Vector Processor study
for a 3.9G BB implementation, terminal side

St. Petersburg, 21.-23. May 2007

Holger Wirz, Nokia NRC/SRC/ASS Bochum
holger.wirz@nokia.com

NOKIA
Connecting People

Outline

- Introduction
- Motivation
- Main LTE parameters
- Overview of LTE BB architecture terminal side
- Potential candidates for vector processing
- EVP Architecture
- EVP Tools
- Benchmark Suite
- Performance Results
- Conclusion

Introduction

- Vector processor:
  - A processor with built-in instructions that perform multiple calculations on vectors (one-dimensional arrays) simultaneously.
- The vector processor study was done during start phase of a *LTE* (also *3.9G, EUTRAN* called) BaseBand algorithm/architecture implementation project for the terminal side
- As vector processor the EVP from Philips was used
- Responsible person for that study was: Dr. Ludwig Schwoerer, NRC Bochum (will move now to *NokiaSiemensNetwork*)

Motivation

Why was that vector processor study done ?

- LTE standard is not fixed => evolution of the standard must be tracked and algorithms might be improved/changed => flexibility needed (SW Defined Radio approach)
- The planned final LTE product should have
  - multiple standards (GSM, WCDMA, LTE), not used at the same time
  - low power consumption (active + standby)
  - competitive in costs compared to a dedicated HW modem

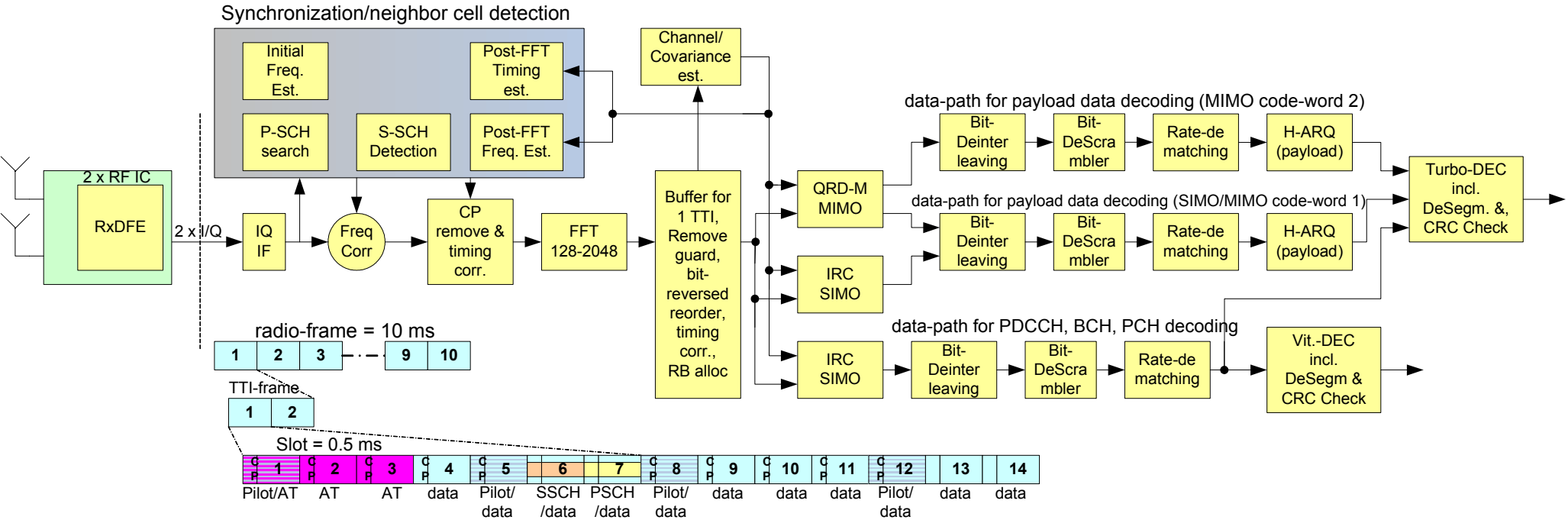
=> vector processors might help (final product should be available in 2011)

