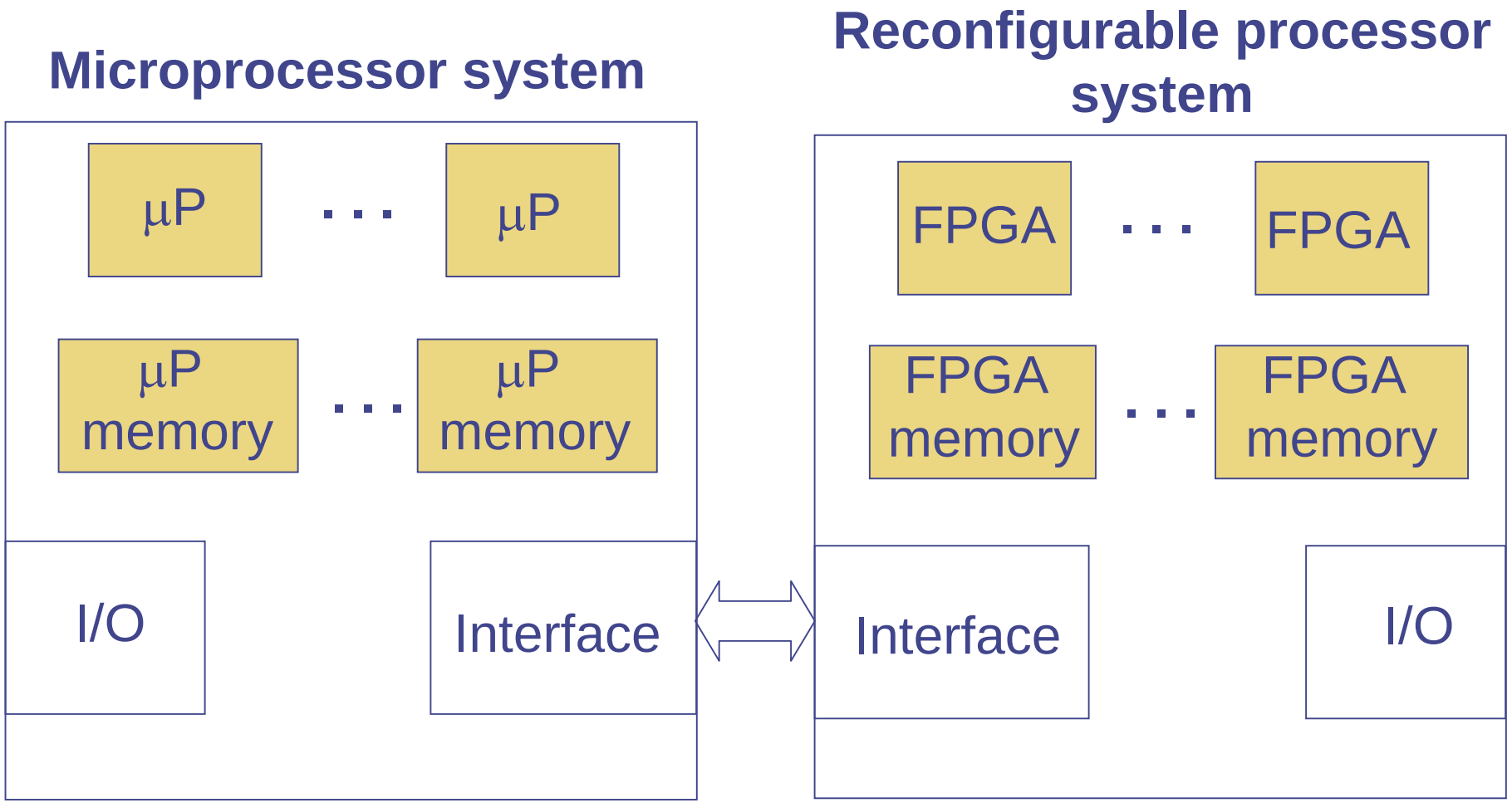
Implementation of Elliptic Curve Cryptosystems on a Reconfigurable Computer

Nghi Nguyen[1], Kris Gaj[1],
David Caliga[2], Tarek El-Ghazawi[3]

[1] George Mason University
[2] SRC Computers
[3] The George Washington University

What is a reconfigurable computer?

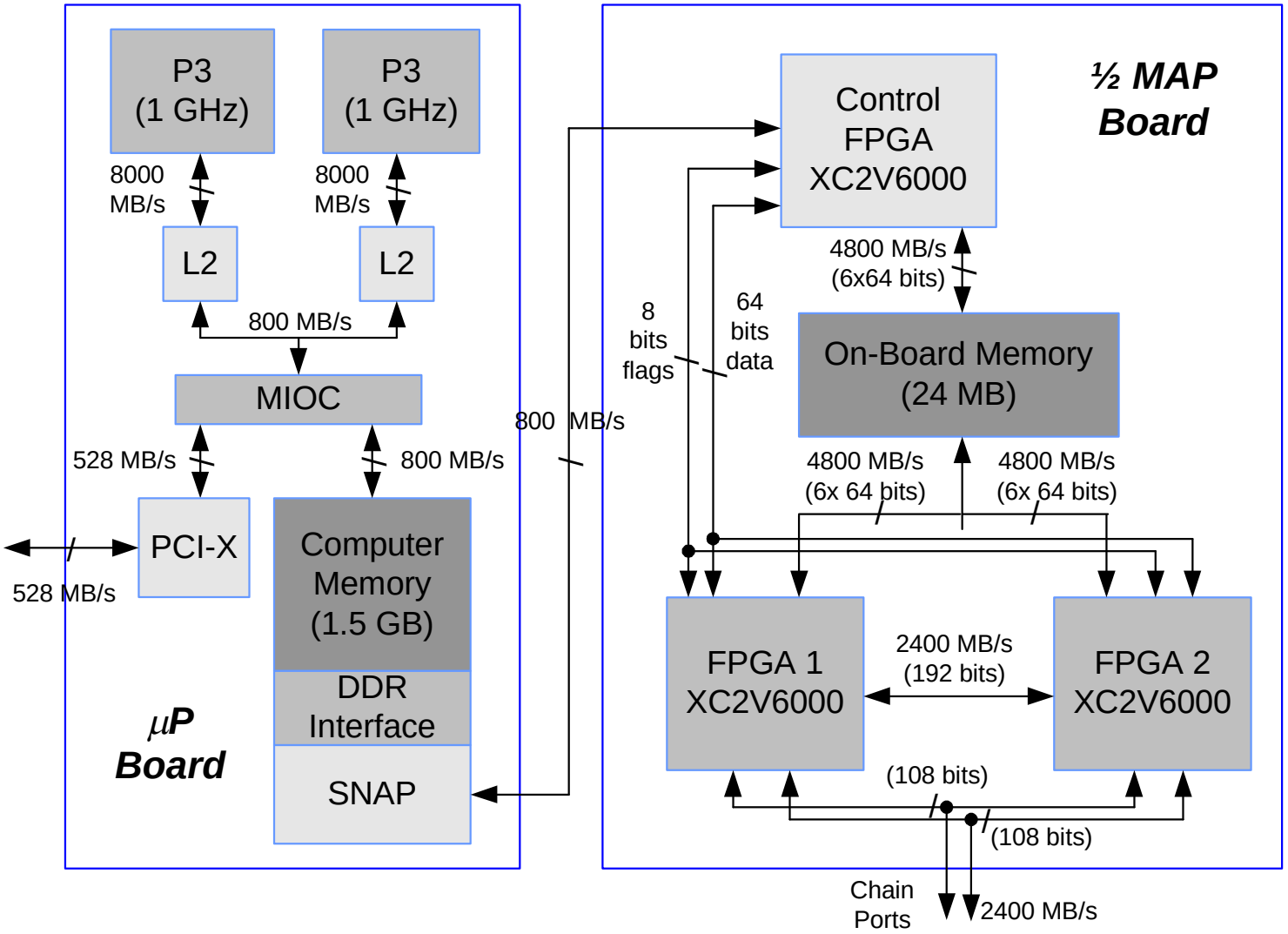


Characteristic Features

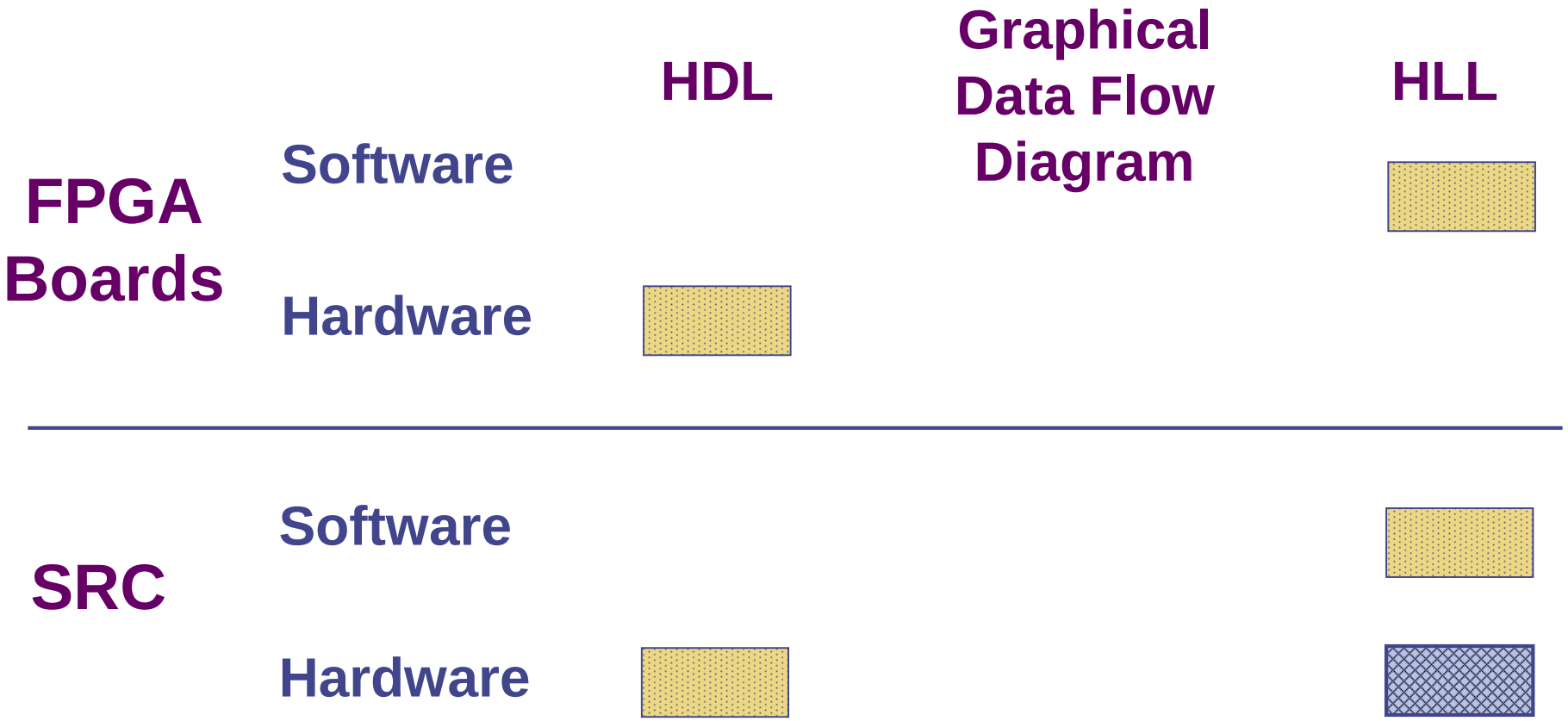
- ✓ close integration of the microprocessor system and the FPGA system
- ✓ integrated programming environment
- ✓ programming does not require hardware expertise
- ✓ suitable for a wide range of applications
- ✓ permits run-time reconfiguration of the FPGA system

