Imperial College
London

Lecture 3

Verilator, Testbench and Vbuddy

Prof Peter YK Cheung
Imperial College London

URL: www.ee.ic.ac.uk/pcheung/teaching/EIE2-IAC/
E-mail: p.cheung@imperial.ac.uk

Learning outcomes

- Different types of **simulators**
- Verify a SystemVerilog (SV) module with **Verilator**
- Template for a **Verilator testbench**
- Using a **shell script** as shortcut
- Verify a SV module using **gtkWave** waveform viewer
- Verify a SV module using **Vbuddy**
- What is in **Lab 1**?

Slides in this lecture are partly derived and modified from:

“*Verilator: Fast, Free, But for me?”*, a talk by Wilson Snyder (created of Verilator) – http://www.veripool.org/papers

Verilator History

- Verilator was born in 1994
  - Verilog was the new Synthesis Language
  - C++ was the Test-bench Language
  - Paul Wasson synthesized Verilog into C++
- Wilson Snyder created Verilator since 2001
  - Open-source and free
  - Strong community with many contributors
  - Works on all platforms (PC, Linux, MacOS)
  - Fast, particularly with multithreading

VERILATOR

Verilator User Base

intel NXP MICROTUNE Imperial College London

ARM Quadrics UBIXUM EMBECOSM

Sun microsystems apm applied micro csr Carnegie Mellon UBC

Axis Semiconductor CoWare MINDSPEED AMD

BROADCOM. hyperstone picoChip NORTHWEST LOGIC

Imperial College London infineon FORTE DESIGN SYSTEMS

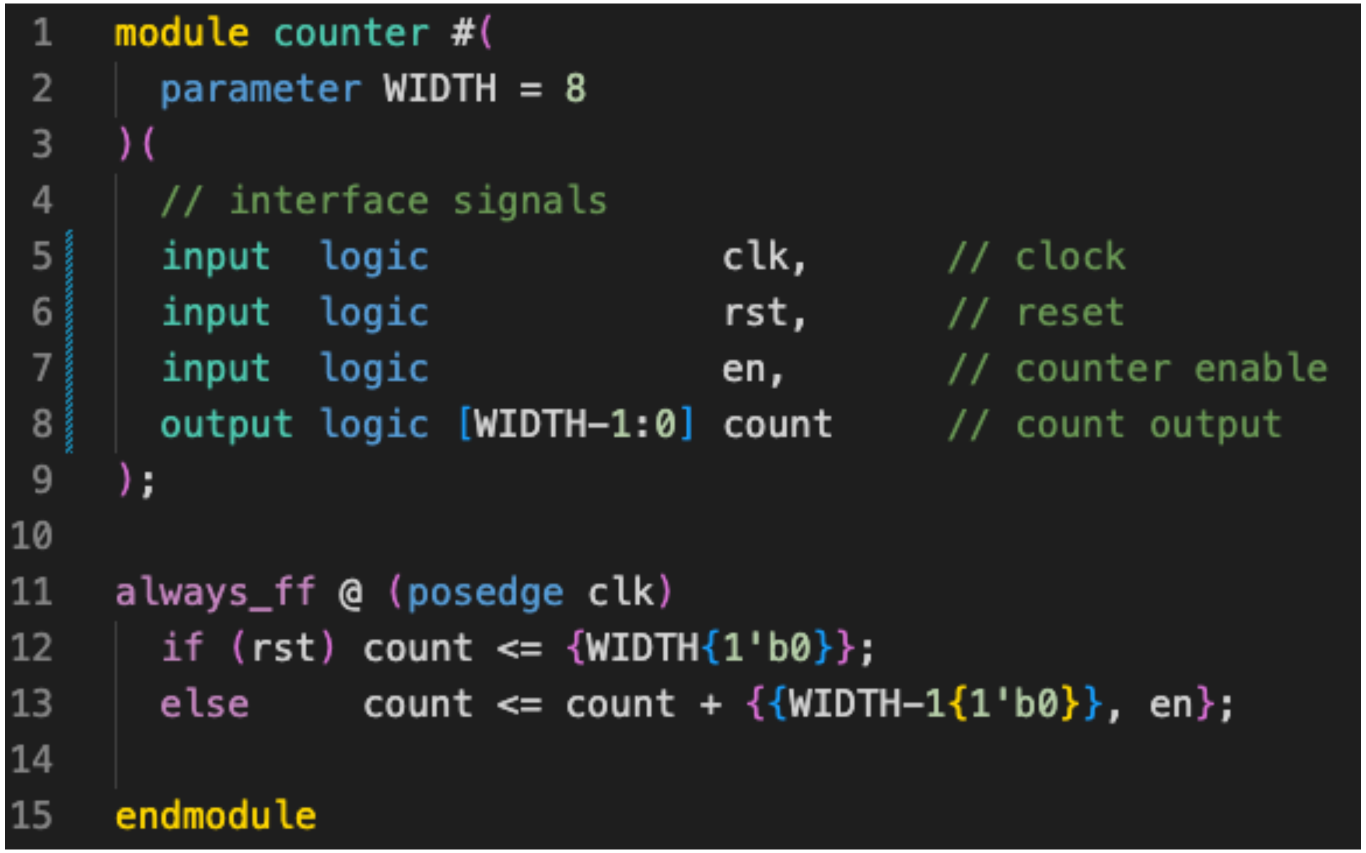
Raytheon GENERAL DYNAMICS Advanced Information Systems ESSENSIUM CVRx Medicine ReEnvisioned

All trademarks registered by respective owners.
Users based on correspondence; there is no official way to determine “users” since there’s no license!

