# *Prototyping the TI TMS320C40 to the Cypress VIC068/VAC068 Interface*

## *Application Report*

***Peter F. Siy and David L. Merriman***
***The MITRE Corporation***
***Timothy V. Blanchard***
***Cypress Semiconductor***

SPRA105
February 1994

## IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

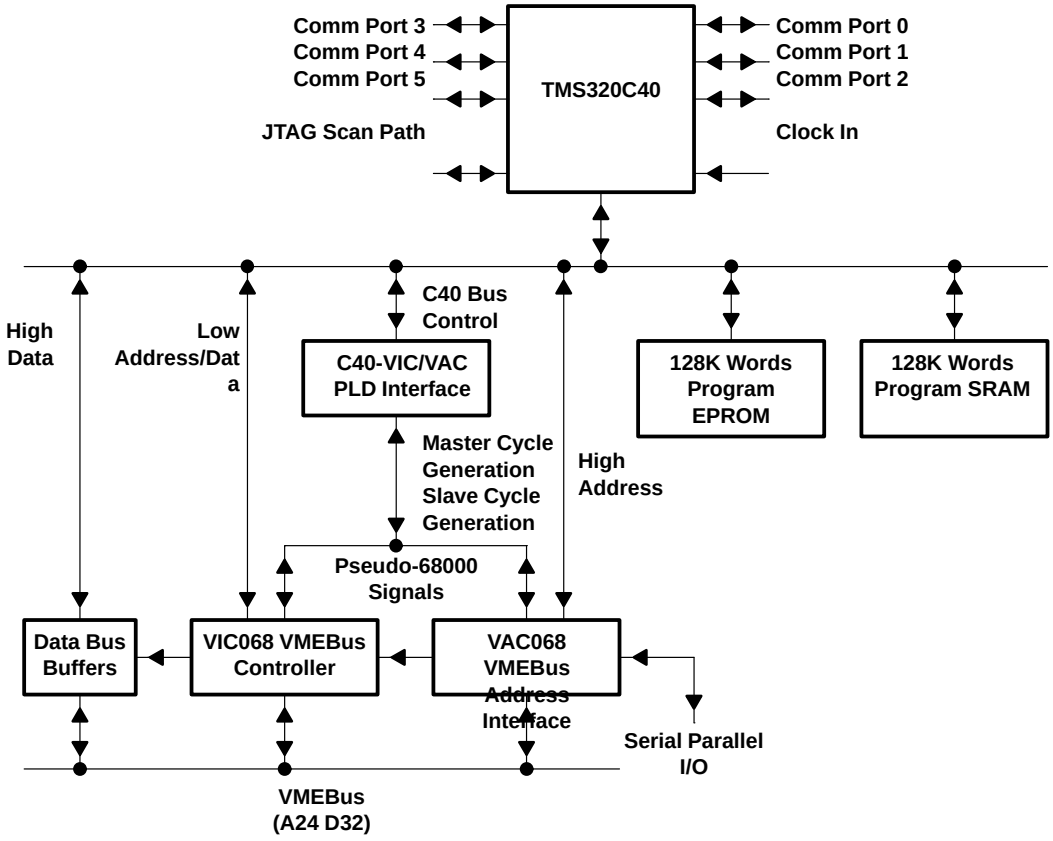
TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

## Introduction

The Texas Instruments TMS320C40 digital signal processor (DSP) represents a state-of-the-art solution to many signal processing problems via its high-speed central processor unit (CPU), unique parallel processing I/O capability, and robust interface to other system components [1]. Likewise, the Cypress Semiconductor VIC068 VMEbus Interface Controller and its companion VAC068 VMEbus Address Controller provide a complete VMEbus interface, including master and slave capability [2,3]. Since these components are effective in a wide variety of applications and since the VIC068/VAC068 is a single- or multiple-TMS320C40 VMEbus card design, we developed a single TMS320C40 VMEbus card for use in a satellite modem application [4].

The VIC068/VAC068 is designed to interface with the Motorola 68000 family of microprocessors, so it was determined that the interfacing to a TMS320C40 required a logic simulation or some form of programmable, configurable prototype. When this design was initiated, only preliminary documentation existed for the VMEbus chip set, and no simulation models were available for either the chip set or the TMS320C40 DSP. Therefore, we first prototyped the interface of these components in a wire-wrap environment before proceeding to a printed circuit board design. This paper provides the high-level as well as low-level details of the prototyping effort so that others may examine our approach and techniques to minimize design time for subsequent efforts. Note that this design has not been optimized for either size or speed. Section 2 outlines the design goals established before design began and gives relevant background regarding the devices involved. The paper focuses on the hardware details, programmable logic source code, and schematics that follow. In addition, the software initialization of the chip set by the ’C40 is described. Throughout this paper, we assume that you are familiar with the ’C40 architecture as well as with the VMEbus and its protocol(s). You can refer to [5,6] for more details on the VMEbus. Figure 1 shows the VIC068/VAC068 prototype block diagram.

**Figure 1. TMS320 – VIC/VAC Prototype Block Diagram**



## Prototype Design

### Design Goals

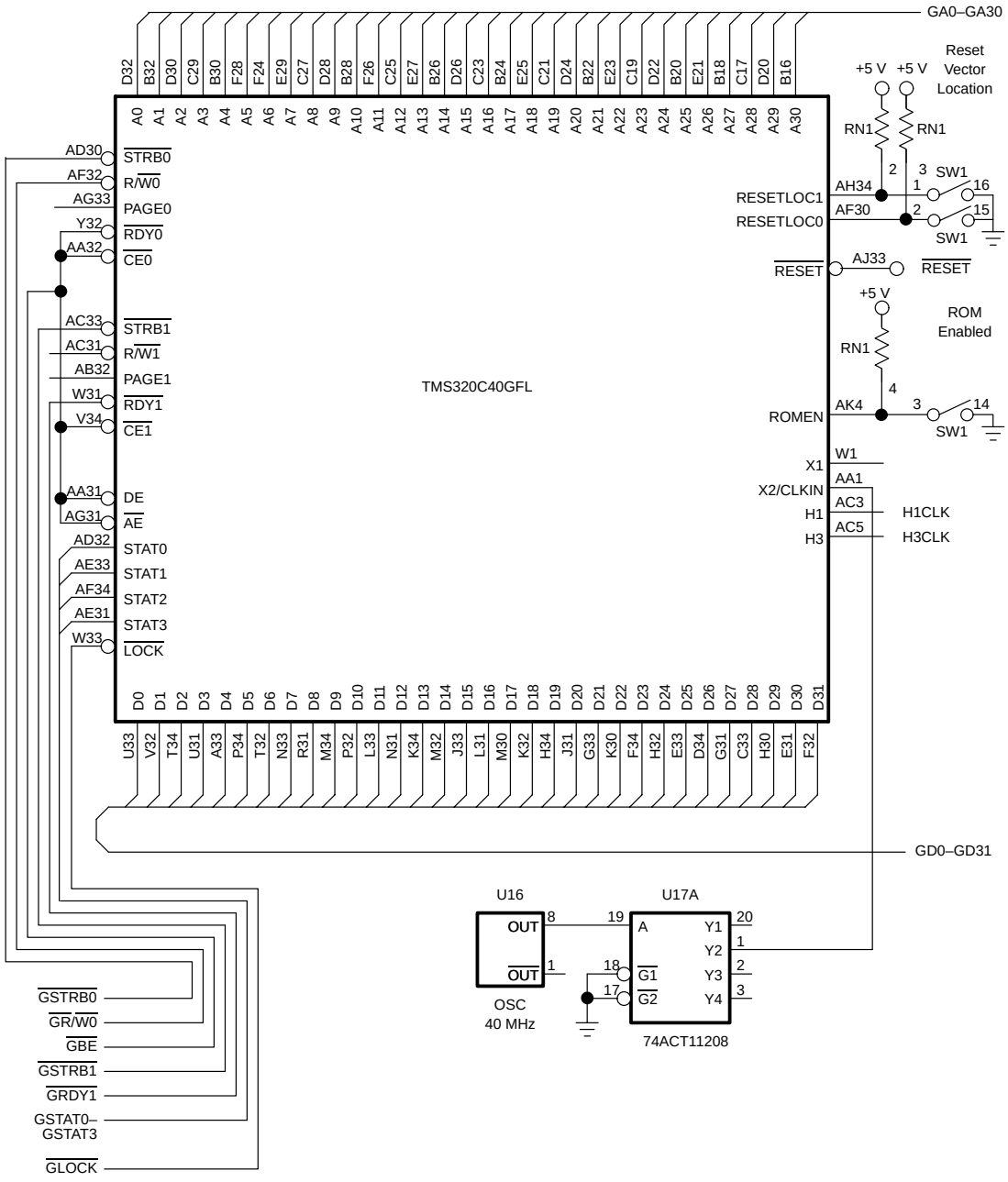
We began by developing a set of design goals for the VME interface that were based on our particular needs. We were interested in a ’C40 card that provided both (VMEbus) master and slave capability for reads, writes, read-modify-writes, write posting, and slave block transfers. We designed the address/data capability according to the most prevalent configuration (for other cards available commercially): 24-bit address and 32-bit data (i.e., A24/D32); however, the design presented here does not preclude 32-bit addressing with minor modifications. Via the VIC068, this design also features system controller capability. We did not incorporate VMEbus interrupt support, because we provided application-specific interrupt inputs directly to the ’C40. We utilized the VAC068 for address control/mapping of the two Universal Asynchronous Receiver/Transmitters (UARTs) that were required for our application, and for general-purpose parallel I/O. The new Cypress CY6C964 Bus Interface Logic Circuit can be used instead of the VAC068. In addition to the VMEbus functionality, we required the interface to be compatible with both the existing ’C40-40 (50-nanosecond cycle time) and the faster ’C40 (40-nanosecond).

