Architecture of Digital Integrated Systems

Prof. Davide Bertozzi

COMPUTER SCIENCE LAB

Fundamentals of SystemC Programming

SYSTEMC™ WITH TLM

ING. DOTT. MARCO BALBONI
marco.balboni@unife.it
skype marcobalboniskype
ENDIF OFFICE 305-306
Tel. +39 0532 974989
Cell. +39 3487621827

MPSoC Research Group Activities

MANY-CORE PLATFORMS MODELLING AND SIMULATIONS

FPGA PROTOTYPING

HW AND SW ENABLING VIRTUALIZATION

EMERGING TECHNOLOGIES SYNTHESIS AND PLACE&ROUTE

Marco Balboni
MPSoC Research Group

ENDIF

INSTALLING SYSTEMC 2.3.1 LIBRARY

- Make sure of having installed in your system the build-essential package.
  If no, run on terminal

```
:~$ sudo apt-get install build-essential
```

  Now you can check both your gcc and g++ version with

```
:~$ g++ --version && gcc --version
```

- Download "SystemC-2.3.1 with TLM" source from Accellera.
  http://www.accellera.org/downloads/standards/systemc/files

- Decompress the package using

```
:~$ tar -xvf systemc-2.3.1.tgz
```

- Make a directory for systemc installation in your */usr/local* path

```
:~$ mkdir /usr/local/systemc-2.3.1
```

- Patch
  Open systemc-2.3.1/src/sysc/datatypes/bit/sc_bit_proxies.h and erase all the words "mutable".
- Set environmental variables

```
:~$ setenv CXX g++    OR    :~$ export CXX=g++
```

- Configure and install

```
:~$ cd systemc-2.3.1                                          optional for x86 architectures
:~$ ./configure --prefix=/usr/local/systemc-2.3.1 [--host=i686-linux-gnu]
:~$ sudo make  && sudo make install
```

COMPILING & BUILDING: MAKEFILE

- Set the library path in your environmental variables (needed to avoid errors during simulation):
  `:~$ export LD_LIBRARY_PATH=/usr/local/systemc-2.3.1/lib-linux64`
  (to make it permanent add this line to your .bashrc file in Ubuntu, or equivalent for other distros)
- Compile source files to build the executable:

MAKEFILE

```
1  ############## Target architecture
2  SYSC_TARGET_ARCH ?= linux64
3  ############## Variables that point to installation paths
4  SYSTEMC          ?= /usr/local/systemc-2.3.1/
5
6  ############## Default compiler and possible option sets
7  CC               = g++
8
9  OPT              = -Wall -Wno-deprecated -fexceptions -O0 -g


   #[...]#

50 CPPSRCS  += main.cpp counter.cpp
51 CSRCS    +=
52 CPPOBJS   = $(CPPSRCS:.cpp=.o)
53 COBJS     = $(CSRCS:.c=.o)
54 OBJS      = $(CPPOBJS) $(COBJS)
55 VPATH    += .
56 EXE        = simulation.x

#[...]#
```

- Set the target architecture considering your specific machine:
  - linux for 32bit machines;
  - linux64 for 64bit ones.
- Set the absolute path of your SystemC installation.
- g++ compiler is set to default version. If you have more than one version you have to specify it (e.g. g++-4.6).
- List of the .cpp sources to be compiled in the project.
- Name of the executable file generated if compilation and building have success.

- Compile/Build `:~$ make clean all`

WAVES VIEWER: GTKWAVE

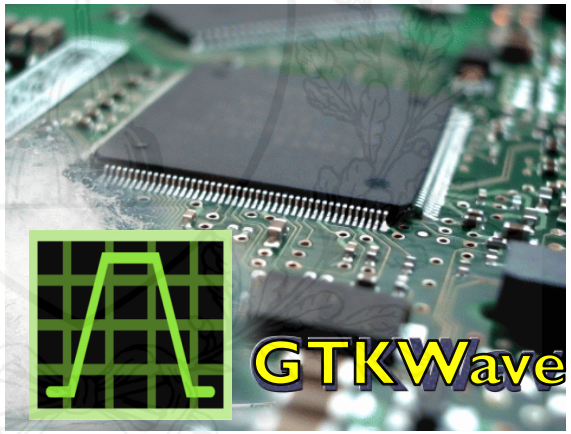
- GTKWave is a fully featured GTK+ base viewer for Unix, Win32 and Mac OSX.
- Installation on Unix systems:

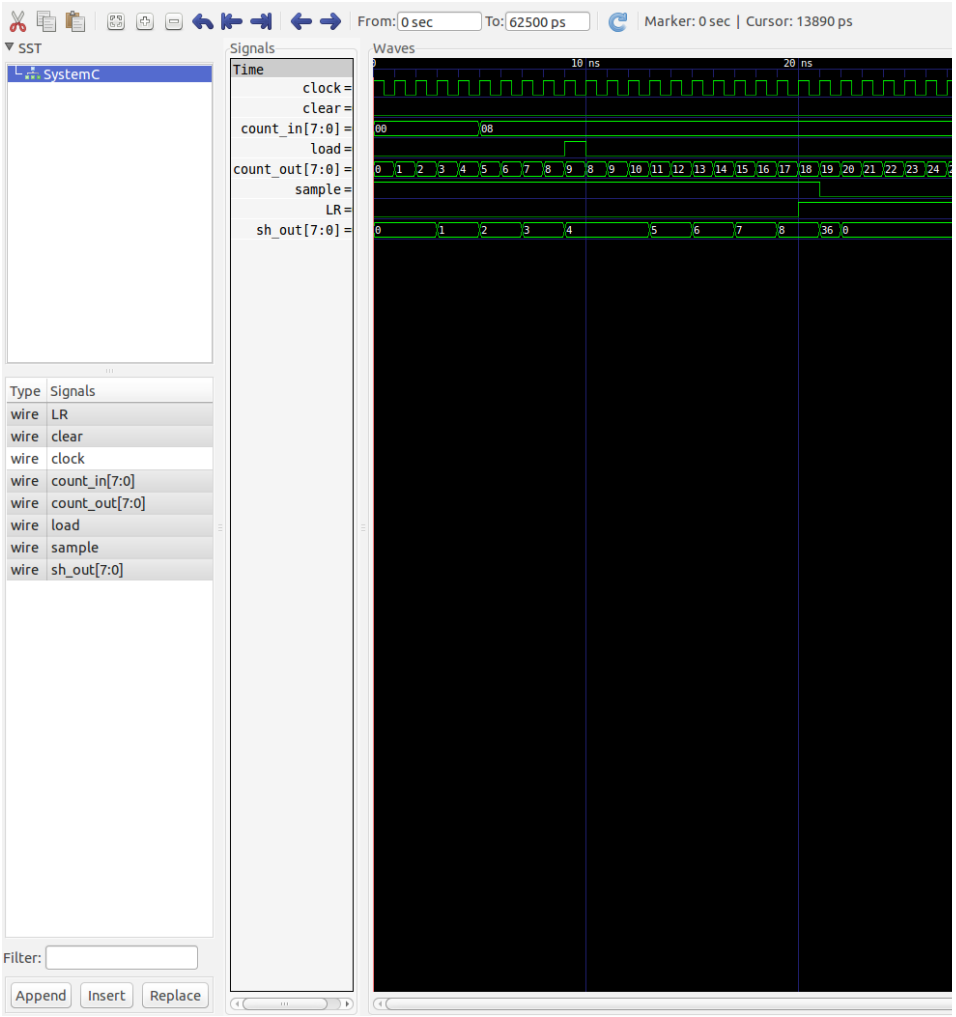
```
:~$ sudo apt-get install gtkwave
```

- Open a Waves VCD (Value Change Dump file)

```
:~$ gtkwave *.vcd
```

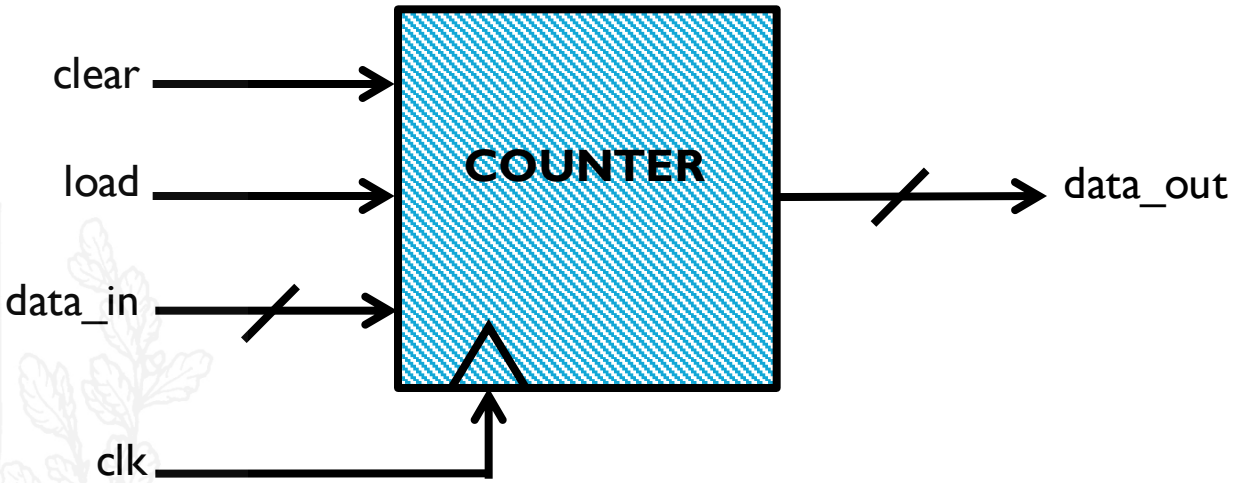
- Default Timescale unit used for tracing is 1 ps.