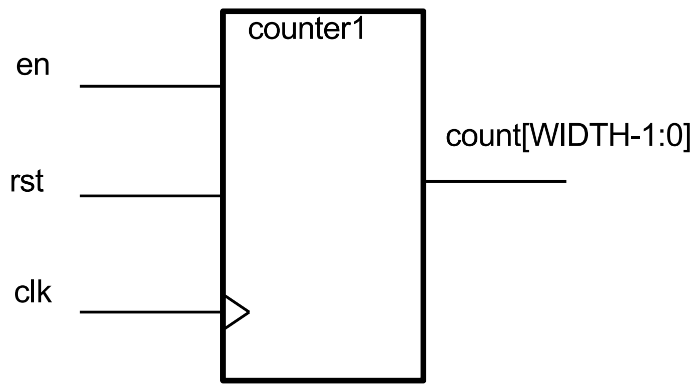
Three types of simulator

1. **Instruction level simulator**
   - Processor instruction-level function
   - No hardware representation, no concept of clock
   - Written in high level language such as C or C++
   - GNU RISC-V toolchain includes such a simulator
2. **Cycle accurate simulator**
   - Simulate HDL specification of hardware, e.g. SystemVerilog
   - All signal values correct cycle-by-cycle
   - No time delay information
   - Verilator is an example – only two values: ”0” or “1”
3. **Event-driven simulator**
   - Change signal produces event in an event queue
   - Simulate delay using timing model
   - Capture glitches
   - Slower and costly. Example: ModelSim

Example: Simple Counter



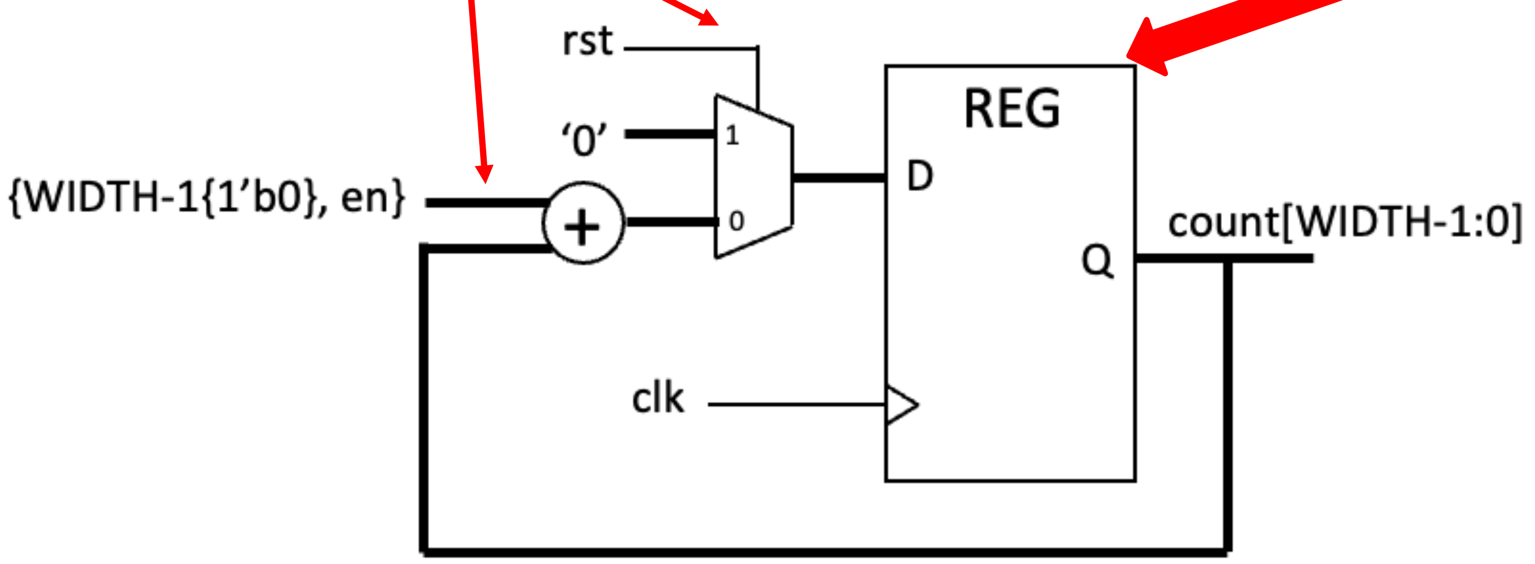
```
module counter #(
  parameter WIDTH = 8
)(
  // interface signals
  input  logic             clk,      // clock
  input  logic             rst,      // reset
  input  logic             en,       // counter enable
  output logic [WIDTH-1:0] count     // count output
);

always_ff @ (posedge clk)
  if (rst) count <= {WIDTH{1'b0}};
  else     count <= count + {{WIDTH-1{1'b0}}, en};

endmodule
```