### Design Considerations

First, we thoroughly examined the VIC068 and VAC068 and reviewed the 680x0 family bus signals and cycles. In particular, we referred to the 68020 user’s manual [7] extensively. The VIC068 and VAC068 interface directly to the Motorola 680x0 family data, address, and control signals and are driven with the familiar 680x0 address and data strobes ($\overline{\text{PAS}}$, $\overline{\text{DS}}$). An asynchronous transfer protocol is implemented via data transfer and size acknowledgment signals $\overline{\text{DSACK0}}$ and $\overline{\text{DSACK1}}$. In addition to these signals, the transfer size signals SIZ0 and SIZ1 are essential elements in the 680x0’s dynamic bus sizing capability and, with the lower address lines, encode the size of the transfer in progress. Also, during the transfer, the function code signals (FC0–FC2) provide information of importance in multiuser environments. Bus arbitration is an integral part of the 680x0 via the bus request ($\overline{\text{BR}}$), bus grant ($\overline{\text{BG}}$), and bus grant acknowledge ($\overline{\text{BGACK}}$) signals and is used directly by the VIC068. Finally, as in many other general-purpose microprocessors, bus cycles for the 680x0 are several clock cycles long.

While the VIC068 and VAC068 are driven by (and can drive) the familiar 680x0 bus signals, the ’C40 bus signals show little similarity to those of the 680x0 family. The ’C40’s bus protocol is common to the TMS320 floating- point DSP product line. An external ready signal allows for wait-state generation and controls the rate of transfer in a synchronous fashion (i.e., cycles can be extended an integer number of clock cycles). As described in [1], the ’C40 has two identical external interfaces: the global memory interface and the local memory interface. Each consists of a 32-bit data bus, a 31-bit address bus, and two sets of control signals. The benchmark of all DSP technology, the ’C40 executes single-cycle instructions (see the ’C4x user’s guide for complete details) and relies on a multistage pipeline for execution speed. Detailed bus status, including type of instruction and type of access, is given for each cycle via the STAT lines. Individual control lines can put the address, data, and control bus(es) in the high-impedance state. There is no read-modify-write signal (as on the 680x0); however, an instruction-driven $\overline{\text{LOCK}}$ signal is available. Each cycle is controlled by a strobe ($\overline{\text{STRB}}$x) signal in conjunction with the corresponding read/write (R/$\overline{\text{W}}$x) strobe. One of the ’C40’s notable features is its range of configuration options. The ’C40 has evolved from its earlier floating-point counterparts into a truly flexible interface via the local and global bus configuration control registers.

**Figure 2. TMS320C40 – VIC/VAC Prototype ’C40 Global Bus**

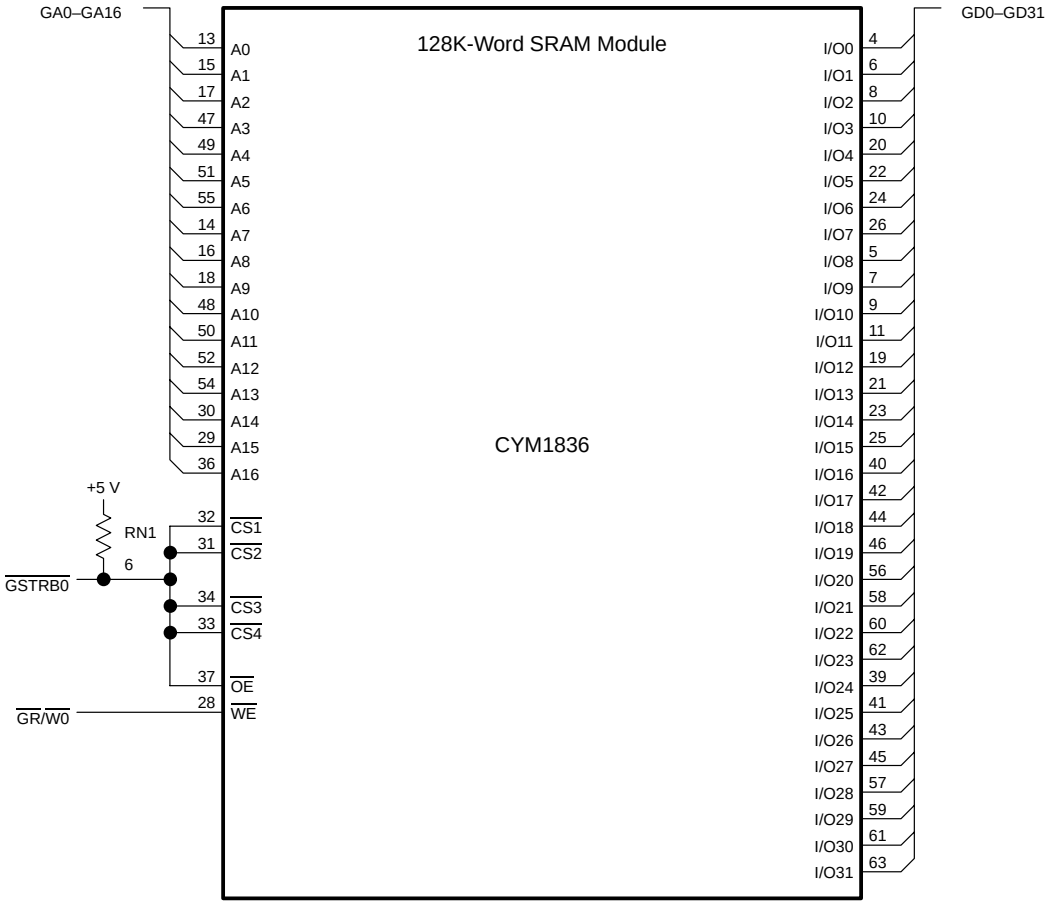


## High-Level Architecture

The high-level architecture for the card places fast 20-ns high-density 4-megabit (128K × 32) Cypress CMOS SRAM modules on both local and global buses of the ’C40 (the size of the memory array does not impact the ’C40-to-VIC068/VAC068 interface design). We designated the global side as program memory and the local side as data memory for our application. In our environment, it is anticipated that the local memory will be fully occupied by DMA coprocessor activity coupled with data fetches during communications-oriented DSP operations. Given this, we chose to place the A24 VME spectrum on the global (program) side, segmenting the local side I/O activity (the critical path for our application) from all VMEbus activity. (However, the interface documented herein can be used on either side because the global and local buses are symmetric.) On the global side, in addition to program SRAM, we also placed two 128K × 16 EPROMs for embedded program store, using the boot load feature of the ’C40.

Because we limited our design to VMEbus A24 addressing, this range fits nicely anywhere in the ’C40 global side address spectrum, from 08000 0000h to 0FFFF FFFFh. Therefore, VMEbus master access is memory mapped into the ’C40 global side address range. When an access occurs in this predefined A24 range, the ’C40 bus signals are transformed into 680x0 bus signals, which drive the VIC068/VAC068 pair and initiate a VMEbus transfer. Global side accesses outside of this range do not generate such signals and occur at full speed (i.e., the speed appropriate for that memory or peripheral). Regarding the “endianess” [8,9] of the interface, we know that the 680x0 family maintains big-endian byte ordering (byte-addressable memory organization) with little-endian bit ordering in each addressable unit. In contrast, the ’C40 is flat in its byte endianess (32-bit word addressable only) and little-endian bit ordered. Therefore, no swapping is done on the interface, because 32-bit word transfers (between processors) maintain D0 as the least significant bit. This forces a tradeoff of transfer speed for a wider range of transfers (byte, word, and three byte) than the ’C40 offers. We chose to limit our transfers to D32. To make transfers of all sizes available, you must preform additional setup and/or decoding before/during the transfer in progress.

