System Simulation Speedup Combining SystemC Models and Reconfigurable Hardware

Ulrich Nageldinger, Andreas Pyttel, Helge Kleve

Infineon Technologies AG
Munich, Germany

Overview

- Motivation
- Resources Used
  - SystemC 2.01
  - Celoxica Handel-C
  - FPGA-Board RC1000PP
- Methodology: Inclusion of HW in SystemC Model
- Example application: Viterbi Channel
- Experimental Results
- Conclusions and Future Work

Motivation: Concept Engineering

Concept Engineering phase:

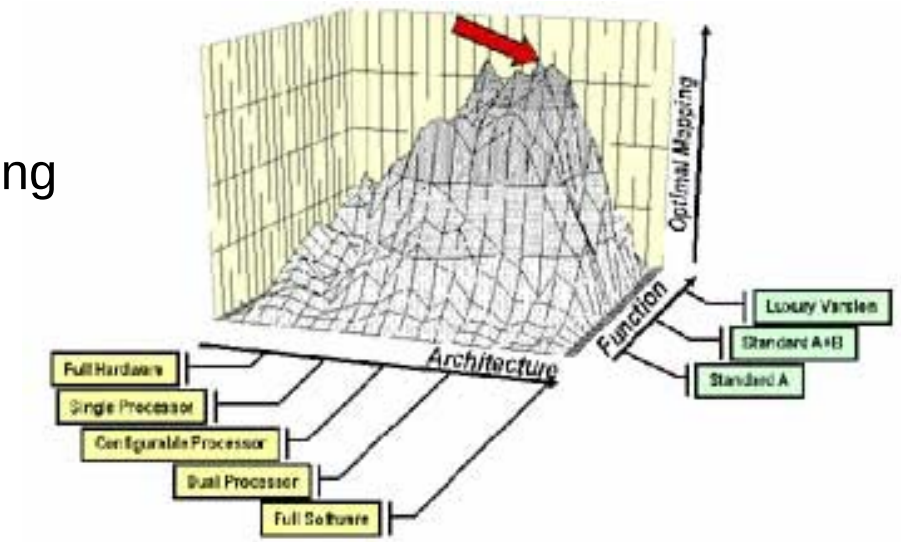
Take high-level design decisions for SoCs

Typical tasks:

- Architectural exploration and evaluation
- Tradeoff analysis for system partitioning and function mapping

By means of:

- Extensive simulation
  - Functional correctness
  - Performance simulation



http://www.ncsim.com/datasheets/vcc_environment.html

Motivation: Hardware-in-the-Loop

- HW-in-the-Loop successful and proven for ***low abstraction levels (RTL)***

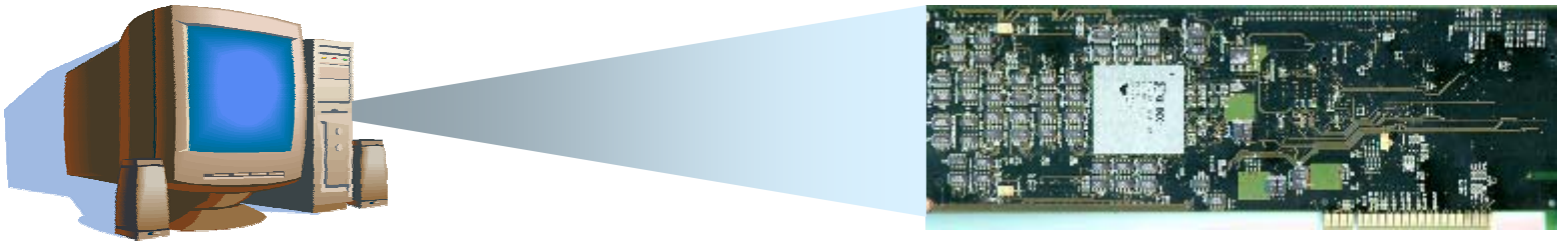
- HW-in-the-Loop at **System Level** ...
  - Integration of existing Hardware-Modules
  - Speed-up of simulation
  - Well-suited for functional simulation

