28. Februar 2008

CoProcessor Design for Crypto-Applications using Hyperelliptic Curve Cryptography

Alexander Klimm, Oliver Sander, Jürgen Becker
Institut für Technik der Informationsverarbeitung

Sylvain Subileau, *Daimler AG*

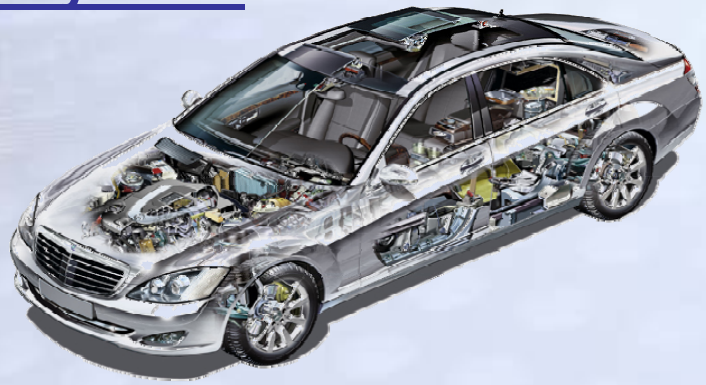
Overview

- Motivation
- Public Key Cryptography
- Hyperelliptic Curve Cryptography (HECC)
- Hardware/Software Codesign for HECC on a Xilinx FPGA
- Measurements and Evaluation
- Outlook & future Work

Motivation

Increased need for security of embedded systems

- Increasing number of embedded devices
  - *cell phones, PDAs, ECUs, etc.*
- Networks of embedded devices
  - *ECUs, ubiquitous computing, etc.*
- Applications need secure systems
  - *chip tuning, c2x, toll systems, etc.*
- Communication and data transfers need to be secured.



Industry very cost driven

- Small Platforms
- low computation power
- small memory space
- short time-to-market

Motivation

Public Key Cryptography (PKC) in Embedded Systems

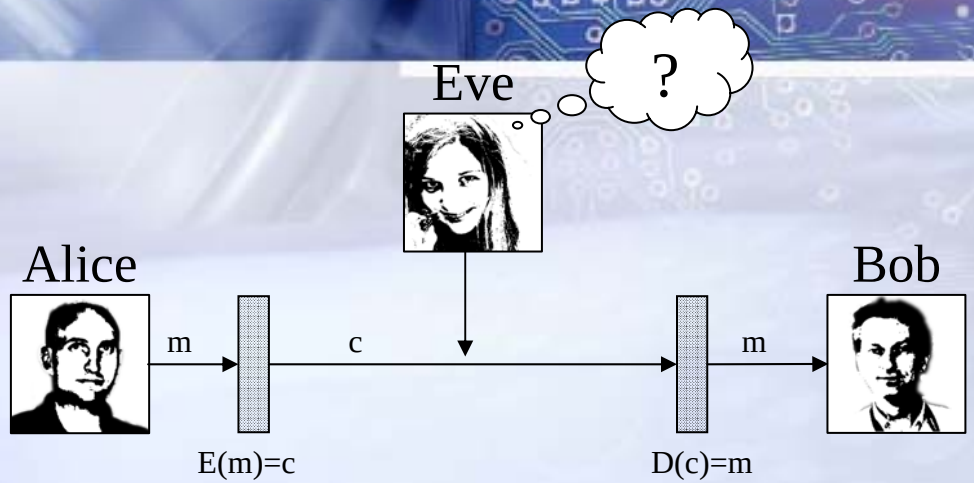
- **Advantages:**
  - Less storage memory for keys needed
  - Secrets stay inside an entity
    - If one entity is compromised the others remain still secure
    - Easier logistics

RSA standard for PKC:
- Long keys (1024 bit)
- Probably not secure enough in the future

ECC & HECC:
- Smaller keys (163 bit) and same level of security as RSA
- computational intensive algorithms

Goal:
- HECC based protocol (few patents)
- Acceleration of used algorithms
- Small hardware platform (FPGA)

Public-Key Cryptography



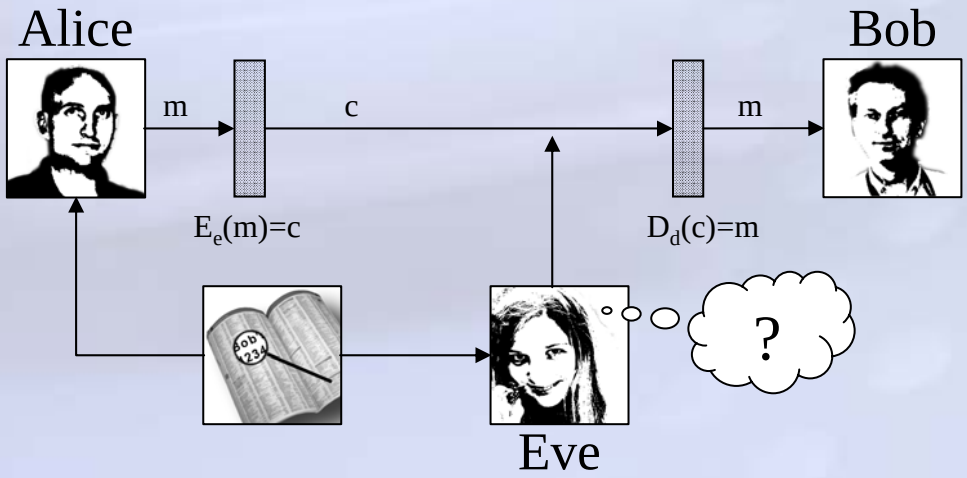
Basic Crypto:

Alice encrypts a message with a key.

Bob decrypts the received cyphertext with his key that matches the encryption key.

Disadvantage:

Alice needs a key for every possible communication partner.

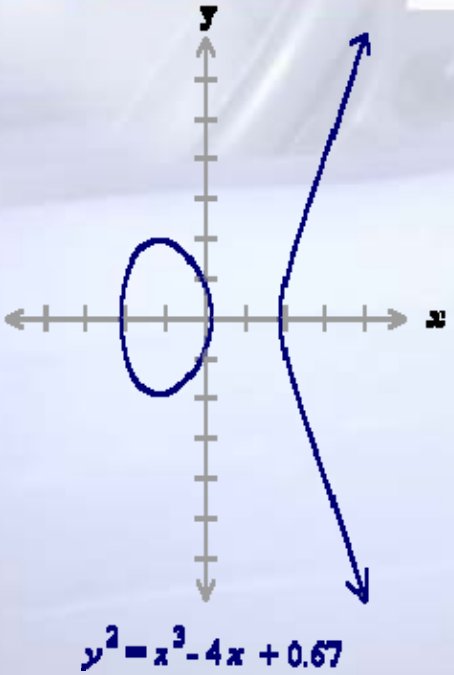


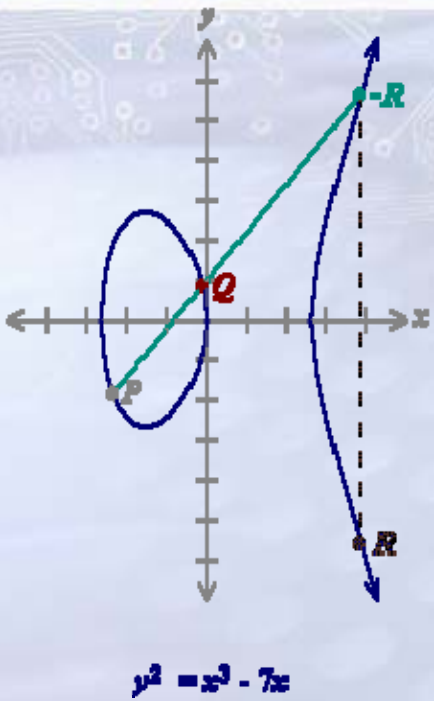
Public-Key Crypto:

All encryption-keys are public.

