VMEbus Controller Synthesis by Communicating Asynchronous Sequential Circuits

Eisele, W.; Eckstein, G.; Beister, J.
Department of Electrical Engineering
Kaiserslautern University, D-67653 Kaiserslautern
E-Mail: eckstein@rhrk.uni-kl.de

December 20, 1994

Abstract

This paper presents the systematic synthesis of a fairly complex digital circuit and its CPLD implementation as an assemblage of communicating asynchronous sequential circuits. The example, a VMEbus controller, was chosen because it has to control concurrent processes and to arbitrate conflicting requests.

Keywords: *asynchronous circuits, VMEbus, bus controller, CPLD, Petri nets*

1 Introduction

The aim of this project was to investigate

- whether the tasks of a fairly complex controller, distinctly more complex than the FIFO controller presented in [1], can be solved with communicating asynchronous sequential circuits,
- whether our synthesis method [1] can be applied to industrial-sized products,
- how to obtain a complex, exact and understandable high level specification.

A VMEbus controller was chosen as the example because of

- its practical importance,
- its relative complexity (25 input and output signals)
- concurrency and conflicts of the processes to be controlled,
- and their essentially asynchronous nature.

Design and implementation include

- the construction of a formal Petri net specification of the controller's behavior across the interface to its environment.
- the extraction (from this formal specification) and construction of a set of finite-state machines, and
- their implementation as an assemblage of communicating asynchronous sequential circuits.

Problems and results will be discussed.

The VMEbus [8] is a multiprocessing bus and is frequently used in the area of computer-supported process control. Besides master and units and slave units, some combined master-slave units may be connected to the bus (Fig. 1). The latter type contains a slave module (e.g. memory) that can be shared between the local master and the other VME masters. The controller that is the subject of this paper deals with internal and external requests for access to the shared slave. Fig.1 shows this VMEbus controller in a typical environment. It is connected to the local processor, the shared slave unit, and the VMEbus, and it takes its orders from a requester. By decoding addresses on both the local bus and the VMEbus, the requester can generate three request signals for

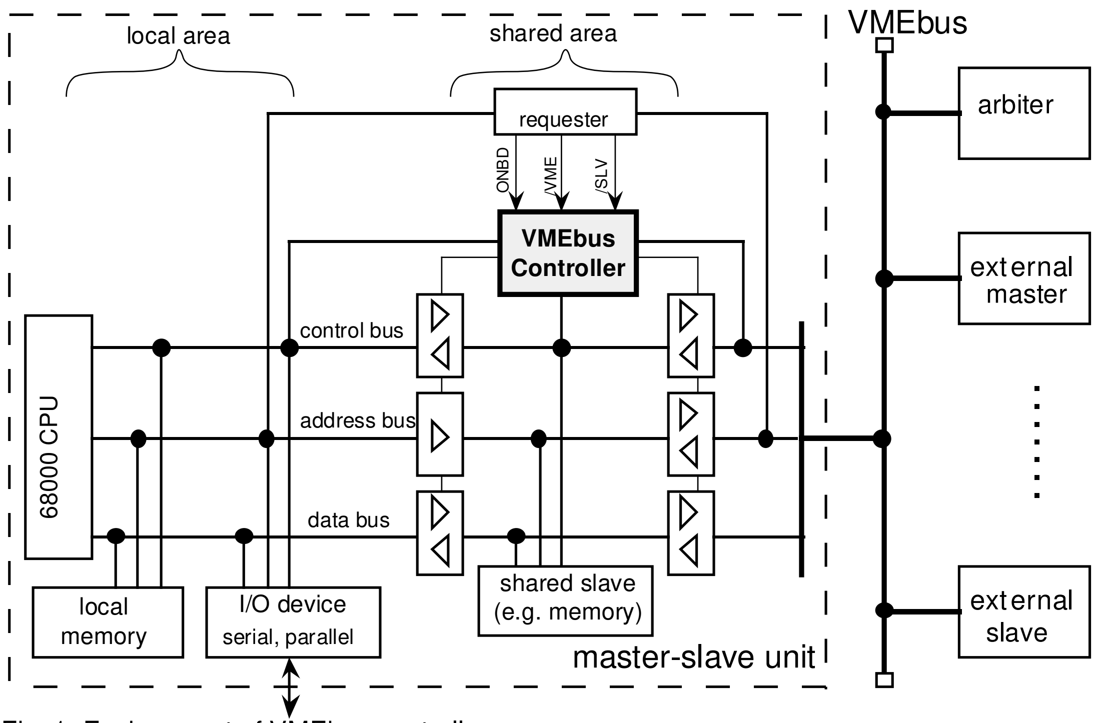

Fig. 1: Environment of VMEbus controller

- an onboard request (the local processor demands access to shared memory),
- a VME request (the local processor wants access to an external slave via the VMEbus),
- and a slave request (an external master wants to access the shared slave).