➢ HW / SW System-Level prototype

➢ Continuation of previous work using VCC

➢ Now: SystemC as basis for integration

Our Approach



PC

- Simulation Environment
- System Models (not HW)
- Interfacing Software

FPGA Board

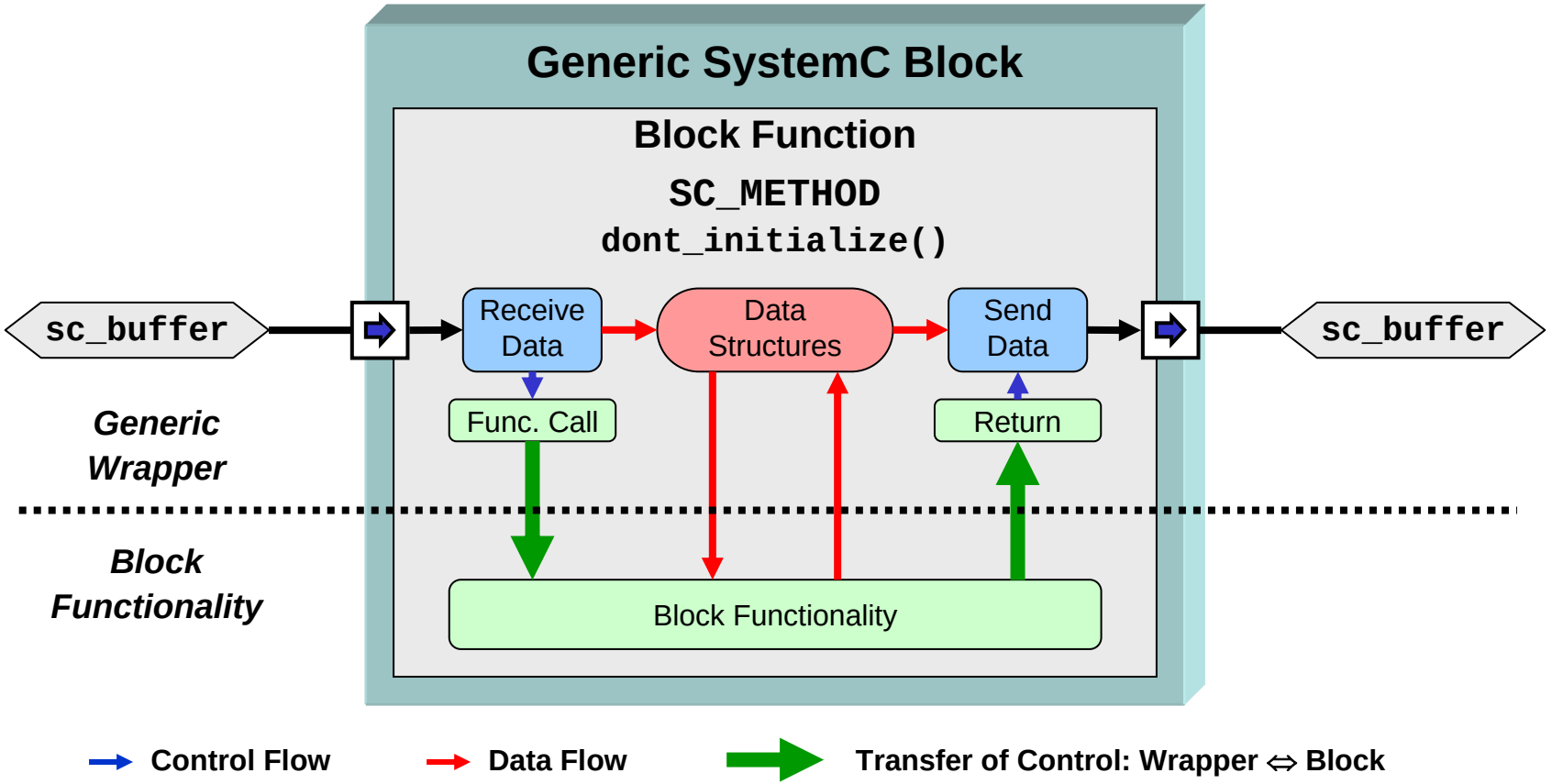
- Hardware-in-the-Loop

FPGA-Board instead of dedicated Prototyping System

- ➢ Low cost solution
- ➢ Only part of the system on the FPGA board

SystemC

- Models of example system transferred from VCC
- Not all of SystemC features were used
- Focus on *functional aspects* (no timing)



