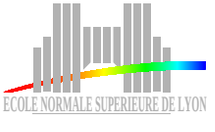
ARITH 18 – June 25–27, 2007

Return of the hardware floating-point elementary functions

Jérémie Detrey, Florent de Dinechin, and Xavier Pujol

Projet Arénaire – LIP
UMR CNRS – ENS Lyon – UCB Lyon – INRIA 5668
http://www.ens-lyon.fr/LIP/Arenaire/


Outline of the talk

- Context
- Double-precision exponential
- Results
- Conclusion

Outline of the talk

- Context
- Double-precision exponential
- Results
- Conclusion

A long time ago...

(in a galaxy not so far away)

- a bit of paleo-bibliography

A long time ago...

(in a galaxy not so far away)

- a bit of paleo-bibliography
  - M. D. Ercegovac (IEEE TC, 1975)
    Radix-16 evaluation of certain elementary functions.
  - G. Paul and M. W. Wilson (ACM TOMS, 1976)
    Should the elementary functions be incorporated into computer instruction sets?
  - C. Wrathall and T. C. Chen. (ARITH 4, 1978)
    Convergence guarantee and improvements for a hardware exponential and logarithm evaluation scheme.
  - P. Farmwald (ARITH 5, 1981)
    High-bandwidth evaluation of elementary functions.
  - M. Cosnard, A. Guyot, B. Hochet, J.-M. Muller, H. Ouaouicha, P. Paul, and E. Zysmann (ARITH 8, 1987)
    The FELIN arithmetic coprocessor chip.

FPUs strike back

- ... then came the floating-point unit
  - dedicated efficient hardware operators
  - only basic operations: $+$, $-$, $\times$, $\div$ and $\sqrt{\ }$

FPUs strike back

- ... then came the floating-point unit
  - dedicated efficient hardware operators
  - only basic operations: $+$, $-$, $\times$, $\div$ and $\sqrt{}$
- what about elementary functions?
  - comparatively rare operations
  - hardware implementation would be a waste of silicon
  - dedicate silicon to more useful units (ALUs, FPUs, caches)

FPUs strike back

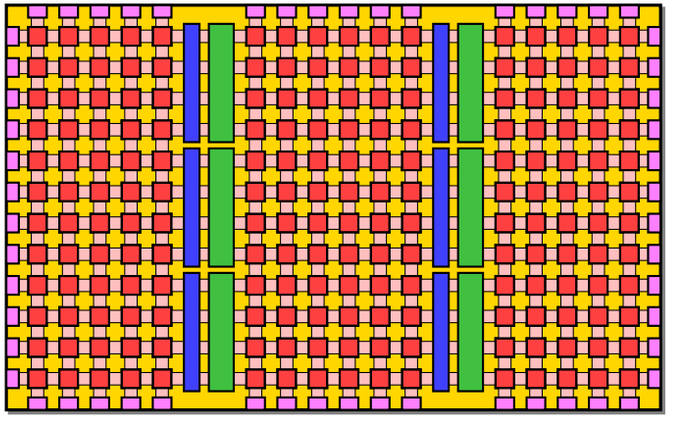
- ... then came the floating-point unit
  - dedicated efficient hardware operators
  - only basic operations: $+$, $-$, $\times$, $\div$ and $\sqrt{}$
- what about elementary functions?
  - comparatively rare operations
  - hardware implementation would be a waste of silicon
  - dedicate silicon to more useful units (ALUs, FPUs, caches)
- only software or micro-code implementations

FPGAs: a new hope?

- Field-Programmable Gate Arrays
- reconfigurable integrated circuits

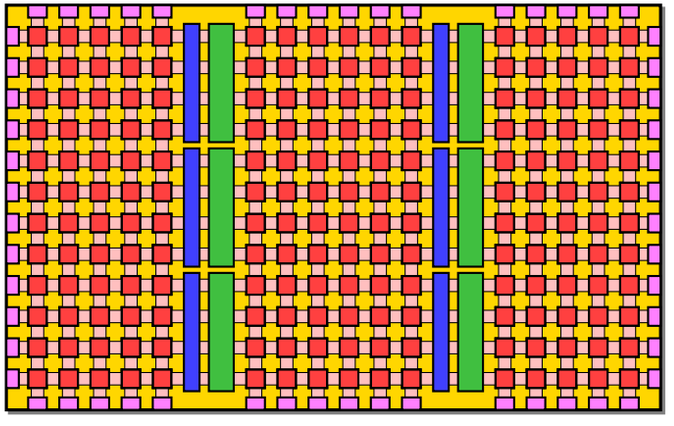
FPGAs: a new hope?

- Field-Programmable Gate Arrays
- reconfigurable integrated circuits
- architecture based on programmable logic cells and routing resources
  - lower performances than ASICs
  - high flexibility
  - fine-grain parallelism
  - lower cost per unit



FPGAs: a new hope?

- Field-Programmable Gate Arrays
- reconfigurable integrated circuits
- architecture based on programmable logic cells and routing resources
  - lower performances than ASICs
  - high flexibility
  - fine-grain parallelism
  - lower cost per unit



- 1 billion transistor FPGAs: huge computational capacity
- many application domains:
  - digital signal and image processing
  - cryptography
  - bioinformatics
  - scientific computing
  - ...

FPGAs and arithmetic

- initially: LUT-based logic cells

FPGAs and arithmetic

- initially: LUT-based logic cells
- currently: only integer arithmetic
  - dedicated logic and routing for fast adders
  - small embedded multipliers (18×18 bits)
  - multiply-and-accumulate blocks
- not enough for many applications

FPGAs and arithmetic

- initially: LUT-based logic cells
- currently: only integer arithmetic
  - dedicated logic and routing for fast adders
  - small embedded multipliers (18×18 bits)
  - multiply-and-accumulate blocks
- not enough for many applications
- strong need for more complex operators
  - other operations: division, square root, elementary functions, ...
  - other number systems: modular arithmetic, real arithmetic, ...

FPGAs and arithmetic

- initially: LUT-based logic cells
- currently: only integer arithmetic
  - dedicated logic and routing for fast adders
  - small embedded multipliers (18×18 bits)
  - multiply-and-accumulate blocks
- not enough for many applications
- strong need for more complex operators
  - other operations: division, square root, elementary functions, ...
  - other number systems: modular arithmetic, real arithmetic, ...

FPLibrary

- library of portable VHDL operators for floating-point
- all operators are parameterized in terms of range and precision

FPLibrary

- library of portable VHDL operators for floating-point
- all operators are parameterized in terms of range and precision

	single precision	double precision
$+/-$	✓	✓
$\times$	✓	✓
$\div$	✓	✓
$\sqrt{\ }$	✓	✓

FPLibrary

- library of portable VHDL operators for floating-point
- all operators are parameterized in terms of range and precision

	single precision	double precision
$+/-$	✓	✓
$\times$	✓	✓
$\div$	✓	✓
$\sqrt{\ }$	✓	✓
$\log x$	✓	
e^x	✓	
$\sin x$ / $\cos x$	✓	

FPLibrary

- library of portable VHDL operators for floating-point
- all operators are parameterized in terms of range and precision

	single precision	double precision
$+/-$	✓	✓
$\times$	✓	✓
$\div$	✓	✓
$\sqrt{\ }$	✓	✓
$\log x$	✓	
e^x	✓	
$\sin x$ / $\cos x$	✓	

- single-precision logarithm and exponential
  - hardware-specific algorithms
  - *ad-hoc* range reduction
  - table-based fixed-point evaluation
  - small and fast operators

FPLibrary

- library of portable VHDL operators for floating-point
- all operators are parameterized in terms of range and precision

	single precision	double precision
$+/-$	✓	✓
$\times$	✓	✓
$\div$	✓	✓
$\sqrt{\ }$	✓	✓
$\log x$	✓	?
e^x	✓	?
$\sin x$ / $\cos x$	✓	

- single-precision logarithm and exponential
  - hardware-specific algorithms
  - *ad-hoc* range reduction
  - table-based fixed-point evaluation
  - small and fast operators

Double precision: using the same method?

- range reduction and reconstruction are scalable

Double precision: using the same method?

- range reduction and reconstruction are scalable
- table-based method for the actual computation
  - exponential growth of the area
  - estimations w.r.t. single precision: 15× larger for the exponential, and 40× larger for the logarithm!!
  - unacceptable overhead for usual FPGAs
- need for another algorithm, suited to higher precisions

Double precision: using the same method?

