MICRONAS


# FPGA Prototyping of a SystemC-Specified Video Design

# Introduction

MICRONAS

Design Scope:


Need to see frame sequence:

- many different sources (DVD, antenna, VHS, …)
- many different design parameters/operating modes

Simulation is not sufficient for algorithm development

→ Need real-time FPGA execution

# Why SystemC

MICRONAS

- It‘s C/C++
- Simulator kernel is open
- Toolchain exists from multiple vendors
- Easy integration/interfacing to existing C-IP

Why not SystemC?

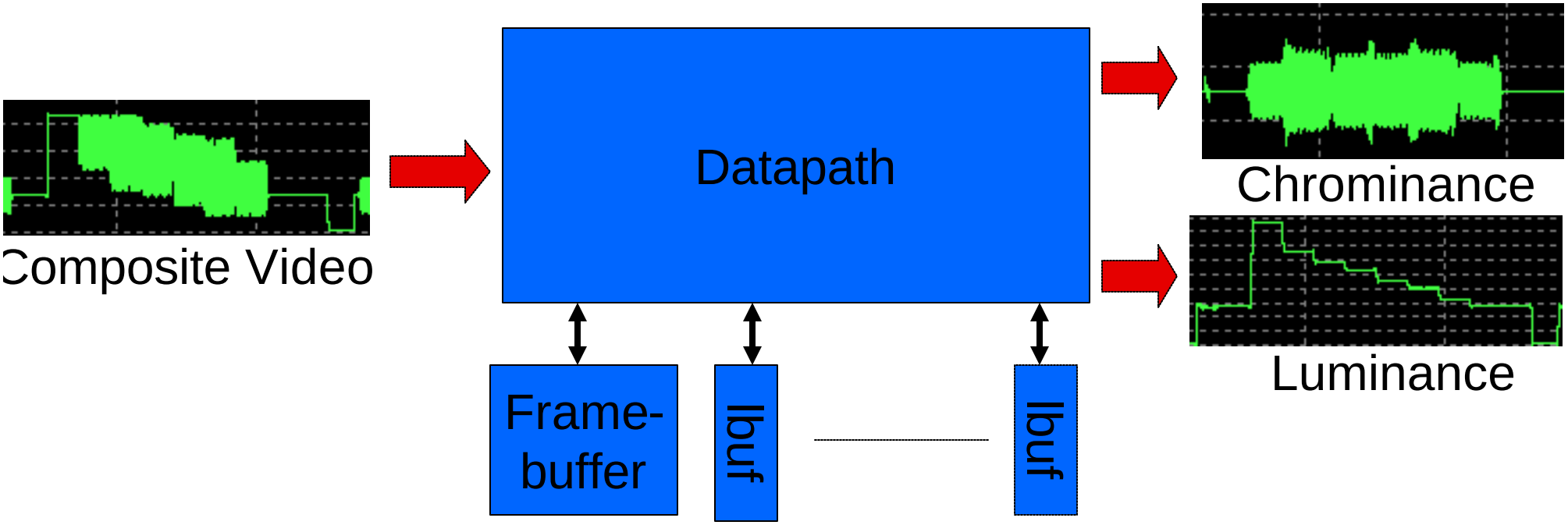
# Outline

MICRONAS

- Design properties
- Tool chain
