



# Nanometer Wireless Transceiver Modeling Using SystemC & Verilog-AMS

Martin Hujer

Mark Barry, Brendan Walsh, Radek Manasek, Jerry O'Mahony, Patrick Feerick

BMAS conference  
15<sup>th</sup> September 2006

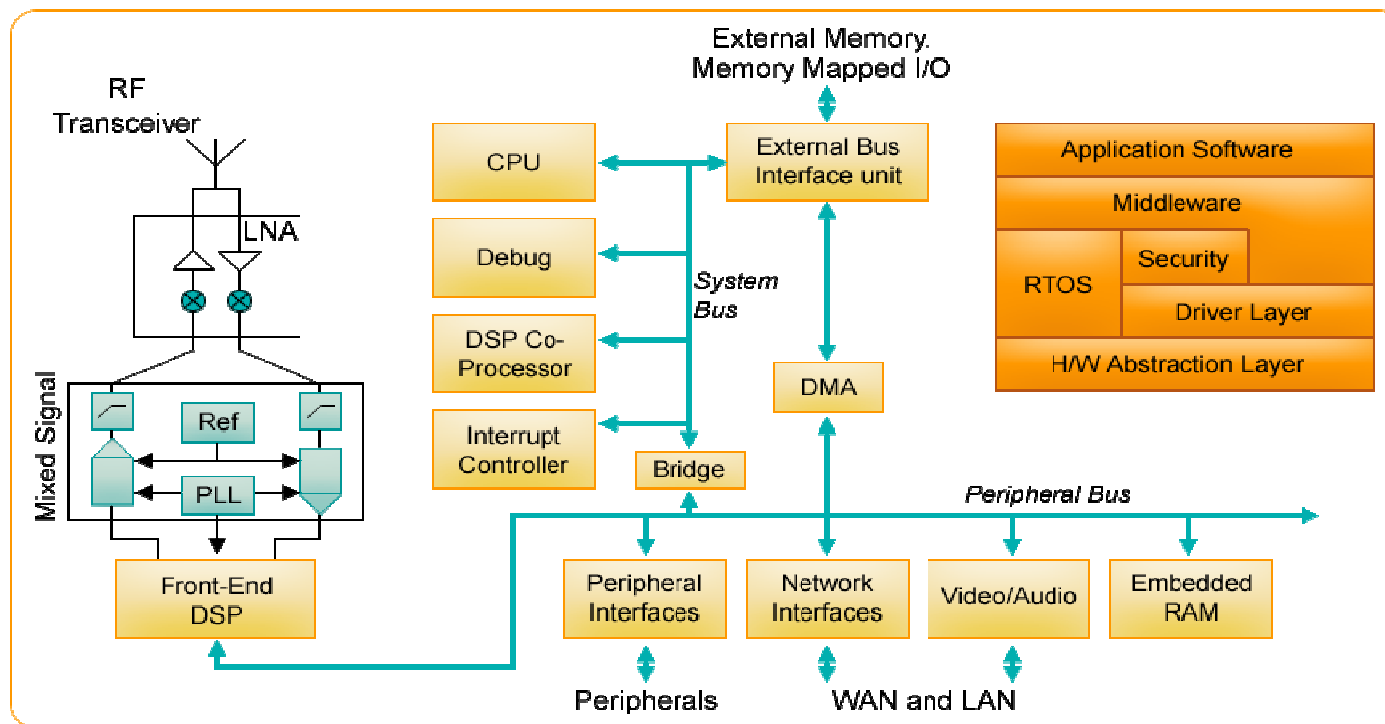
Silicon & Software Systems Ltd.

## Overview

- **Introduction**
- **Wireless transceiver architecture**
- **Top level model**
- **Verilog-AMS block model example**
- **SystemC block model example**
- **Example analysis using model**
- **Simulation results**
- **Conclusion**