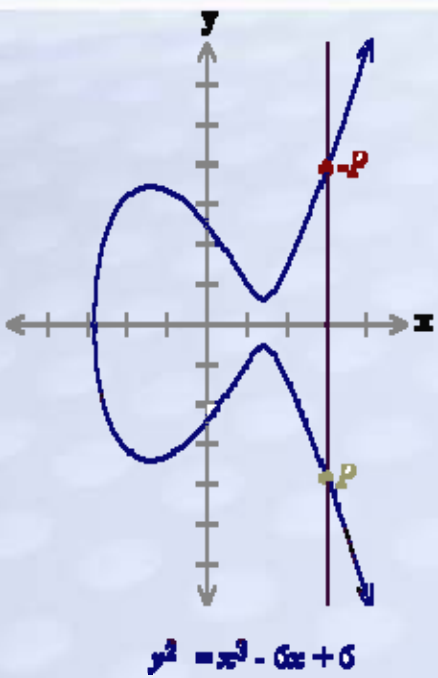
Only Bob can decrypt a message that is meant for him with his secret private key.

Elliptic Curves
- PointAdd -





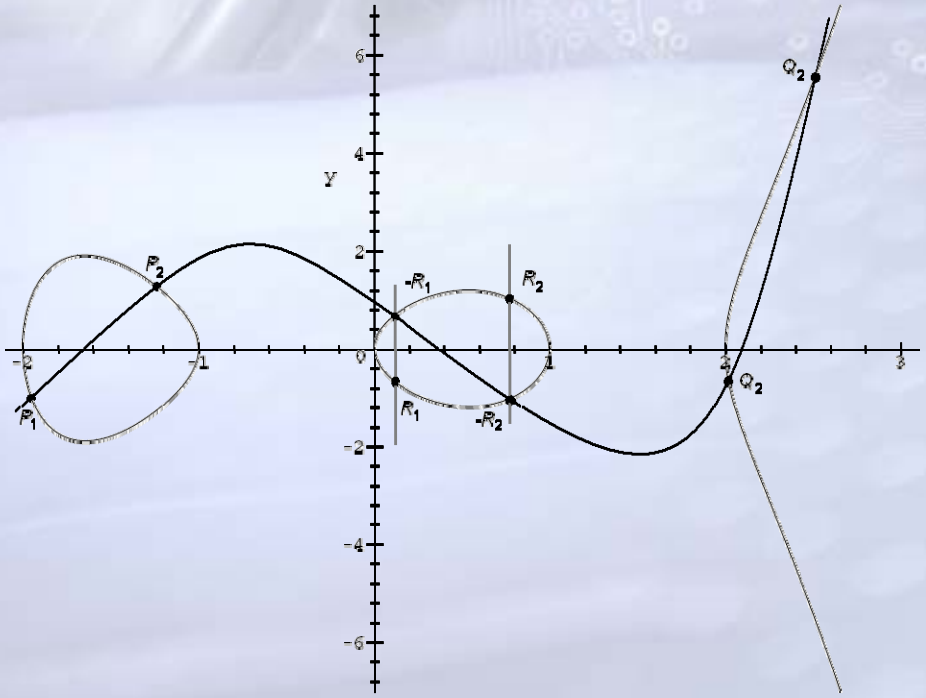
P+Q = R



P+(-P) = O
O: defined as Point of Infinity

Quelle: Certicom Corp.

HECC – Hyperelliptic Curve Cryptography
- PointAdd -



$$C_2 : v^2 = u^5 - 5u^3 + 4u + 3$$

PointAdd Algorithmus $(g = 2 \text{ und } \deg u_1 = \deg u_2 = 2)$

Eingabe: Zwei Divisorenklassen $[u_1, v_1], [u_2, v_2]$ mit $u_i = x^2 + u_{i1}x + u_{i0}$ und $v_i = v_{i1}x + v_{i0}$

Ausgabe: Die Divisorenklasse $[u', v'] = [u_1, v_1] \oplus [u_2, v_2]$

1. **Berechne:** $r = \mathrm{Res}(u_1, u_2)$

   $z_1 \leftarrow u_{11} - u_{21}$, $z_2 \leftarrow u_{20} - u_{10}$, $z_3 \leftarrow u_{11} z_1 + z_2$ und $r \leftarrow z_2 z_3 + z_1^2 u_{10}$
2. **Berechne:** $inv = (r / u_2) \bmod u_1$

   $inv_1 \leftarrow z_1$ und $inv_0 \leftarrow z_3$
3. **Berechne:** $s' = rs = ((v_1 - v_2) inv) \bmod u_1$

   $w_0 \leftarrow v_{10} - v_{20}$, $w_1 \leftarrow v_{11} - v_{21}$, $w_2 \leftarrow inv_0 w_0$ und $w_3 \leftarrow inv_1 - w_1$

   $s_1' \leftarrow (inv_0 + inv_1)(w_0 + w_1) - w_2 - w_3(1 + u_{11})$ und $s_0' \leftarrow w_2 - u_{10} w_3$

   wenn $s_1' = 0$ dann siehe unten.
4. **Berechne:** $s'' = x + s_0 / s_1 = x + s_0' / s_1'$ und s_1

   $w_1 \leftarrow (r s_1')^{-1}$, $w_2 \leftarrow r w_1$ und $w_3 \leftarrow s_1'^2 w_1$

   $w_4 \leftarrow r w_2$, $w_5 \leftarrow w_4^2$ und $s_0'' \leftarrow s_0' w_2$

   wir haben $w_1 = 1/r^2 s_1$, $w_2 = 1/s_1'$, $w_3 = s_1$ und $w_4 = 1/s_1$
5. **Berechne:** $l' = s'' u_2 = x^3 + l_2' x^2 + l_1' x + l_0'$

   $l_2' \leftarrow u_{21} + s_0''$, $l_1' \leftarrow u_{21} s_0'' + u_{20}$ und $l_0' \leftarrow u_{20} s_0''$
6. **Berechne:** $u' = (s(l + h + 2v_2) - t)/u_1 = x^2 + u_1' x + u_0'$