**Figure 3. TMS320C40 – VIC/VAC Prototype Program SRAM**

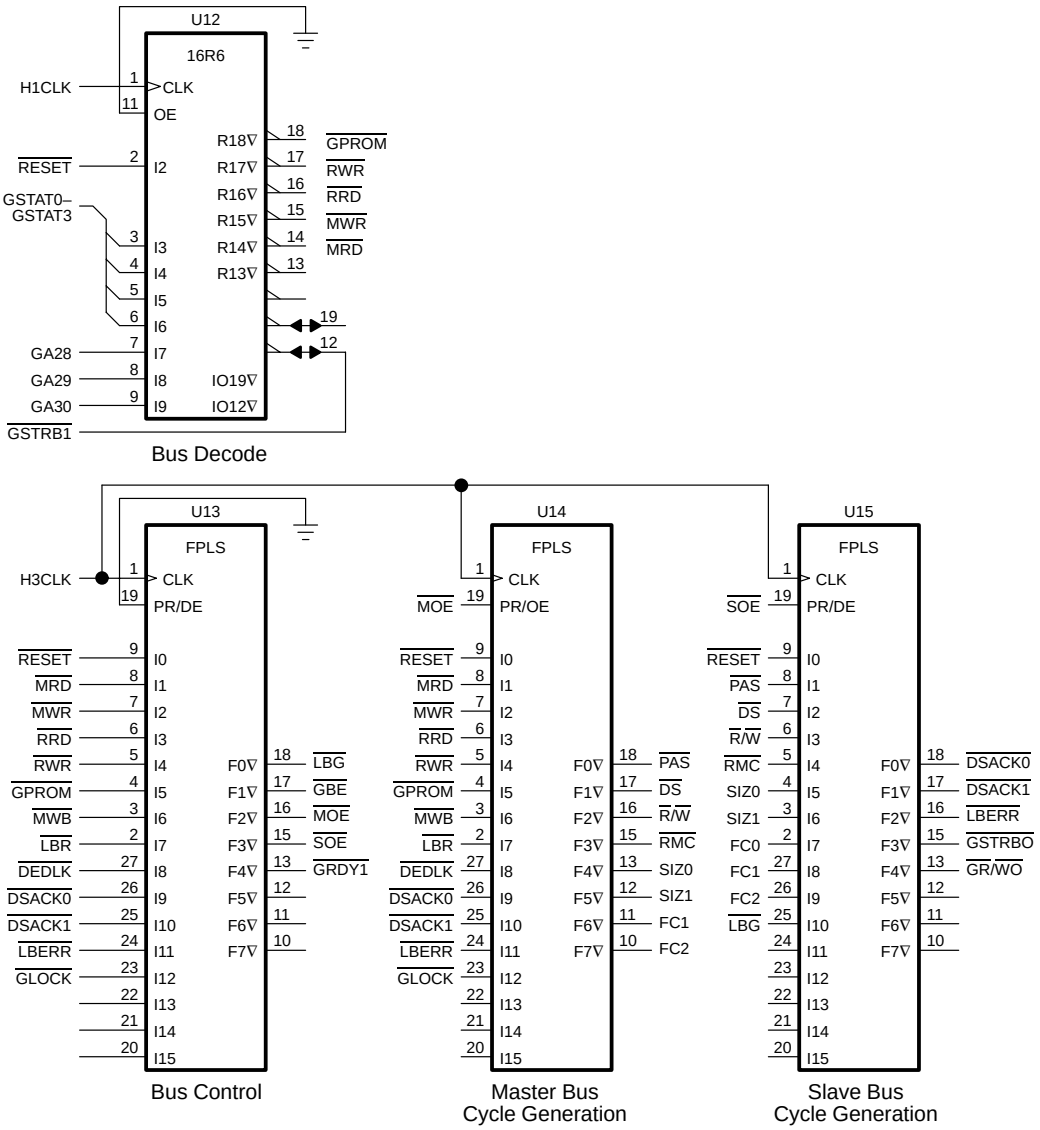



## Hardware Description

After examining the VIC068/VAC068 interface and capabilities and comparing them with the ’C40, we initiated a prototype design. Based on the preceding discussion, the strategy is to map from the given set of ’C40 bus signals to a set of 680x0-like signals, driving their counterparts on the VIC068 and VAC068 for master cycles (the ’C40 is reading from or writing to the VMEbus). Not only can the ’C40 initiate VMEbus cycles as a bus master, but also the card should respond to slave cycles. Most often, slave access is gained through shared memory on the (slave) card. On the ’C40-based VME card, one set of signals is required to respond to bus requests from the VIC068/VAC068, and an additional set is required to “hold off” the ’C40 global side during such transfers.

To accomplish this transformation of signals, programmable logic is applied. We wanted to keep the design time to a minimum while maintaining the most flexible (i.e., programmable) design. Based on this, we used the Texas Instruments TIB82S105BC programmable logic sequencer. This device is a field-programmable Mealy-type state machine with 16 inputs, 8 outputs, 48 product terms, and 14 pairs of sum terms; it operates at a maximum frequency of 50 MHz [10]. Development tools for these sequencers are plentiful and inexpensive — we used Data I/O’s ABEL version 4.0 for all programmable logic device (PLD) development.

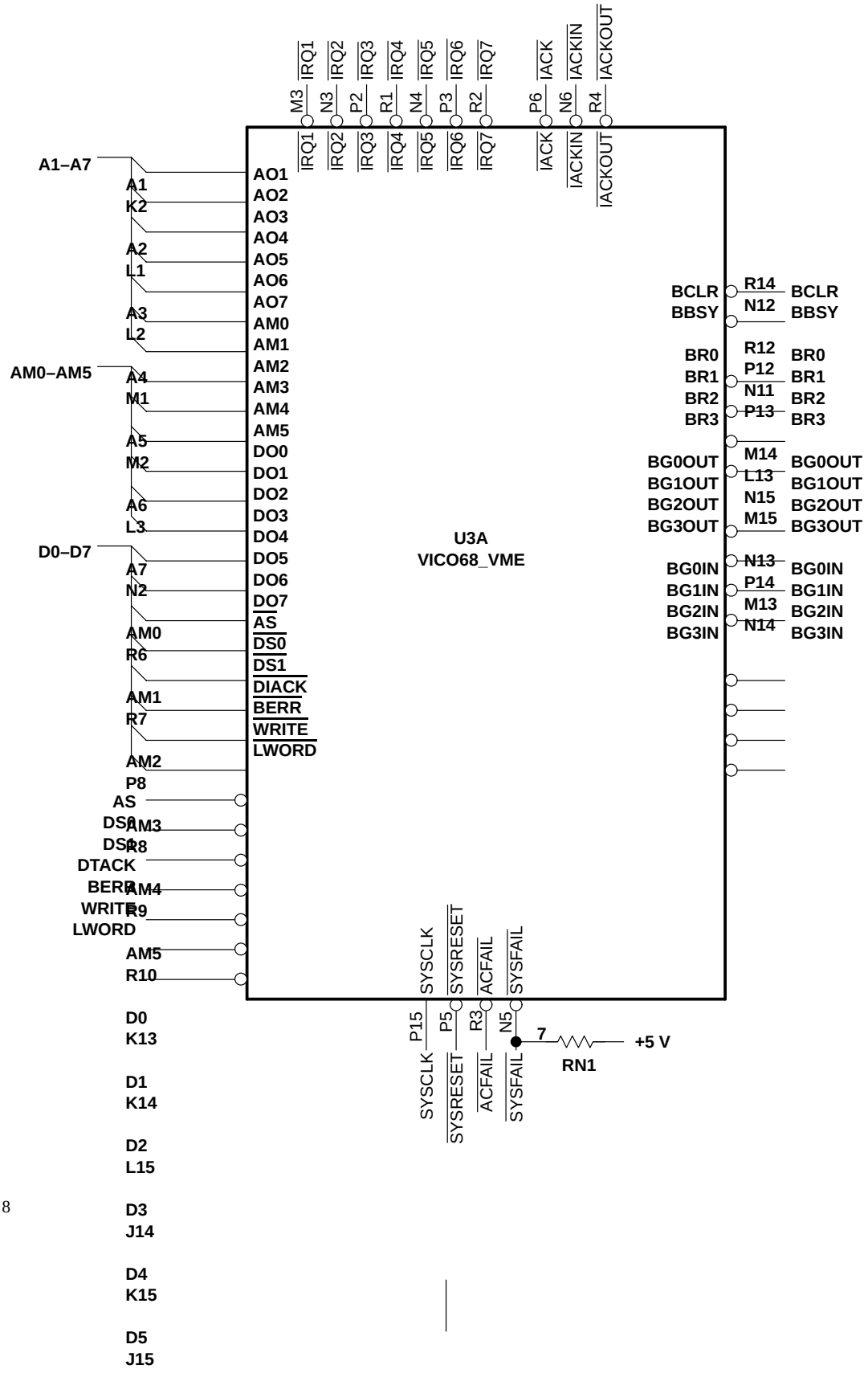
**Figure 4. TMS320C40 – VIC/VAC Prototype Programmable Logic**




## Reset Circuitry

We found that the VIC068/VAC068 required a "global" reset to operate correctly. In particular, the VIC068 $\overline{\text{IRESET}}$ signal should be driven with the incoming reset request. This signal can be buffered as shown to provide the delayed signal to the $\overline{\text{IPL0}}$ input —thereby providing the required stimulus for a VIC068 global reset. Given these inputs, the VIC068, in turn, drives the $\overline{\text{RESET}}$ line (and the $\overline{\text{SYSRESET}}$ line if the VIC068 is configured as a system controller) for 200 milliseconds. We used the $\overline{\text{RESET}}$ output on the VIC068 to reset both the 'C40 and the VAC068, as well as programmable logic on-board.

**Figure 5. TMS320C40 – VIC/VAC Prototype VICO68 VMEbus Interface**



## Address Bus Decoding

The VIC068/VAC068 interface (and consequently the VMEbus itself) is mapped into the 'C40 global side at 0D000 0000h. In our application, we divided the global side into two halves via the STRB ACTIVE field in the 'C40 global memory control register. We placed zero-wait-state devices (fast SRAM) in the lower half and placed slower memory (EPROM) and peripherals (the VIC068/VAC068 pair) in the upper half. Therefore, the 'C40 addresses program memory via STRB0 and addresses the VMEbus via STRB1. As shown in the accompanying schematics, U12 is a 16R6 programmable device (in particular, a Texas Instruments TIBAL16R6-5C was used). It decodes each global 'C40 STRB1 bus cycle by using the 'C40 H1 clock. Cycle type decoding is performed fully via the STAT lines (instead of using the R/$\overline{W}$ strobe) and allows for future expansion/reconfiguration if required. As shown, the STRB1 range is divided into eight distinct segments by using GA28—GA30 (GA31 is implicitly a logic 1). Outputs of the decoding operation are VMEbus master write ($\overline{MWR}$), master read ($\overline{MRD}$), VIC068/VAC068 register write ($\overline{RWR}$), register read ($\overline{RRD}$) and EPROM read ($\overline{GPROM}$). The VIC068/VAC068 documentation shows that the VAC068 is hard-wired, starting at address 0FFFD 0000h, and designates VIC068 selection, starting at address 0FFFC 0000h. A memory map for the global side, as decoded by the 16R6 logic device, is shown in Table 1. The ABEL source code is provided in the appendix.