SRC Hardware & Software

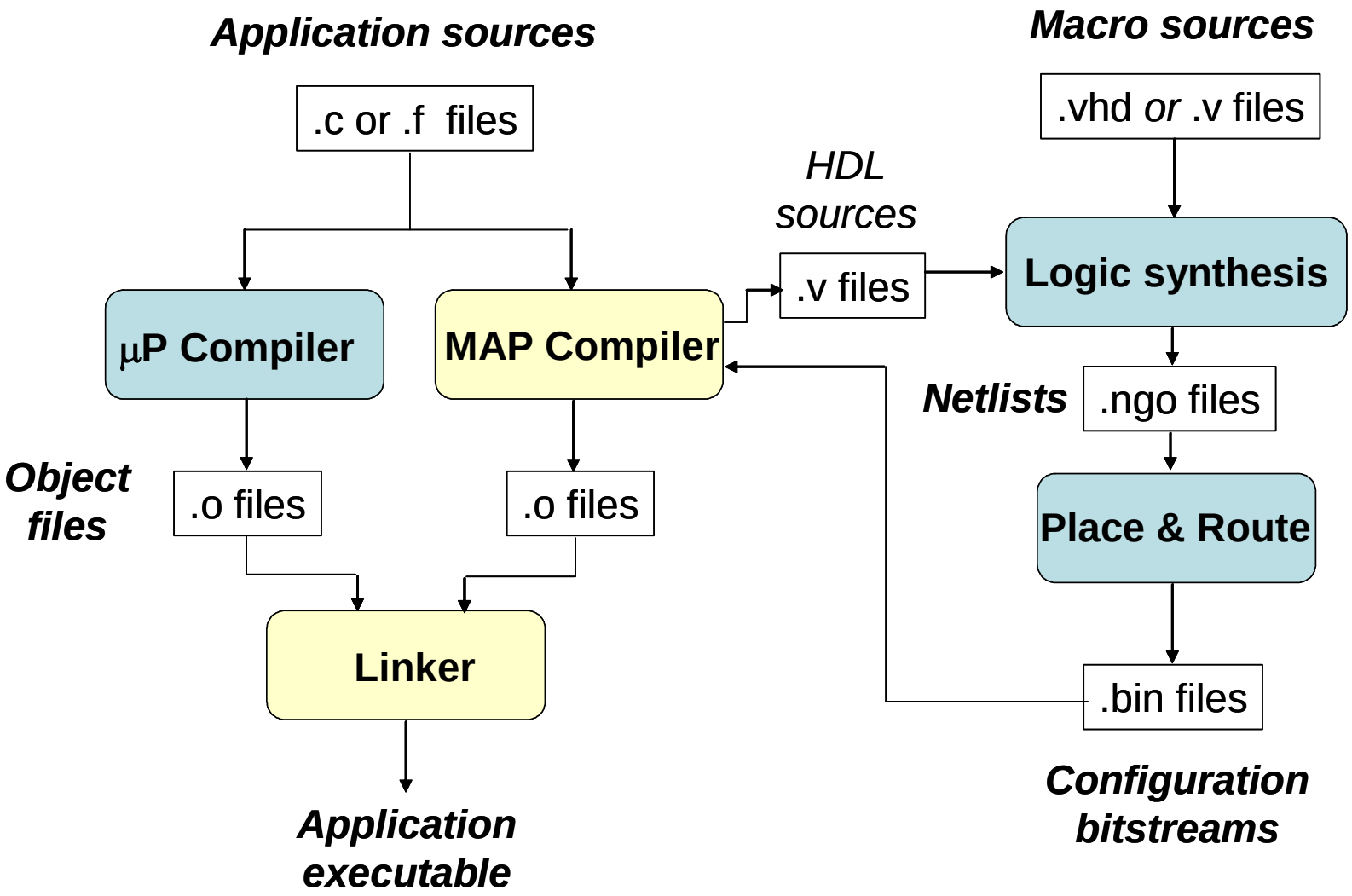
SRC Hardware Architecture



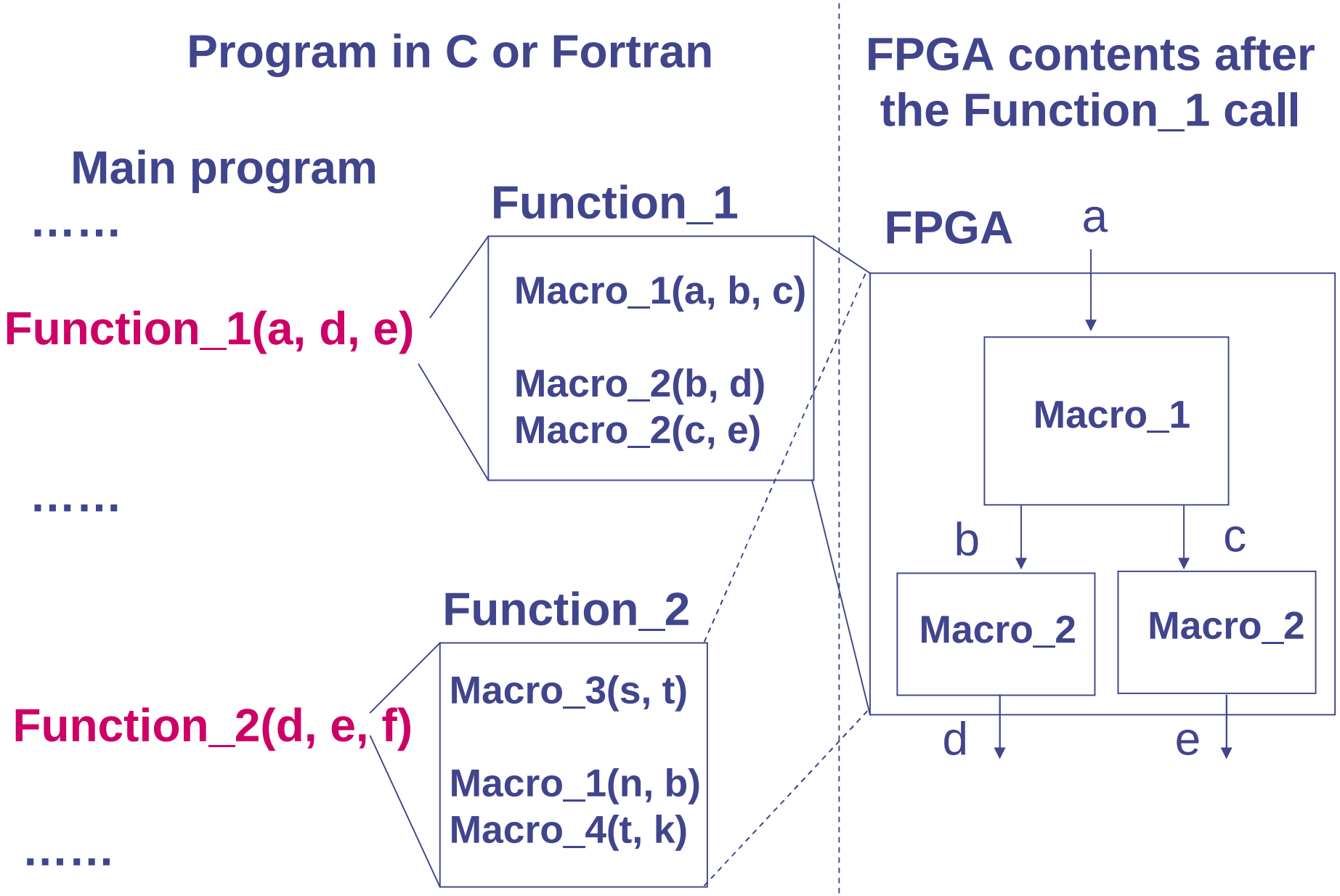
SRC vs. FPGA Accelerator Boards Programming