- range reduction and reconstruction are scalable
- table-based method for the actual computation
  - exponential growth of the area
  - estimations w.r.t. single precision: $15\times$ larger for the exponential, and $40\times$ larger for the logarithm!!
  - unacceptable overhead for usual FPGAs
- need for another algorithm, suited to higher precisions
- iterative method
  - smaller architecture
  - higher scalability
  - longer critical path

Outline of the talk

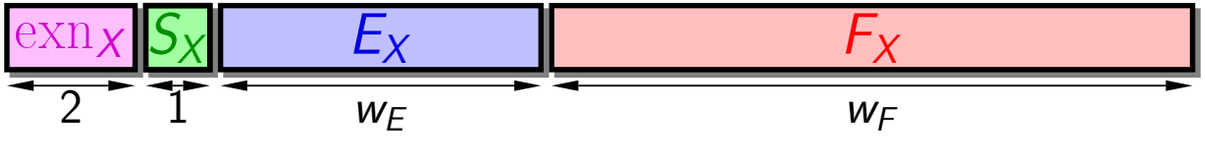
- Context
- Double-precision exponential
- Results
- Conclusion

Number format

- 2 parameters: w_E (range) and w_F (precision)
- inspired from the IEEE-754 standard:

$$X = (-1)^{S_X} \cdot 1.F_X \cdot 2^{E_X - E_0}$$

Number format



- 2 parameters: w_E (range) and w_F (precision)
- inspired from the IEEE-754 standard:

$$X = (-1)^{S_X} \cdot 1.F_X \cdot 2^{E_X - E_0}$$

- 2 extra bits for exceptional cases: zero, infinity or *Not-a-Number* (NaN)

Evaluation method

- range reduction:

$$X = k \cdot \log 2 + Y \qquad \text{with } k \in \mathbb{Z} \text{ and } 0 \leq Y < 1$$

- we obtain:

$$R = e^X = 2^k \cdot e^Y$$

Evaluation method

- range reduction:

$$X = k \cdot \log 2 + Y \qquad \text{with } k \in \mathbb{Z} \text{ and } 0 \leq Y < 1$$

- we obtain:

$$R = e^X = 2^k \cdot e^Y$$

- fixed-point e^Y?

Evaluation method

- range reduction:

$$X = k \cdot \log 2 + Y \qquad \text{with } k \in \mathbb{Z} \text{ and } 0 \leq Y < 1$$

- we obtain:

$$R = e^X = 2^k \cdot e^Y$$

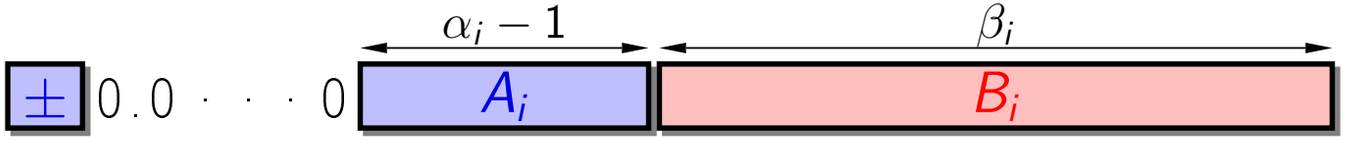
- fixed-point e^Y?
  generalization of an idea by Wong and Goto (IEEE TC 1994)
  - successive range reductions of the fixed-point argument Y
  - once the argument sufficiently reduced, direct evaluation of the exponential
  - reconstructions using rectangular multipliers
  - computes $e^Y - 1$

Iterative method: range reductions

- for step each i, we consider the argument Y_i (starting with $Y_0 = Y$)

Iterative method: range reductions

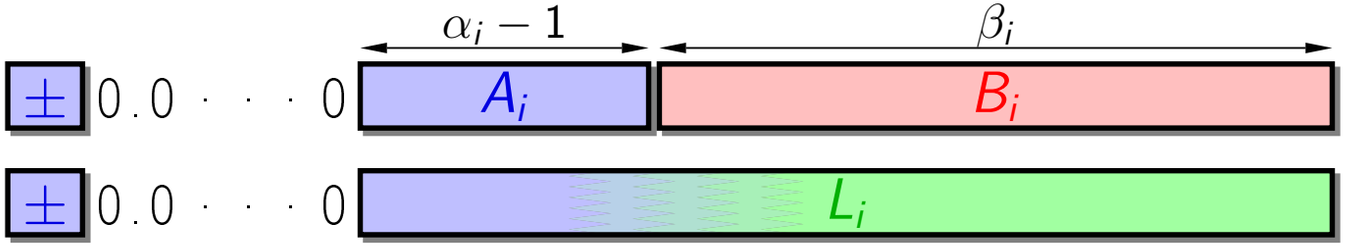
- for step each i, we consider the argument Y_i (starting with $Y_0 = Y$)



- splitting Y_i as $A_i + B_i$, we address two look-up tables with A_i:
  - $e^{A_i} - 1$, rounded to its α_i most significant bits, noted $\widetilde{e^{A_i}} - 1$
  - $L_i = \log\left(\widetilde{e^{A_i}}\right)$, rounded to its $\alpha_i + \beta_i$ most significant bits

Iterative method: range reductions

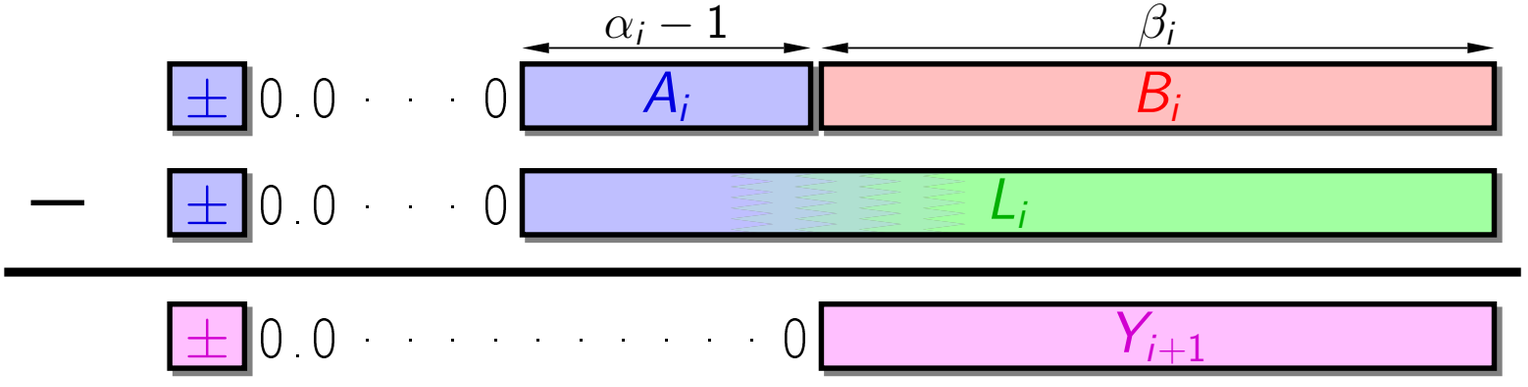
- for step each i, we consider the argument Y_i (starting with $Y_0 = Y$)



- splitting Y_i as $A_i + B_i$, we address two look-up tables with A_i:
  - $e^{A_i} - 1$, rounded to its α_i most significant bits, noted $\widetilde{e^{A_i}} - 1$
  - $L_i = \log\left(\widetilde{e^{A_i}}\right)$, rounded to its $\alpha_i + \beta_i$ most significant bits
- by construction, $L_i \approx Y_i$

Iterative method: range reductions

- for step each i, we consider the argument Y_i (starting with $Y_0 = Y$)



- splitting Y_i as $A_i + B_i$, we address two look-up tables with A_i:
  - $e^{A_i} - 1$, rounded to its α_i most significant bits, noted $\widetilde{e^{A_i} - 1}$
  - $L_i = \log\left(\widetilde{e^{A_i}}\right)$, rounded to its $\alpha_i + \beta_i$ most significant bits
- by construction, $L_i \approx Y_i$
- we then define Y_{i+1} as $Y_i - L_i$:
  - the $\alpha_i - 1$ most significant bits of Y_i are cancelled
  - Y_{i+1} is a $1 + \beta_i$-bit number

