Pipelined FPGA Adders

Bogdan Pasca

joint work with:

Florent de Dinechin and **Hong Diep Nguyen**

FPL, Aug. 31st - Sep. 2nd, 2010

Why a paper on integer addition on FPGAs in 2010?

Why a paper on integer addition on FPGAs in 2010?

The FloPoCo project

The FloPoCo project

- Goals
  - provide new and exotic arithmetic operators
    - highly complex operators (floating-point elementary functions)
    - specialized operators (constant multipliers, squarers)

The FloPoCo project

- Goals
  - provide new and exotic arithmetic operators
    - highly complex operators (floating-point elementary functions)
    - specialized operators (constant multipliers, squarers)
    - ... (currently about ∞ operators in about 50 classes)

The FloPoCo project

- Goals
  - provide new and exotic arithmetic operators
    - highly complex operators (floating-point elementary functions)
    - specialized operators (constant multipliers, squarers)
    - ... (currently about ∞ operators in about 50 classes)
  - optimized for a wide range of contexts
    - adapt to the application's precision needs
    - match target hardware
    - run at target frequency
    - minimize resource consumption

The FloPoCo project

- Goals
  - provide new and exotic arithmetic operators
    - highly complex operators (floating-point elementary functions)
    - specialized operators (constant multipliers, squarers)
    - ... (currently about ∞ operators in about 50 classes)
  - optimized for a wide range of contexts
    - adapt to the application's precision needs
    - match target hardware
    - run at target frequency
    - minimize resource consumption
  - in an easy to use open-source framework.

The FloPoCo project

- Goals
  - provide new and exotic arithmetic operators
    - highly complex operators (floating-point elementary functions)
    - specialized operators (constant multipliers, squarers)
    - ... (currently about ∞ operators in about 50 classes)
  - optimized for a wide range of contexts
    - adapt to the application's precision needs
    - match target hardware
    - run at target frequency
    - minimize resource consumption
  - in an easy to use open-source framework.
- Method:
  - assemble sub-components built for the same target frequency

Why a paper on integer addition on FPGAs in 2010?

The FloPoCo project

- Goals
  - provide new and exotic arithmetic operators
    - highly complex operators (floating-point elementary functions)
    - specialized operators (constant multipliers, squarers)
    - ... (currently about ∞ operators in about 50 classes)
  - optimized for a wide range of contexts
    - adapt to the application's precision needs
    - match target hardware
    - run at target frequency
    - minimize resource consumption
  - in an easy to use open-source framework.
- Method:
  - assemble sub-components built for the same target frequency

Addition is the most useful of these components

flopoco -frequency=400 FPExp 11 52 (double precision)

```
(...)
| | | | | |---Entity IntAdder_16_f325_slice_SRL_noBUFFER_uid10_Alternative:
| | | | | | Not pipelined
(...)
| | | | |---Entity IntAdder_52_f400_slice_SRL_noBUFFER_uid16_Classical:
| | | | | Pipeline depth = 1
| | | |---Entity IntCompressorTree_52_2_uid15:
| | | | Pipeline depth = 1
| | |---Entity IntTruncMultiplier_30_34_35_signed:
| | | Pipeline depth = 4
| | |---Entity IntAdder_42_f400_slice_SRL_BUFFER_uid17_Classical:
| | | Pipeline depth = 1
| |---Entity PolynomialEvaluator_d2:
| | Pipeline depth = 15
|---Entity FunctionEvaluator:
| Pipeline depth = 17
|---Entity IntAdder_48_f400_slice_SRL_noBUFFER_uid18_Classical:
| Pipeline depth = 2
|---Entity IntAdder_48_f400_slice_SRL_noBUFFER_uid19_Classical:
| Pipeline depth = 2
| | |---Entity IntAdder_85_f325_slice_SRL_noBUFFER_uid22_Classical:
| | | Pipeline depth = 1
| |---Entity IntCompressorTree_85_3_uid21:
| | Pipeline depth = 2
|---Entity IntMultiplier_47_48_uid20:
| Pipeline depth = 6
|---Entity IntAdder_57_f400_slice_SRL_noBUFFER_uid23_Classical:
| Pipeline depth = 2
|---Entity IntAdder_65_f400_slice_SRL_noBUFFER_uid24_Classical:
| Pipeline depth = 2
Entity FPExp_11_52_400:
Pipeline depth = 52
```

flopoco -frequency=400 FPExp 11 52 (double precision)

```
(...)
| | | | | |---Entity IntAdder_16_f325_slice_SRL_noBUFFER_uid10_Alternative:
| | | | | | Not pipelined
(...)
| | | | |---Entity IntAdder_52_f400_slice_SRL_noBUFFER_uid16_Classical:
| | | | | Pipeline depth = 1
| | | |---Entity IntCompressorTree_52_2_uid15:
| | | | Pipeline depth = 1
| | |---Entity IntTruncMultiplier_30_34_35_signed:
| | | Pipeline depth = 4
| | |---Entity IntAdder_42_f400_slice_SRL_BUFFER_uid17_Classical:
| | | Pipeline depth = 1
| |---Entity PolynomialEvaluator_d2:
| | Pipeline depth = 15
|---Entity FunctionEvaluator:
| Pipeline depth = 17
|---Entity IntAdder_48_f400_slice_SRL_noBUFFER_uid18_Classical:
| Pipeline depth = 2
|---Entity IntAdder_48_f400_slice_SRL_noBUFFER_uid19_Classical:
| Pipeline depth = 2
| | |---Entity IntAdder_85_f325_slice_SRL_noBUFFER_uid22_Classical:
| | | Pipeline depth = 1
| |---Entity IntCompressorTree_85_3_uid21:
| | Pipeline depth = 2
|---Entity IntMultiplier_47_48_uid20:
| Pipeline depth = 6
|---Entity IntAdder_57_f400_slice_SRL_noBUFFER_uid23_Classical:
| Pipeline depth = 2
|---Entity IntAdder_65_f400_slice_SRL_noBUFFER_uid24_Classical:
| Pipeline depth = 2
Entity FPExp_11_52_400:
Pipeline depth = 52
```

flopoco -frequency=400 FPExp 11 52 (double precision)

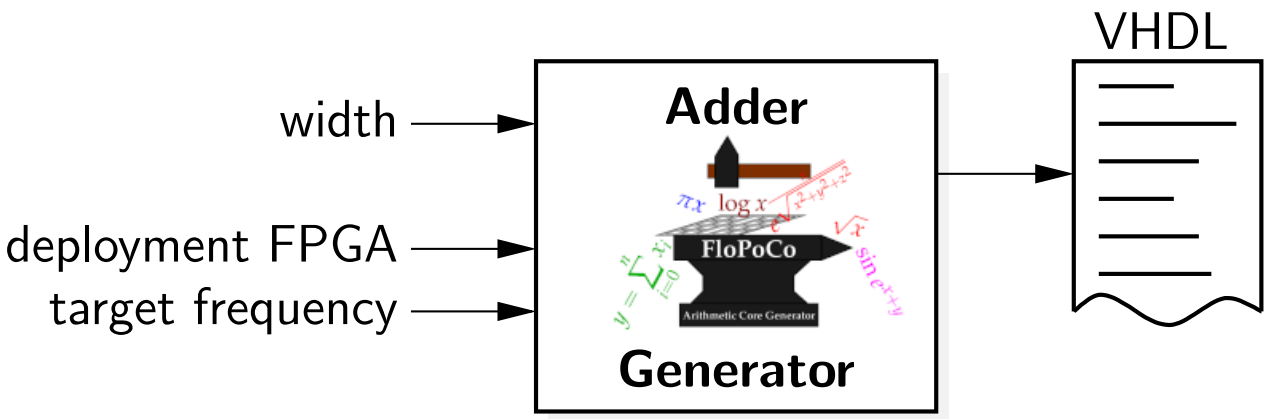
```
(...)
| | | | | |---Entity IntAdder_16_f325_slice_SRL_noBUFFER_uid10_Alternative:
| | | | | | Not pipelined
(...)
| | | | |---Entity IntAdder_52_f400_slice_SRL_noBUFFER_uid16_Classical:
| | | | | Pipeline depth = 1
| | | |---Entity IntCompressorTree_52_2_uid15:
| | | | Pipeline depth = 1
| | |---Entity IntTruncMultiplier_30_34_35_signed:
| | | Pipeline depth = 4
| | |---Entity IntAdder_42_f400_slice_SRL_BUFFER_uid17_Classical:
| | | Pipeline depth = 1
| |---Entity PolynomialEvaluator_d2:
| | Pipeline depth = 15
|---Entity FunctionEvaluator:
| Pipeline depth = 17
|---Entity IntAdder_48_f400_slice_SRL_noBUFFER_uid18_Classical:
| Pipeline depth = 2
|---Entity IntAdder_48_f400_slice_SRL_noBUFFER_uid19_Classical:
| Pipeline depth = 2
| | |---Entity IntAdder_85_f325_slice_SRL_noBUFFER_uid22_Classical:
| | | Pipeline depth = 1
| |---Entity IntCompressorTree_85_3_uid21:
| | Pipeline depth = 2
|---Entity IntMultiplier_47_48_uid20:
| Pipeline depth = 6
|---Entity IntAdder_57_f400_slice_SRL_noBUFFER_uid23_Classical:
| Pipeline depth = 2
|---Entity IntAdder_65_f400_slice_SRL_noBUFFER_uid24_Classical:
| Pipeline depth = 2
Entity FPExp_11_52_400:
Pipeline depth = 52
```

flopoco -frequency=200 FPExp 11 52 (double precision)

```
(...)
| | | | | |---Entity IntAdder_16_f200_slice_SRL_noBUFFER_uid10_Alternative:
| | | | | | Not pipelined
(...)
| | | | |---Entity IntAdder_52_f200_slice_SRL_noBUFFER_uid16_Alternative:
| | | | | Not pipelined
| | | |---Entity IntCompressorTree_52_2_uid15:
| | | | Not pipelined
| | |---Entity IntTruncMultiplier_30_34_35_signed:
| | | Pipeline depth = 2
| | |---Entity IntAdder_42_f200_slice_SRL_BUFFER_uid17_Classical:
| | | Not pipelined
| |---Entity PolynomialEvaluator_d2:
| | Pipeline depth = 7
|---Entity FunctionEvaluator:
| Pipeline depth = 9
|---Entity IntAdder_48_f200_slice_SRL_noBUFFER_uid18_Alternative:
| Pipeline depth = 1
|---Entity IntAdder_48_f200_slice_SRL_noBUFFER_uid19_Alternative:
| Pipeline depth = 1
| | |---Entity IntAdder_85_f200_slice_SRL_noBUFFER_uid22_Alternative:
| | | Pipeline depth = 1
| |---Entity IntCompressorTree_85_3_uid21:
| | Pipeline depth = 1
|---Entity IntMultiplier_47_48_uid20:
| Pipeline depth = 5
|---Entity IntAdder_57_f200_slice_SRL_noBUFFER_uid23_Alternative:
| Pipeline depth = 1
|---Entity IntAdder_65_f200_slice_SRL_noBUFFER_uid24_Alternative:
| Pipeline depth = 1
Entity FPExp_11_52_200:
Pipeline depth = 23
```

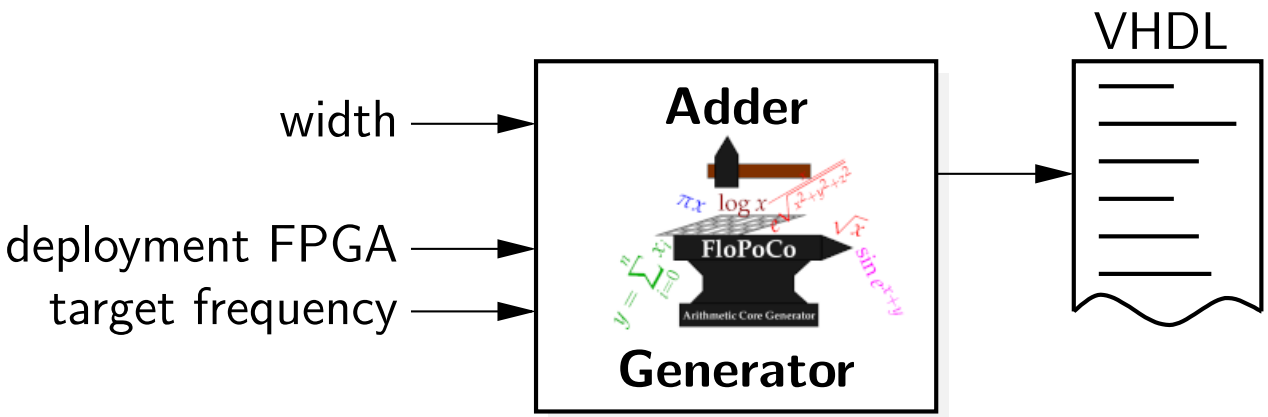
This article

How to generate "best" **addition operator** from user specifications



This article

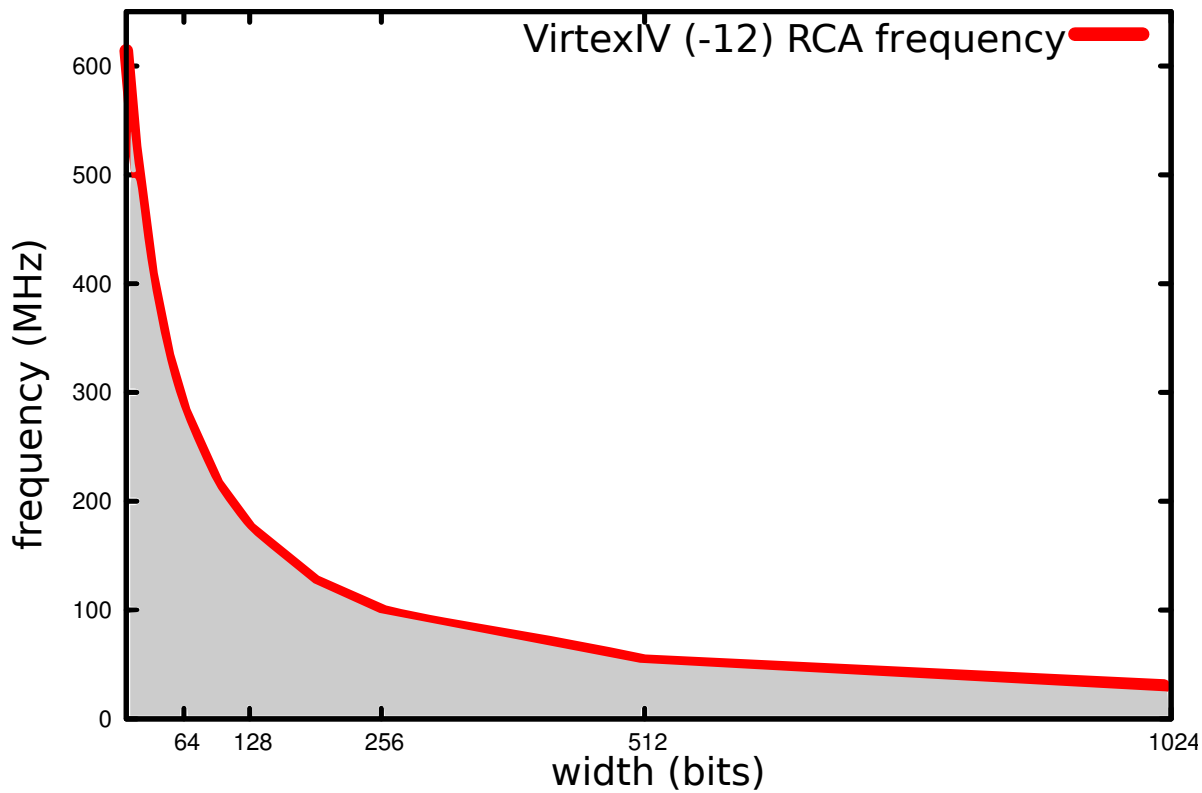
How to generate "best" **addition operator** from user specifications



