Analyzing and Verifying Locally Clocked Circuits with the Concurrency Workbench

Garth Baulch    David Hemmendinger    Cherrice Traver
Department of Electrical Engineering and Computer Science
Union College, Schenectady, NY 12308
gbaulch@acm.org; {hemmendd, traverc}@cs.union.edu

Abstract

Locally Clocked Modules (LCMs) allow asynchronous communication between synchronous computational elements. The Concurrency Workbench models concurrent systems in the CCS process algebra. We describe the use of the Concurrency Workbench to specify, simulate, and verify implementations of LCMs and discuss its application to the specification of asynchronous circuits.

1. Introduction

Globally asynchronous systems of locally clocked modules are one solution to the clock skew problem for large systems [3, 10]. Traver describes a testable structural model for these systems, consisting of a synchronous element with a stoppable clock and an asynchronous interface to control handshaking between modules and initiation and termination of the clock [13]. This paper addresses the verification of a modified version of this structural model using the Concurrency Workbench (CWB), a tool for analyzing finite-state systems. The CWB has been used to verify several communication protocols [5] and mutual exclusion algorithms [14]; our purpose is to demonstrate its utility in asynchronous circuit verification.

Circuit verification must show that a specification is correct, that an implementation meets it, and that important properties hold of it. All three tasks can be carried out with the CWB. We first state the specification in terms of agents in its process algebra, simulate it, and prove appropriate temporal properties of it. We then describe the circuit-level implementation separately in the CWB as a different set of agents, and prove that it is equal to the specification. In this case the proof of important properties can be carried out on either specification or implementation. Finally, we have used the CWB to obtain and verify new specifications of the LCM that permit greater concurrency while preserving the essential temporal properties.

2. LCM specification

The modified LCM model used in this investigation differs from that presented in [13] by using bundled data channels and a single control for the output latches. Asynchronous request and acknowledge control signals are associated with the input and output bundled data channels. The input control signals (ir and ia) and output control signals (ocr and oca) follow a return-to-zero protocol. The use of bundled data simplifies the internal structure but adds a delay constraint to the input and output request signals.

The top level specification of the model includes the input and output 4-cycle handshaking along with the restriction that the output data cannot become valid before the input data and that new input data cannot be requested before output data has been acknowledged.

The LCM is composed of the modules shown in Figure 1: a synchronous core, a stoppable clock with a clock-enable generator to start and stop it, input and output latches, and a basic control module. The basic control logic coordinates the input and output asynchronous handshaking with the control signals to and from the clock-enable generator circuit. This control logic can be designed to be speed-independent, but the clock-enable generator has unavoidable timing restrictions due to its interface with the synchronous portion of the LCM.

![Figure 1. Internal structure of LCM](image)

The signals sr and sa link the basic control logic module to the clock enable generator circuit and these two
signals also follow an asynchronous handshaking cycle. The ce signal enables the stoppable clock, and the completion signal c indicates when synchronous computation is complete.

We specify the LCM behavior with signal transition graphs (STGs) [4] that describe the ordering of internal signals. Figure 2 shows the STGs for the two modules that control communication. The dashed line marks a portion that is expanded below.

![Figure 2. STGs: (a) control logic; (b) clock enable](image)

3. Verification of the LCM with the CWB

The Concurrency Workbench is based on Milner's CCS process algebra [9], in which a set of agents can be composed to execute concurrently, each engaging in its own actions and synchronizing with another agent when they are bound together by an input and an output action with the same name. The CWB permits the simulation of a system composed of a set of agents, both step-by-step and exhaustively if the state-space is finite. It also allows proofs of observational equivalence of two systems of agents. It includes a branching-time temporal logic model-checker for proofs of temporal assertions about agents [12], such as that they never deadlock, or always become ready to accept input again. The design of the CWB and the algorithms it uses for model-checking are well-described in [5].

Table 1 gives some examples of CWB processes; actions a and 'a are complementary; if one is input, the other is output. The agent (Y1Z)m.id in the table is observationally equal to agent X because the m.id actions are bound together and hidden. Agents P, Q are not equal, however, although they have the same visible traces. An observer can detect that Q can engage in action a and then be unable to engage in c, because it chose the first alternative, while after agent P engages in a, it can always do either b or c.

