12 Arbiter

This arbiter is a good example for a synchronous scalable state machine.

12.1 Introduction

The purpose of the bus arbiter is to grant access on each clock cycle to a single client among a number of clients contending for the use of a bus (or another resource). The inputs to the circuit are a set of request signals $req_0, \ldots, req_{k-1}$ and the outputs are a set of acknowledge signals $ack_0, \ldots, ack_{k-1}$ (Fig. 12-1). Normally the arbiter asserts the acknowledge signal of the requesting client with the lowest index. However, as requests become more frequent, the arbiter is designed to fall back on a round robin scheme, so that every requester is eventually acknowledged. This is done by circulating a token in a ring of arbiter cells, with one cell per client. The token moves once every clock cycle. If a given client's request persists for the time it takes for the token to make a complete circuit, that client is granted immediate access to the bus.

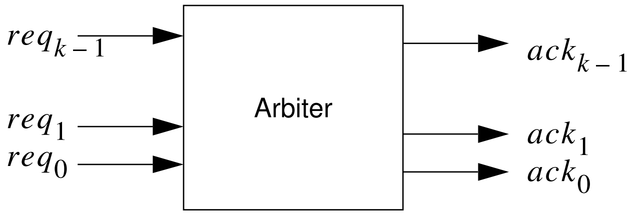

Figure 12-1: Black box view of the arbiter

12.2 Specification

The desired properties to be verified are:

1. No two acknowledge outputs are asserted simultaneously.
2. Every persistent request is eventually acknowledged.
3. Acknowledge is not asserted without request.

By restricting *N* to be two (or some other small constant), the problem becomes simple yet illustrative. Ultimately, it should be possible to verify a design where *N* is a parameter, i.e. do the verification for an arbitrary *N*.

12.2.1 Formal specification example

Data path free control circuits may be described easily using propositional temporal logics like CTL [ClEm81]. The above properties result in the following CTL expressions:

1. $\bigwedge_{i \neq j} \mathrm{AG}\neg(ack_i \wedge ack_j)$
2. $\bigwedge_i \mathrm{AGAF}(req_i \rightarrow ack_i)$
3. $\bigwedge_i \mathrm{AG}(ack_i \rightarrow req_i)$

12.3 Implementation

The basic cell of the arbiter is shown in Fig. 12-2. This cell is repeated k times, as shown in Fig. 12-3 for $k = 4$. Each cell has a request input and an acknowledge output. The grant of cell i is passed to cell $i+1$, and indicates that no client of index less than or equal to i are requesting. Hence a cell may assert its acknowledge output if its grant input is asserted. Each cell has a register T which stores a one when the token is present. The T registers form a circular shift register which shifts up one place each clock cycle. Each cell also has a register W (for "waiting") which is set to one when the request input is asserted and the token is present. The register remains set while the request persists, until the token returns. At this time, the cell's override and acknowledge outputs are asserted. The override signal propagates through the cells below, negating the grant input of cell 0, and thus preventing any other cells from acknowledging at the same time.

