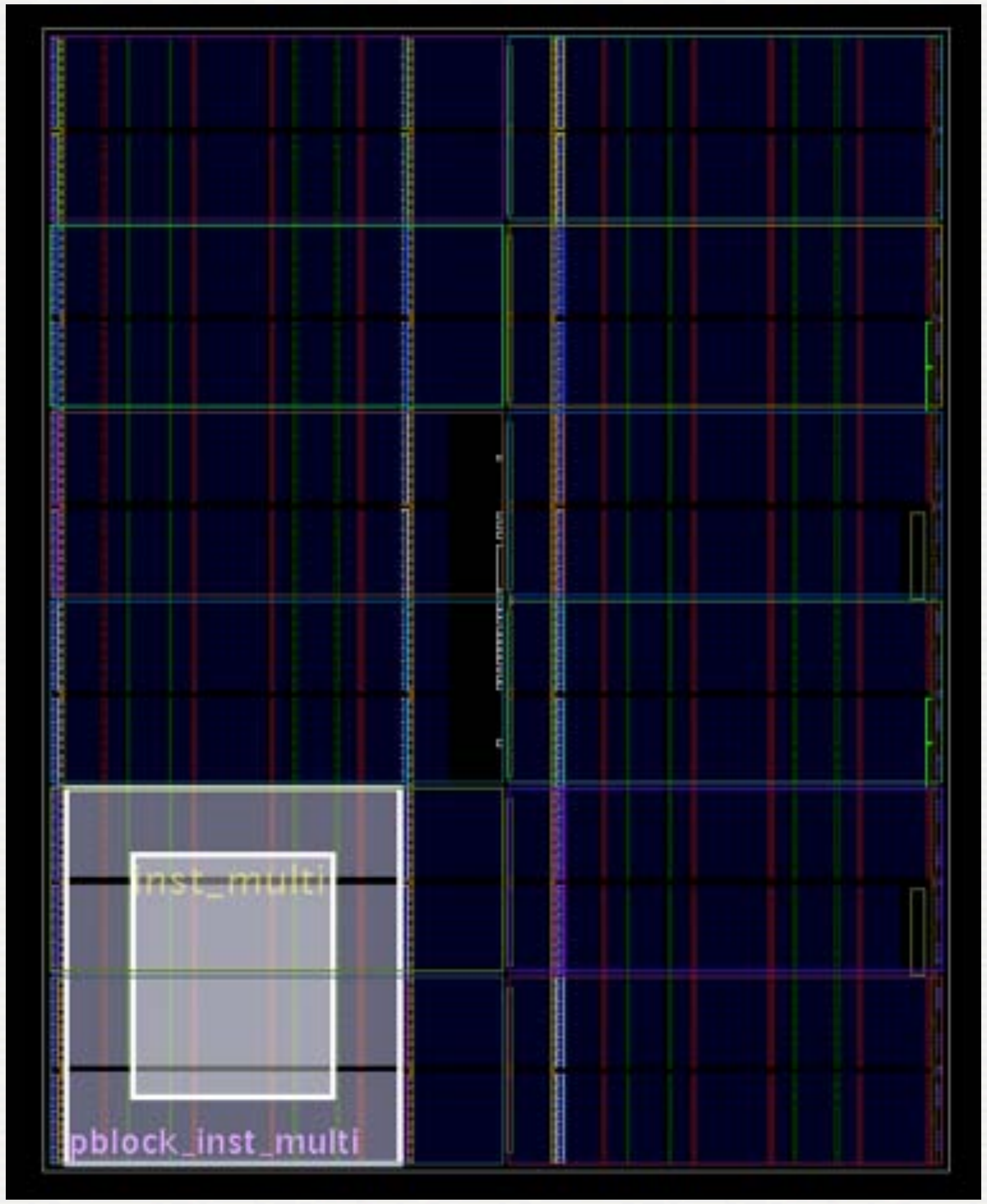
[Faust et al. ’10]

- Multiplexers are integrated in add/shift networks

☺ Extremly fast reconfiguration (single clock cycle)

☹ Only a limited set of coefficients possible!

PARTIAL RECONFIGURATION



- Partial regions of the FPGA are reconfigured via ICAP

☺ Least resources

☺ Arbitrary coefficients...

☹ ... but synthesis needed for each coefficient set

☹ Slow reconfiguration (≈µs/ms)!

RECONFIGURABLE LUTS

- Changing the LUT content only
- Routing has to be fixed
- First academic tool available (TLUT flow, [Bruneel et al. ’11])

☺ Fast reconfiguration (a few clock cycles, ≈ns/µs)

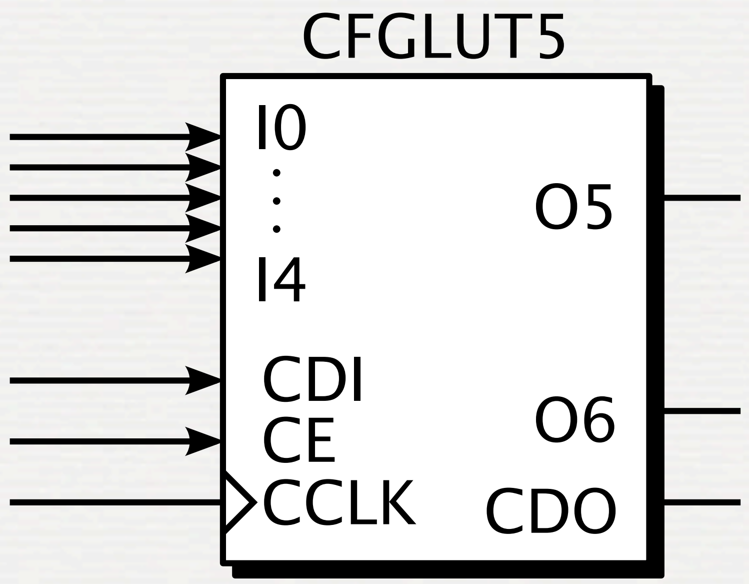
☺ Arbitrary coefficients...