SRC Compilation Process



Run Time Reconfiguration in SRC

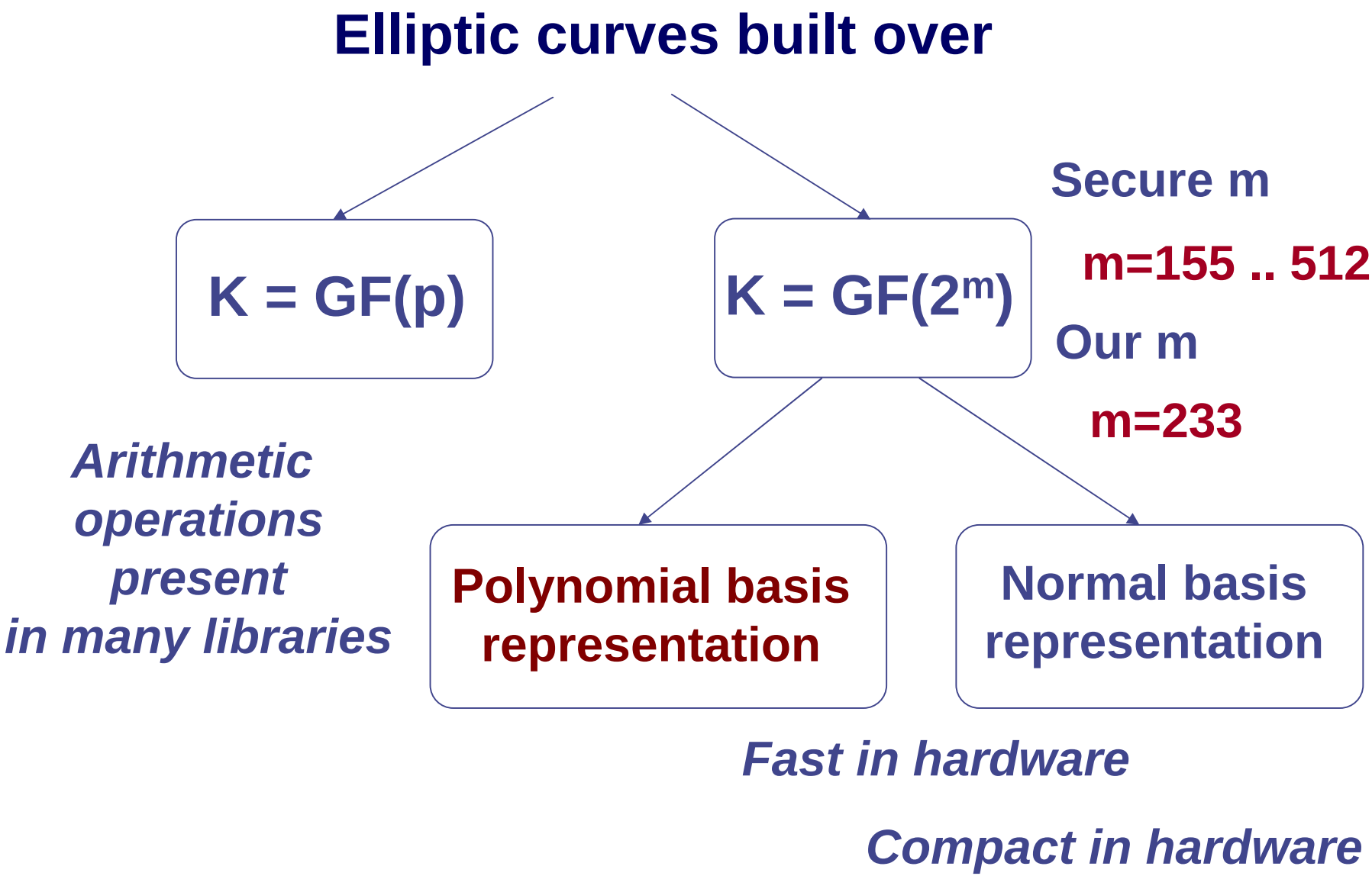


Elliptic Curve Cryptosystems

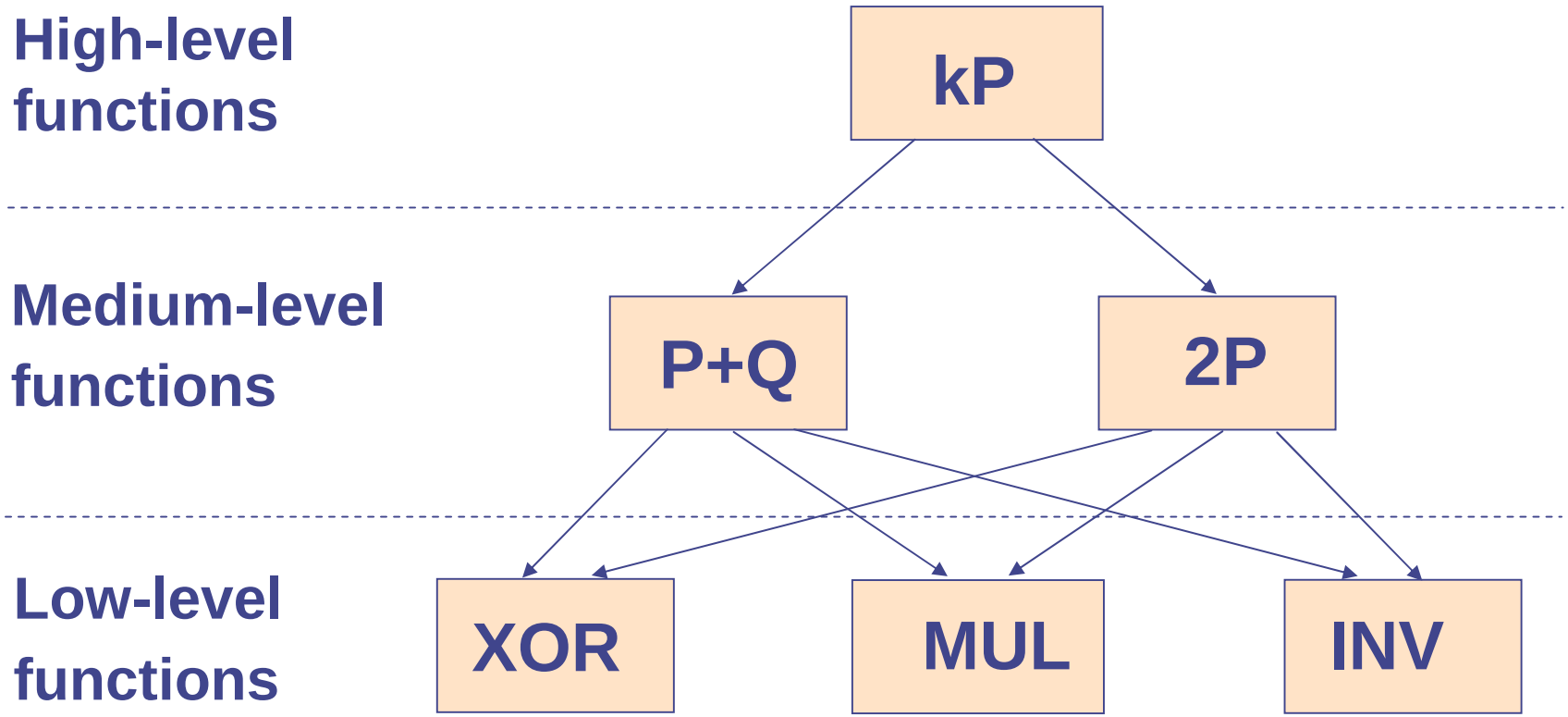
Elliptic Curve Cryptosystems

- ✓ public key (asymmetric) cryptosystems
- ✓ first true alternative for RSA
- ✓ several times shorter keys
- ✓ fast and compact implementations, in particular in hardware
- ✓ a family of cryptosystems, instead of a single cryptosystem

Three Classes of Elliptic Curves



ECC Hierarchy



Basic operations of Elliptic Curve Cryptosystems (1)

Basic operations in Galois Field GF(2^m)

- addition and subtraction (xor): $x+y$, $x-y$
- multiplication: $x \cdot y$
- inversion: x^{-1}

Basic operations on points of an Elliptic Curve

over Galois Field GF(2^m)

- addition of points: $P + Q$
- doubling a point: $2P$

where $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$

Basic operations of Elliptic Curve Cryptosystems (2)

Complex operations on points of an Elliptic Curve

over Galois Field GF(2^m)

- scalar multiplication: $k \cdot P = \underbrace{P + P + \ldots + P}_{k \text{ times}}$
- double scalar multiplication: $k \cdot P + l \cdot Q$

Addition, P+Q

$$R = P + Q$$

$P = (x_P, y_P)$
$Q = (x_Q, y_Q)$
$R = (x_R, y_R)$

$x_R = \lambda^2 + \lambda + x_P + x_Q + a_2$
$y_R = \lambda(x_P - x_R) - y_P$

where
$\lambda = (y_1 + y_2)(x_1 + x_2)^{-1}$

Number of field operations:

3 multiplications
1 inversion

Doubling, 2P

$$R = 2P$$

$P = (x_P, y_P)$
$R = (x_R, y_R)$

$x_3 = a_6(x_P^{-1})^2 + x_P^2$
$y_3 = x_P^2 + (x_P + y_P x_P^{-1})x_R + x_R$

a_2, a_6 – coefficients of a curve

Number of field operations:

5 multiplications
1 inversion

Scalar Multiplication - kP

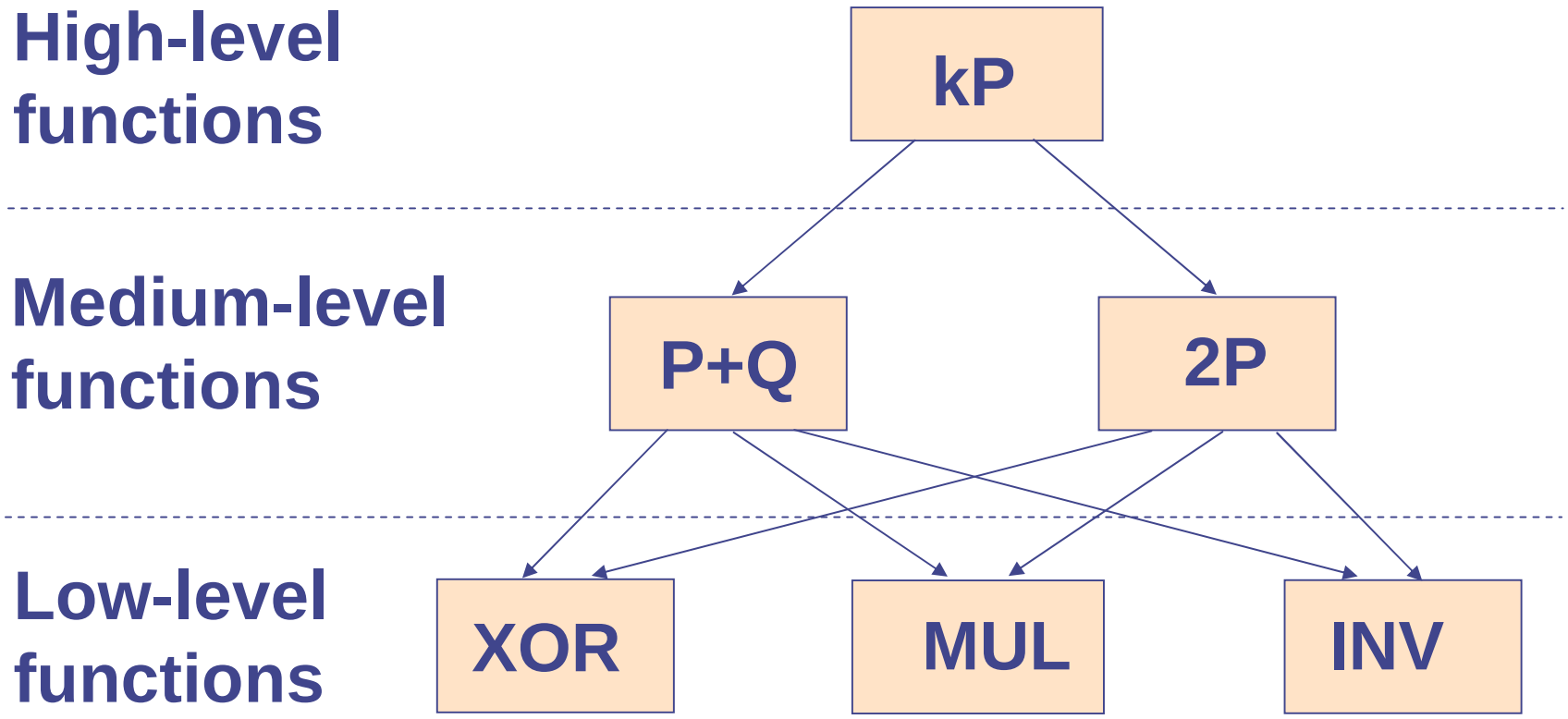
$$R = k\,P = \underbrace{P + P + \ldots + P}_{k \text{ times}}$$

$$k = (k_{m-1}, k_{m-2}, \ldots, k_1, k_0)_2$$

```
R = O
S = P
for ( i=0 to m-1 )
    if( k_i = 1 )
        R = R + S
    end if
    S = 2S
end for
return R
```