The task of the VMEbus controller is to recognize and arbitrate the requests, to control the data transfer cycles accordingly, and to operate the bus transceivers.
The specification of this controller corresponds to the data sheet of the SCB 68172 VMEbus controller formerly manufactured by VALVO [9]. Fig. 2 shows a model.

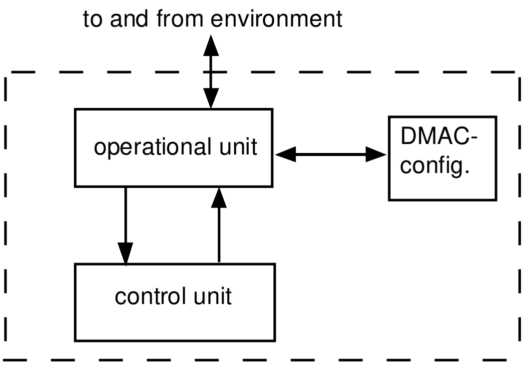

Fig. 2: VMEbus controller model

2 The design process

The general design method by which the controller was designed has already been presented in [1] and will only be summarized here:

- *Partitioning into operational and control unit.* (Fig. 2; the DMAC configuration unit is relevant only during the system initialization phase). The operational unit implements certain standard control operations and performs combinational modification of input variables from the environment in order to simplify the task of the control unit.
- *Obtaining an accurate formal specification of the set of possible processes at the interface of the control unit.* The specification is in terms of events (signal transitions) and their causal relationships; it takes into account the possible behavior of the environment and the required reaction of the control unit. The basis of specification is the signal transition Petri net (STPN; called STG in [1], but without certain restrictions imposed on STGs [3] [4]). Events internal to the control unit are not modelled, being a matter of design, not of specification. The resulting STPN is shown in Fig. 3.
- *Decomposition (by extraction) of the Petri net into sub-STPNs*, corresponding to the intended partitioning of the control unit into subunits.
- *A step graph is generated from each subnet:* The causal partial ordering of events generated by the Petri net is transformed into a linear quasi-ordering of the same events in time.
- *The primitive flow tables of interconnected finite state machines are constructed algorithmically from the step graphs.*
- *Asynchronous circuit design:* Enhancement of primitive flow tables through measures dealing with nonfundamental mode operation due to concurrent input changes; state reduction, and state assignment to avoid critical races.
- *RS-flip-flops as feedback elements:* two-level combinational S and R circuits with cross-coupled output gates according to LEWIN.

3 Some aspects of specification and design

The control unit in Fig. 2 has to govern rather complex processes. Hence the construction of the overall STPN as a whole would be very difficult. Therefore, the control unit was first decomposed into several small units (not those shown in Fig. 4), and their interaction with their respective environments modelled using STPNs. Next, these STPNs were succesively composed, yielding the overall STPN shown in Fig. 3. The signals with which the transitions are labelled are those occuring in Fig. 4. Starting from the initial marking, one can disitinguish four main processes in Fig. 3. From left to right:

- the slave-cycle (involving /SLVI↓-/VMEEN↓ •/MASTEN↑)
- the onboard cycle (involving ONBDI↑)
- the VME cycle (involving /VMEI↓-/BBSY↓-/VMEEN↓)
- the DMAC cycle (involving /LBR↓-/BBSY↓ followed by the VME cycle.

The control unit is at rest at the initial marking. It may receive either a single or two conflicting requests, since a slave request from without can occur concurrently with either a VME or an onboard request from within the master-slave unit. The control arbitrates the conflicts, always deciding in favor of the slave request in order to avoid deadlocks, and starts the proper cycle, which it then observes and controls. Delegating these complex and concurrent processes to a single sequential circuit, necessarily operating in non-fundamental mode, would be very unsatisfactory. Therefore, the control unit was partitioned intuitively into three components, SW1 to SW3, as shown in Fig. 4.