Main used LTE paramters for the PHY

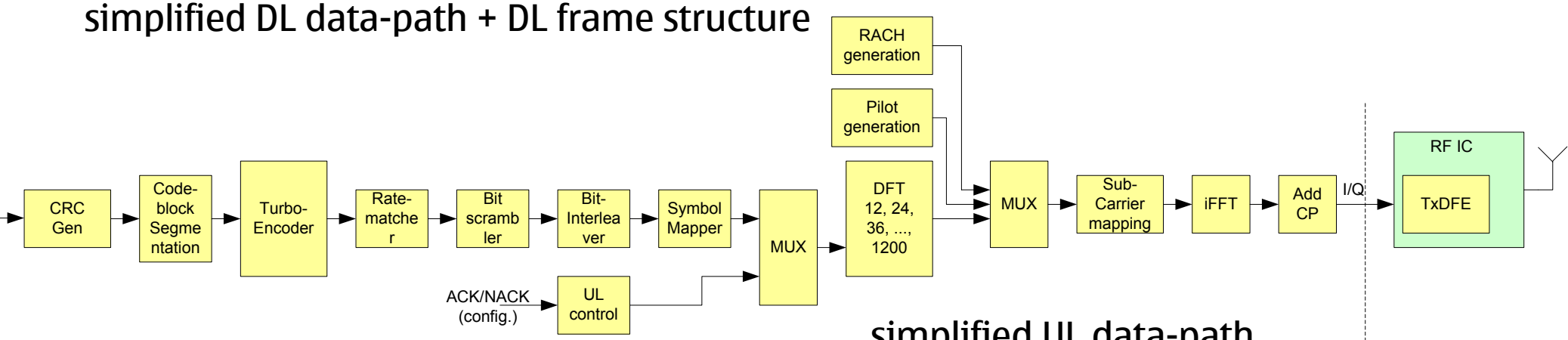
- FDD mode implemented (TDD later)
- Normal & extended cyclic prefix mode
- Flexible BW: 1.25, 2.5, 5, 10, 15, 20 MHz
- Modulations: QPSK, 16QAM, 64QAM
- Max. latency (UL & DL): 10 ms
- DL:
  - OFDMA, resource block size: 12 sub-carriers
  - MIMO 2x2 spatial multplexing, Alamouthi: space-frequency transmit diversity, SIMO/SISO (IRC, MRC)
  - peak data-rate: 100 Mbps (MIMO 2x2, 16QAM)
- UL:
  - SC-FDMA
  - 1 Tx antenna only used => allows virtual MIMO (on BS Rx side)
  - Peak data-rate: 50 Mbps for 16QAM (75 Mbps for 64QAM)

NOKIA
Connecting People

Overview of the digital BB architecture (terminal side)