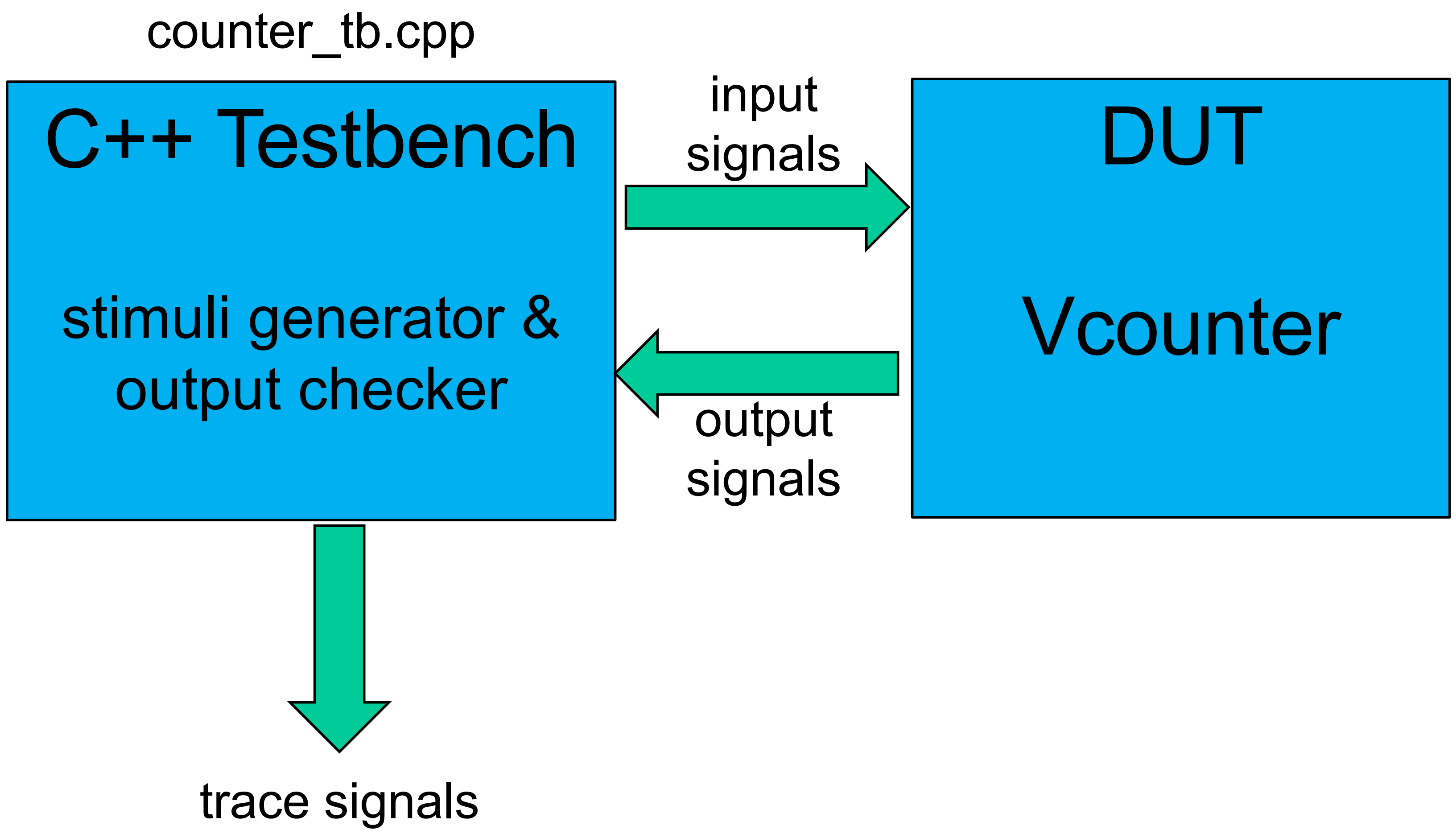
Mapping from SV to hardware

```
if (rst) count <= {WIDTH{1'b0}};
else     count <= count + {{WIDTH-1{1'b0}}, en};
```

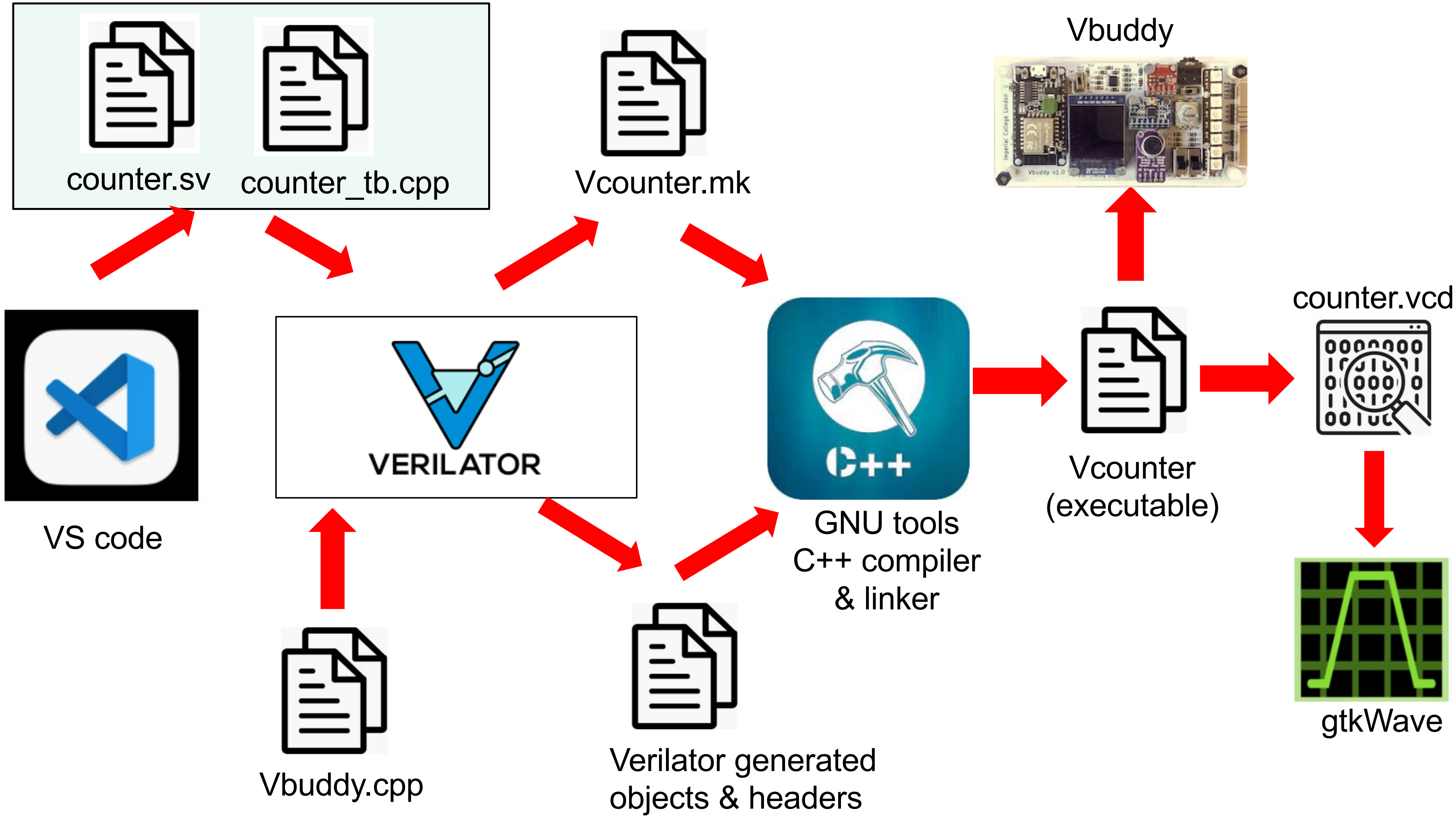
```
    output logic [WIDTH-1:0] count
  always_ff @ (posedge clk)
```