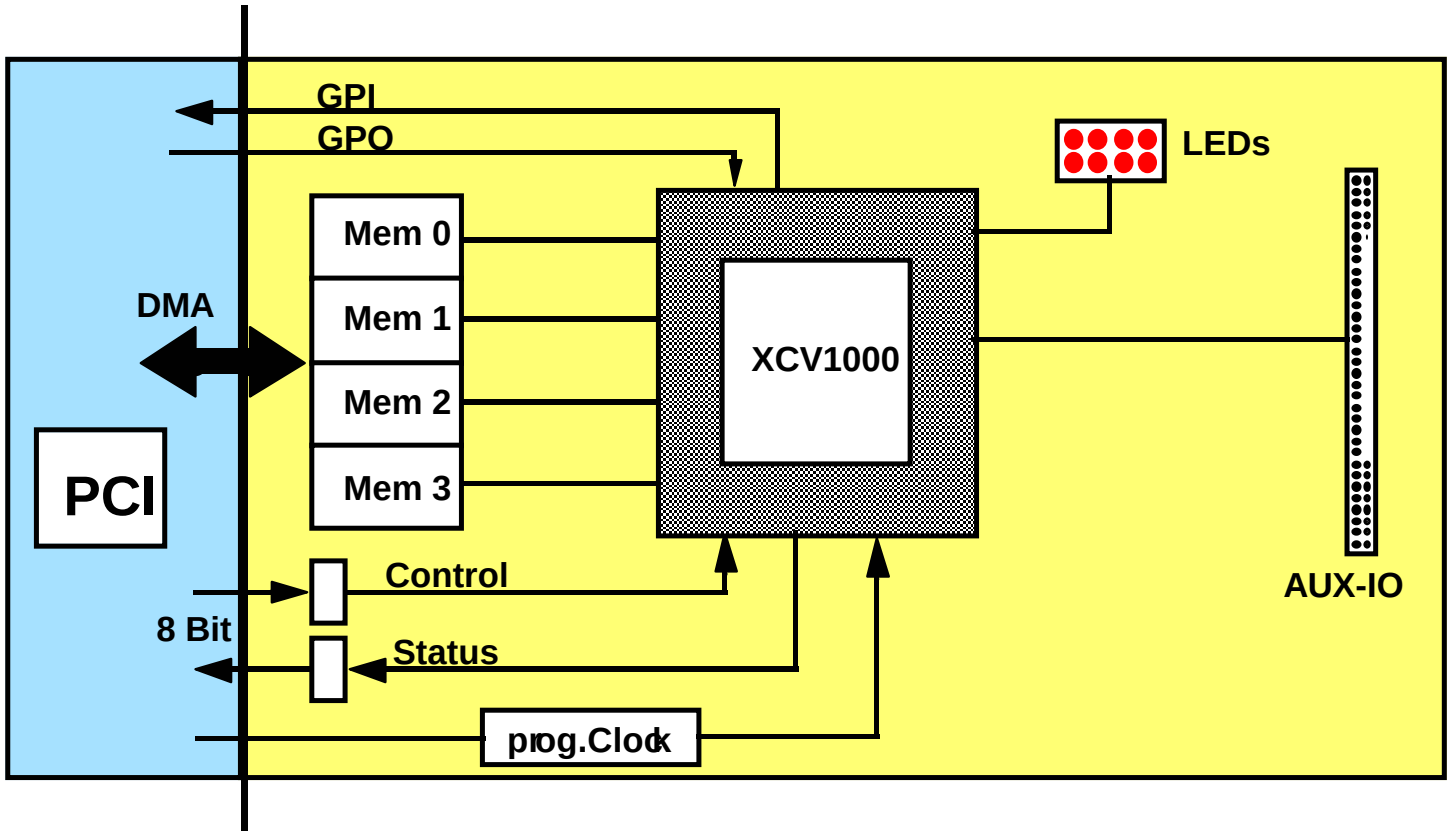
Handel-C Basics

- Aimed to map high-level algorithms to (reconfigurable) Hardware
  - **Rather a programming language than a HDL**
- C-like Syntax
  - **Standard control structures and operators**
  - **Data-types: int, fixed-point (with specific bitwidth)**
  - **arrays, structures also possible**
- Macro Support
  - **Standard CPP: #define (and: #include, #ifdef, ...)**
  - **Additional macro concept for HW modules**
- Explicit parallelism
  - **Through `par { }` construct**
- Independent processes possible
  - **Communication through synchronized „channels“**

⇒ ***First experiments done with existing Handel-C generated HW***

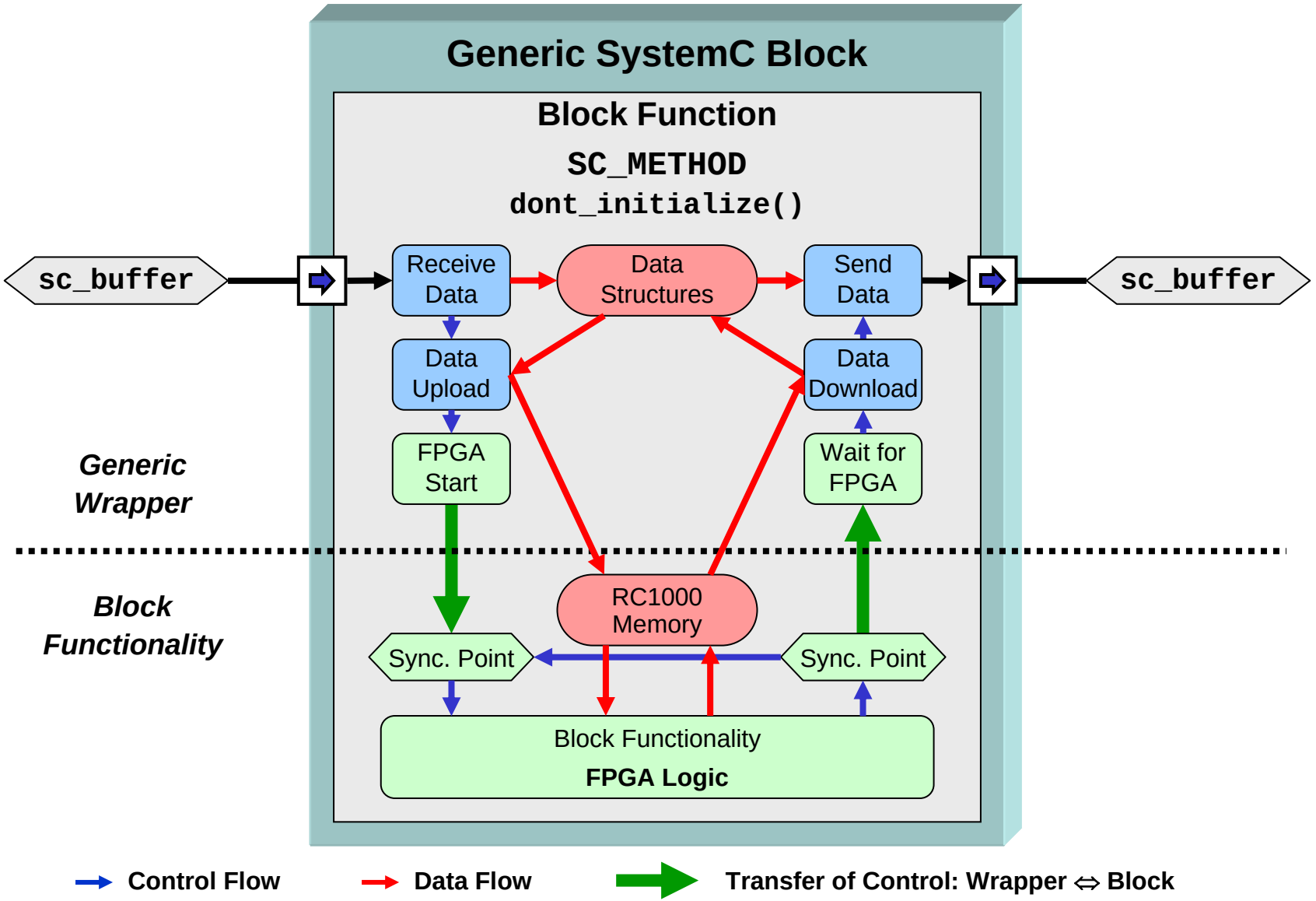
⇒ ***Todo: Comparison with SystemC-synthesized HW***

FPGA-Board: RC1000PP



- 1.000.000 Equivalent Gates (Xilinx XCV 1000 BG 560)
- 8 Bit control port for synchronised communication
  - Start execution on FPGA by writing port
  - Get status back from FPGA by reading port
- 8 MB RAM in 4 banks