**Table 1. Global Side Memory Map**

| Address | Unit Addressed |
|---|---|
| 08000 0000h | SRAM |
| 0C000 0000h | EPROM |
| 0D000 0000h | VMEbus A24 |
| | Address 00 0000h |
| 0FFFC 0000h | VIC068 Register Set |
| 0FFFD 0000h | VAC068 Register Set |

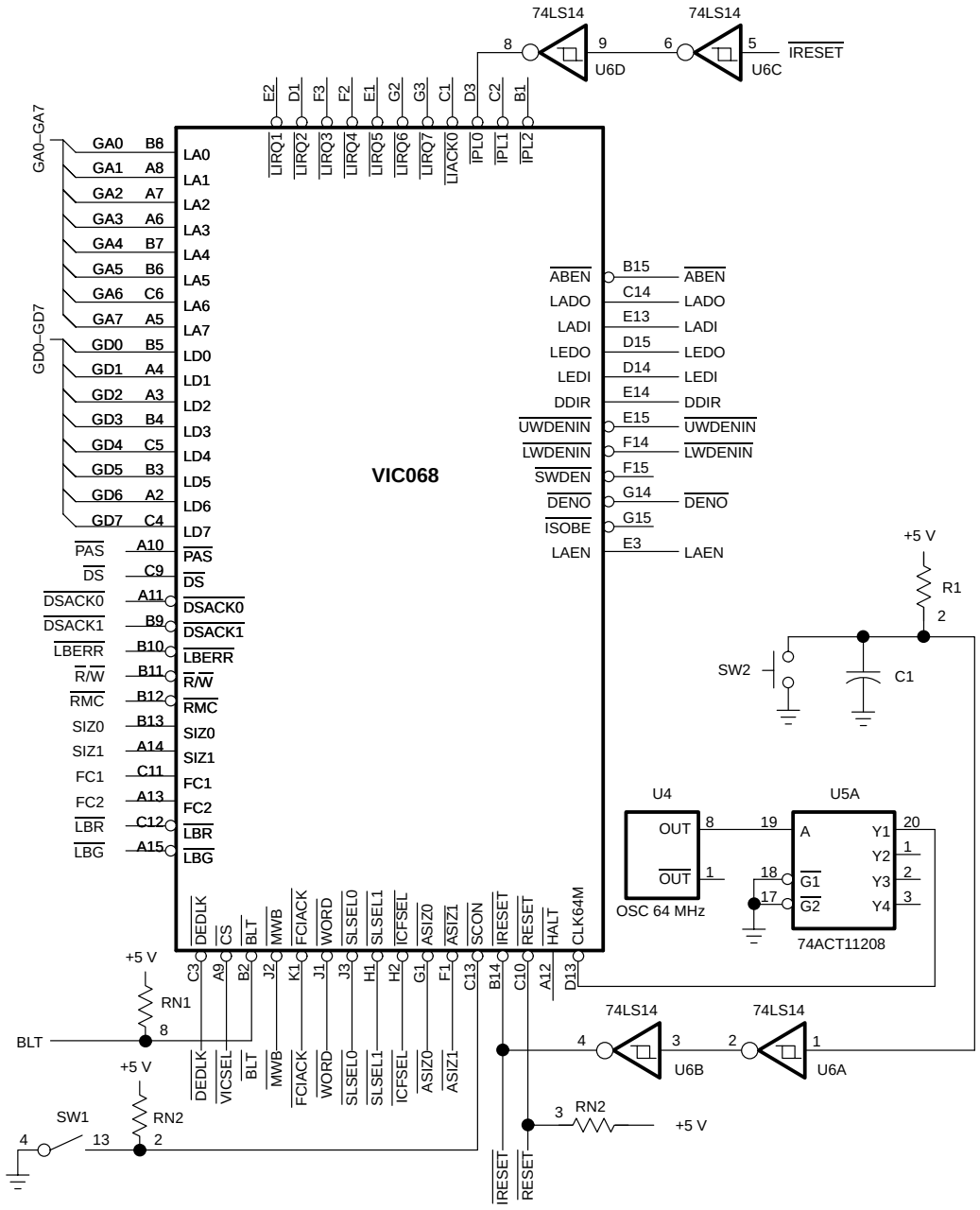
## Bus Control

Once a cycle in the VMEbus address range is detected by the address-decoding programmable logic, the sequencers provide the signals required for both master and slave cycles. U13 is the first of three sequencers and facilitates overall bus control, providing these enable signals: 'C40 global bus ($\overline{GBE}$), master cycle sequencer output ($\overline{MOE}$), slave cycle sequencer output ($\overline{SOE}$), VMEbus slave local bus grant ($\overline{LBG}$), and a 'C40 ready signal ($\overline{GRDY1}$). Notice that a full complement of inputs is presented to the bus control sequencer. This was done to accommodate all possible cycles and allow reconfiguration without hardware changes. While the 'C40 H3 clock (20 MHz) was used here, this is not an absolute requirement, because the array of sequencers operates asynchronously, once a master or slave cycle begins. However, using H3 simplifies the sequencer code because the H3 clock serves as a convenient reference to the 'C40 cycle in progress.

A master cycle begins with U12 generating a master read or write signal or a register read or write. This enables the output of the master bus cycle signal generation sequencer U14 (in fact, this signal is asserted during all bus activity other than slave cycles). A master cycle ends with the assertion of the acknowledge signals $\overline{DSACK0}$ and $\overline{DSACK1}$ and/or the local bus error signal $\overline{LBERR}$, all generated by the VIC068 in response to acknowledge signals received over the VMEbus. The sequencer responds to these signals by asserting $\overline{GRDY}$ to supply the ready signal $\overline{RDY1}$ for this 'C40 STRB1 access. In this design, external ready signals are used exclusively (versus ANDing or ORing with internal ready), and the generation of the ready signal conforms to the second of two methods described in [1]: high between accesses.

Slave cycles are initiated by the assertion of the VIC068 local bus request ($\overline{LBR}$) input signal. Then, U13 provides bus control by first disabling the ’C40 global bus (deasserting $\overline{GBE}$) and the master cycle sequencer output ($\overline{MOE}$) and then by enabling the outputs on the slave cycle sequencer ($\overline{SOE}$), U15. When the bus has been successfully “seized”, the local bus grant signal ($\overline{LBG}$) is asserted. Slave cycles are terminated by the deassertion of the local bus request input signal.

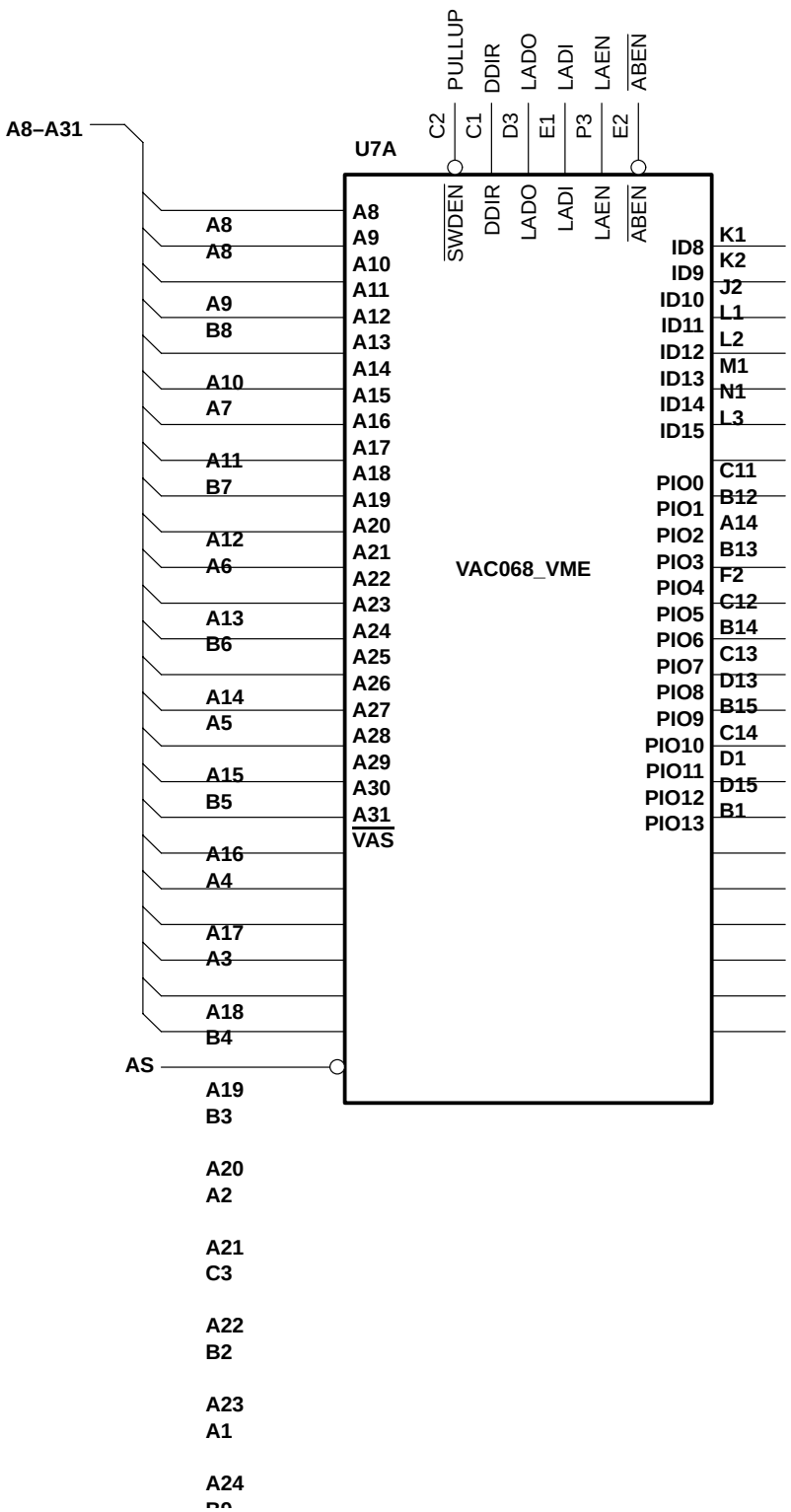
**Figure 6. TMS320C40 – VIC/VAC Prototype VICO68 Local Bus Interface**