# Introduction

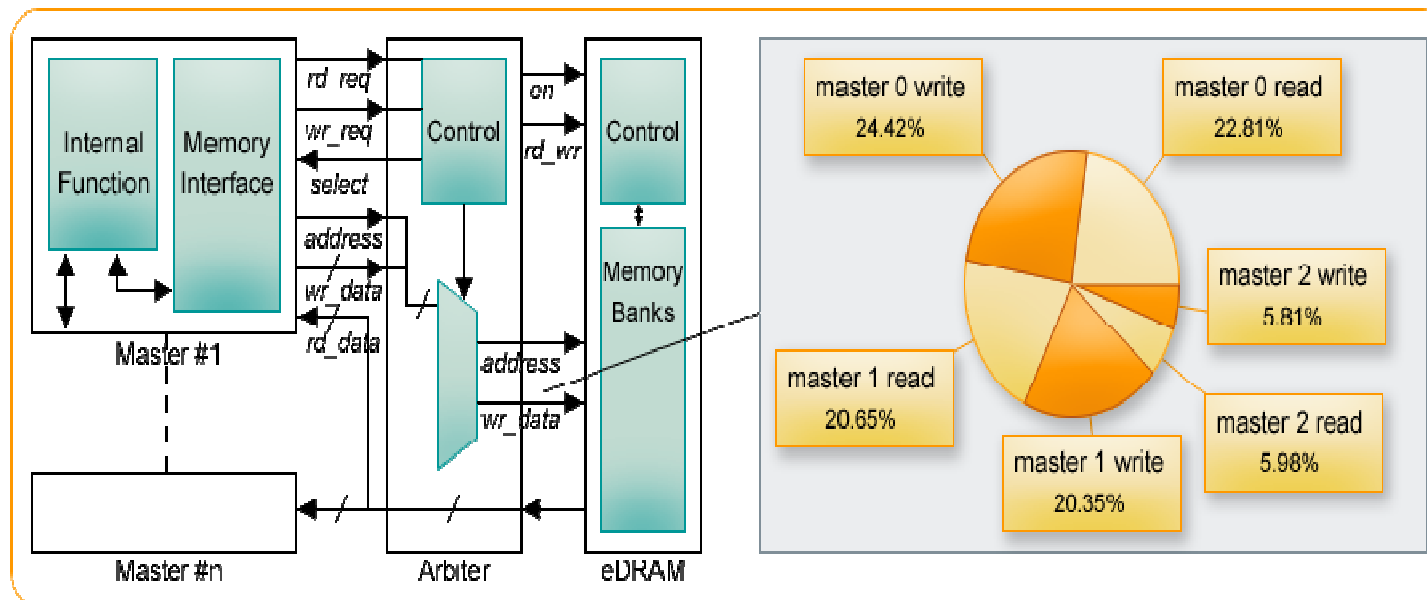
- **Trend towards increasing integration of RF, Analog and Digital**
  - In fast growing area of wireless communications
  - Due to advances in nanometer CMOS processes
  - Driven by cost, form factor and in some cases power consumption



## Motivation (I)

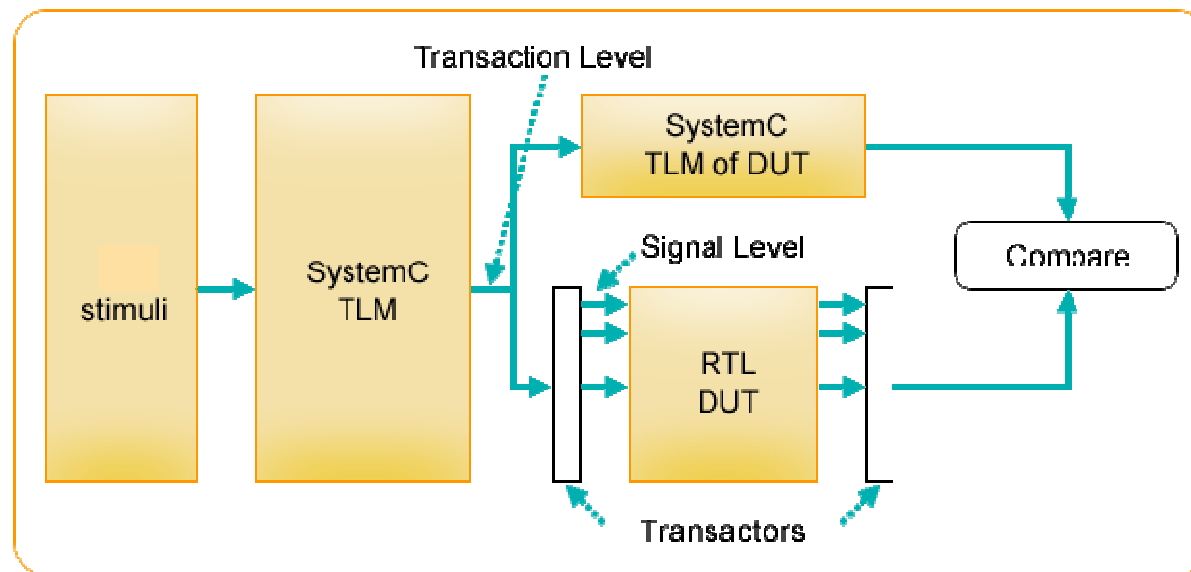
### ■ Why Verilog-AMS and SystemC?

- Verilog-AMS is faster to write and simulate than transistor level schematic
- SystemC is faster to write and simulate than RTL
- IC architectural analysis months in advance of implementation available
- Industry standard languages