We model an STG in the CWB with an agent for each node. Edges are represented by complementary actions of the two nodes that they join. Each agent also has an action representing the transition itself. For example, the oa – graph node becomes OAL = odm.oal.'oam.OAL.

We adopt the convention that ' denotes an output action, and that p.m suffixes label the actions representing edges (denoting plus, minus transitions) and h, l label the actions representing the high, low transitions themselves. A complete STG is the parallel composition of all the node-agents, together with fork and join agents where appropriate, with all the edge-actions restricted so that they are invisible. Figure 3 illustrates the CWB description of the indicated section of the STG in Figure 2. We comment later on how to represent the relationship between pairs of STG nodes representing high and low transitions of the same signal.

\[
\begin{align*}
\text{OAL} &= \text{odm.oal.'oam.OAL} \\
\text{IAH} &= \text{irp.oam.IAHfork + oam.irp.IAHfork} \\
\text{IAHfork} &= \text{iah.'iap1.'iap2.IAH + 'iap2.'iap1.IAH} \\
\text{SRH} &= \text{iap1.'zrh.'irp.SRH} \\
\text{STGsegment} &= \{\text{OAL, IAH, SRH}\} \setminus \{\text{oam, iap1}\}
\end{align*}
\]

![Figure 3. CWB description of STG segment](image)

The complete CWB model of the specification of the LCM is the composition of three STG agents, for the control logic + environment, for the clock-enable generator, and a simple one for the synchronous core. The last abstracts away the computational detail and models only the clock-enable and completion signals. The CWB will show all possible traces of this LCM specification. There are nine possible initial prefixes that repeat, such as:

\[
\text{ih'} \text{iah'} \text{irl'} \text{odh oah'} \text{ial irh'} \text{odl oal'} \text{iah'} \text{ih'} \text{iah'} \text{odh oah'} \text{ial irh'} \text{odl oal'} \text{iah'}
\]

In this case there are few enough traces that one could see by inspection that the behavior is correct, but in general we would need to prove temporal properties satisfied by all possible traces. The CWB allows the definition of properties in the modal mu-calculus [12] such as Always, and Eventually, with which one can assert liveness. Always Eventually irh is true of our LCM agent, meaning that it will always accept input again. These temporal operators are generally useful; we have also defined temporal operators specifically for the analysis of LCMs, such as Can'tBefore a: b: action a can't occur before b. It is also possible to prove freedom from deadlock; in fact this is a built in CWB command.

3.1. Implementation

We focus on the implementation of the asynchronous clock-enable generator and basic control logic circuits given in [13].

To prove the implementation correct we model logic gates in the CWB in a straightforward way. A wire w is an
agent with actions wh, wl representing its high and low values. A gate is a collection of agents representing the states that it may be in as a result of receiving an input value or generating a new output. An inverter with input values ah, al and initially high output is

\[
\text{Not} = 'h'.\text{Not}h \\
\text{Not}h = \text{ah}. '1'.\text{Not}l + \text{al}.\text{Not}h \\
\text{Not}l = \text{ah}.\text{Not}l + \text{al}. 'h'.\text{Not}h
\]

After generating the initial output, Not becomes an agent absorbing low inputs silently and generating a low output when the input goes high, with a transition to the complementary state. The other gates are similar. Fanout must be represented explicitly with For\(k\) agents that duplicate a high or low input on two outputs. It is important to note that all the components behave asynchronously; that is, there are no assumptions about delays in gates, forks, or wires. This is a minimal condition, and a circuit that behaves correctly under it will be delay-insensitive.

To verify the LCM control-logic circuit we model the circuit from [13] in the CWB, and construct a minimal environment, constrained only to perform the 4-cycle handshake on input and output sides independently: \(\text{ENV} = (\text{IN} | \text{OUT})\), where

\[
\text{IN} = 'irh'.\text{iah}. 'irl'.\text{ial}.\text{IN} \\
\text{OUT} = odh. 'oah'.odl. 'oal'.\text{OUT}
\]

We compose the environment and control-logic agents with the STG for the clock-enable generator and the synchronous core agent to obtain a partially circuit-level LCM. We simulate this agent to observe its behavior, useful for debugging, and more important, we prove it observationally equal to the LCM with the control-logic STG specification.

