# Cycle Accurate Model

# Introduction

The cycle accurate model is a C++ class based model for RTL and hardware modeling. It can be used as a software simulator before Verilog/VHDL coding. Although called “Cycle Accurate”, it can also describe transection level model, if you take “cycle” as “phase” in SystemC. If you do not want to use SystemC or any commercial EDA simulation tools such as VCS, this model is a good start for modeling and simulating your hardware design.

The lib file for the model includes 3 files:

* common.h:

Commonly used head file, including declaration of global variables, definition of ca\_signal class, and the declaration of assistant functions used for dump VCD(Value Changed Dump) file.

* common.cpp:

Including definition of global variables, and functions used for dump VCD file.

* ca\_module.h

The definition of an abstract class ca\_module. All of the user defined modules are derived from this class.

# Global Variables

There global variables are defined in common.cpp:

* cycle

This is the cycle counter for the simulation.

* vcd\_file

A file handle for waveform file

* time\_record

A bool flag indicate whether record current simulation cycle in the VCD file.

# Abstract Base Class ca\_module

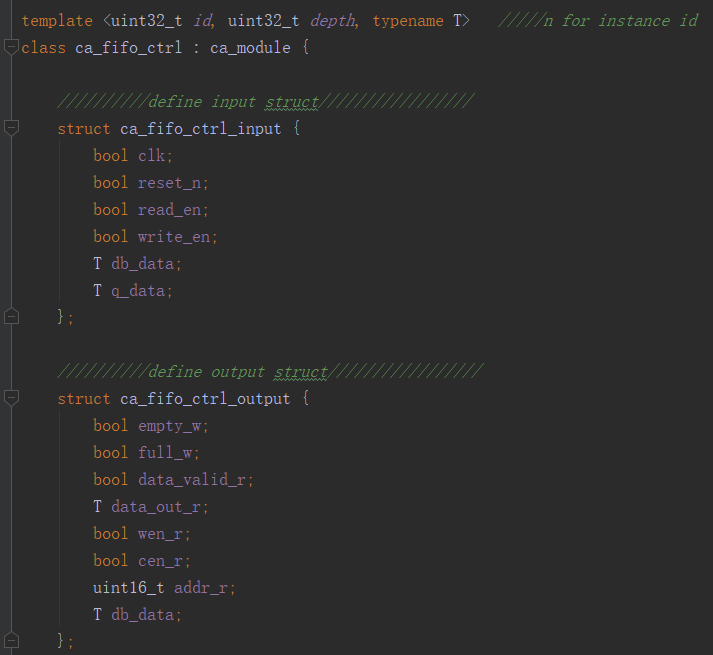
The ca\_module class is an abstract base class for all the user defined modules. Five pure virtual functions are defined, three for simulation (connect\_submod, run and update) and two for dump VCD file (is\_trace, dump\_sigs).

# Modules

The user defined modules should be derived from the ca\_module class.

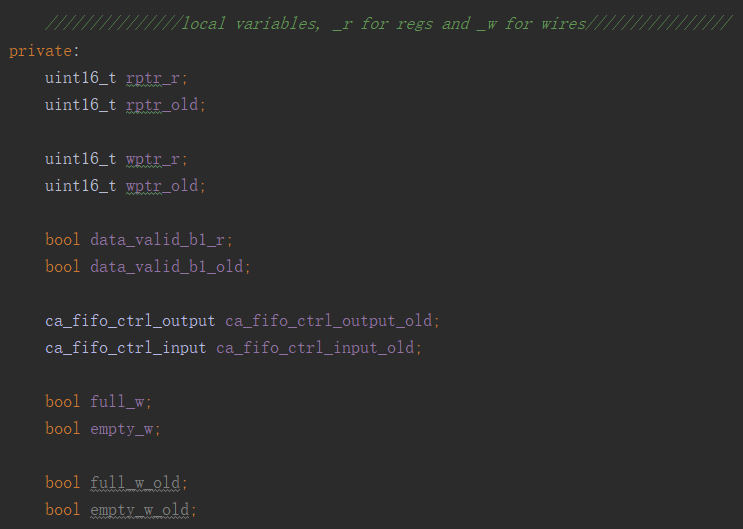
## Input and output structure

In the ca\_fifo\_ctrl example, the input and output struct are defined at the beginning of the class:



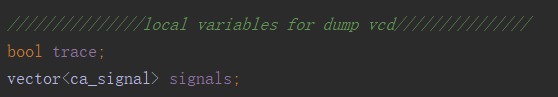
## Signals

The next section is the registers and wires used in the module, defined as private members. The suffix \_r indicates the reg signal type, \_w for wire signal type, \_old for the value of last cycle.



