CHAMELEON

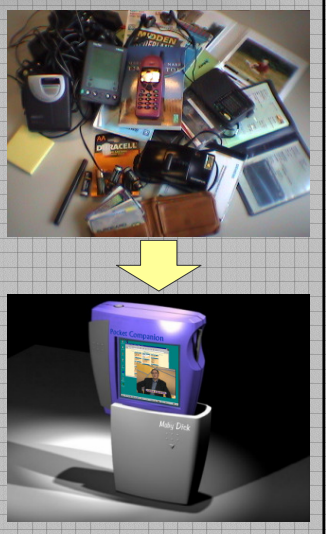
MAPPING OF DSP ALGORITHMS ON THE MONTIUM ARCHITECTURE

University of Twente
The Netherlands

PAUL HEYSTERS
UNIVERSITY OF TWENTE
DEPARTMENT OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE
EMAIL: HEYSTERS@CS.UTWENTE.NL
WWW: CHAMELEON.CTIT.UTWENTE.NL

Our vision on future wireless handhelds

- Mobile companion
  - Replaces many items that people carry around
  - Provides a rich variety of Internet applications
  - Any application
  - Anytime
  - Anywhere



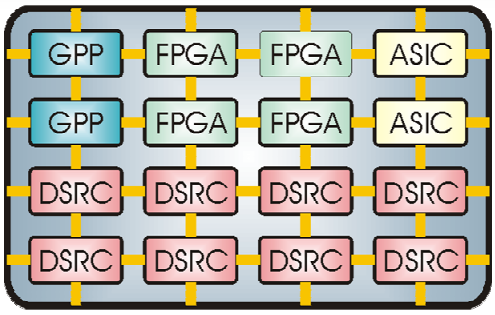
Key architectural requirements for mobile companions

- High performance
  - 3G/4G wireless technology
  - Multimedia processing
- Flexibility
  - Support multiple wireless standards
  - Keep pace with evolving standards
  - New applications
- Energy-efficiency
  - Battery has limited amount of energy

Flexibility vs. efficiency

- Conventional architectures tend to emphasize on only one of the requirements
  (high performance, flexibility, energy-efficiency)
- Coarse-grained reconfigurable architectures
  - Compromise programmability and fixed functionality
  - Flexible and efficient within an *algorithm domain*
- Mobile companion has multiple algorithm domains
  - Heterogeneous architecture

Chameleon system-on-chip for mobile computing

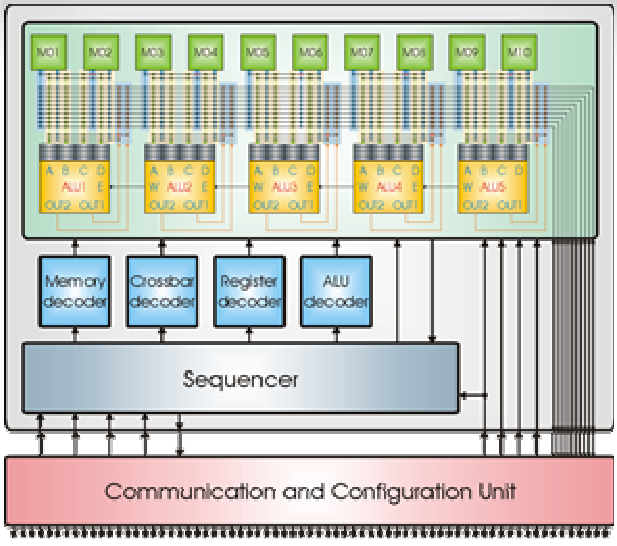


- General-purpose
- Fine-grained
- Coarse-grained
- Application specific

DSRC = domain specific reconfigurable

- Domain specific processing tiles connected by an on-chip interconnection network
- *Match algorithm with hardware that can execute it efficiently*

The Montium processing tile

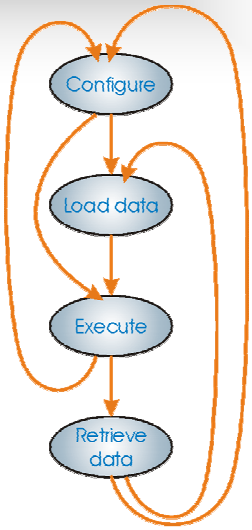


Tile Processor
- Computational part
- Processing Part Array
  - 5 *Processing Parts*
- Shared control part
  - Decoders
  - Sequencer

Communication and Configuration Unit
- Interface
- Slave of a remote master

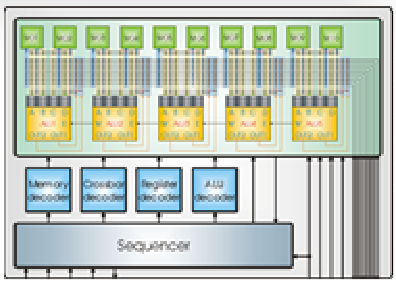
The CCU controls the Montium Tile Processor

- Configure the Tile Processor with an algorithm & load the input data
  - May be done in parallel!
- Let the Tile Processor compute algorithm
  - Data may stream
- a) Retrieve results & load new input data
  b) Retrieve results & reconfigure & load new input data
  c) Leave results in memory & reconfigure



Characteristics of the Montium Tile Processor

- Design goals
  - Energy-efficiency
  - Flexibility
  - Little control overhead
  - Avoid compiler and scheduler bottlenecks
- Algorithm domain
  - Digital Signal Processing
- Features
  - 16-bit datapath
  - Signed integer and signed fixed-point arithmetic
  - Streaming I/O possible



Montium Tile Processor	
Process	0.12 µm
Voltage	1.10V
Temperature	125C
Area (excluding wires)	1,8 mm²
Clock speed	45-150 MHz

Application example of the Montium Tile Processor

- Software Defined Radio
  - The same hardware can be used to implement multiple communication standards:
  - HiperLAN/2
    - 2 or 3 Montium tiles @100MHz. (depending on the modulation type: BPSK, QPSK, 16-QAM or 64-QAM).
  - Bluetooth
    - 1 Montium tile @100MHz.

