GHDL, What's new ?

Tristan Gingold

FOSDEM'16

Agenda

- What is GHDL ?
- Improvements since FOSDEM’15
- Multi language simulation - Proof of concept

What is GHDL ?

What is GHDL ?

If you have missed last year presentation:

- Free, open-source VHDL simulator
- Generate object code (with gcc, LLVM or it’s generator)
- Support VHDL 1987 till 2008
- github.com/tgingold/ghdl

What is GHDL ?

- Command line tool (no IDE)
- Generate VCD, GHW or FST waveforms
  - GTKwave compatible
- VHPI and VPI support
- Partial support of PSL

Typical use cases

- FOSS activist: use a free tool
- Nightly testing
- Coverage (with gcov)

What’s new ?

What’s new ?

- Switch to git
- Move to github
- Docs on ghdl.readthedocs.org
- Continuous integration on travis-ci.org

GHDL 0.33 released

- For Linux x86, x86-64
  - generic, debian and Ubuntu
- For Mac OS X
- For Windows
- First version with good VHDL 2008 support

New features

- More VHDL 2008 features
- Many bugs fixed
- Support of VUnit
  - OSS unit testing framework
- Support of OSVVM
  - Open Source VHDL Verification Methodology

Post release features

- Mcode for x86-64:
    - Mcode is builtin code generator
    - Code generation is much faster than llvm or gcc
    - Scale very well
    - Easy to build (no dependencies)
    - Was i386 + fpu only
    - Now i386 or x86-64 + SSE2

Post release features

- Optimizations
  - Better handling of signals
  - Explicit process state

Future work

- More optimisations
- Multi language simulation
- Synthesis ?

Multi language simulation

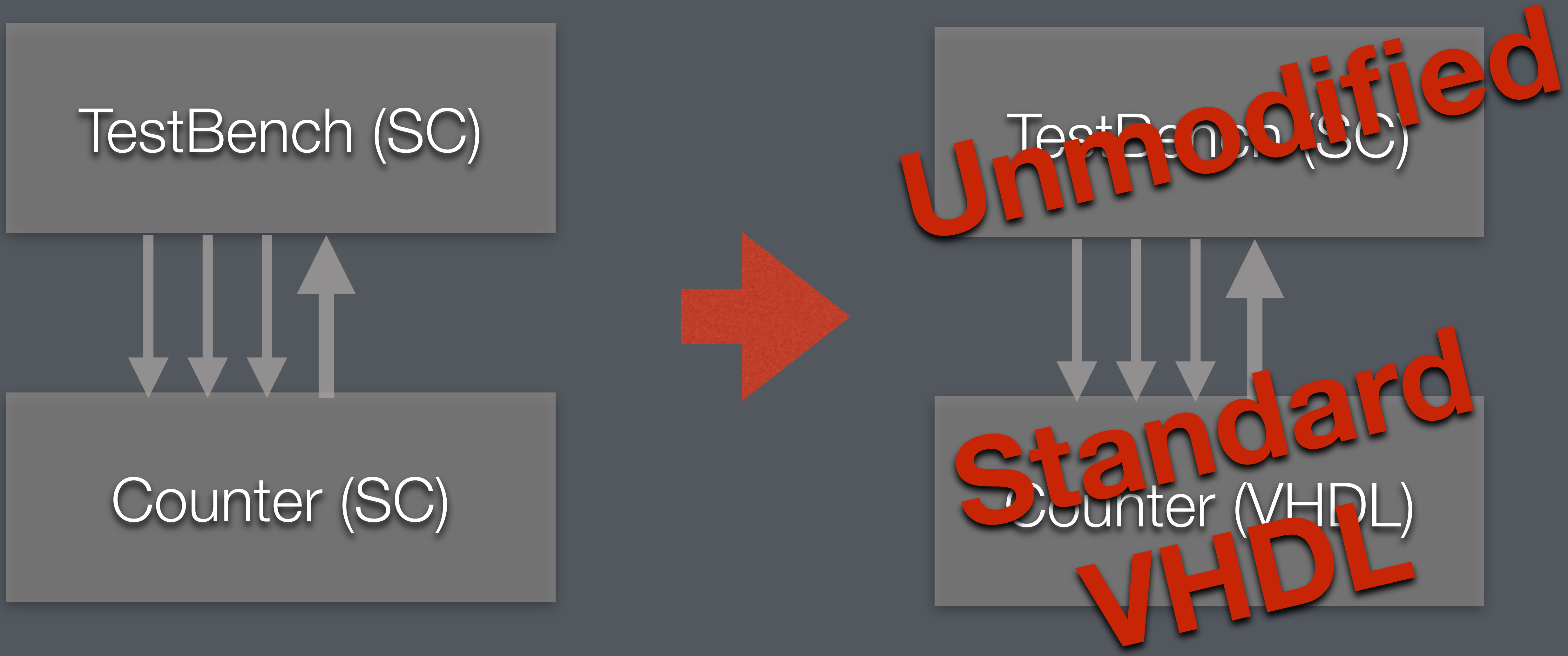
VHDL + SystemC

- Purpose: simulate both VHDL and SystemC
  - Using GHDL

Why SystemC ?