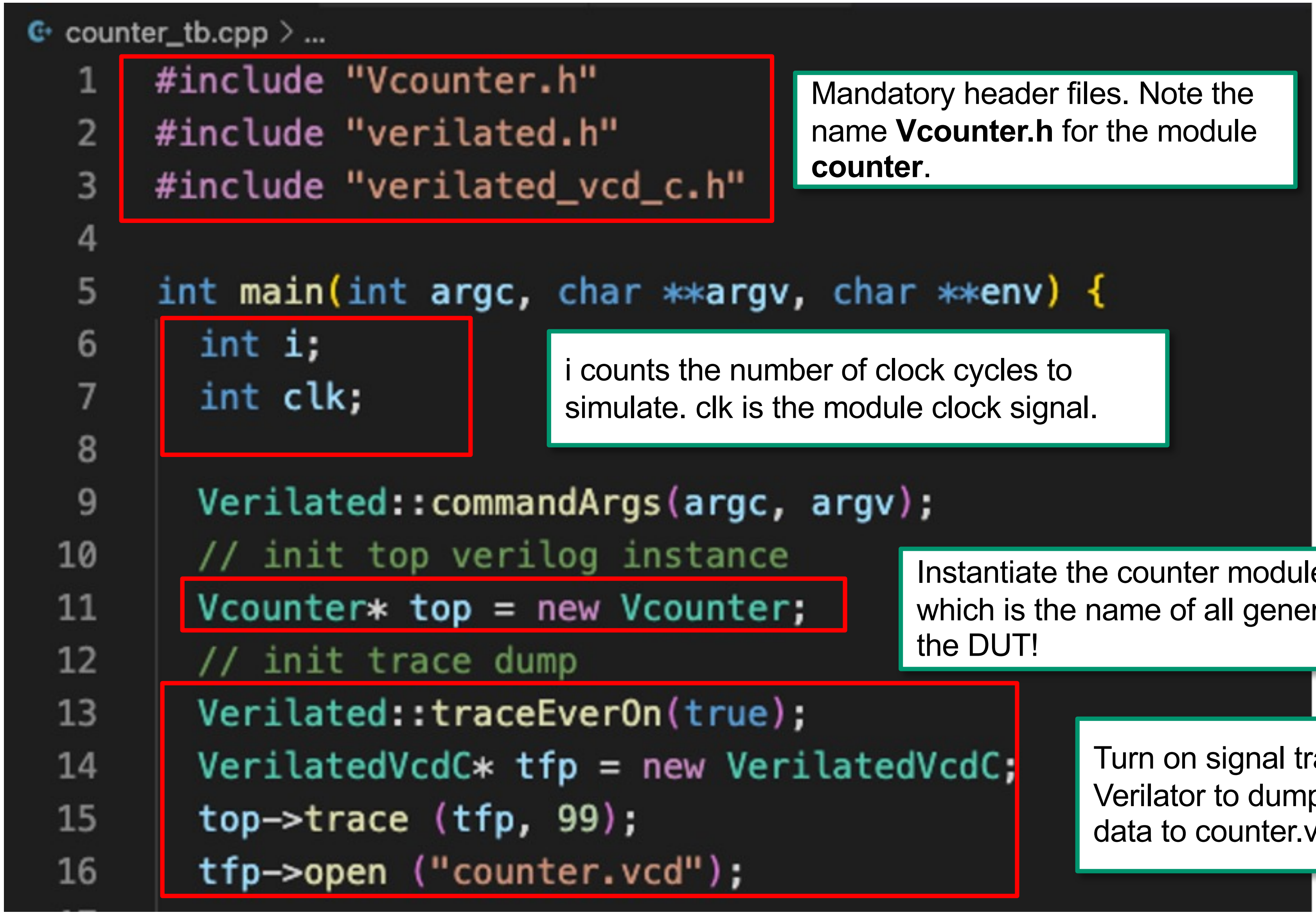
Testbench


counter_tb.cpp
C++ Testbench
stimuli generator & output checker
input signals
output signals
DUT
Vcounter
trace signals

How does Verilator work?