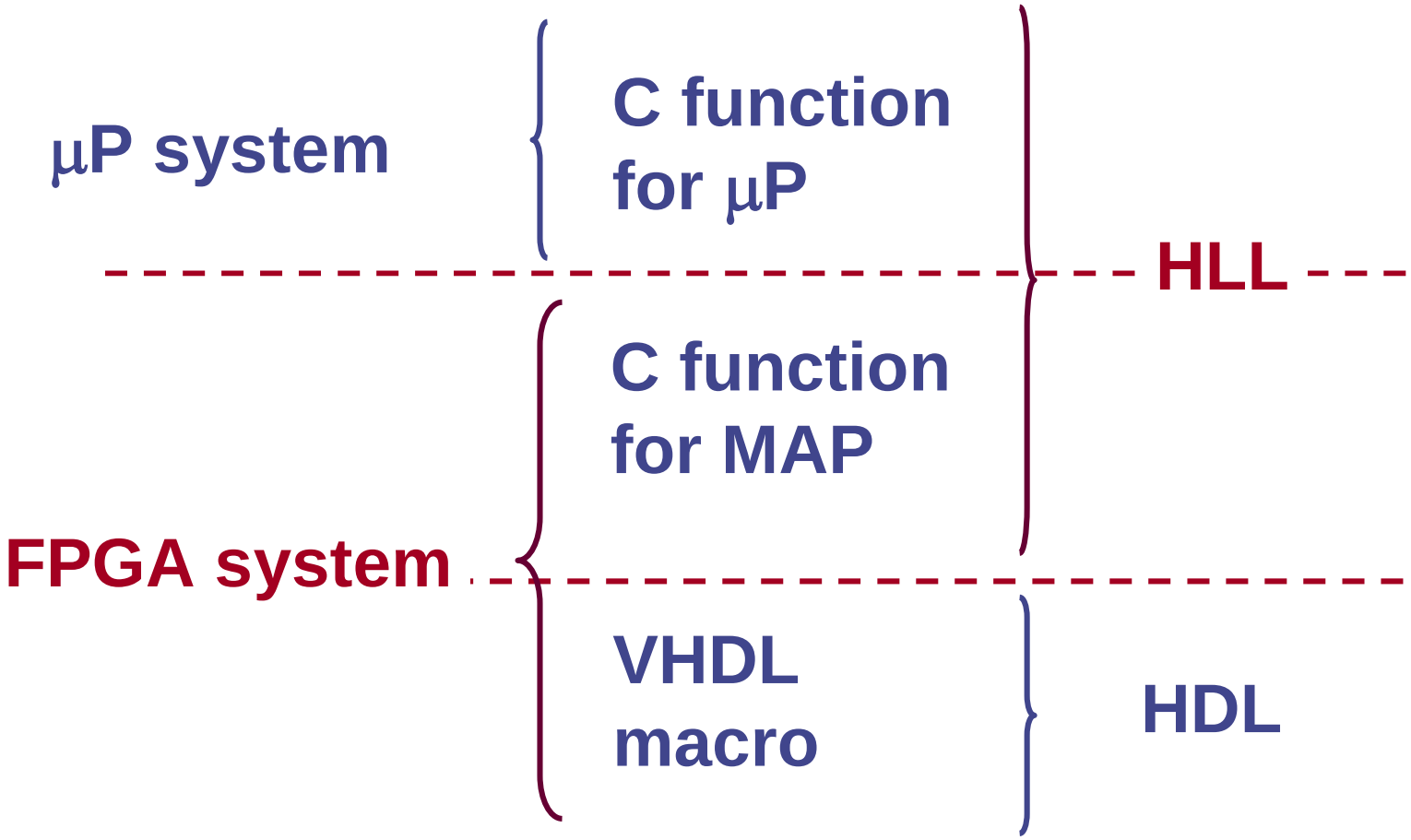
can be performed in parallel

ECC Hierarchy



Investigated Partitioning Schemes

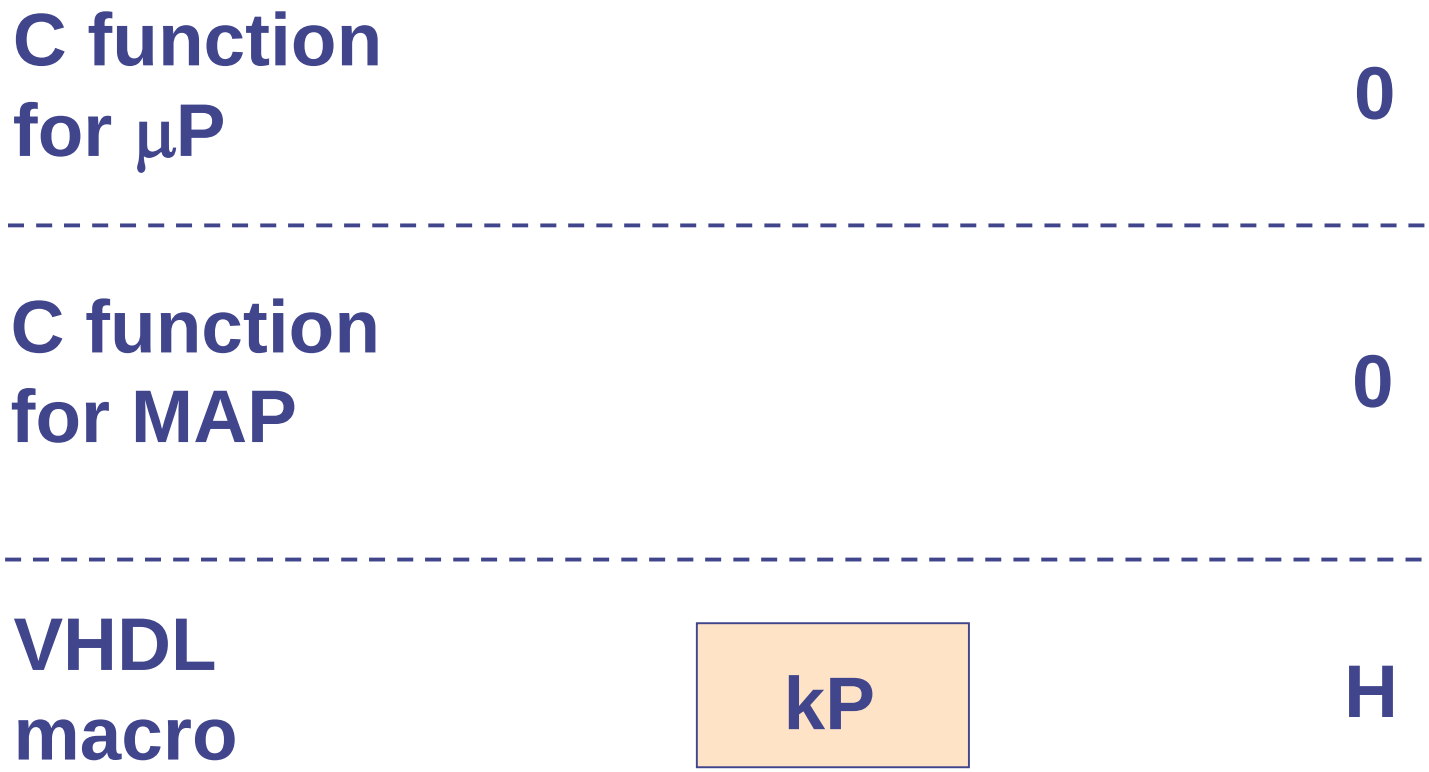
SRC Program Partitioning



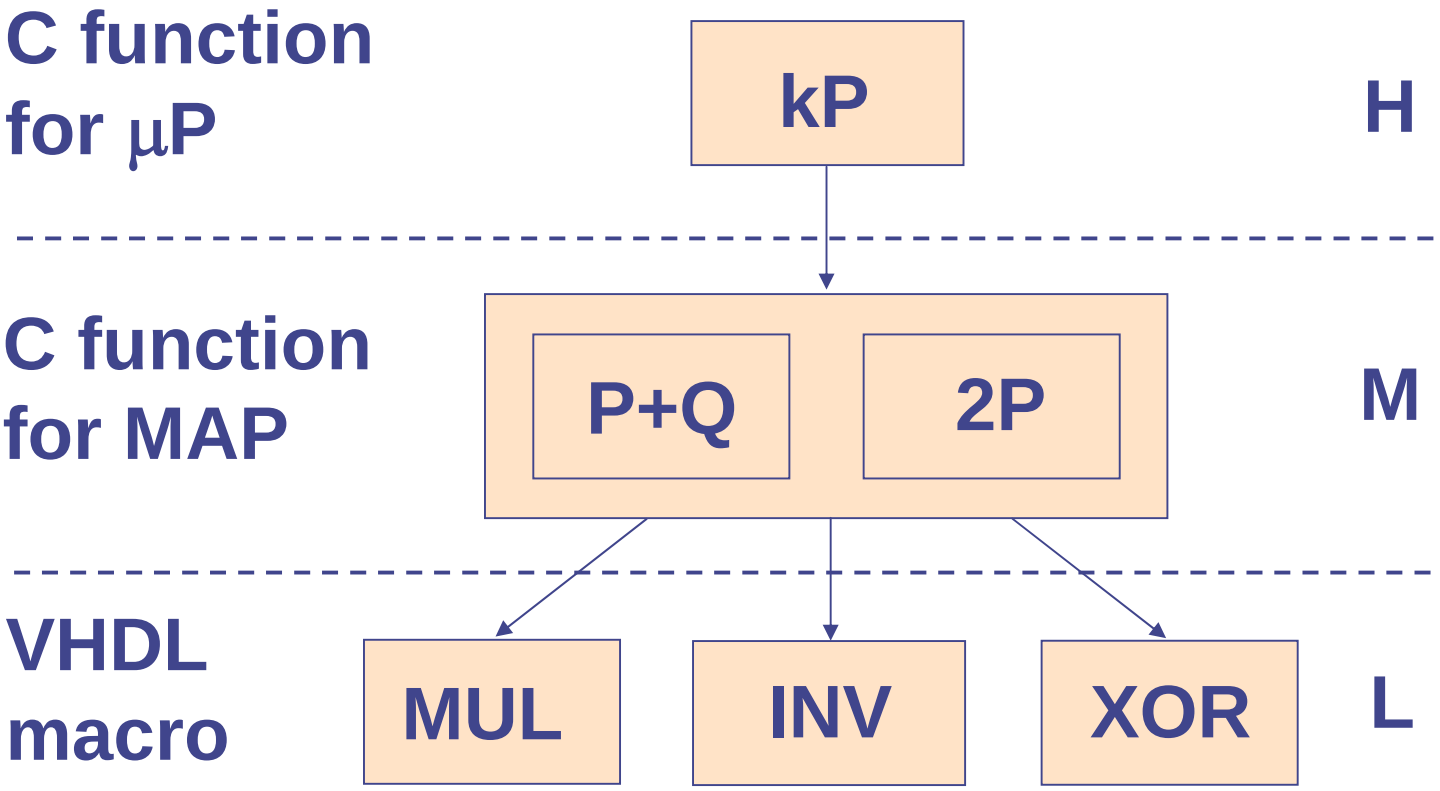
H00 Partitioning (µP Software Only)