   $u_0' \leftarrow (s_0'' - u_{11})(s_0'' - z_1 + h_2 w_4) - u_{10}$

   $u_0' \leftarrow u_0' + l_1' + (h_1 + 2v_{21}) w_4 + (2u_{21} + z_1 - f_4) w_5$

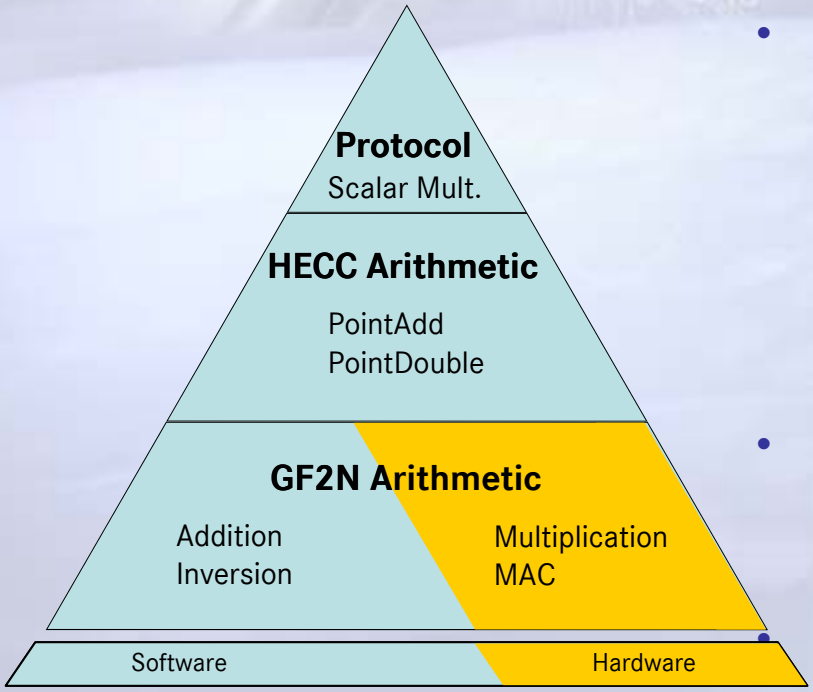
   $u_1' \leftarrow 2s_0'' - z_1 + h_2 w_4 - w_5$
7. **Berechne:** $v' = (-h - (l + v_2)) \bmod u' = v_1' x + v_0'$

   $w_1 \leftarrow l_2' - u_1'$, $w_2 \leftarrow u_1' w_1 + u_0' - l_1'$ und $v_1' \leftarrow w_2 w_3 - v_{21} - h_1 + h_2 u_1'$

   $w_2 \leftarrow u_0' w_1 - l_0'$ und $v_0' \leftarrow w_2 w_3 - v_{20} - h_1 + h_2 u_0'$

Gebe zurück: $[u', v']$

HW/SW Codesign
- Design Approach -



- HECC Implementations so far only in SW
- SW Implementions are too slow:
  Duration of one Scalar Multiplication:
  Freescale Star12 (16 Bit, 16 MHz): > 5000 ms
  PowerPC (32 Bit, 80 MHz): > 500 ms
  Optimized Code ca. 100 ms

  Goal: ≤ 50 ms
- Implementation on MicroBlaze (32 Bit, 33 MHz) and outsourcing of time-consuming calculations to HW

Evaluation of performance and adaptation of HW/SW

System Components

FPGA Gatewayplatform Spartan3-5000 (33 MHz)

Microcontroller

- MicroBlaze (32 bit)

opb_uartlite

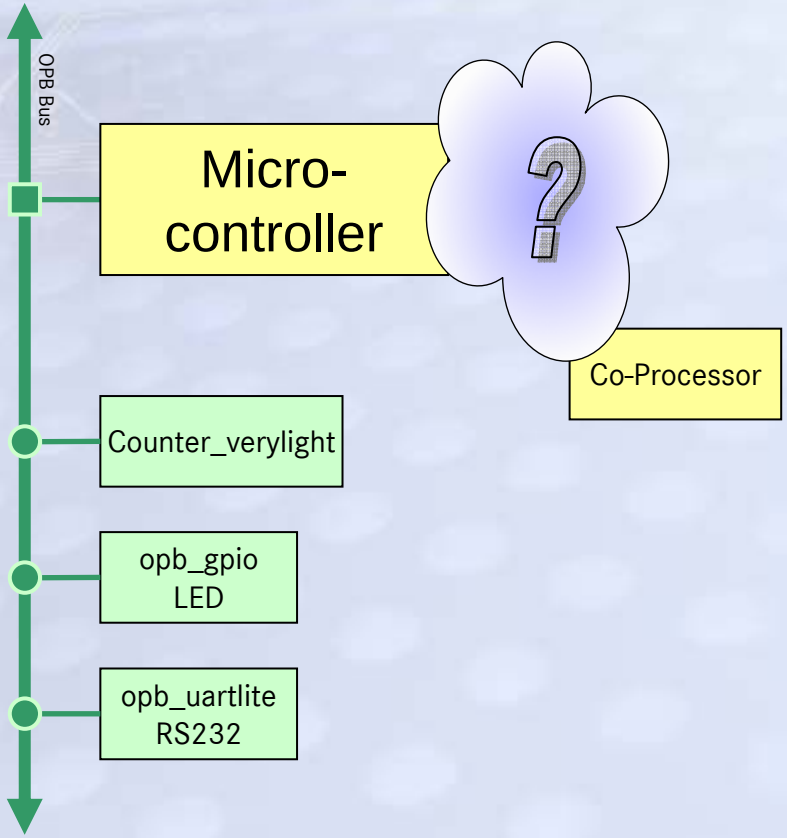
- Output of Data to PC (i.e. Testlogs)

opb_gpio: LED

- Debug outputs

Counter-verylight

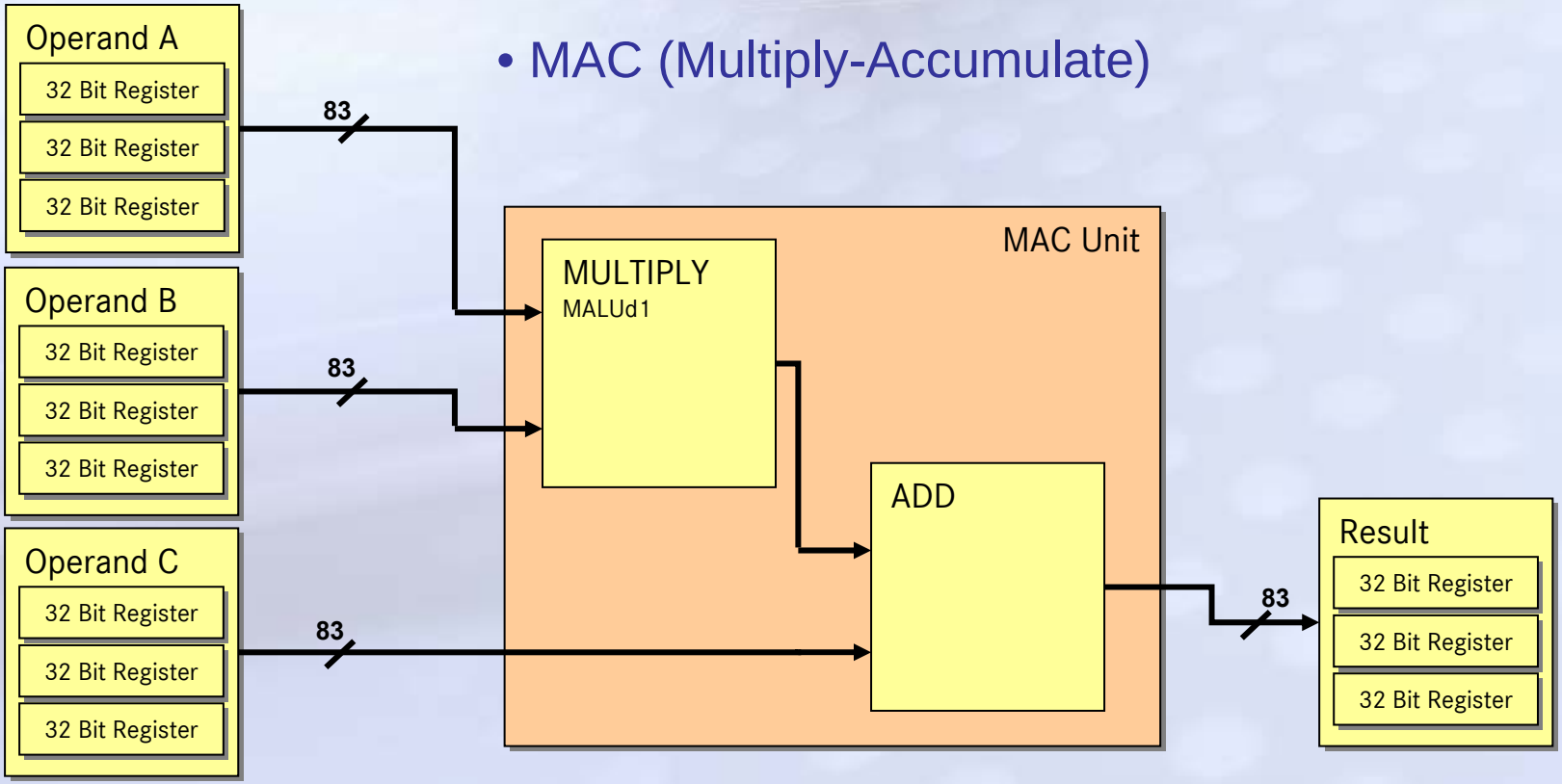
- a precise Counter to count the system‘s clock-cycles