☹ ... but (again) synthesis needed for each coefficient set

⇨ Not, if a generic architecture is transformed to fixed routing

RECONFIGURABLE LUTS

- FPGA components to realize reconfigurable LUTs
  - Older Xilinx FPGAs (Virtex 1-4): Shift-Register LUT (SRL16)
  - Newer Xilinx FPGAs (Virtex 5/6, Spartan 6, 7-Series): CFGLUT5 (similar to SRLC32E but with two output functions)
  - Other FPGA vendors: Distributed RAM or block RAM



METHODS OF RECONFIGURATION

1. Integrating multiplexers into the design
   ⇨ Logic fixed, routing flexible
2. Partial reconfiguration (e.g., using ICAP)
   ⇨ Logic flexible, routing flexible
3. Reconfigurable LUTs
   ⇨ Logic flexible, routing fixed

LUT BASED FIR FILTER

- Two well-known methods that employ LUTs in a fixed structure, suitable for FIR filters:

  1. Distributed Arithmetic [Crosisier et al. ’73] [Zohar ’73] ... ... [Kumm et al. ’13]

  2. LUT based multipliers [Chapman ’96] [Wiatr et al. ’01]

The main question is:

"Which architecture performs best?“

DISTRIBUTED ARITHMETIC

- Main idea is rearranging the underlying inner product

- Resulting function (realized as LUT) is identical for each bit *b*

⇨ Less configuration memory

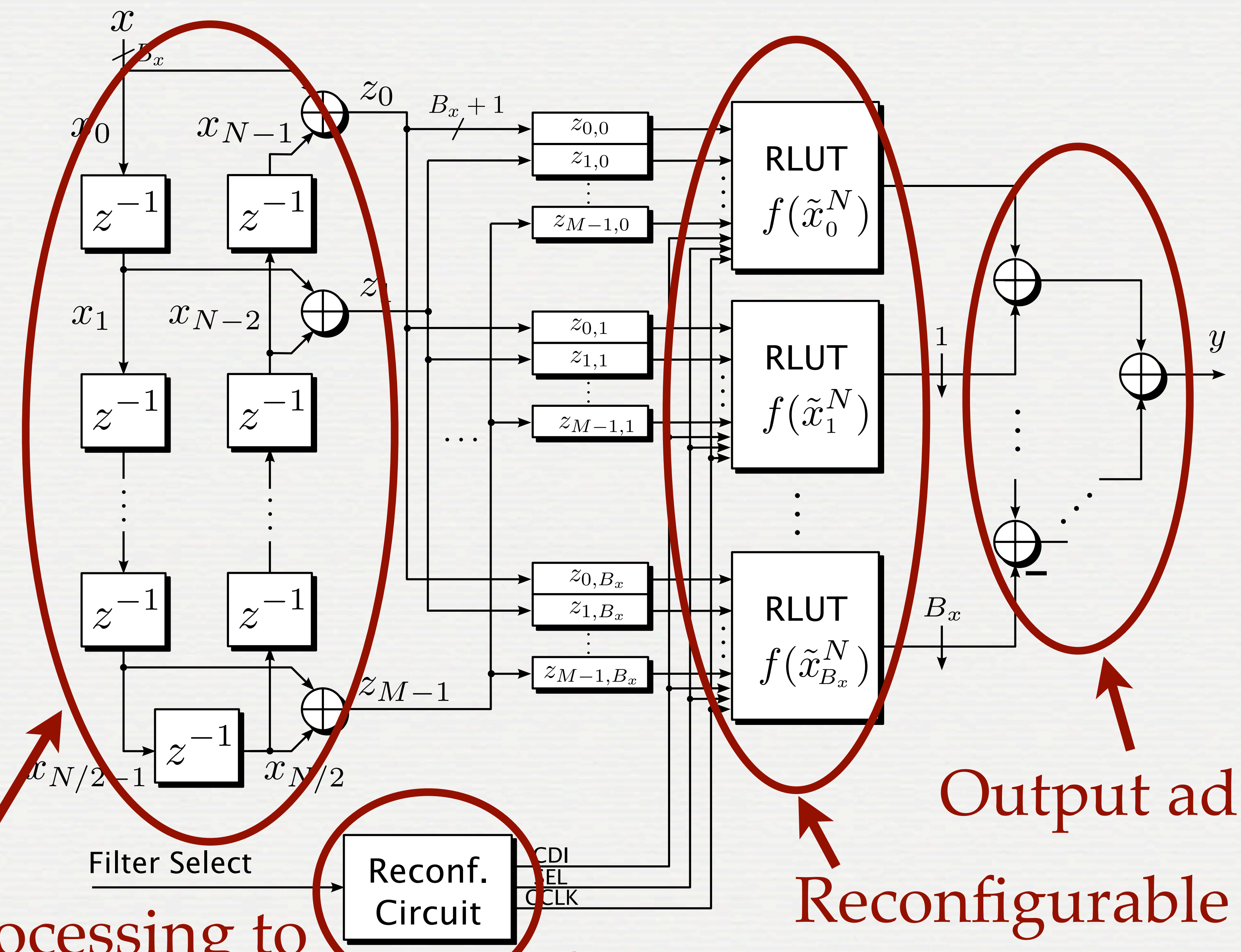
$$y = \mathbf{c} \cdot \mathbf{x} = \sum_{n=0}^{N-1} c_n \, x_n$$

$$= \sum_{n=0}^{N-1} c_n \sum_{b=0}^{B_x-1} 2^b x_{n,b}$$

$$= \sum_{b=0}^{B_x-1} 2^b \underbrace{\sum_{n=0}^{N-1} c_n x_{n,b}}_{=f(\tilde{x}_b^N) \text{ (LUT)}}$$