PROJECT I: COUNTER

- **Blocks Diagram**

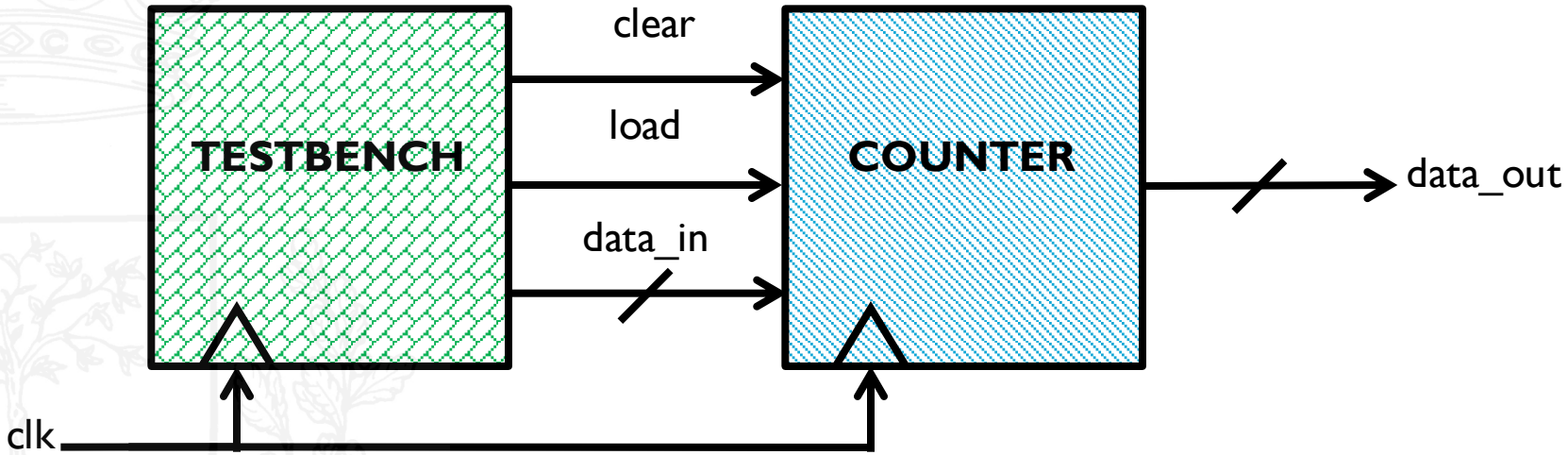


- **Functional Specifications**

The blocks diagram shows a 8bit counter.
The counter output (**data_out**) can be:
- increased by 1 unit each **clk** rising edge;
- set to a specific value placed in **data_in**, by setting the input **load** to 1;
- cleared by setting the input **clear** high.