simplified DL data-path + DL frame structure



simplified UL data-path

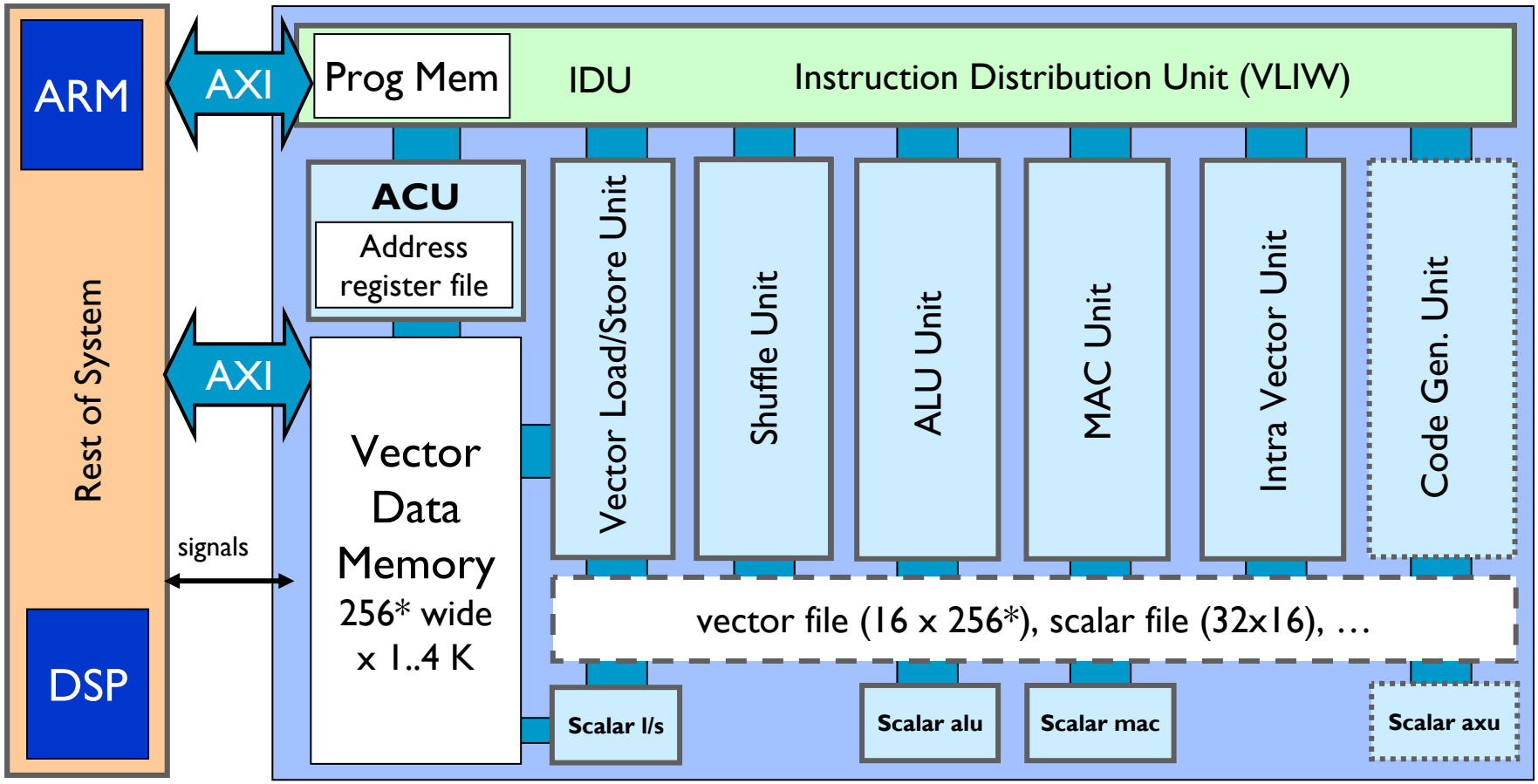
NOKIA
Connecting People

Candidates for vector processor implementation

- Potential candidates for vector processors:
  - Sliding cross-correlation (synchronization)
  - CORDICs
  - FFT/DFT
  - channel estimation and equalization
  - MIMO demapper (ML, QRD-M, Serial/Parallel Interference Cancellation)
  - SIMO demapper (Interference Rejection Combining, Maximum Ratio Combining)

- Performance typically scales with vector size

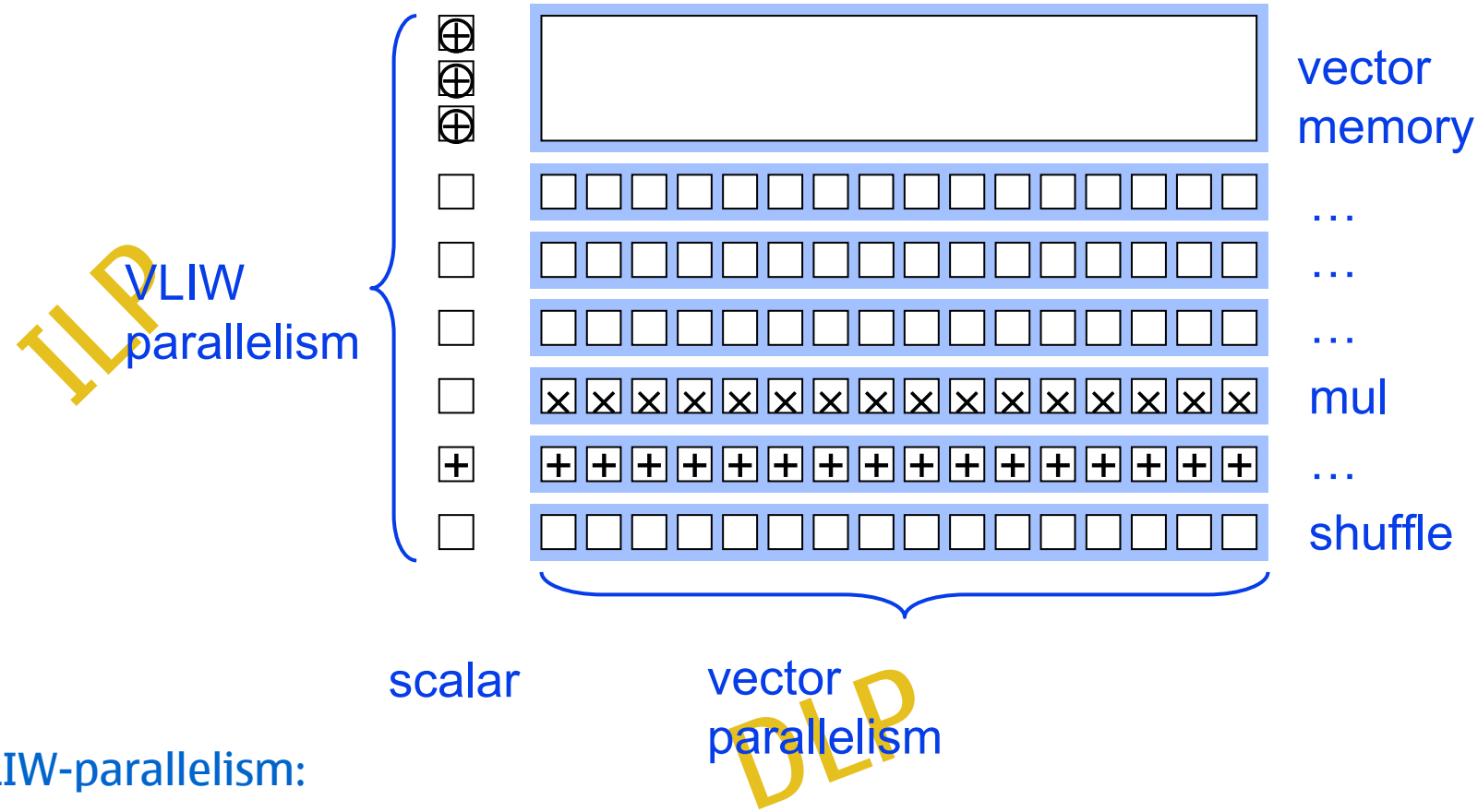
EVP architecture



90 nm CMOS: 450 k gates • 2mm² • 300 MHz • 0.5 mw/MHz core only • 1 mW/MHz typ. memory config. • 6 GIPS@300 MHz