Mapping of Algorithms on the Montium Tile Processor

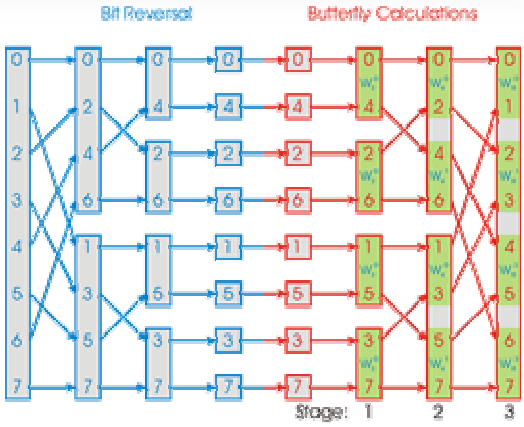
- Automated mappings
  - High-level design entry tools are essential for the (economic) viability of any reconfigurable architecture
  - Montium compiler is being developed:
    1. Translate C++ source code to CDFG
    2. Cluster CDFG and ALU datapath mapping
    3. Scheduling of the clusters
    4. Resource allocation
- Hand mappings
  - To verify the flexibility of the architecture

Cycle-by-cycle hand-mapped algorithms

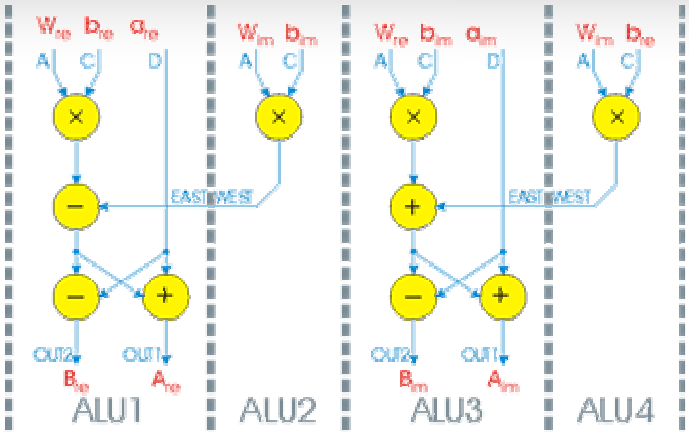
- Correlation
- Finite Impulse Response (FIR)
  - Maximum of 512 taps in one ALU
  - Maximum of 2560 taps in one tile
- Matrix multiply
  - [32x32] matrix × [32x32] matrix
  - [64x64] matrix × [64x1] vector (can be done streaming)
- Max-Log-MAP (Forward Recursion)
  - Maximum block size: 510 bits
  - (2m)+2 clocks (m is the block size)
    - Entire Max-Log-MAP estimated at about 9m clocks
- Discrete Cosine Transform (DCT) (8x8 point)
- Fast Fourier Transform (FFT)
  - Up to 1024 points in one tile
  - $((n/2)+2)\times\log_2 n$ clocks (=5140 for n=1024 points)

Detailed mapping example: 8 point radix 2 FFT

- FFT8
  - 3 stages
  - 12 FFT butterflies
- Store in local memories
  - 8 input samples
  - 4 twiddle factors
- Bit reversal
  - Twiddle factor are read from bit-reversed addresses
  - In the last stage the final results are stored on bit-reversed addresses



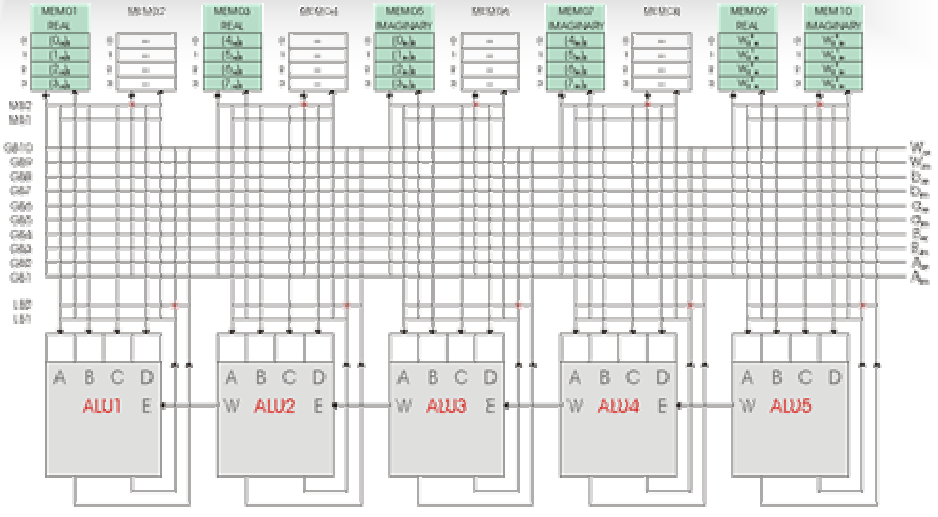
ALU mapping of the FFT butterfly



$$A_{re} = a_{re} + (W_{re} \cdot b_{re} - W_{im} \cdot b_{im}) \qquad A_{im} = a_{im} + (W_{re} \cdot b_{im} + W_{im} \cdot b_{re})$$

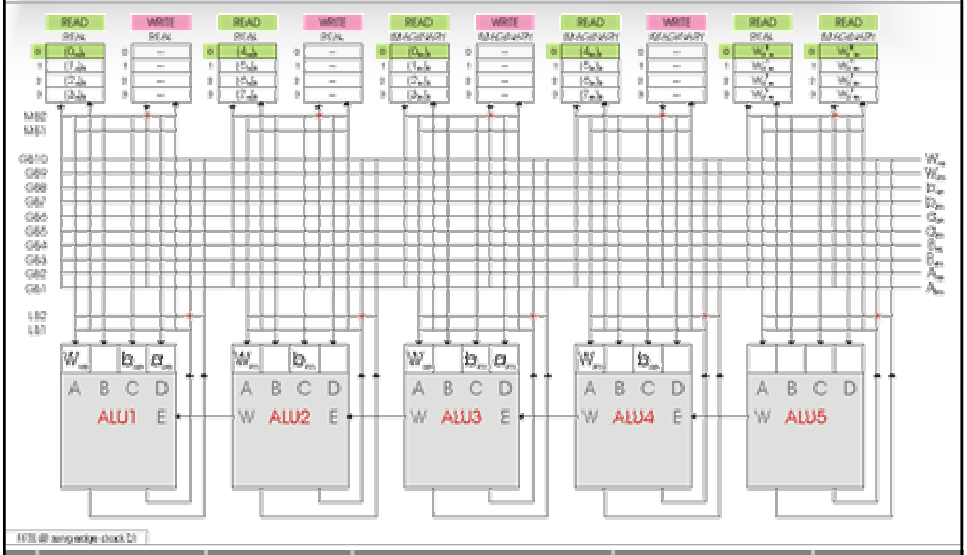
$$B_{re} = a_{re} - (W_{re} \cdot b_{re} - W_{im} \cdot b_{im}) \qquad B_{im} = a_{im} - (W_{re} \cdot b_{im} + W_{im} \cdot b_{re})$$