- And not Verilog ?
- SystemC is simpler
- Simulation cycle is like VHDL
- Use the same compilation model as GHDL
  - Object files
  - Final link

Proof of Concept



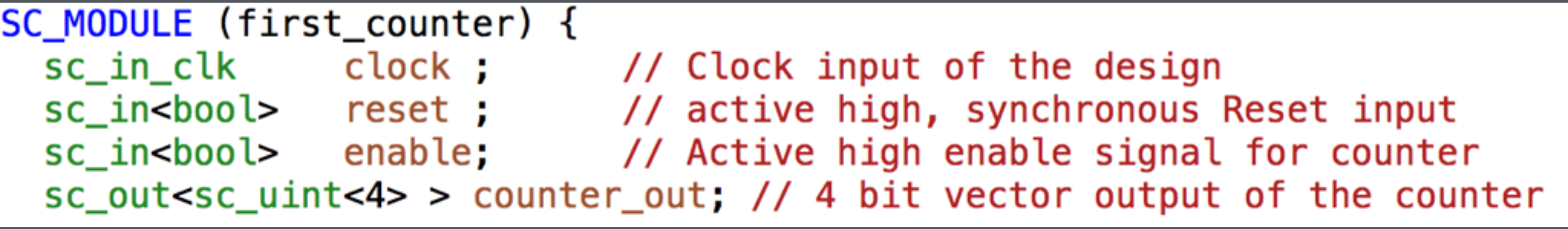
Proof of concept: constraints

- Main is SystemC
- No change in test bench or VHDL code
- No change in SystemC kernel

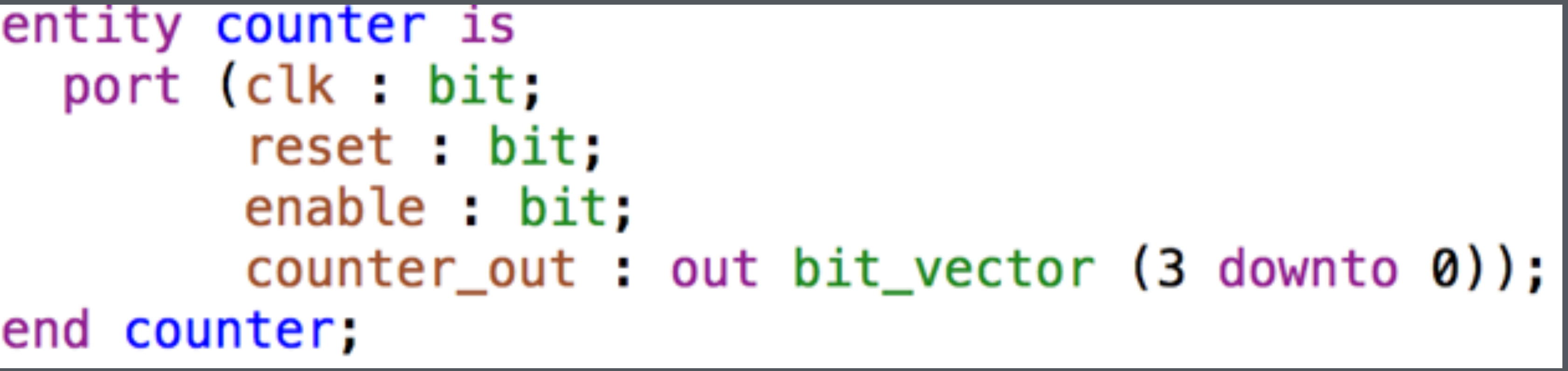
Proof of concept: simplifications

- Only one VHDL module
- Multi-language semantic weakly defined
- Can change GHDL code