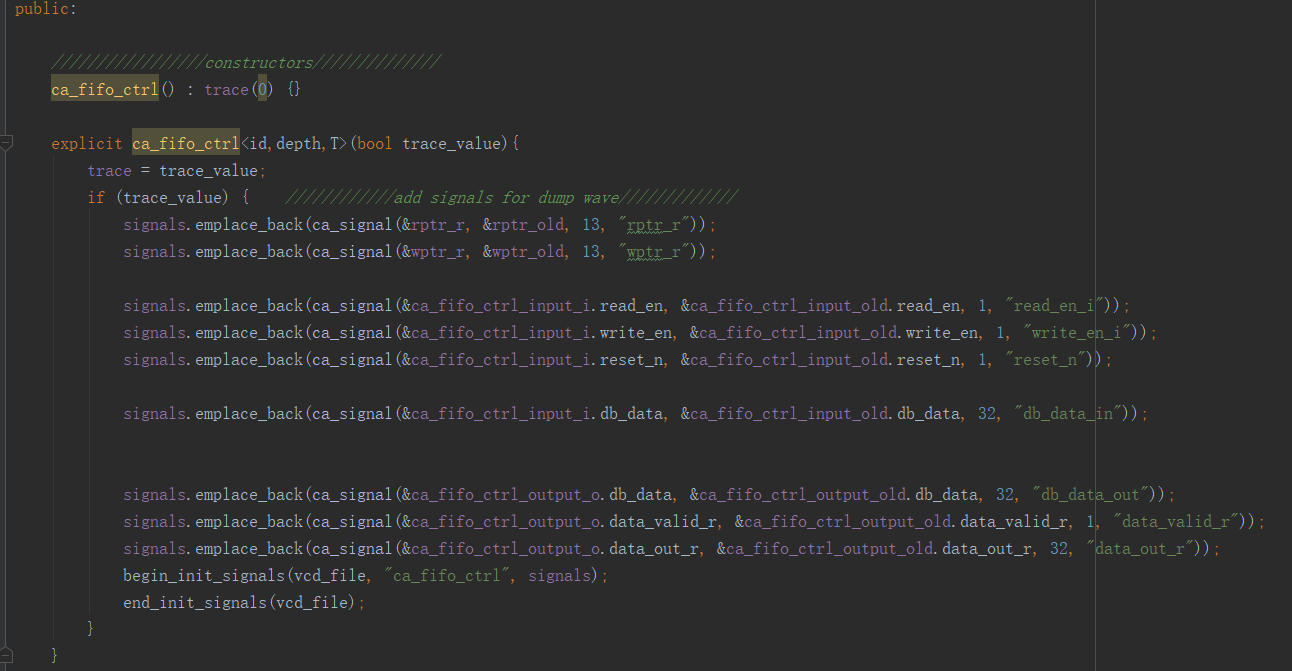
## Local variables

Next is the local variables for dump VCD files. The bool variable trace is the flag for whether dump waves for this module, the signals vector is used for store the variables which you want to dump waves



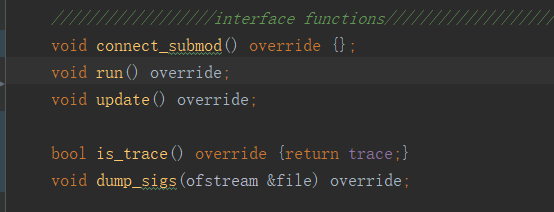
## Constructors

Two constructor functions are defined in the ca\_fifo\_ctrl example. You do not need constructor function if you do not want to dump waves for this example.