## Motivation (II)

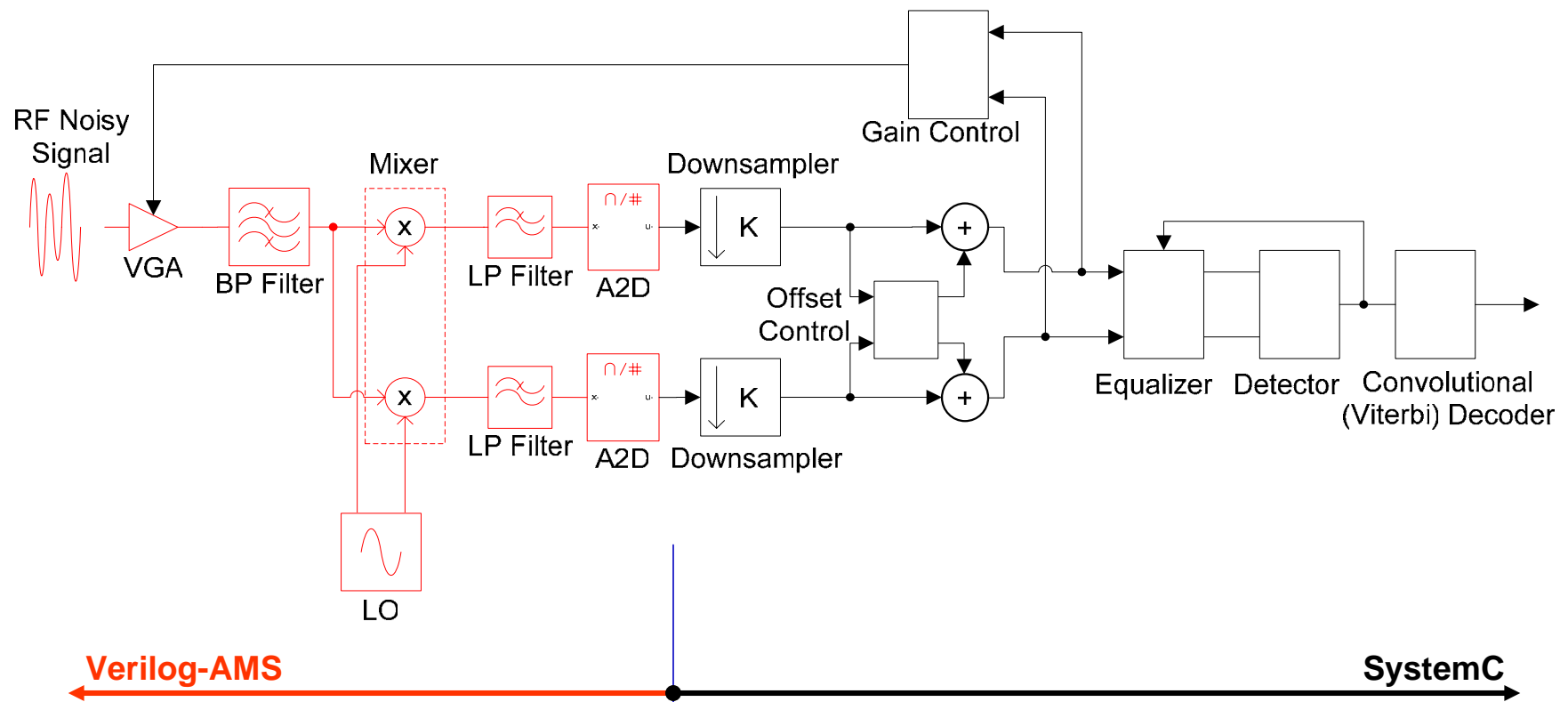
- **Verilog-AMS & SystemC can be re-used to verify HDL and transistor level design**
  - Check connectivity with simulations
  - Use as a test harness to drive the implementation during verification
  - Use as a golden reference model to verify implementation functionality
  - Excellent path to implementation!