Two points of view governed the partitioning:

- Reduced input complexity for the components. SW3 owes its existence to this aspect: It handles the generation of acknowledge signals based on the cycle information given by SW1.
- Concurrency considerations. Processes on the VMEbus, including slave requests, usually take place concurrently with processes on the local bus. Therefore, SW2 was created to deal with the VMEbus.

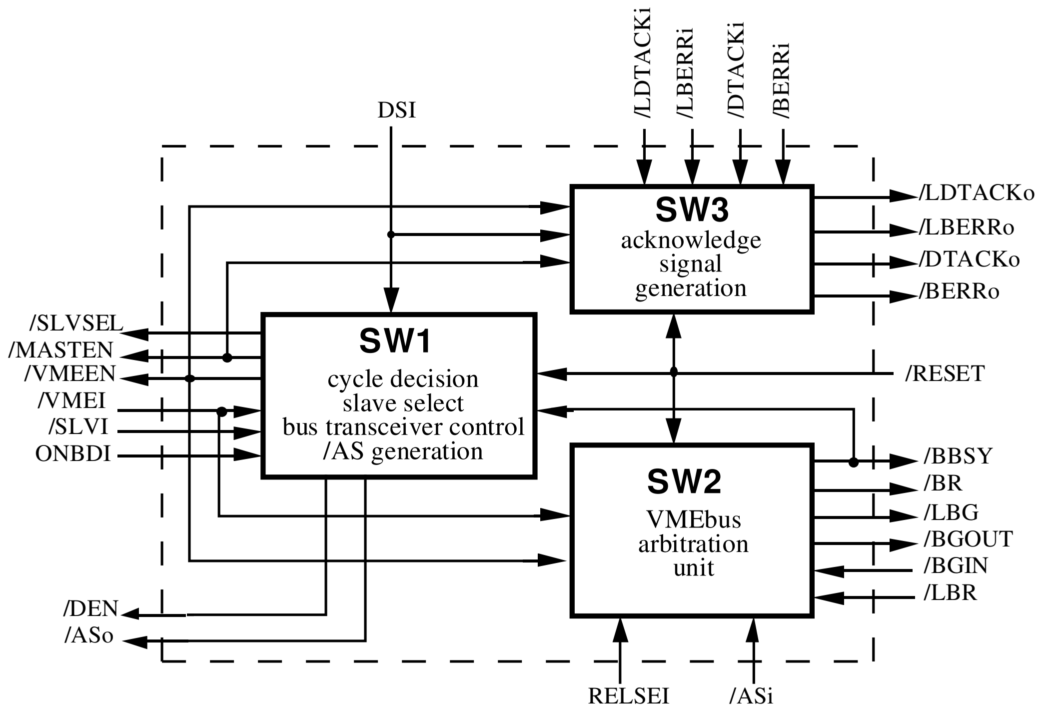

Fig.4: Components of the control unit

As a result, control tasks were assigned to the components as follows:

- SW1 recognizes and arbitrates requests, decides on what type of cycle to start, controls bus transceivers, passes on master signals (such as /AS), and controls slave selection;
- SW2 controls the VMEbus protocol (the master-slave unit's share of VMEbus arbitration);
- SW3 passes on slave acknowledge signals according to the cycle in progress

For each of the components thus created, its signal transition Petri net was extracted from the overall STPN of Fig. 3. Note that the component STPNs are not obtained by cuts; instead, they overlap and interleave to compose the overall net. The next steps were the generation first of step graphs, then of primitive flow tables; enhancement for non-fundamental-mode (NFM) operation, followed by internal state reduction. Liu-type state assignments were used, and R-S flip-flops chosen because each of the combinational R- and S-circuits needs fewer first-level gates than a minimum-hazard next-state circuit for direct feedback.