Format of the Testbench (1)



counter_tb.cpp > ...

```cpp
#include "Vcounter.h"
#include "verilated.h"
#include "verilated_vcd_c.h"

int main(int argc, char **argv, char **env) {
    int i;
    int clk;

    Verilated::commandArgs(argc, argv);
    // init top verilog instance
    Vcounter* top = new Vcounter;
    // init trace dump
    Verilated::traceEverOn(true);
    VerilatedVcdC* tfp = new VerilatedVcdC;
    top->trace (tfp, 99);
    tfp->open ("counter.vcd");
```

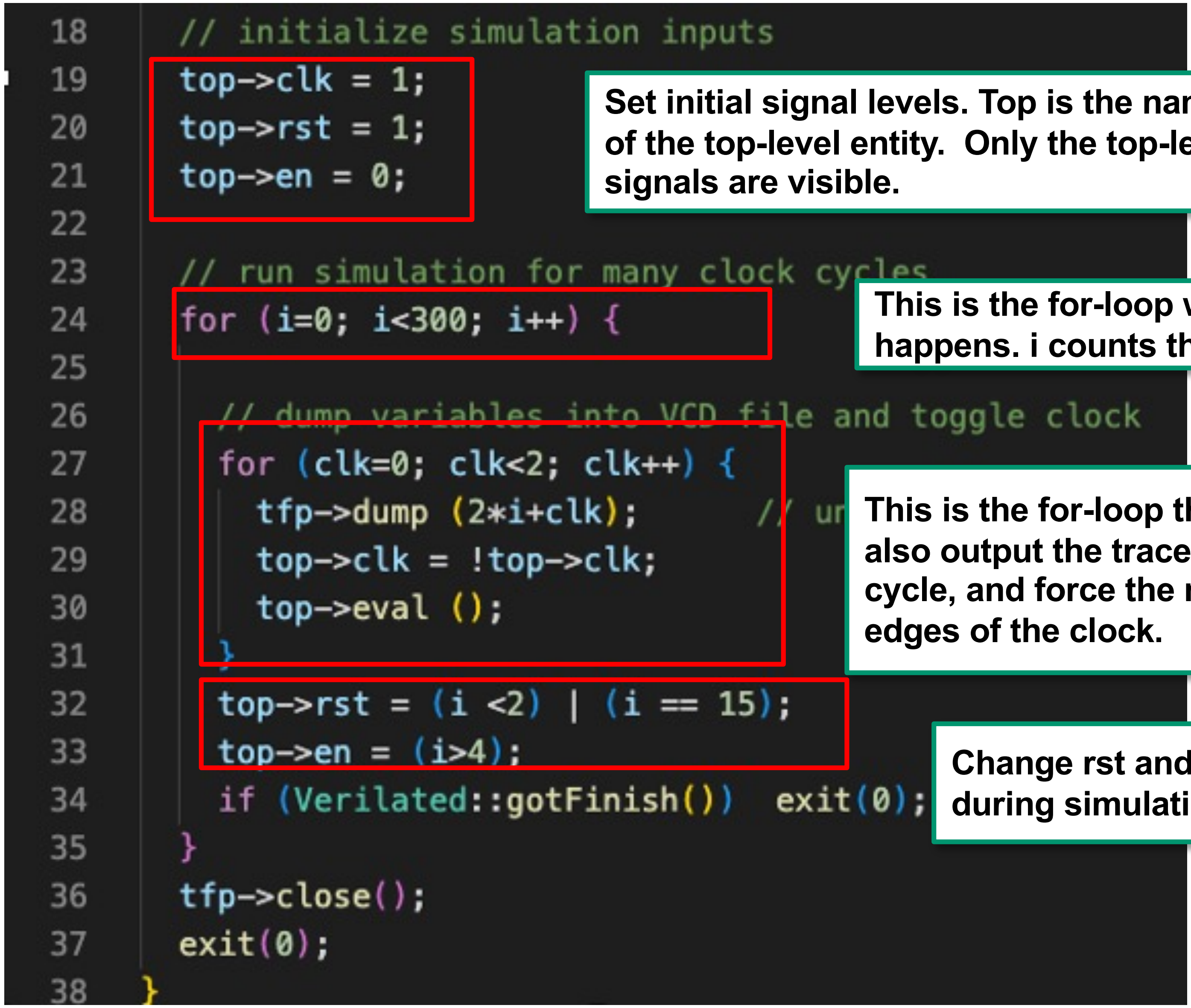
Mandatory header files. Note the name **Vcounter.h** for the module **counter**.

i counts the number of clock cycles to simulate. clk is the module clock signal.

Instantiate the counter module as **Vcounter**, which is the name of all generated files. This is the DUT!