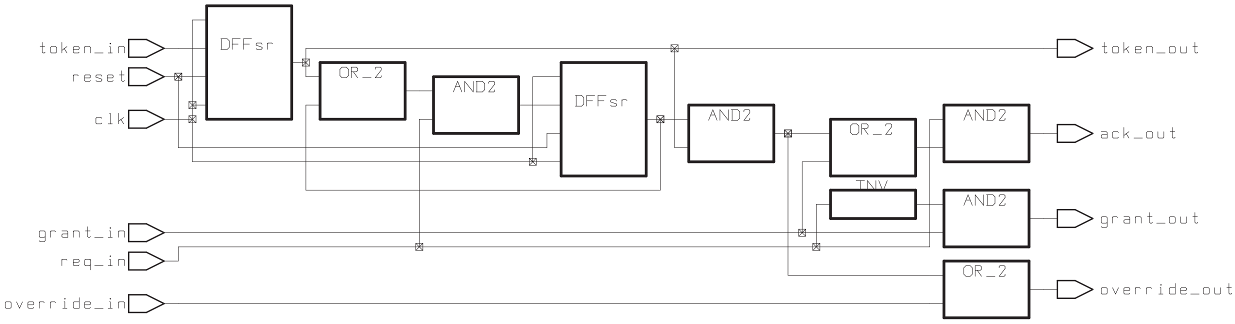

Figure 12-2: Cell_2_plus

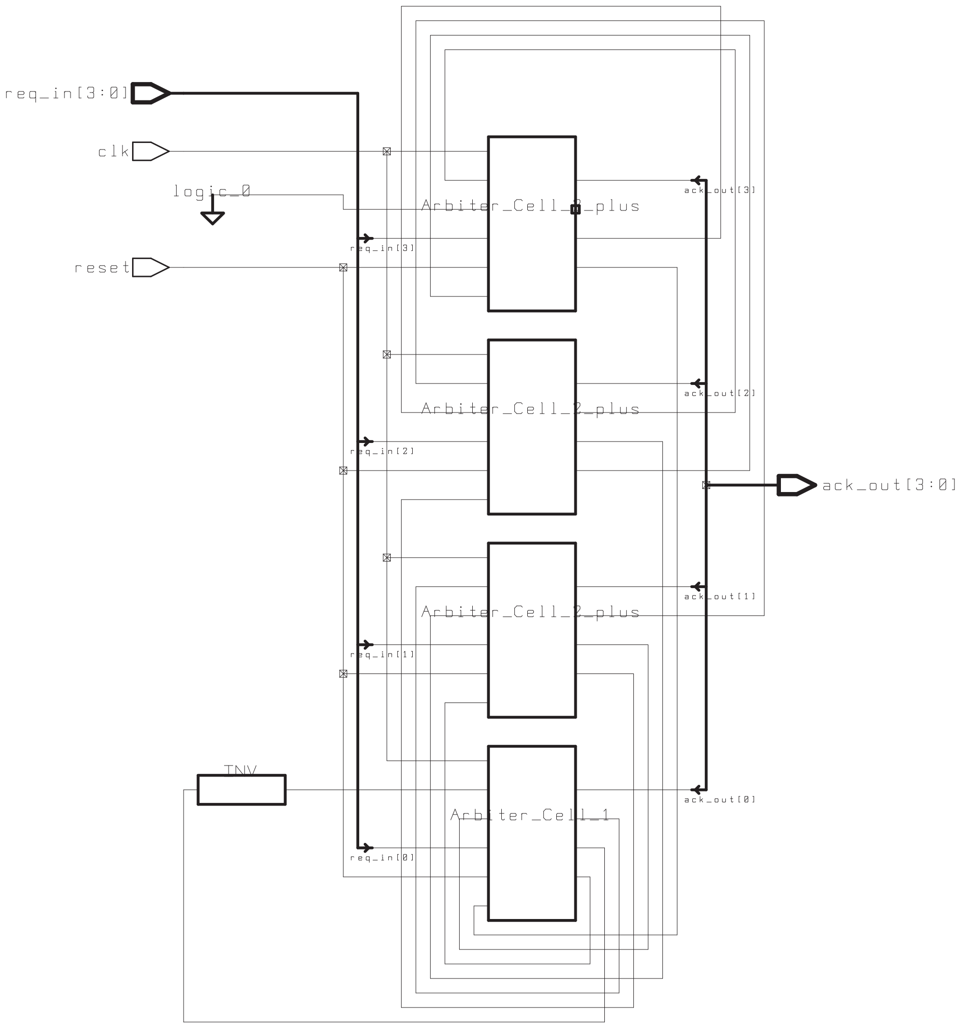

Figure 12-3: Arbiter with four inputs/outputs

The circuit is initialized so that all of the W registers are reset and exactly one T register is set. To achieve this one cell has a different implementation as shown in Fig. 12-4.

Fig. 12-5 and Fig. 12-6 show different simulation runs with and without request collisions.