Iterative method: computing the exponential

- the reduction process is iterated until the step k such that

$$Y_k < 2^{-\lceil w_F/2 \rceil}$$

Iterative method: computing the exponential

- the reduction process is iterated until the step k such that

$$Y_k < 2^{-\lceil w_F/2 \rceil}$$

- we can then approximate the exponential as

$$e^{Y_k} - 1 \approx Y_k$$

Iterative method: reconstructions

- at each step i, we have:
  - $\widetilde{e^{A_i}} - 1$, from the corresponding range reduction step
  - $e^{Y_{i+1}} - 1$, from the previous reconstruction, with $Y_{i+1} = Y_i - \log\left(\widetilde{e^{A_i}}\right)$

Iterative method: reconstructions

- at each step i, we have:
  - $\widetilde{e^{A_i}} - 1$, from the corresponding range reduction step
  - $e^{Y_{i+1}} - 1$, from the previous reconstruction, with $Y_{i+1} = Y_i - \log\left(\widetilde{e^{A_i}}\right)$
- we then compute $e^{Y_i} - 1$ as

$$\left(\widetilde{e^{A_i}} - 1\right) \times \left(e^{Y_{i+1}} - 1\right) + \left(\widetilde{e^{A_i}} - 1\right) + \left(e^{Y_{i+1}} - 1\right)$$

Iterative method: reconstructions

- at each step i, we have:
  - $\widetilde{e^{A_i}} - 1$, from the corresponding range reduction step
  - $e^{Y_{i+1}} - 1$, from the previous reconstruction, with $Y_{i+1} = Y_i - \log\left(\widetilde{e^{A_i}}\right)$
- we then compute $e^{Y_i} - 1$ as

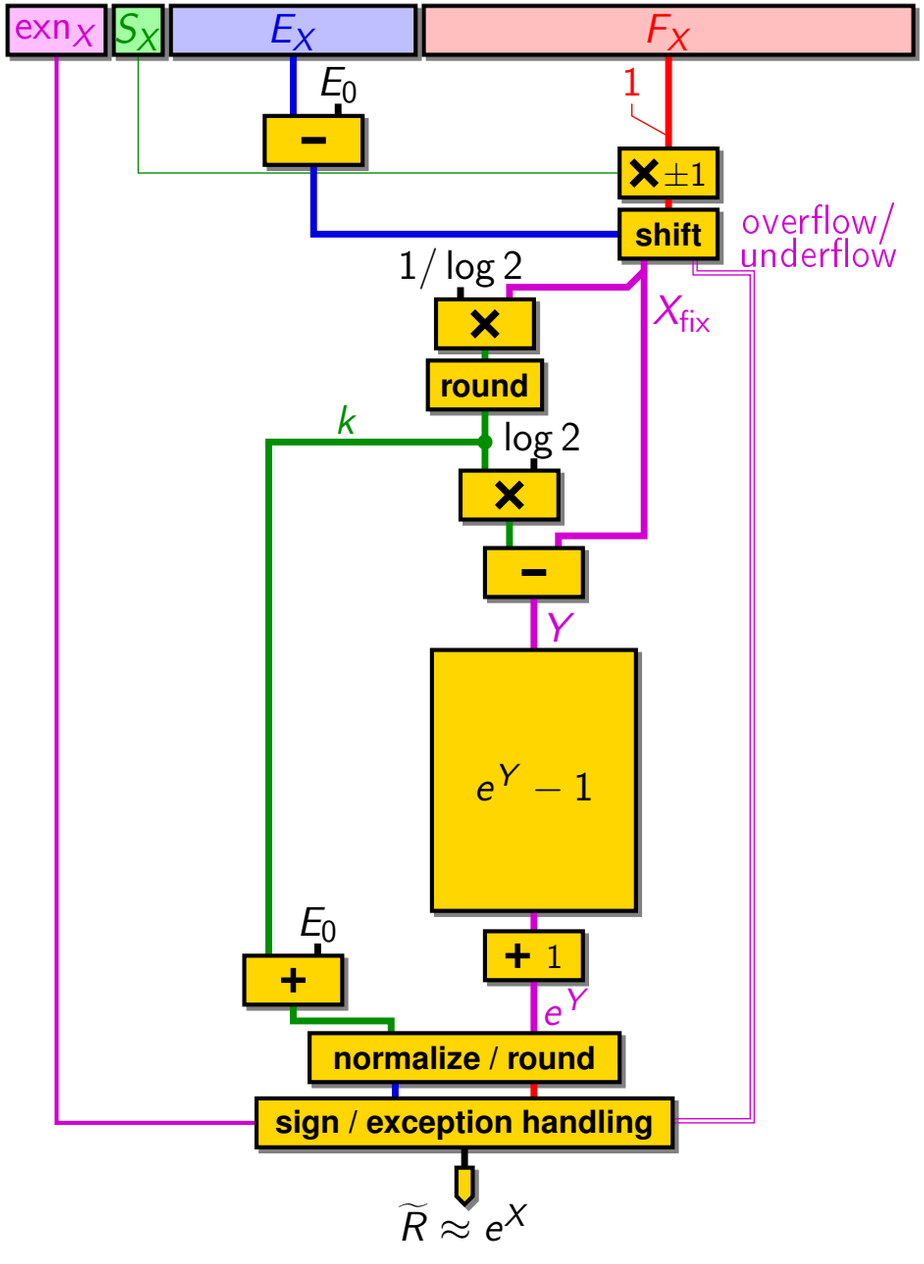
$$\left(\widetilde{e^{A_i}} - 1\right) \times \left(e^{Y_{i+1}} - 1\right) + \left(\widetilde{e^{A_i}} - 1\right) + \left(e^{Y_{i+1}} - 1\right)$$
$$= \widetilde{e^{A_i}} \cdot e^{Y_{i+1}} - 1$$

Iterative method: reconstructions

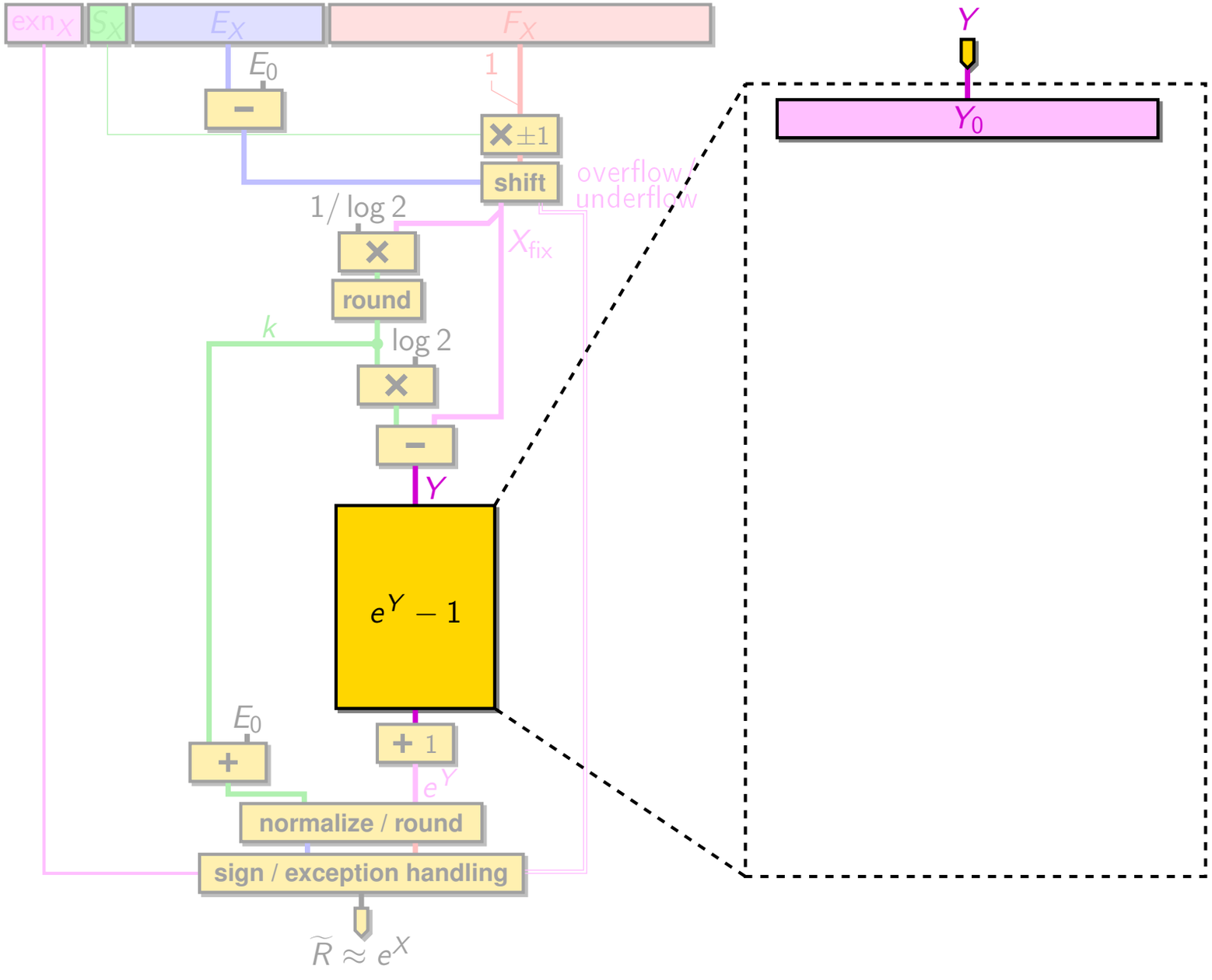
- at each step i, we have:
  - $\widetilde{e^{A_i}} - 1$, from the corresponding range reduction step
  - $e^{Y_{i+1}} - 1$, from the previous reconstruction, with $Y_{i+1} = Y_i - \log\left(\widetilde{e^{A_i}}\right)$
- we then compute $e^{Y_i} - 1$ as