4 Realization in Complex PLDs

The main advantage of Complex PLDs (CPLDs) over FPGAs and even over certain ASICs is that they have sufficiently many, sufficiently wide programmable gates [2]. Only two logic levels were needed for the combinational circuits. Therefore, circuit delays are short, and delay differences that might lead to spurious pulses (due to the remaining hazards and NFM operation) are small. Some interface signals, such as /AS, alternate between being input and output signals of the controller. For designing and implementing the control unit, they were split into two distinct signals, distinguished by suffixes "i" and "o" (Figs. 3 and 4). Using bidirectional I/O cells of the CPLD, the operational unit splits the signals, remerges them and adapts them to the VMEbus. It turned out that the entire controller - control unit (3 components), operational unit and DMAC configuration unit (Figs. 2, 4) - partitioned as given in the table below, could be realized on two Lattice ispLSI1016 CPLDs, using 20% of the available product terms on each.

seq. circuit	input var.	output var.	state var. in total	state var. that are output var. too	internal states	in CPLD No.
SW1	5	5	4	2	7	1
SW2	6	4	5	3	8	2
SW3	7	4	1	0	2	1
DMAC conf.	2	1	2	1	4	2
operat. unit	13+1+16	12+2+16	0	0	1	1+2

The reason for partitioning the design was just that a single 1016 CPLD would have too few output enable terms. It would even be possible to fit the controller onto a single larger CPLD, such as an ispLSI1024; however, the necessary development tools were not available to us. Without the DMAC configuration unit, the controller would fit into a single 1016 CPLD.

5 Conclusions

The implemented controller was put to a series of tests, some designed to produce conflicting requests with variable skew. The logical design proved to be correct by construction (one of the goals of the project). An intermittent failure caused by too long a rise-time was corrected by inserting a driver. Long-term tests designed to produce conflicting requests were passed without failure. By design and implementation, our circuit reacts very fast and needs fewer gates than an equivalent clocked controller (no clock distribution networks, no clocked flip-flops; some additional expense for hazard prevention). We have compared the speeds of the commercial VMEbus controller SCB68172 (8MHz) and our implementation. Some output signals of the SCB are not clocked. Measurements showed that both circuits have similar gate delays. In operation, the observed speed ratios depend upon the cycle type and range from 0.85 (in favor of the SCB) to 24 (in favor of our circuit). On the average, the asynchronous implementation presented here is about 8 times faster than the SCB. If an asynchronous controller is to be used, one necessary requirement is an environment that produces signals free of spurious pulses. Furthermore, if there is input concurrency, then the controller must be designed to cope with NFM operation wherever possible. If this is done, as in the controller presented here, its state transitions will be very safe. A different problem arises if NFM inputs to one component result in NFM inputs to another because of a first begun and then aborted response of the first component. In contrast to the previous effects, this problem cannot be recognized directly from the STPNs. It requires the designer's skill to find out where it can occur, and to deal individually with each occurence. All in all, we believe that the design method and the chosen implementation result in fast assemblages of communicating circuits that can provide satisfactory solutions to problems of controlling concurrent processes.

References

[1] Beister, J.; Wollowski, R.: "Controller Implementation by Communicating Asynchronous Sequential Circuits Generated from a Petri Net Specification of Required Behavior". In: "Synthesis for Control Dominated Circuits", G. Saucier and J. Trilhe, eds.; Amsterdam: Elsevier 1993, pp. 103-115

[2] Beister, J.; Kuhn, M.; Wollowski, R.: "An asynchronous controller for a daisy-chainable VMEbus interrupter". 3rd International Workshop on FieldProgrammable Logic and Applications, Oxford 1993.

[3] Chu, T. A.: "Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications". Phd thesis, MIT, June 1993.

[4] Lavagno, L.; Keutzer, K.; Sangiovanni-Vincentelli, A.: "Algorithms for Synthesis of Hazard-free Asynchronous Circuits". In: "Proceedings of the Design Automation Conference", pp. 302-308, June 1991.

[5] Eckstein, G.: "NFM-gerechte Codierung und Realisierung der Teilautomaten eines ungetakteten Schaltwerksverbunds am Beispiel eines VME-Bus-Controllers". Diploma thesis, Dept. of Elec. Engineering, University of Kaiserslautern, Germany 1994.

[6] Eckstein, G.: "BUSCONV2: Ergebnisse der Aenderungen am Kommunikationsprotokoll des ungetakteten VME-Bus-Controllers". Tech. Report, Dept of Elec. Engineering, University of Kaiserslautern, Germany 1994.