$$\tilde{x}_b^N = \left(x_{0,b}, x_{1,b}, \ldots, x_{N-1,b}\right)^T$$

DISTRIBUTED ARITHMETIC OVERALL ARCHITECTURE



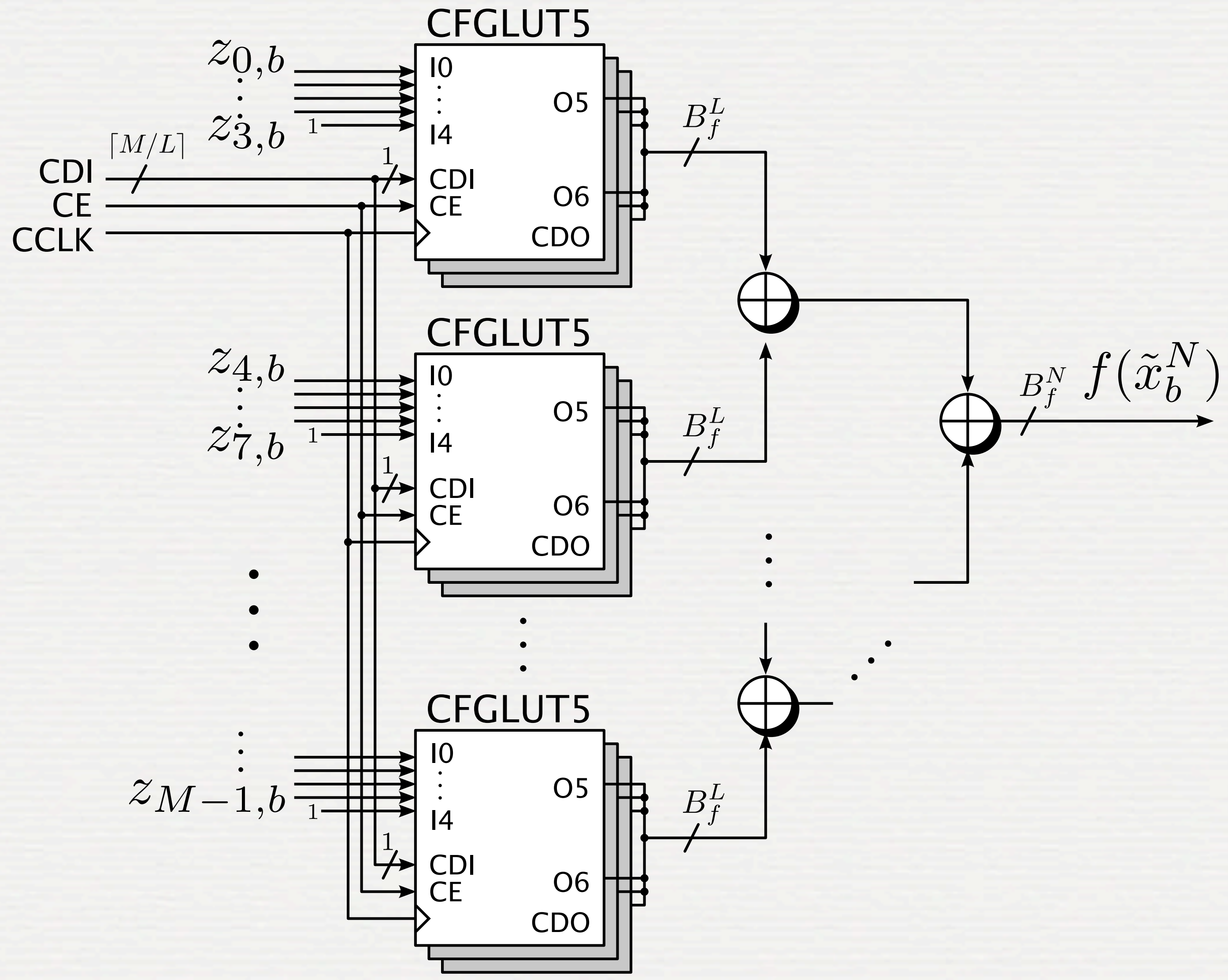
Output adder tree

Reconfigurable LUTs

Pre-processing to exploit coefficient symmetry

Reconfiguration circuit

DISTRIBUTED ARITHMETIC MAPPING TO CFGLUT5

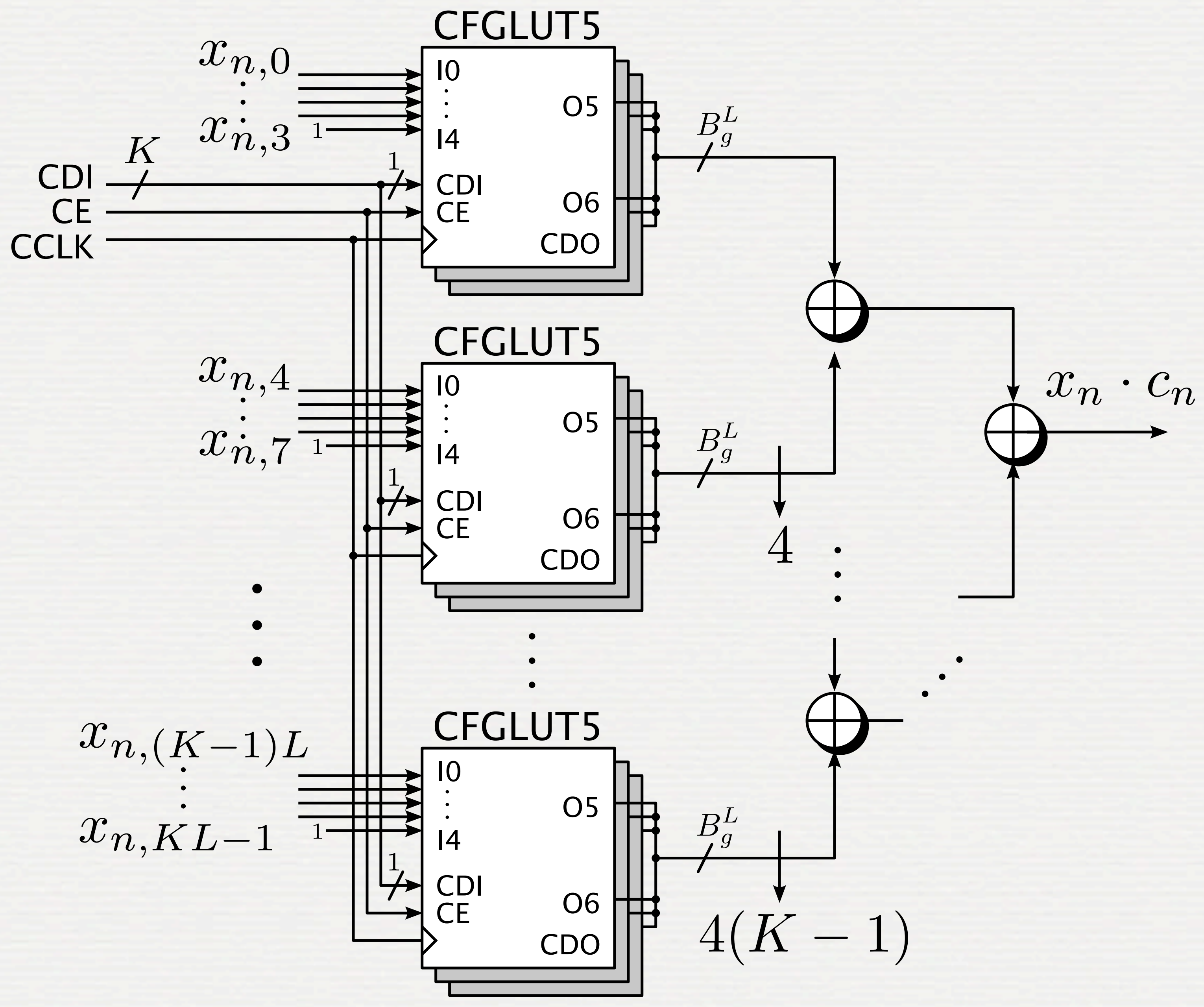


