

```

package mylib

import spinal.core._

class Timer(width: Int) extends Component {

  /**
   * This is the component definition that corresponds to
   * the VHDL entity of the component
   */
  val io = new Bundle {
    val tick = in Bool
    val clear = in Bool
    val limit = in UInt(width bits)
    val full = out Bool
  }

  val counter = Reg(UInt(width bits))
  when(io.tick && !io.full) {
    counter := counter + 1
  }

  when (io.clear) {
    counter := 0
  }

  io.full := counter === io.limit
}

object TimerVhdl {
  // Let's go
  def main(args: Array[String]) {
    SpinalVhdl(new Timer(8))
  }
}

object TimerVerilog {
  // Let's go
  def main(args: Array[String]) {
    SpinalVerilog(new Timer(8))
  }
}

package mylib

import spinal.core._

class Timer(width: Int) extends Component {

  /**
   * This is the component definition that corresponds to
   * the VHDL entity of the component

```

```

    */
    val io = new Bundle {
        val tick = in Bool
        val clear = in Bool
        val limit = in UInt(width bits)
        val full = out Bool
    }

    val counter = Reg(UInt(width bits))
    when(io.tick && !io.full) {
        counter := counter + 1
    }

    when (io.clear) {
        counter := 0
    }

    io.full := counter === io.limit
}

object TimerVhdl {
    // Let's go
    def main(args: Array[String]) {
        SpinalVhdl(new Timer(8))
    }
}

object TimerVerilog {
    // Let's go
    def main(args: Array[String]) {
        SpinalVerilog(new Timer(8))
    }
}

////////////////////////////////////
//
// Filename:   helloworld_tb.cpp
//
// Project:    Verilog Tutorial Example file
//
// Purpose:    To demonstrate a Verilog main() program that calls a local
//              serial port co-simulator.
//
// Creator:    Dan Gisselquist, Ph.D.
//              Gisselquist Technology, LLC
//
////////////////////////////////////
//
// Written and distributed by Gisselquist Technology, LLC
//
// This program is hereby granted to the public domain.
//

```

```
// This program is distributed in the hope that it will be useful, but WITHOUT
// ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
// FITNESS FOR A PARTICULAR PURPOSE.
```

```
//
```

```
////////////////////////////////////
```

```
//
```

```
//
```

```
#include <verilatedos.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <signal.h>
#include "verilated.h"
#include "VTimer.h"
#include "testb.h"
```

```
int    main(int argc, char **argv) {
    Verilated::commandArgs(argc, argv);
    TESTB<VTimer>    *tb
        = new TESTB<VTimer>;
```

```
    Verilated::traceEverOn(true); // Verilator must compute traced signals
    VL_PRINTF("Enabling waves...\n");
```

```
    tb->opentrace("Timer.vcd");
    //tb->m_core->reset = 0;
    for(unsigned clocks=0;clocks < 10000;clocks++) {

        if (clocks==10) tb->m_core->io_limit = 128;
        if (clocks==12)    tb->m_core->io_tick = 1;
        //if (clocks==30) tb->m_core->io_switchs = 65;
        //if (clocks==31) tb->m_core->io_write_valid;

        //if (clocks==100) tb->m_core->io_switchs = 68;
        if (clocks==300) tb->m_core->io_clear=1;
        if (clocks==301) tb->m_core->io_clear=0;
```

```
        tb->tick();
```

```
    }
```

```
    printf("\n\nSimulation complete\n");
```

```
}
```

```

////////////////////////////////////
//
// Filename:   testb.h
//
// Project:    Verilog Tutorial Example file
//
// Purpose:    A wrapper providing a common interface to a clocked FPGA core
//              being exercised by Verilator.
//
// Creator:    Dan Gisselquist, Ph.D.
//              Gisselquist Technology, LLC
//
////////////////////////////////////
//
// Written and distributed by Gisselquist Technology, LLC
//
// This program is hereby granted to the public domain.
//
// This program is distributed in the hope that it will be useful, but WITHOUT
// ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
// FITNESS FOR A PARTICULAR PURPOSE.
//
////////////////////////////////////
//
//
#ifndef TESTB_H
#define TESTB_H

#include <stdio.h>
#include <stdint.h>
#include <verilated_vcd_c.h>

#define TBASSERT(TB,A) do { if (!(A)) { (TB).closetrace(); } assert(A); } while(0);

template <class VA> class TESTB {
public:
    VA            *m_core;
    VerilatedVcdC* m_trace;
    uint64_t      m_tickcount;

    TESTB(void) : m_trace(NULL), m_tickcount(0) {
        m_core = new VA;
        Verilated::traceEverOn(true);
        m_core->clk = 0;
        eval(); // Get our initial values set properly.
    }
    virtual ~TESTB(void) {
        closetrace();
        delete m_core;
        m_core = NULL;
    }
}

```

```

virtual void  opentrace(const char *vcdname) {
    if (!m_trace) {
        m_trace = new VerilatedVcdC;
        m_core->trace(m_trace, 99);
        m_trace->open(vcdname);
    }
}

virtual void  closetrace(void) {
    if (m_trace) {
        m_trace->close();
        delete m_trace;
        m_trace = NULL;
    }
}

virtual void  eval(void) {
    m_core->eval();
}

virtual void  tick(void) {
    m_tickcount++;

    // Make sure we have our evaluations straight before the top
    // of the clock. This is necessary since some of the
    // connection modules may have made changes, for which some
    // logic depends. This forces that logic to be recalculated
    // before the top of the clock.
    eval();
    if (m_trace) m_trace->dump((vluint64_t)(10*m_tickcount-2));
    m_core->clk = 1;
    eval();
    if (m_trace) m_trace->dump((vluint64_t)(10*m_tickcount));
    m_core->clk = 0;
    eval();
    if (m_trace) {
        m_trace->dump((vluint64_t)(10*m_tickcount+5));
        m_trace->flush();
    }
}

unsigned long tickcount(void) {
    return m_tickcount;
}

};

#endif

```

verilator -Wall --trace -cc Timer.v --exe --build sim_main.cpp

cd obj_dir

```
./Vtimer
Enabling waves...
```

Enabling waves...

Simulation complete

gtdwave Timer.vcd