- elementary block **resource estimation**
- **selection** among possible implementations

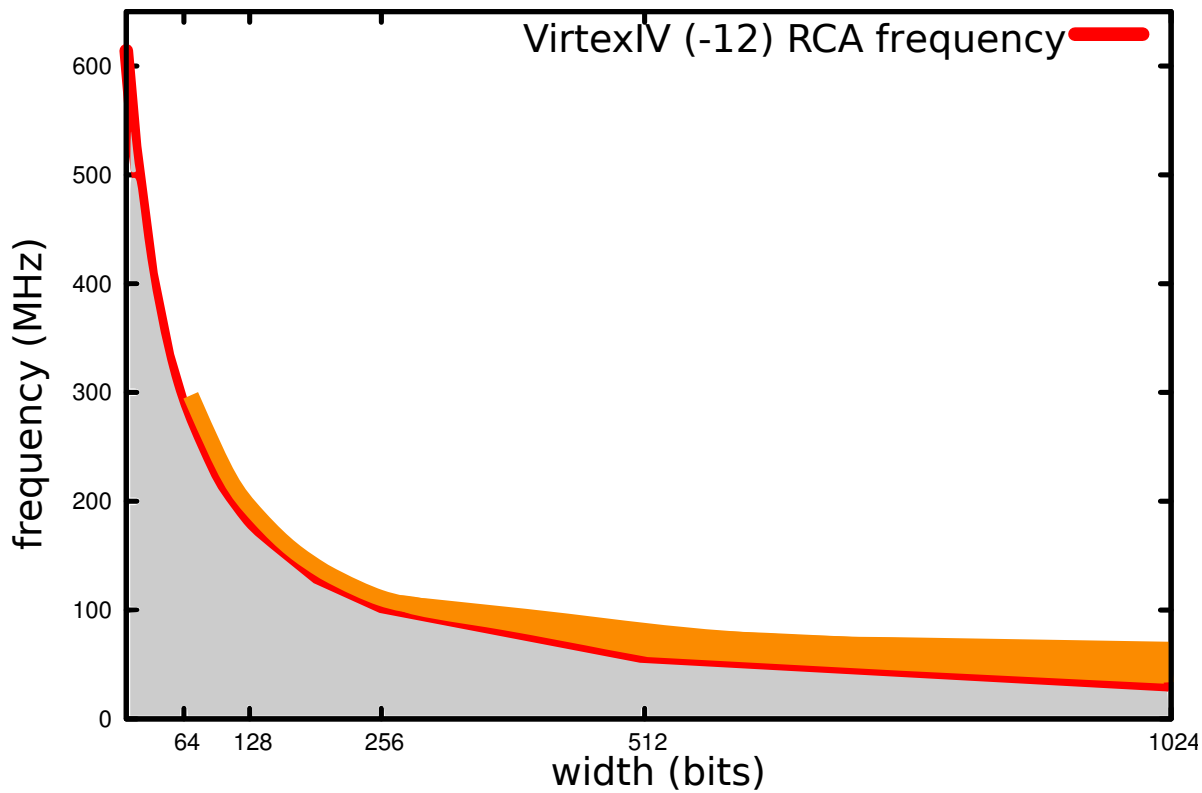
What?

```
signal R, X, Y: std_logic_vector(w-1 downto 0);
signal Cin: std_logic;
begin
...
R <= X + Y + Cin;
...
end architecture;
```



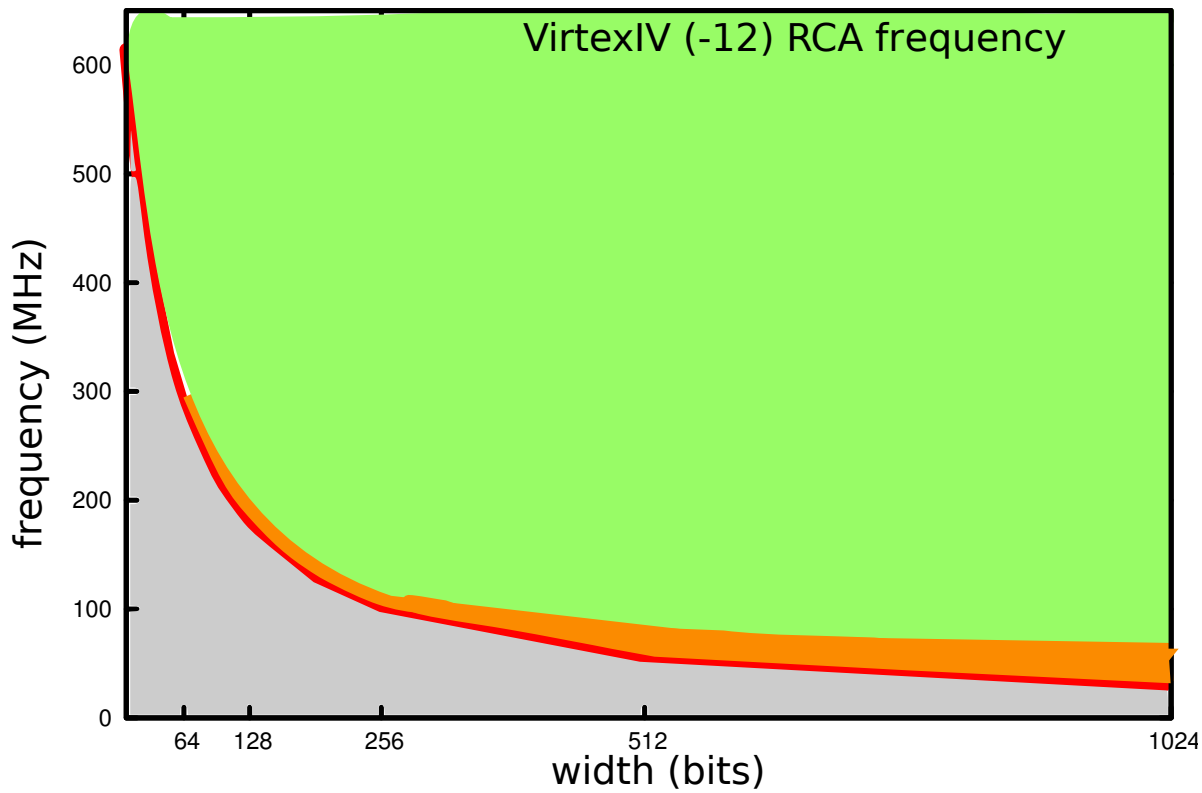
What?

```
signal R, X, Y: std_logic_vector(w-1 downto 0);
signal Cin: std_logic;
begin
...
1: fast_but_large_adder port map(X, Y, Cin, R); --pipeline depth=0
...
end architecture;
```



What?

```
signal R, X, Y: std_logic_vector(w-1 downto 0);
signal Cin: std_logic;
begin
...
l: fast_pipelined_adder port map(X, Y, Cin, R); --pipeline depth=X
...
end architecture;
```



Literature

- large number of VLSI studies [Mul89, EL04, DP96, US93]
  Q: Why not reuse them?

Literature

- large number of VLSI studies [Mul89, EL04, DP96, US93]
  Q: Why not reuse them?
  - most hypotheses false for FPGA
  - new architecture $\rightarrow$ **new rules**
  - some ideas can still be recycled

Literature

- large number of VLSI studies [Mul89, EL04, DP96, US93]
  Q: Why not reuse them?
  - most hypotheses false for FPGA
  - new architecture $\rightarrow$ **new rules**
  - some ideas can still be recycled
- Q: What about FPGA literature?
  - [XY98]: low-latency unpipelined adders
  - [MJE97]: Compression tree using a (naive) final pipelined adder
  - Other more recent papers on accumulation or compression trees

---

[XY98] Shanzhen Xing and William W.h. Yu.
FPGA Adders: Performance Evaluation and Optimal Design.

[MJE97] Peiro Marcos Martinez, Valls Javier, and Boemo Eduardo.
On the design of FPGA-based Multioperand Pipeline Adders.

Literature

- large number of VLSI studies [Mul89, EL04, DP96, US93]
  Q: Why not reuse them?
  - most hypotheses false for FPGA
  - new architecture $\rightarrow$ **new rules**
  - some ideas can still be recycled
- Q: What about FPGA literature?
  - [XY98]: low-latency unpipelined adders
  - [MJE97]: Compression tree using a (naive) final pipelined adder
  - Other more recent papers on accumulation or compression trees

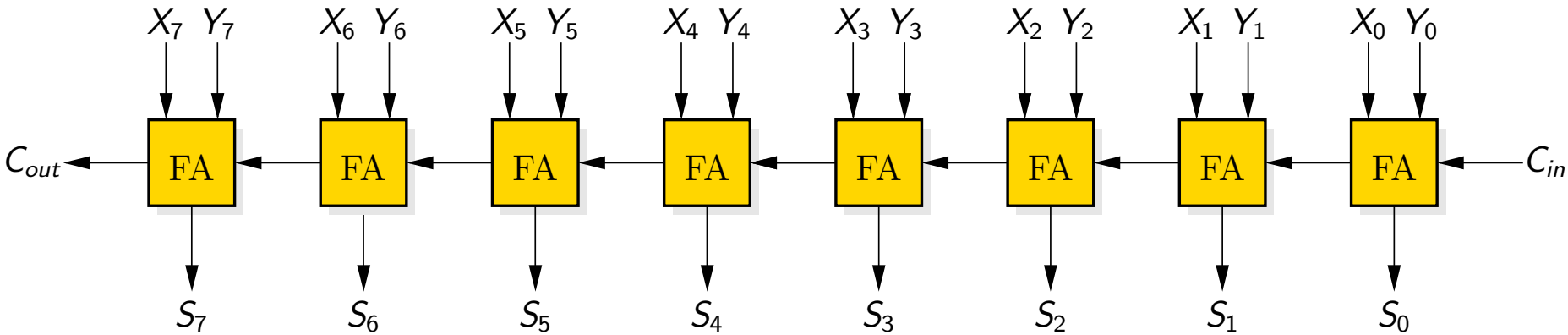
But no comprehensive study on **pipelined FPGA addition**

[XY98] Shanzhen Xing and William W.h. Yu.
FPGA Adders: Performance Evaluation and Optimal Design.

[MJE97] Peiro Marcos Martinez, Valls Javier, and Boemo Eduardo.
On the design of FPGA-based Multioperand Pipeline Adders.

Ripple-Carry-Adder

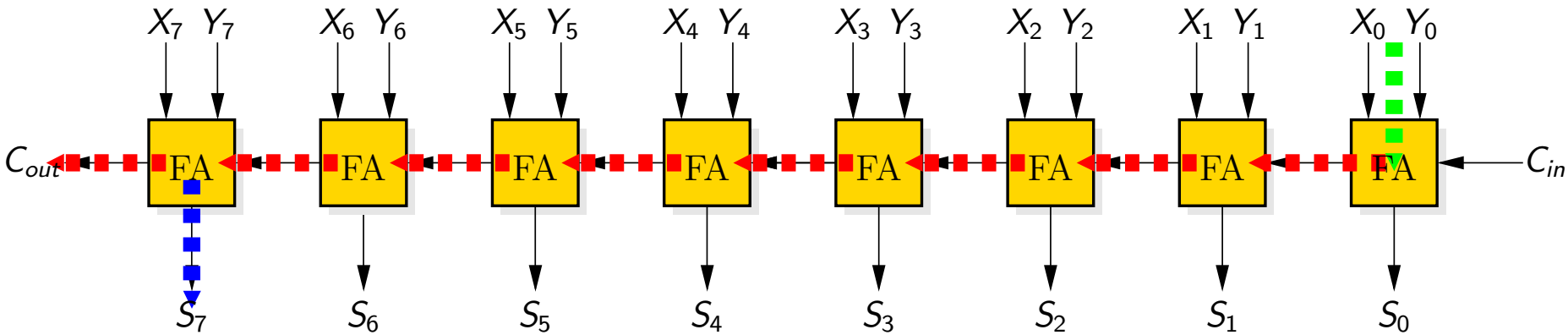
```
signal R, X, Y: std_logic_vector(w-1 downto 0);
signal Cin: std_logic;
begin
...
R <= X + Y + Cin;
...
end architecture;
```