LUT MULTIPLIER FIR FILTER

- Basic Idea: Split a multiplication into smaller chunks which fit into the FPGA LUT:

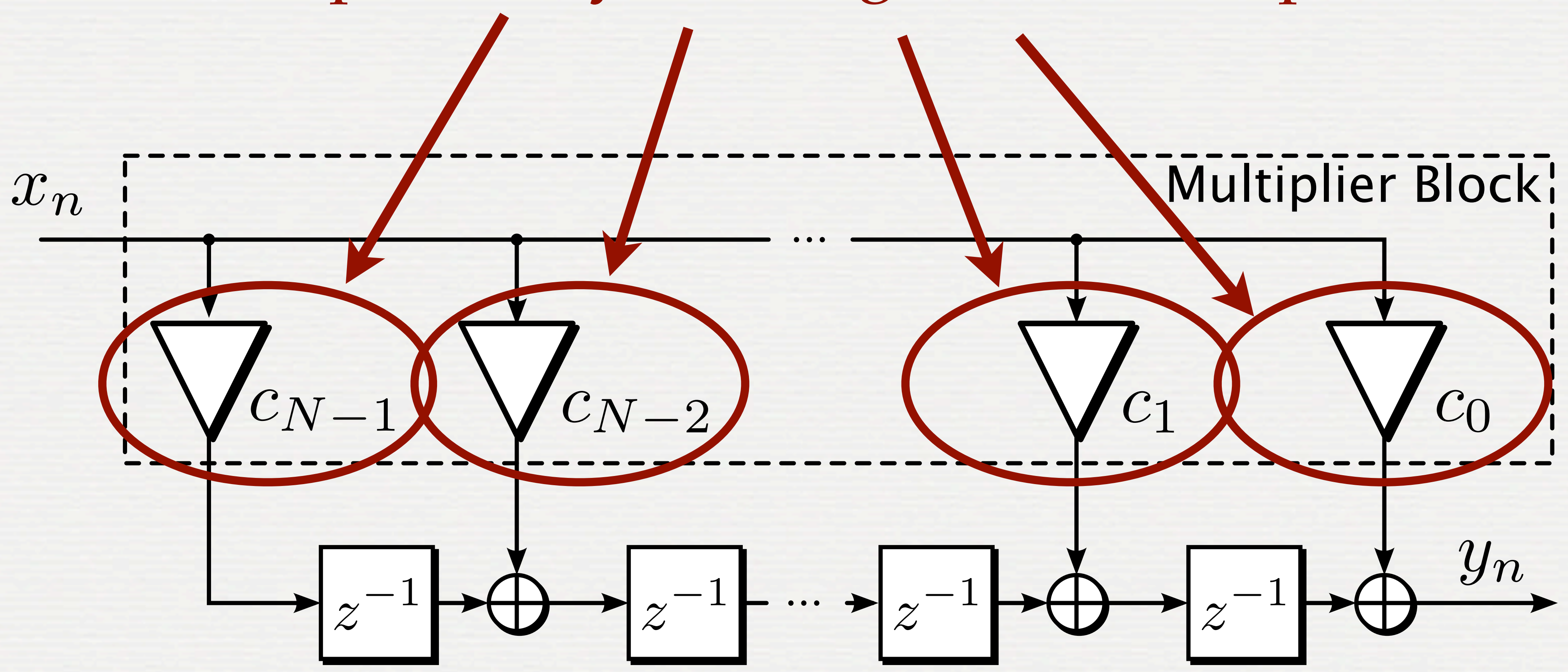
$$\underbrace{c_n \cdot x_n}_{B_c \times B_x \text{ mult.}} = \underbrace{c_n \sum_{b=0}^{L-1} 2^b x_{n,b}}_{B_c \times L \text{ mult.}} + 2^L \underbrace{c_n \sum_{b=0}^{L-1} 2^b x_{n,b+L}}_{B_c \times L \text{ mult.}} + \dots$$