Proof of Concept

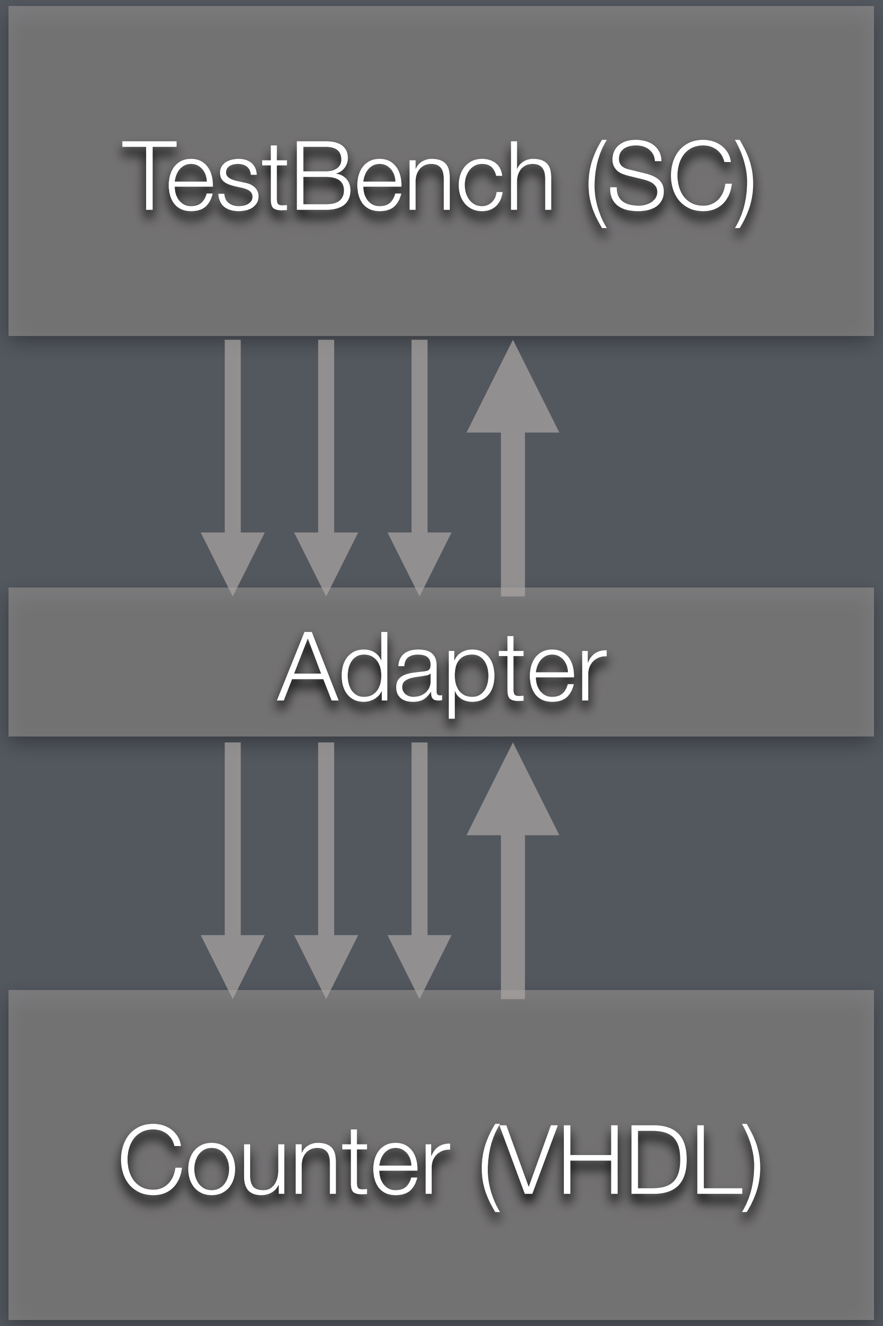


```
SC_MODULE (first_counter) {
  sc_in_clk     clock ;      // Clock input of the design
  sc_in<bool>   reset ;      // active high, synchronous Reset input
  sc_in<bool>   enable;      // Active high enable signal for counter
  sc_out<sc_uint<4> > counter_out; // 4 bit vector output of the counter
```



```
entity counter is
  port (clk : bit;
        reset : bit;
        enable : bit;
        counter_out : out bit_vector (3 downto 0));
end counter;
```