## Interface functions

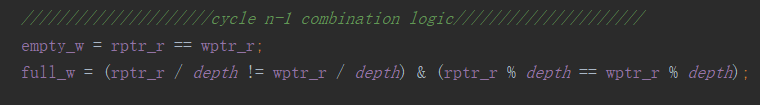
The interface functions are declared as follows:



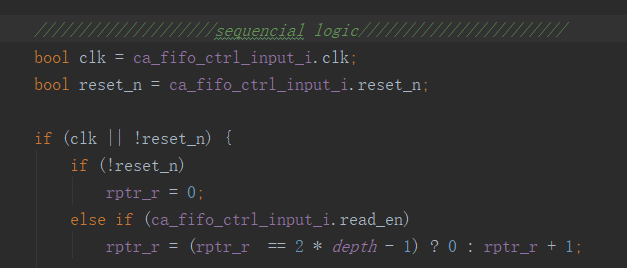
Because this module has no submodules, the connect\_submodule is an empty function and do nothing.

## void run()

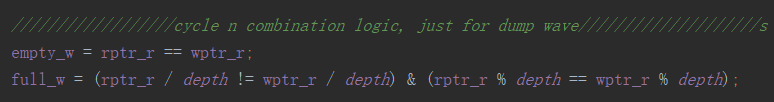
The run function uses the valus of the n-1 cycle to calculate the value of the n cycle. The first section is the calculation of the combination logic of the n-1 cycle:



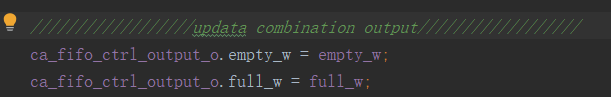
The next section is the sequential logic calculation for the n cycle value, Note that the right side of the assignment must be \_old and \_w variables, which has the value of n-1 cycle.



Then calculate the n cycle value of the combination logic:

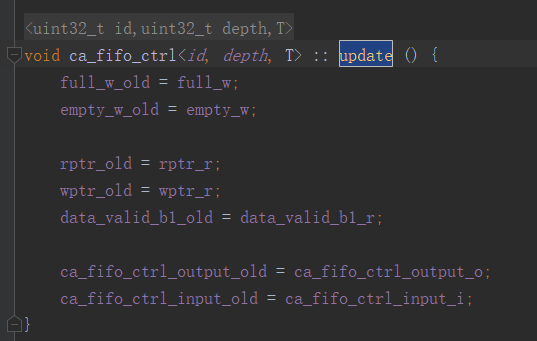


Finally assign the value to the combinational logic outputs:



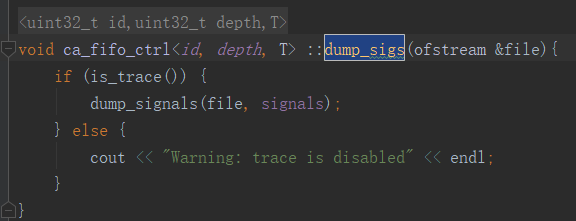
## void update()

The update function store the n cycle value to the \_old variables, which will be used at the next iteration:



## dump\_sigs

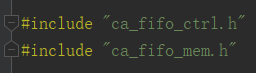
dump\_sigs function is used for dump registers and wires, dump\_inputs is used for dump input signals:



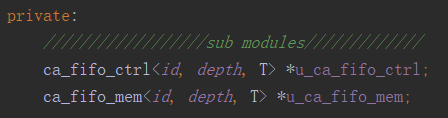
# Hierarchy Design

The top module of hierarchy design need something more besides the submodule design:

## Include submodule headers

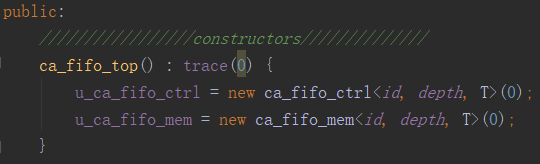


## Submodule members

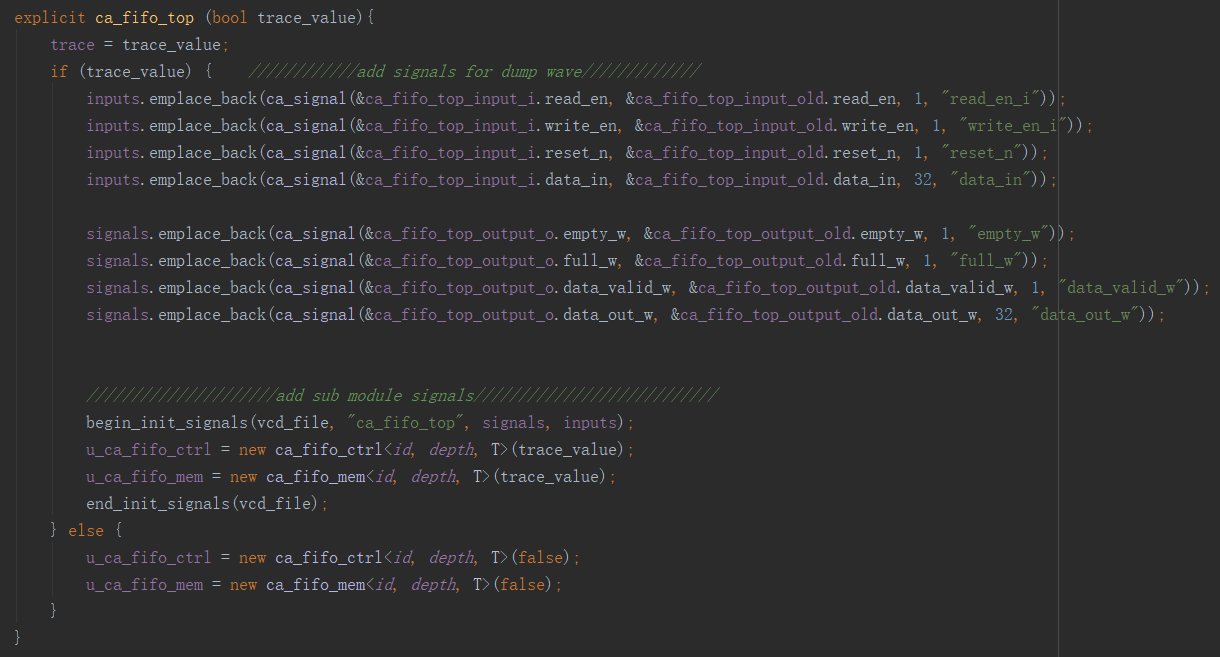


## Constructors

Submodules should be instantiated in the constructors:

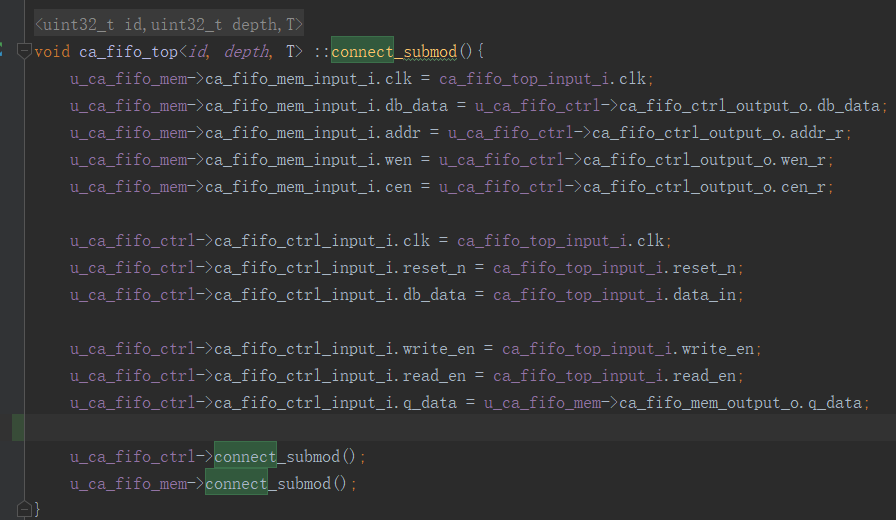


If you want to dump waves for the top module signals, add the signals and inputs to the lists. Then initial the signals, then instantiate submodules, which will call the sub module constructors and add submodule signals. Finally end initials signals. If you do not need to dump waves, just call the submodule constructors with false parameter.