- w-bit adder: connect w FA in series

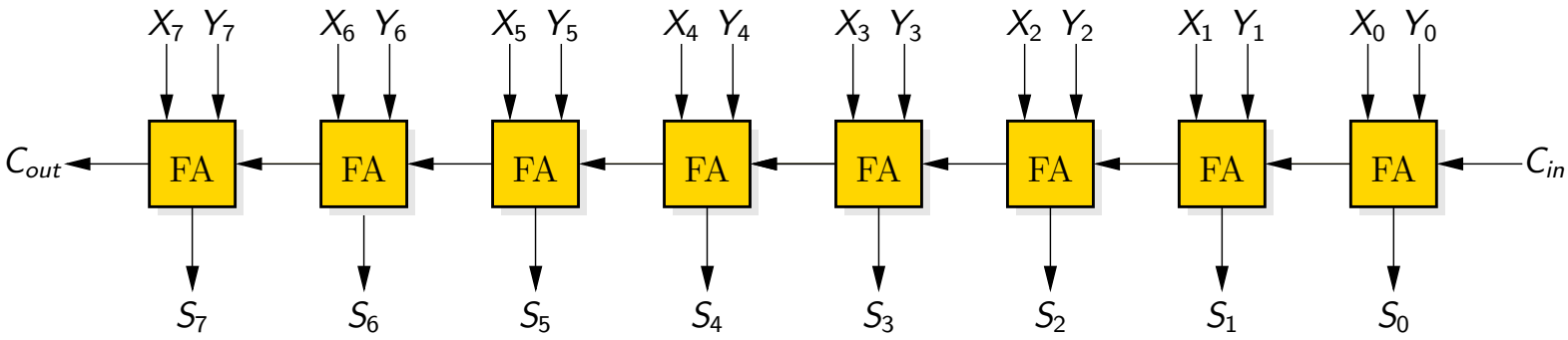
Ripple-Carry-Adder

```
signal R, X, Y: std_logic_vector(w-1 downto 0);
signal Cin: std_logic;
begin
...
R <= X + Y + Cin;
...
end architecture;
```

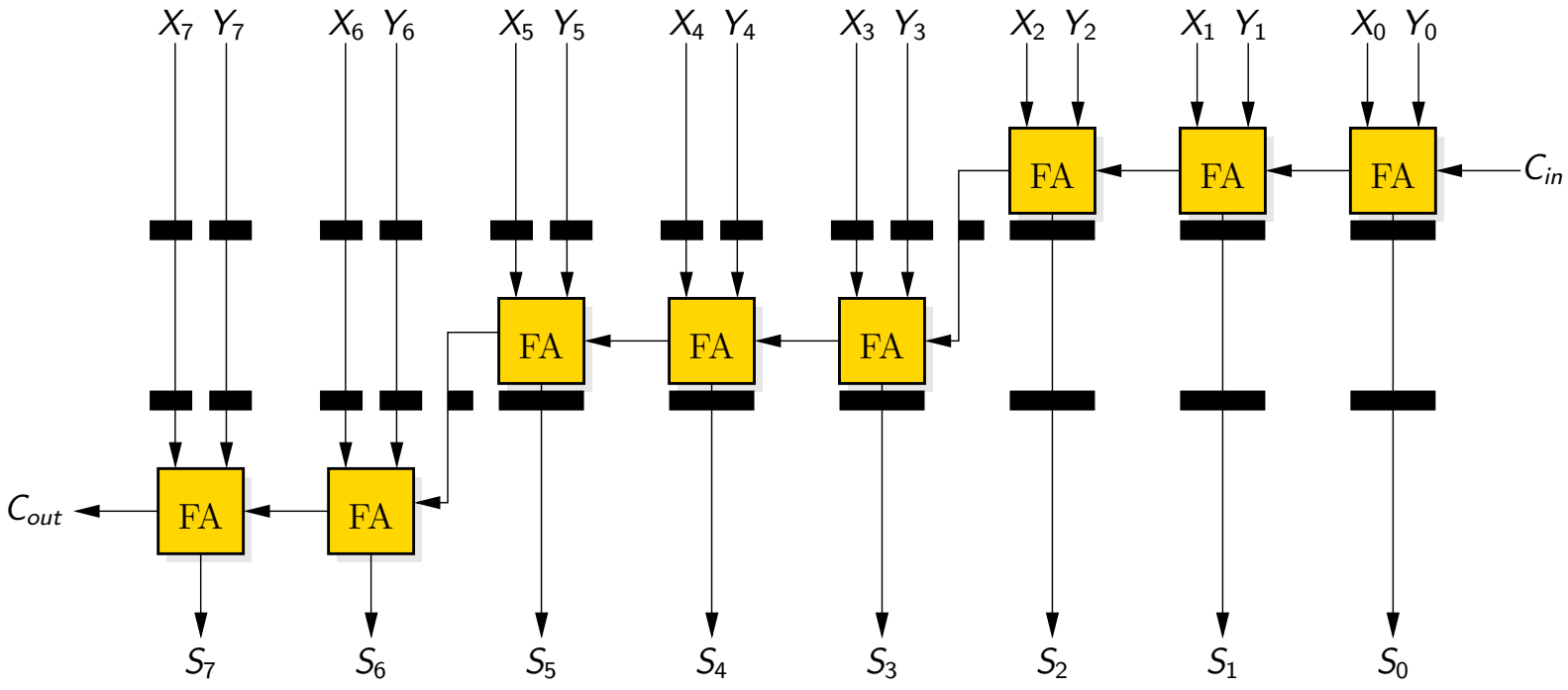


- w-bit adder: connect w FA in series
- $T = delay_1 + (w-1)delay_{C_{out}} + max(delay_{C_{out}}, delay_S)$

Pipeline a w-bit addition: How To ?

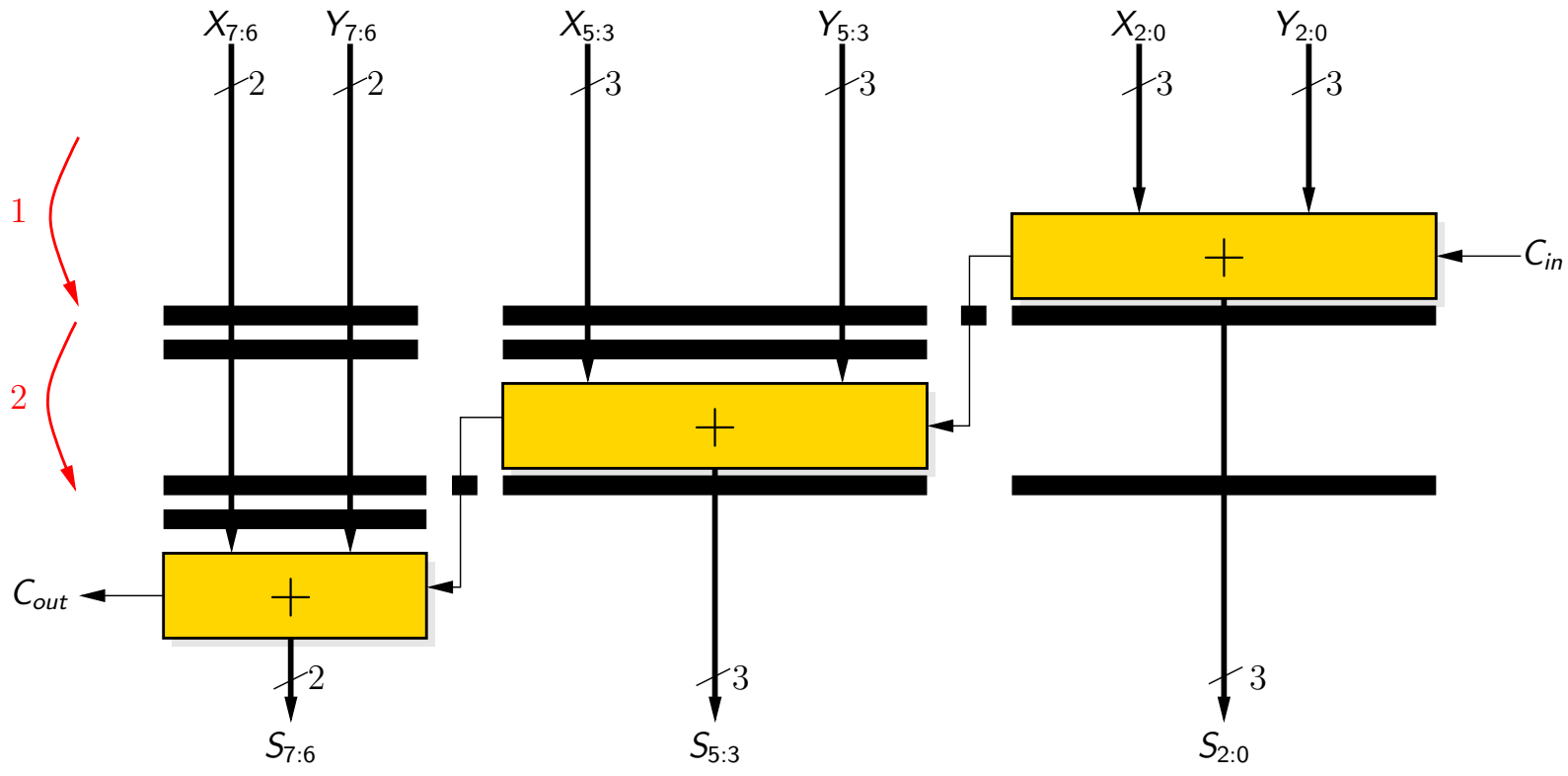


Pipeline a w-bit addition: How To ?



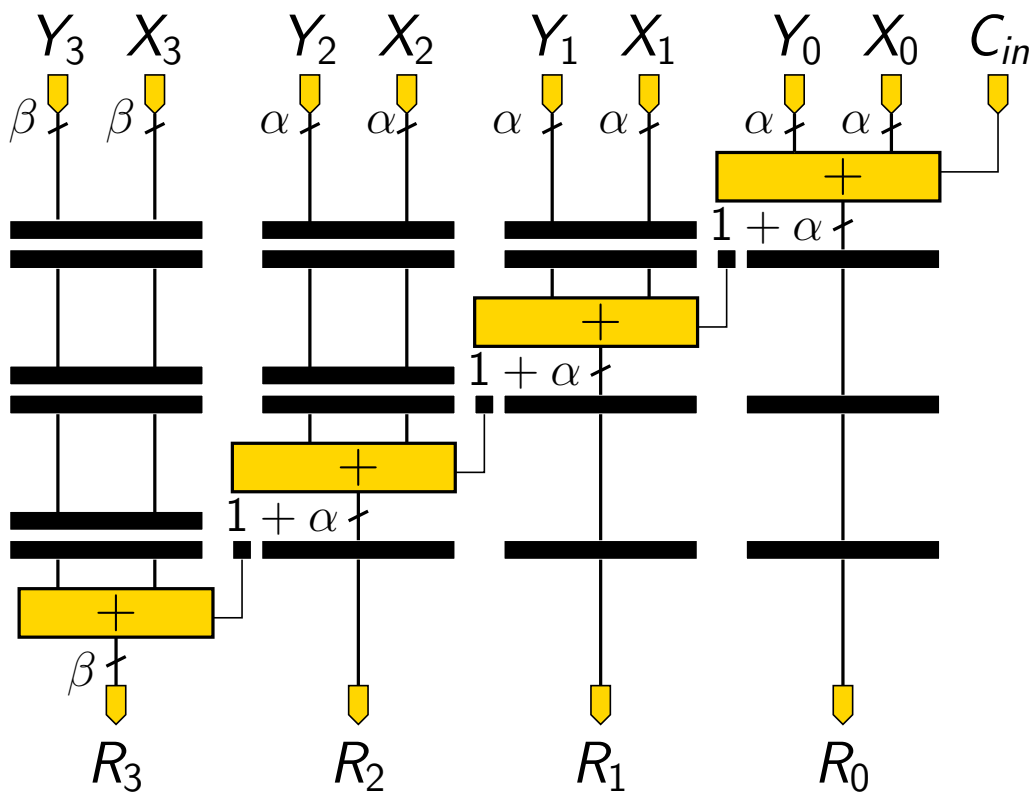
- **break carry-propagation path** such that $T_{RCA_\alpha} < 1/f$
- $\alpha = \frac{1/f - delay_1 - max(delay_{C_{out}}, delay_S)}{delay_{C_{out}}} + 1$
- synchronize I/O using registers $\rightarrow$ **register overhead**

Pipeline a w-bit addition: How To ?



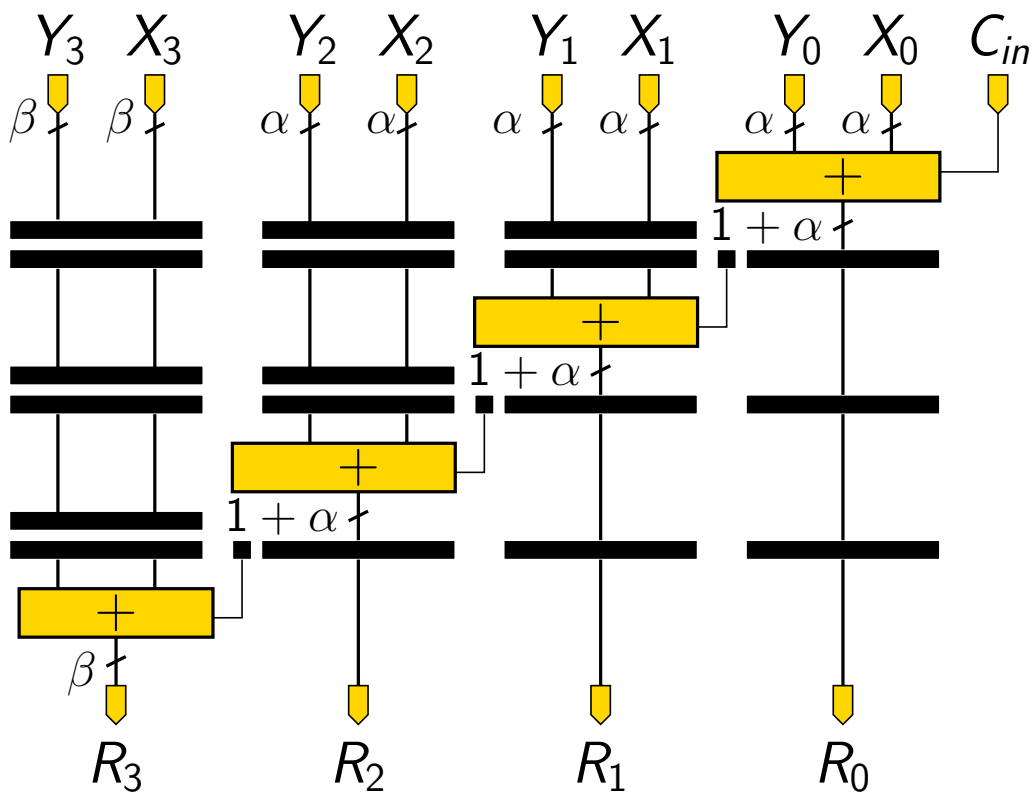
- **break carry-propagation path** such that $T_{RCA_\alpha} < 1/f$
- $\alpha = \frac{1/f - delay_1 - max(delay_{C_{out}}, delay_S)}{delay_{C_{out}}} + 1$
- synchronize I/O using registers $\rightarrow$ **register overhead**

Classical Pipelined Addition Architecture



- encountered in literature [EL04, MJE97]
- named also **traditional**

Classical Pipelined Addition Architecture

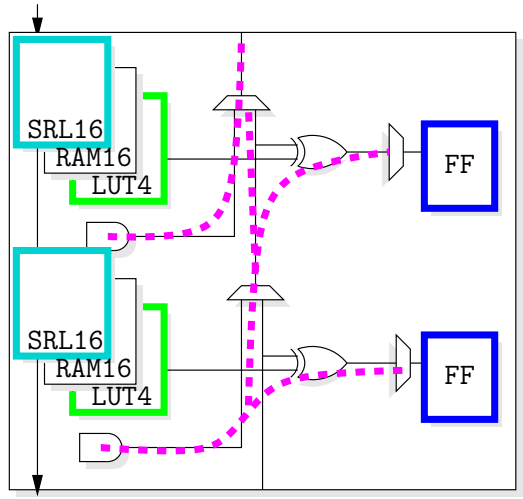


- encountered in literature [EL04, MJE97]
- named also **traditional**

How many resources does it take ?