# Wireless Receiver

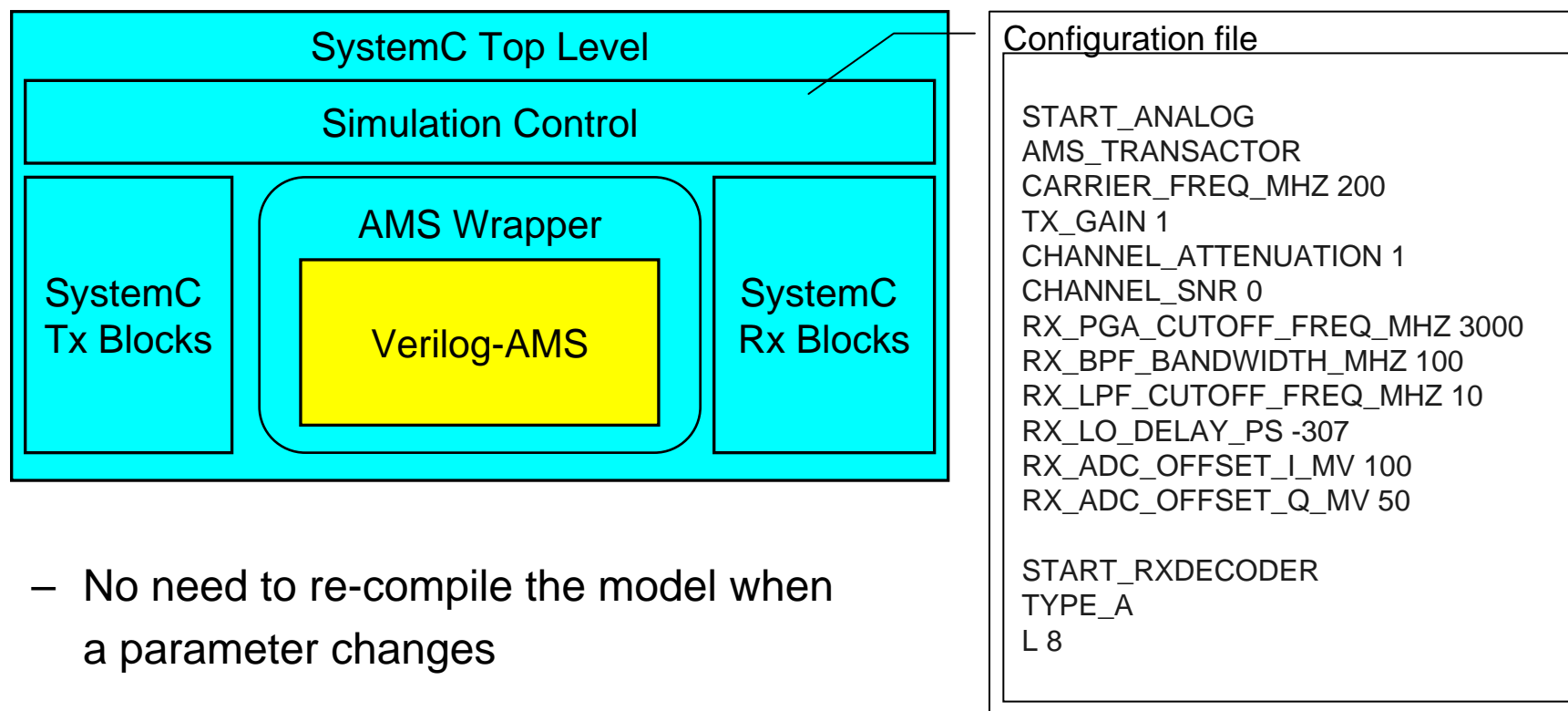
## Goals

- Model a wireless direct conversion receiver (for highly integrated solution)
- Highly configurable model used for early architectural analysis



## Top Level Model

- **Top level simulation is in SystemC run in Cadence NC-SIM**
  - SystemC wrapper for Verilog-AMS auto-generated by *ncshell*
  - Results can be written out and analyzed in Matlab

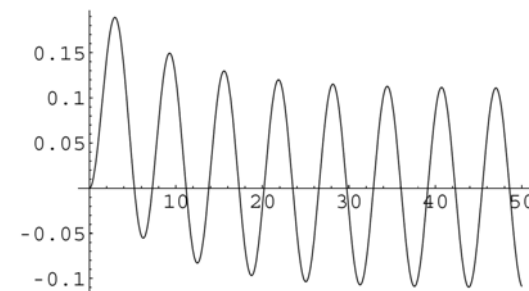


- No need to re-compile the model when a parameter changes

## Verilog-AMS Example: LP Filter

- **Firstly a theoretical model is derived for a generic filter**
  - Can be designed in Mathematica or Matlab etc
  - Fifth order Bessel filter

$$H(s) = \frac{p1.p2.p3.p4.p5}{(s+p1)(s+p2)(s+p3)(s+p4)(s+p5)}$$



- **The Verilog-AMS model is written for a generic filter**
  - Only need to define the poles to get a specific filter

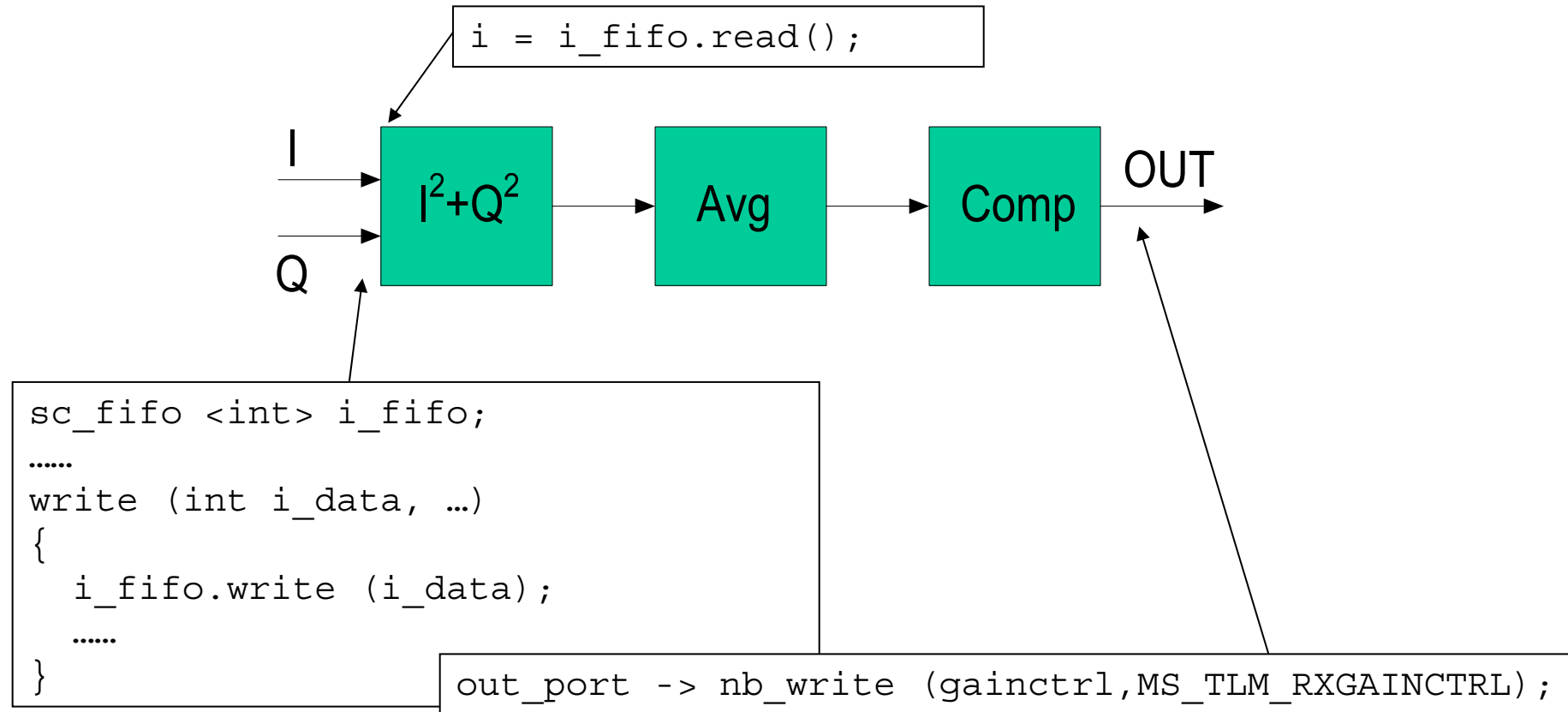
```
voutVar = laplace_np( V(...),
                    {1},
                    {rp1 * `M_TWO_PI*f0, ip1 * `M_TWO_PI*f0,
                     etc }
                    );
```