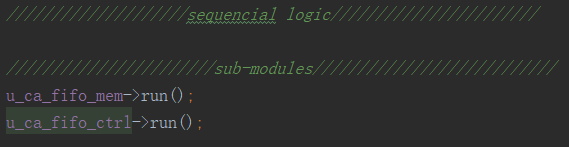
## Connect submodules

Connect submodule inputs and outputs, and then call submodule’s connect\_submod function.



## run

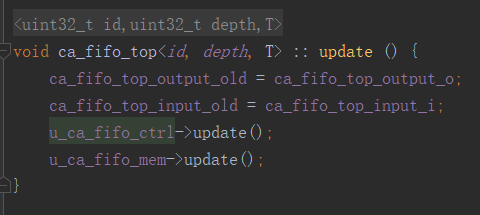
Run the submodules after sequential logic:



Other parts are the same as submodules.

## update:

store the cycle n value to the \_old variables and then update submodules:



# Dump VCD Wave Files

VCD files is a standard format for waves. The standard reference could be found here: <https://zipcpu.com/blog/2017/07/31/vcd.html>.

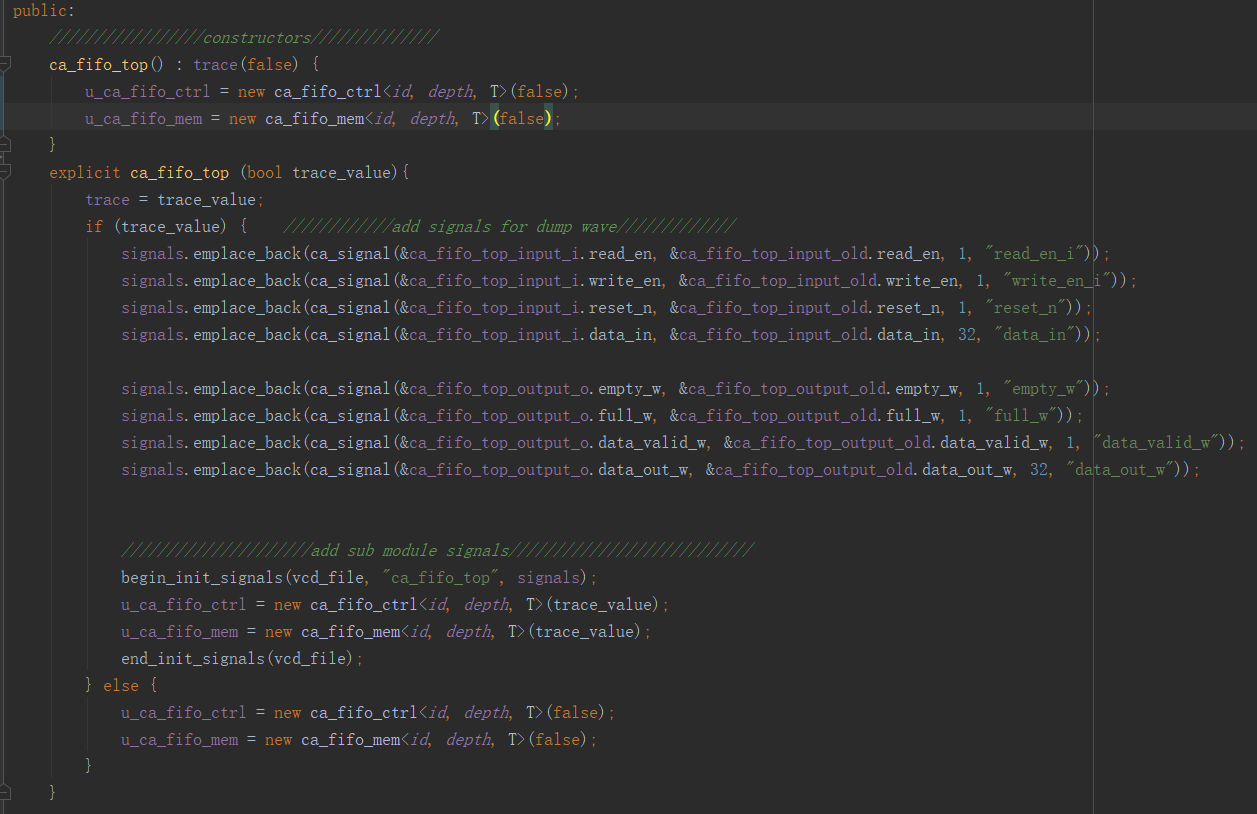
The VCD file can be opened with gtkwave, an opensource wave viewer: <http://gtkwave.sourceforge.net/>. Verdi can convert VCD file to FSDB file automatically and view the wave.