CoProzessordesign
- Hardware Modules -

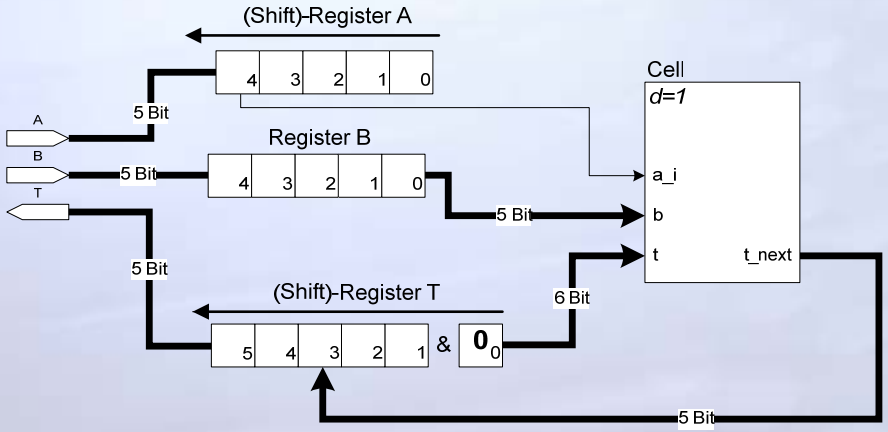
Hardware Units used:

- GF(2^n) Multiplier - MALUd1
- GF-Add (XOR-Logic of two 83 Bit input signals)
- MAC (Multiply-Accumulate)



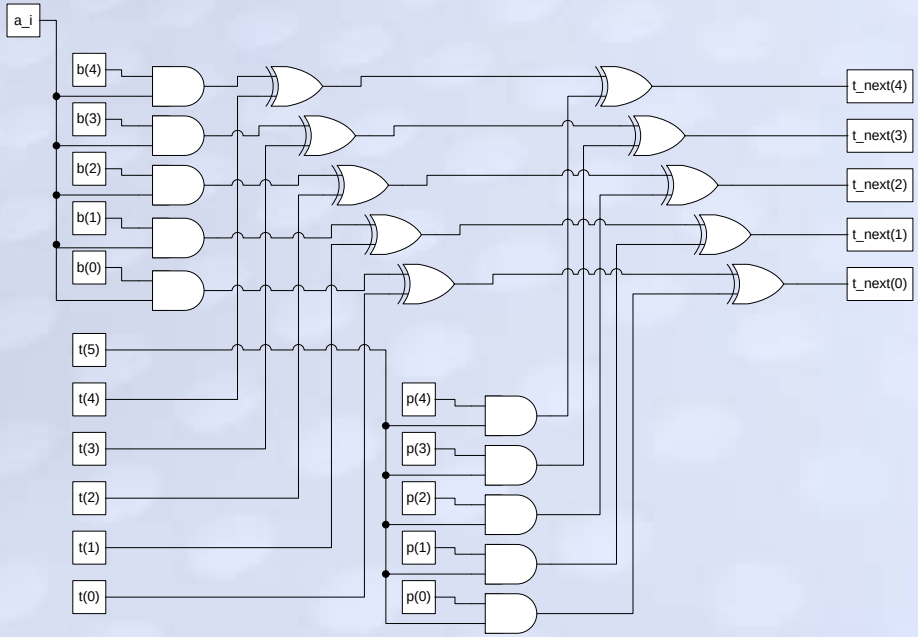
MALUd1 - Setup
- GF2n Multiplication -

Shift&Add Algorithm with simultaneous reduction of result

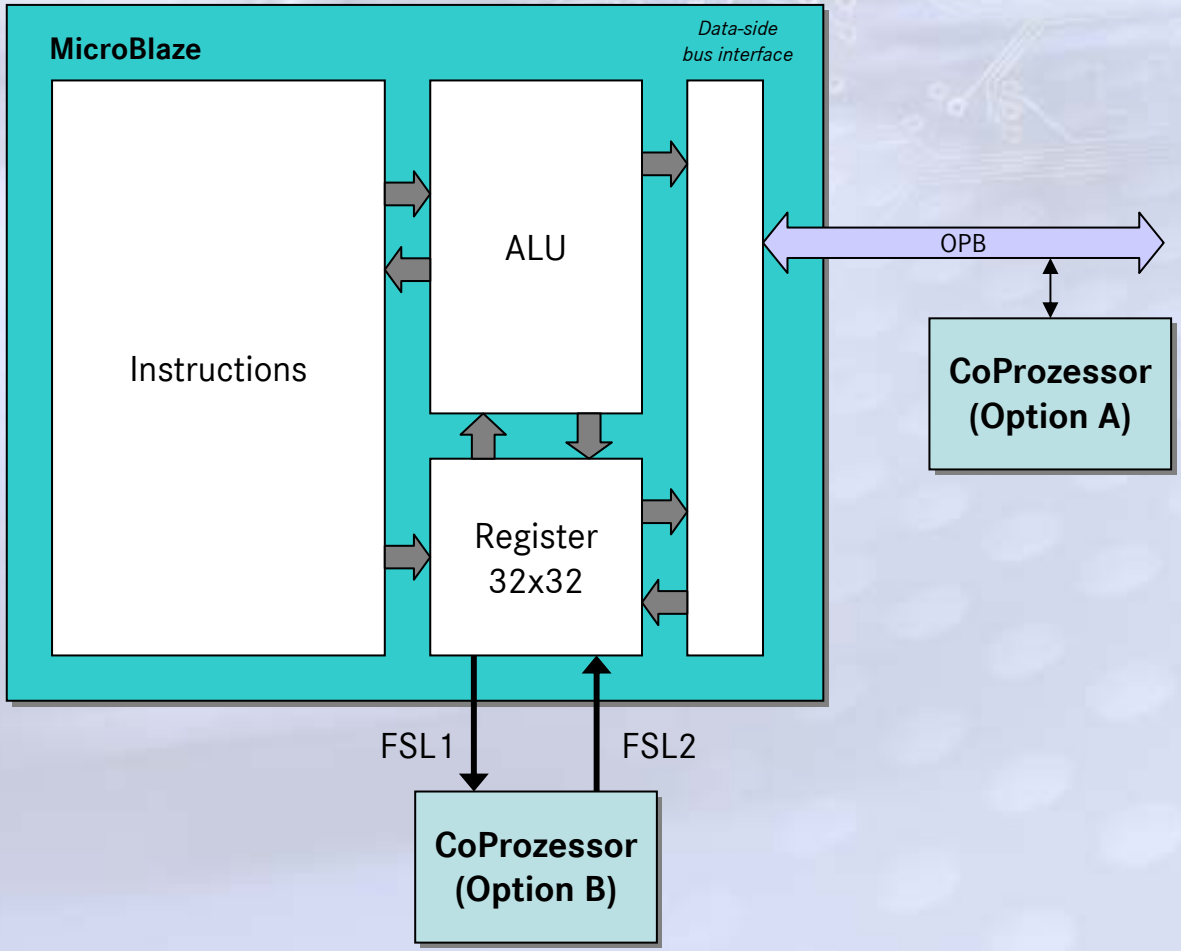


Setup of module „Cell“

- Reduction by adding a reduction polynom (hardcoded, XOR)