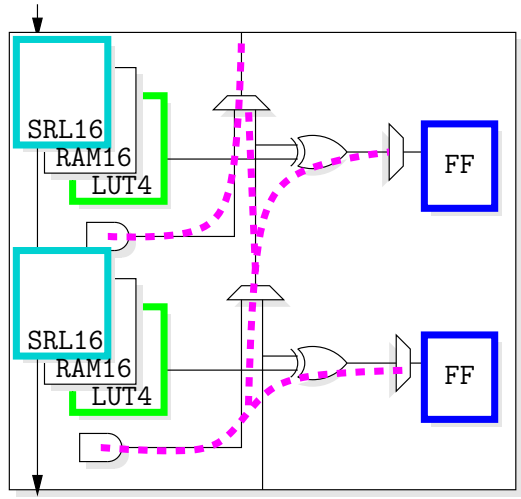
Resource Estimation Techniques

heterogeneous FPGA: LUTs, Registers, RAM, DSP-blocks etc



Resource Estimation Techniques

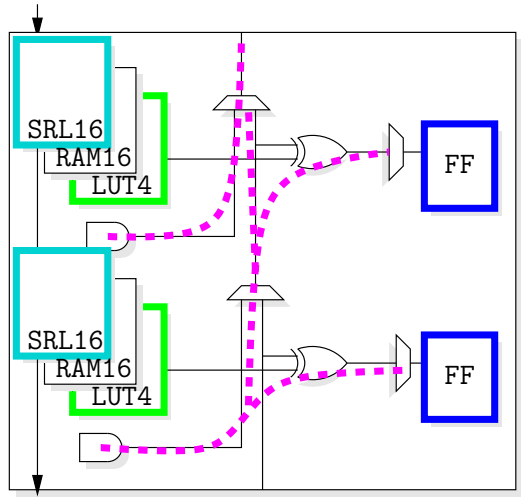
heterogeneous FPGA: LUTs, Registers, RAM, DSP-blocks etc



- Q: What to count ?
  - Depends on the designer needs
  - In our case we prefer to count the **atomic entities**
    - Slices on VirtexII, Spartan3, Virtex4 (should be half-slices)
    - LUTs, Registers on Virtex5

Resource Estimation Techniques

heterogeneous FPGA: LUTs, Registers, RAM, DSP-blocks etc

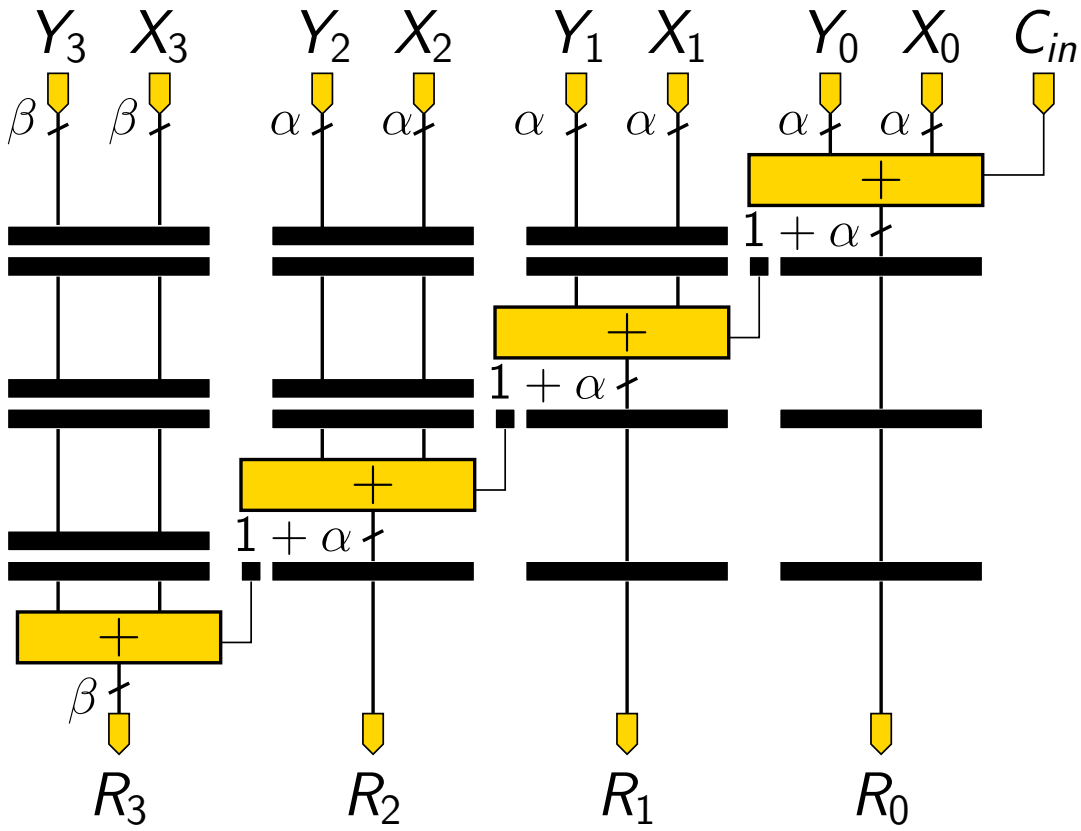


- Q: What to count ?
  - Depends on the designer needs
  - In our case we prefer to count the **atomic entities**
    - Slices on VirtexII, Spartan3, Virtex4 (should be half-slices)
    - LUTs, Registers on Virtex5

Case Study:

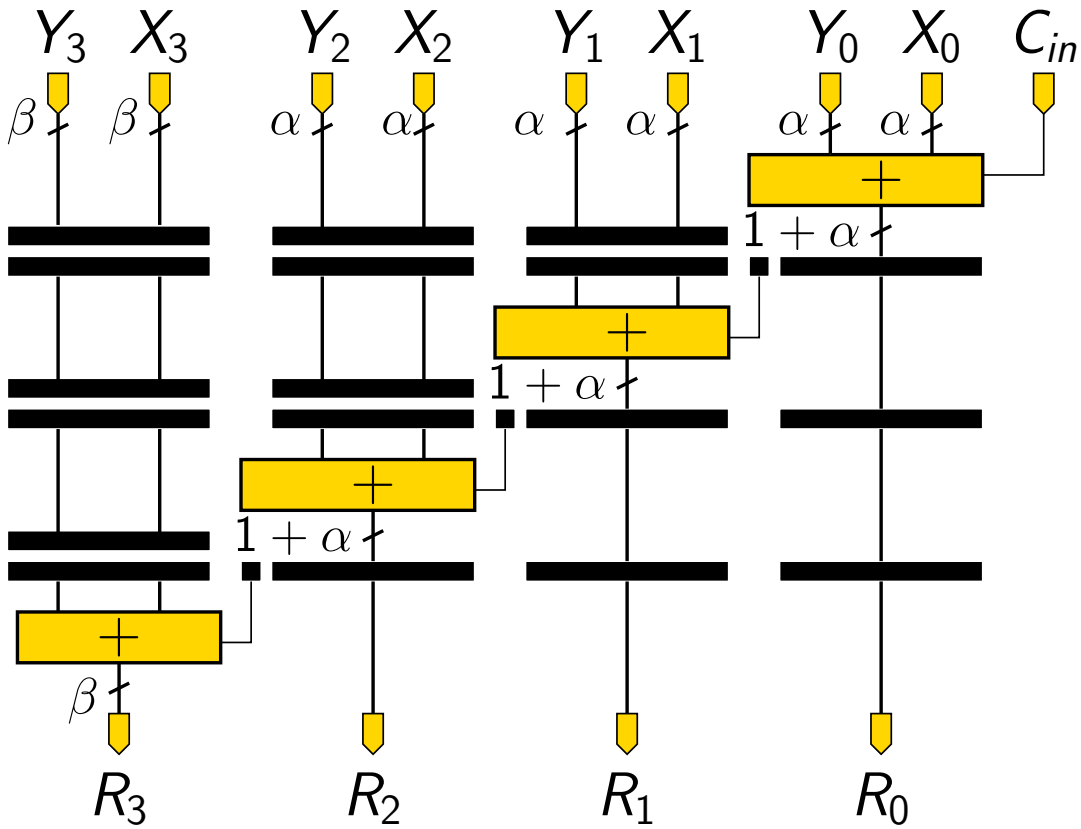
We evaluate the resources required by the classical architecture

Case Study: Classical Architecture



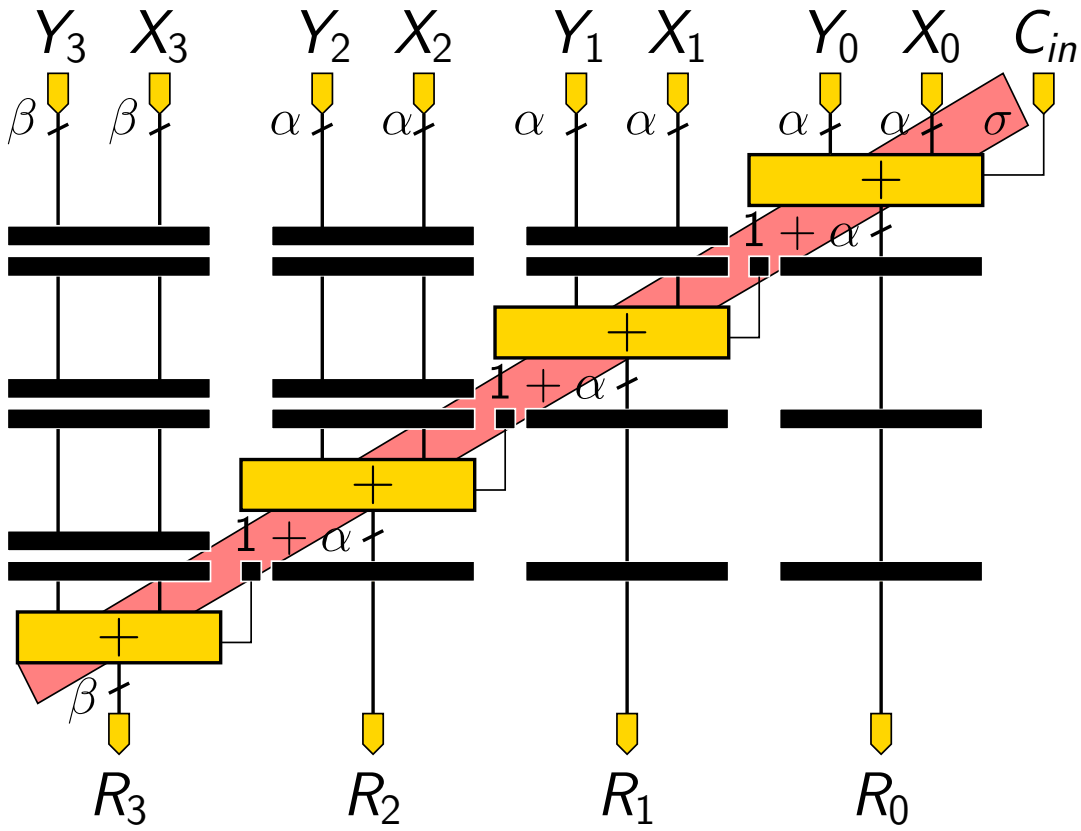
- k chunks, w-bit addition: $w = (k-1)\alpha + \beta$

Case Study: Classical Architecture



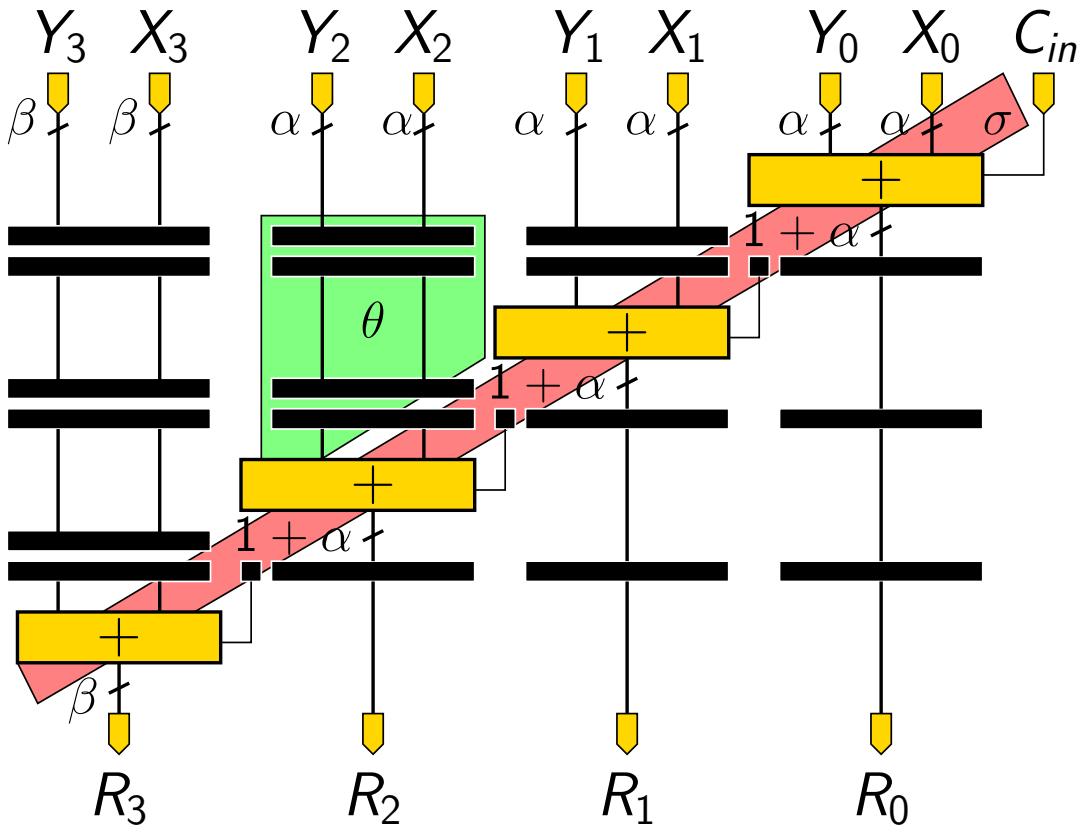
- k chunks, w-bit addition: $w = (k-1)\alpha + \beta$
- $L =$