Initial data and twiddle factors are already stored in memories



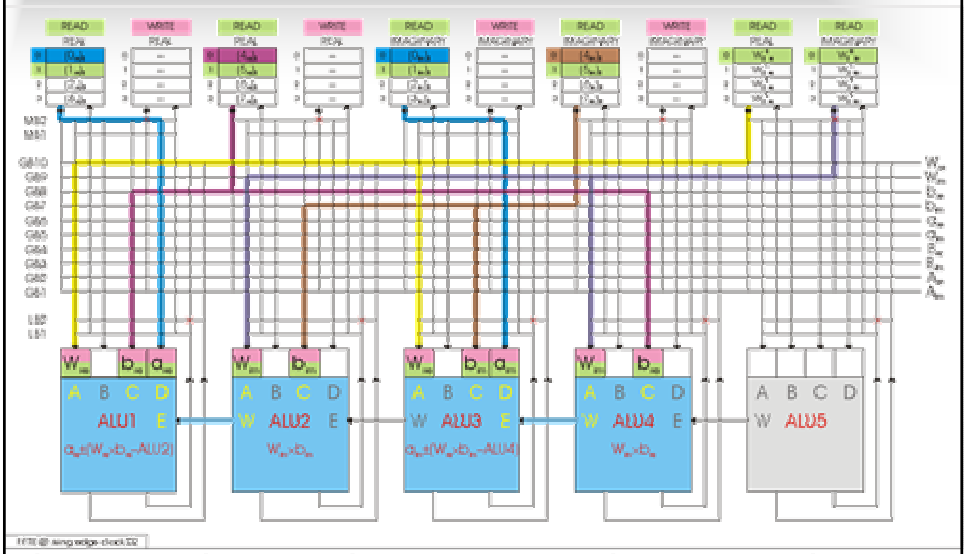
FFT8 - stage: 1
@ rising edge clock: 01

University of Twente
The Netherlands



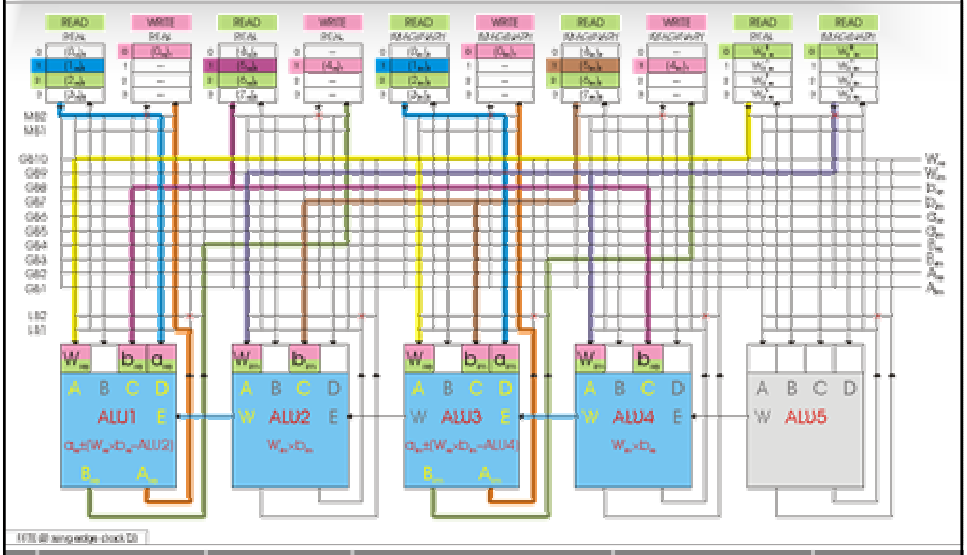
FFT8 - stage: 1
@ rising edge clock: 02

University of Twente
The Netherlands



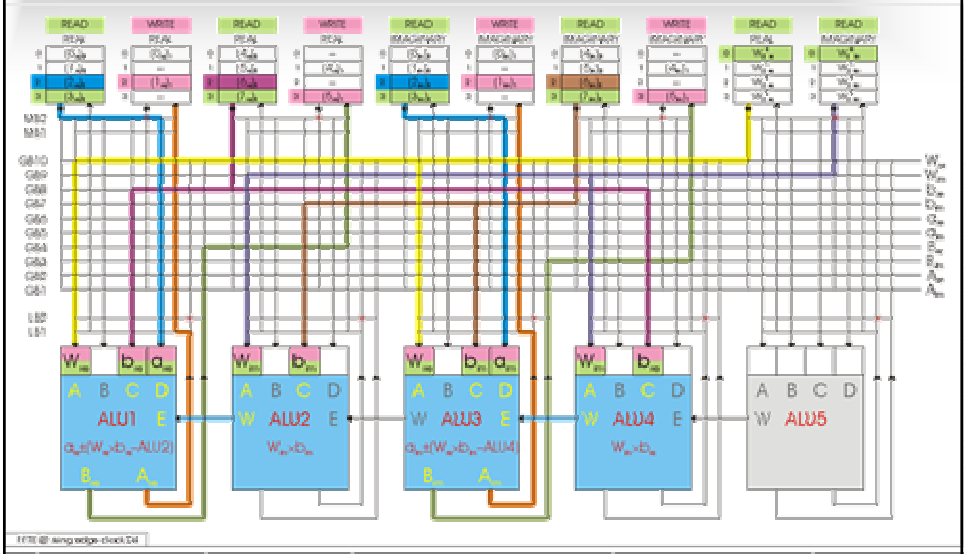
FFT8 - stage: 1
@ rising edge clock: 03

University of Twente
The Netherlands



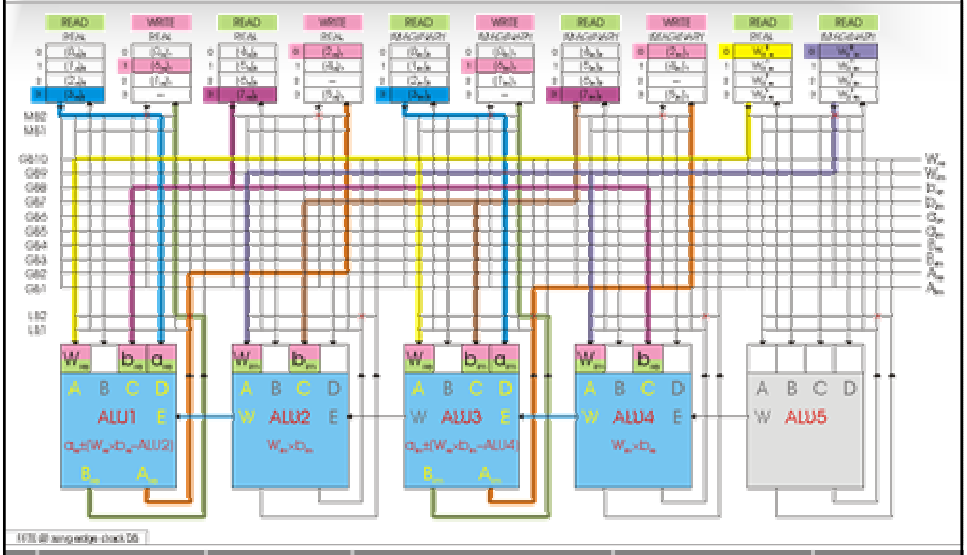
FFT8 - stage: 1
@ rising edge clock: 04

University of Twente
The Netherlands



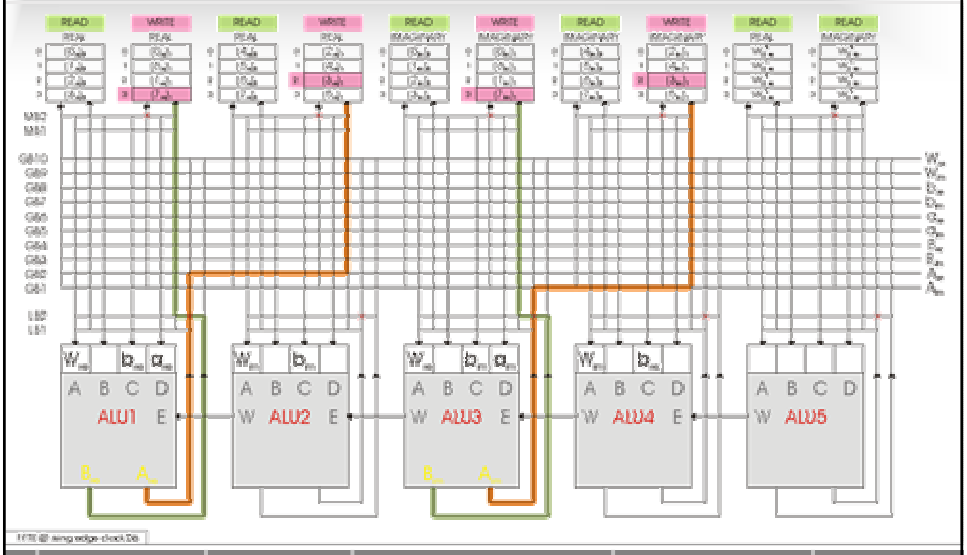
FFT8 - stage: 1
@ rising edge clock: 05

University of Twente
The Netherlands



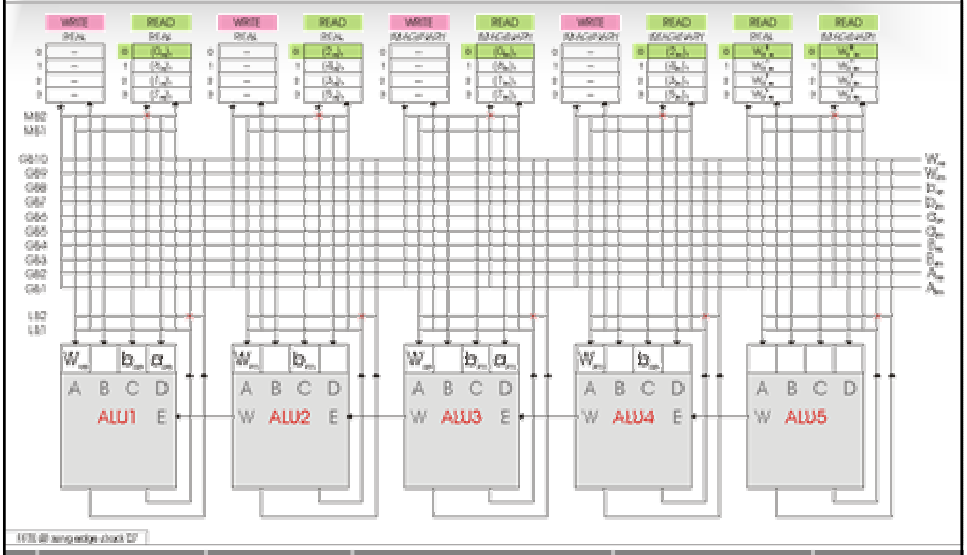
FFT8 - stage: 1
@ rising edge clock: 06

University of Twente
The Netherlands



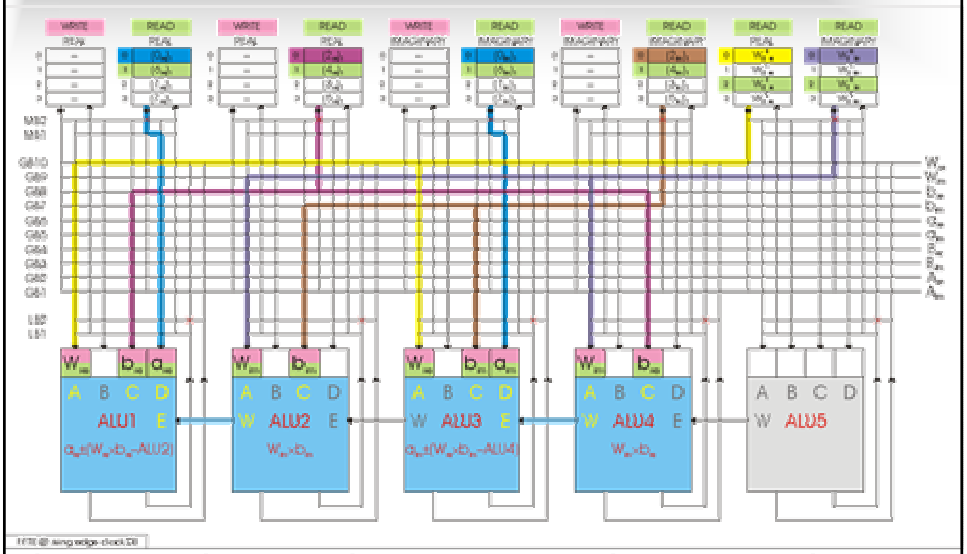
FFT8 - stage: 2
@ rising edge clock: 07

University of Twente