Connection of Peripherals to the MicroBlaze

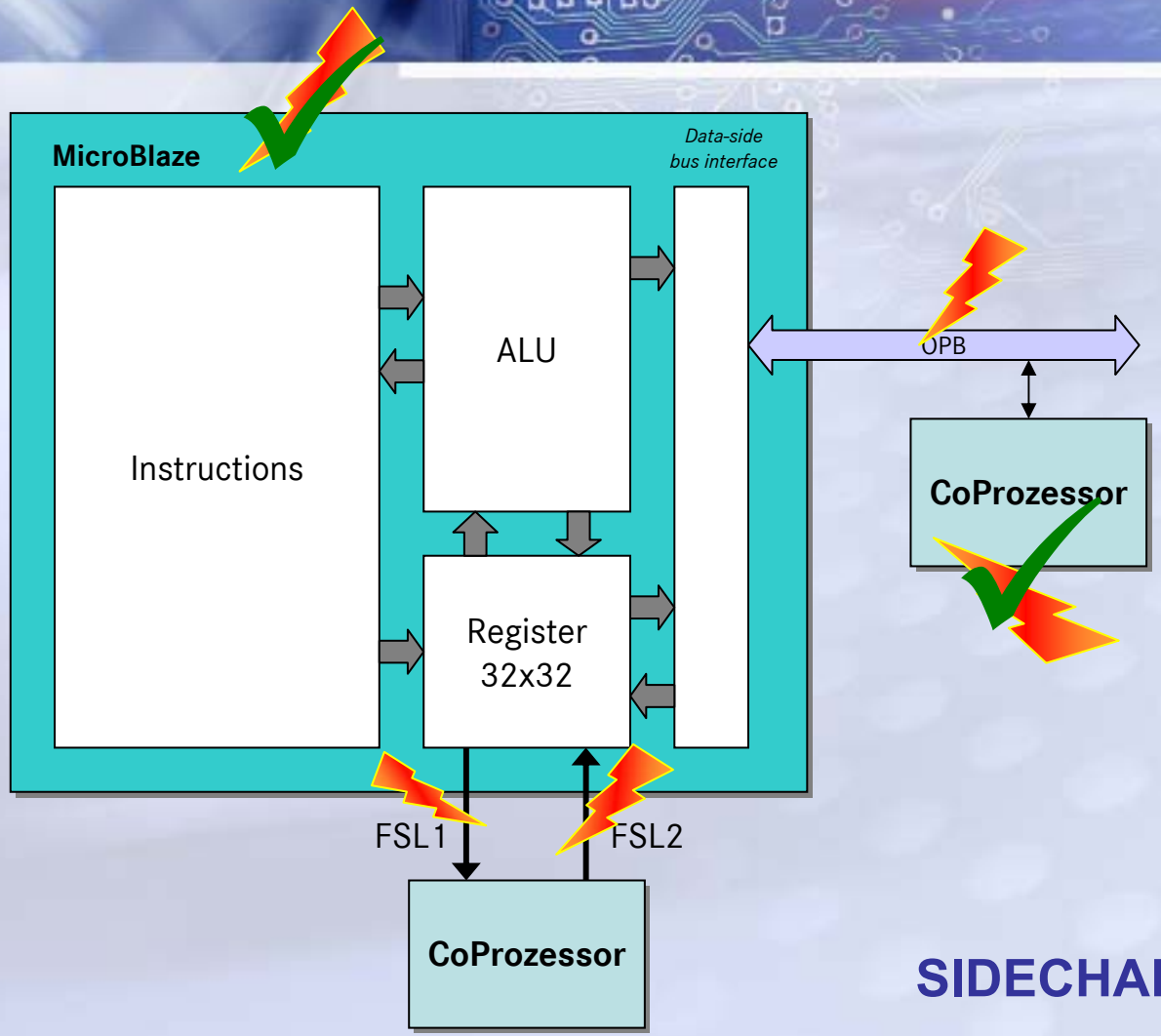


Interface options

a) On-Chip Peripheral Bus (OPB)

b) Fast Simplex Link (FSL)

