

## Lecture 3

# Verilator, Testbench and Vbuddy

Prof Peter YK Cheung  
Imperial College London

URL: [www.ee.ic.ac.uk/pcheung/teaching/EIE2-IAC/](http://www.ee.ic.ac.uk/pcheung/teaching/EIE2-IAC/)  
E-mail: [p.cheung@imperial.ac.uk](mailto: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 User Base



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"

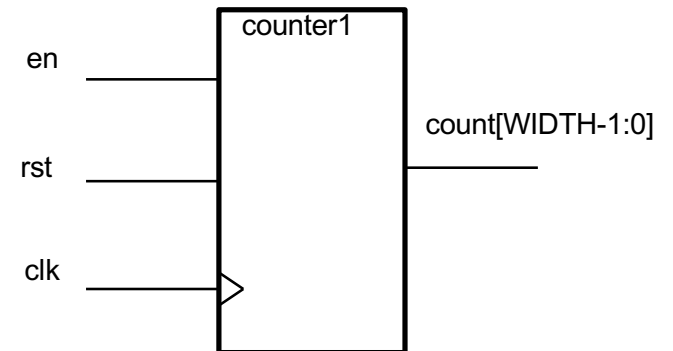
## 3. Event-driven simulator

- Change signal produces event in an event queue
- Simulate delay using timing model
- Capture glitches
- Slower and costly. Example: ModelSim

# Example: Simple Counter

---

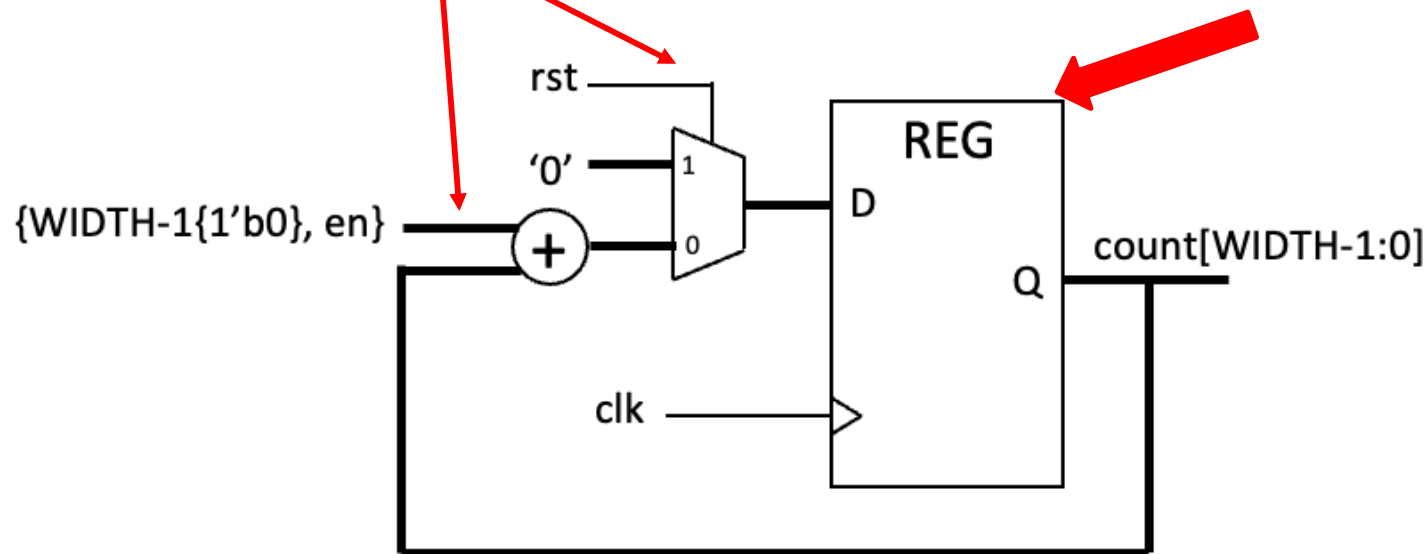
```
1  module counter #(
2      parameter WIDTH = 8
3  ) (
4      // interface signals
5      input  logic      clk,      // clock
6      input  logic      rst,      // reset
7      input  logic      en,       // counter enable
8      output logic [WIDTH-1:0] count // count output
9  );
10
11  always_ff @ (posedge clk)
12      if (rst) count <= {WIDTH{1'b0}};
13      else    count <= count + {{WIDTH-1{1'b0}}, en};
14
15  endmodule
```