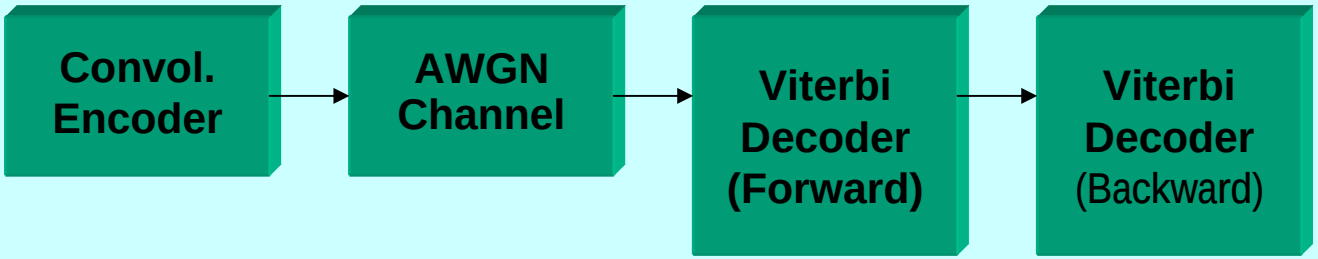
Integration of FPGA Hardware



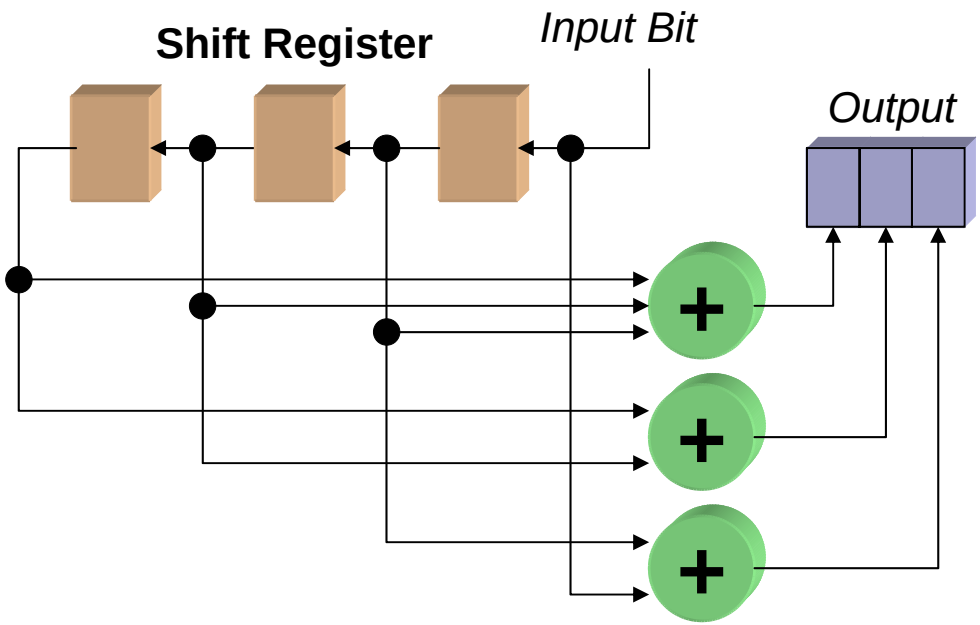
Example: Channel Simulation

A channel featuring Viterbi Coder/Encoder was chosen, because:

- the Viterbi algorithm is a standard and widely used.
- the algorithm demands many bit-level operations that require extensive programming in C vs. easy implementation in FPGAs.
- the channel simulation requires floating-point cosine root functions for noise modeling.



Viterbi Channel Coder



- Convolutional encoder for Viterbi Channel:
- Simple 1/3 code chosen (not an optimal code)
- Coding can be seen as sequence of FSM-state-transitions
- Very simple HW implementation
- SW implementation more difficult due to bit-level operations

Noise Function

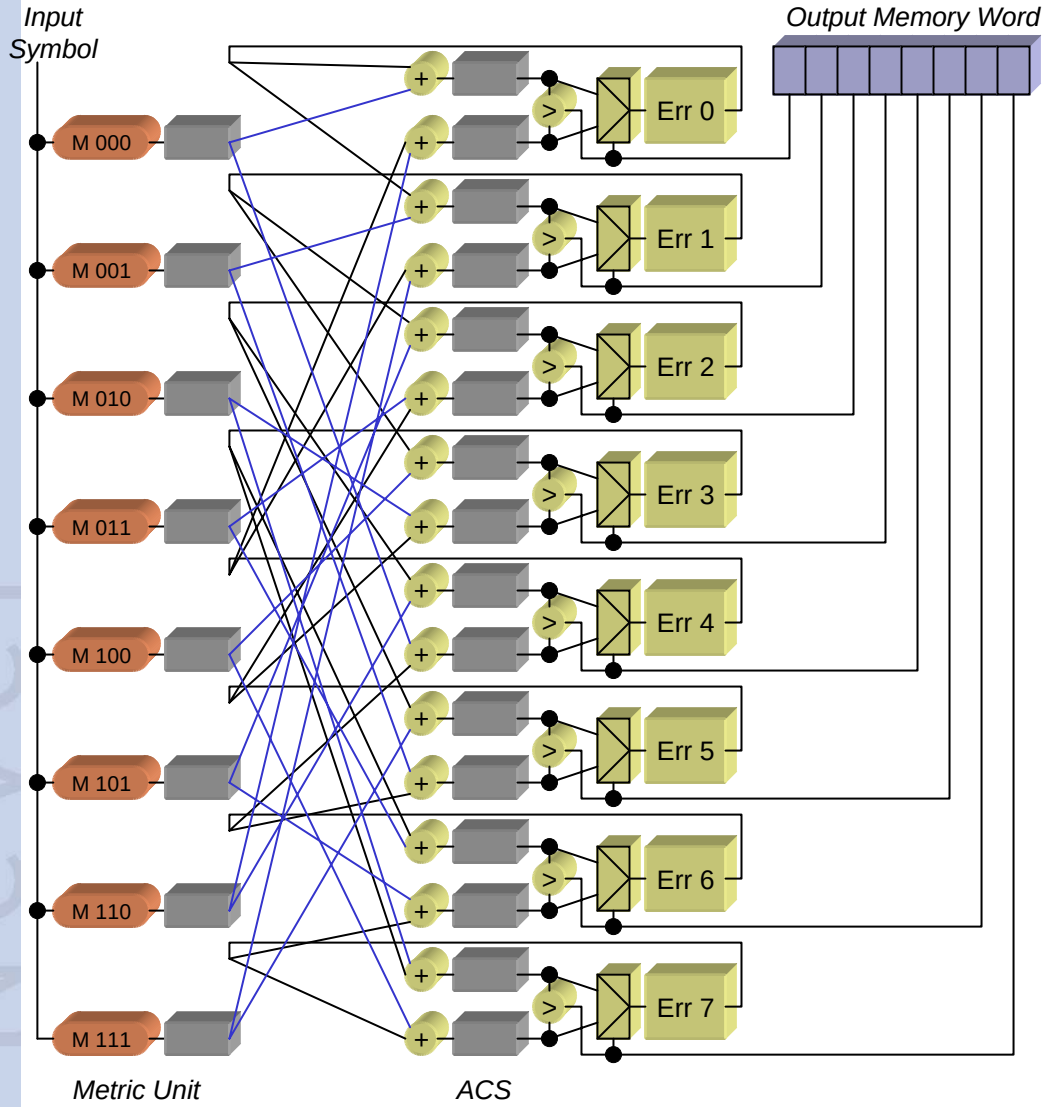
Simplified noise function:

Let U_1, U_2 be uniform random numbers (C func. `rand()`)

$$Noise = \sigma \cdot \sqrt{2 \cdot \ln \frac{1}{U_1}} \cdot \cos(U_2)$$

- Noise calculation is done in SW on the PC for both SW-only and HW/SW implementations
- Adding noise to the signal is done on the PC for the implementations without HW, on the FPGA for the HW/SW approach

Viterbi Decoder

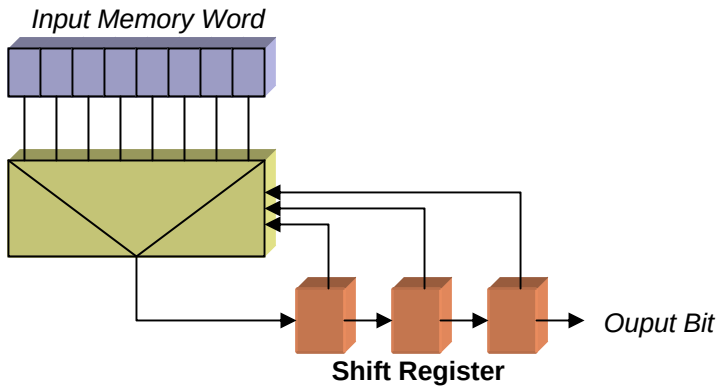


Viterbi algorithm reconstructs the message by finding the most likely sequence of transitions.

Forward phase (shown here):

- Eight-bit output word is stored for second phase in RC1000 memory
- Eight parallel metric units (hamming distance)
- Eight parallel Add-compare-select units
- Full shuffle backfeed
- Hamming-distance metric allows efficient hardware implementation

Viterbi Decoder (Backward Phase)



- Backward phase of Viterbi Algorithm:
- Processes the memory words from the forward phase back-to-front
- Very simple HW implementation
- SW implementation more sophisticated due to bit-level operations (multiplexer, shift register)

Experiment

We implemented different scenarios

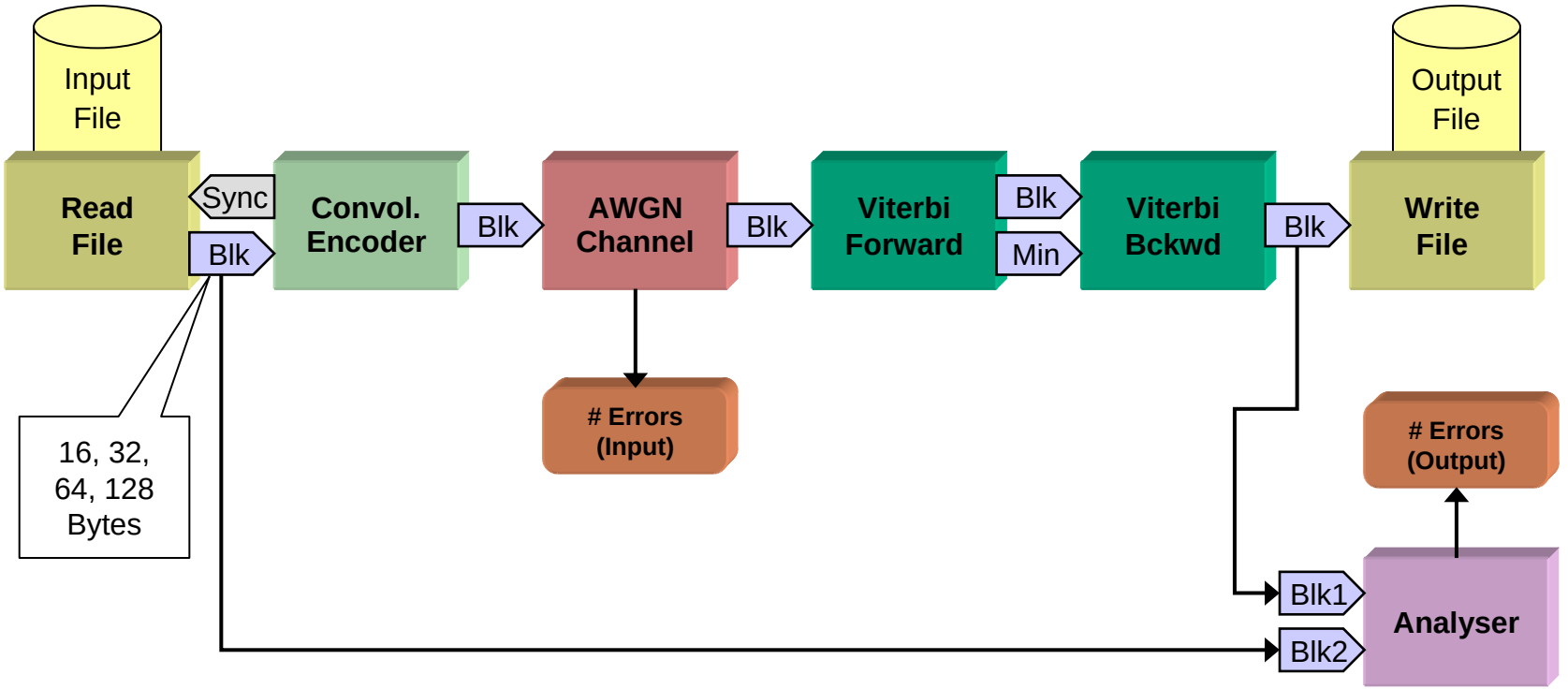
to measure execution speed-up:

- SystemC-only implementation
- SystemC with HW support

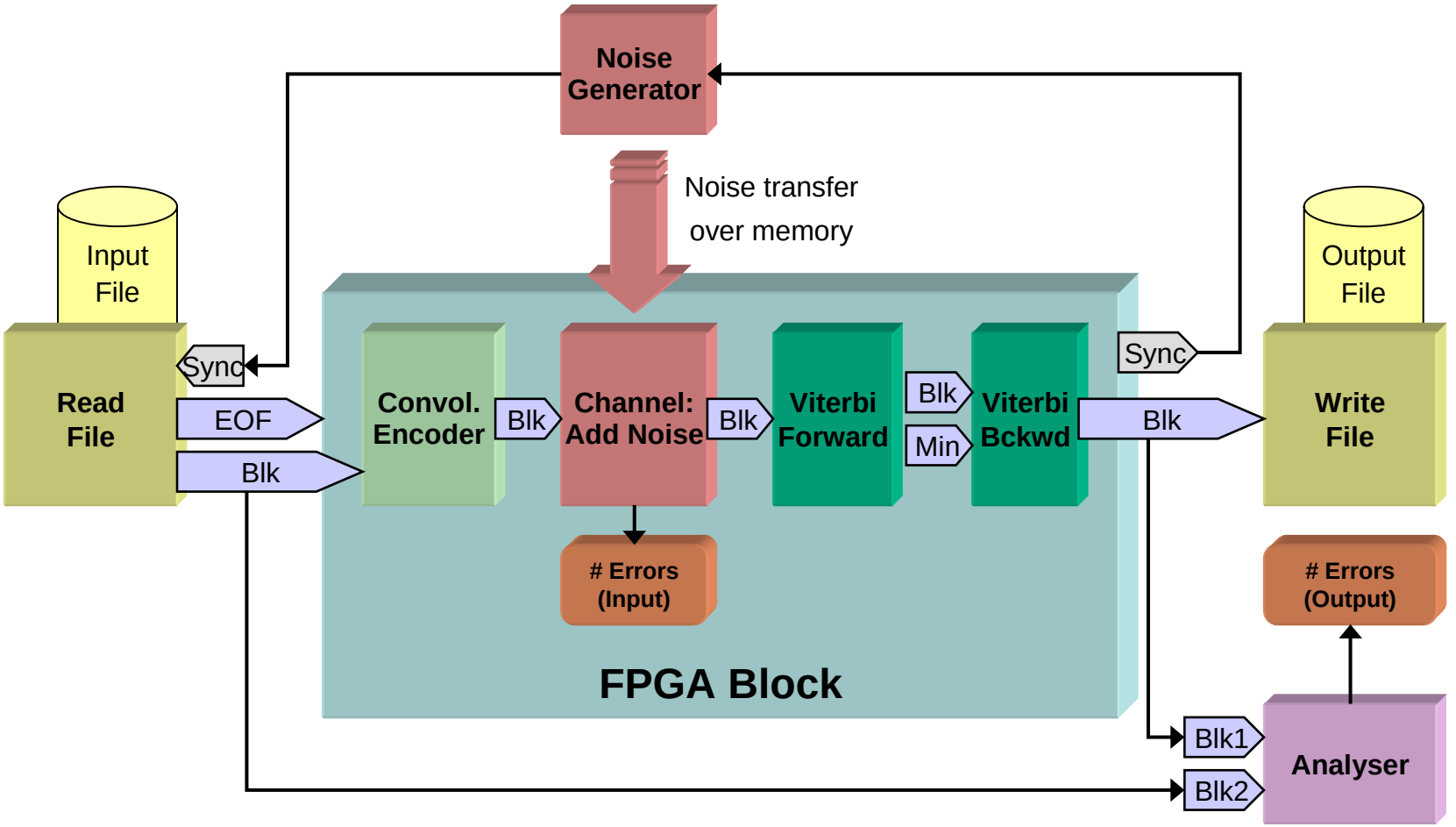
... and compared them, together with previous results:

- VCC implementation without HW
- VCC implementation with HW support
- „Maximum speed“ console implementation in C
  (no model abstraction)