LUT MULTIPLIER MAPPING TO CFGLUT5

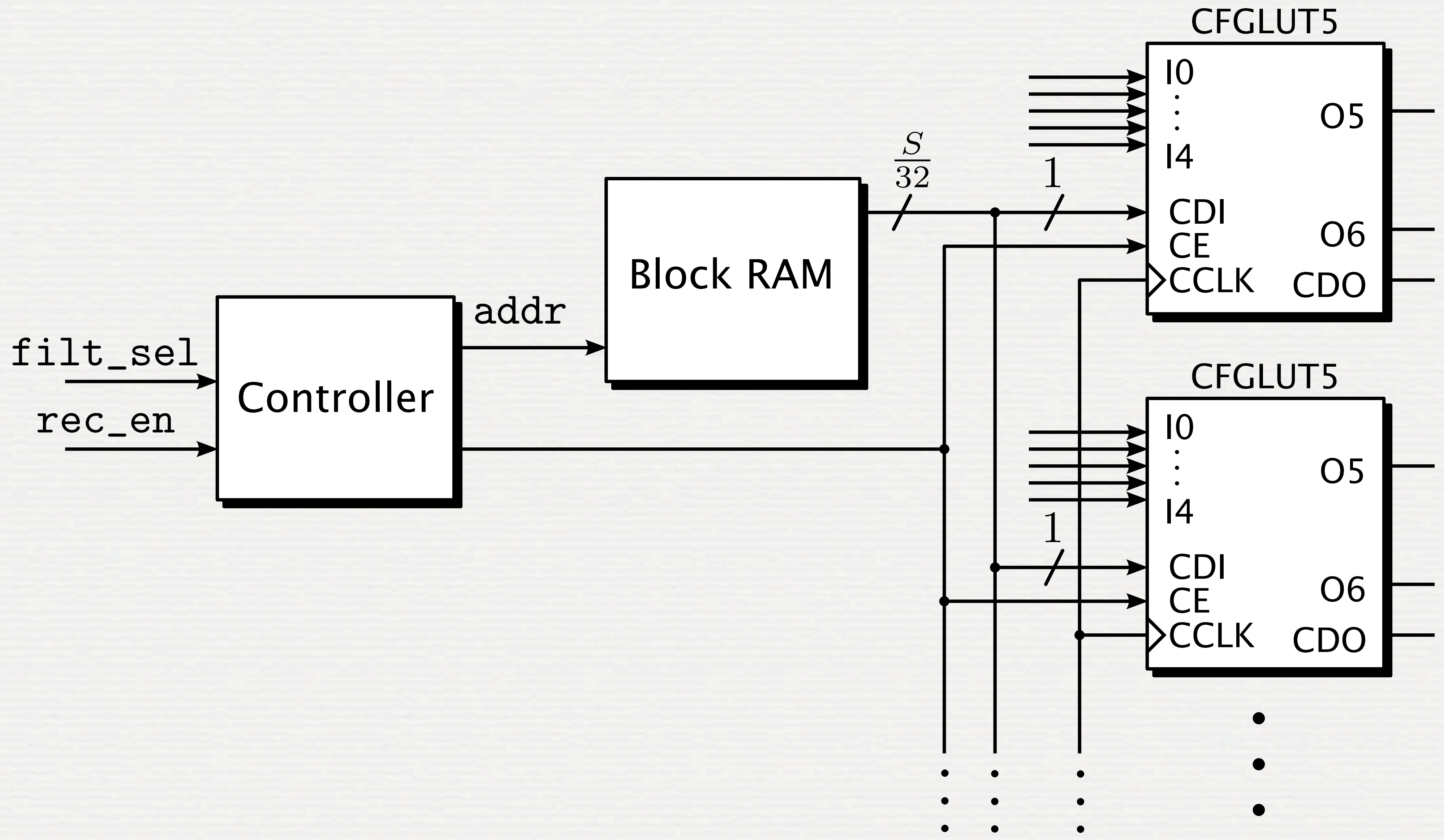


LUT MULTIPLIER OVERALL ARCHITECTURE

Replaced by reconfigurable multipliers



CONTROL ARCHITECTURE



RESOURCE COMPARISON

Distributed Arithmetic	LUT Multiplier FIR
$B_x + 1$ LUTs with M inputs	M LUTs with B_x inputs
CFGLUTs:	CFGLUTs:
$(B_x + 1) \lceil M/4 \rceil \lceil B_c/2 + 1 \rceil$	$M \lceil B_x/4 \rceil \lceil B_c/2 + 2 \rceil$
$\approx \frac{1}{4}(B_x + 1)M(B_c/2 + 1)$	$\approx \frac{1}{4}B_x M(B_c/2 + 2)$

$M = \lceil N/2 \rceil$: No. of unique taps

B_x/B_c: input/coefficient bit width

RESOURCE COMPARISON

Distributed Arithmetic	LUT Multiplier FIR
$B_x + 1$ LUTs with M inputs	M LUTs with B_x inputs
CFGLUTs: $(B_x+1)\lceil M/4\rceil\lceil B_c/2+1\rceil \approx \frac{1}{4}(B_x+1)M(B_c/2+1)$	CFGLUTs: $M\lceil B_x/4\rceil\lceil B_c/2+2\rceil \approx \frac{1}{4}B_xM(B_c/2+2)$

Surprisingly, CFGLUT requirements are very similar!

RESOURCE COMPARISON

Distributed Arithmetic	LUT Multiplier FIR
Adders: $M + B_x + (B_x + 1)\lceil M/4 \rceil$	Adders: $2M - 1 + M\lceil B_x/4 \rceil$

⇨ So, LUT multiplier based FIR filters are better when...

$$2M - 1 + MB_x/4 < M + B_x + (B_x + 1)M/4$$
$$\vdots$$
$$\frac{3}{4}M - 1 < B_x$$

...,i.e., the input word size B_x is greater than approximately half the number of coefficients $M = \lceil N/2 \rceil$