NOKIA
Connecting People

EVP architecture : parallelism in EVP



VLIW-parallelism:

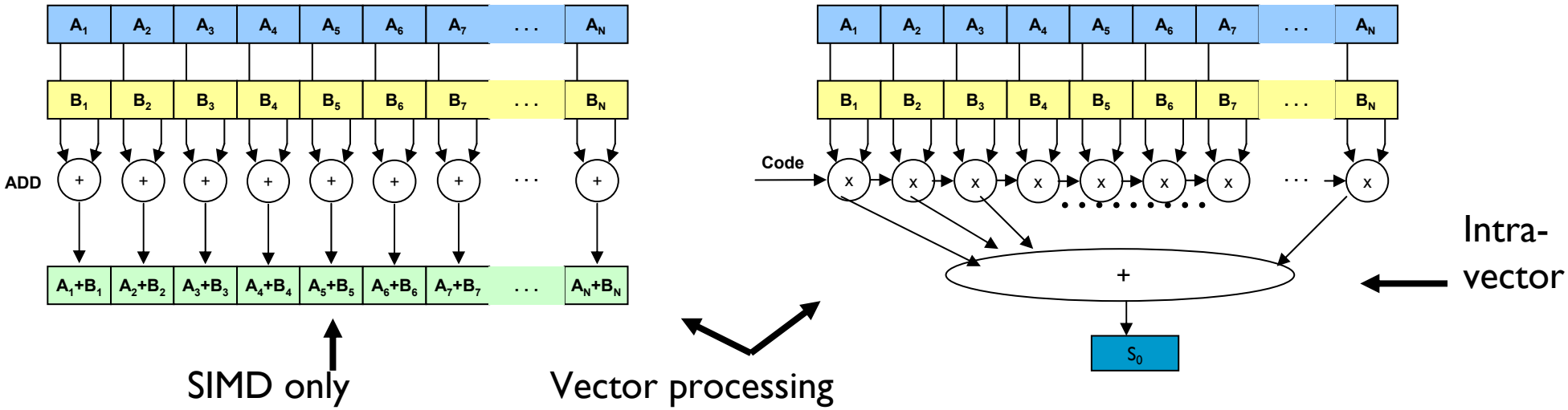
- Independent instructions on different functional units execute in parallel

Scalar units:

- compute scalar elements withina program and handle the program flow

NOKIA
Connecting People

EVP architecture : SIMD / Vector processing



intra-vector operations:

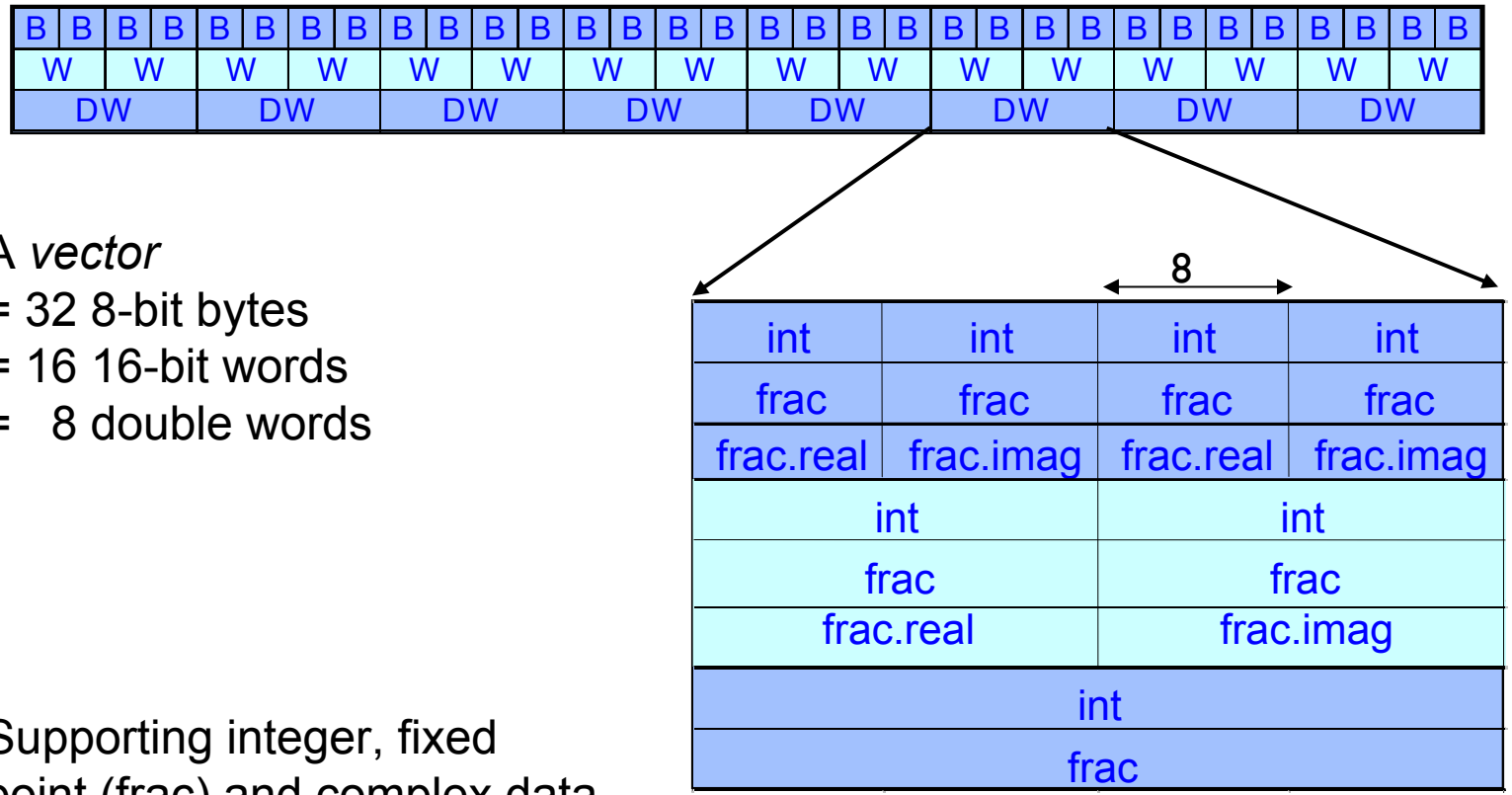
- perform operations on several elements within a vector