Case Study: Classical Architecture



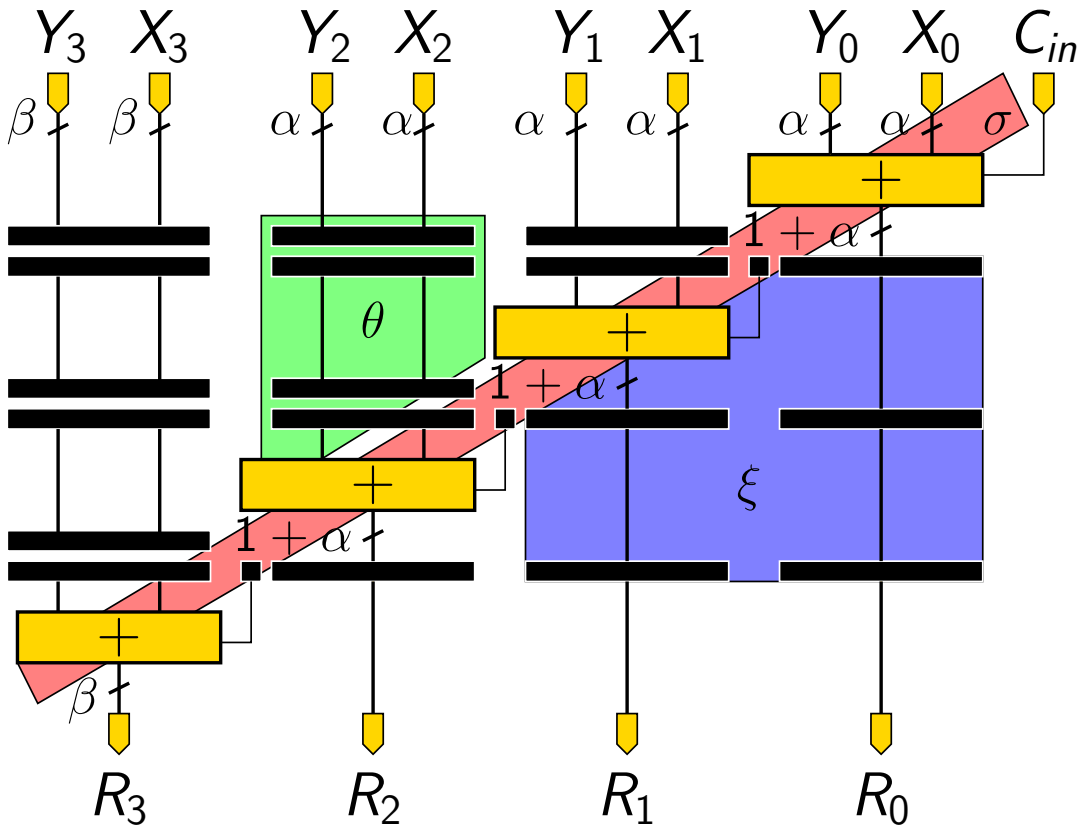
- k chunks, w-bit addition: $w = (k-1)\alpha + \beta$
- $L = w +$

Case Study: Classical Architecture



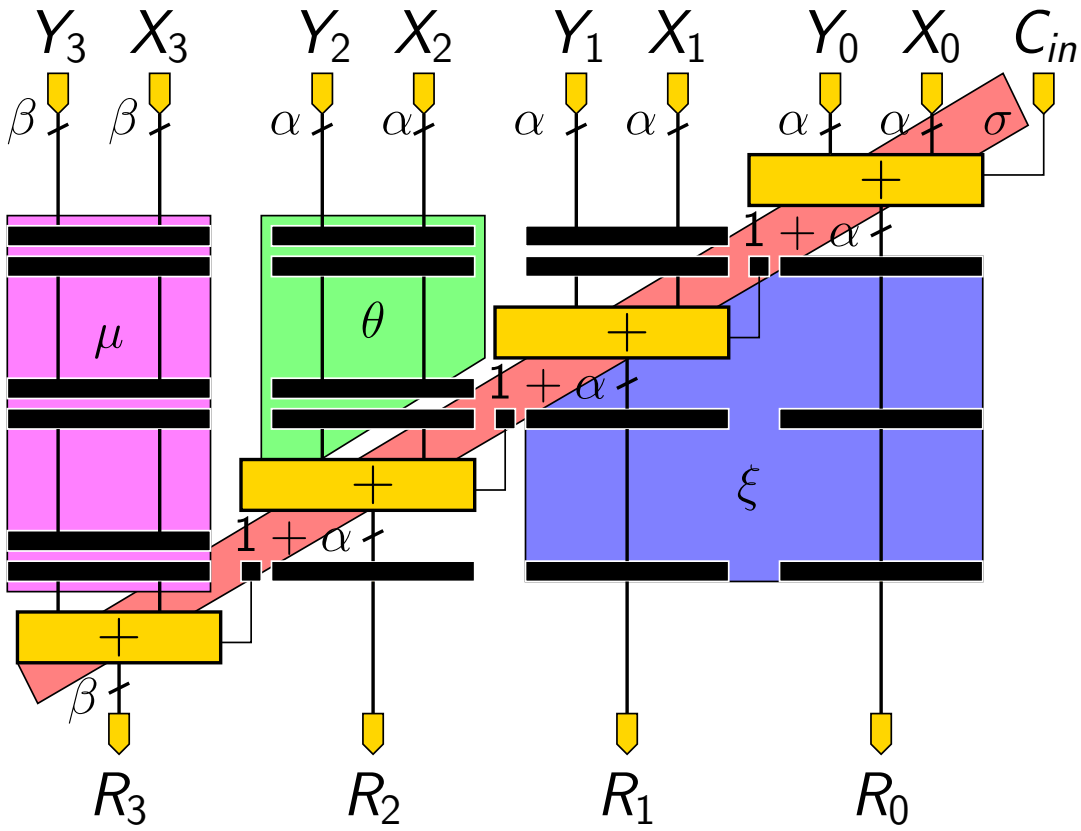
- k chunks, w-bit addition: $w = (k-1)\alpha + \beta$
- $L = w + 2(k-3)\alpha +$

Case Study: Classical Architecture



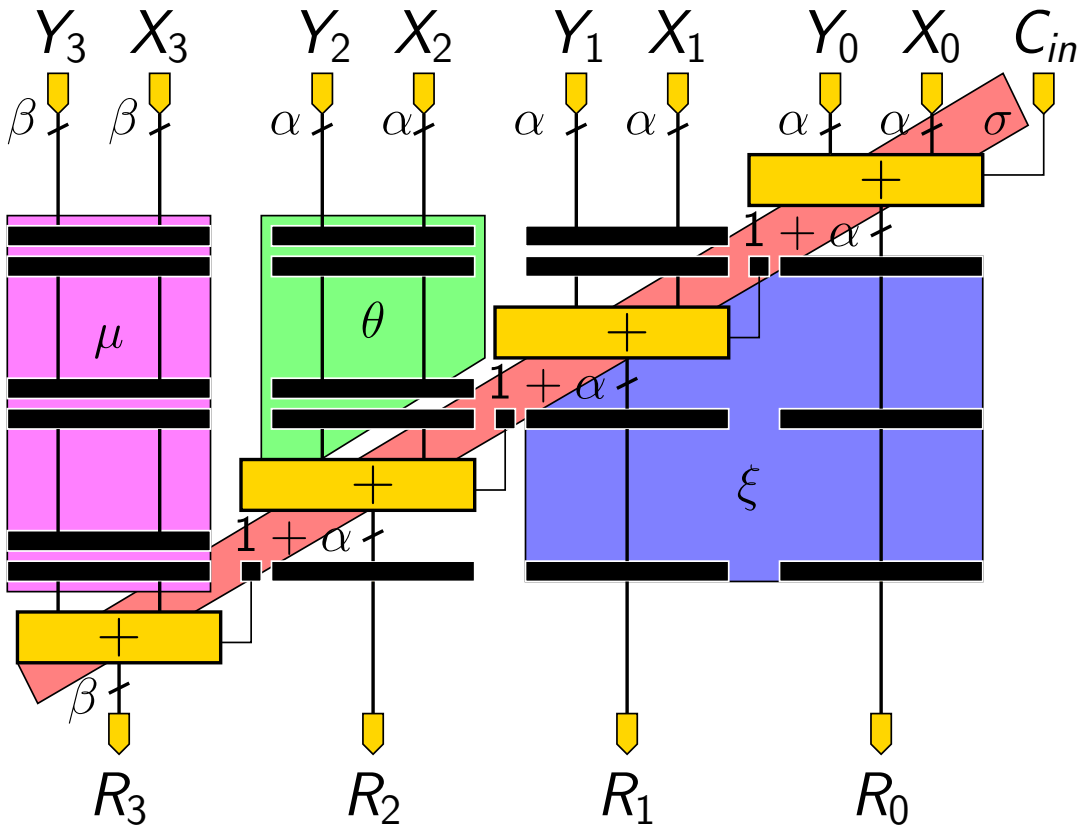
- k chunks, w-bit addition: $w = (k-1)\alpha + \beta$
- $L = w + 2(k-3)\alpha + (k-2)\alpha +$

Case Study: Classical Architecture



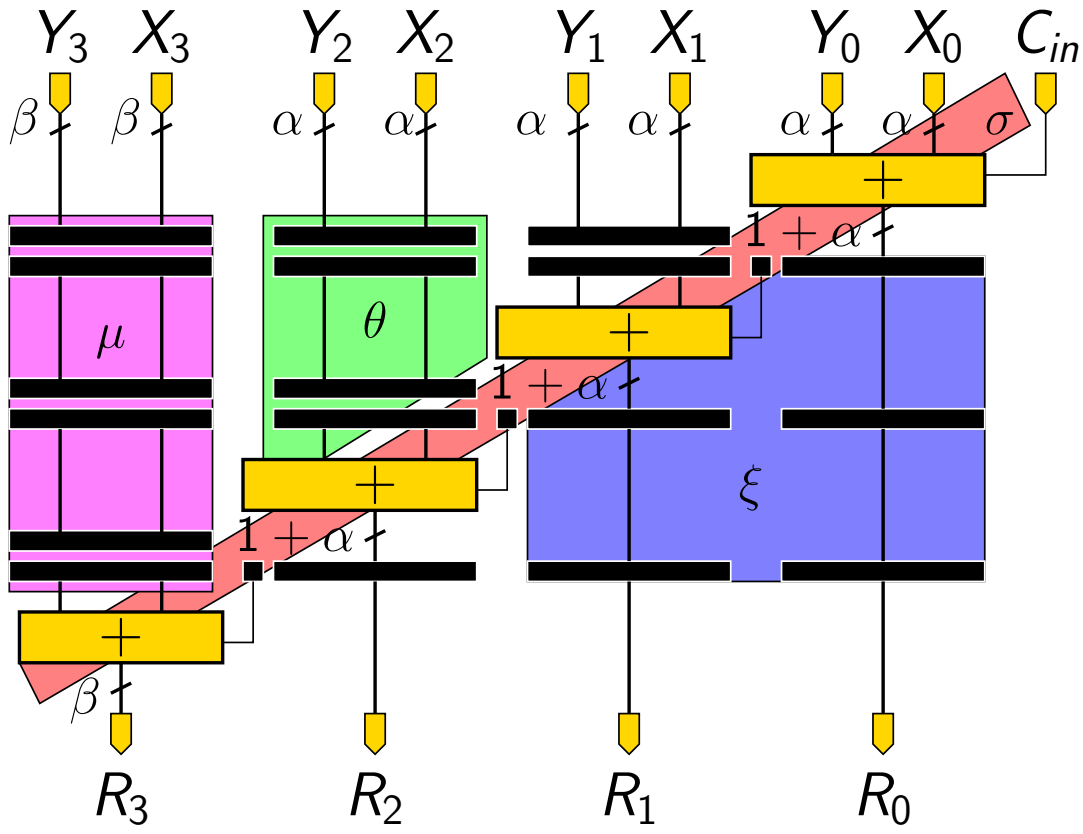
- k chunks, w-bit addition: $w = (k-1)\alpha + \beta$
- $L = w + 2(k-3)\alpha + (k-2)\alpha + 2\beta$