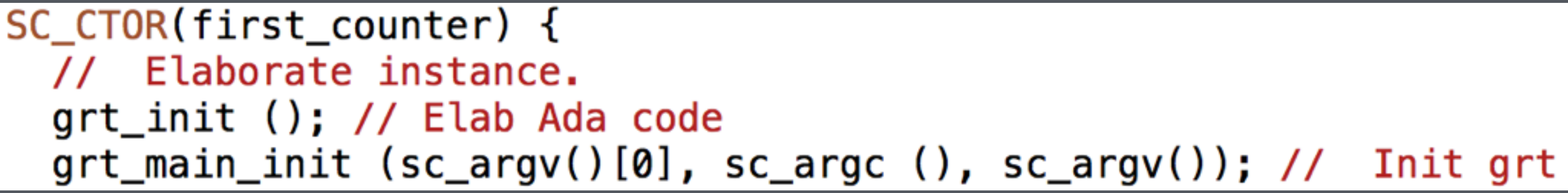
Proof of Concept: Adapter



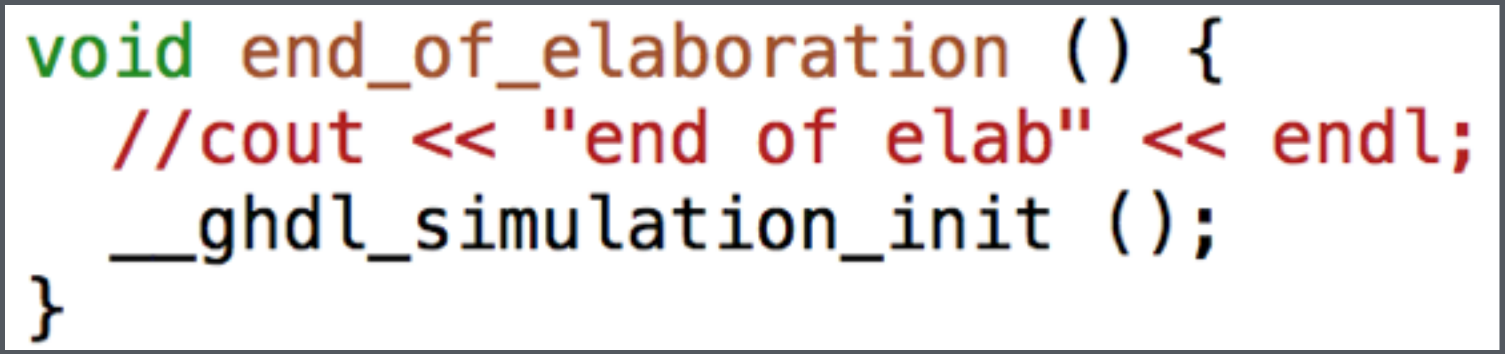
Adapter

- Initialize GHDL simulator
- Instantiate the VHDL blocks
- Convert values between SystemC and VHDL

GHDL initialization

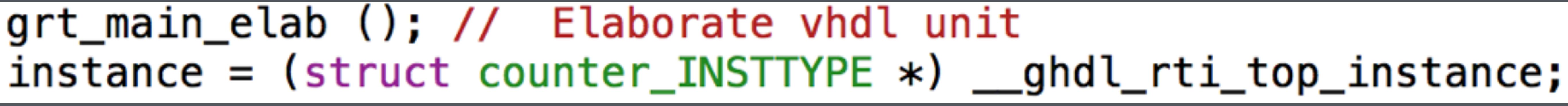


```
SC_CTOR(first_counter) {
  //  Elaborate instance.
  grt_init (); // Elab Ada code
  grt_main_init (sc_argv()[0], sc_argc (), sc_argv()); //  Init grt
```

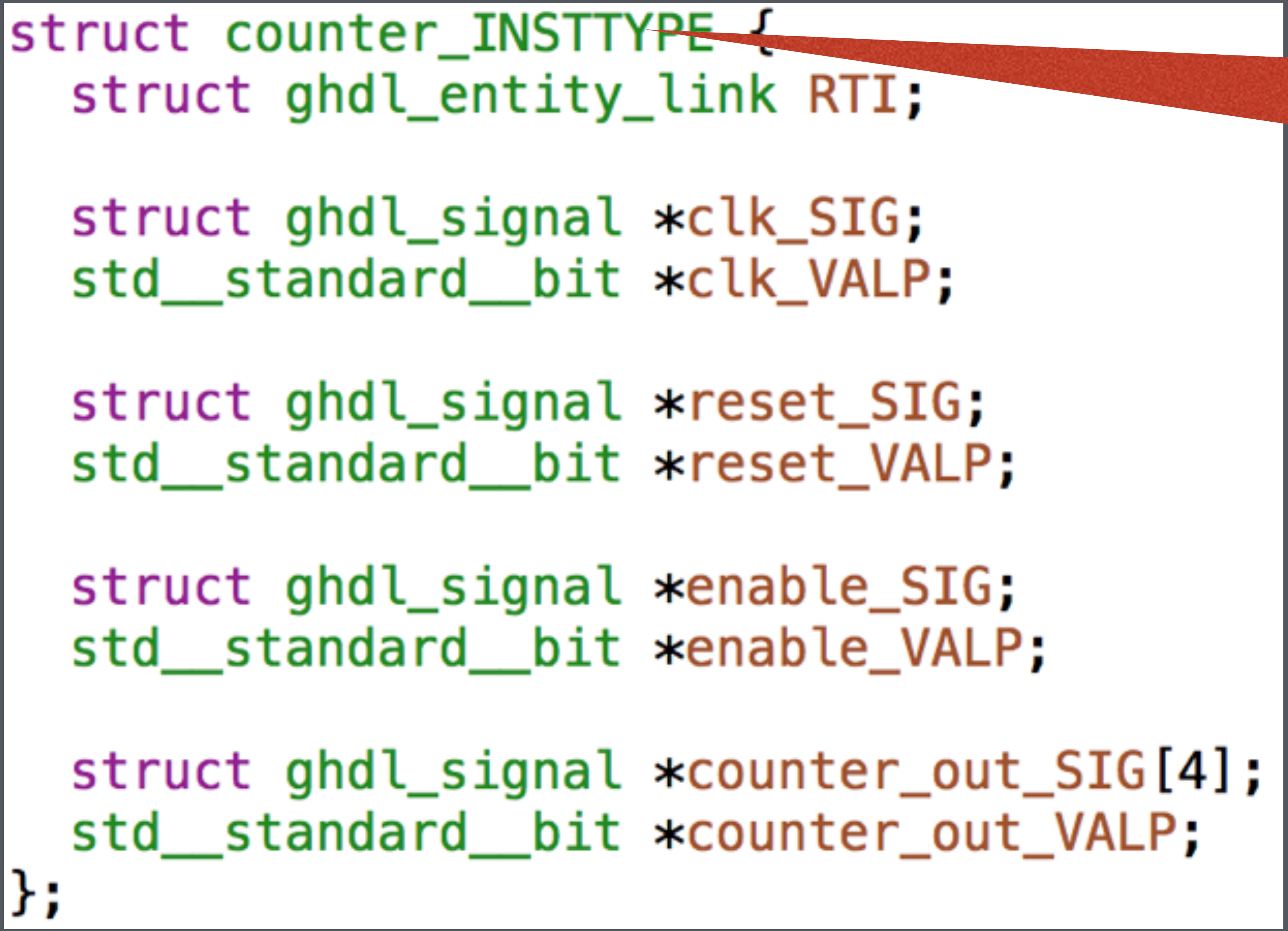


```
void end_of_elaboration () {
  //cout << "end of elab" << endl;
  __ghdl_simulation_init ();
}
```