### Master Bus Cycle Generation

The master bus cycle generation sequencer U14 runs in tandem with the bus control sequencer U13, and the sequencer code found in U13 and U14 results from one common state diagram. It was necessary to split these functions because of the number of outputs per sequencer. Therefore, the inputs to U14 are identical to those on U13. Master bus cycles proceed according to the appropriate cycle (read or write) definition in [7]. The function code lines are driven to indicate the widest possible audience, supervisory data. Termination of a master cycle ends with the assertion of the acknowledge signals $\overline{\text{DSACK0}}$ and $\overline{\text{DSACK1}}$ and/or the local bus error signal $\overline{\text{LBERR}}$ as described above. Note that VIC068/VAC068 register accesses are also master accesses in the 'C40 global side address range. While the sequencer code does not initiate read-modify-write cycles, you could use the 'C40 $\overline{\text{GLOCK}}$ input to do this.

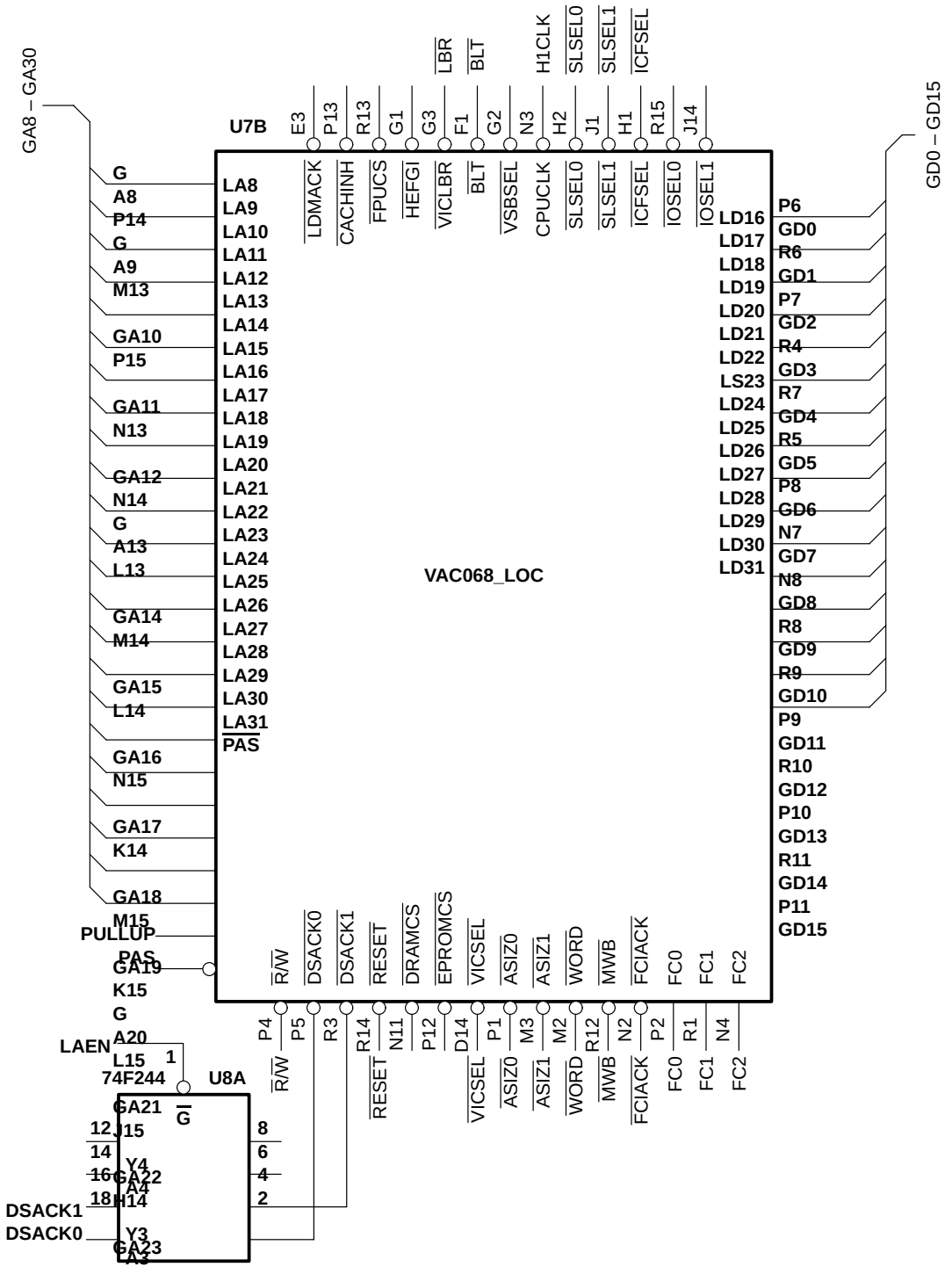
**Figure 7. TMS320C40 – VIC/VAC Prototype VACO68 VME Interface**


A8–A31
U7A
C2
C1
D3
E1
P3
E2
PULLUP
DDIR
LADO
LADI
LAEN
ABEN
SWDEN
DDIR
LADO
LADI
LAEN
ABEN
A8
A9
A10
A11
A12
A13
A14
A15
A16
A17
A18
A19
A20
A21
A22
A23
A24
A25
A26
A27
A28
A29
A30
A31
VAS
VAC068_VME
ID8
ID9
ID10
ID11
ID12
ID13
ID14
ID15
PIO0
PIO1
PIO2
PIO3
PIO4
PIO5
PIO6
PIO7
PIO8
PIO9
PIO10
PIO11
PIO12
PIO13
K1
K2
J2
L1
L2
M1
N1
L3
C11
B12
A14
B13
F2
C12
B14
C13
D13
B15
C14
D1
D15
B1
A8
A8
A9
B8
A10
A7
A11
B7
A12
A6
A13
B6
A14
A5
A15
B5
A16
A4
A17
A3
A18
B4
AS
A19
B3
A20
A2
A21
C3
A22
B2
A23
A1
A24

### Slave Bus Cycle Generation

Slave cycles are initiated by the VAC068 in response to a request over the VMEbus in the selected range as determined in the appropriate VAC068 register (discussed in *VIC068/VAC068 Software Initialization, page 15* ). Inputs to the sequencer are the common 680x0 bus signals driven by the VIC068 for slave cycles (and alternately driven by the master sequencers for master cycles). Assertion of the local bus grant signal $\overline{\text{LBG}}$ by U13 indicates the absence of the ’C40 on the global bus, thereby allowing access of shared global SRAM by the VIC068/VAC068 pair. Assuming the correct transfer size, the memory strobe signals $\overline{\text{GSTRB0}}$ and GR/W0 are driven, providing access to the shared global SRAM. After this, acknowledgement is provided via $\overline{\text{DSACK0}}$ and $\overline{\text{DSACK1}}$, ending the slave cycle. Note that while VAC068 documentation states that its DSACK signals can be put in the high-impedance state on the assertion of LAEN, we found this not to be the case with our configuration. Therefore, U8A was required to artificially put those signals in the high-impedance state so that the slave sequencer could control the data acknowledgement.

**Figure 8. TMS320C40 – VIC/VAC Prototype VACO68 Local Interface**



## VIC068/VAC068 Software Initialization

While the VIC068/VAC068 pair register set is, at first glance, overwhelming, we found that very few registers require attention before the pair can be used for either master or slave operations. The VAC068 should be initialized first because it controls both master and slave address mapping. When initialization is complete, the VIC068 is programmed. You can fine-tune the interface by using the programmable delay registers for the interface after initial capability is verified. As we programmed the VIC068/VAC068 using C, we developed vic.h and vac.h header files, which give base and offset definitions for the complete register set of each device.