Case Study: Classical Architecture



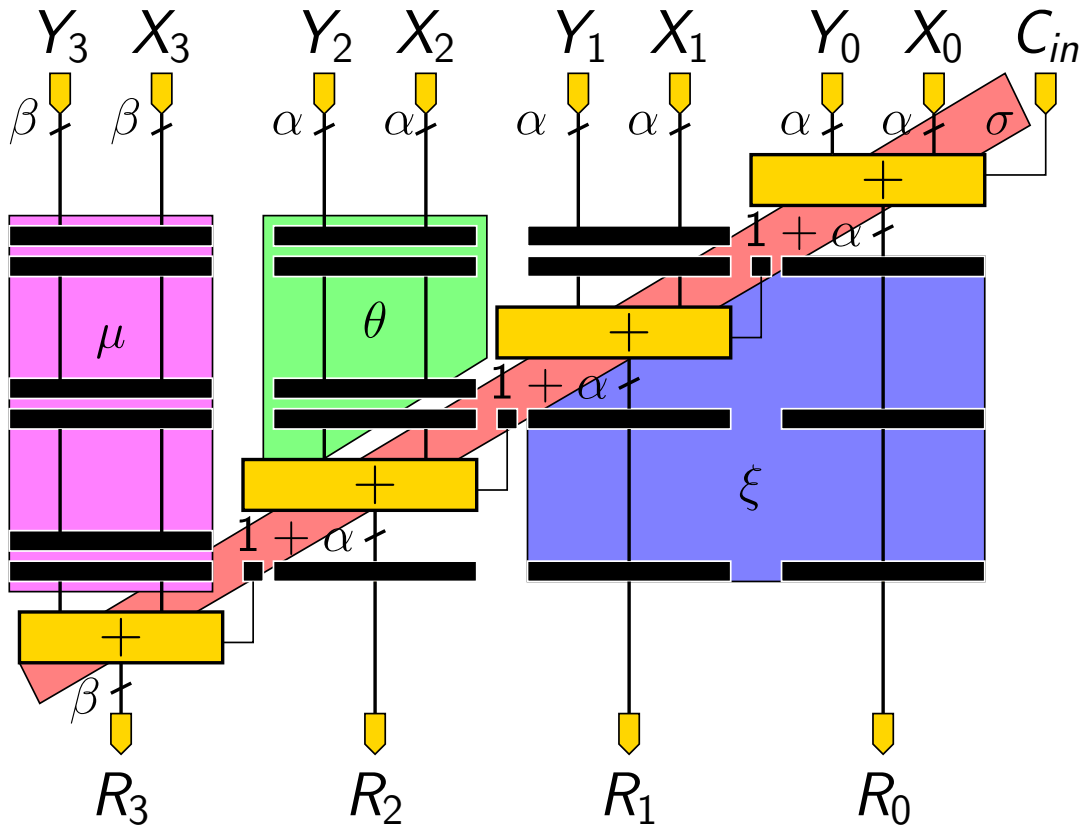
- k chunks, w-bit addition: $w = (k-1)\alpha + \beta$
- $L = w + 2(k-3)\alpha + (k-2)\alpha + 2\beta = w + (3k-8)\alpha + 2\beta$

Case Study: Classical Architecture



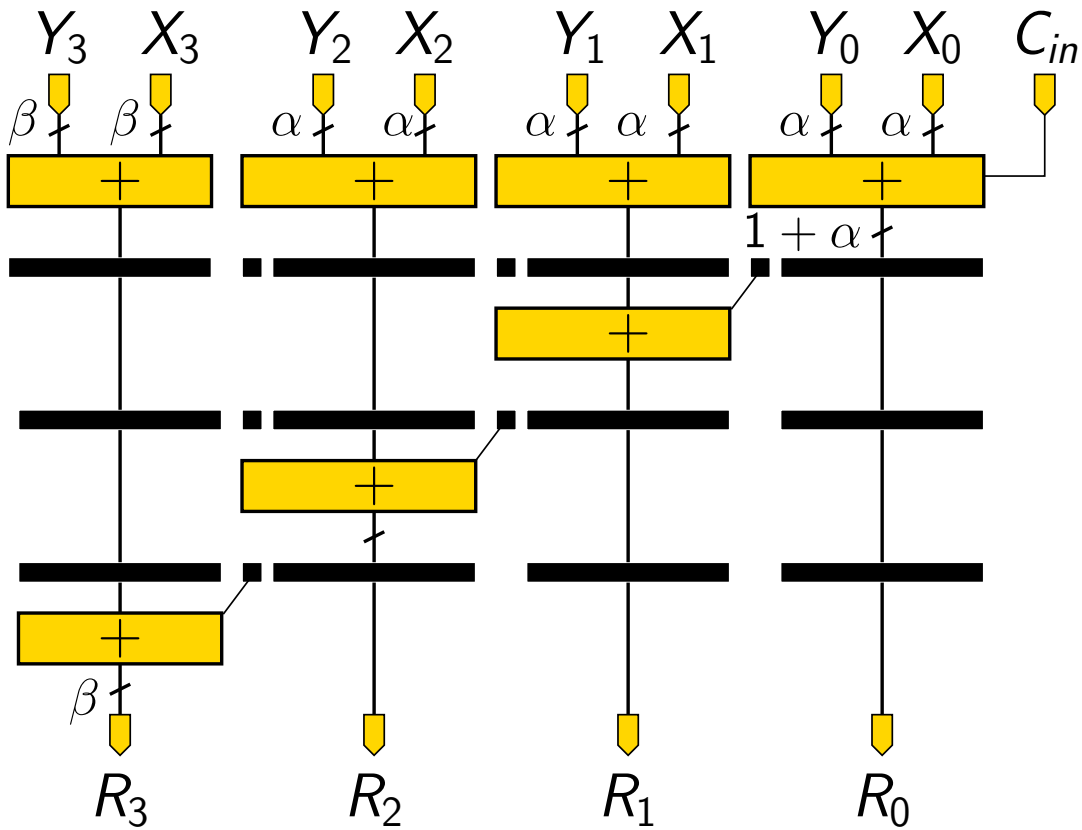
- k chunks, w-bit addition: $w = (k-1)\alpha + \beta$
- $L = w + 2(k-3)\alpha + (k-2)\alpha + 2\beta = w + (3k-8)\alpha + 2\beta$
- $R = (3k-8)\alpha + 2\beta + \alpha + 2\alpha + k - 1$

Case Study: Classical Architecture



- k chunks, w-bit addition: $w = (k-1)\alpha + \beta$
- $L = w + 2(k-3)\alpha + (k-2)\alpha + 2\beta = w + (3k-8)\alpha + 2\beta$
- $R = (3k-8)\alpha + 2\beta + \alpha + 2\alpha + k - 1$
- $2S = w + (3k-8)\alpha + 2\beta + 3\alpha + k - 1 - \alpha$

Alternative Architecture



- perform additions as soon as possible
- propagate carry-out bits for diagonal addition
- **lower LUT and register consumption** than first version
  Details and formulas in the paper

What If ...

1. operator latency is a problem

What If ...

1. operator latency is a problem
   - pipeline depth of previous architectures increase linearly with k
   - k increases when f or w increase

What If ...

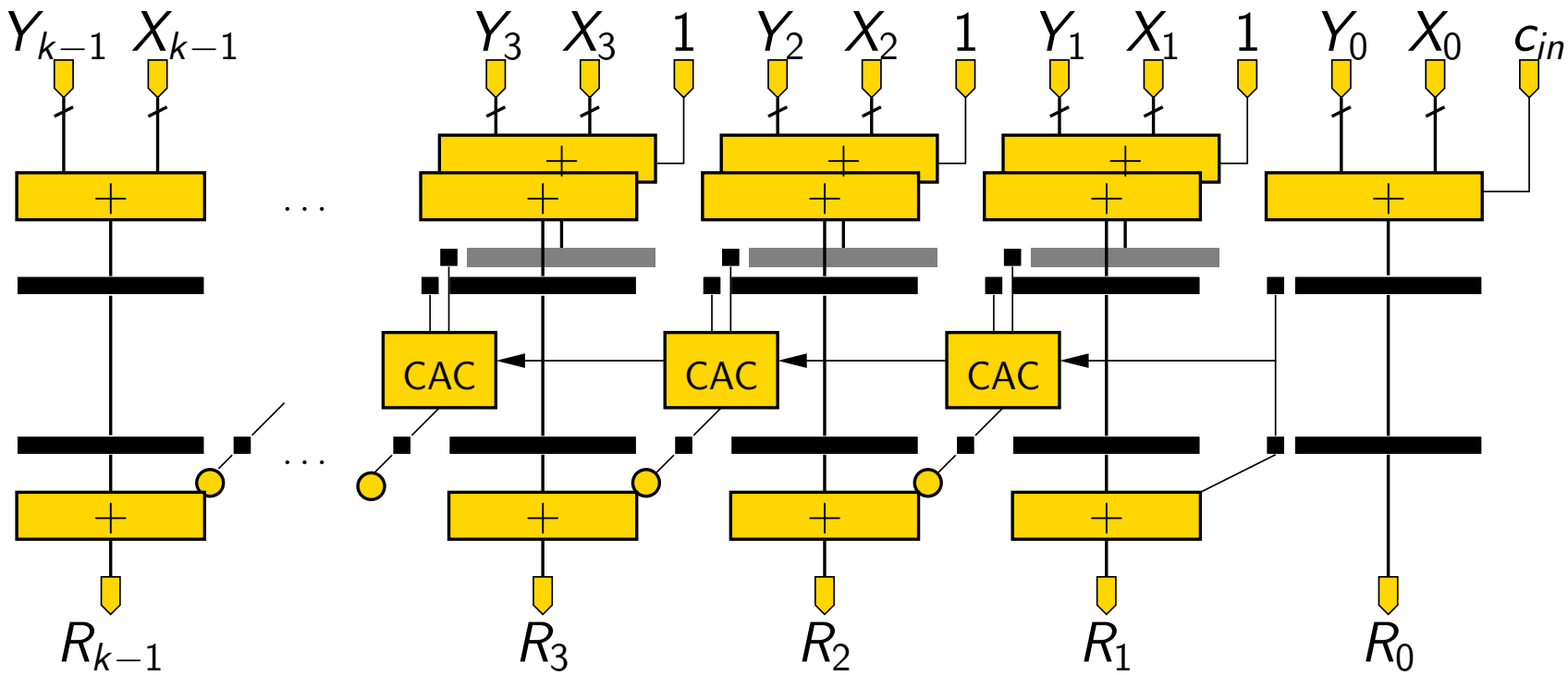
1. operator latency is a problem
   - pipeline depth of previous architectures increase linearly with k
   - k increases when f or w increase
   - 128-bit addition @ 400MHz, Virtex4 (-12) takes 3 cycles

What If ...

1. operator latency is a problem
   - pipeline depth of previous architectures increase linearly with k
   - k increases when f or w increase
   - 128-bit addition @ 400MHz, Virtex4 (-12) takes 3 cycles
2. no SRL are available
   - register count explodes, even for the alternative architecture

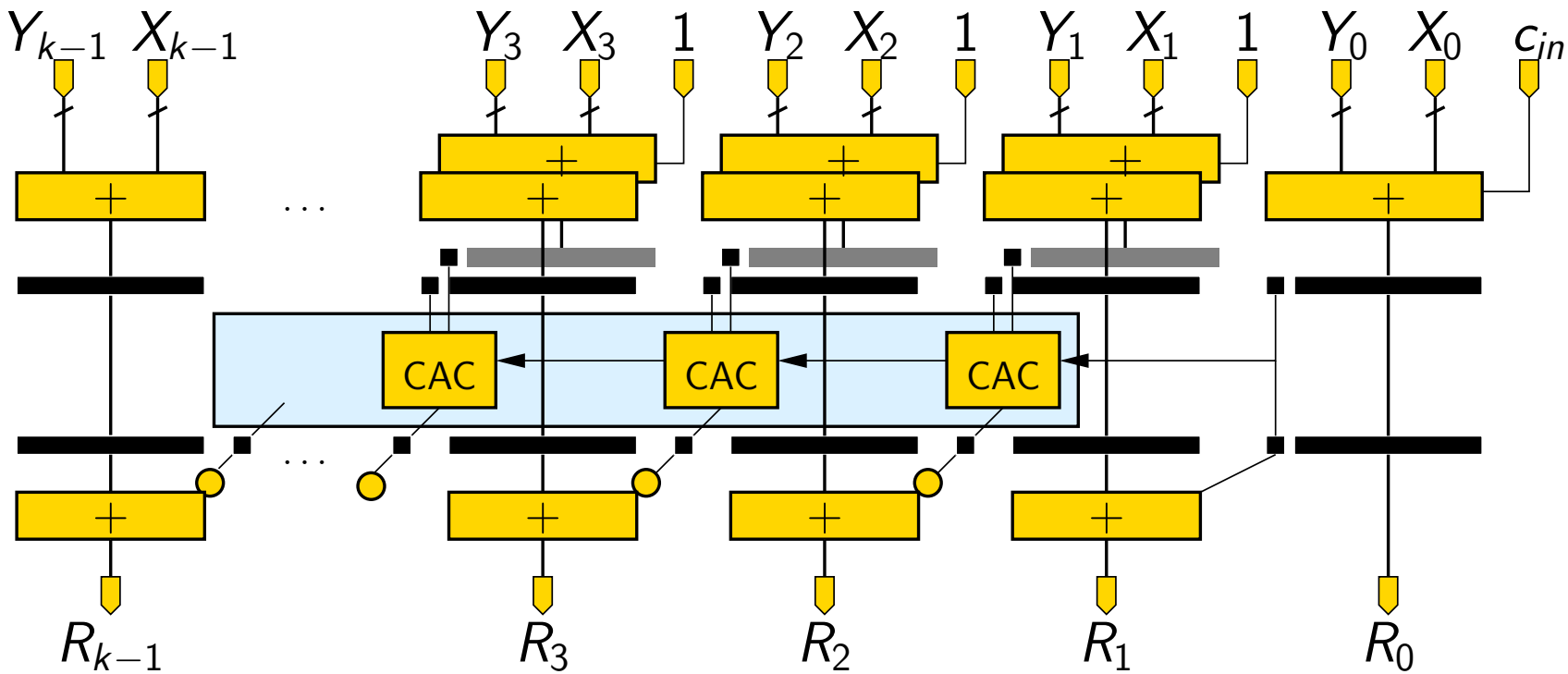
Short-Latency Architecture

- basic idea:
  - perform a **carry-select addition**



Short-Latency Architecture

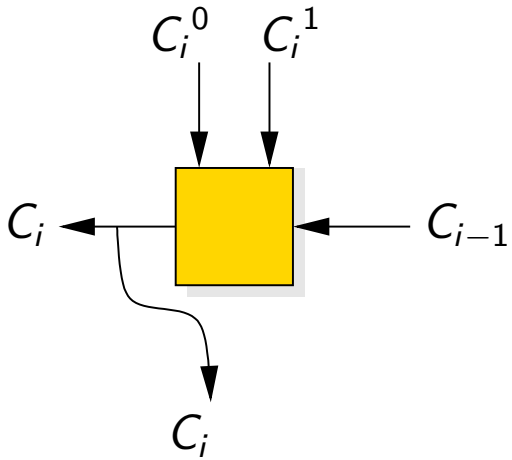
- basic idea:
  - perform a **carry-select addition**
  - use dedicated fast-carry lines to speed-up carry-bit computation