$$\begin{aligned} & \left(\widetilde{e^{A_i}} - 1\right) \times \left(e^{Y_{i+1}} - 1\right) + \left(\widetilde{e^{A_i}} - 1\right) + \left(e^{Y_{i+1}} - 1\right) \\ = \; & \widetilde{e^{A_i}} \cdot e^{Y_{i+1}} - 1 \\ = \; & \widetilde{e^{A_i}} \cdot e^{Y_i} \cdot e^{-\log\left(\widetilde{e^{A_i}}\right)} - 1 \end{aligned}$$

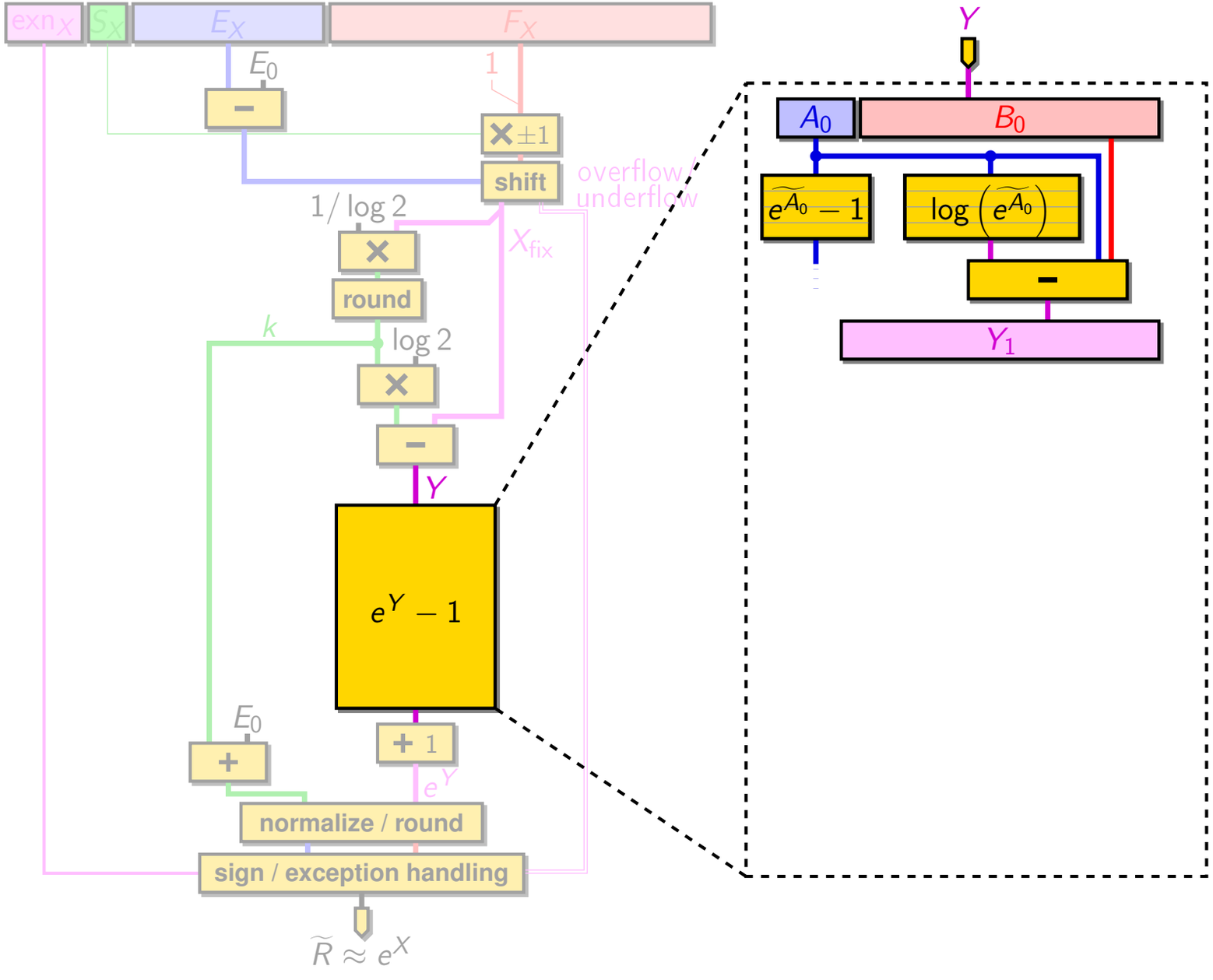
Architecture



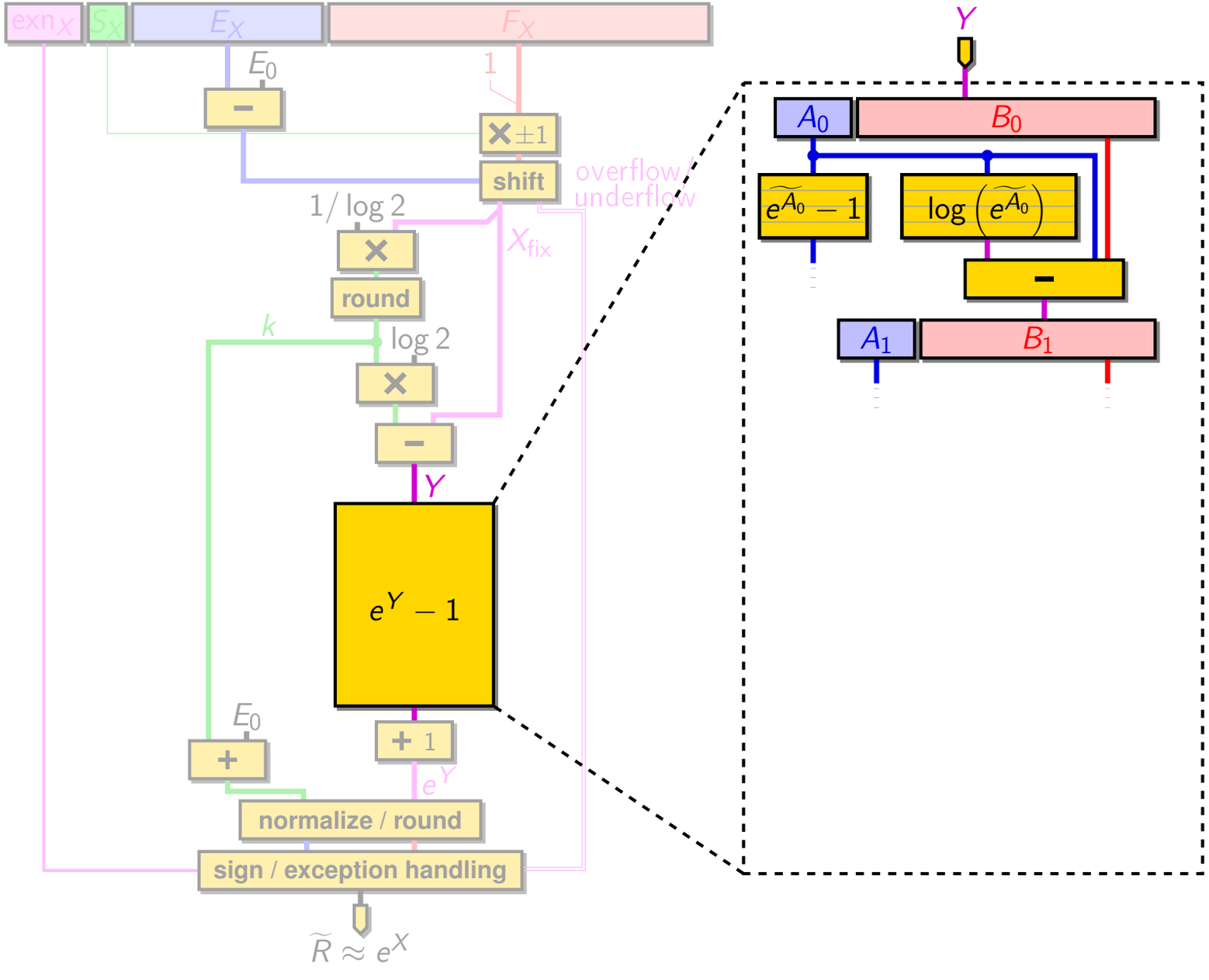
Architecture



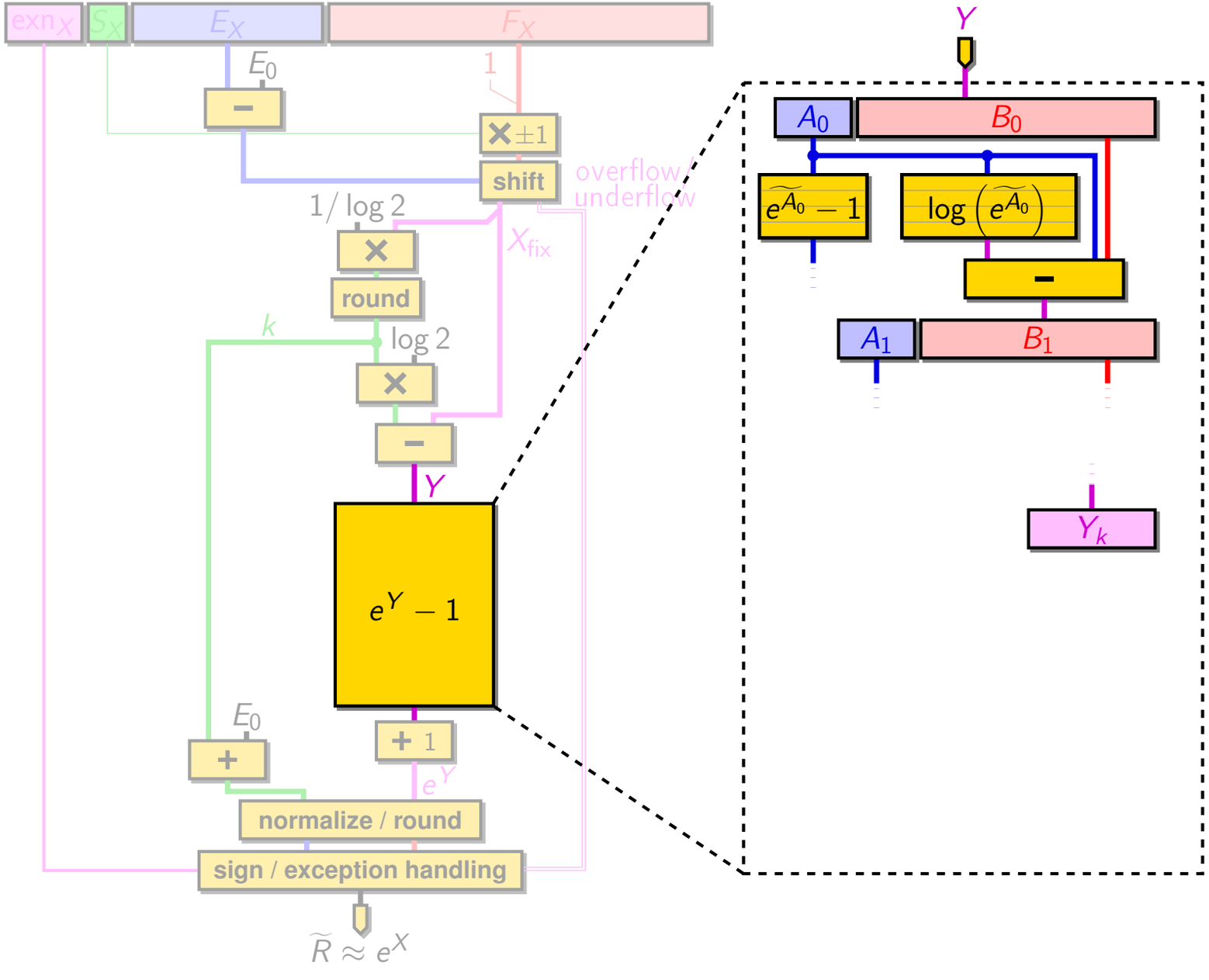