VHDL instantiation



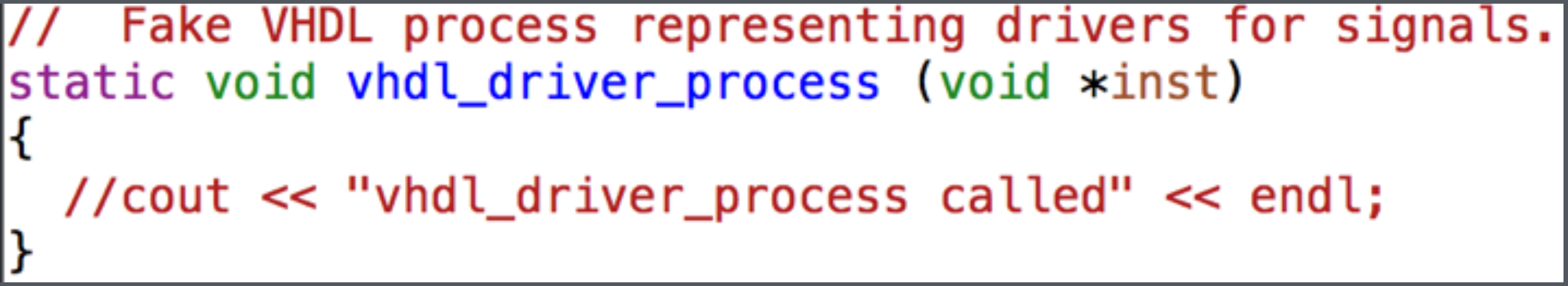
```
grt_main_elab (); //  Elaborate vhdl unit
instance = (struct counter_INSTTYPE *) __ghdl_rti_top_instance;
```



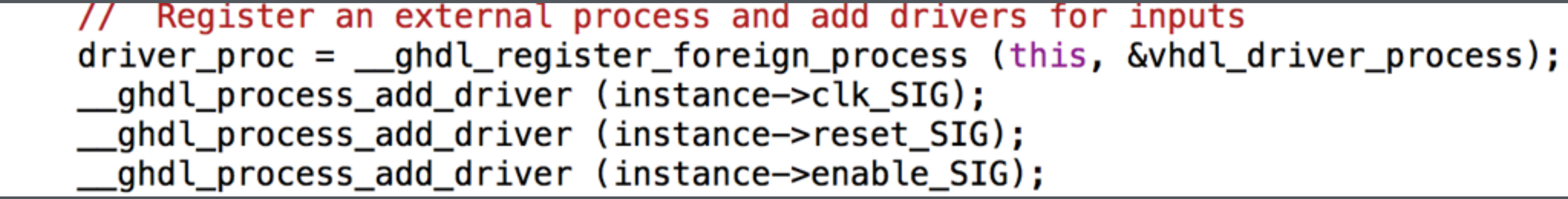
```
struct counter_INSTTYPE {
  struct ghdl_entity_link RTI;

  struct ghdl_signal *clk_SIG;
  std__standard__bit *clk_VALP;

  struct ghdl_signal *reset_SIG;
  std__standard__bit *reset_VALP;

  struct ghdl_signal *enable_SIG;
  std__standard__bit *enable_VALP;

  struct ghdl_signal *counter_out_SIG[4];
  std__standard__bit *counter_out_VALP;
};
```