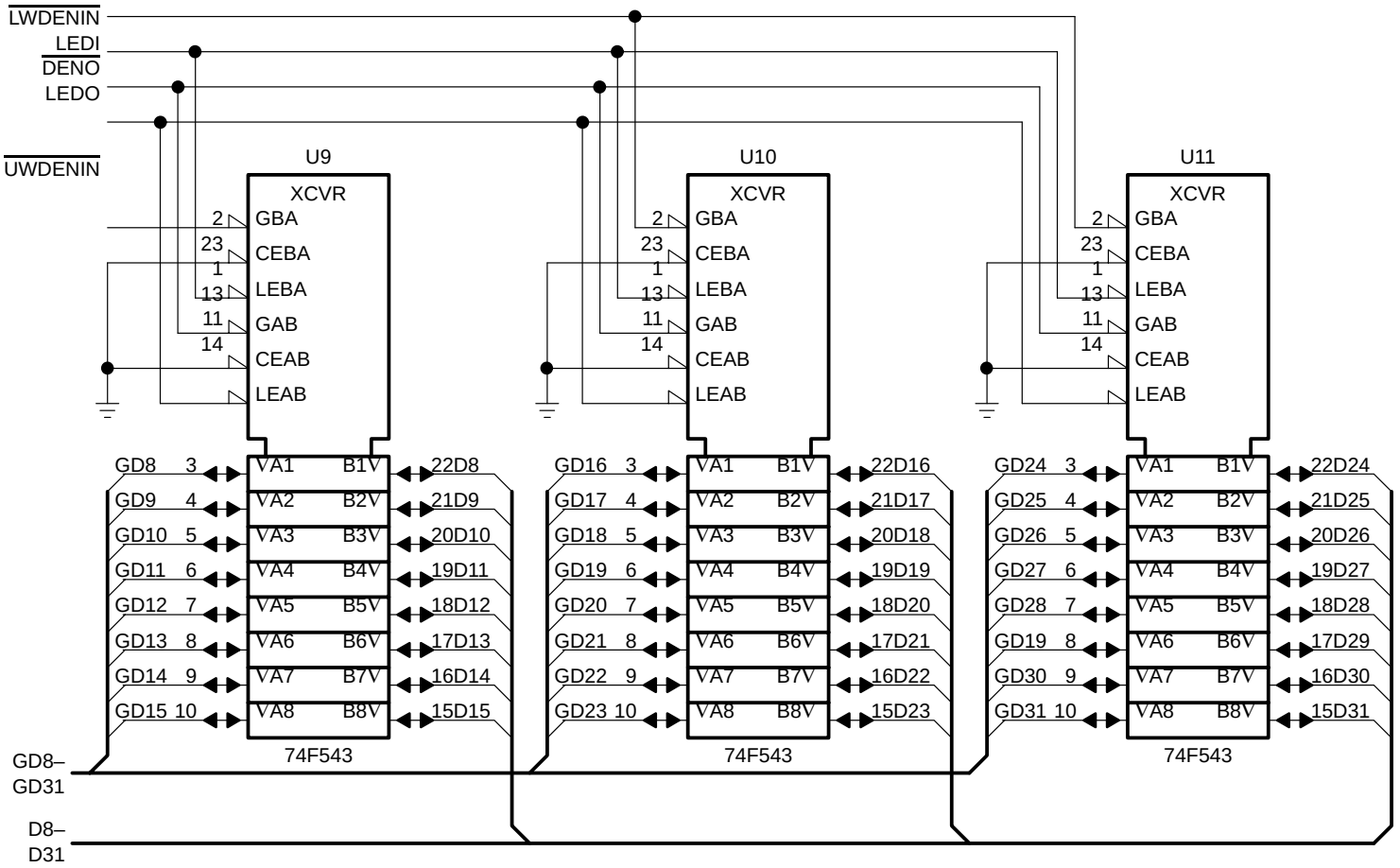
Before programming the VIC068/VAC068 pair, you must bring the VAC068 out of its initial Force EPROM mode (which asserts $\overline{\text{EPROMCS}}$ for all accesses) by reading from the EPROM space beginning at 0FF00 0000h. While the address-decoding programmable logic device U12 does not provide for access to this range, we can initiate a dummy access to this region by manipulating the 'C40 global memory interface control register. We first set the SWW and WTCNT fields so that the register will provide zero-wait-state, internal ready dependent (only) accesses to the appropriate strobe (STRB1 for our case). We then perform a read from address 0FF00 0000h, reset the SWW field to external ready accesses, and perform a second read to the VAC068 — this time at the VAC068 register base 0FFFD 0000h. This second read provides the required access to snap the sequencers back to their default states.

After the Force EPROM mode is exited, we first verify that the VAC068 can be addressed by reading the device ID via the VAC068 ID register. Then, we program the slave SLSEL0 base address register, the SLSEL0 mask register, and the master A24 base address register. To enable the VAC068 decode and compare functions, the last step is to write to the VAC068 ID register. The VIC068 ID register is similarly polled; following the successful read of that register, we set the address modifier source register and the slave select 0 control 0 register. This completes the initial programming of the pair. Now, we can extinguish the SYSFAIL LED (if applicable) by writing to the interprocessor communication 7 register. The initial register settings for our application are provided in Table 2.

**Table 2. VIC068/VAC068 Initial Register Settings**

| Address | Register | Size (Bits) | Setting |
|---|---|---|---|
| 0FFFD 0200h | VAC SLSEL0 Base | 16 | 0010h |
| 0FFFD 0300h | VAC SLSEL0 Mask | 16 | 00F0h |
| 0FFFD 0800h | VAC A24 Base | 13 | 0D10h |
| 0FFFD 2900h | VAC ID | 16 | Write to Enable VAC |
| 0FFFC 00B4h | VIC Address Modifier | 8 | 03Dh |
| 0FFFC 00C0h | VIC Slave Select 0 Control 0 | 8 | 014h |

**Figure 9. TMS320C40 – VIC/VAC Prototype VMEbus Data Bus Interface**



## Conclusion

We have developed a prototype interface between the ’C40 DSP and the Cypress VIC068/VAC068 with a minimum amount of programmable logic in the form of simple PLDs and sequencers. The result is a reconfigurable, programmable interface for A24/D32 VMEbus master/slave capability. While the initial transfer speed is low, you can improve it by increasing the sequencer’s clock rate and eliminating unnecessary states in the prototype sequencer code. You can initiate read-modify-write cycles with the existing hardware by using the ’C40 LOCK instruction group. Ultimately, the knowledge gained from this effort could be used to develop an FPGA interface that improves both speed and size. In the future, simulation models for state-of-the-art devices such as the ’C40 and VIC068/VAC068 should precede the actual hardware release, allowing early proof-of-concept with in-place CAE tools.

## Acknowledgements

The authors would like to thank James E. S. Wilkins of The MITRE Corporation for his essential contributions to the prototype effort and also to David Fuchs, Texas Instruments, Waltham, Massachusetts, for his timely support and encouragement. A special note of thanks go to the staff at Data I/O for their support of the ABEL programming language.

## References

1. *TMS320C4x User’s Guide*, Texas Instruments, 1991.
2. *VIC068 VMEbus Interface Controller Specification*, Cypress Semiconductor, 1991.
3. *VAC068 VMEbus Address Controller Specification*, Cypress Semiconductor, 1991.
4. Siy, P. F., and W. T. Ralston, “Application of the TI ’C40 in Satellite Modem Technology,” presented at the *Third Annual International Conference on Signal Processing Applications and Technology*, Boston, MA, November, 1992.
5. *IEEE Standard for a Versatile Backplane Bus: VMEbus*. New York: Wiley-Interscience, 1987.
6. W.D. Peterson. *The VMEbus Handbook*, VFEA International Trade Association, Scottsdale, AZ, 1990.
7. *MC68020 32-Bit Microprocessor User’s Manual*, Motorola, Inc., 1984.
8. Henessey, J. L., and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. San Mateo: Morgan Kaufmann Publishers, Inc., 1990.
9. Dewar, R. B. K., and M. Smosna. *Microprocessors: A Programmer’s View.* New York: McGraw-Hill, Inc., 1990.
10. *Programmable Logic Data Book*, Texas Instruments, 1990.