Architecture



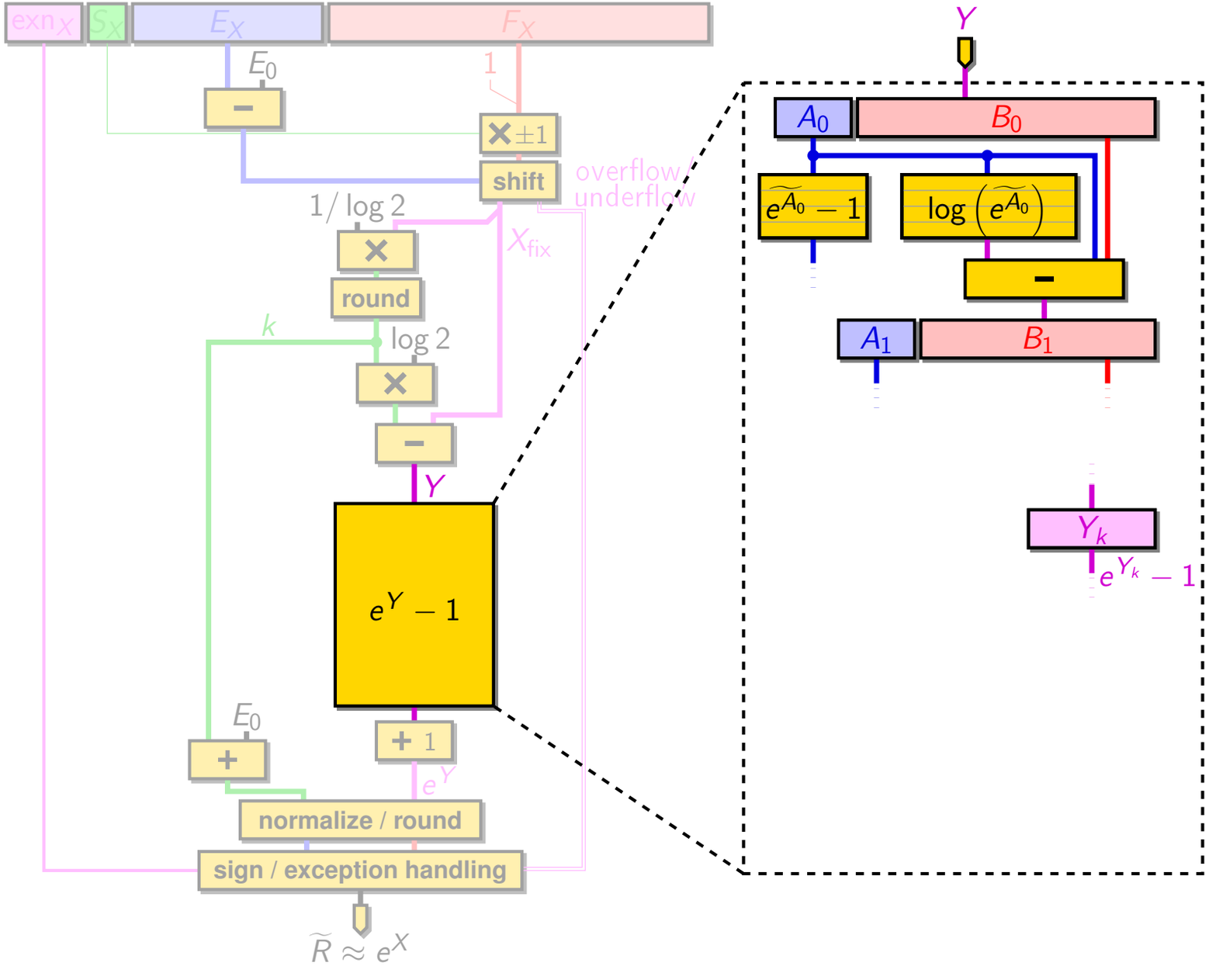
Architecture



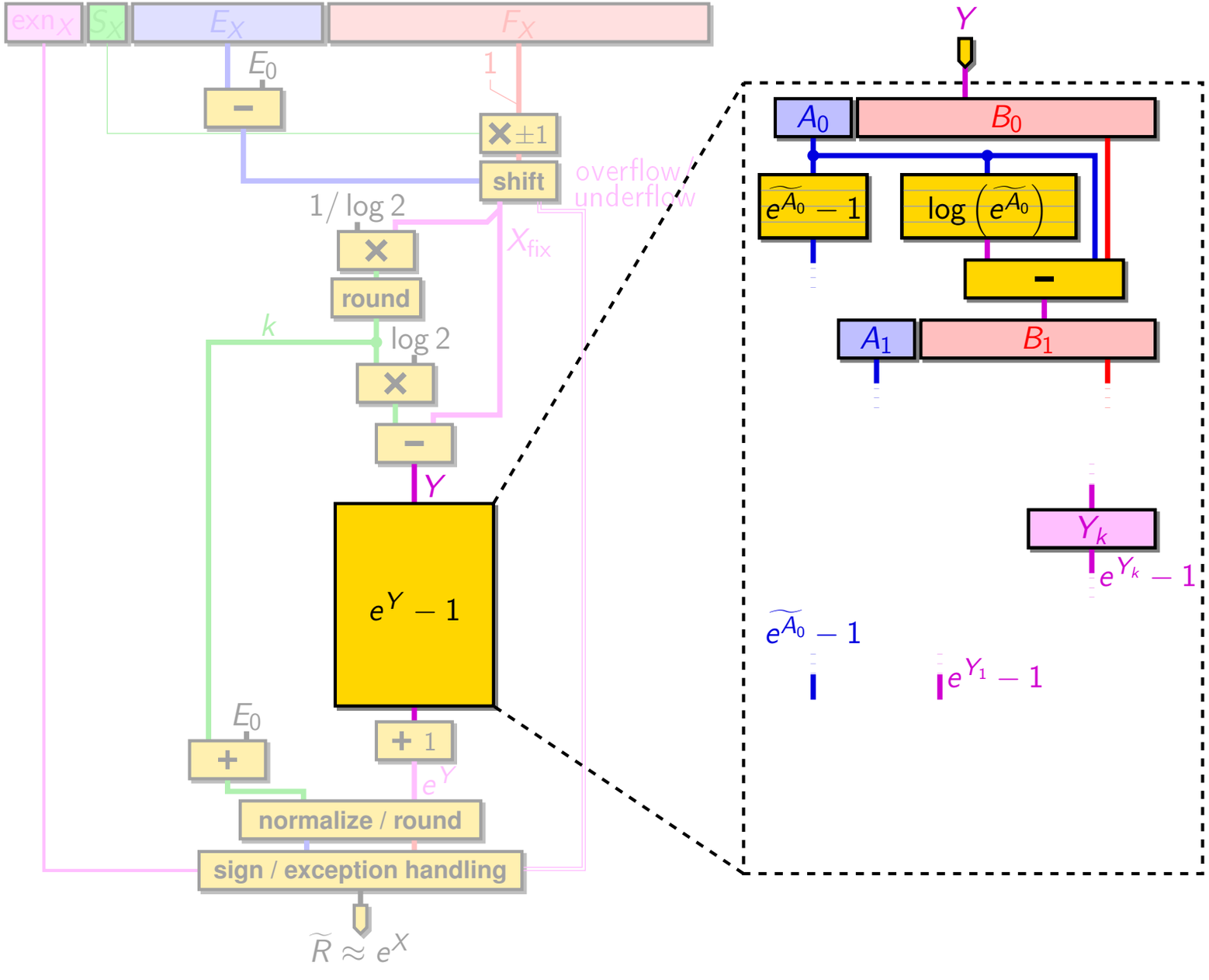
Architecture



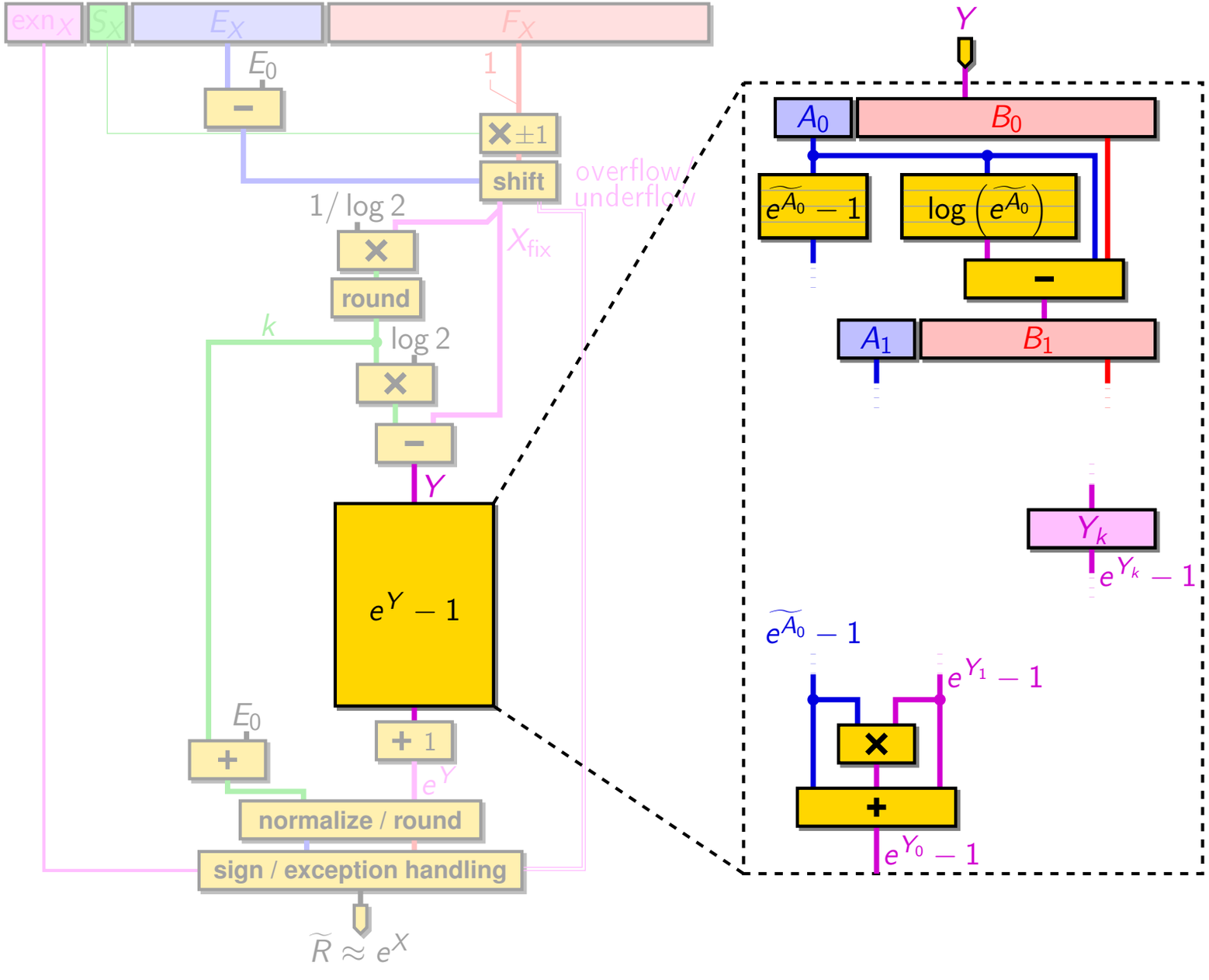
Architecture



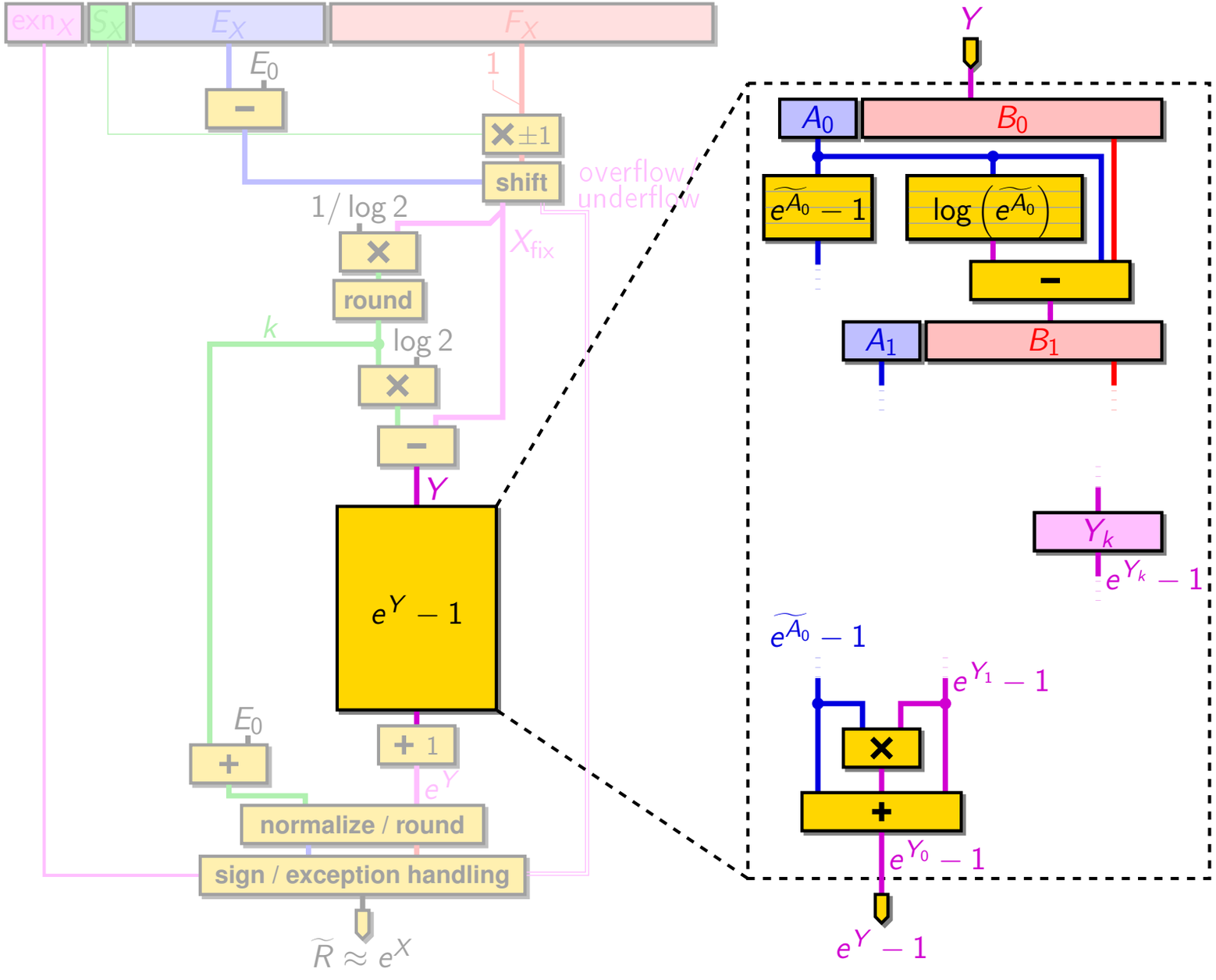