PROJECT 2: TESTBENCH

- **Blocks Diagram**

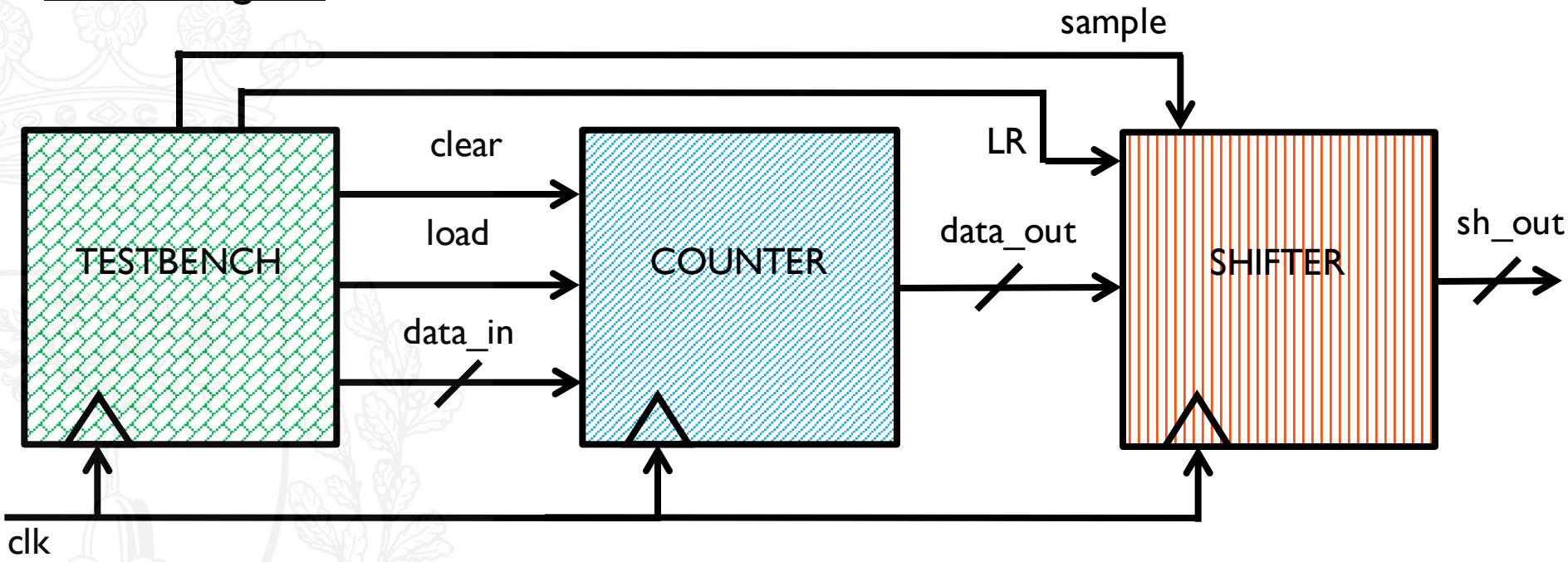


- **Functional Specifications**

The testbench has to drive the inputs of the counter block.
In particular:
- it initializes **clear=0**, **data_in=0**, **load=0**;
- at 5nsec it sets **data_in=8**;
- at 9nsec it sets **load=1** for 1 clock cycle;
- at 52nsec it sets **clear=1** for 1 clock cycle.

PROJECT 3: SHIFTER

- **Blocks Diagram**

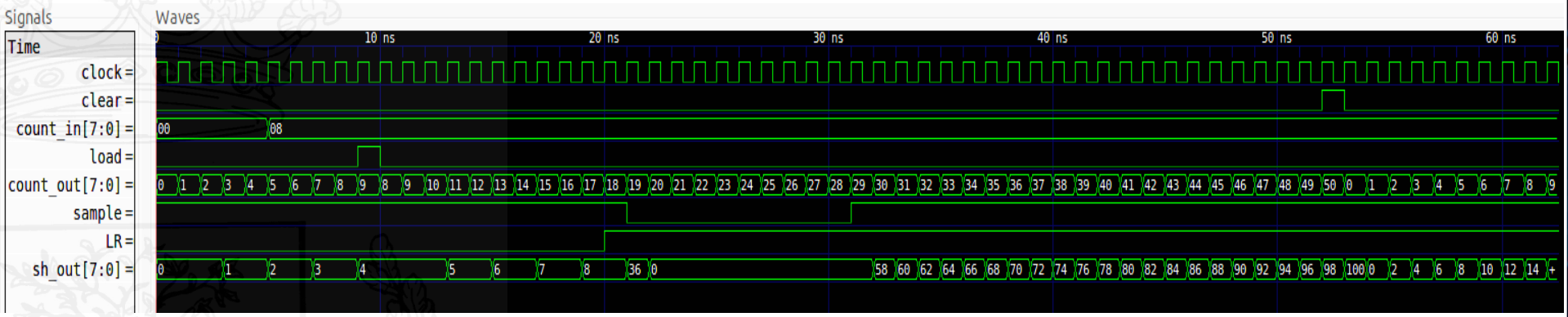


- **Functional Specifications**

A very basic 8bit shifter. The shifter can be loaded on a **clk** rising edge by placing a value on input **data_in** (in this case it is the output provided by counter block) and setting input **sample=1**. The shifter will shift the data left or right depending on the value of input **LR**. If **LR** is low the shifter will shift right by one bit, otherwise left by one bit.

!!!!! Testbench needs modifications too, in order to drive new inputs for the shifter (**LR**, **sample**).

WAVEFORMS



- Expected Waveforms if compiling, building, binding, simulaton were good ☺.

NB: have a look at "Practice Exercizes 1-2-3" on the site to find and grab the code.