Setup - SystemC



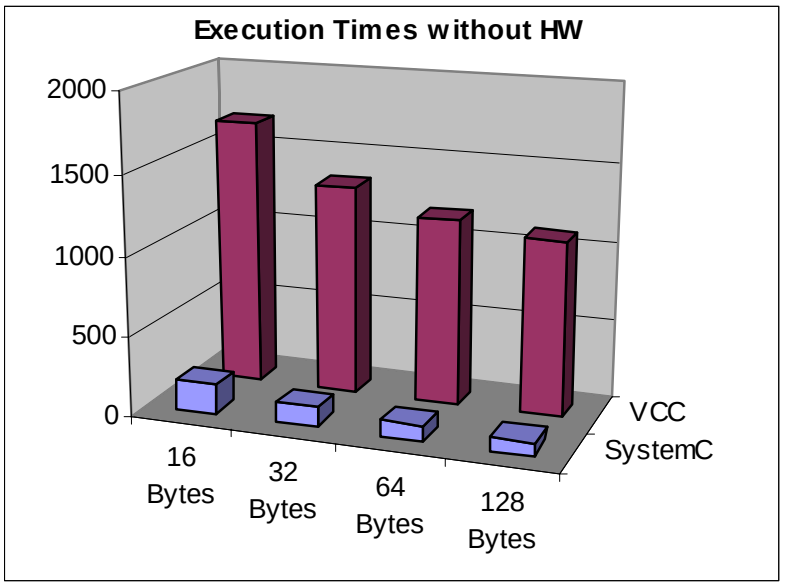
Setup – SystemC with HW

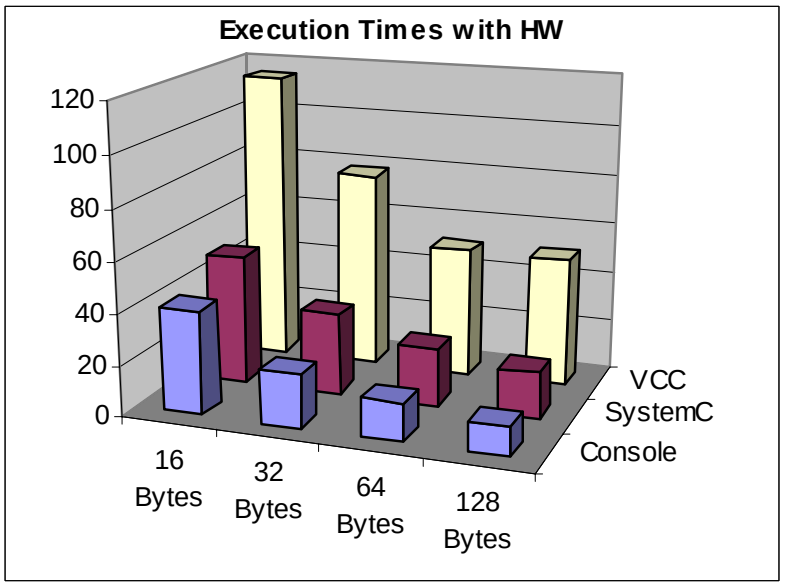


Results

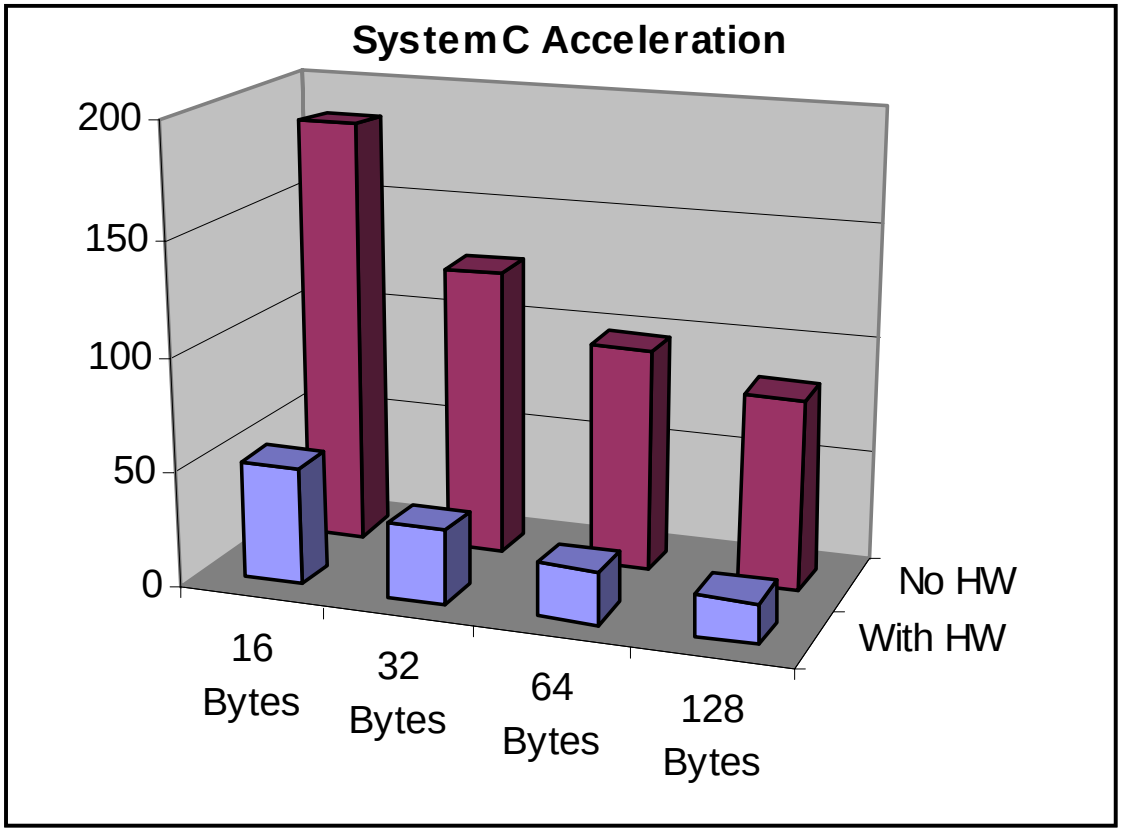
- Each experiment consists of seven runs with a 112201 Bytes File
- FPGA clock was 20 MHz