Architecture



Architecture



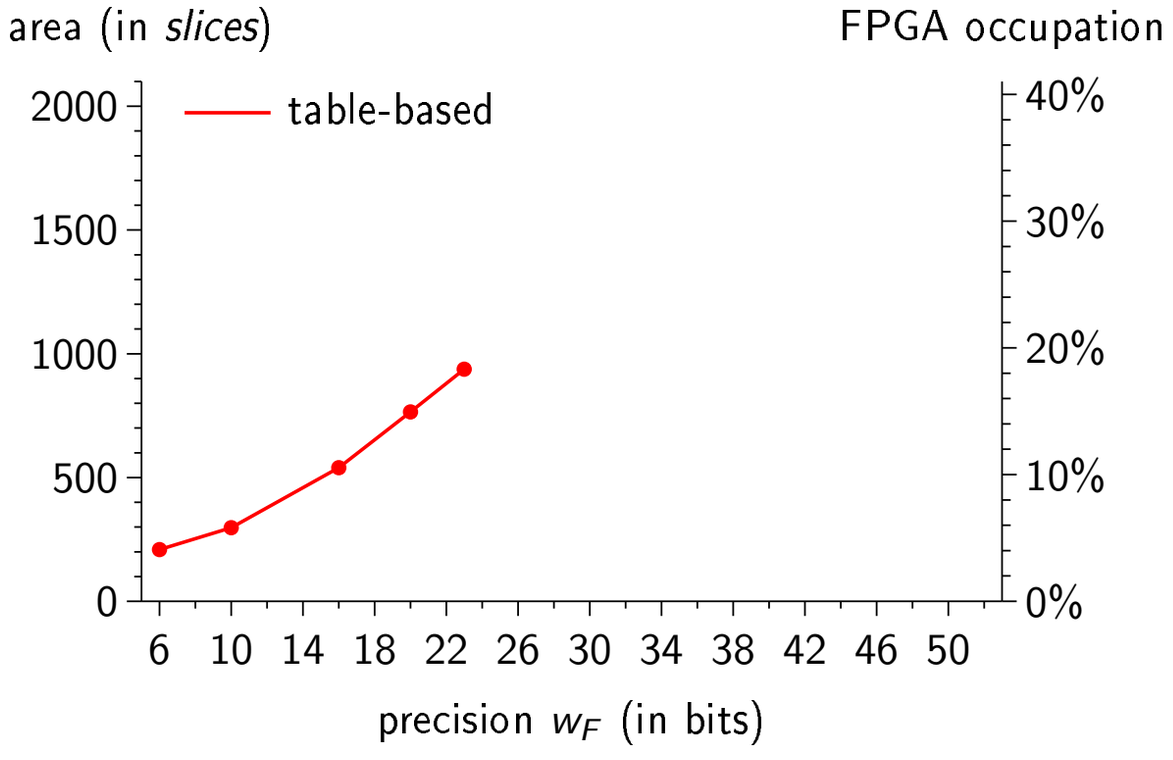
Architecture



Outline of the talk

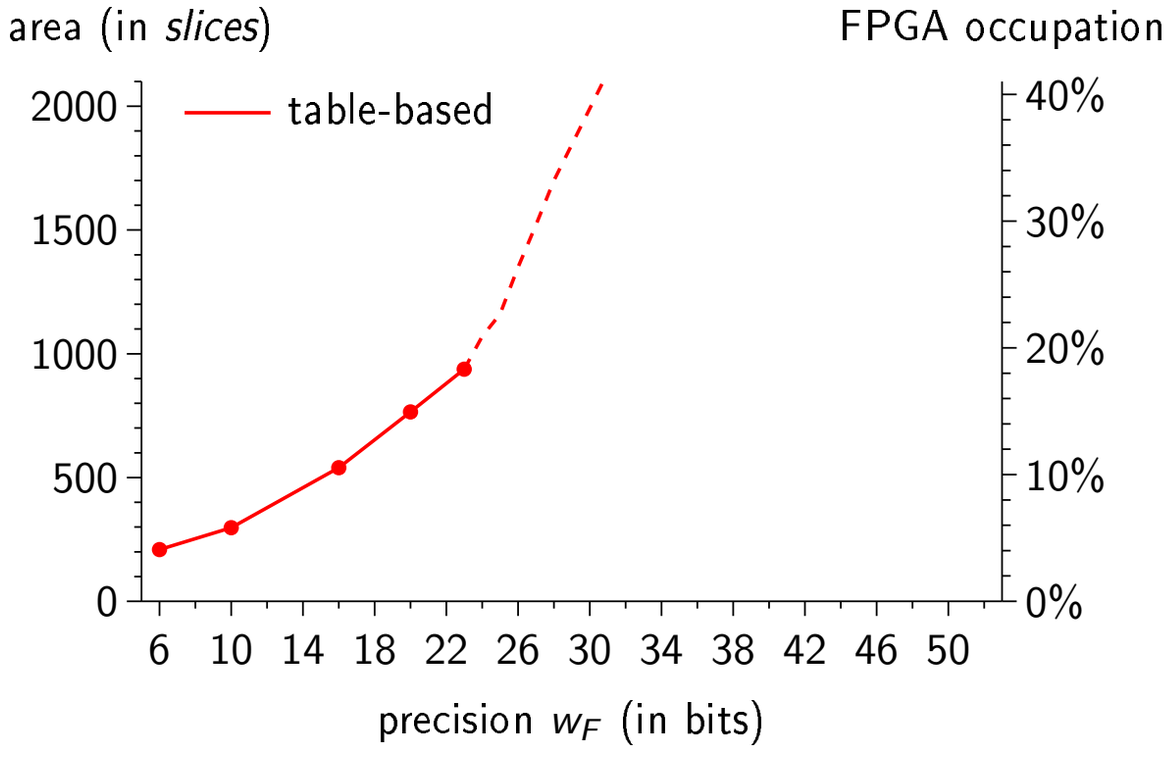
- Context
- Double-precision exponential
- Results
- Conclusion