C function for µP	kP	H
C function for MAP		0
VHDL macro		0

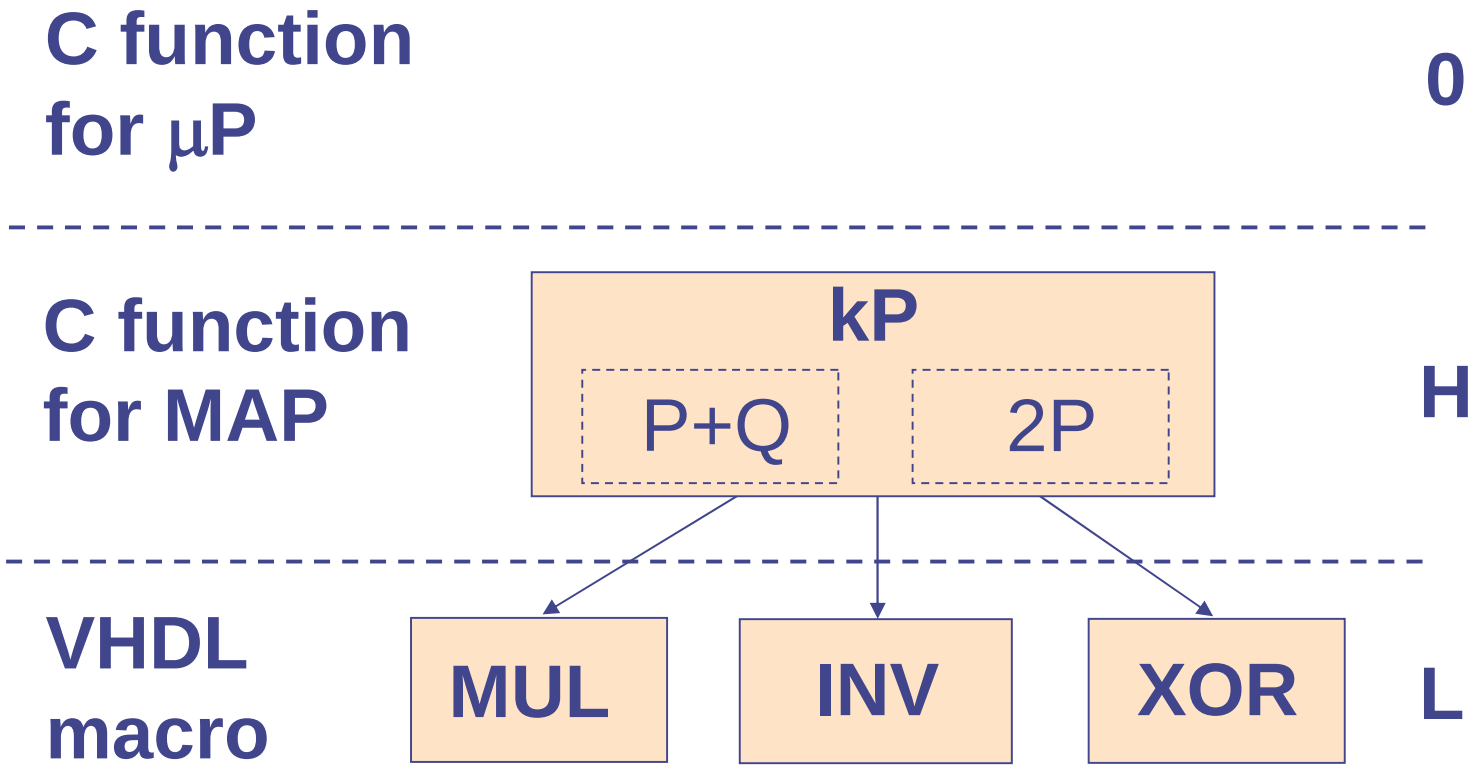
00H Partitioning (VHDL only)



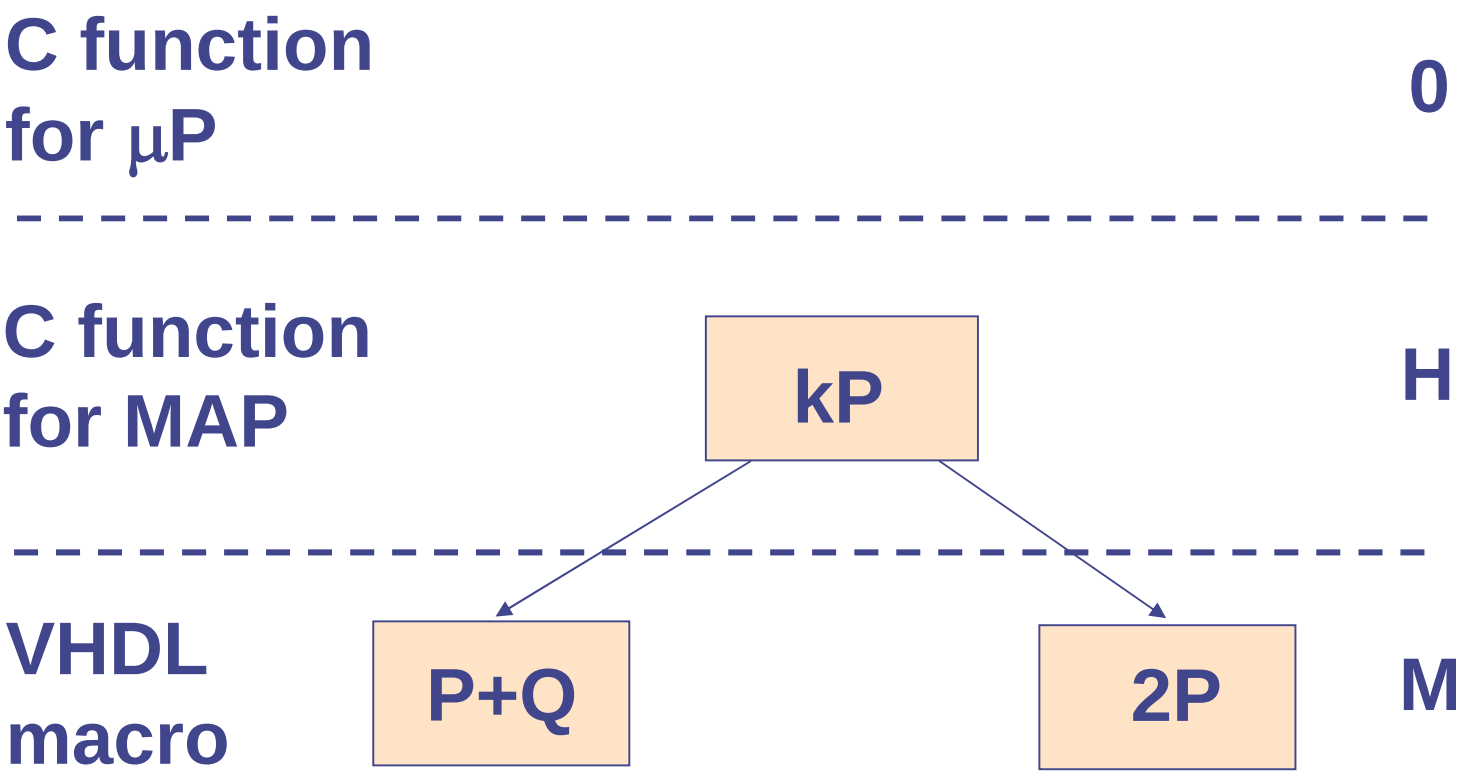
HML Partitioning



0HL Partitioning



0HM Partitioning



GF(2^m) Multiplier

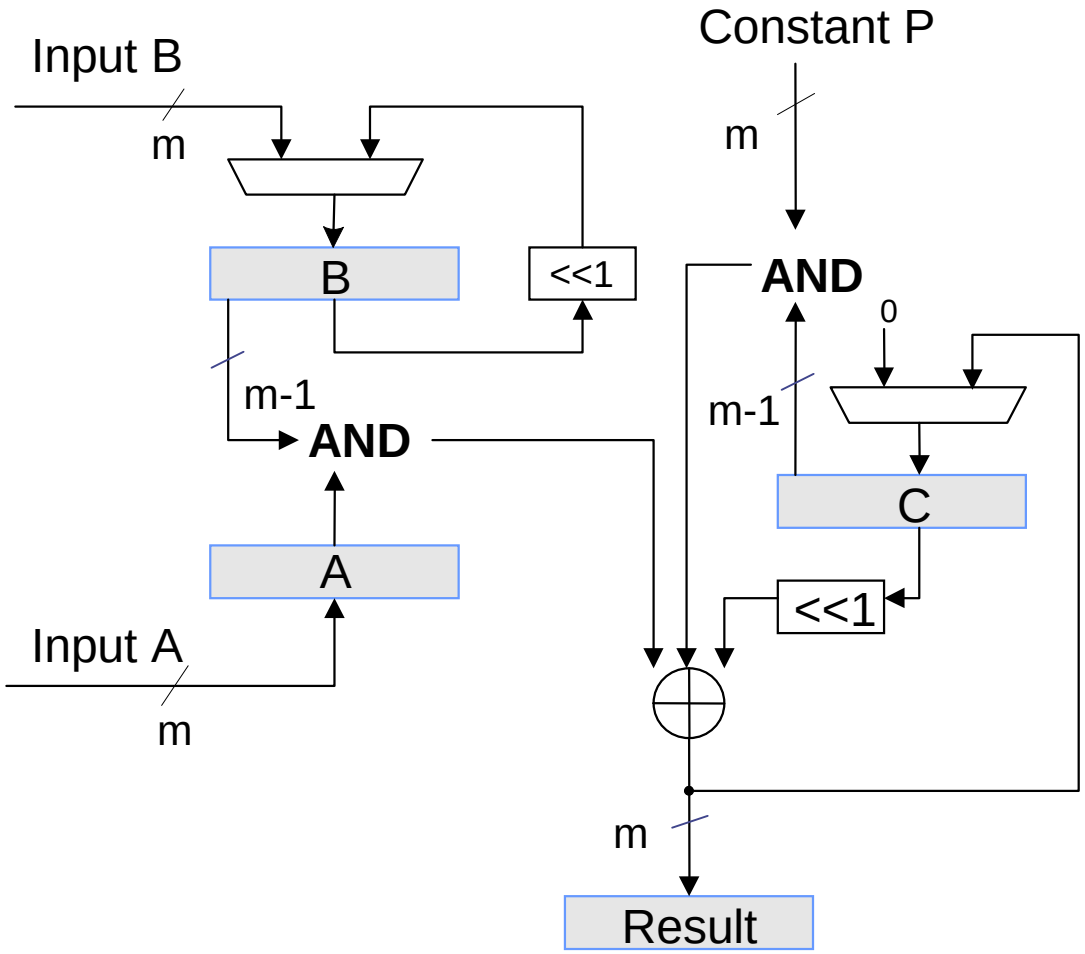
Input:

A, B $\in$ GF(2^m)

Output:

C = A*B mod P

1. C = 0
2. for i = m-1 to 0 do
3. C = C<<1 + A*b_i
4. C = C + c_m*P
5. end for
6. return C



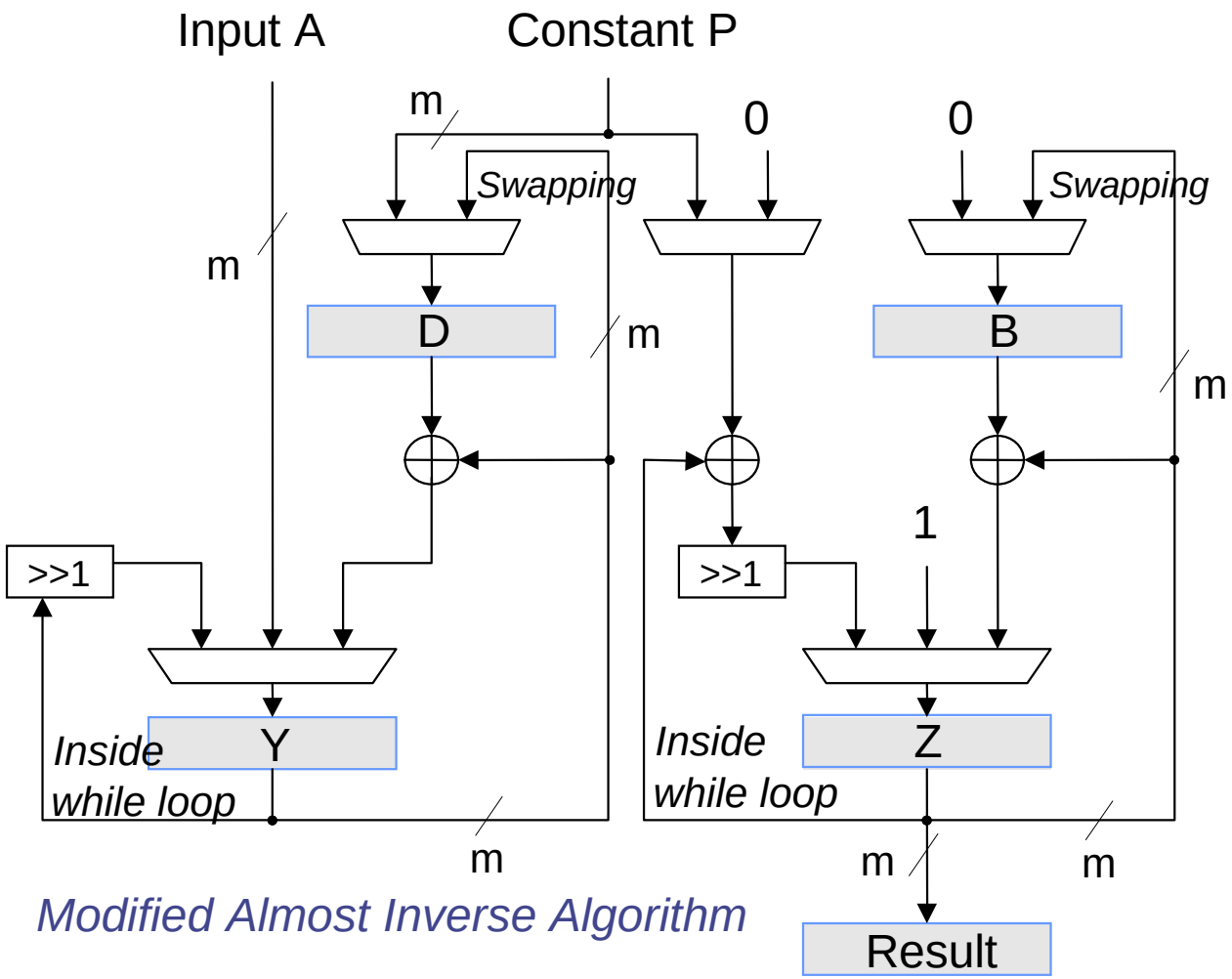
m+1 clock cycles per multiplication

GF(2^m) Inverter

Input: $A \in GF(2^m)$

Output: $C = A^{-1} \bmod P$

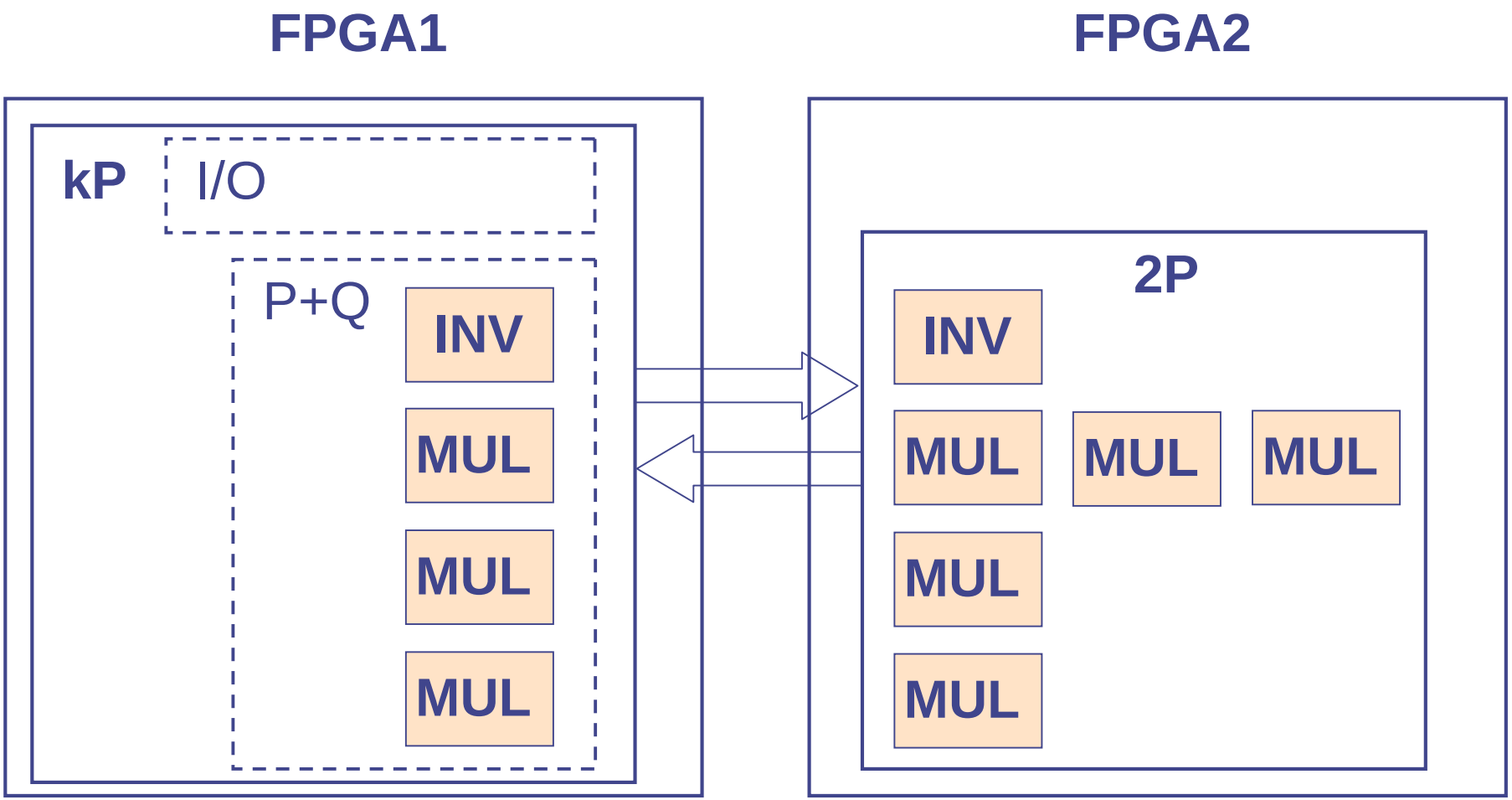
1. Y=A, D=P, B=0, Z=1
2. loop
3. while $y_0 = 0$ do
4. Y=Y>>1
   X=(X + z_0*P)>>1
5. end while
6. if (Y=1)
7. return Z
8. if (D>Y) then
9. D<=>Y, B<=>Z
10. Y=Y+D, Z=Z+B
11. end loop