## Appendix A: Address Bus Decoder — ABEL Source

| Module | Decode |
|---|---|
| Title | Global Bus Decode |
| Date | 24 March 1992 |
| Revision | 1.0 |
| Part | TIBPAL16R6-5C |
| Abel Version | 4.00 |
| Designer | Peter F. Siy |
| Company | MITRE Corp. |
| Location | Bedford, MA |
| Project | C40 I/O Board |

U12 device ’P16R6’;

```
"Inputs"
clk, reset                      pin 1,2;     "clock, reset"
gstat0,gstat1,gstat2,gstat3     pin 3,4,5,6; "C40 status"
ga28,ga29,ga30                  pin 7,8,9;   "C40 address"
gstrb1                          pin 12;      "C40 strobe 1"
oute                            pin 11;      "output enable"

"Outputs"
mrd,mwr          pin 13,14;   "master read & write"
rrd,rwr          pin 15,16;   "register read & write"
gprom            pin 17;      "PROM select"

"Misc"
ga31 = 1;                 "dummy var"

"Sets"
stat = [gstat3,gstat2,gstat1,gstat0];   "status"
addr = [ga31,ga30,ga29,ga28];           "ms nibble"
output = [gprom,rwr,rrd,mwr,mrd];       "output"

H,L,X,C,Z = 1,0,.X.,.C.,.Z.;

equations
output.c = clk;
output.oe = !oute;

"Master Read"
!mrd := reset & (addr == ^hd) & (stat == [1,0,X,X]) & !gstrb1;

"Master Write"
!mwr := reset & (addr == ^hd) & ((stat == [1,1,0,1]) #
(stat == [1,1,1,0])) & !gstrb1;

"Register Read"
!rrd := reset & (addr == ^hf) & (stat == [1,0,X,X]) & !gstrb1;

"Register Write"
!rwr := reset & (addr == ^hf) & ((stat == [1,1,0,1]) #
(stat == [1,1,1,0])) & !gstrb1;

"PROM Read"
!gprom := reset & (addr == ^hc) & (stat == [1,0,X,X]) & !gstrb1;
```

## Appendix A (Continued)

```
test_vectors

([clk,reset,gstat3,gstat2,gstat1,gstat0,ga30,ga29,ga28,
gstrb1,oute] -> output)

[C,X,X,X,X,X,X,X,X,1] ->     Z;  "1 test for high-z"
[C,0,X,X,X,X,X,X,X,0] ->    ^b11111;"2 test for reset"
[C,1,1,0,X,X,1,0,1,0,0] ->   ^b11110;"3 test for master read"
[C,1,1,1,0,1,1,0,1,0,0] ->   ^b11101;"4 test for master write"
[C,1,1,1,1,0,1,0,1,0,0] ->   ^b11101;"5 test for master write"
[C,1,1,0,X,X,1,1,1,0,0] ->   ^b11011;"6 test for register read"
[C,1,1,1,0,1,1,1,1,0,0] ->   ^b10111;"7 test for register write"
[C,1,1,1,1,0,1,1,1,0,0] ->   ^b10111;"8 test for register write"
[C,1,1,0,X,X,1,0,0,0,0] ->   ^b01111;"9 test for PROM read"
[C,1,1,0,X,X,0,0,0,0,0] ->   ^b11111;"10 test bad address"
[C,1,0,0,0,0,1,1,1,0,0] ->   ^b11111;"11 test bad status"
```

end decode

## Appendix B: Bus Control Sequencer — ABEL Source

```
module    bus_control
title        'C40 Bus Control
Date         30 March 1992
Revision     1.0
Part         TIB82S105BC
Abel Version 4.00
Designer     Peter F. Siy
Company      MITRE Corp.
Location     Bedford, MA
Project      'C40 I/O Card '

U13 device 'F105';

"Inputs"
clk, reset             pin 1,9;         "clock, reset"
mrd,mwr,rrd,rwr,gprom  pin 8,7,6,5,4;   "decoded cycle"
mwb,lbr                pin 3,2;         "master/slave requests"
dedlk                  pin 27;          "m/s deadlock"
dsack0,dsack1,lberr    pin 26,25,24;    "cycle responses"
glock                  pin 23;          "C40 lock"
oe                     pin 19;          "output enable"

"Outputs"
lbg       pin 18       istype 'buffer,reg_RS';          "slave grant"
gbe       pin 17       istype 'buffer,reg_RS';          "C40 g bus enable"
soe,moe   pin 15,16    istype 'buffer,reg_RS';          "pls oe(s)"
grdy1     pin 13       istype 'buffer,reg_RS';          "C40 ready 1"

"Sets"
cycle = [gprom,rwr,rrd,mwr,mrd];     "cycle request"
ack = [dsack1,dsack0];               "acknowledge"
output = [grdy1,soe,moe,gbe,lbg];    "output"

"State Description"
P4,P3,P2,P1,P0node 41,40,39,38,37 istype 'reg_RS';
sreg = [P4,P3,P2,P1,P0];
```

## Appendix B (Continued)

```
 S0 = [0,0,0,0,0];
 S1 = [0,0,0,0,1];
 S2 = [0,0,0,1,0];
 S3 = [0,0,0,1,1];
 S4 = [0,0,1,0,0];
 S5 = [0,0,1,0,1];
 S6 = [0,0,1,1,0];
 S7 = [0,0,1,1,1];
 S8 = [0,1,0,0,0];
 S9 = [0,1,0,0,1];
 S10 = [0,1,0,1,0];
 S11 = [0,1,0,1,1];
 S12 = [0,1,1,0,0];
 S13 = [0,1,1,0,1];
 S14 = [0,1,1,1,0];
 S15 = [0,1,1,1,1];
 S16 = [1,0,0,0,0];
 S17 = [1,0,0,0,1];
 S18 = [1,0,0,1,0];
 S19 = [1,0,0,1,1];
 S20 = [1,0,1,0,0];
 S31 = [1,1,1,1,1];

"Misc"
H,L,X,C,Z = 1,0,.X.,.C.,.Z.;

equations
output.OE = !oe;    "set output enable"
output.CLK = clk;   "clock the output regs"
sreg.CLK = clk;     "and state regs"

@page
state_diagram sreg
state S0:

if !reset then S0 WITH
 lbg.S = 1;      "slave disable"
 gbe.R = 1;      "enable C40 global side"
 soe.S = 1;      "slave disable"
 moe.R = 1;      "enable master pls"
 grdy1.S = 1;    "not ready"
 ENDWITH;

else if (!mrd # !mwr & lbr) then S1;    "master read/write"
else if (!rrd # !rwr & lbr) then S4;    "reg read/write"
else if (!gprom & lbr) then S8;         "EPROM read"
else if (!lbr # !dedlk) then S16 WITH   "slave request"
 gbe.S = 1;                             "disable global side"
 moe.S = 1;                             "and master pls"
 ENDWITH;

else S0 WITH
 lbg.S = 1;      "slave disable"
 gbe.R = 1;      "enable C40 global side"
 soe.S = 1;      "slave disable"
 moe.R = 1;      "enable master pls"
 grdy1.S = 1;    "not ready"
 ENDWITH;

@page
"Master Read/Write"
```

## Appendix B (Continued)

```
state S1:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;

else if !dedlk & ((!mwb) # (mwb)) then S16 WITH
 moe.S = 1;
 gbe.S = 1;
 ENDWITH;

else if !mwb then S2;”wait for !mwb”

else S1;

state S2:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;

else if !dedlk & ((!mwb) # (mwb)) then S16 WITH;
 moe.S = 1;
 gbe.S = 1;
 ENDWITH;

else if ((!dsack1 & !dsack0) # !lberr) then S3 WITH
 grdy1.R = 1;
 ENDWITH;

else S2;

state S3:
goto S0 WITH
 grdy1.S = 1;
 ENDWITH;

@page
”Register Read/Write”

state S4:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;

else if !dsack1 then S5 WITH
 grdy1.R = 1;
 ENDWITH;

else S4;

state S5:
goto S0 WITH
 grdy1.S = 1;
 ENDWITH;

@page
”EPROM Read, 150ns EPROMs”
```

## Appendix B (Continued)

```
state S8:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;

else goto S9;

state S9:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;

else goto S10;

state S10:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;d   ”not ready”
 ENDWITH;

else goto S11;

state S11:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;

else goto S12 WITH
 grdy1.R = 1;
 ENDWITH;

state S12:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;d   ”not ready”
 ENDWITH;

else goto S0 WITH
 grdy1.S = 1;
 ENDWITH;

@page
”Local Bus Request”
```

## Appendix B (Continued)

```
state S16:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;
else goto S17 WITH
 soe.R = 1;      ”enable slave PLS”
 ENDWITH;
state S17:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;
else goto S18 WITH
 lbg.R = 1;      ”finally allow slave access”
 ENDWITH;
state S18:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;
else if lbr then goto S19 WITH
 lbg.S = 1;      ”slave disable”
 ENDWITH;
else S18;
state S19:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;
else goto S20 WITH
 soe.S =1;       ”disable slave pls”
 ENDWITH;
state S20:
if !reset then S0 WITH
 lbg.S = 1;      ”slave disable”
 gbe.R = 1;      ”enable C40 global side”
 soe.S = 1;      ”disable slave pls”
 moe.R = 1;      ”enable master pls”
 grdy1.S = 1;    ”not ready”
 ENDWITH;
else goto S0 WITH
 moe.R = 1;
 gbe.R = 1;
 ENDWITH;
```