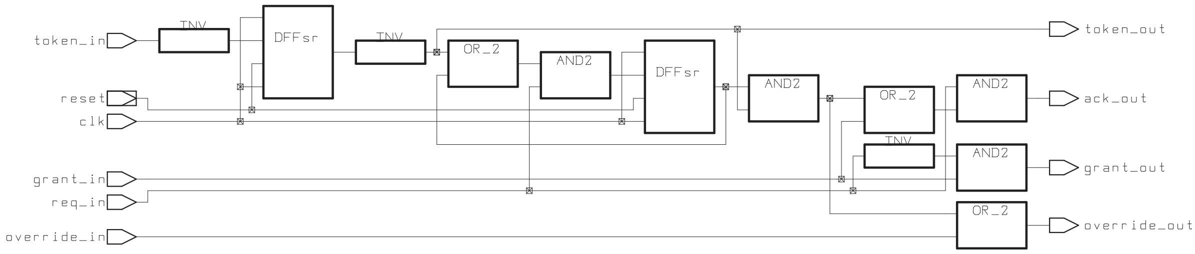

Figure 12-4: Cell_1

12.4 Status and Acknowledgments

The circuit has been originally proposed by J. Staunstrup [Stau93]. The implementation and formal specification has been taken from [McMi93a, p. 40ff.]. Parts of the description are directly quoted from there. Thanks to C.J. Thomas for providing the VHDL descriptions.

12.5 Literature

[ClEm81] E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, Yorktown Heights, New York, May 1981. Springer-Verlag.

[McMi93a] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[Stau93] J. Staunstrup. IFIP WG 10.2 Collection of Circuit Verification Examples, November 1993.