Tradeoff: Performance vs. Secure System



Processor

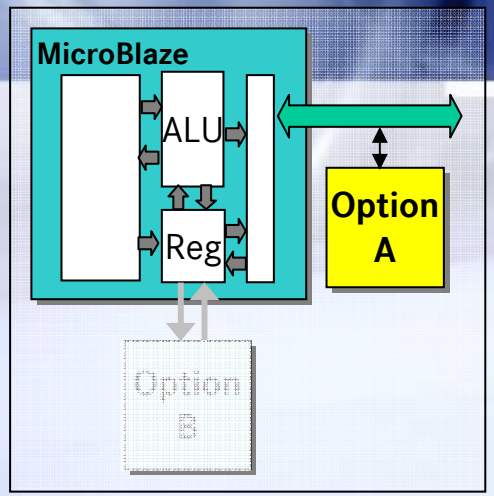
- Software
- Algorithms

Peripheral System

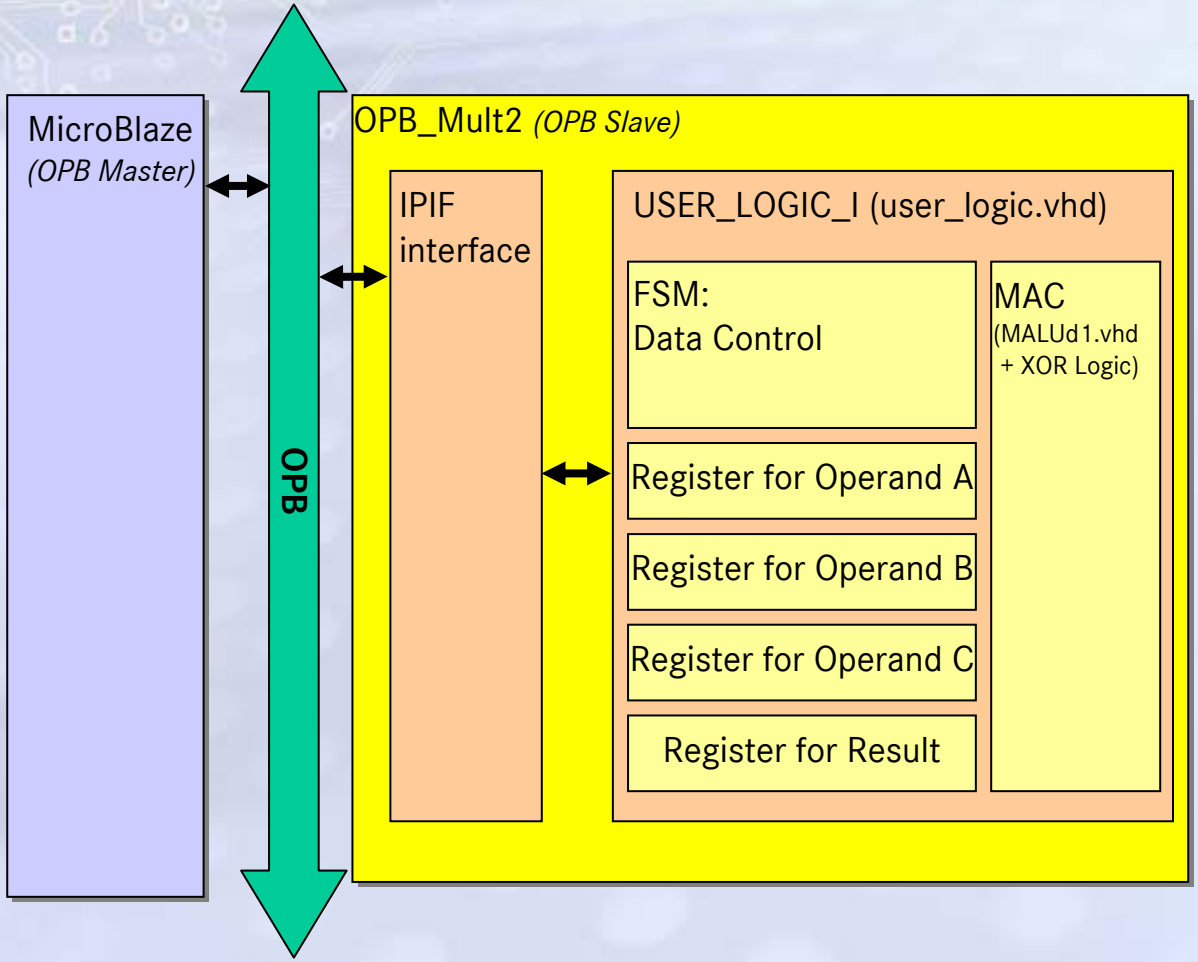
- Busses
- Memory
- Implementation

SIDECHANNEL AWARENESS !

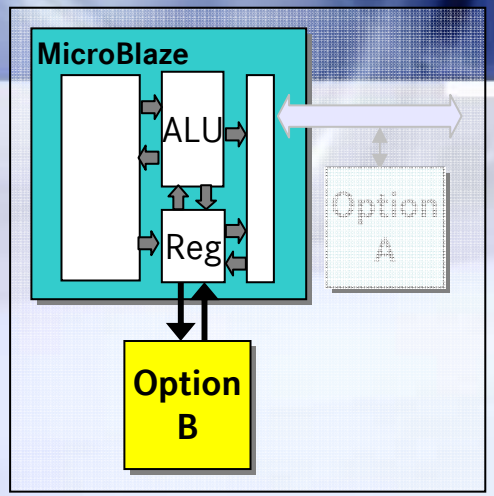
Interface MAC/MicroBlaze via OPB
- Overview -

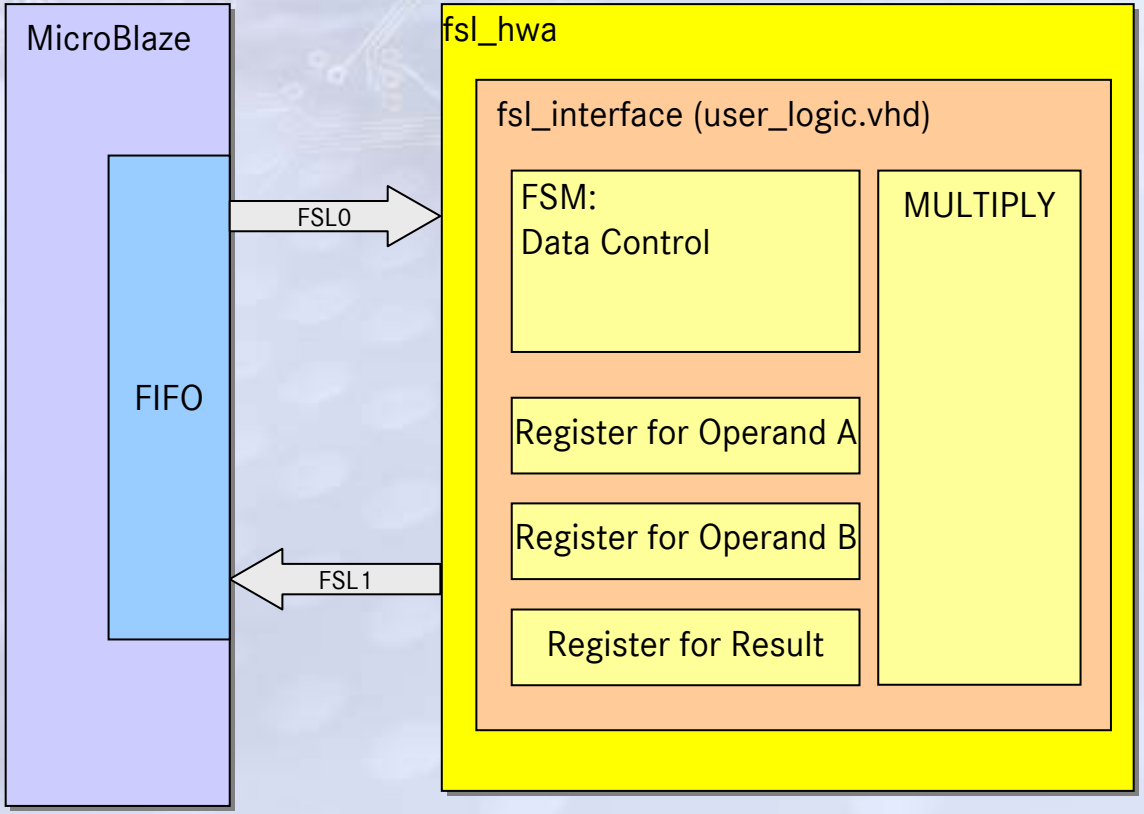


- **Registers are accessible (read/write) by Software.**
- **Busprotocol is implemented by IPIF.**
- **Function of Slave is implemented in user_logic.vhd.**