- From SystemC to FPGA
- Wishlist
- Conclusion

# Design (3D Comb Filter)

MICRONAS



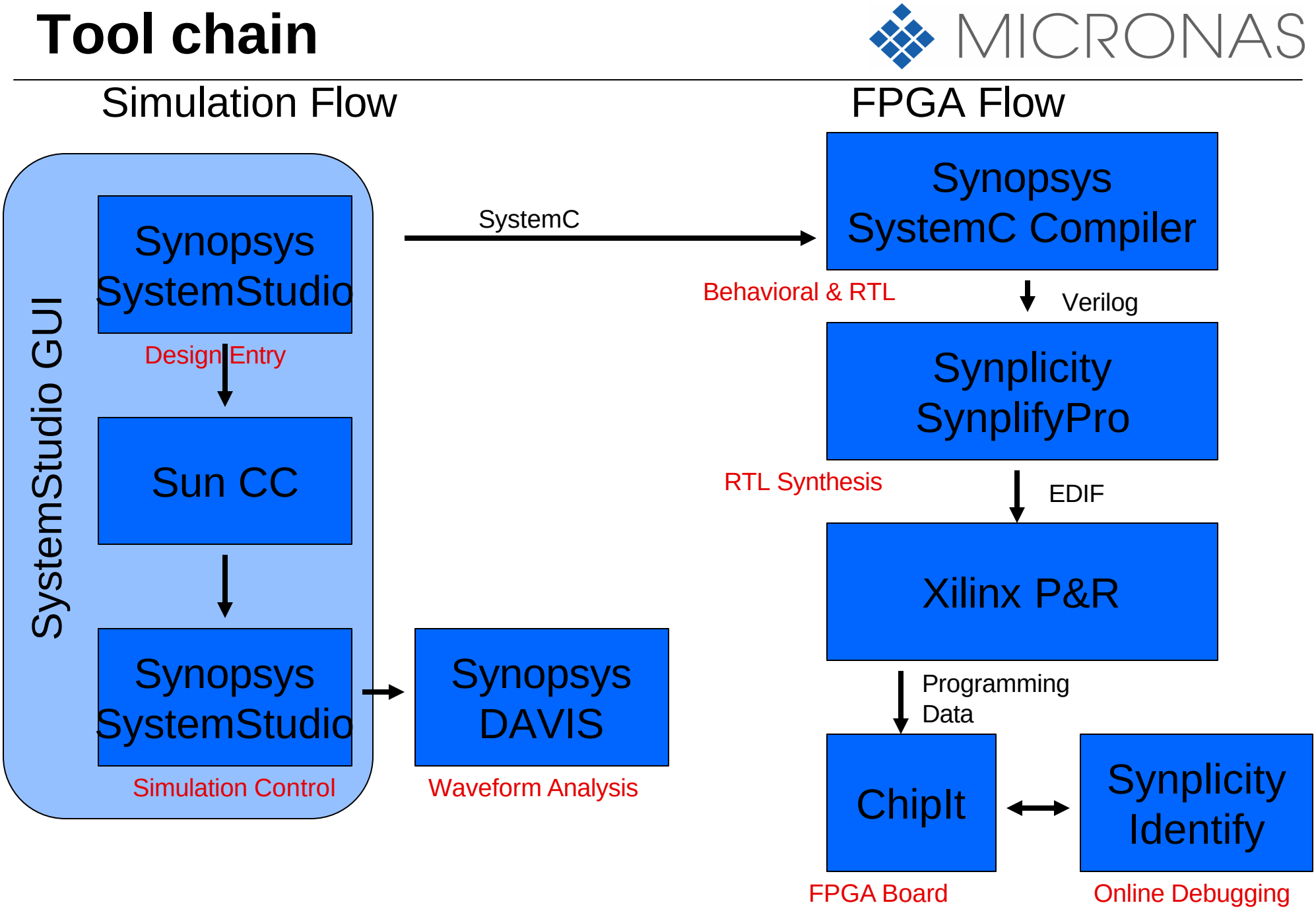

95% dataflow (DSP)

slow clock (20.25 Mhz)

~70 leaf modules

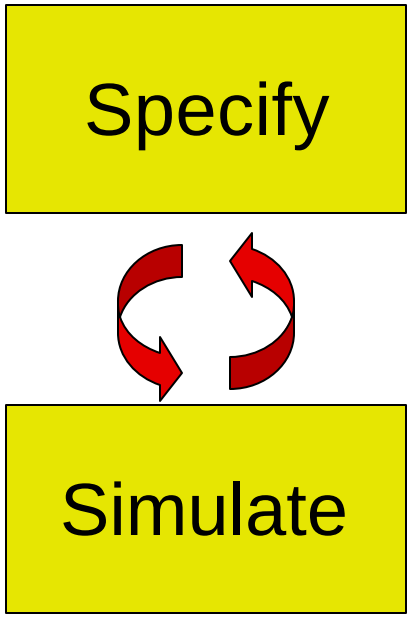
part of an SOC

# Tool chain


Simulation Flow
FPGA Flow
SystemStudio GUI
Synopsys SystemStudio
Design Entry
Sun CC
Synopsys SystemStudio
Simulation Control
SystemC
Synopsys DAVIS
Waveform Analysis
Synopsys SystemC Compiler
Behavioral & RTL
Verilog
Synplicity SynplifyPro
RTL Synthesis
EDIF
Xilinx P&R
Programming Data
ChipIt
FPGA Board
Synplicity Identify
Online Debugging