# Mapping from SV to hardware

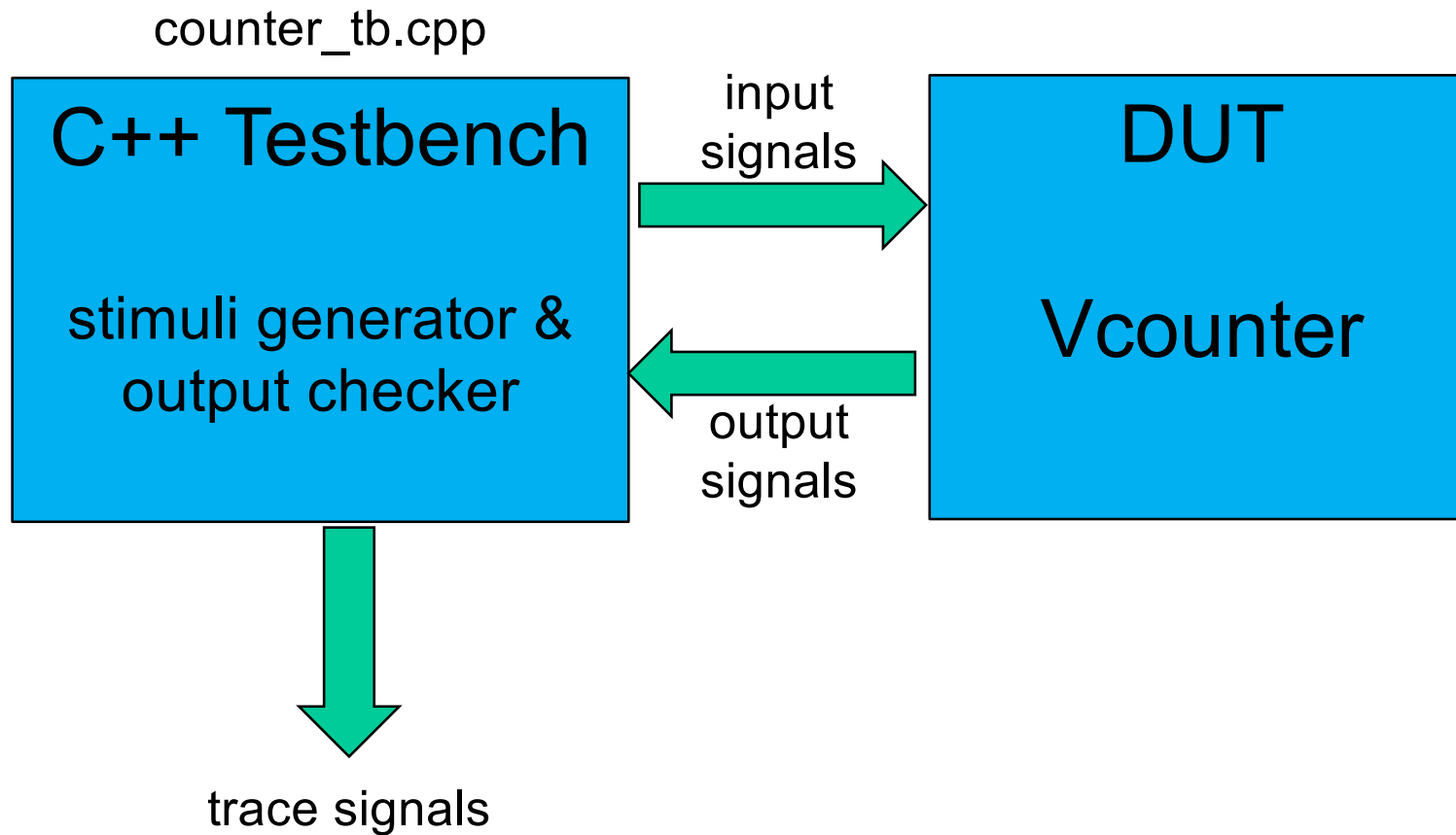
```
if (rst) count <= {WIDTH{1'b0}};  
else count <= count + {{WIDTH-1{1'b0}}, en};
```