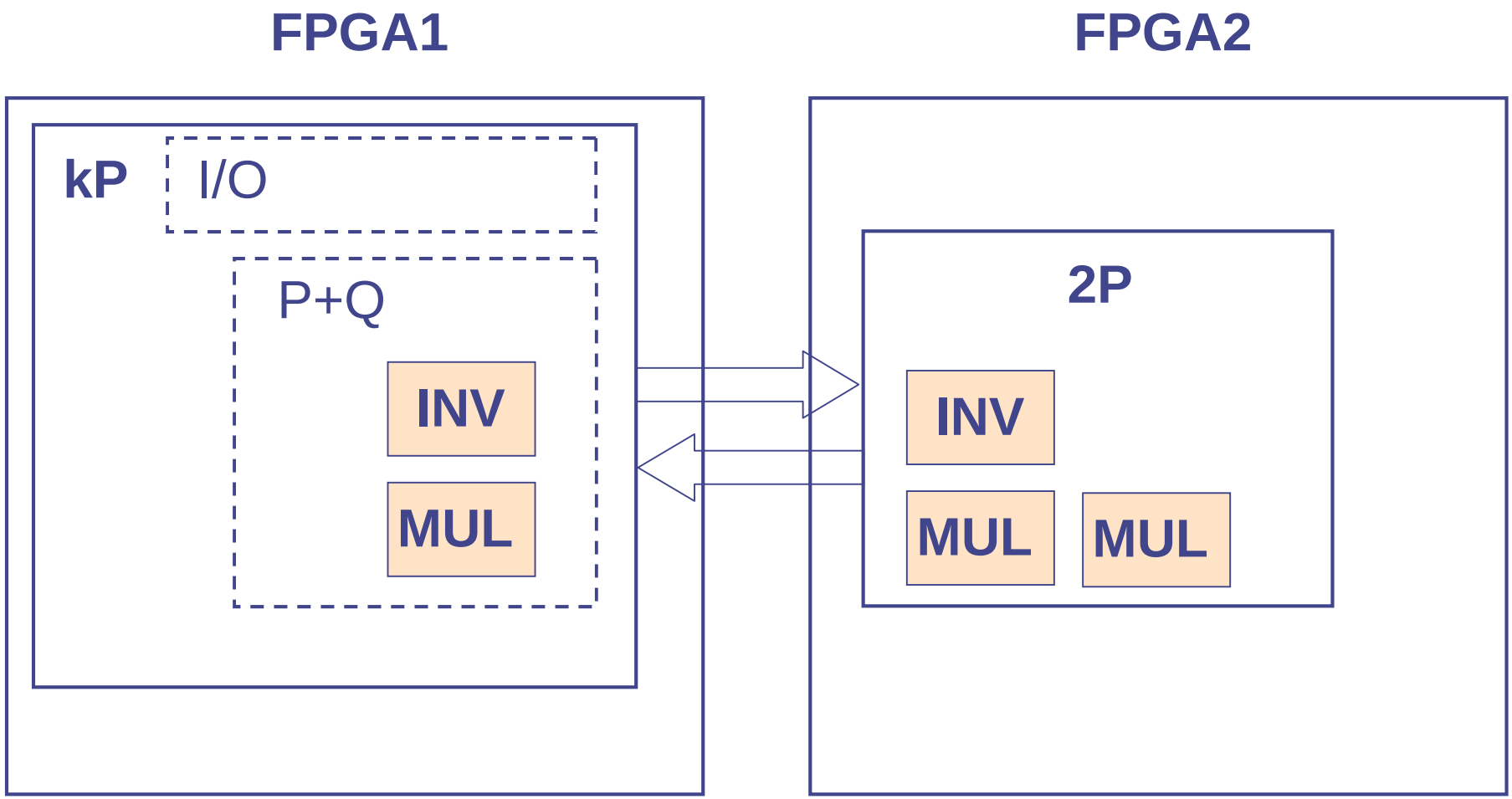
Modified Almost Inverse Algorithm

Time of inversion is input-dependent
Typically, 3-4 times m, on average

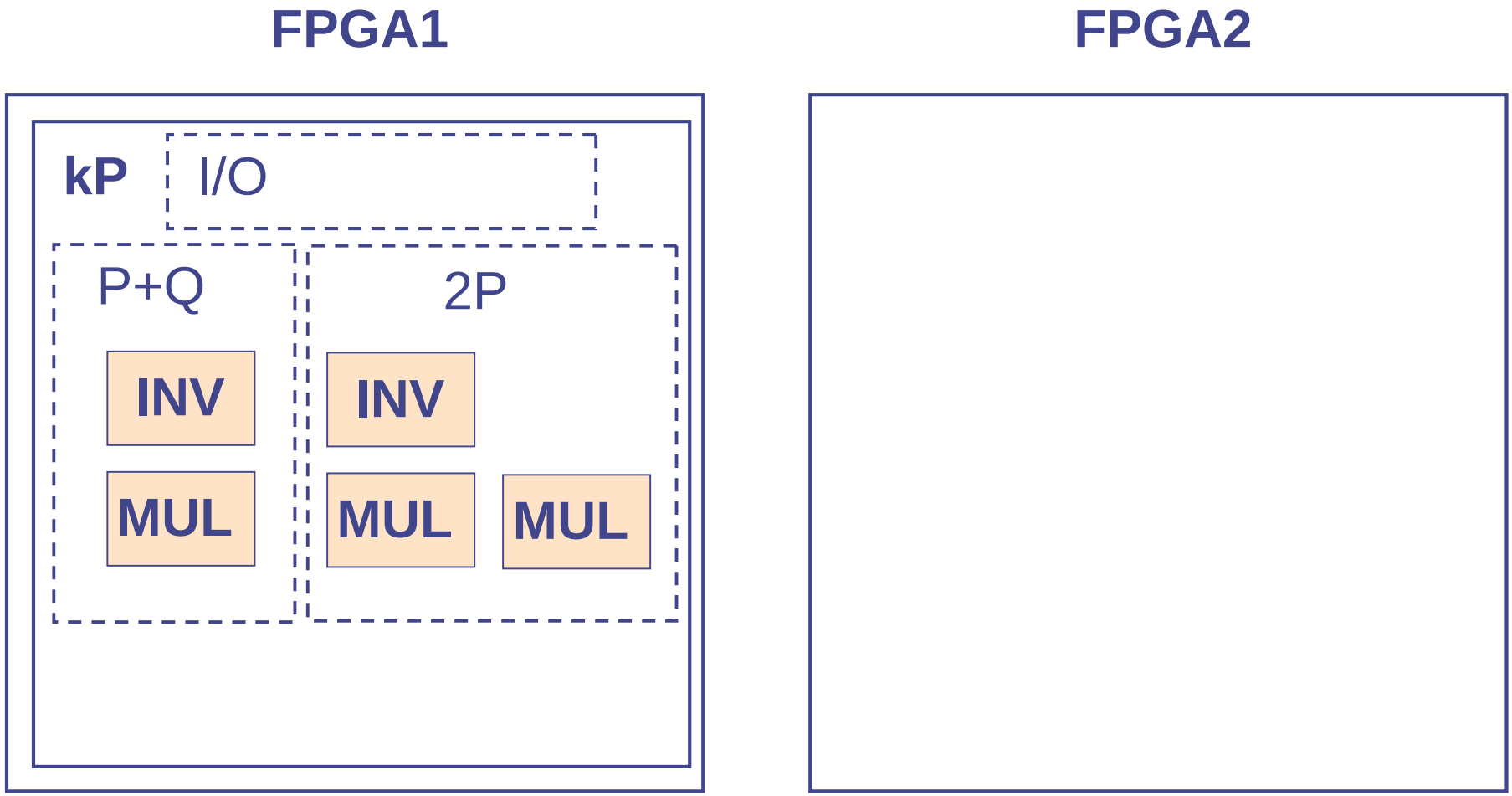
Unrolled Implementation Approach Using Two FPGA Devices



Iterative Implementation Approach Using Two FPGA Devices

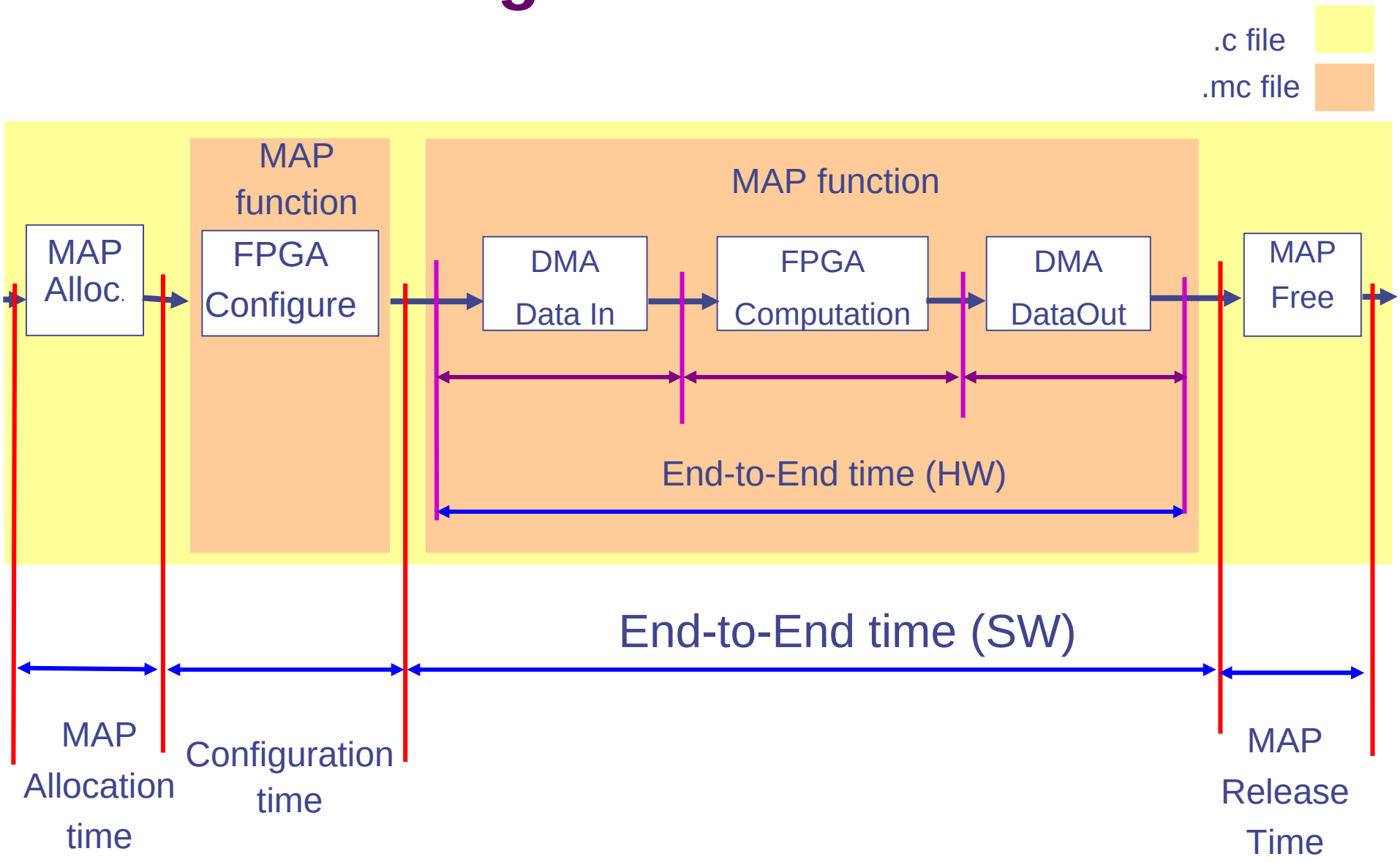


Iterative Implementation Approach Using One FPGA Device



Results

Timing Measurements



Timing measurements

Algorithm Parti-tioning Scheme	No. of FPGAs	End-to-End Time (µs)	DMA Data In Time (µs)	DMA Data Out Time (µs)	FPGA Compu-tation Time (µs)	Total Over-head (µs)	Speed -up vs. Softw are	Slow-down vs. VHDL macro
H00 software	N/A	**31,050**	N/A	N/A	N/A	N/A	1.00	9.27
HML	1 chip	99,867	9,651	2,945	8,659	**91,208**	0.31	29.82
	2 chips	101,145	9,683	2,751	3,710	**89,630**	0.31	30.2
0HL Iterative	1 chip	7,167	41	14	6,794	373	4.33	2.14
	2 chips	3,914	40	15	3,544	370	7.93	1.17
0HL Unrolled	2 chips	**3,615**	40	11	3,247	368	8.59	**1.08**
0HM	1 chip	4,936	41	13	4,565	371	6.29	1.47
	2 chips	3,522	41	10	3,154	368	8.82	1.05
00H VHDL	1 chip	**3,349**	42	12	**2,979**	370	9.27	1.00

Resource Utilization

Algorithm Parti-tioning Scheme	No. of FPGAs	% CLB slices (out of 33,792)	CLB slice # increase vs. pure VHDL	% LUTs (out of 67,580)	LUT # increase vs. pure VHDL	% FFs (out of 67,580)	FF # increase vs. pure VHDL
Software	N/A	0	N/A	0	N/A	0	N/A
HML	1 chip	61	1.56	29	1.32	53	1.71
	2 chips	57+48 = 105	2.69	19+15 = 34	1.55	40+31 = 71	2.29
0HL Iterative	1 chip	62	**1.59**	30	**1.36**	52	**1.68**
	2 chips	59+48 = 107	**2.74**	19+16 = 35	**1.59**	40+31 = 71	**2.29**
0HL Unrolled	2 chips	79+49 = 128	**3.28**	20+14 = 34	**1.55**	44+28 = 72	**2.32**
0HM	1 chip	46	**1.18**	23	**1.05**	36	**1.16**
	2 chips	37+14 = 51	**1.31**	17+9 = 26	**1.18**	31+11 = 42	**1.35**
VHDL only	1 chip	39	1.00	22	1.00	31	1.00