C representation of the VHDL entity, as compiled by GHDL

Pseudo process for Inputs

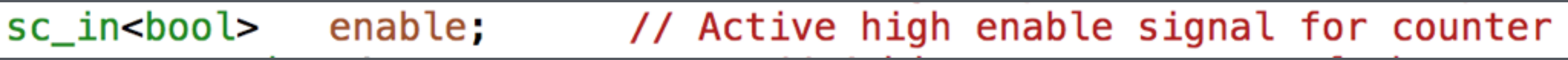


```
//  Fake VHDL process representing drivers for signals.
static void vhdl_driver_process (void *inst)
{
  //cout << "vhdl_driver_process called" << endl;
}
```

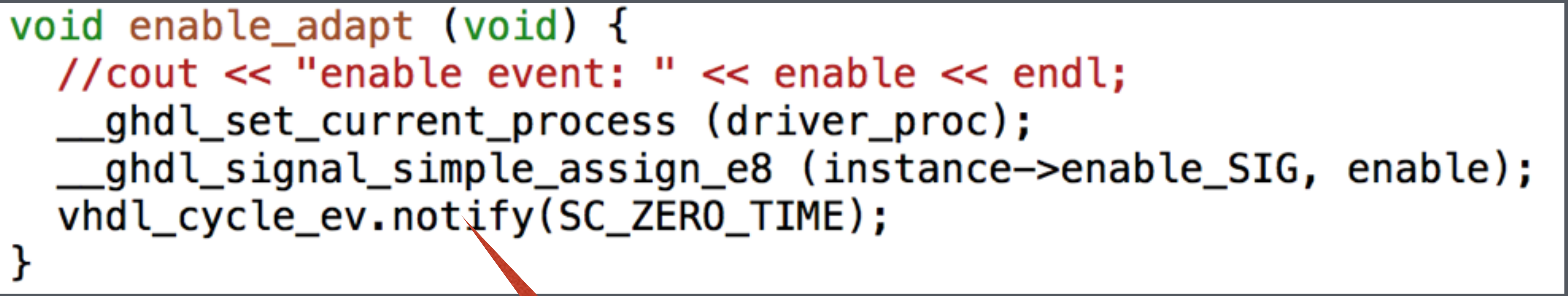


```
    //  Register an external process and add drivers for inputs
    driver_proc = __ghdl_register_foreign_process (this, &vhdl_driver_process);
    __ghdl_process_add_driver (instance->clk_SIG);
    __ghdl_process_add_driver (instance->reset_SIG);
    __ghdl_process_add_driver (instance->enable_SIG);
```

Value Conversions: input

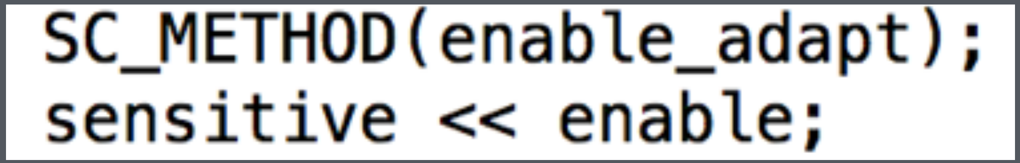


```
sc_in<bool>    enable;        // Active high enable signal for counter
```



```
void enable_adapt (void) {
  //cout << "enable event: " << enable << endl;
  __ghdl_set_current_process (driver_proc);
  __ghdl_signal_simple_assign_e8 (instance->enable_SIG, enable);
  vhdl_cycle_ev.notify(SC_ZERO_TIME);
}
```