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

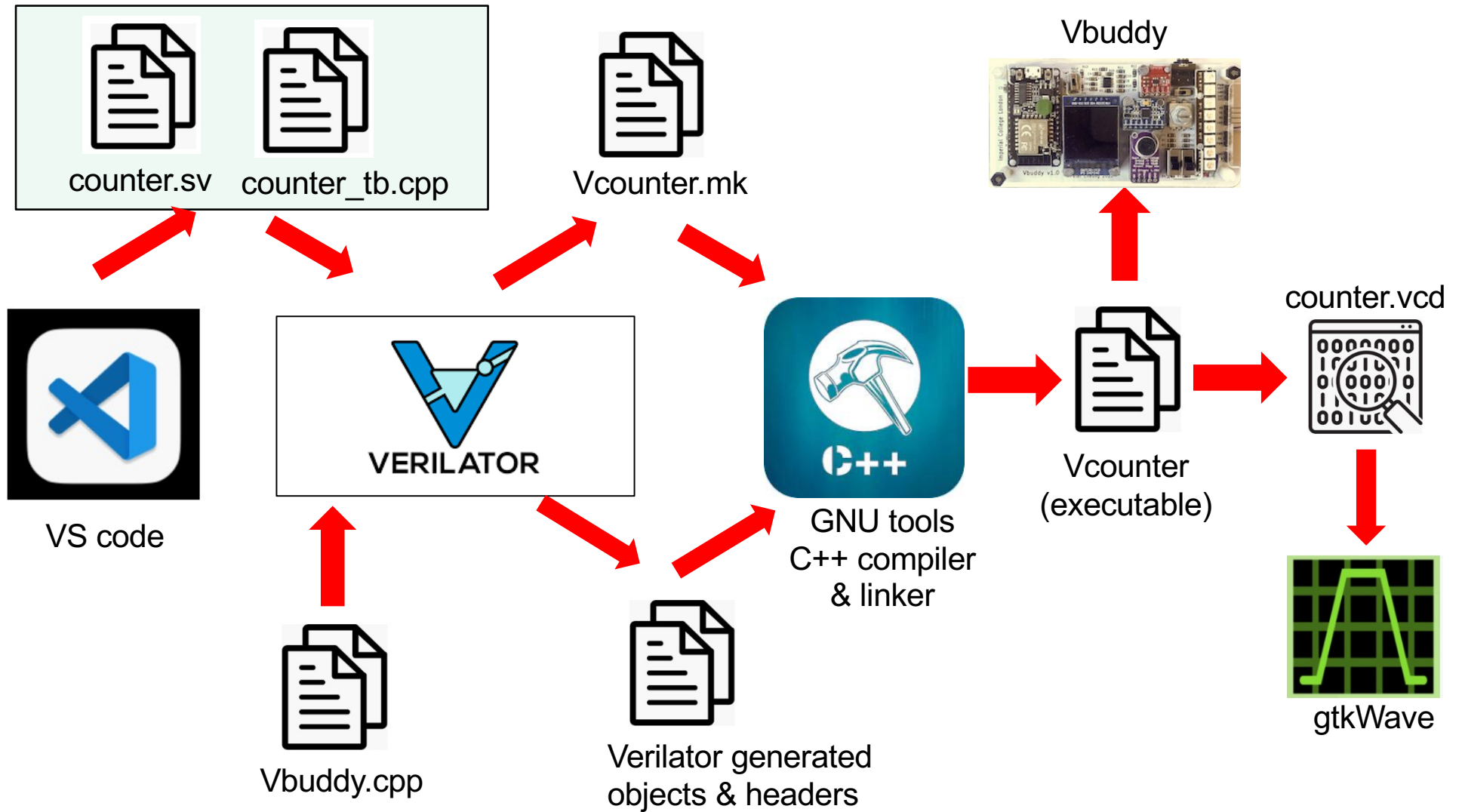


# Testbench

---



# How does Verilator work?



# Format of the Testbench (1)

counter\_tb.cpp > ...

```
1 #include "Vcounter.h"  
2 #include "verilated.h"  
3 #include "verilated_vcd_c.h"
```

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

```
4  
5 int main(int argc, char **argv, char **env) {
```

```
6     int i;  
7     int clk;
```

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

```
9     Verilated::commandArgs(argc, argv);
```

```
10    // init top verilog instance
```

```
11    Vcounter* top = new Vcounter;
```

```
12    // init trace dump
```

```
13    Verilated::traceEverOn(true);
```

```
14    VerilatedVcdC* tfp = new VerilatedVcdC;
```

```
15    top->trace (tfp, 99);
```

```
16    tfp->open ("counter.vcd");
```