The next step is to show that when the STG for the clock-enable circuit is replaced by its gate-level implementation, the resulting LCM is again equal to the STG specification. There is a new difficulty here: the clock-enable circuit is not delay-insensitive, and has a constraint on internal propagation delay, which must be less than the time between high and low transitions on the completion signal from the synchronous core. We have simulated this circuit with an additional agent to show when propagation through gates has occurred, and have demonstrated that the behavior is correct when the delay constraint is met. Further work at this level of modeling requires the use of synchronous rather than asynchronous combinations of the CWB agents.*

### 4. Refinements

The STG specification of the LCM in Figure 2 permits relatively little concurrency, as shown by its small set of traces. The CWB has helped to develop two other STGs that specify the same top-level behavior of the control logic but greater concurrency. Each STG has been modeled in the CWB, has been proved to satisfy the appropriate liveness and safety properties and the basic control logic circuit for each has been proved equivalent to its STG specification, as before. Figure 4 has one such improved STG.

![Figure 4: An improved STG specifying the LCM](image)

An attempt further to relax constraints in the STG failed when we overlooked one of the conditions given in [4] for an STG to be speed-independent. The CWB showed the error by proving that the implementation of this version was not equivalent to its specification.

Our CWB model of STGs uses a simplification that needs justification. It does not model the possibility that a second transition on a signal might occur so quickly that

* The present CWB has flaws in its synchronous combinator, and this work awaits release of the next version.
the first transition had not yet been "seen" by the transitions depending on it. Chu’s three constraints on STGs that are speed-independent forbid such events [4]. One states, for instance, that if a transition on signal S enables immediate successor transitions, the complementary transition on S must depend on all those successors, so that it cannot fire prematurely. Our simplification is appropriate if a graph meets these constraints. Chu’s constraints are stated in terms of graph properties (such as that all edges from a node lie on paths to the complementary nodes), and these properties can be stated in the temporal mu-calculus of the CWB. Since the constraints quantify over all paths or cycles in a graph at present the user must identify the paths or cycles and make the CWB assertions for each. We have developed prototype preprocessors to automate the job by accepting STG descriptions in a simpler notation that is expanded into the CWB model, with all appropriate temporal assertions generated for testing.

5. Results, conclusions and further work

The Concurrency Workbench provides a way to write executable specifications in the CCS process algebra, and the ability to prove important properties of systems modeled in it. It offers easy experimentation with designs, and its simulations and proof attempts often catch errors early. We have succeeded in proving that circuit-level implementations meet specifications expressed with STGs and with temporal assertions. The latter are frequently a natural way to specify system requirements such as deadlock-freedom. Specifically, by means of elementary models of logic gates and representations of STGs in the CWB, we have proved that the STG specification of a locally clocked system is equivalent to its gate-level implementation. We have also proved that a less-constrained specification satisfies the safety and liveness properties that motivated the original one, and that its implementation is also equivalent to it.

The CWB combines capabilities offered by related tools: SMV is a temporal logic model checker that lacks the CWB’s simulation capacity [8], while Cercal can prove equivalences but does not support temporal logic [7]. Other approaches to the design and verification of asynchronous circuits use theorem-proving rather than model-checking, such as Synchronized Transitions, which combines a UNITY-like model of parallel computation [2] with the Larch Prover [11]. Still others, such as Martin’s CSP [6] and Tangram [1], apply transformational techniques to compile high-level specifications manually or automatically into circuits.

The present form of the CWB has some weaknesses. It doesn’t permit the definition of parameterized agents that can be multiply instantiated, so that lots of symbol-pushing is required. We have partially solved this problem with preprocessors, and the forthcoming version of the CWB supports parameterization. Since our CWB models are low-level, a component may have many states. The CWB can minimize systems, but this strategy doesn’t always lead to large state-space reduction until all components are connected to constrain one another.

On balance, the CWB is a useful tool. We are extending its use to larger designs, such as systems of interconnected LCMs, and are extending the preprocessors for automatic generation of assertions about structural properties of STGs. We are also developing systematic models of gate-level systems with different assumptions about synchrony, so that we can use either asynchronous or synchronous models as appropriate.

References