[7] Lattice:"plsi and ispLSI Data Book", Hillsboro, USA 1992.

[8] VMEbus Specification, ANSI IEEE STD1014-87, VITA. 1987.

[9] Philips, Valvo: VMEbus Controller (BUSCON) SCB68172 Data sheet, December 1986.

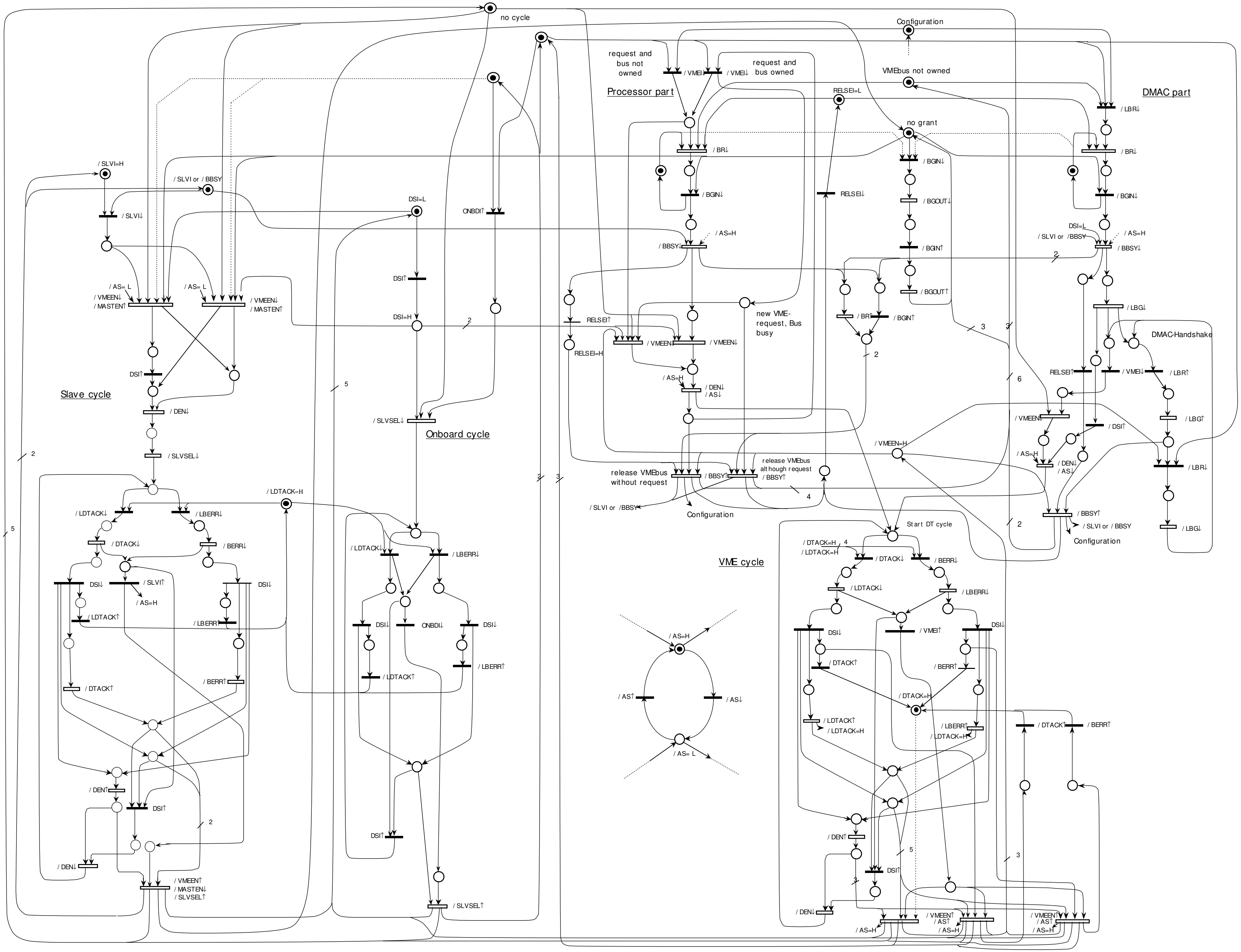

Fig. 3: Signal Transition Petri Net of the whole control unit