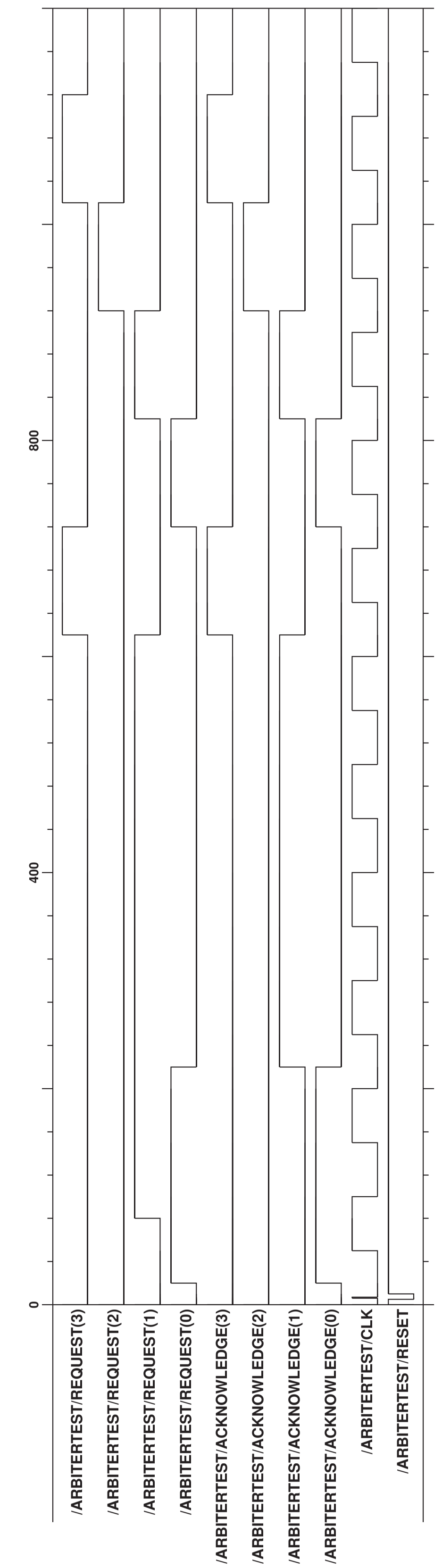

Figure 12-5: Example simulation with collision

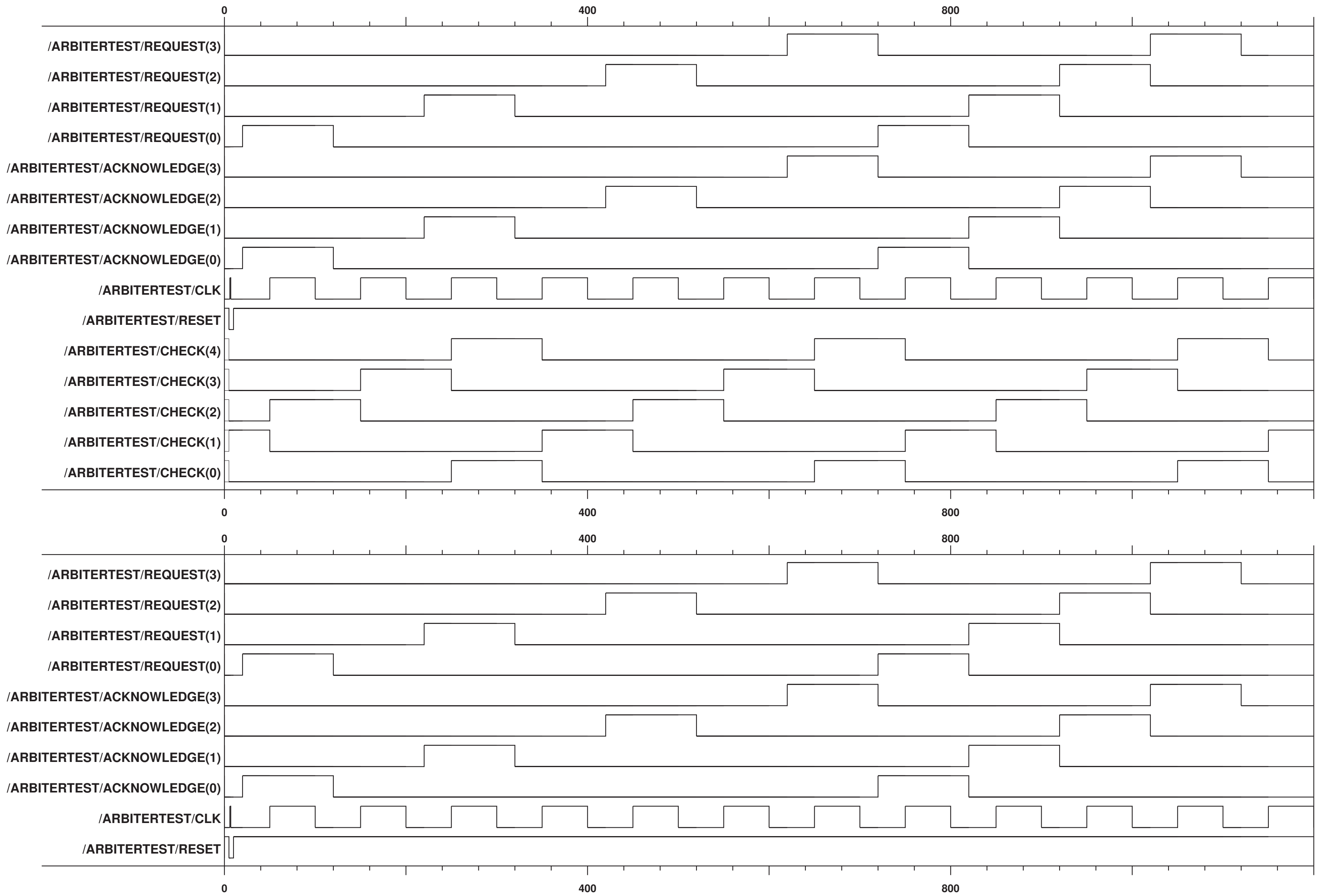

Figure 12-6: Example simulation without collision