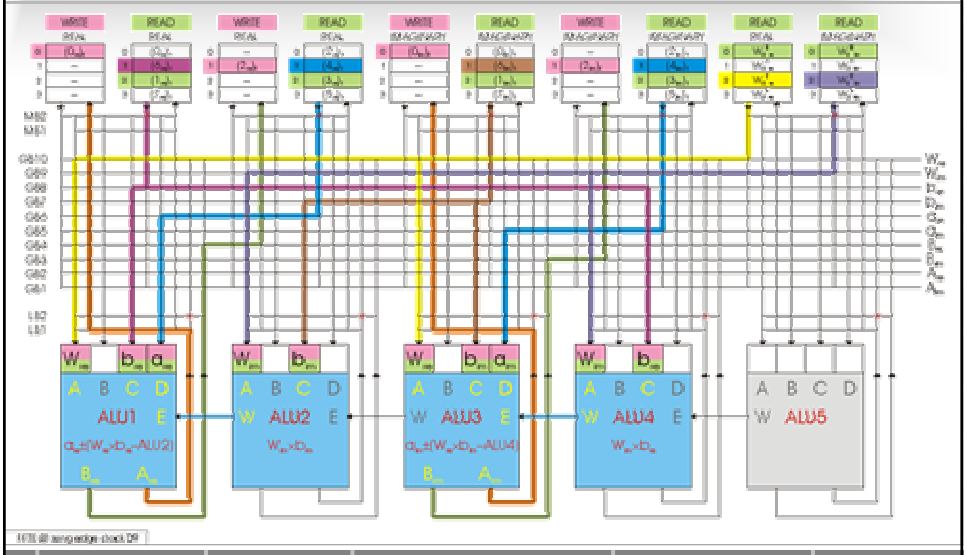
FFT8 - stage: 2
@ rising edge clock: 08

University of Twente



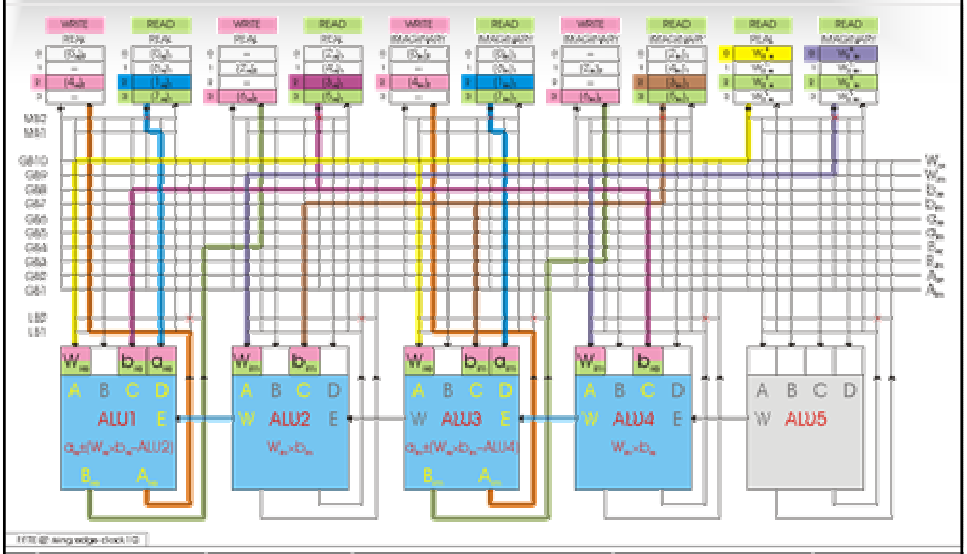
FFT8 - stage: 2
@ rising edge clock: 09

University of Twente
The Netherlands



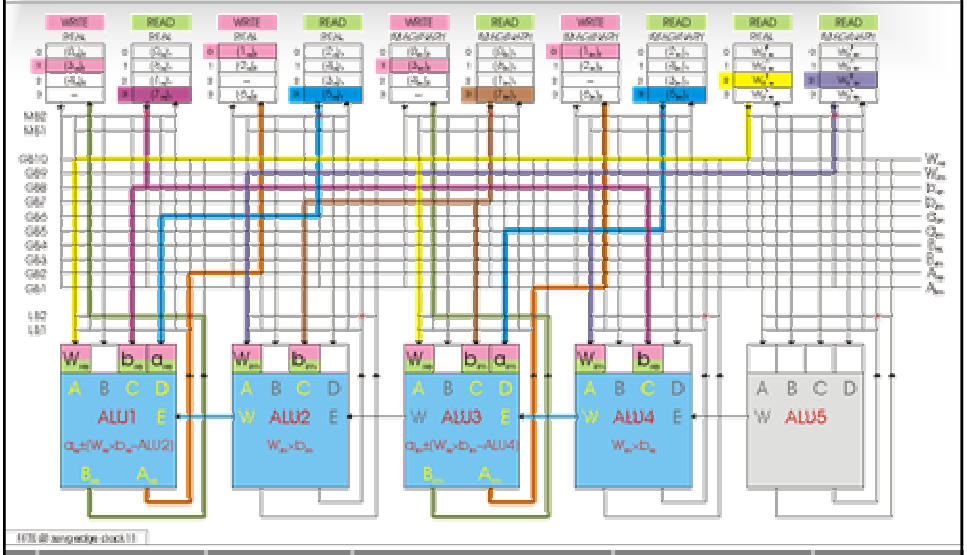
FFT8 - stage: 2
@ rising edge clock: 10

University of Twente
The Netherlands



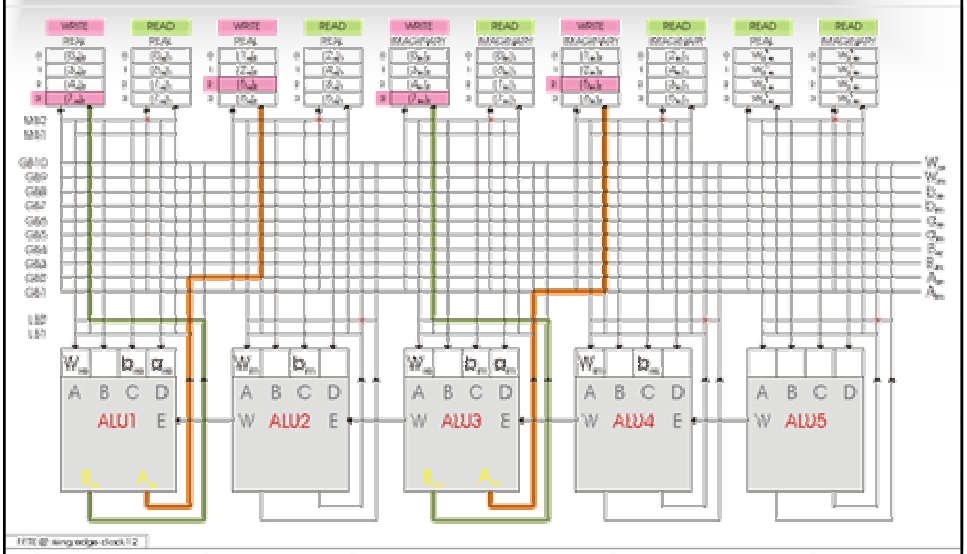
FFT8 - stage: 2
@ rising edge clock: 11

University of Twente
The Netherlands



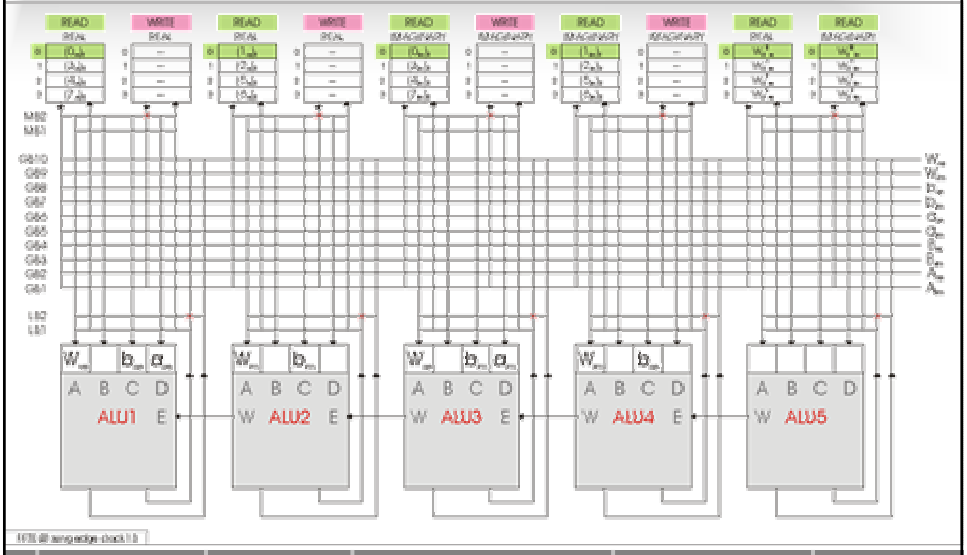
FFT8 - stage: 2
@ rising edge clock: 12

University of Twente
The Netherlands



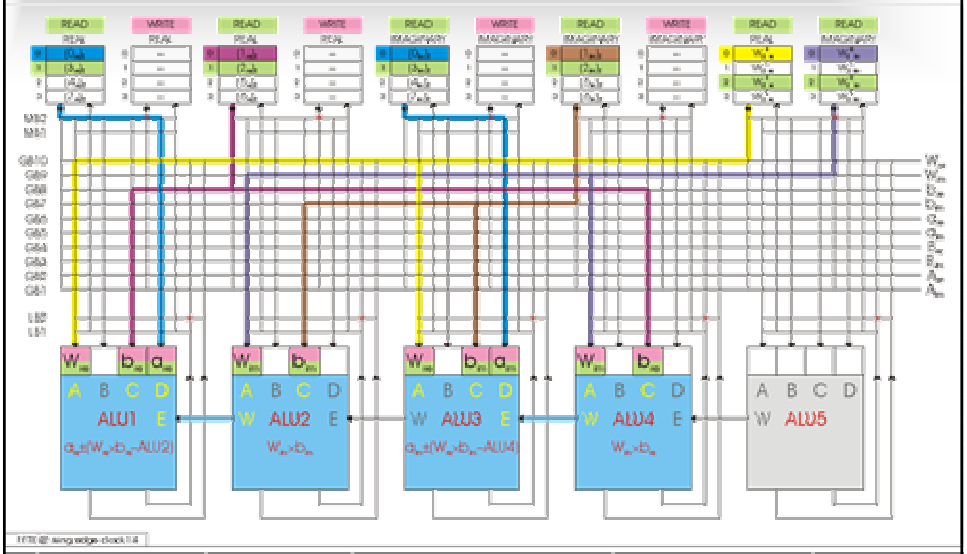
FFT8 - stage: 3
@ rising edge clock: 13

University of Twente
The Netherlands



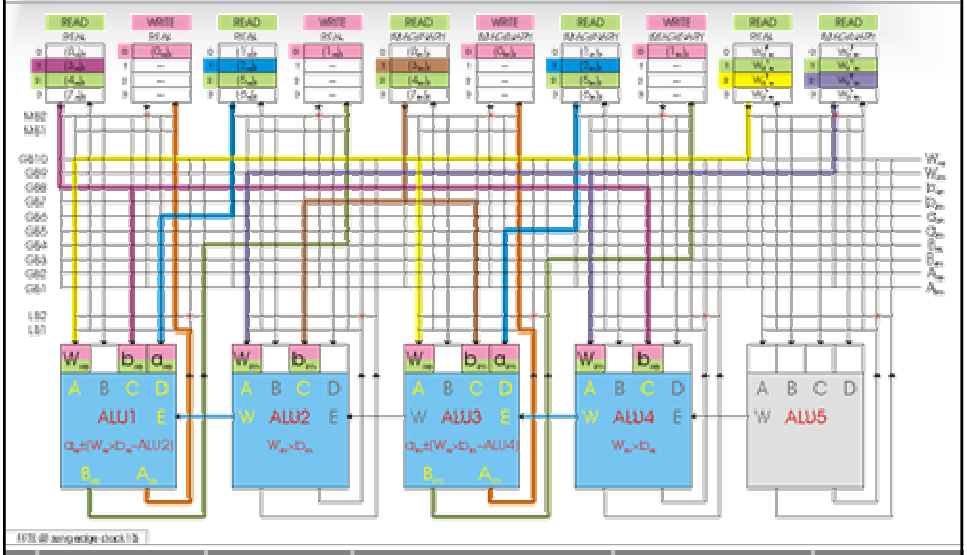
FFT8 - stage: 3
@ rising edge clock: 14

University of Twente
The Netherlands



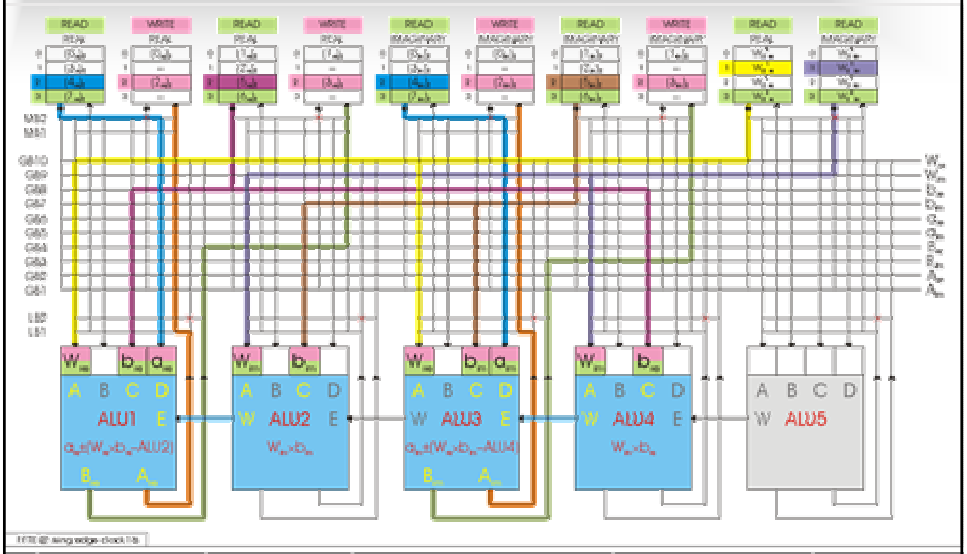
FFT8 - stage: 3
@ rising edge clock: 15

University of Twente