- **Takes less than an hour to define a new filter!**



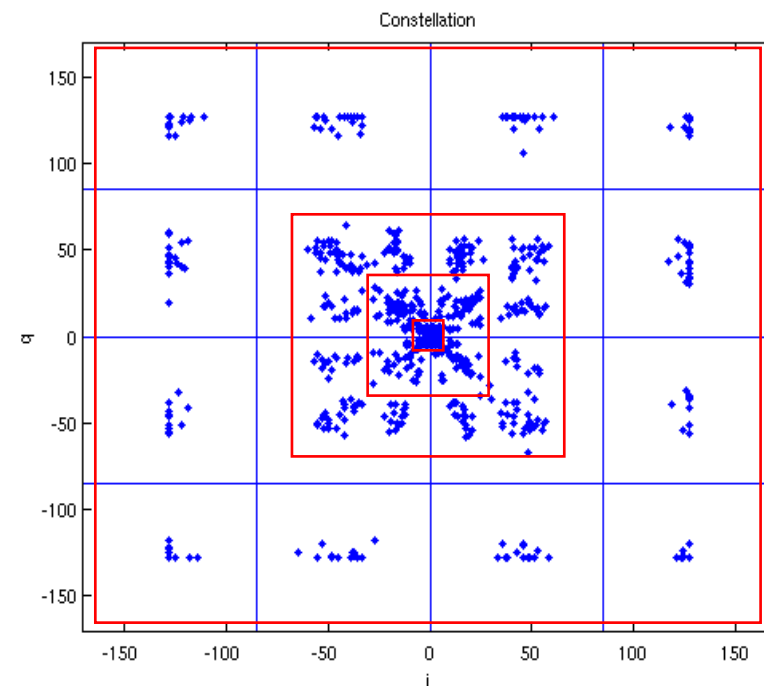
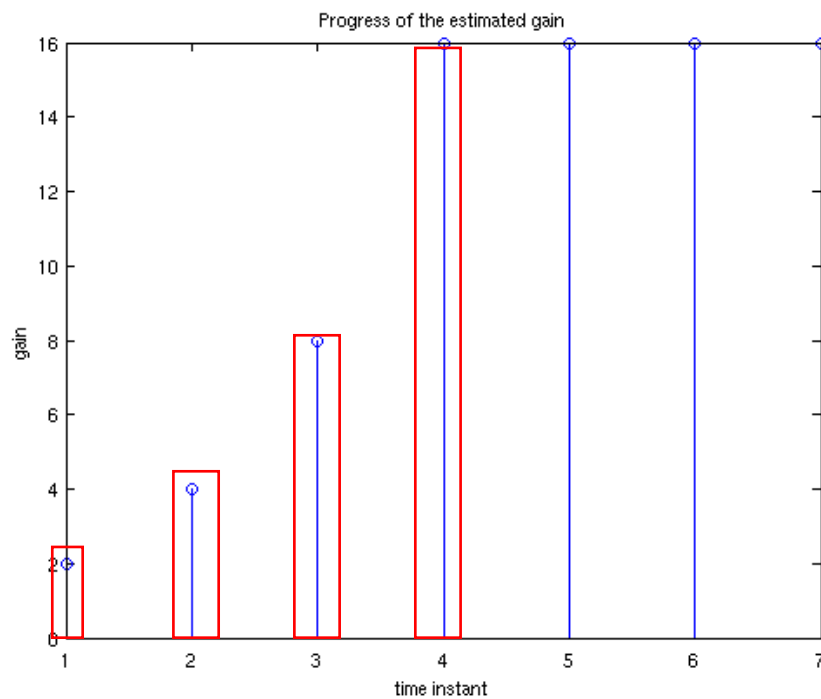
## SystemC Example: Gain Control