Number of lines of code

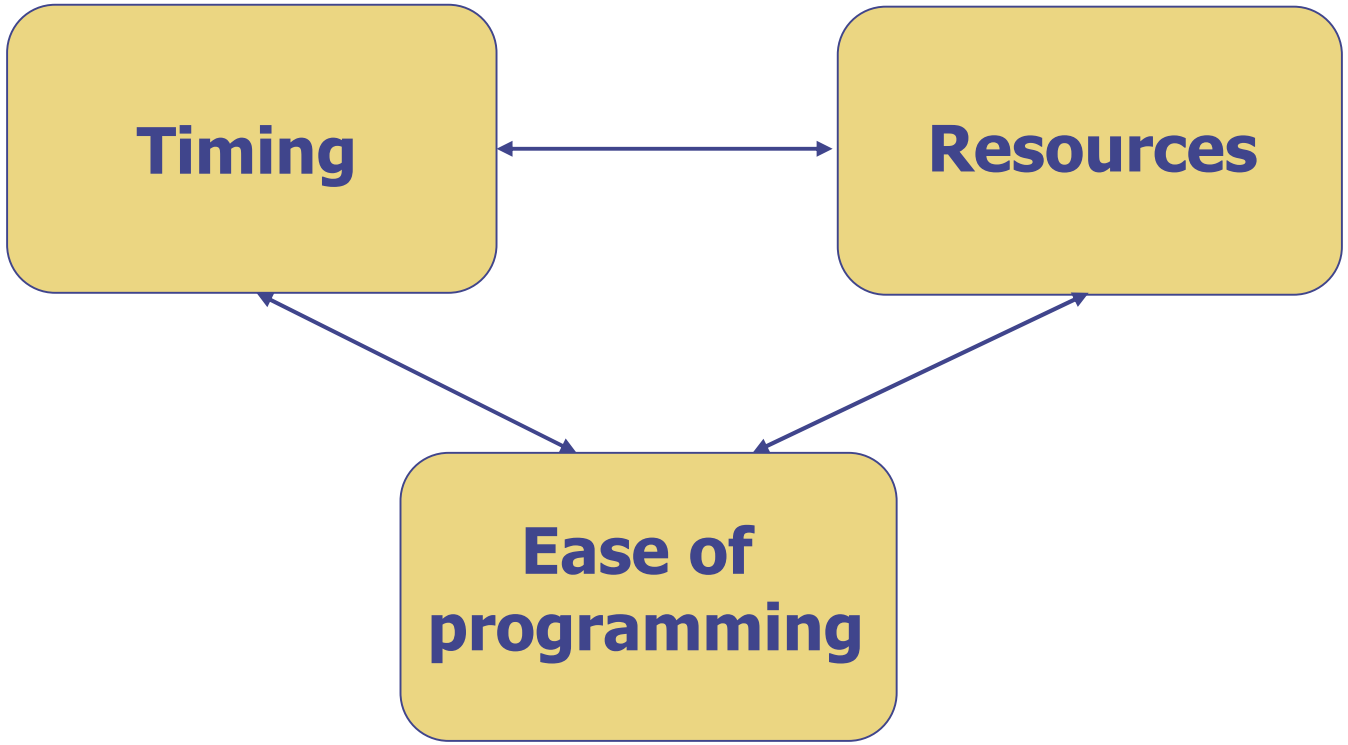
Algorithm Partitioning Scheme	No. of FPGAs	VHDL	Macro Wrapper	MAP C	Main C
Software	N/A	N/A	N/A	N/A	726
HML	1 chip	654	146	195	192
	2 chips	**654**	146	**236**	192
0HL Iterative	1 chip	654	146	240	143
	2 chips	654	146	290	143
0HL Unrolled	2 chips	**683**	304	**349**	145
0HM	1 chip	1,372	213	145	143
	2 chips	**1,372**	213	**188**	143
00H	1 chip	**1,532**	107	**73**	143

Conclusions

- Elliptic Curve Cryptosystem implementation challenging for reconfigurable computers because of
  - optimization for latency rather than throughput
  - limited amount of parallelism
- From 8 to 9 times speed-up over highly optimized microprocessor implementation demonstrated using four different algorithm partitioning schemes
  - 0HL iterative 2-chip
  - 0HL unrolled 2-chip
  - 0HM 2-chip
  - 00H 1-chip

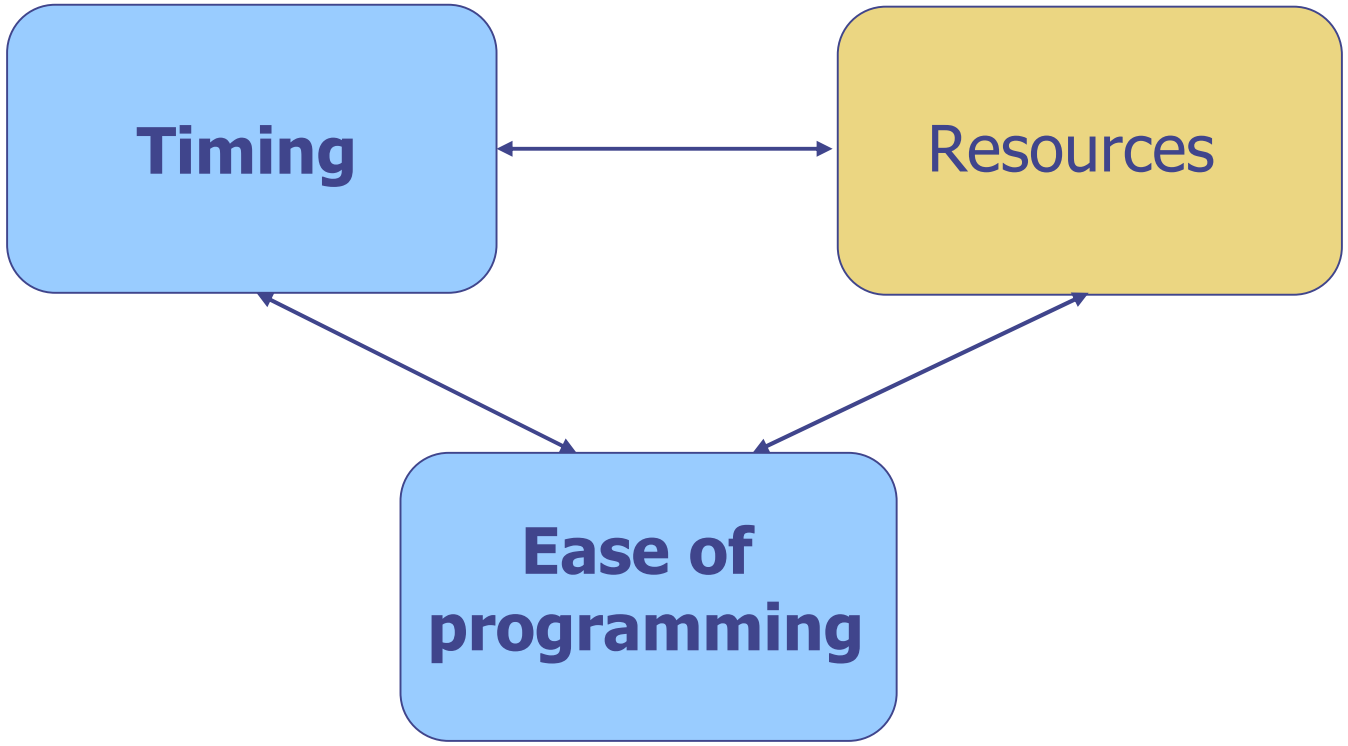
Conclusions – cont.

Clear trade-offs:



Conclusions – cont.

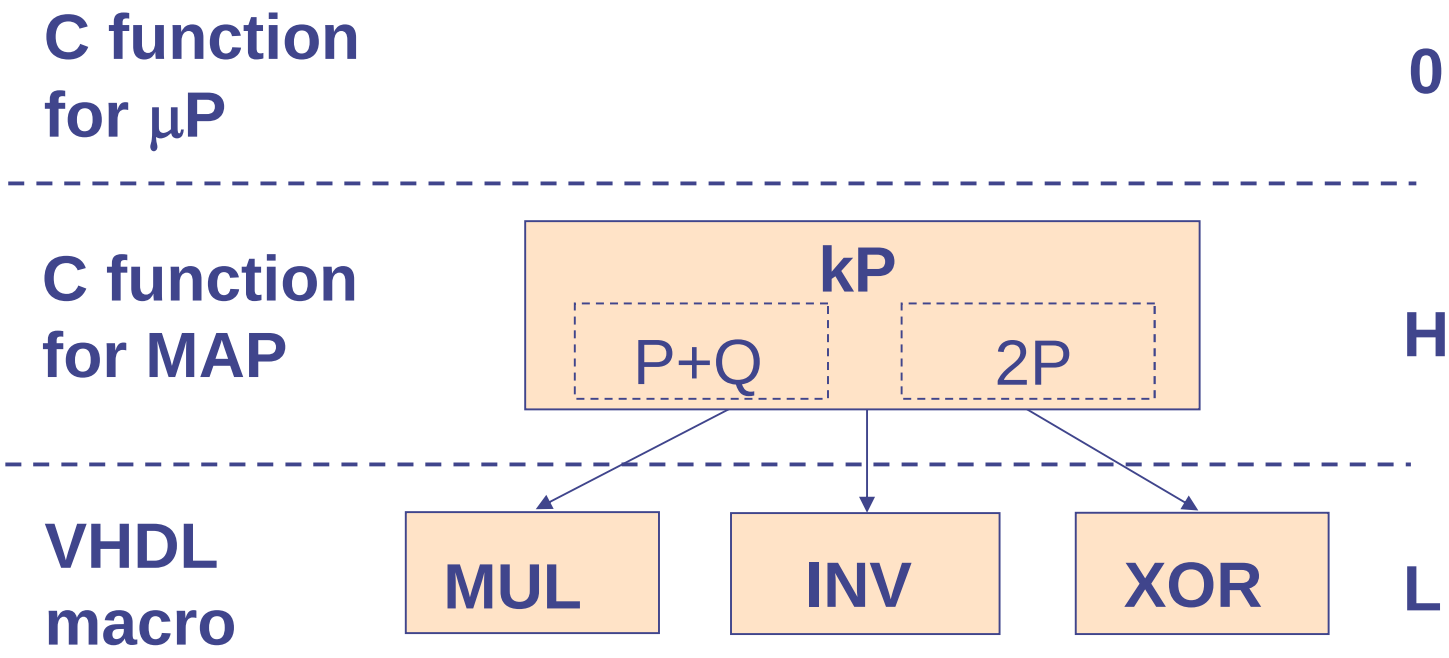
Assuming focus on:



Conclusions – cont.

The best implementation approach:

OHL partitioning scheme, 2-chip, unrolled



Only 8% increase in the execution time compared to pure VHDL