Interface Multiplier/MicroBlaze via FSL - Overview -

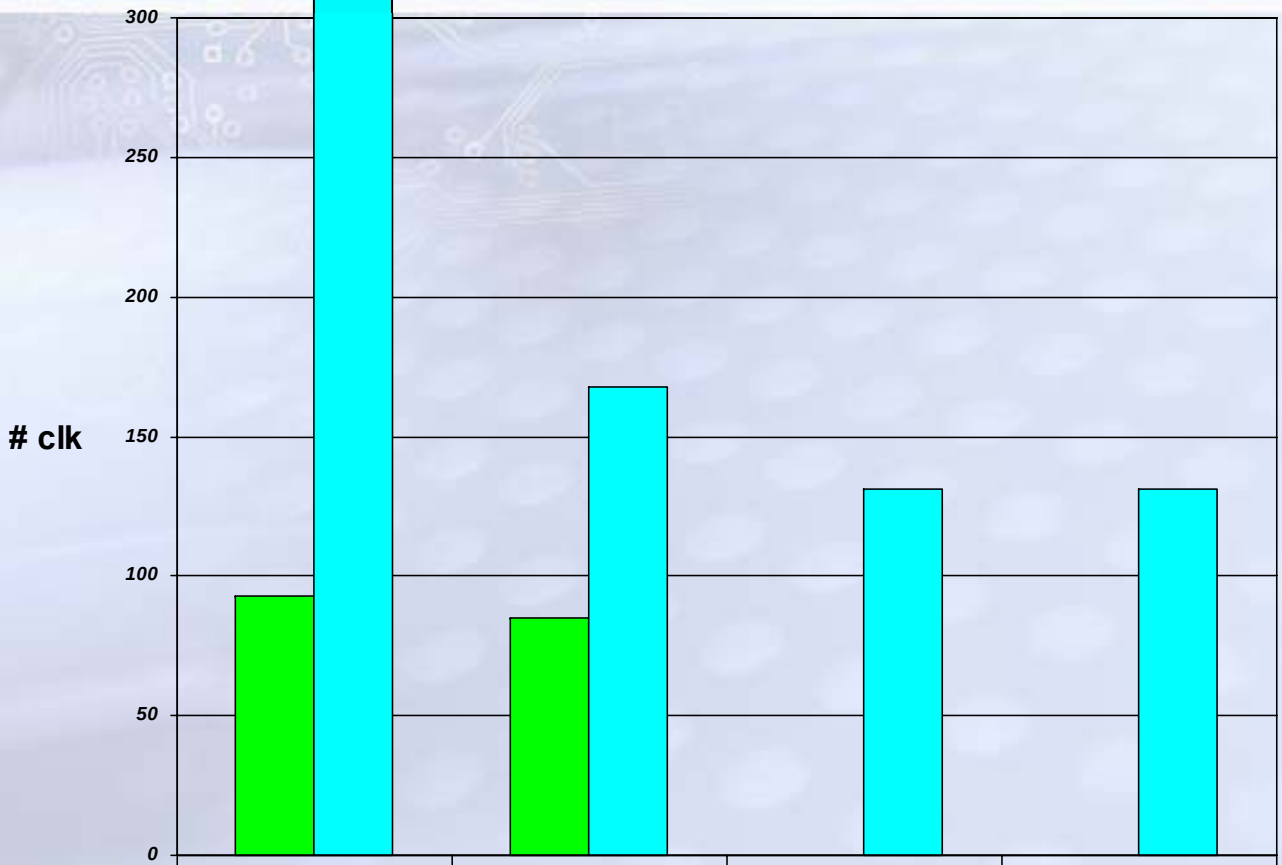




- **Data transfer over two FIFO, embedded into the MicroBlaze.**
- **Dataflow Control in User Logic**

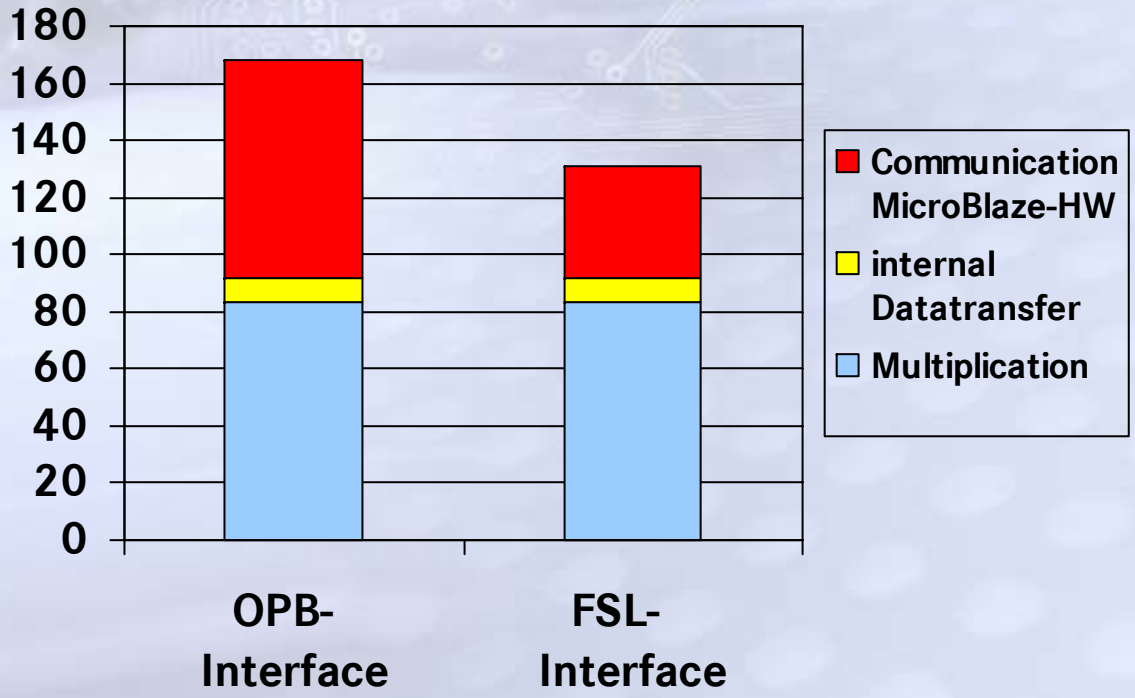
Comparision of B ic GF Operations

- Very high benefit for GF-Multiplication
- Almost no gain for GF-Addition
- MAC Unit beneficial for PointAdd/-Double operations



	Software	OPB	FSL	FSL-MAC
gf-add (u1+u2)	93	85	not implemented	not implemented
gf-mult (u1*u2)	12391	168	131	131