- Code for gain controller block is written in SystemC
  - Much faster to write than RTL and much faster to simulate



## Analysis: Gain Control

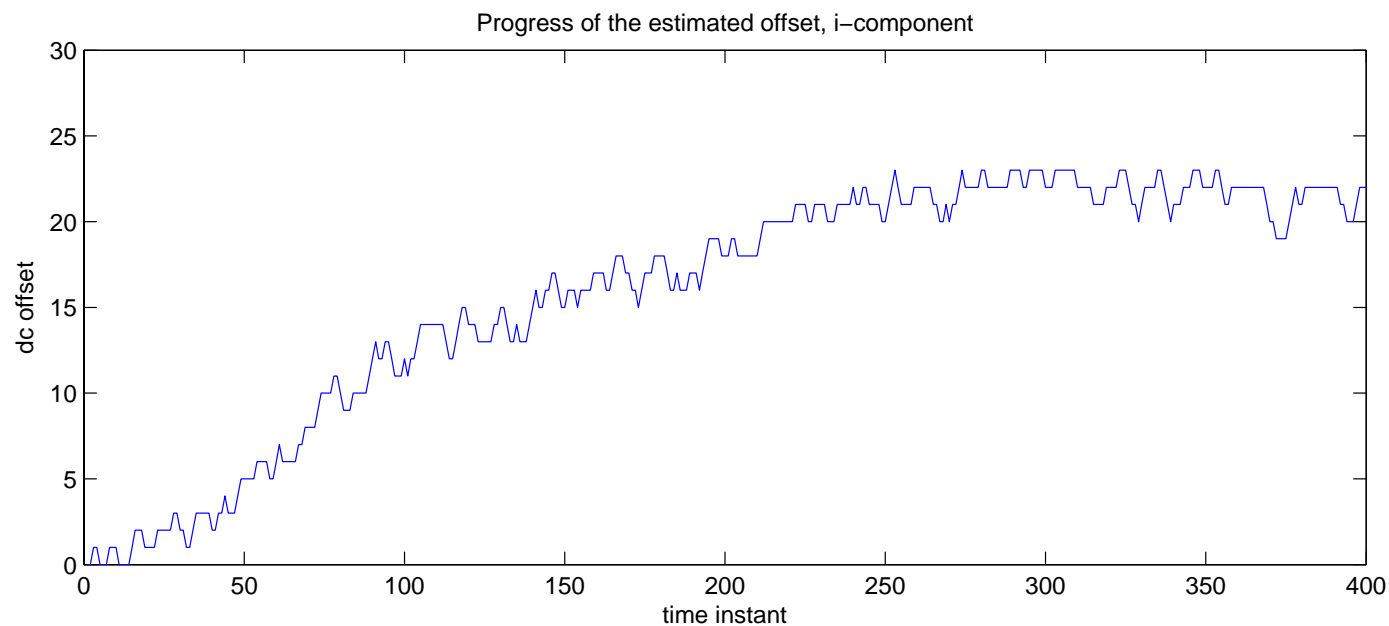
- Snapshot of gain control locking in AGC at start of simulation



- Possible modeling of transient or parasitic effects, e.g. when the gain of the VGA is switched

