

This is the first part of a series on modeling and verification levels of abstraction. In this article we will review the basic framework for defining levels of abstraction. The second article will focus specifically on the different types of Transaction level modeling styles, definitions, and standards. The third article will focus on practical application for these levels.

As an industry we have developed a set of working definitions to describe the level of abstraction that is presented by a particular design representation. These definitions, however, are vague at best and leave room for interpretation even at the levels that we believe we all understand. In this article we're going to take a look at commonly defined levels of abstraction, why we care, and what they are used for.

Gate Level

Even the lowly gate level has its share of confusion. When we say "gate level", what we usually mean is actually "library primitive level". In other words, the level at which we have mapped out logic onto the actual library primitives that are available in whatever library we are targeting for our ASIC or FPGA. A "gate" here can be a nand gate, or an adder, or even something more complex.

Alternately, we could consider a gate level representation to be real logic gates. Here, too, we run into the problem of where to draw the line on complexity: what, exactly is a "gate"? How complex a function do we need to have before it is not a gate?

Our working definition of gate level is usually whatever comes out of the synthesis tool that we can send to the back-end folks to begin layout on the chip. In practical terms, a "gate" is anything that can be placed and routed – in other words, a netlist. The gate level "netlist" by itself tells us nothing about the behavior of the circuit unless you include the library primitives that have their functionality described in them. It does, however, tell us everything about the actual structure of the circuit.

Models at this level are always "pin accurate", meaning that behavior on a pin in the actual device maps directly to behavior in a pin (or port) of the model. Gate level models are timing accurate with respect to the actual logic, and estimated or derived interconnect delay can be back-annotated onto the nets to reflect the timing of the actual implementation. They are structurally accurate since they represent the actual structure of the logic, and they are functionally because the full functionality of the circuit is defined.

Gate level models are important because they represent the actual implementation of the device. It defines the connectivity and the set of library primitives that are required to drive the place and route of the actual chip.

The interface to the device takes the form of I/O signals transitioning and, typically, being clocked into the I/O registers. At the gate level, the exact time that the I/O signals

transition might be modeled accurately with respect to the clocking signal in order to verify the timing of the I/O logic.

RTL

Register Transfer Level (often referred to as “RTL Level” which is redundant) refers to the level where a circuit’s behavior is described a series of synchronous data transfers between modules. At its simplest, this takes the form of clouds of combinatorial logic n between registers.

RTL models are timed, although the timing of internal signals may not match the actual implementation. Signals may exist in the RTL logic that do not exist in the implementation, and vice versa. RTL models do, however, support the ability to have as many sequential event transitions within a clock cycle as required.

RTL models, coupled with logic synthesis, provide a way to specify circuit behavior in a more efficient and productive way, but at a low enough level to be able to drive synthesis tools to create the actual gate level representation.

RTL models are written in Verilog or VHDL. While other languages could be used, the whole point of using RTL is to drive synthesis and that means Verilog or VHDL.

Cycle Level

One step up from RTL is the Cycle level of accuracy. At this level the design is represented by a series of registers along with logic, much as the RTL. The behavior of the circuit, however, is guaranteed to match the RTL implementation only at the cycle boundaries. When the registers have been clocked, the resulting values will compare between the lower levels. What happens in between, and how the new register input values are derived, is not defined.

Cycle accurate models are still bit accurate models, but event activity only occurs on the clock edges and not in between. In practical terms, the main differences between cycle and rtl models is that cycle level models are not synthesizable and run faster than RTL.

The interface to the cycle level model is usually identical to that of the RTL model or the gate level model. Signals transition on the I/O pins and are clocked into the I/O registers.

Transaction Level

The transaction level is where definitions really start to diverge from person to person. Different people have different ideas about what transaction level modeling actually includes. In fact, the space carved out by transaction level modeling includes a very wide range of modeling and communication styles.

Transaction level models can be completely un-timed models on the high end of the abstraction scale. The registers remain, although only the ones that are relevant to the programming interface. Other registers might not be represented in the design at this level.

At the lower level of this range, the models may be fully timed and much of the actual design structure might be represented in the model. Individual operations or communications, however, do not always occur at the single cycle boundaries. The timing might be consistent with a lower level model, but only at multi-cycle boundaries. For example, a data transfer that might take 8 individual clock cycles at the bit level could be done in just a single transaction.

Transaction models include some level of design structure. At the high end of the transaction level this may only be major blocks or even specific registers. Transaction models are not bit-level accurate, and are generally not single-cycle accurate, although they may be multi-cycle accurate. Transaction level models are typically written in C/C++ or SystemC

There are many definitions of the sub-levels for transaction modeling ranging from OSCI to OCP to various EDA companies to academia to end users. In the next article in this series we will take a more detailed look at the transaction level(s).

A call to a JPEG decoder, for example, at the higher end of the transaction level could look something like:

```
Wr_decoder_control ( 'reset' )  
Wr_decoder_data (image)  
Wr_decoder_control ( 'decode' )  
Rd_decoder_data ( image )
```

At this level we still have the main blocks of the decoder circuit and the basic sequence of operations. First we reset the decoder, then we write the image data to its internal memory, then we set the control bits to tell it to decode, then we read the decoded data. Note that at this level there is no concept of time so we do not have to model the time it takes the decoder to actually perform the operation.

At this level we still have preserved the functional interface to the model, however, and that means that while much of the actual implementation details are missing from the

inside of the model, it “looks” like the implementation from a programmers perspective. The reads and write to the control registers and data buffers model the actual implementation, and software written to stimulate this model can be reused on the final implementation itself.

Functional Level

The highest level of abstraction has less controversy about what it includes, but there are many different labels for what to call it. This might be referred to as functional, behavioral, algorithmic, executable specification etc.

At this level include having no timing and having no resemblance to any of the actual hw implementation structure. There are no pins and no cycles. Functional models are typically written in C or C++, or often created using tools like Matlab. The call to our JPEG decoder at this level might look simply like:

```
Jpg_decode ( image )
```

Where “image” is a data structure that contains the image data. This is a single call to a function, and only the actual operation of the function is represented with none of the actual control sequences required for the actual hardware implementation.

At this level performance is very fast but we have no information about the actual implementation. Because of the high performance and compact representation, this level of abstraction is useful for modeling system functionality and even entire end-to-end systems, including transmission line characteristics. It is not useful for modeling system performance characteristics because operations happen “instantly”.

Conclusion

We took a quick look at various levels of abstraction in this first article. The gate and RTL levels are fairly well agreed upon and understood. Cycle-level modeling is also largely understood in it’s characteristics. These levels are the ones that are most commonly used today at different stages of the design process. The level that represents the main thrust in changing the common methodology today is the transaction level, and that is still very much in flux, but is the next level to become the focus for common usage. The highest level, functional, is used to validate early algorithm and basic system functionality but is completely divorced from the actual implementation.

Each level has its place and provides an optimal level of support for a part of the design process. As designs become more complex we are moving to higher and higher levels of abstraction to model complete system behavior. This in turn is driving the development of standards, tools, and methodologies to support verification at these levels.