The Netherlands



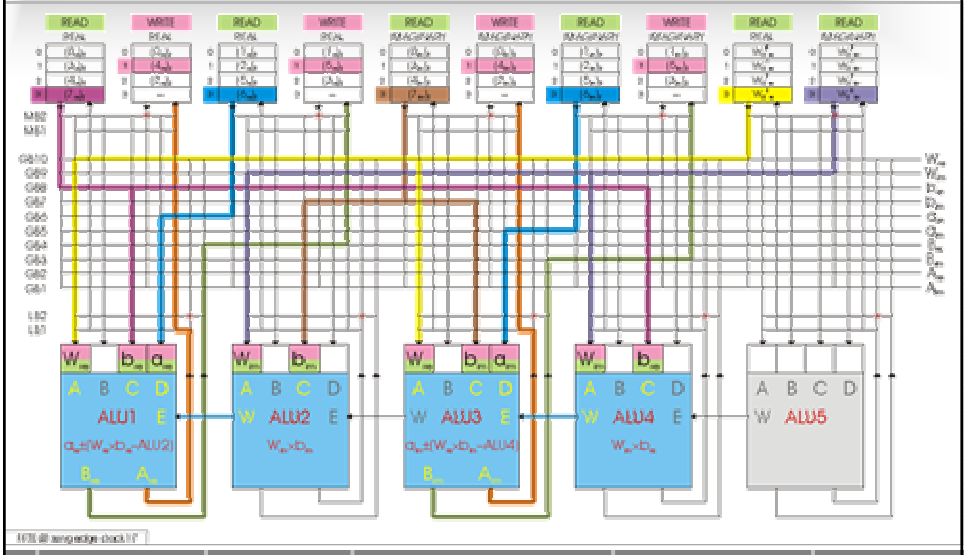
FFT8 - stage: 3
@ rising edge clock: 16

University of Twente
The Netherlands



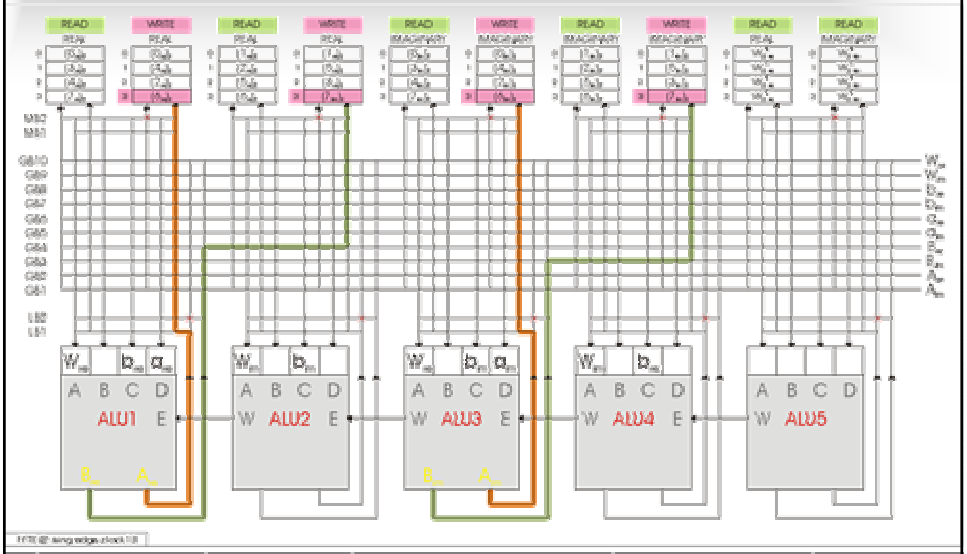
FFT8 - stage: 3
@ rising edge clock: 17

University of Twente
The Netherlands

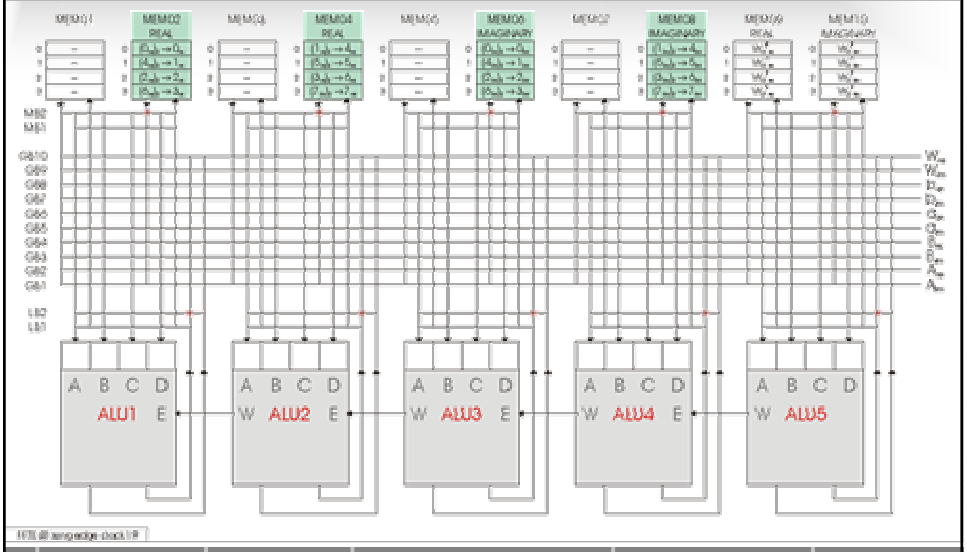


FFT8 - stage: 3
@ rising edge clock: 18

University of Twente
The Netherlands

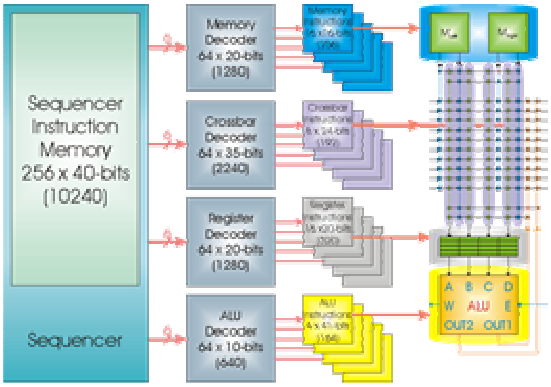


After 18 clock cycles the final results are in the memories



Control structure of the Montium Tile Processor

- *PPA datapath offers a lot of flexibility*
- A particular algorithm requires only a limited amount of flexibility
- Two levels of simple decoders are used to reduce control overhead
  - PP-level decoders (local instructions)
  - PPA-level decoders (PPA instructions)
- A simple state machine controls the decoders

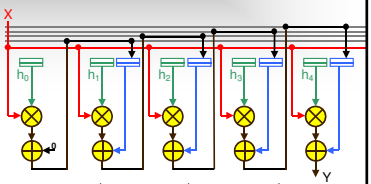


Montium Tile Processor configuration memory map