Turn on signal tracing, and tell Verilator to dump the waveform data to counter.vcd

Format of the Testbench (2)



```
  // initialize simulation inputs
  top->clk = 1;
  top->rst = 1;
  top->en = 0;

  // run simulation for many clock cycles
  for (i=0; i<300; i++) {

    // dump variables into VCD file and toggle clock
    for (clk=0; clk<2; clk++) {
      tfp->dump (2*i+clk);      // ur
      top->clk = !top->clk;
      top->eval ();
    }
    top->rst = (i <2) | (i == 15);
    top->en = (i>4);
    if (Verilated::gotFinish())  exit(0);
  }
  tfp->close();
  exit(0);
}
```

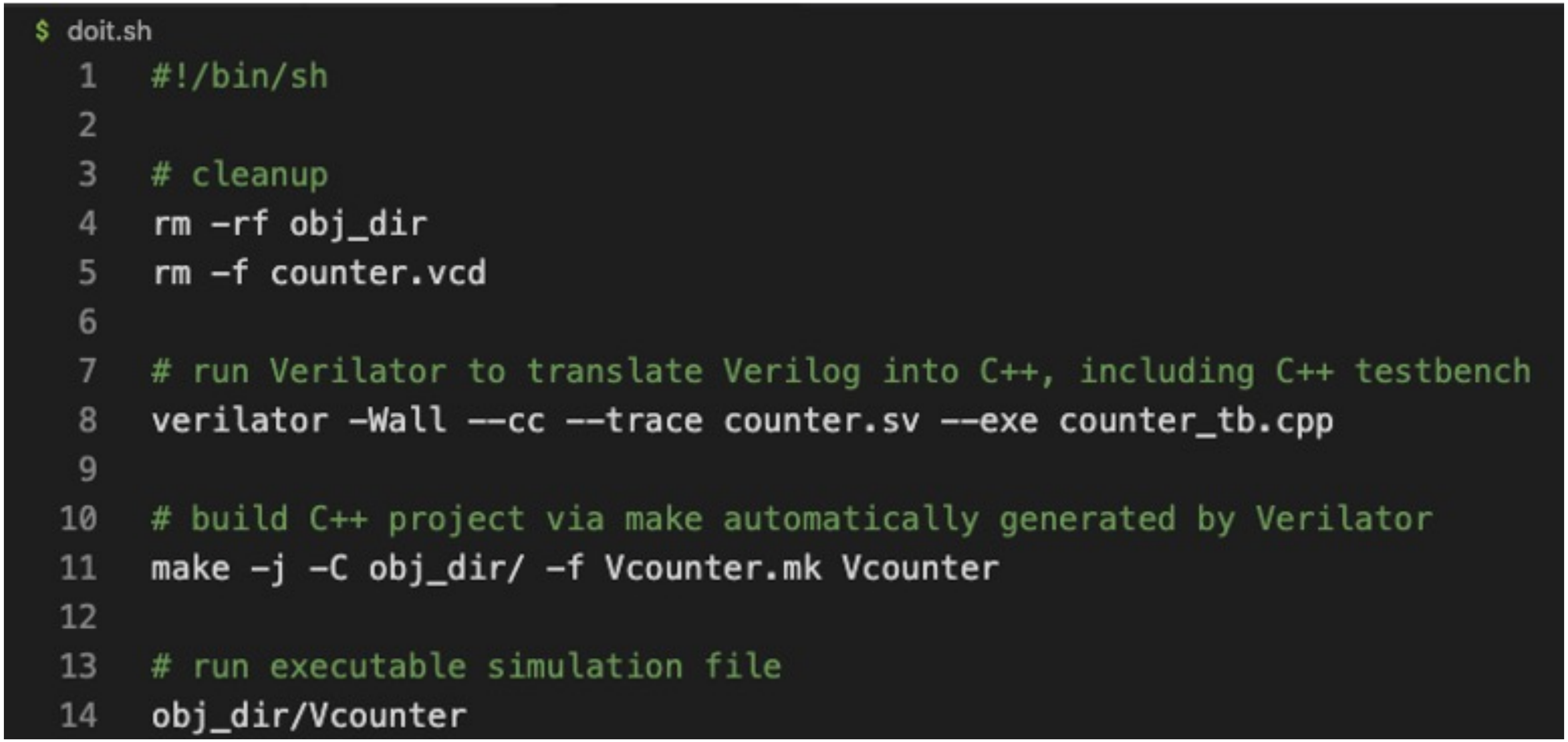
Set initial signal levels. Top is the name of the top-level entity. Only the top-level signals are visible.

This is the for-loop where simulation happens. i counts the clock cycles.

This is the for-loop that toggles the clock. It also output the trace for each half of the clock cycle, and force the model to evaluate on both edges of the clock.

Change rst and en signals during simulation.