intra-vector units are:

- the vector shuffle unit, for permutations of elements within a vector
- the actual intra-vector unit, which e.g. performs the computation of the minimum within a vector

NOKIA Connecting People

EVP architecture: Data allocation to 256 bit vector

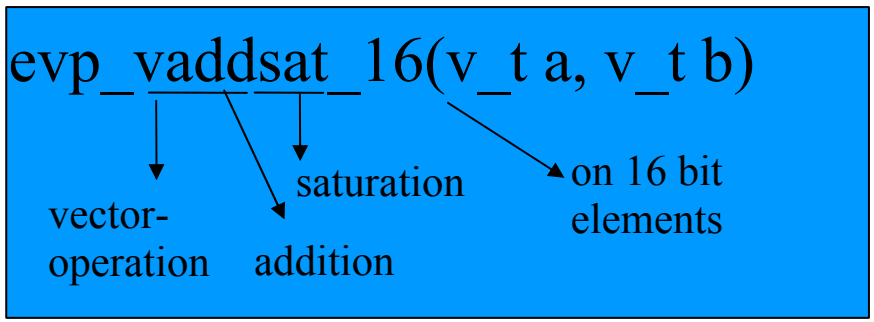


A *vector*
= 32 8-bit bytes
= 16 16-bit words
= 8 double words

Supporting integer, fixed point (frac) and complex data

Programming the EVP: EVP-C

- EVP-C:
  - C based language
  - Scalar computations /control flow : C
  - Vector operations: intrinsic functions
- Easy to learn:
  - Standardized, convenient structure
  - Good documentation and support



- Vectorization has to be done by the programmer
  - Requires knowledge or additional tools

Programming the EVP: Development tools

- Emulation library for C++
  - Step-by-step vectorization
  - verification
- Compiler: Generation of VLIW-code (scheduling of scalar + vector operations)
  - Inefficient for complex programs
  - Inefficient register handling for programs with many variables
- linker
- cycle true, bit-true simulator
- integrated debugger:
  - graphical interface is especially good suited for debugging, as registers can be inspected at run-time and breakpoints may be set

Mapping algorithms on the EVP: Benchmark Suite

- FFT
- QRD-M MIMO decoding
  - QRD
  - M- Algorithm
- Zeroforcing MIMO decoding
- Cholesky Decomposition
- Singular Value Estimation