# Strategy (1)

MICRONAS



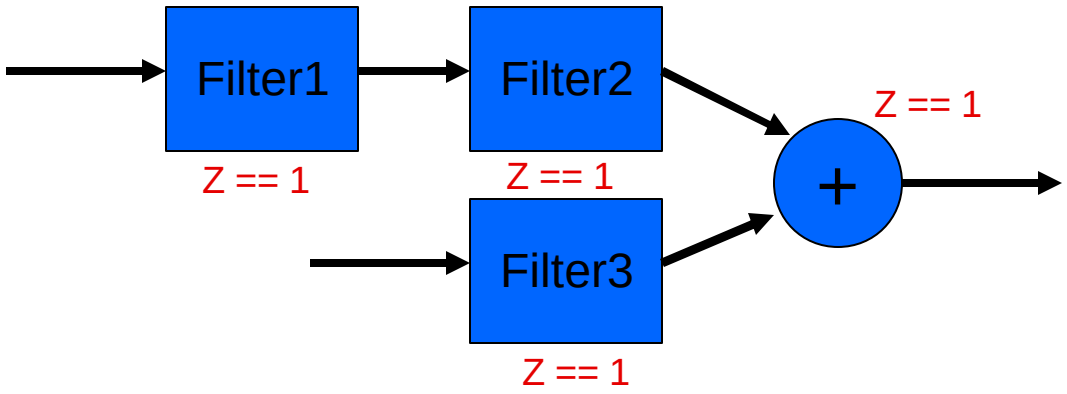

Specification:

- starting from Schematic view
- all blocks with SC_METHOD()
- latency always one clock cycle
- mostly bitwidth accurate modelling
- no consideration of synthesis codingstyle

# Specification

MICRONAS

Top-down hierarchy:



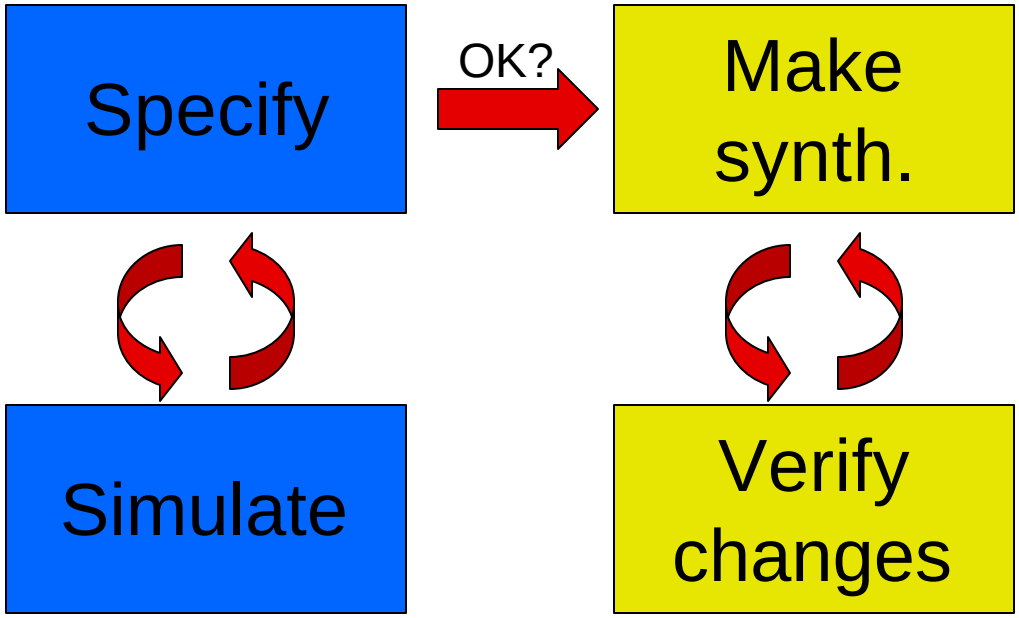

Blocklevel:

```
SC_MODULE(Filter3) {
  ...
  SC_CTOR(Filter3) {
    ...
    SC_METHOD(action);
    sensitive << clk.pos();
  }
  ...
}
```

```
void Filter3::action(void) {
  // Filter delay loop
  // Read input value
  // Compute filter function
  outp.write(<value>);
}
```

# Strategy (2)

MICRONAS



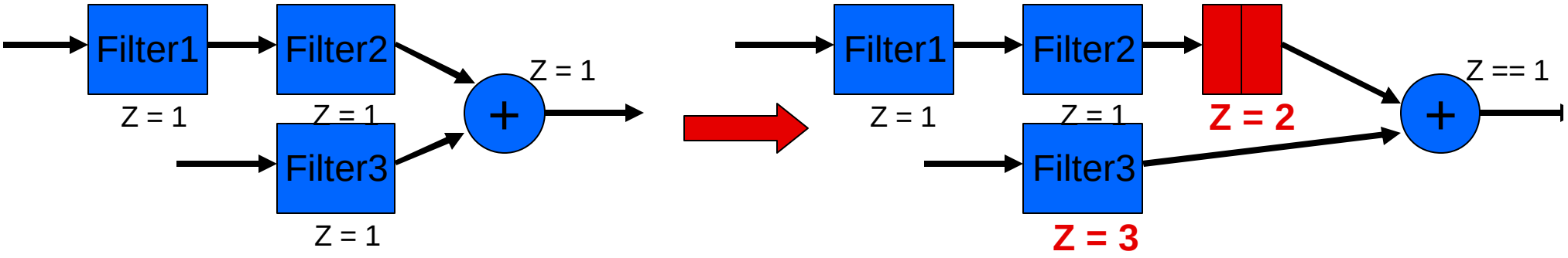

Make synthesizable:

- adjust codingstyle (e.g. `sensitive_pos` VS. `sensitive`)
- determine block latencies (iterative)
  - latency > 1: pipelined SC_CTHREAD
- re-balance parallel paths in datapath
- resolve tool issues

# Make synthesizable

MICRONAS

## Latency balancing (hierarchy):

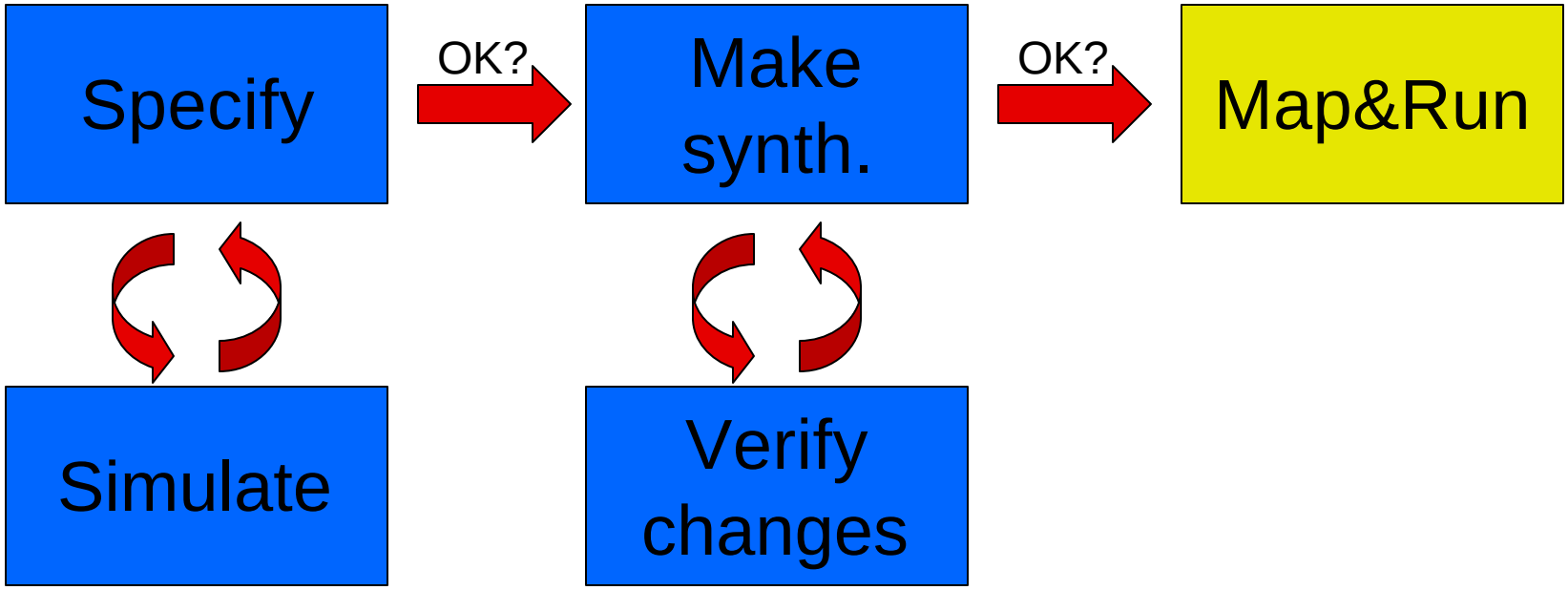



## Blocklevel:

```
SC_CTOR(Filter3) {
  SC_CTHREAD(action,clk.pos());
  watching(reset.delayed());
  ...
}
```

```
void Filter3::action(void) {
  // Reset path
  wait();
  while(1) {
    // Filter delay loop
    // Read input value
    // Compute filter function
    // Write output value
    MC_PIPELINE_OUT(3,outp,<outval>);
  }
}
```

# Strategy (3)




Map & Run:

- create peripherals
  - I2C, SDRAM, A/D D/A, clocking
- resolve tool issues
- debug & play