Communication-Overhead



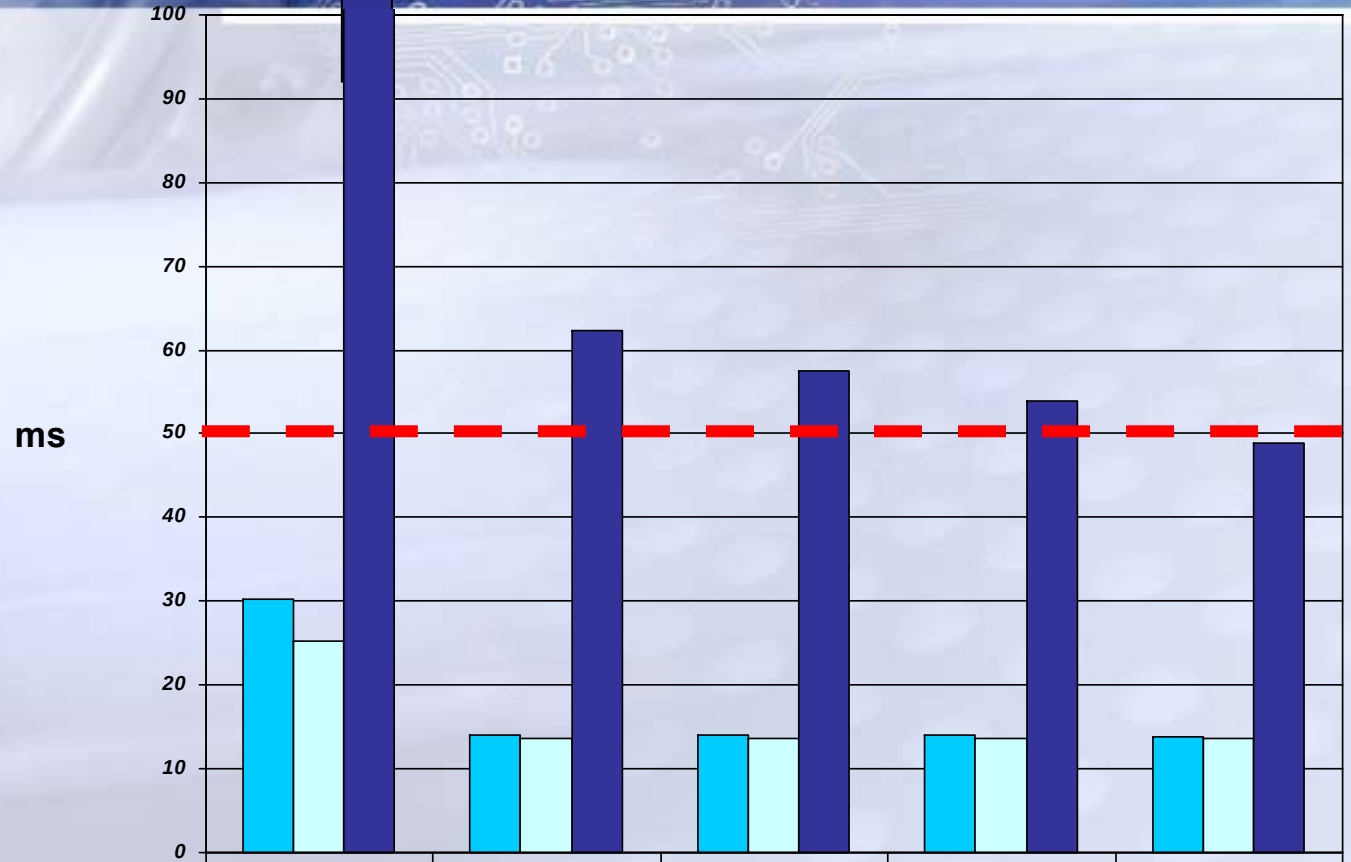
GF-Multiplication:

OPB : 50% of processing time needed for communication between MicroBlaze and HW.

FSL : over 30% of processing time needed for communication between MicroBlaze and HW.

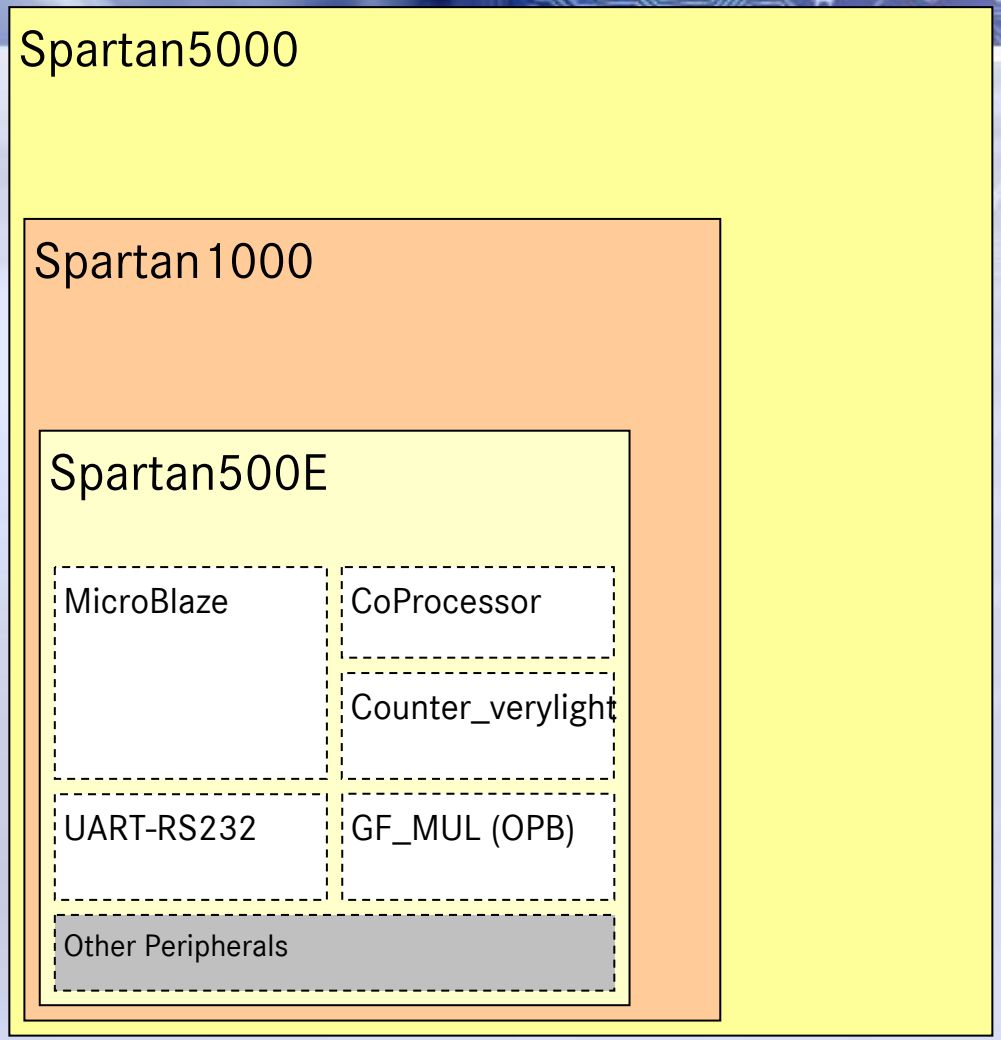
Speed Measurement

- Comparison Point Add/Point Double & Scalar Multiplication -



	Software	OPB-Mul	OPB-MAC	FSL-MUL	FSL-MAC
hecc-pointadd	30,276	14,014	13,981	13,959	13,925
hecc-pointdouble	25,343	13,591	13,571	13,55	13,529
hecc-scalarmult	2789	62,39	57,479	53,981	48,858

Resources Used on FPGA Platform
- Spartan 500 E -



Available Resources

Spartan3E S500 ft 256-4:

4656 Slices, 9312 FF, 9312 LUTs,

Used Resources

MicroBlaze:

1020 Slices, 811 FF, 1597 4 input LUTs

CoProzessor:

517 Slices, 509 FF, 920 4 input LUTs

UART - RS232 (OPB):

258 Slices, 277 FF, 438 4 input LUTs

Counter_verylight:

188 Slices, 211 FF, 298 4 input LUTs

Summary

Goals reached:

- Time Constraints are met if FSL interfacing is employed.
- Minimal resources are used. System can be implemented on a fairly small FPGA (Spartan 500E).

Evaluation:

- FSL Interface fast, but FIFOs tend to be very power consuming.
- OPB Interface too slow to meet timing requirements.

Next Steps:

- Evaluation of the systems security (Side-Channels).
- Optimization of SW, and CoProzessor.

Future Work

- **Architectures**
  - CoProcessor
  - PicoBlaze
  - NiosII
  - …
- **FPGA Security**
  - Storage of secrets, secure memory on FPGAs
  - avoidance of unauthorized access to FPGA and/or its bitstream
  - How are known side-channel attacks a danger to FPGA implementation?
  - …
- **Countermeasures against security threats**

Thank you for your attention.

Any Questions?

Alexander Klimm

Universität Karlsruhe (TH)
ITIV (Institut für Technik der Informationsverarbeitung)

eMail: klimm@itiv.uni-karlsruhe.de