## Analysis: Circuit Offset

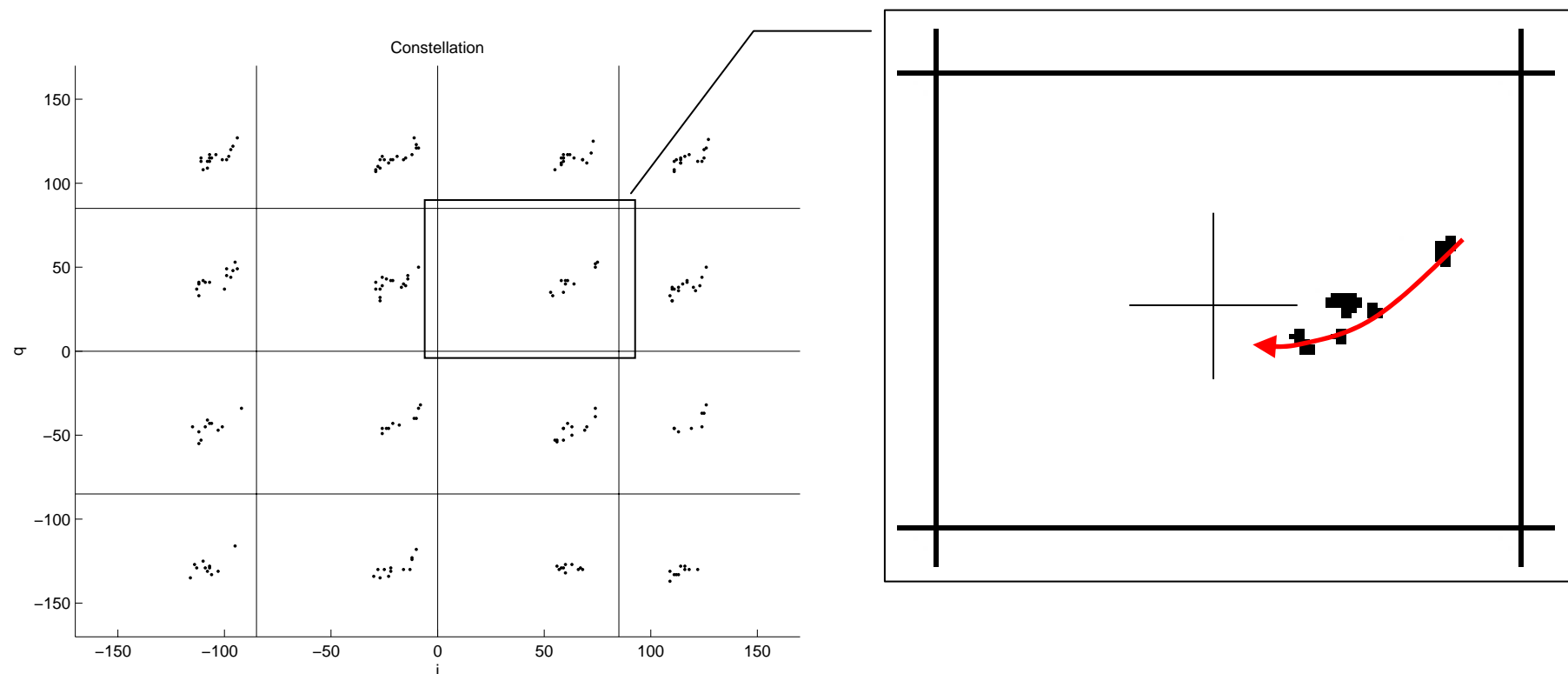
- **Offset in analog circuits (post mixer)**
  - Voltage shift of I and Q components
  - Mixer, LPFs, ADCs
  
- **Offset control block**
  - Estimates current offset for following elimination