# Tool issues

MICRONAS

SystemC -> Verilog:

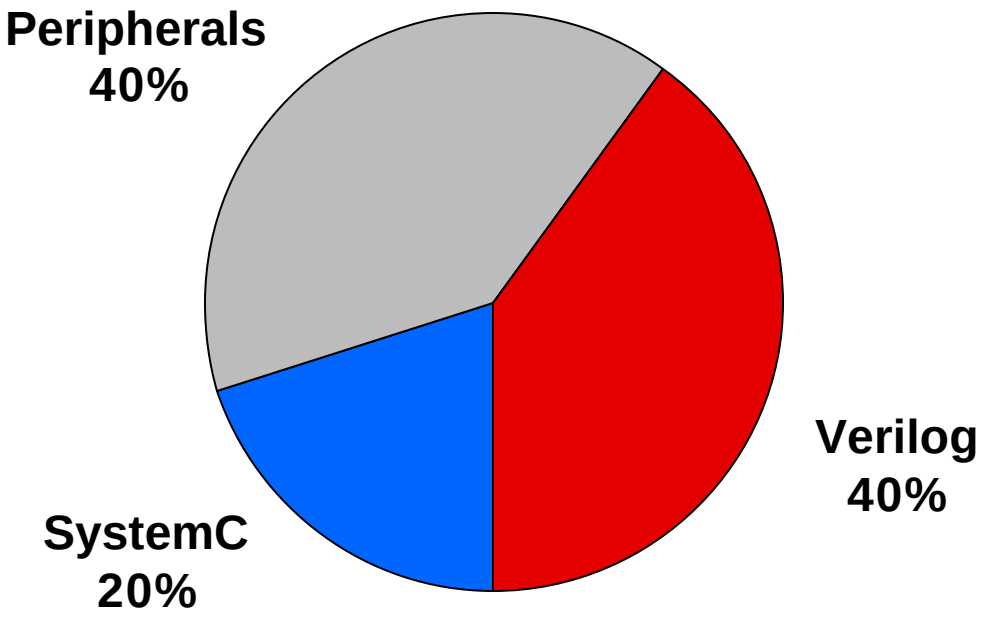
- bug in RTLout -> need 2 different tool versions
  -> need to write VHDL/Verilog-mix
- illegal Verilog in one module

Verilog -> EDIF:

- LRM interpretation:
  -> rewrite reg signed [w-1:0] arrayvar [l-1:0]
  -> need to be careful with $signed(), $unsigned()
- bugs in Verilog frontend:
  -> need to manually rewrite several modules
- will be fixed in upcoming versions

# Overall mapping effort

MICRONAS




- SystemC: synthesis-codingstyle and latency balancing
- Peripherals: SDRAM-Interface, I2C, Clocking
- Verilog: fixing tool issues

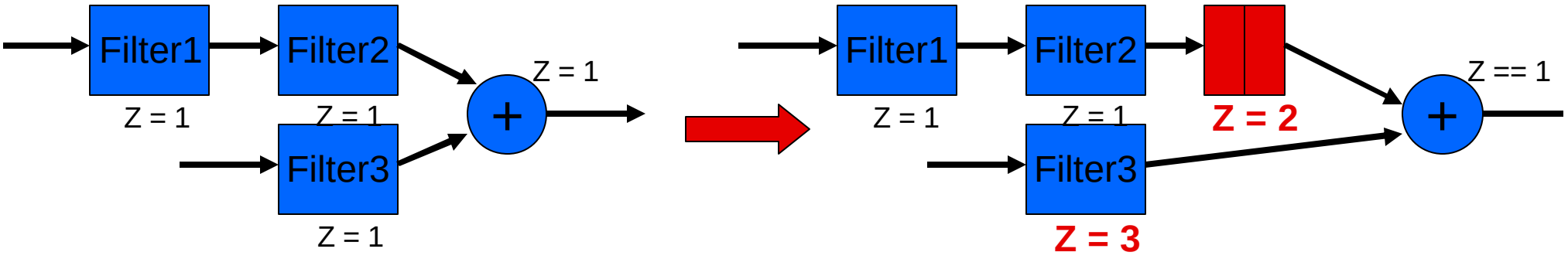
# Wishlist

MICRONAS

Flexible SystemC codingstyle:

- support everything that´s statically determined
  i.e. inheritance, templates, constructor arguments, …
- standardized synthesis codingstyle

Automatic latency management:



# Conclusion

MICRONAS

- It works
- Problems were not where anticipated
- We will continue doing this
- Increased confidence in the algorithms
- Fun-factor not to be underestimated