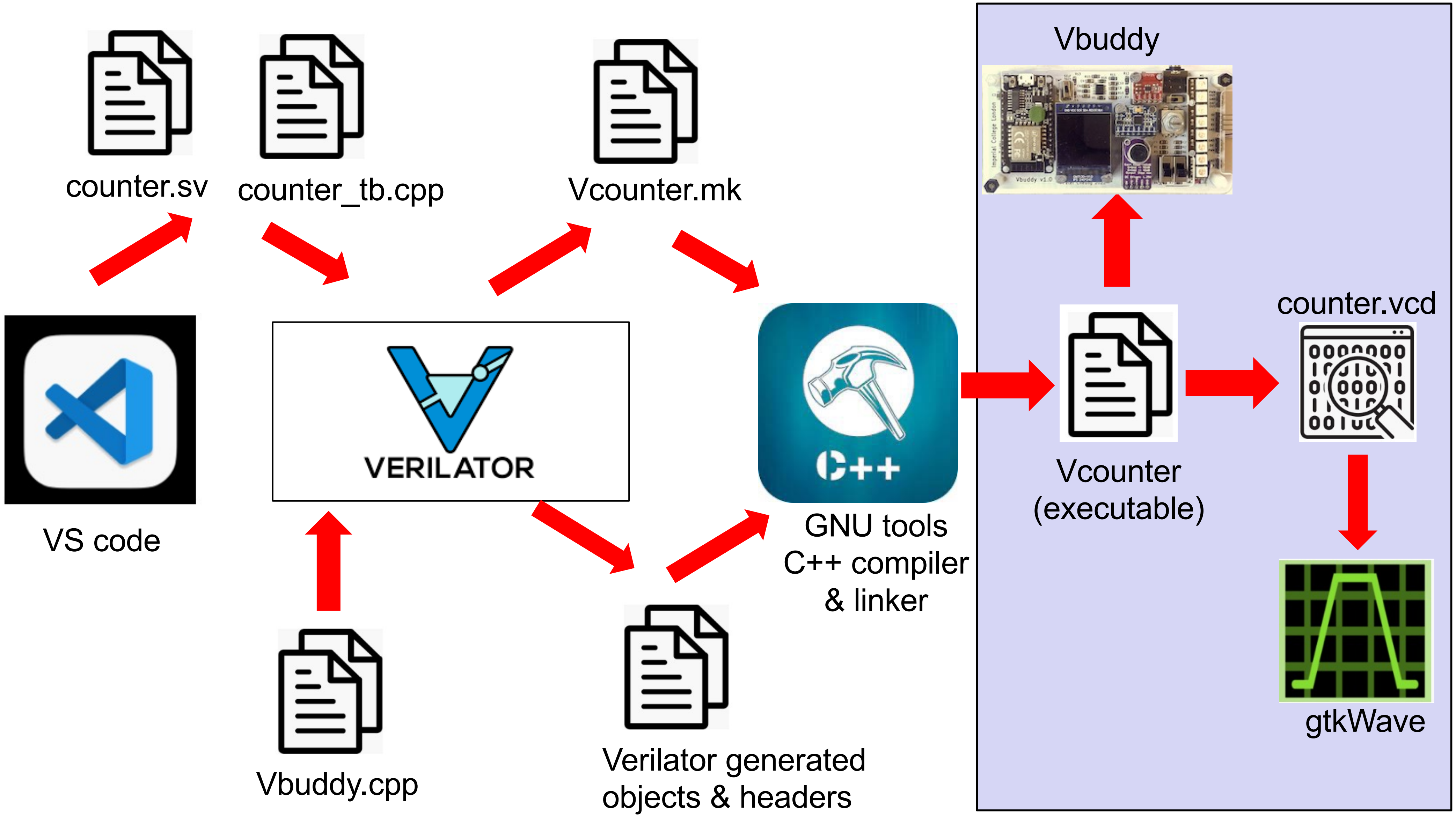
Making the final simulation model



$ doit.sh

```
#!/bin/sh

# cleanup
rm -rf obj_dir
rm -f counter.vcd

# run Verilator to translate Verilog into C++, including C++ testbench
verilator -Wall --cc --trace counter.sv --exe counter_tb.cpp

# build C++ project via make automatically generated by Verilator
make -j -C obj_dir/ -f Vcounter.mk Vcounter

# run executable simulation file
obj_dir/Vcounter
```

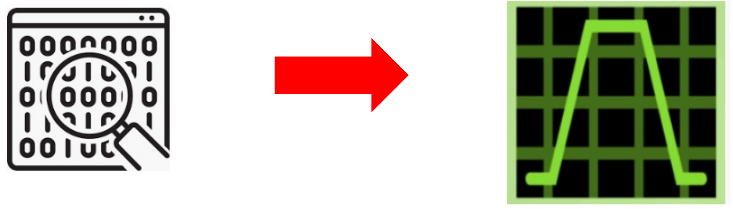
Vcounter is the executable model of counter