RESULTS: 1ST EXPERIMENT

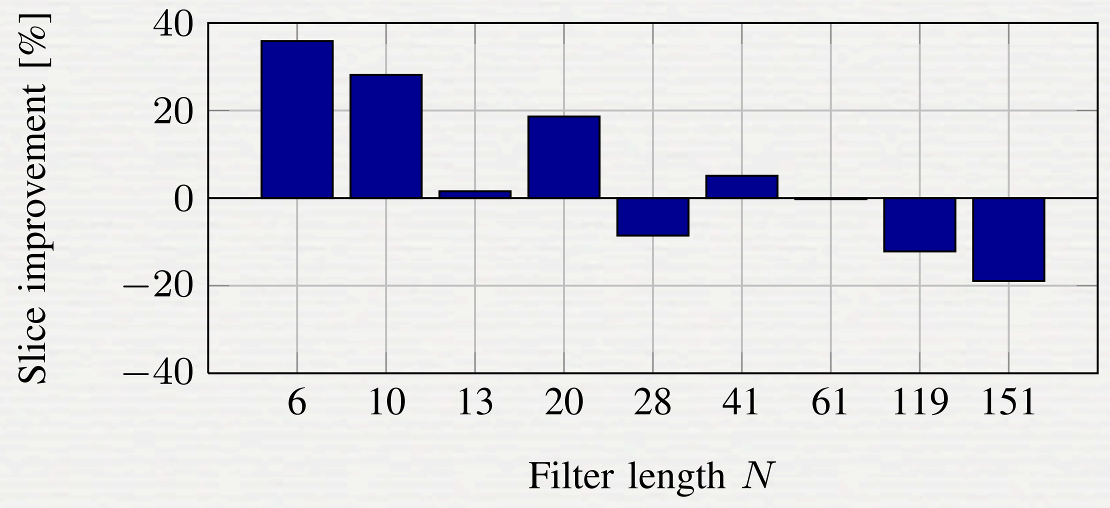
- Synthesis experiment for Virtex 6
- Nine benchmark filters with length N=6...151
- Input word size $B_x \in \{8, 16, 24, 32\}$

⇨ Very fast reconfiguration times: 49...106 ns

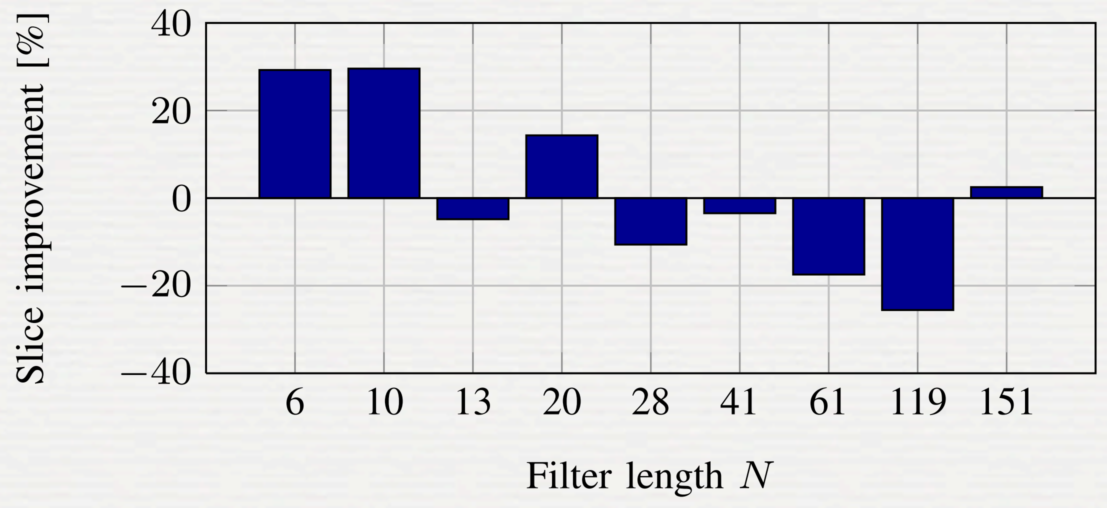
⇨ High clock frequencies: 472 MHz/494 MHz (DA/LUT mult.)

RESULTS: 1ST EXPERIMENT

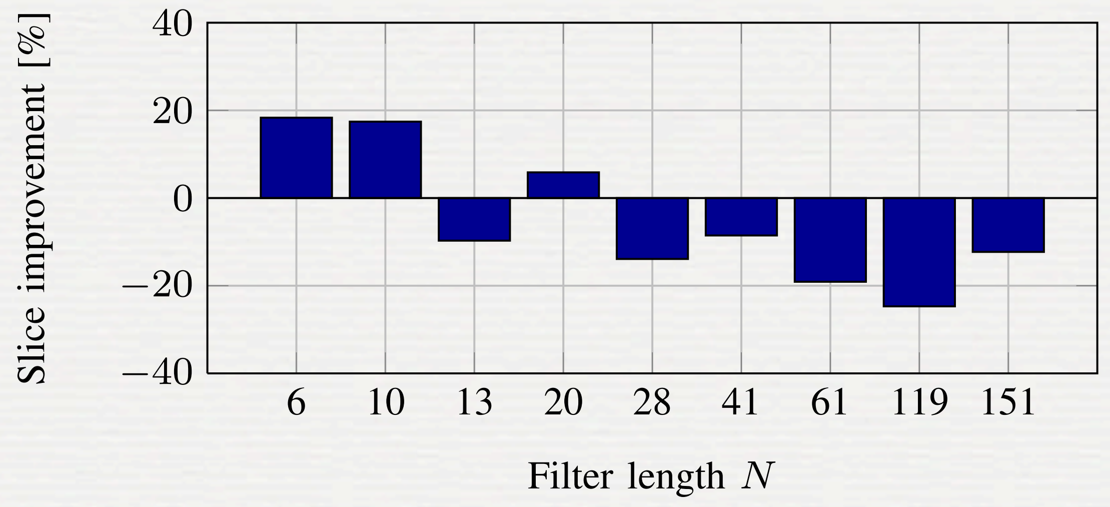
LUT Multiplier improvement compared to DA:



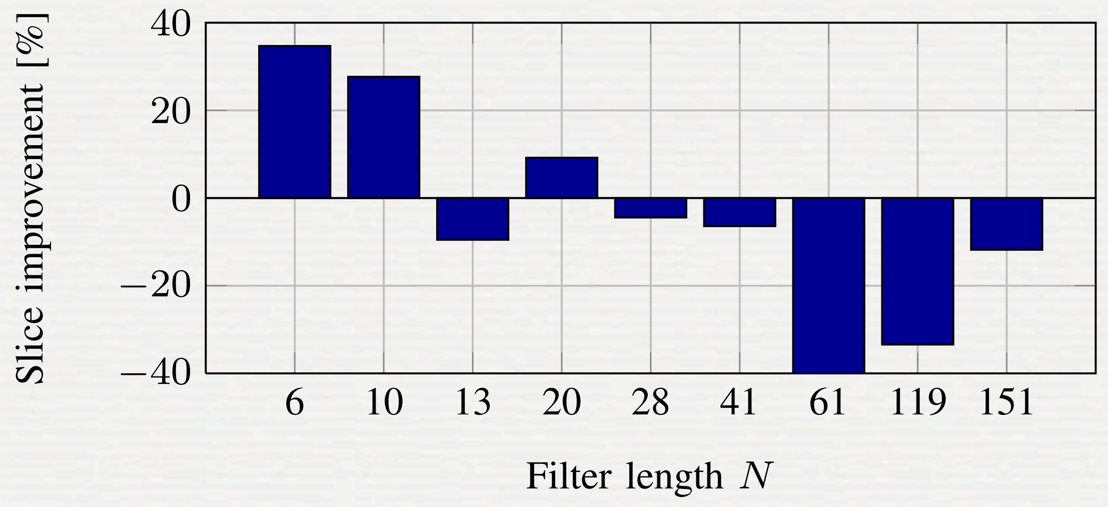
(a) Input word size $B_x = 8\,\text{bit}$



(b) Input word size $B_x = 16\,\text{bit}$



(c) Input word size $B_x = 24\,\text{bit}$

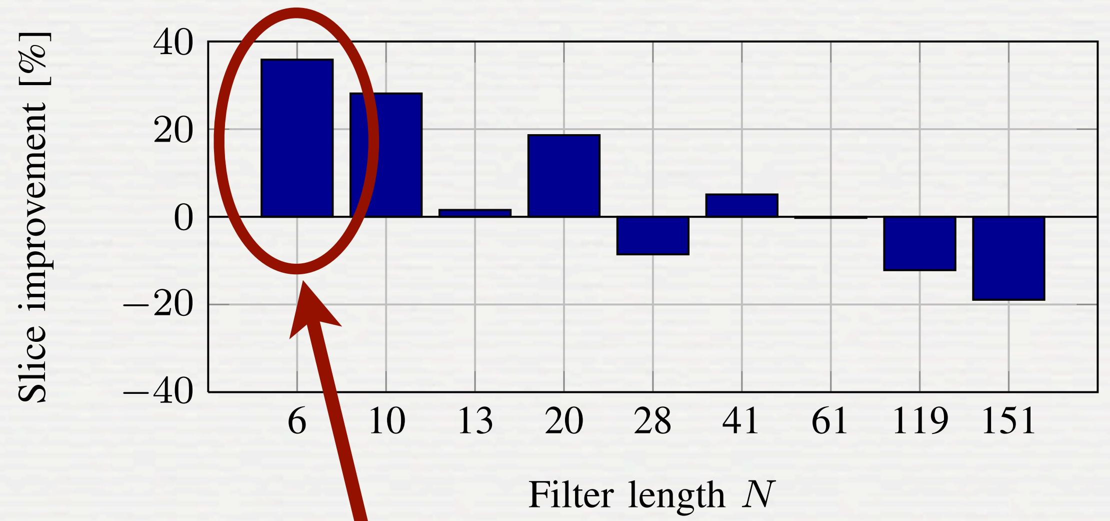


(d) Input word size $B_x = 32\,\text{bit}$

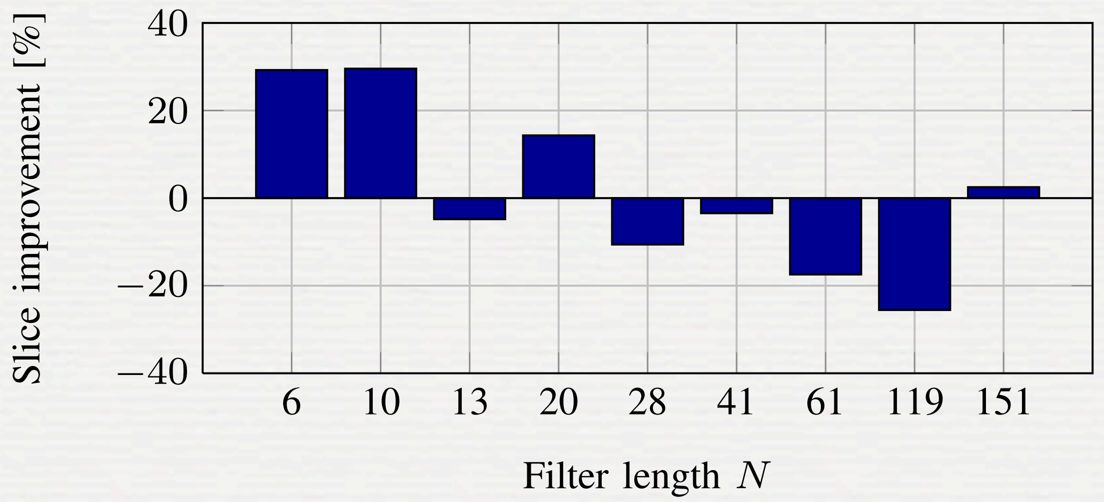
As expected, the LUT multiplier architecture is best for low *N*