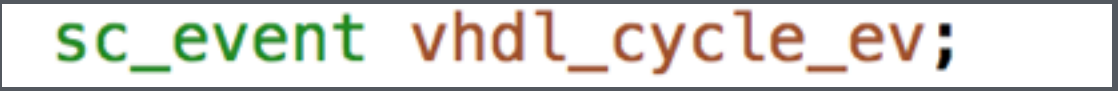
Ask SystemC to run GHDL kernel



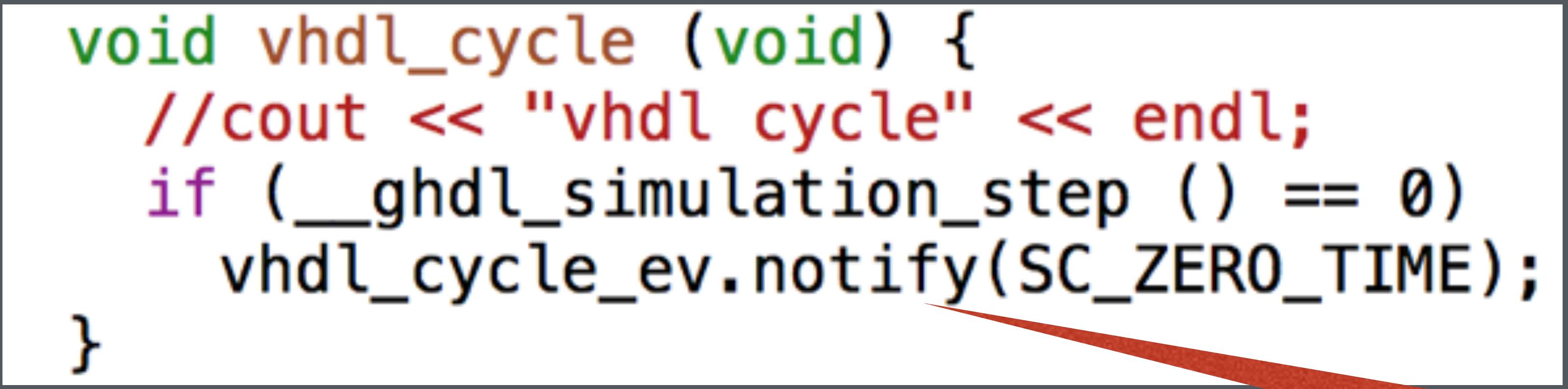
```
SC_METHOD(enable_adapt);
sensitive << enable;
```

So if an input change, it is propagated to VHDL

Kernel interleaving

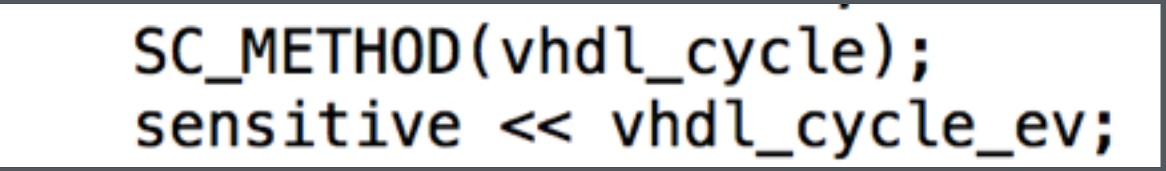


```
sc_event vhdl_cycle_ev;
```



```
void vhdl_cycle (void) {
  //cout << "vhdl cycle" << endl;
  if (__ghdl_simulation_step () == 0)
    vhdl_cycle_ev.notify(SC_ZERO_TIME);
}
```

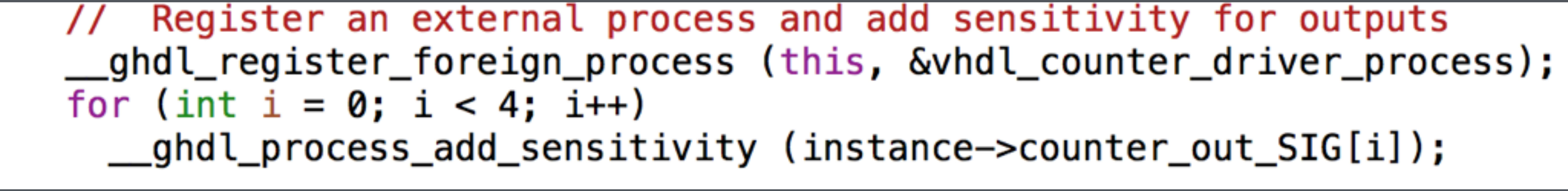
Need a new delta cycle



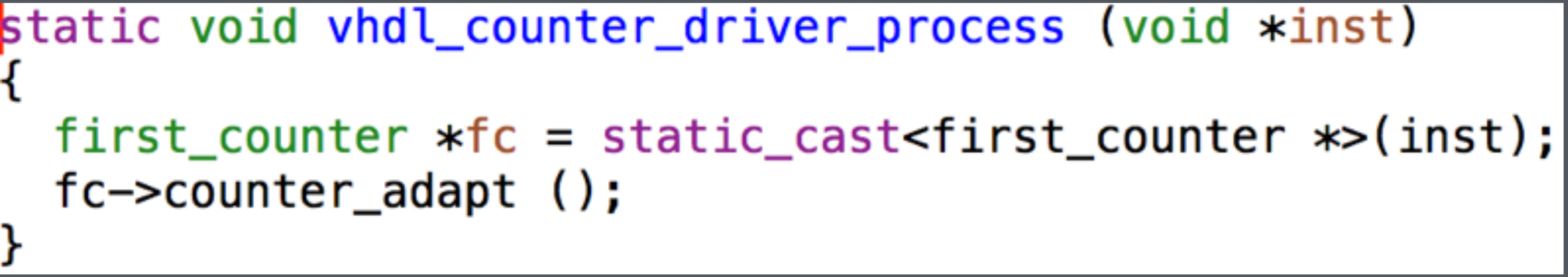
```
  SC_METHOD(vhdl_cycle);
  sensitive << vhdl_cycle_ev;
```