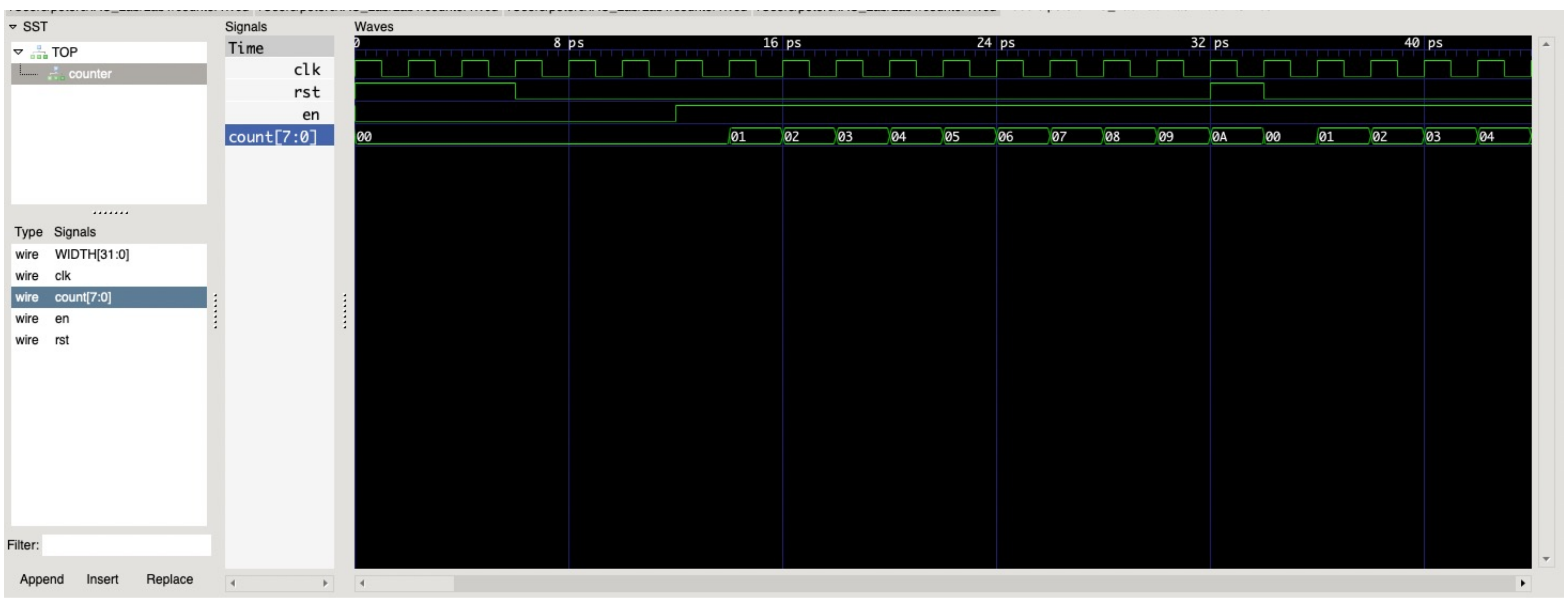


Checking the simulation results

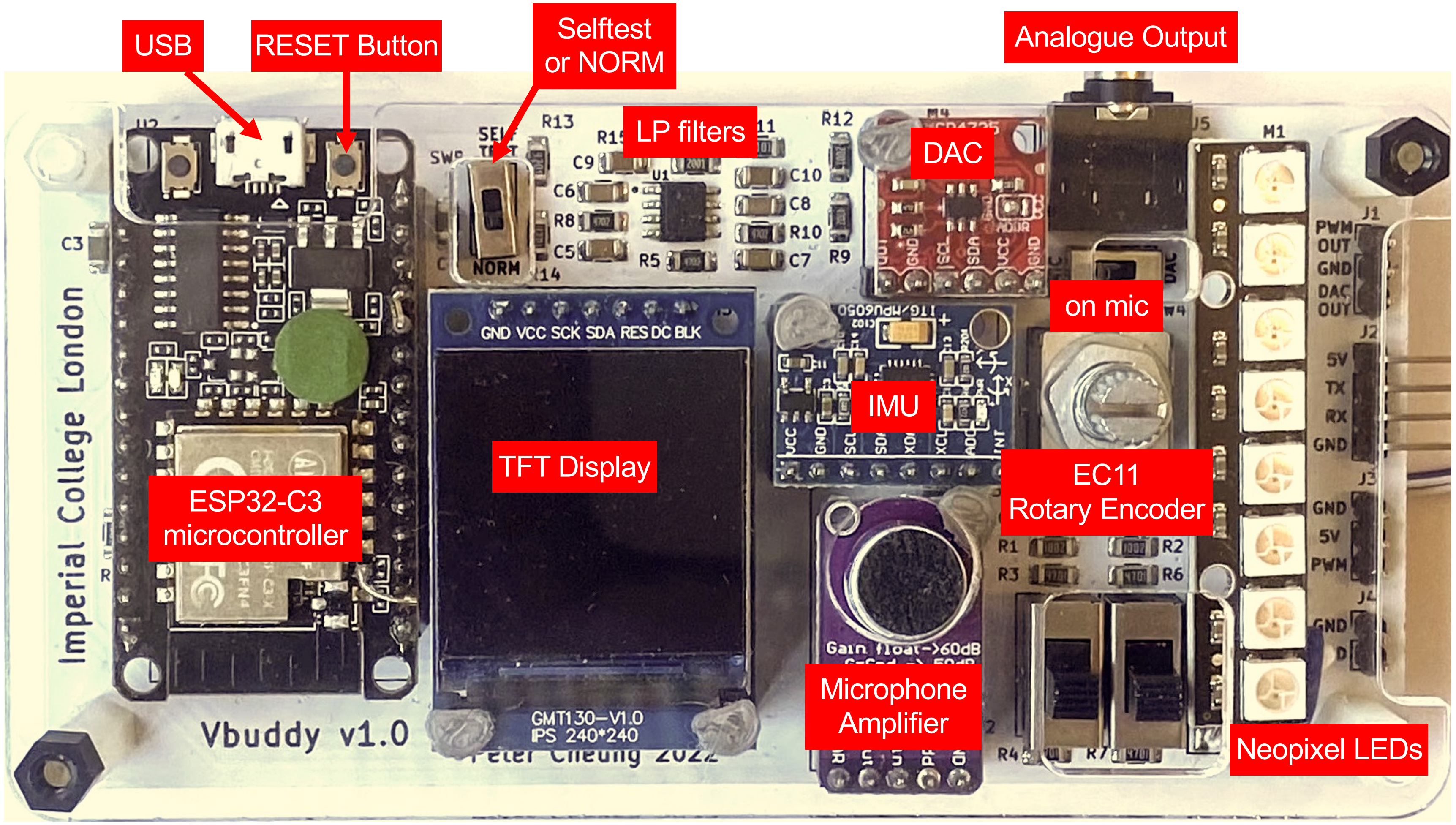
counter.vcd

gtkWave





Vbuddy


USB
RESET Button
Selftest or NORM
LP filters
DAC
Analogue Output
on mic
IMU
TFT Display
EC11 Rotary Encoder
ESP32-C3 microcontroller
Microphone Amplifier
Neopixel LEDs
Imperial College London
Vbuddy v1.0
GND VCC SCK SDA RES DC BLK
GMT130-V1.0
IPS 240*240