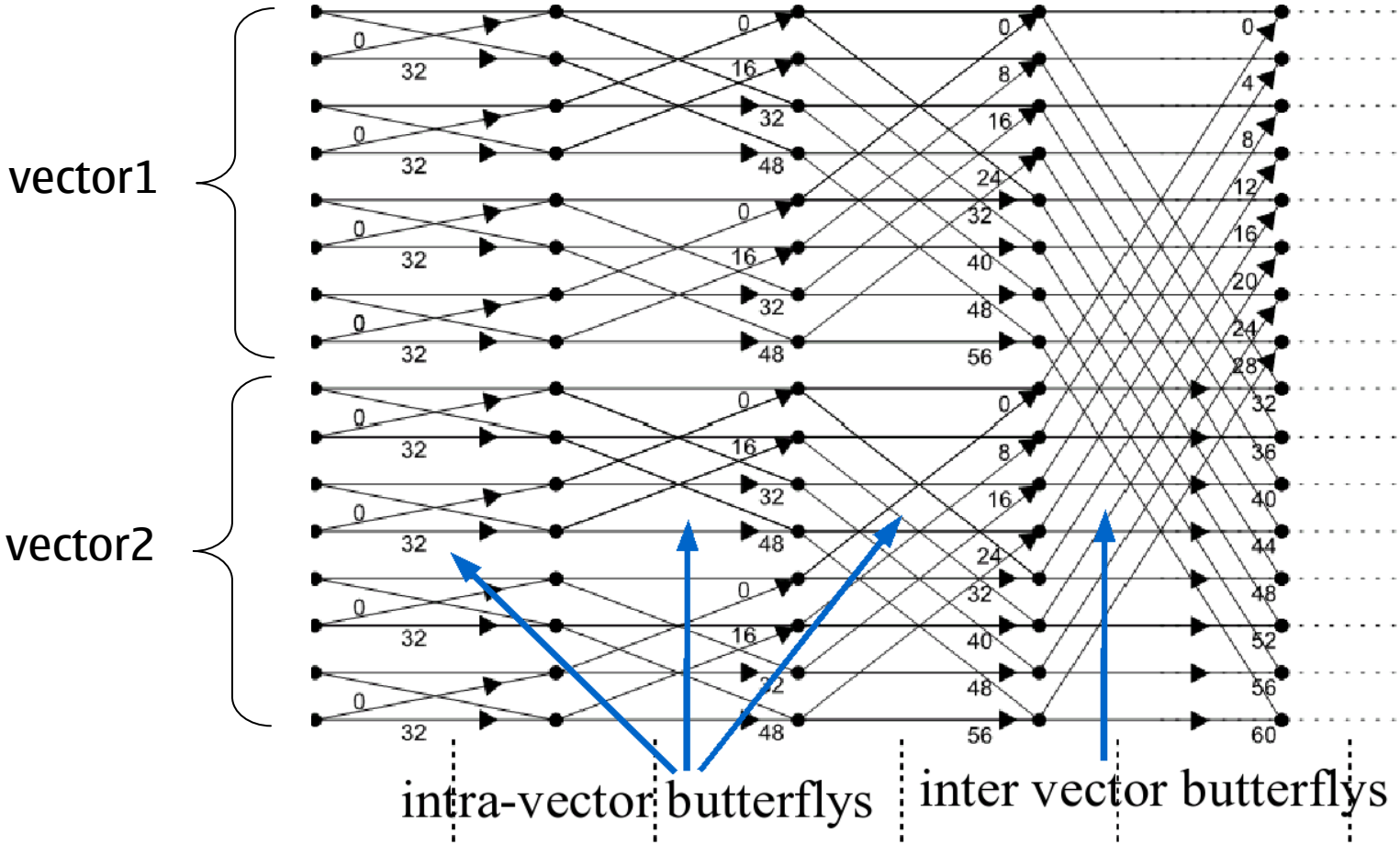
Benchmark: FFT

- Design parameters
  - Radix-2: most efficient for software implementation
  - 16+16 bit complex data (real + imag.), 8 elements per vector (vector is 16 x 16 = 256 bits)

- Constraint: Only 16 vector registers available
  - Blocks operate on 8 data vectors in parallel
  - 8 vectors for temporal values, twiddle factors

Benchmark: FFT

FFT-Fragment for 16 points



NOKIA
Connecting People

Benchmark: FFT – Results (1)

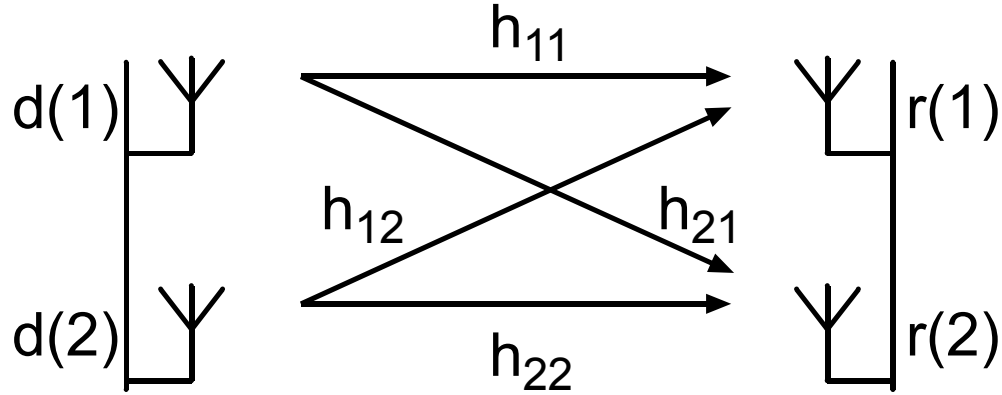
FFT size	First version	Manually optimized (2 week student work)	Philips' best results
64	64	50	50
128	183	144	128
256	575	365	280
512	1521	1318	624
1024	5974	3125	1440 Estimate

Benchmark: FFT – Results (2)

- Starting from the 64 point FFT, fast Fourier transforms for 128 up to 1024 points have been implemented without further optimisation, so that the performance could be evaluated.
- The cycle time for these implementations increases exponentially with growing FFT length.
- As the FFT length increases, the amount of wasted instructions for inserted move operations and temporal load-/store-instructions grows unproportionally.
- This can be signicantly reduced by manual handling of storage and avoiding move instruction.