So VHDL kernel is run in case of event

Value Conversions: Output



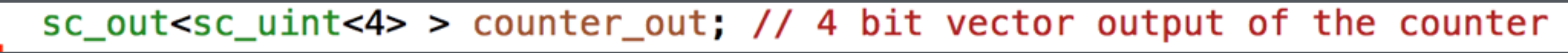
```
   //  Register an external process and add sensitivity for outputs
   __ghdl_register_foreign_process (this, &vhdl_counter_driver_process);
   for (int i = 0; i < 4; i++)
     __ghdl_process_add_sensitivity (instance->counter_out_SIG[i]);
```



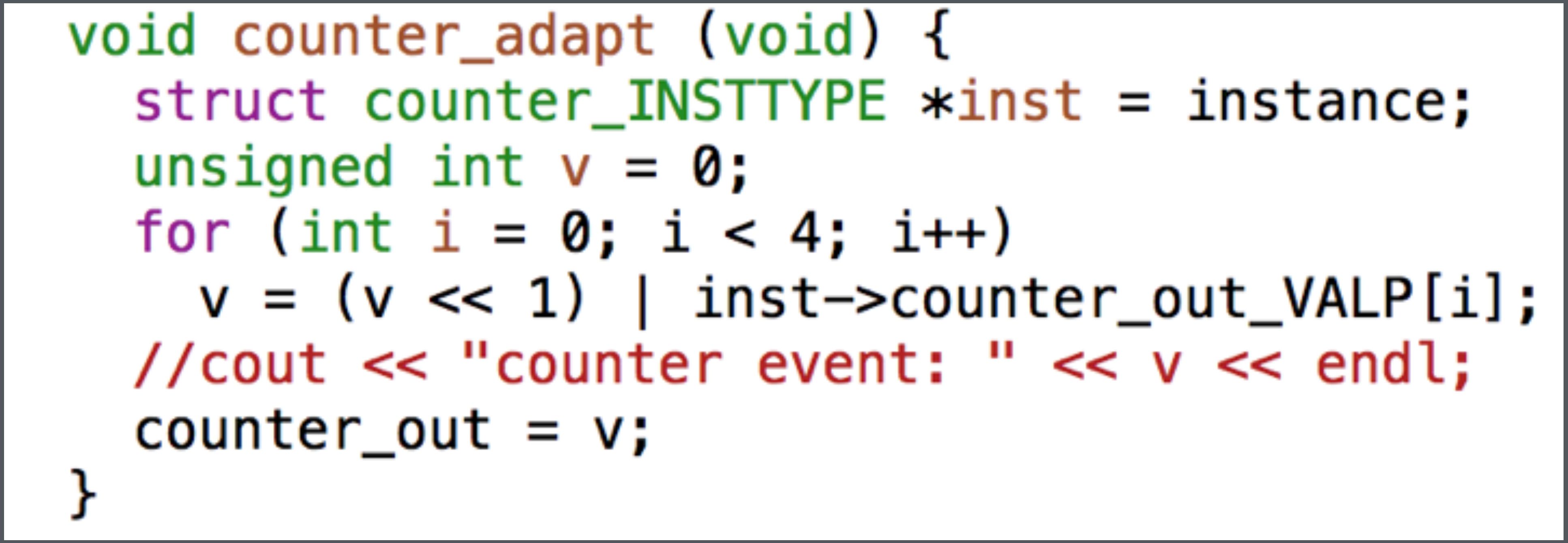
```
static void vhdl_counter_driver_process (void *inst)
{
  first_counter *fc = static_cast<first_counter *>(inst);
  fc->counter_adapt ();
}
```

So if a VHDL signal change, a method is called

Value Conversions: Output



```
sc_out<sc_uint<4> > counter_out; // 4 bit vector output of the counter
```



```
void counter_adapt (void) {
  struct counter_INSTTYPE *inst = instance;
  unsigned int v = 0;
  for (int i = 0; i < 4; i++)
    v = (v << 1) | inst->counter_out_VALP[i];
  //cout << "counter event: " << v << endl;
  counter_out = v;
}
```

Convert from a VHDL signal to a SystemC signal

Proof of concept: Conclusions

- Just of Proof of Concept but:
- It works!
- required only changes in GHDL runtime (kernel)
  - need to split the simulation loop

Proof of concept: Future

- Generate the adapter automatically
  - Mapping is not obvious
- Avoid delta-cycles between SC and VHDL
  - Need to modify SystemC kernel
- Multiple VHDL instances
- Instantiation of SC by VHDL blocks