		Execution Time(s) for Block Sizes of			
		16 Bytes	32 Bytes	64 Bytes	128 Bytes
SW	SystemC	03:06	02:06	01:38	01:23
SW	VCC	28:11	22:06	19:38	18:21
HW/SW	Console	00:41	00:21	00:15	00:11
HW/SW	SystemC	00:51	00:32	00:23	00:18
HW/SW	VCC	01:55	01:18	00:51	00:51





Results: SystemC Acceleration



- Average acceleration: 4.11 x
- Range from 3.6 to 4.6

Conclusion and Future Research

- Integrated HW components at system-level
- Low-cost solution
- SystemC shows relatively slow performance decrease compared to pure C solution (no graphic interface)

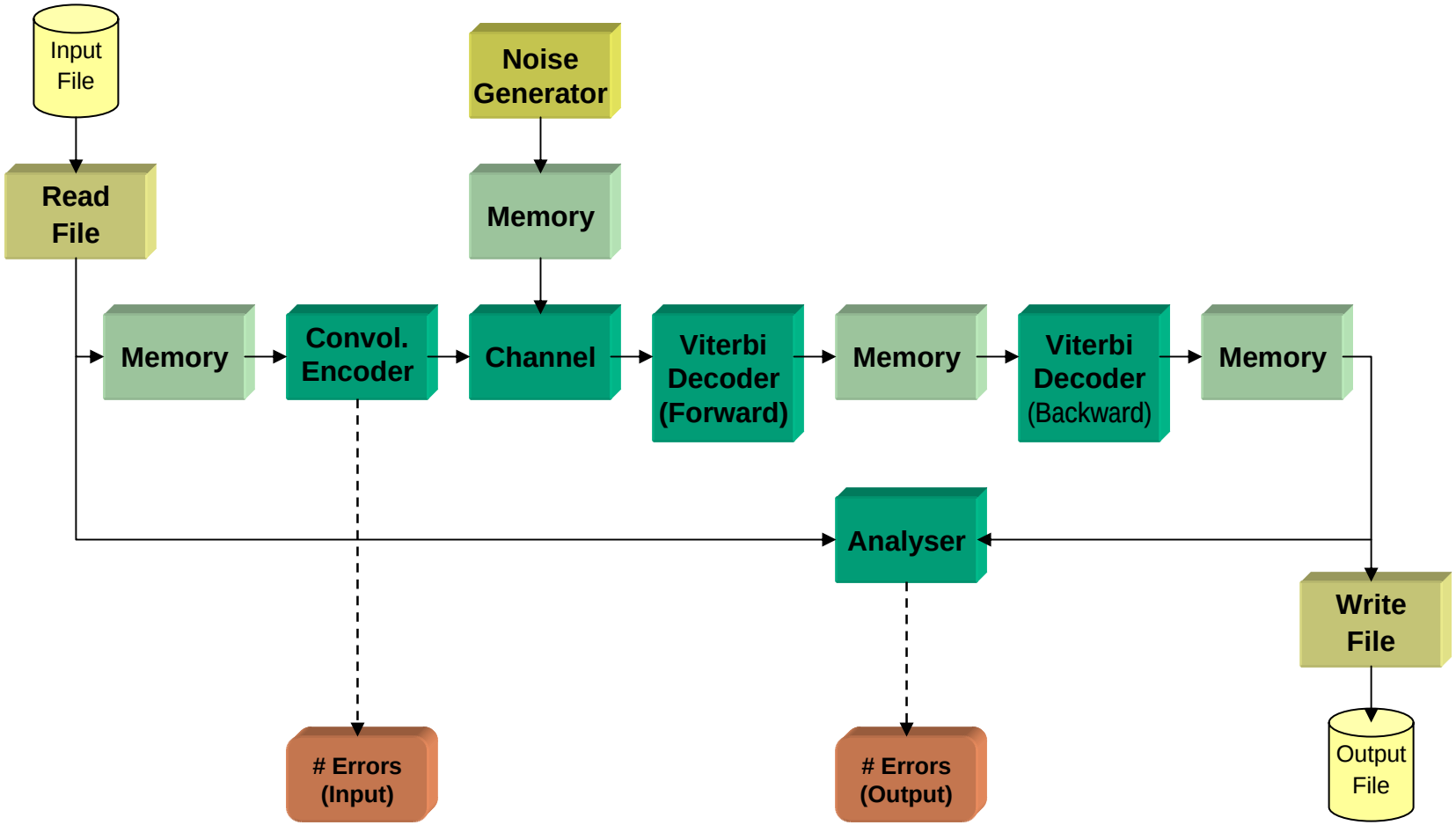
- Future work targets to include also other hardware (e.g., complete processor boards) into a system-level simulation
- Step towards unified methodology:
  Use SystemC for HW synthesis

Thank You !

Migration C to Handel-C

- RC1000 board features only one control port
  - **need to merge several SystemC blocks into one, resulting in less flexibility**
  - **communication links modeled using Handel-C channels**
- Control constructs / calculation can mostly be reused
  - **Exception: pointers become explicit RAM accesses**
- Main work: utilizing parallelism / optimization
  - **Done by inserting appropriate constructs ("par") and manual loop unrolling**
  - **Partially supported by macro concept**
- Migration from word-level to bit-level operators
  - **Relatively simple, as C-notation can be retained**
  - **Additional support by special Handel-C constructs**

Setup – Console HW/SW



Legend: Functional Block in SW | Functional Block in HW

Handel-C to FPGA