Benchmark: QRD-M MIMO decoding

MIMO-systems:



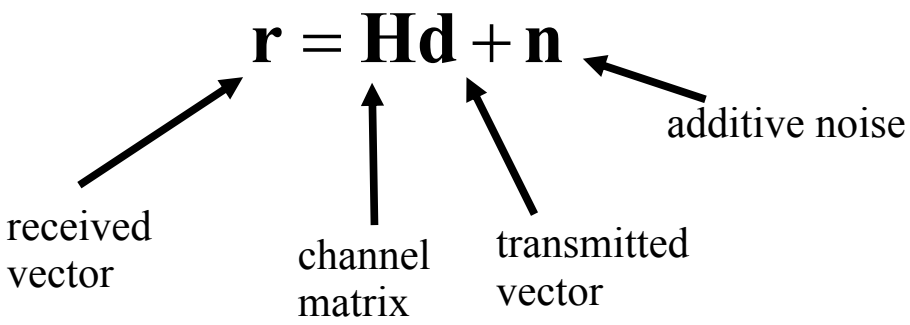
- Multiple transmit antennas and receive antennas
- ML-Detection of transmitted symbols is computationally expensive
  - e.g. 4 transmit antennas, 16 QAM mapping:

    → 16^4 = 65536 possible transmit vectors
- QRD-M reduces the amount of computations by limiting the candidate sets

NOKIA
Connecting People

Benchmark: QRD-M MIMO decoding

System model:



Challenge: Find the symbol vector which was most likely sent

QRD-M detection: 2 steps

- QR-decomposition of the channel matrix
- M-algorithm for the detection of the transmit vector

Benchmark: QRD-M MIMO decoding

QRD: Transform the channel matrix into upper triangular form

$$H = Q * R \longleftarrow \text{upper triangular matrix}$$

Unitary rotation matrix

$$H * s = r \rightarrow R * d = Q^H * r = r_q$$

$$\begin{pmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ 0 & R_{22} & R_{23} & R_{24} \\ 0 & 0 & R_{33} & R_{34} \\ 0 & 0 & 0 & R_{44} \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix} = \begin{pmatrix} r_{q1} \\ r_{q2} \\ r_{q3} \\ r_{q4} \end{pmatrix}$$

Method: Givens Rotations

- Each rotation zeros one matrix element
- 6 steps for 4 x 4
- Received vector and channel matrix are transformed in parallel

NOKIA
Connecting People

Benchmark: QRD-M MIMO decoding (QR-decomposition)

QR-decomposition per sub-carrier:

- No vectorization within the algorithm
- Instead “SIMD” implementation: 8 matrices (sub-carriers) are processed in parallel

Performance bottleneck (for 4x4 system):

- Register usage
- Extended channel matrix requires 20 vectors (for 4 x 4) + 4 vectors for the Givens-matrices
- Only 16 registers available
- Inefficient register allocation

  →manual handling of temporal storage required

Resulting performance:

4 x 4:	203 cycles for 8 QR-decompositions
2 x 2:	31 cycles for 8 QR-decompositions

Benchmark: QRD-M MIMO decoding (M-Algorithm)

Bottlenecks:

- Transition QR-decomposition (8 matrices in parallel) to M-algorithm (works sequentially on all subcarriers)
- Significant scalar part => Amdahl's law
- Compiler-efficiency for complex code

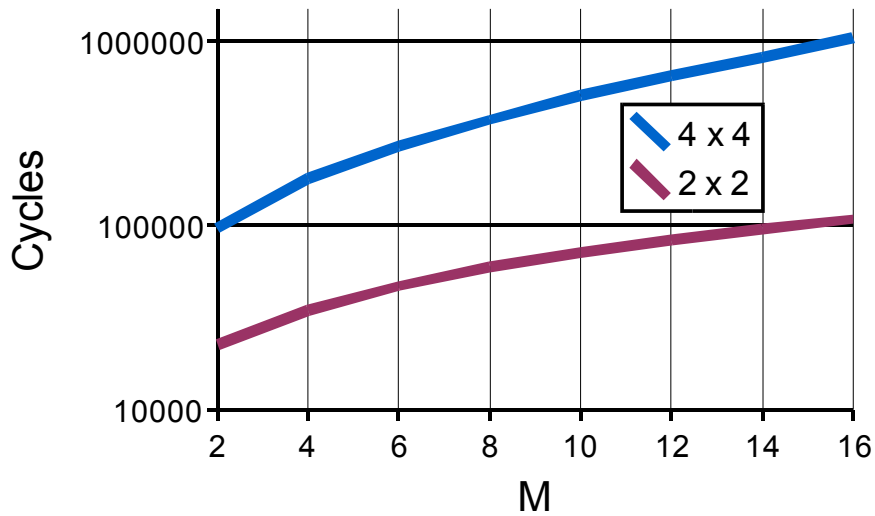
Resulting performance:

Not sufficient for 3.9G (M = 16, 2x2):