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)

```
18 // initialize simulation inputs
19 top->clk = 1;
20 top->rst = 1;
21 top->en = 0;
22
23 // run simulation for many clock cycles
24 for (i=0; i<300; i++) {
25
26 // dump variables into VCD file and toggle clock
27 for (clk=0; clk<2; clk++) {
28     tfp->dump (2*i+clk); // ur
29     top->clk = !top->clk;
30     top->eval ();
31 }
32 top->rst = (i < 2) | (i == 15);
33 top->en = (i > 4);
34 if (Verilated::gotFinish()) exit(0);
35 }
36 tfp->close();
37 exit(0);
38 }
```

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 outputs the trace for each half of the clock cycle, and forces the model to evaluate on both edges of the clock.

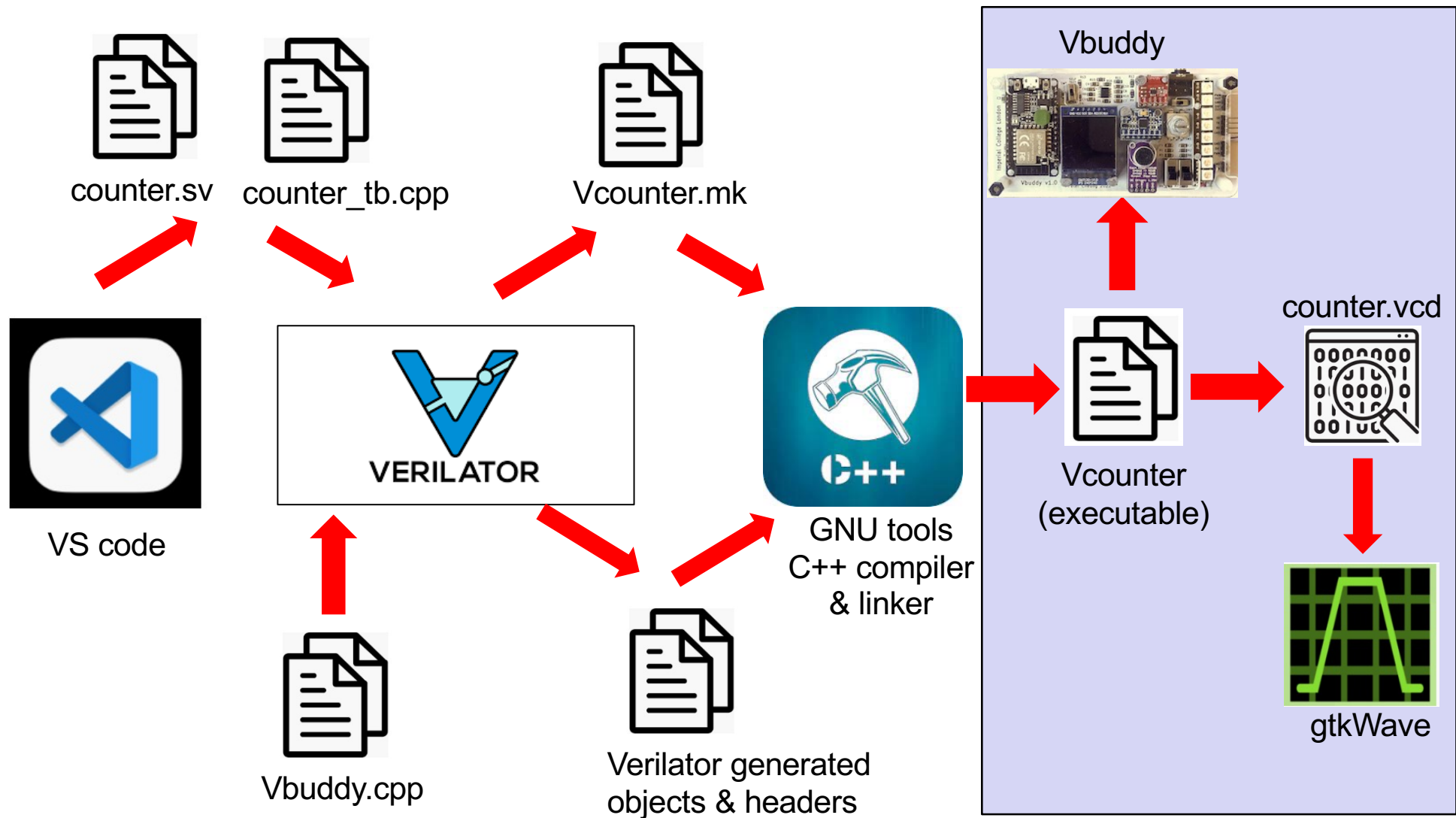
Change rst and en signals during simulation.