Operator area (exponential)



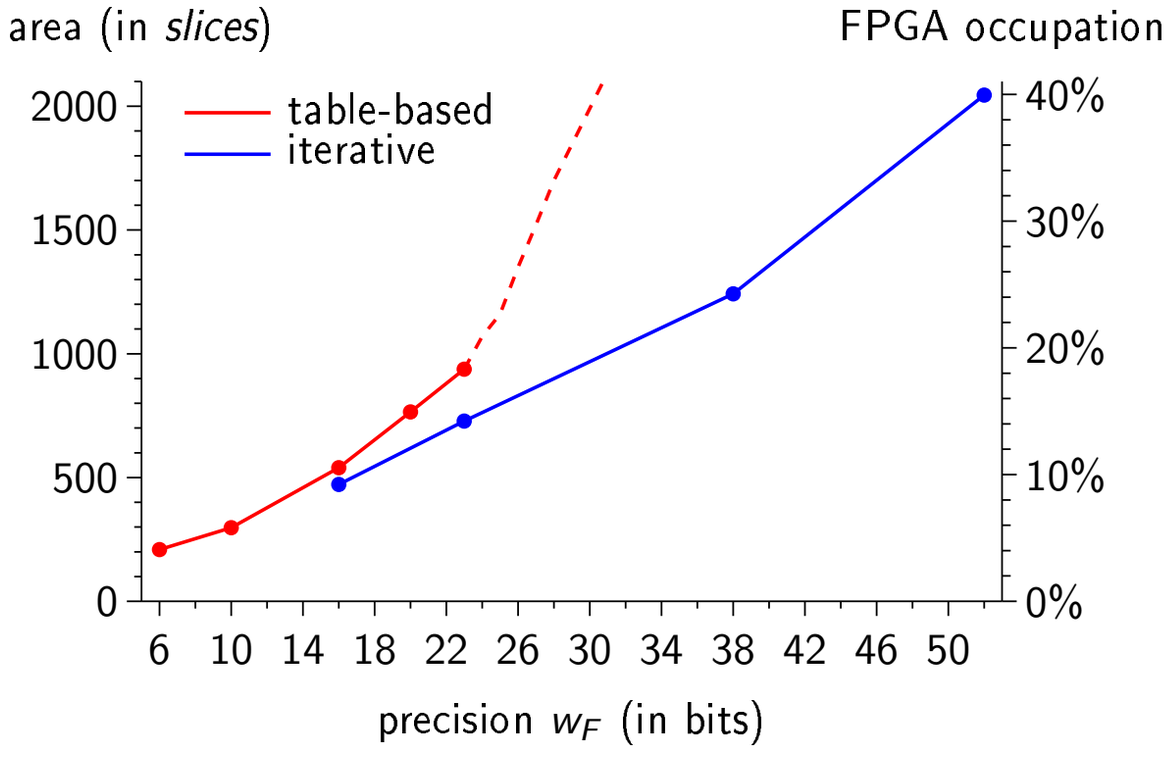
- single precision $(w_E, w_F) = (8, 23)$ (table-based method): 938 slices (18% of a Virtex-II 1000 FPGA)

Operator area (exponential)



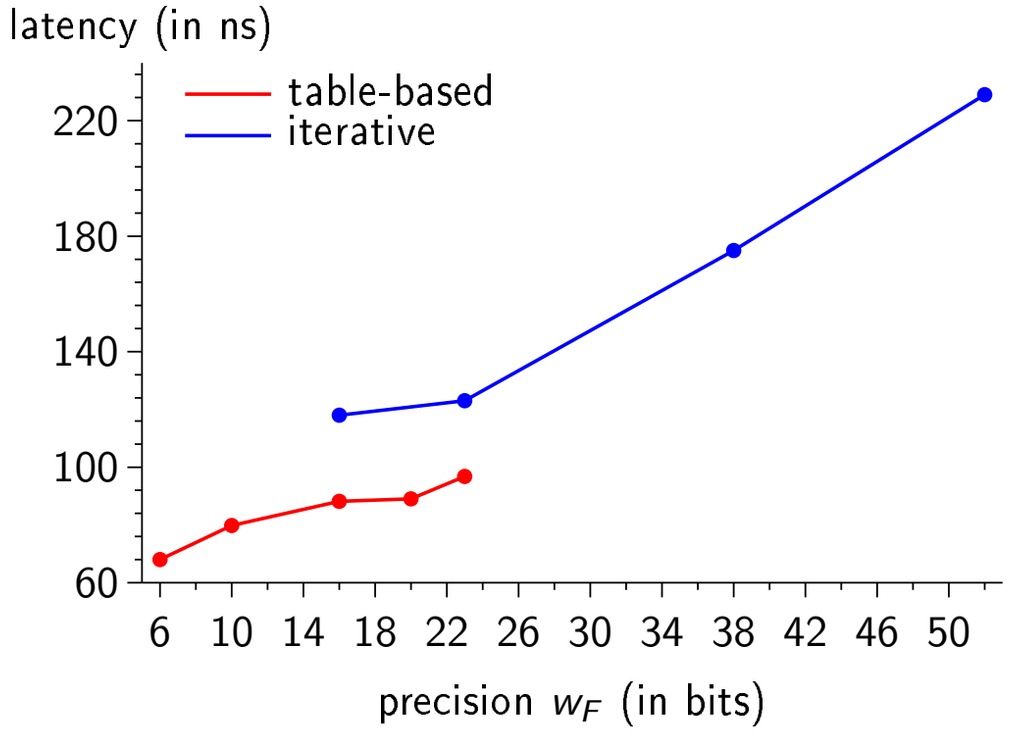
- single precision $(w_E, w_F) = (8, 23)$ (table-based method): 938 slices (18% of a Virtex-II 1000 FPGA)

Operator area (exponential)



- single precision $(w_E, w_F) = (8, 23)$ (table-based method): 938 slices (18% of a Virtex-II 1000 FPGA)
- double precision $(w_E, w_F) = (11, 52)$ (iterative method): 2045 slices (40%)

Operator latency (exponential)



- single precision $(w_E, w_F) = (8, 23)$ (table-based method): 97 ns
- double precision $(w_E, w_F) = (11, 52)$ (iterative method): 229 ns

Outline of the talk

- Context
- Double-precision exponential
- Results
- Conclusion

Our contribution

- exponential and logarithm operators
- up to double precision
- guaranteed faithful rounding
- scalable method
- hardware-specific algorithms: fast and cheap operators

Future work

- pipeline
- implement double precision for other functions for FPLibrary
- study compound functions

Future work

- pipeline
- implement double precision for other functions for FPLibrary
- study compound functions
- careful error analysis:
  - certified algorithms and operators
  - generic proofs (Gappa)
- most of this work is not FPGA-specific: extend it to ASICs

Thank you for your attention

- more information & download page:
  http://www.ens-lyon.fr/LIP/Arenaire/

Thank you for your attention

- more information & download page:
  http://www.ens-lyon.fr/LIP/Arenaire/

Questions?