RESULTS: 1ST EXPERIMENT

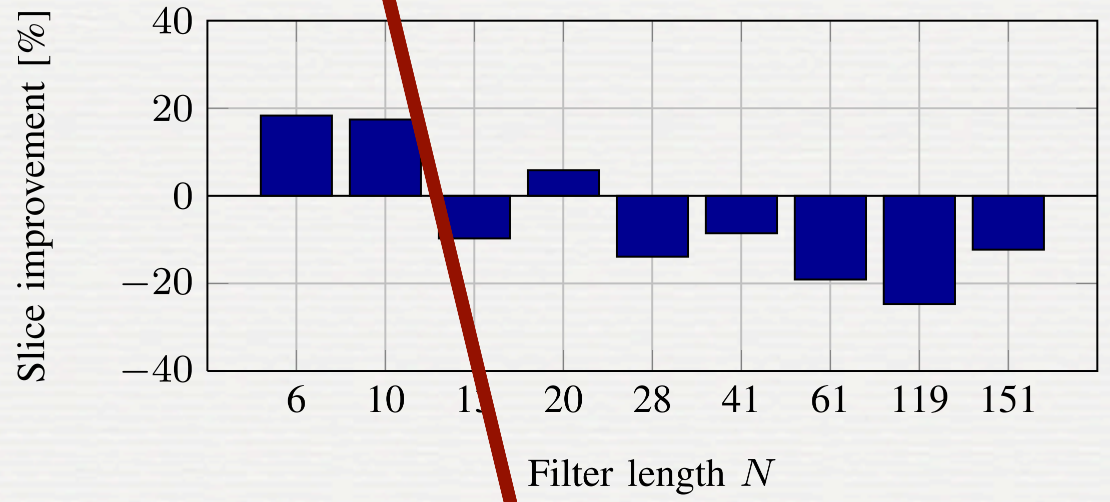
LUT Multiplier improvement compared to DA:



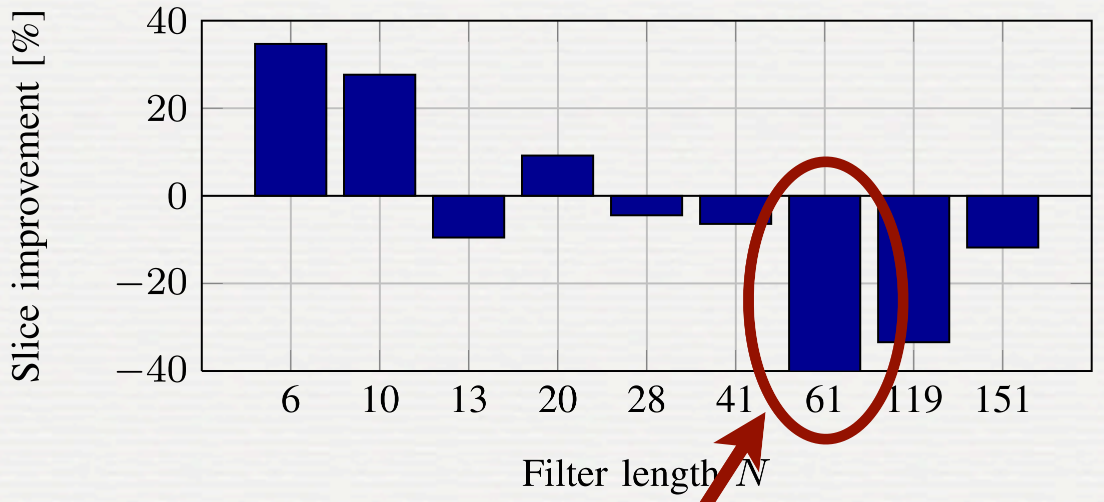
(a) Input word size $B_x = 8$ bit



(b) Input word size $B_x = 16$ bit



(c) Input word size $B_x = 24$ bit



(d) Input word size $B_x = 32$ bit

Choosing the right architecture can save up to 40% slices

RESULTS: 2ND EXPERIMENT

- Comparison with partial reconfiguration via ICAP
- Ten different filters with N=41 were highly optimized using PMCM optimization RPAG [Kumm et al. '12]

Method	S [bit]	Slices	f_{clk} [MHz]	T_{rec} [ns]
RPAG with ICAP	746496	502...569	386.7...448.8	233280
Reconf. FIR DA	1920	1071	521.9	61.3
Reconf. FIR LUT	14784	1108	487.8	65.6

Configuration memory is reduced by a factor of 1/388 (DA) and 1/50 (LUT Mult.) ☺

RESULTS: 2ND EXPERIMENT

- Comparison with partial reconfiguration via ICAP
- Ten different filters with N=41 were highly optimized using PMCM optimization RPAG [Kumm et al. '12]

Method	S [bit]	Slices	f_{clk} [MHz]	T_{rec} [ns]
RPAG with ICAP	746496	502...569	386.7...448.8	233280
Reconf. FIR DA	1920	1071	521.9	61.3
Reconf. FIR LUT	14784	1108	487.8	65.6

Slice requirements are roughtly doubled ☹

RESULTS: 2ND EXPERIMENT

- Comparison with partial reconfiguration via ICAP
- Ten different filters with N=41 were highly optimized using PMCM optimization RPAG [Kumm et al. '12]

Method	S [bit]	Slices	f_{clk} [MHz]	T_{rec} [ns]
RPAG with ICAP	746496	502...569	386.7...448.8	233280
Reconf. FIR DA	1920	1071	521.9	61.3
Reconf. FIR LUT	14784	1108	487.8	65.6

Perfomance is similar

RESULTS: 2ND EXPERIMENT

- Comparison with partial reconfiguration via ICAP
- Ten different filters with N=41 were highly optimized using PMCM optimization RPAG [Kumm et al. '12]

Method	S [bit]	Slices	f_{clk} [MHz]	T_{rec} [ns]
RPAG with ICAP	746496	502...569	386.7...448.8	233280
Reconf. FIR DA	1920	1071	521.9	61.3
Reconf. FIR LUT	14784	1108	487.8	65.6

Reconfiguration time is drastically reduced
by a factor of 1/3556! ☺

CONCLUSION

- Two different reconfigurable FIR filter architectures for arbitrary coefficient sets were analyzed

- Both are implemented using reconfigurable LUTs (CFGLUTs)

- The LUT multiplier architecture typically needs less slices when input word size is greater than approx. half the number of coefficients (and vice versa)

- Both architectures offer reconfiguration times of about 3500 times faster than partial reconfiguration using ICAP

- This is paid by twice the number of slice resources

RECOSOC CONCLUSION

If you have a reconfigurable
FPGA circuit which allows a fixed routing:

Use reconfigurable LUTs!

THANK YOU!