# Making the final simulation model

---

```
$ doit.sh
1  #!/bin/sh
2
3  # cleanup
4  rm -rf obj_dir
5  rm -f counter.vcd
6
7  # run Verilator to translate Verilog into C++, including C++ testbench
8  verilator -Wall --cc --trace counter.sv --exe counter_tb.cpp
9
10 # build C++ project via make automatically generated by Verilator
11 make -j -C obj_dir/ -f Vcounter.mk Vcounter
12
13 # run executable simulation file
14 obj_dir/Vcounter
```

# Vcounter is the executable model of counter

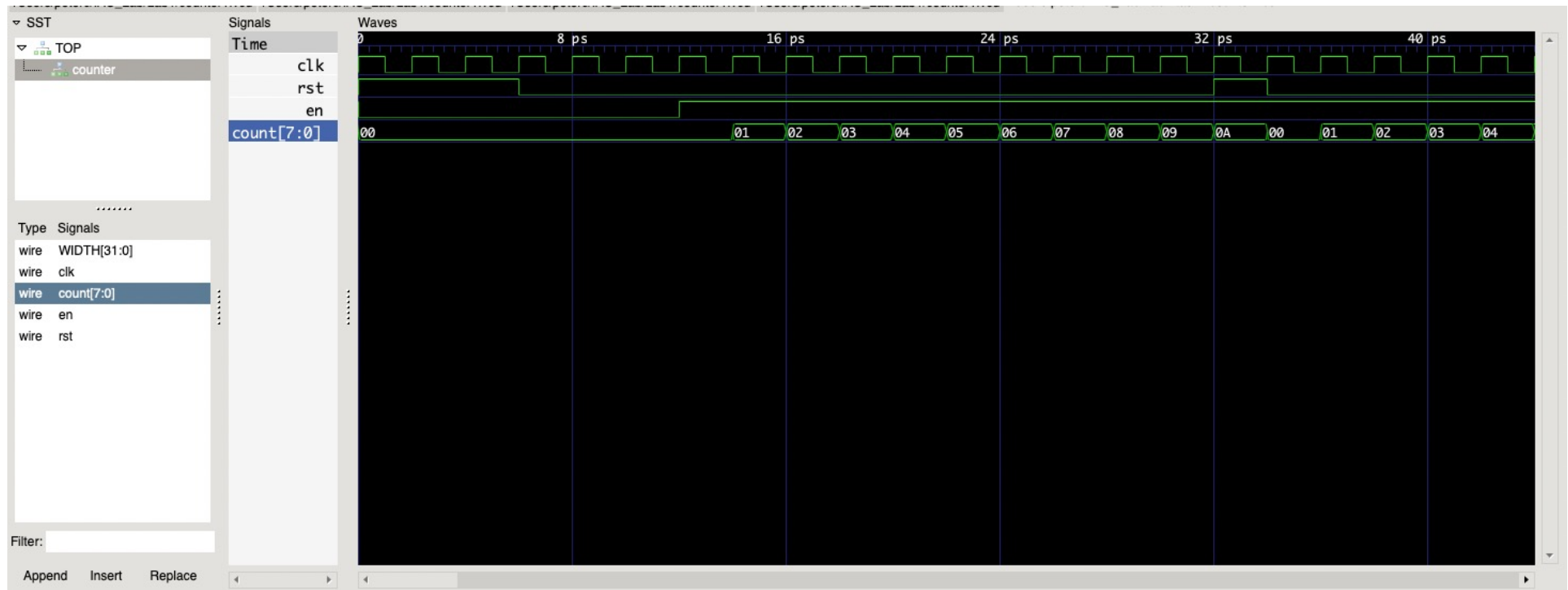


# Checking the simulation results

counter.vcd



gtkWave



# Vbuddy