- Grand Total: 21820 bits
  - Worst case configuration data is about 2.6 Kbyte per tile
  - Worst case configuration time is about 1400 clock cycles

Simple, but realistic, configuration example

- 5-tap FIR filter
  - Initialize
    - Load coefficients from memories
    - Clear pipeline registers
  - Loop (512 times)
    - Load samples from Memory 1
    - Store results in Memory 9
  - Reconfiguration
    - 120 clock cycles
    - 240 bytes
- 64-point FFT
  - Reconfiguration
    - 473 clock cycles
    - 946 bytes




```
000 8000  503 0249  91E 0000  A20 0000
001 001C  580 0400  950 10C2  A21 0802
002 0000  600 0000  951 1002  A22 0800
003 0010  601 1111  958 0000  A30 0110
004 0200  602 2222  920 0000  A60 0000
005 2090  640 0120  921 0802  A61 005A
006 0208  700 4000  922 0800  A68 1000
007 40A1  701 0000  930 0110  AC0 0000
008 0408  8C0 0000  960 0000  AC1 0200
009 40B0  8C1 2400  961 005A  AC2 2600
00A 0408  880 0000  968 1000  A80 0000
00B 00C0  881 0080  9C0 0000  A81 0080
00C 8408  888 0000  98E 0000  A88 0000
00D 008C  88E FF80  99E 0000  A8E FF80
00E 8408  89E 0000  9D0 10C2  A9E 0000
00F 00D0  8D0 10C2  9D1 1003  AD0 10C2
010 8408  8D1 1000  9D8 0000  AD1 1005
011 009C  8D8 0000  9A0 0000  AD8 00E0
200 0001  8A0 0000  9A1 0802  AA0 0000
201 02FD  8A1 0802  9A2 0800  AA1 0802
202 0410  8A2 0800  9B0 0110  AA2 0800
203 0610  8B0 0100  9E0 0000  AB0 0110
204 0800  8E0 0000  9E1 005A  AE0 0000
400 0000  8E1 005A  9E8 1000  AE1 005A
401 1000  8E2 0000  A40 0000  AE8 1000
402 1000  8E3 00F0  A0E 0000  B88 2000
440 0120  8E8 1000  A1E 0000  B90 3000
500 0000  8E9 0000  A50 10C2  B98 5000
501 0000  940 0000  A51 1004  BA0 7000
502 0000  90E 0000  A58 0000  BA8 9000
```

5-tap FIR filter
pseudo code for sequencer

Address	Sequencer instruction	PPA instruction
0	Wait for data valid	NOP
1	NOP	rd_M1,2,4,6,8,10; alu=0
2	NOP	rd_M1; ld_reg; alu=MAC
3	LoopCounter := 509	rd_M1; ld_reg; alu=MAC; wr_M9
4	Loop to address 4	rd_M1; ld_reg; alu=MAC; wr_M9
5	Signal "done"	wr_M9
6	Wait for data invalid	NOP
7	Clear "done"	NOP
8	Jump to address 0	NOP

Note that 99% of the time the PPA instructions do not change
→ little control overhead

Conclusion

- Heterogeneous tiled architecture for mobile systems
  - Match algorithm with efficient domain specific hardware
- Coarse-grained reconfigurable Montium Tile Processor
  - Mappings of various algorithms show the *flexibility* of the architecture
  - Control overhead remains limited due to the simple control structure
  - *Performance* can be obtained by the parallelism in a tiled architecture
- Ongoing work
  - More benchmark applications
  - Energy estimations
  - Comparisons and evaluation

CHAMELEON

Thank you!

University of Twente
The Netherlands

CHAMELEON.CTIT.UTWENTE.NL