All the tool functions used to dump VCD file is defined in common.cpp. In the user defined module, you need to define the signals for which you want to dump. Usually it is defined in the constructor functions, as shown in 4.4.

The ca\_signal class is defined in common.h. 4 members are defined in the ca\_signal class:

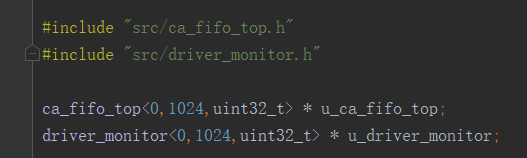
* void\* data: used to store the pointer of the value of the n cycle;
* void\* old\_data: used for storing the pointer of the n-1 cycle, usually defined with suffix \_old
* uint8\_t len: indicate the bit length of the signal.
* string name: used to store the name of the signal.

In the constructor, 4 steps are need for dump wave: add signals, begin\_init\_signals, call the constructor of the submodules, and end\_init\_signals:

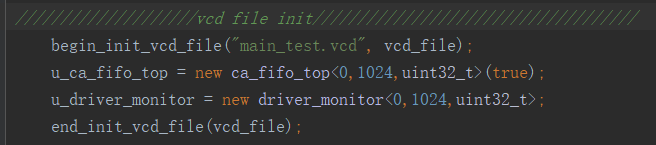
Simulate the design

A driver module and a main function are needed to simulated the design. A driver module can could be a special ca\_moudule subclass with only output interface. The driver module example is driver\_monitor.h

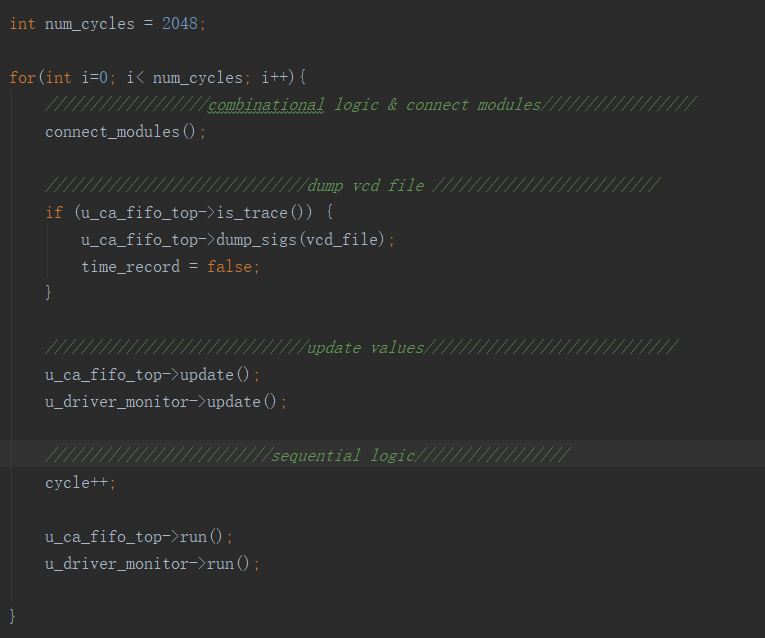
In the main.cpp file, include the header files of the design and driver module, and define them:



In the main function, first initial the VCD file, and instantiate the modules:



Then use a loop (for loop if you want to simulate a certain cycles, or wihle(1) loop if you want the driver\_monitor module break the loop) to simulate the design:



# Future work

## Support multi clock domain design

## Unify the inputs and signals wave dump(Done)

Because in the run() function, we use the n-1 cycle values to calculate n cycle values, the inputs can not get the n cycle value until the n+1 cycle(in the submodule\_connect), the inputs and other signals are dumped using different functions(use n-1 time marker for inputs and n for other signals).

## Transection Level Model examples

## More designs coming