⇒ In highest BW = 20 MHz, 1200 used sub-carriers/OFDM symbol

⇒ OFDM symbol duration is 66.6 µs

Cycle count for 256 subcarriers



NOKIA
Connecting People

Amdahl's law

Limitations of Vector Processors

- Amdahl's law: Performance of parallel program is limited by sequential part of the program

Assume that α is the non-vectorizable fraction of the program, and that vector execution is *t* times faster than sequential execution.

$$speedup(\alpha, t) = \frac{1}{\alpha + (1-\alpha)/t}$$

- As *t* approaches infinity, speedup approaches *1/α*

Performance Evaluation: cycle counts

Algorithm	Cycle count	Notes
64 FFT	50	-
128 FFT	128	-
256 FFT	280	-
512 FFT	624	-
1024 FFT	1440	-
Cholesky 2×2	56	40 matrices
2×2 QR-D	32	8 matrices
4×4 QR-D	205	8 matrices
2×2 M-Algo.	22668/34956	256 matrices, M=2/4
4×4 M-Algo.	97025/180993	256 matrices, M=2/4
4×4 Zerof.	19711	256 matrices
QR-Iter.(2 p)	1640	2 matrices
QR-Iter.(4 p)	1668	4 matrices
2×2 QRD-A*	9464/22840	256 matrices @ 40/5 dB

QR-Iteration:

is needed for singular value (rank) estimation of a channel matrix

QRD-A*-algorithm:

is a novel algorithm for MIMO-decoding. It guarantees to find the ML solution and is very efficient for 2x2-systems

Performance Evaluation: Utilization (code analysis of loop body)

Algorithm	VALU	VMAC	VSHU	VLSU	IVU	Scalar	VBCST
64 FFT	49	33	26	22	0	0	0
128 FFT	114	72	50	80	0	0	0
256 FFT	258	160	100	242	0	0	0
512 FFT	834	512	300	890	0	0	0
1024 FFT	1794	1088	600	2102	0	0	0
2×2 Cholesky	10	0	5	30	0	0	5
2×2 QR-D	10	10	4	11	0	0	0
4×4 QR-D	68	92	28	48	0	1	0
2×2 M-Algo.	41	13	8	11	10	54	7
4×4 M-Algo.	167	35	35	160	55	360	43
4×4 Zerof.	46	20	0	24	12	37	13
QR-Iter.(2 p)	347	268	117	16	0	17	96
QR-Iter.(4 p)	406	270	99	23	0	9	96
2×2 QRD-A*	40	17	12	12	7	32	4

Performance Evaluation: VLIW Parallelism (code analysis of loop body)

Algorithm	Cycles	Operations	Parallelism ∅	Parallelism ∅ (without nops)
64 FFT	50	130	2.600	2.600
128 FFT	144	317	2.201	2.210
256 FFT	365	762	2.088	2.094
512 FFT	1318	2609	1.980	2.037
1024 FFT	3125	5906	1.890	1.987
Cholesky	56	74	1.321	1.563
2×2 QR-D	32	43	1.344	1.458
4×4 QR-D	205	292	1.424	1.580
2×2 M-Algo.	130	181	1.392	1.548
4×4 M-Algo.	583	906	1.554	1.607
4×4 Zerof.	146	178	1.219	1.267
QR-Iter.(2 p)	768	1076	1.401	1.557
QR-Iter.(4 p)	804	1109	1.379	1.510
2×2 QRD-A*	134	173	1.291	1.459

The VLIW paradigm is not utilised very well, except for the FFT with its regular computational scheme. Within most other algorithms, an average of well below two instructions is executed in parallel.

NOKIA
Connecting People

Conclusion: +

+ Very well suited for all matrix-vector algorithms, good results with vectorizable architectures (BB modem part)
+ Very well suited for recent Standards (WLAN, DVB-T/H, Wimax, all SISO <60MBit/s), especially for OFDM standards
+ For 3.9G (MIMO 2x2, 64QAM: >100MBit/s) a multi-EVP concept is needed, Complete UL Tx of terminal side could most probably be mapped to one EVP
+ The special units of the EVP, the vector shuffle unit VSHU and the intra-vector unit IVU have a great influence on the performance.
+ Short learning phase
+ EVP-C: easy to learn
    - Standardized, convenient structure
    - Good documentation and support

Conclusion: -

- with very complex algorithms (FFT1024, QRD-M) the used compiler was often not predictable => a lot of manual optimization was needed
- FEC (tree-search algorithms such as M, Viterbi) inefficient on EVP → HW Accelerator better
- Current compiler version:
  - Inefficient for complex programs
  - Inefficient register handling for programs with many variables
- Manual vectorization needed in general

Used abbreviations

- ACU: Address Configuration Unit
- ALU: Arithmetic Logic Unit
- DLP: Data Level Parallelism
- GIPS: Giga Instructions Per Second
- GOPS: Giga Operations Per Second
- IDU: Instruction Distribution Unit
- ILP: Instruction Level Parallelism
- MAC: Multiply & ACcumulate
- SIMD: Single Instruction, Multiple Data
- VILW: Very Large Instruction Word