Short-Latency Architecture: Carry-Adder-Cell

- knowing C_{i-1}, ${C_i}^0$ and ${C_i}^1$ we compute C_i

$$C_i = f(C_{i-1}, {C_i}^0, {C_i}^1)$$

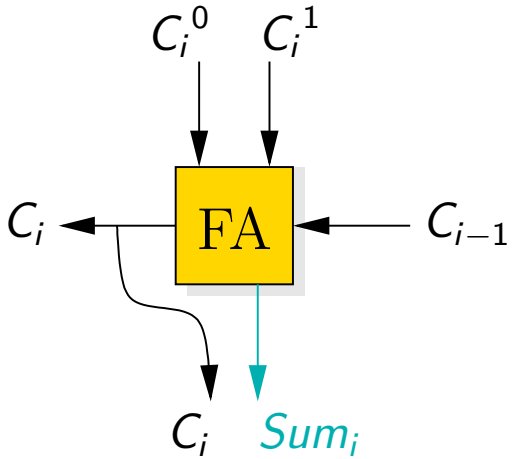


C_{i-1}	${C_i}^0$	${C_i}^1$	C_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Short-Latency Architecture: Carry-Adder-Cell

- knowing C_{i-1}, ${C_i}^0$ and ${C_i}^1$ we compute C_i

$$C_i : Sum_i = C_{i-1} + {C_i}^0 + {C_i}^1$$

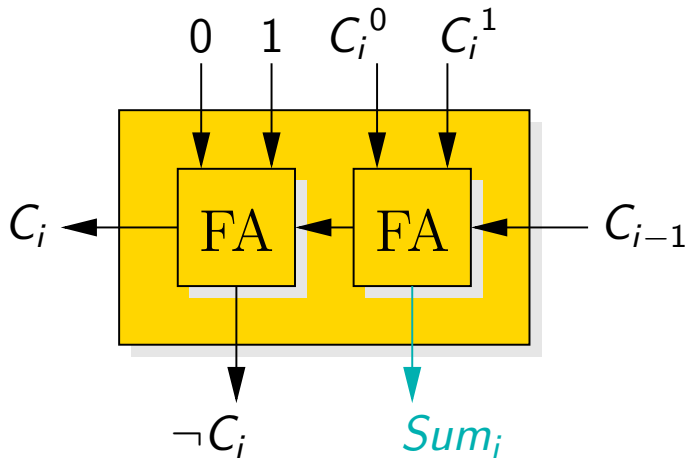


C_{i-1}	${C_i}^0$	${C_i}^1$	C_i	Sum_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Short-Latency Architecture: Carry-Adder-Cell

- knowing C_{i-1}, ${C_i}^0$ and ${C_i}^1$ we compute C_i

$$C_i : \neg C_i : Sum_i = C_{i-1} + {C_i}^0 + {C_i}^1 + 2$$



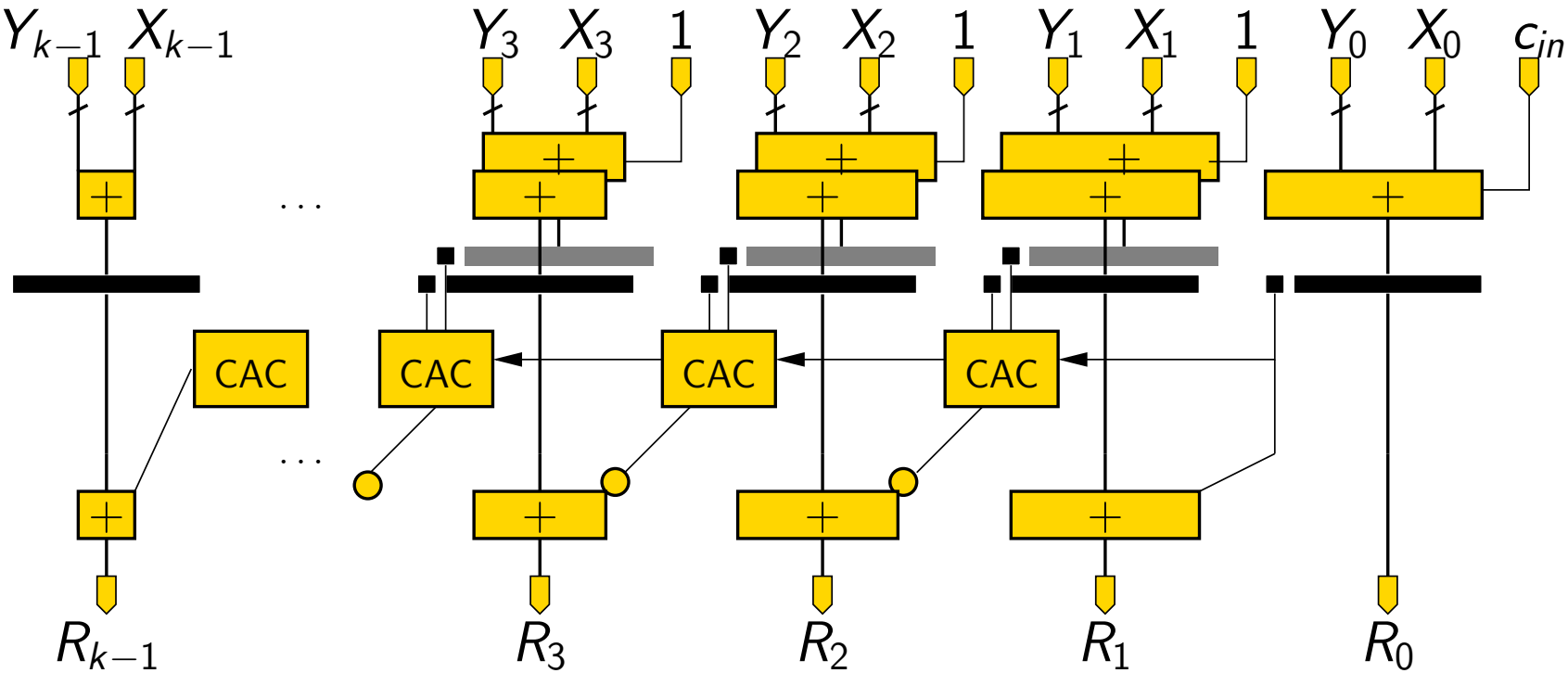
C_{i-1}	${C_i}^0$	${C_i}^1$	C_i	$\neg C_i$	Sum_i
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	1

- portable VHDL that uses fast-carry lines

```
op0 <= "0" & C4_0 & "0" & C3_0 & "0" & C2_0 & "0" & C1_0;
op1 <= "1" & C4_1 & "1" & C3_1 & "1" & C2_1 & "1" & C1_1;
--perform the short carry additions
rawCarrySum <= op0_d1 + op1_d1 + C0;
```

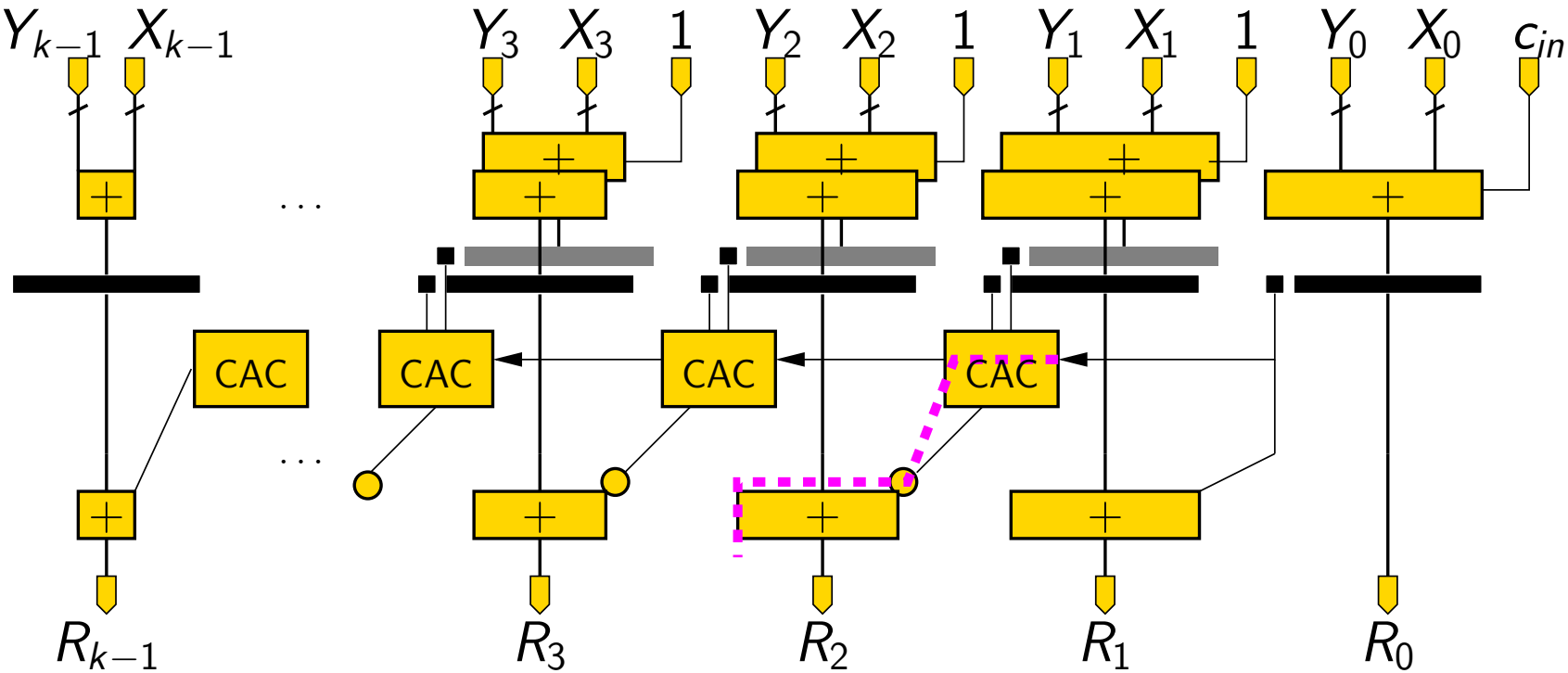
Short-Latency Architecture: Optimization

- for some values of (w, f) discard second register level



Short-Latency Architecture: Optimization

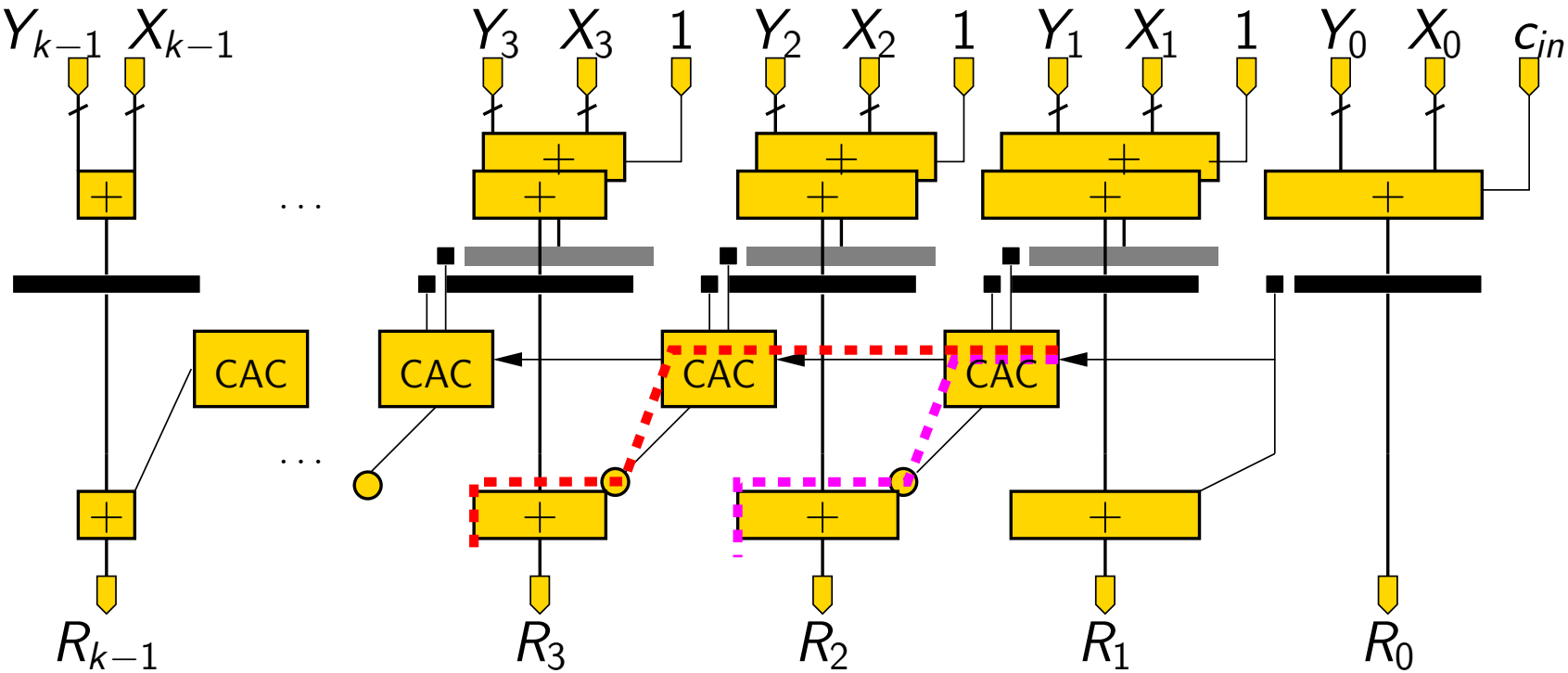
- for some values of (w, f) discard second register level



- greedy algorithm for determining chunk sizes
  - ensure for each chunk: $delay_{CAC} + delay_{routing} + delay_{sum} < 1/f$