## Analysis: Circuit Offset

- **Constellation diagram**

- Shows the offset correction algorithm converging to the centre point

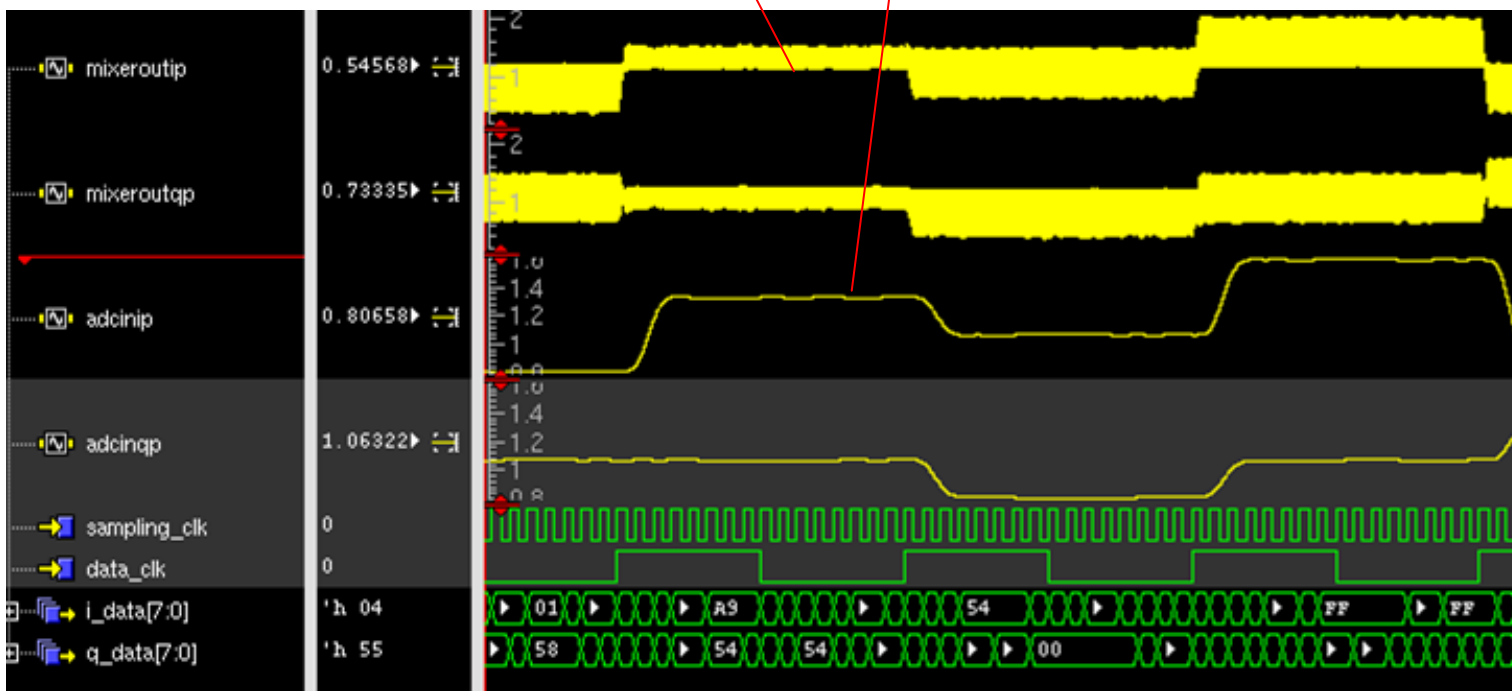


## Example of Waveforms

- Transient waveform from Cadence

I & Q input of the LPF  
- Baseband + 2x carrier frequency

I & Q output of the LPF  
- Baseband only



## Simulation Results

- Dual 64bit 2.6GHz Opteron™ processors
- 7.6GB RAM
- Red Hat Enterprise Linux 3

| Carrier Frequency | Analysis Time | CPU Time |
|-------------------|---------------|----------|
| 200 MHz           | 10 $\mu$ s    | 29 s     |
| 2.4 GHz           | 10 $\mu$ s    | 171 s    |
| 2.4 GHz           | 50 $\mu$ s    | 815 s    |

- $\uparrow$ Analysis time  $\rightarrow$   $\uparrow$ CPU time,  $\uparrow$ Carrier frequency  $\rightarrow$   $\uparrow$ CPU time
- Simulation of the analog part is the most time consuming

## Conclusion

- **It is very fast to write Verilog-AMS & SystemC models**
  - Much faster to write than RTL or schematics
  - Allows early architectural analysis before implementation is available
- **Used industry standard tools and languages**
  - Well supported ecosystem of tools and stable languages
  - Good interfaces to other tools such as Matlab, RTL simulators etc
- **Good simulation speed even for complex transceivers**
  - It is highly dependent on carrier frequency and analysis time
  - Possible to do comprehensive mixed mode analysis such as on AGC
- **Transceiver model is highly configurable**
  - Significant changes to modulation scheme etc can be made very quickly
  - Perfect for looking at trade-offs in early architectural analysis

## Questions

**Thank you for your attention.**



Martin.Hujer@s3group.com  
info@s3group.com  
www.s3group.com



## Appendix 1: Verilog-AMS Example

```
module ms_tlm_rxlpfilter ( voutm, voutp, vdd, vss, vinm, vinp );

    parameter real rp1 = -1.3851; // Pole1 real part (normalised)
    parameter real ip1 = -0.7201; // Pole1 imaginary part (normalised)
    .....
    parameter real f0 = 1000; // Cut-off frequency

    inout      vdd, vss;
    output     voutp, voutm;
    input      vinp, vinm;
    electrical vdd, vss, vinp, vinm, voutp, voutm;
    electrical _vmid;
    real       voutVar;

    analog begin
        V(_vmid, vss) <+ 0.5 * V(vdd, vss);

        voutVar = laplace_np ( V(vinp, vinm), { 1 },
            {rp1 * `M_TWO_PI*f0, ip1 * `M_TWO_PI*f0, rp2 * `M_TWO_PI*f0,
              ip2 * `M_TWO_PI*f0, rp3 * `M_TWO_PI*f0, ip3 * `M_TWO_PI*f0,
              rp4 * `M_TWO_PI*f0, ip4 * `M_TWO_PI*f0, rp5 * `M_TWO_PI*f0,
              ip5 * `M_TWO_PI*f0} );

        V(voutp, _vmid) <+ 0.5 * voutVar;
        V(voutm, _vmid) <+ -0.5 * voutVar;

    end
endmodule
```

## Appendix 2: System-C Example

```
class ms_tlm_rxgainctrl : public sc_module, public ms_tlm_generic_data_blocking_if
{
public:
    sc_port <ms_tlm_generic_data_non_blocking_if> out_port;

    // constructor
    ms_tlm_rxgainctrl(sc_module_name name_);
    void write(int i_data, int q_data, source_id_codes source_id);

    sc_fifo <int> i_fifo;
    sc_fifo <int> q_fifo;

    .....
};

class ms_tlm_rxgainctrl_type_a : public ms_tlm_rxgainctrl
{
public:
    void do_rxgainctrl_process_thread ();    // the infinite loop to process the buffer
    SC_HAS_PROCESS (ms_tlm_rxgainctrl_type_a);

    ms_tlm_rxgainctrl_type_a(sc_module_name name_, ...);
    ~ms_tlm_rxgainctrl_type_a();

    .....
};
```

## Appendix 2: System-C Example

```
ms_tlm_rxgainctrl_type_a::ms_tlm_rxgainctrl_type_a(sc_module_name name, ...):
    ms_tlm_rxgainctrl (name_)
{
    .....
    SC_THREAD (do_rxgainctrl_process_thread);
}

void ms_tlm_rxgainctrl_type_a::do_rxgainctrl_process_thread()
{
    out_port -> nb_write(0,MS_TLM_RXGAINCTRL);

    while(true) {
        i = i_fifo.read();
        q = q_fifo.read();

        // Operate
        .....

        out_port -> nb_write(gainctrl, MS_TLM_RXGAINCTRL);
    }
}

void ms_tlm_rxgainctrl::write (int i_data, int q_data, source_id_codes source_id)
{
    i_fifo.write (i_data);
    q_fifo.write (q_data);
}
```