Short-Latency Architecture: Optimization

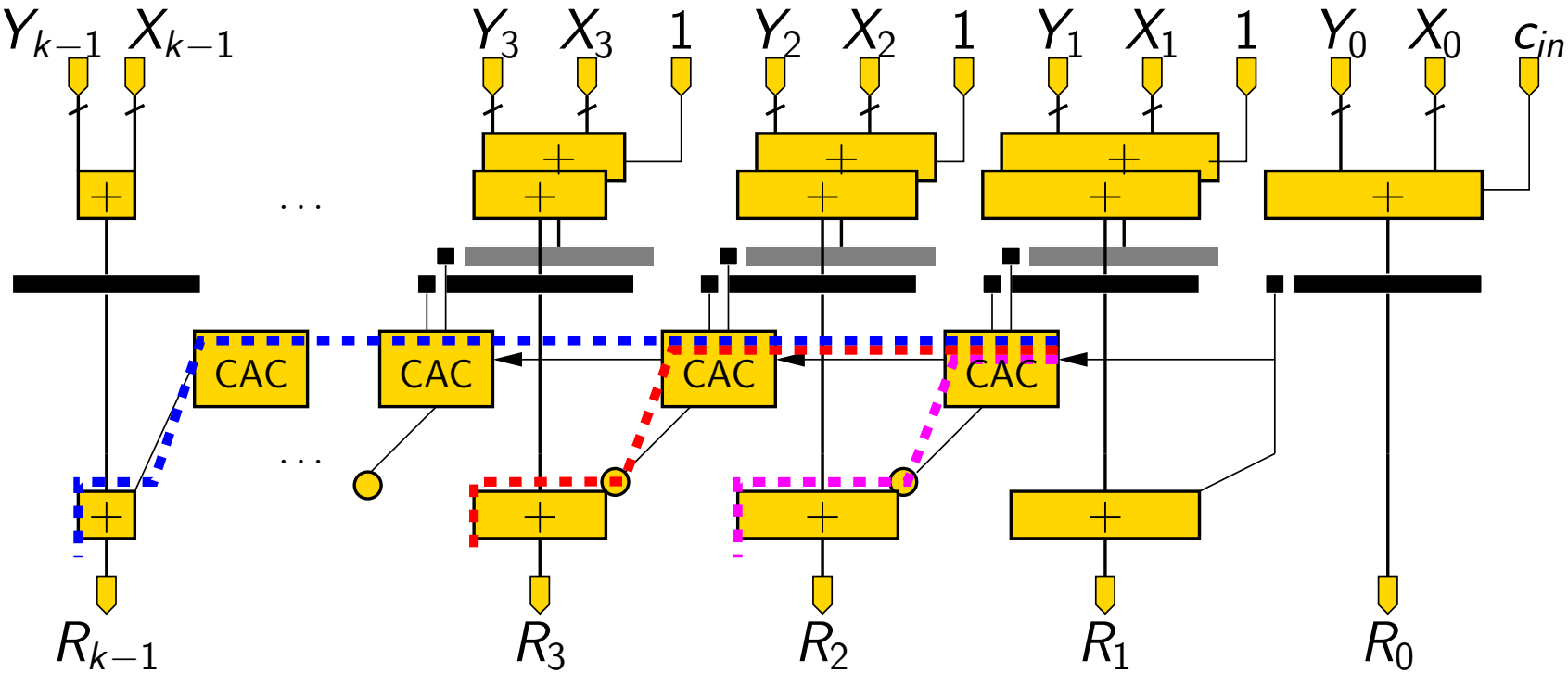
- for some values of (w, f) discard second register level



- greedy algorithm for determining chunk sizes
  - ensure for each chunk: $delay_{CAC} + delay_{routing} + delay_{sum} < 1/f$

Short-Latency Architecture: Optimization

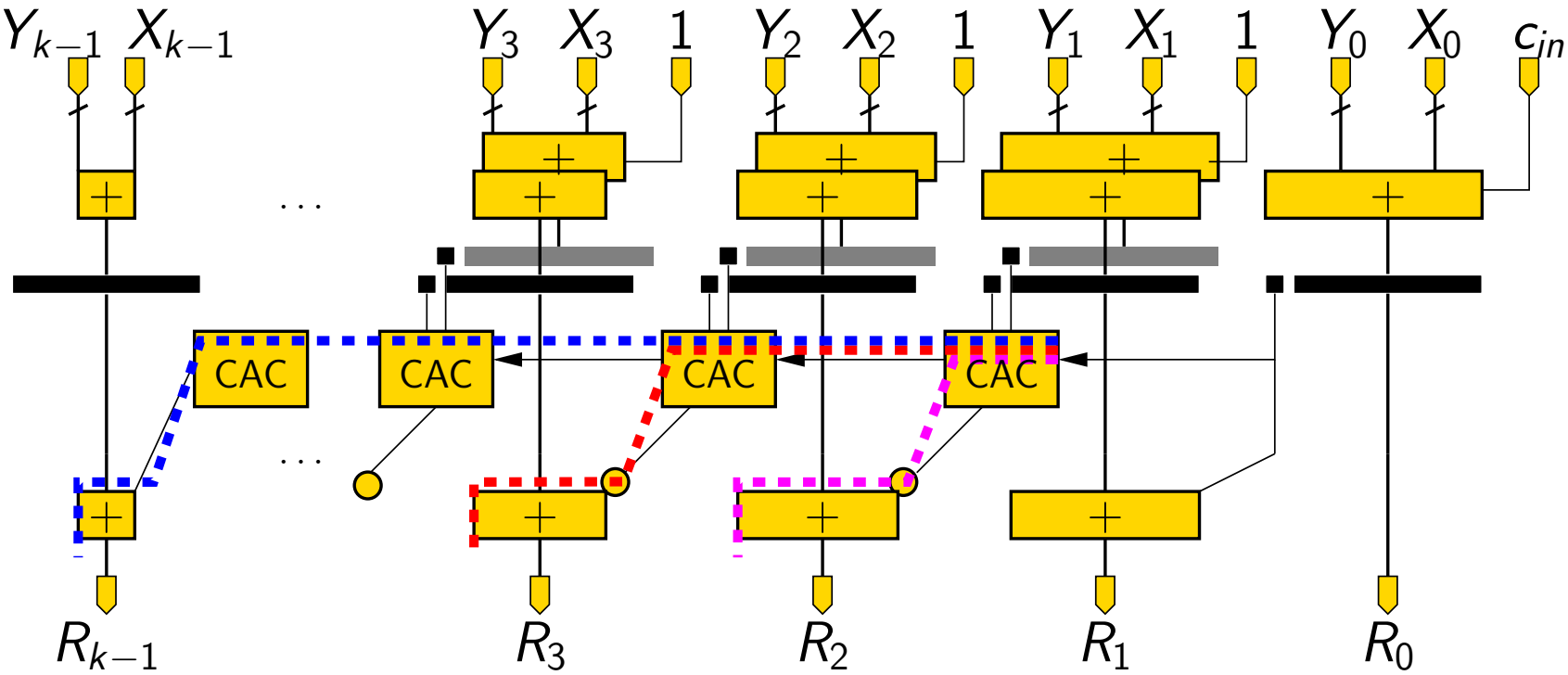
- for some values of (w, f) discard second register level



- greedy algorithm for determining chunk sizes
  - ensure for each chunk: $delay_{CAC} + delay_{routing} + delay_{sum} < 1/f$

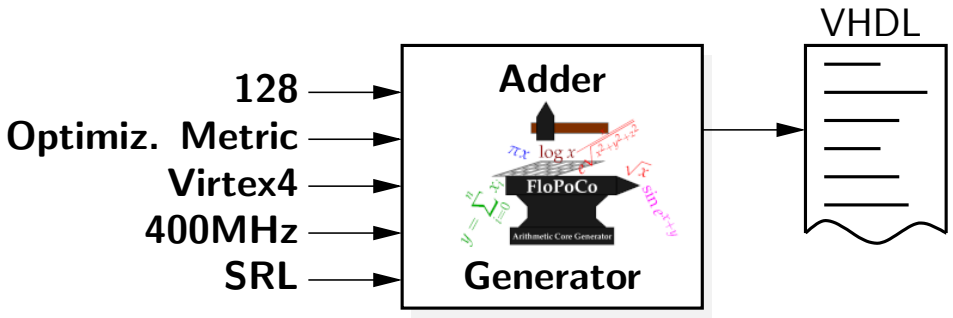
Short-Latency Architecture: Optimization

- for some values of (w, f) discard second register level

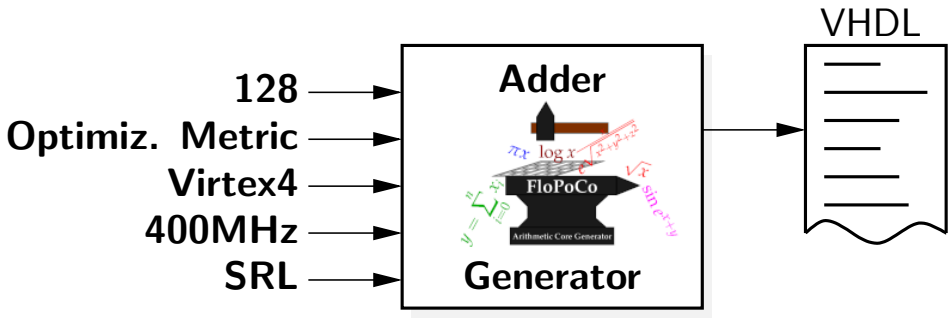


- greedy algorithm for determining chunk sizes
  - ensure for each chunk: $delay_{CAC} + delay_{routing} + delay_{sum} < 1/f$
  - save some registers + 1 cycle count
- if no solution, insert second register level

Reality Check

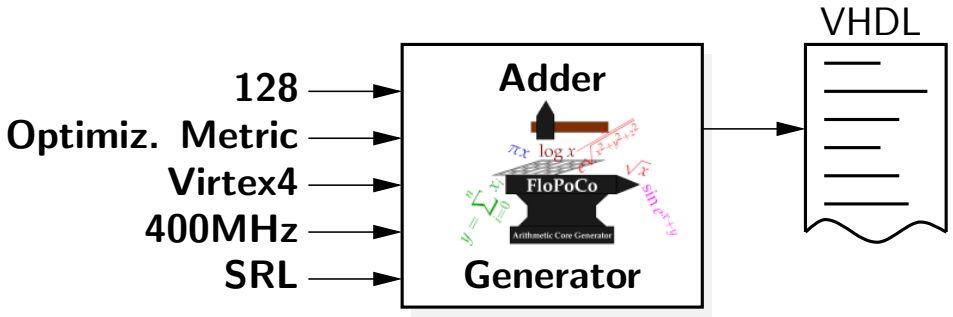


Reality Check



Architecture	SRL	Results			Estimations			Mismatch		
		L	R	S	L	R	S	L	R	S
Classical	N	128	573	309	128	573	300	0	0	2%
	Y	318	292	198	318	292	194	0	0	2%
Alternative	N	222	392	216	223	393	207	0.4%	0.2%	4%
	Y	352	199	183	352	199	177	0	0	3%
Short-Latency	N	288	264	216	293	263	214	2%	0.3%	0.9%
	Y	416	136	216	421	137	211	2%	0.7%	2%

Reality Check



Architecture	SRL	Results			Estimations			Mismatch		
		L	R	S	L	R	S	L	R	S
Classical	N	128	573	309	128	573	300	0	0	2%
	Y	318	292	198	318	292	194	0	0	2%
Alternative	N	222	392	216	223	393	207	0.4%	0.2%	4%
	Y	352	199	183	352	199	177	0	0	3%
Short-Latency	N	288	264	216	293	263	214	2%	0.3%	0.9%
	Y	416	136	216	421	137	211	2%	0.7%	2%

Largest mismatch 4%: good approximation formulas

Synthesis Results

Size	Freq	Target	SRL	Optim.	Classic. Cost	Classic. D	Altern. Cost	Altern. D	Short-Lat. Cost	Short-Lat. D	Gain[1]
32bit	200	Spartan3 (-5)	Y	SLICE	62	4	62	4	76	2	0%
			N		110		84		64		41%
64bit	450	Virtex4 (-12)	Y	SLICE	96	2	81	2	109	2	15%
			N		113		82		110		27%
128bit	450	Virtex4 (-12)	Y	SLICE	247	5	230	5	258	2	6%
			N		516		369		258		50%
128bit	450	Virtex5 (-3)	Y	REG	322	4	232	4	143	2	56%
			N		718		525		267		63%

- proposed architectures gain(for examples):
  - 15% less slices, 56% less registers when SRL
  - 50% less slices, 63% less registers when NO SRL

[1] compared to the classical implementation

Conclusions

- binary addition pervasive in FPGA applications
- adder operator generator implemented in FloPoCo
- classical pipeline, plus two novel pipelined addition architectures
- selection of the best implementation given the specifications by accurate resource estimation
- performance and resource improvements for operators using it

Conclusions

- binary addition pervasive in FPGA applications
- adder operator generator implemented in FloPoCo
- classical pipeline, plus two novel pipelined addition architectures
- selection of the best implementation given the specifications by accurate resource estimation
- performance and resource improvements for operators using it

In 2010 we can still improve elementary arithmetic operations on FPGA!

Future Work

- applicable to Altera targets, but need different resource estimation formulas
- low-latency architecture could be improved if synthesis tools were more clever
- implement similar resource evaluation for larger components

Future Work

- applicable to Altera targets, but need different resource estimation formulas
- low-latency architecture could be improved if synthesis tools were more clever
- implement similar resource evaluation for larger components

On-going work on "pipelining made easy" in FloPoCo

Thank you for your attention !

- try it at
  http://www.ens-lyon.fr/LIP/Arenaire/Ware/FloPoCo/
  New release 2.1.0 from Aug. 10, 2010
- or, come to us and ask for a demo

[DP96] L. Dadda and V. Piuri.
Pipelined Adders.
Computers, IEEE Transactions on, 45(3):348–356, Mar 1996.

[EL04] M. D. Ercegovac and T. Lang.
Digital Arithmetic.
Morgan Kaufmann Publishers, 2004.

[MJE97] Peiro Marcos Martinez, Valls Javier, and Boemo Eduardo.
On the design of FPGA-based Multioperand Pipeline Adders.
In *XII Design of Circuits and Integrated System Conference*, 1997.

[Mul89] J. M. Muller.
Arithmétique des Ordinateurs.
Masson, Paris, 1989.

[US93] I.H. Unwala and E.E. Swartzlander.
Superpipelined Adder Designs.
In *Circuits and Systems, 1993., ISCAS '93, 1993 IEEE International Symposium on*, pages 1841–1844, May 1993.

[XY98] Shanzhen Xing and William W.h. Yu.
FPGA Adders: Performance Evaluation and Optimal Design.
IEEE Design and Test of Computers, 15:24–29, 1998.