## Appendix B (Continued)

```
@page
"Power-Up"

 state S31:
goto S0 WITH
 lbg.S = 1;      "slave disable"
 lbg.R = 0;      "dummy err 6099"
 gbe.R = 1;      "enable C40 global side"
 gbe.S = 0;      "dummy err 6099"
 soe.S = 1;      "disable slave PLS"
 soe.R = 0;      "dummy err 6099"
 moe.R = 1;      "enable master pls"
 moe.S = 0;      "dummy err 6099"
 grdy1.S = 1;    "not ready"
 grdy1.R = 0;    "dummy err 6099"
 ENDWITH;

@page
test_vectors

([clk,reset,gprom,rwr,rrd,mwr,mrd,lbr,mwb,
 dsack1,dsack0,dedlk,lberr,glock,oe] ->
 [sreg,grdy1,soe,moe,gbe,lbg])

[1,X,X,X,X,X,X,X,X,X,X,X,X,0] -> [S31,X,X,X,X,X];  "1 power up"
[0,X,X,X,X,X,X,X,X,X,X,X,X,0] -> [S31,X,X,X,X,X];  "2 power up"
[C,0,X,X,X,X,X,X,X,X,X,X,X,0] -> [S0,1,1,0,0,1];   "3 reset state"
[C,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0] -> [S1,1,1,0,0,1];   "4 master read"
[C,1,1,1,1,1,1,0,1,0,1,1,1,1,1,0] -> [S2,1,1,0,0,1];   "5 mwb asserted"
[C,1,1,1,1,1,1,0,1,0,0,0,1,1,1,0] -> [S3,0,1,0,0,1];   "6 data acked"
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S0,1,1,0,0,1];   "7 ready for nxt"
[C,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0] -> [S1,1,1,0,0,1];   "8 master write"
[C,1,1,1,1,1,1,0,1,0,1,1,1,1,1,0] -> [S2,1,1,0,0,1];   "9 mwb asserted"
[C,1,1,1,1,1,1,0,1,0,0,0,1,1,1,0] -> [S3,0,1,0,0,1];   "10 data acked"
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S0,1,1,0,0,1];   "11 ready for nxt"
[C,1,1,1,1,0,1,1,1,1,1,1,1,1,1,0] -> [S4,1,1,0,0,1];   "12 reg read"
[C,1,1,1,1,0,1,1,1,1,1,0,1,1,1,0] -> [S5,0,1,0,0,1];   "13 data ackd"
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S0,1,1,0,0,1];   "14 ready for nxt"
[C,1,0,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S8,1,1,0,0,1];   "15 prom read"
[C,1,0,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S9,1,1,0,0,1];   "16 prom read"
[C,1,0,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S10,1,1,0,0,1];  "17 wait"
[C,1,0,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S11,1,1,0,0,1];  "18 wait"
[C,1,0,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S12,0,1,0,0,1];  "19 wait"
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S0, 1,1,0,0,1];  "20 ready for nxt"
[C,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0] -> [S16,1,1,1,1,1];  "21 slave request'
[C,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0] -> [S17,1,0,1,1,1];  "22 en slve pls"
[C,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0] -> [S18,1,0,1,1,0];  "23 slave grant"
[C,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0] -> [S18,1,0,1,1,0];  "24 slave aces"
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S19,1,0,1,1,1];  "25 rescend grant"
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S20,1,1,1,1,1];  "26 disable sl pls"
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S0, 1,1,0,0,1];  "27 end sl access"
[C,1,1,1,1,1,1,1,1,1,1,1,0,1,1,0] -> [S16,1,1,1,1,1];  "29 deadlock"
[C,1,1,1,1,1,1,1,0,1,1,1,1,1,1,0] -> [S17,1,0,1,1,1];  "30 en slve pls"
[C,1,1,1,1,1,1,1,0,1,1,1,1,1,1,0] -> [S18,1,0,1,1,0];  "31 slave grant"
[C,1,1,1,1,1,1,1,0,1,1,1,1,1,1,0] -> [S18,1,0,1,1,0];  "32 slave aces"
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S19,1,0,1,1,1];  "33 rescend grant"
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S20,1,1,1,1,1];  "34 disable sl pls"
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S0, 1,1,0,0,1];  "35 end sl access"
```

end bus_control

## Appendix C: Master Cycle Generation Sequencer — ABEL Source

```
module       master
title        'C40 Bus Control
Date         31 March 1992
Revision     1.0
Part         TIB82S105BC
Abel Version 4.00
Designer     Peter F. Siy
Company      MITRE Corp.
Location     Bedford, MA
Project      'C40 I/O Card '

U14 device 'F105';

"Inputs"
clk, reset              pin 1,9;          "clock, reset"
mrd,mwr,rrd,rwr,gprom   pin 8,7,6,5,4;    "decoded cycle"
mwb,lbr                 pin 3,2;          "master/slave requests"
dedlk                   pin 27;           "m/s deadlock"
dsack0,dsack1,lberr     pin 26,25,24;     "cycle responses"
glock                   pin 23;           "C40 lock"
oe                      pin 19;           "output enable"

"Outputs"
pas    pin 18 istype 'buffer,reg_RS';    "68K address strobe"
ds     pin 17 istype 'buffer,reg_RS';    "68K data strobe"
rw     pin 16 istype 'buffer,reg_RS';    "68K read/write bar"
rmc    pin 15 istype 'buffer,reg_RS';    "68K read-mod-write"
siz0   pin 13 istype 'buffer,reg_RS';    "68K size 0"
siz1   pin 12 istype 'buffer,reg_RS';    "68K size 1"
fc1    pin 11 istype 'buffer,reg_RS';    "68K function 1"
fc2    pin 10 istype 'buffer,reg_RS';    "68K function 0"

"Sets"
cycle = [gprom,rwr,rrd,mwr,mrd];           "cycle request"
ack = [dsack1,dsack0];                     "acknowledge"
output = [pas,ds,rw,rmc,siz0,siz1,fc1,fc2];"68K ouputs"

"State Description"
P4,P3,P2,P1,P0node 41,40,39,38,37 istype 'reg_RS';
sreg = [P4,P3,P2,P1,P0];
 S0 = [0,0,0,0,0];
 S1 = [0,0,0,0,1];
 S2 = [0,0,0,1,0];
 S3 = [0,0,0,1,1];
 S4 = [0,0,1,0,0];
 S5 = [0,0,1,0,1];
 S6 = [0,0,1,1,0];
 S7 = [0,0,1,1,1];
 S8 = [0,1,0,0,0];
 S9 = [0,1,0,0,1];
 S10 = [0,1,0,1,0];
 S11 = [0,1,0,1,1];
 S12 = [0,1,1,0,0];
 S13 = [0,1,1,0,1];
 S14 = [0,1,1,1,0];
 S15 = [0,1,1,1,1];
 S16 = [1,0,0,0,0];
 S17 = [1,0,0,0,1];
 S18 = [1,0,0,1,0];
 S19 = [1,0,0,1,1];
```

## Appendix C (Continued)

```
 S20 = [1,0,1,0,0];
 S21 = [1,0,1,0,1];
 S22 = [1,0,1,1,0];
 S23 = [1,0,1,1,1];
 S24 = [1,1,0,0,0];
 S25 = [1,1,0,0,1];
 S26 = [1,1,0,1,0];
 S27 = [1,1,0,1,1];
 S28 = [1,1,1,0,0];
 S29 = [1,1,1,0,1];
 S30 = [1,1,1,1,0];
 S31 = [1,1,1,1,1];

"Misc"
rwmemnode 42 istype 'reg_RS'; "r/w flag"
H,L,X,C,Z = 1,0,.X.,.C.,.Z.;

equations
output.OE = !oe;              "set output enable"
output.CLK = clk;             "clock the output regs"
sreg.CLK = clk;               "and state regs"
rwmem.CLK = clk;              "and r/w store"

@page

state_diagram sreg
state S0:
if (!reset # !dedlk) then S0 WITH
 pas.S = 1;                   "no strobe"
 ds.S= 1;                     "no strobe"
 rw.S = 1;                    "read"
 rwmem.S = 1;                 "flag for mem"
 rmc.S = 1;                   "no rmc
 siz0.R = 1; "set for"
 siz1.R = 1; "32-bit xfers"
 fc1.R = 1;  "set for supervisory"
 fc2.S = 1;  "data access"
 ENDWITH;

else if (!mrd & !rwmem & lbr) then S1 WITH     "master read"
 rw.S = 1;                                     "assert read/write"
 rwmem.S =1;
 ENDWITH;

else if (!mrd & rwmem & lbr) then S2 WITH      "master read"
 pas.R = 1;      "assert pas"
 ds.R = 1;       "and ds"
 ENDWITH;

else if (!mwr & rwmem & lbr) then S8 WITH      "master write"
 rw.R = 1;       "assert r/w"
 rwmem.R = 1;
 ENDWITH;

else if (!mwr & !rwmem & lbr) then S9 WITH     "master write"
 pas.R = 1;      "assert pas only"
 ENDWITH;

else if (!rrd & !rwmem & lbr) then S16 WITH    "reg read"
 rw.S = 1;       "assert r/w"
 rwmem.S = 1;
 ENDWITH;

else if (!rrd & rwmem & lbr) then S17 WITH "reg read"
 pas.R = 1;      "assert pas"
 ds.R = 1;       "and ds"
 ENDWITH;
```

## Appendix C (Continued)

```
else if (!rwr & rwmem & lbr) then S24 WITH ”reg write”
 rw.R = 1;
 rwmem.R = 1;
 ENDWITH;

else if (!rwr & !rwmem & lbr) then S25 WITH
 pas.R = 1;      ”assert pas only”
 ENDWITH;

else S0 WITH
 pas.S = 1;      ”no strobe”
 ds.S= 1;        ”no strobe”
 rw.S = 1;       ”read”
 rwmem.S = 1;    ”flag for mem”
 rmc.S = 1;      ”no rmc”
 siz0.R = 1;     ”set for”
 siz1.R = 1;     ”32–bit xfers”
 fc1.R = 1;      ”set for supervisory”
 fc2.S = 1;      ”data access”
 ENDWITH;

@page
”Master Read”

state S1:
if (!reset # !dedlk) then S0 WITH
 pas.S = 1;      ”no strobe”
 ds.S= 1;        ”no strobe”
 rw.S = 1;       ”read”
 rwmem.S = 1;    ”flag for mem”
 rmc.S = 1;      ”no rmc”
 siz0.R = 1;     ”set for”
 siz1.R = 1;     ”32–bit xfers”
 fc1.R = 1;      ”set for super”
 fc2.S = 1;      ”data access”
 ENDWITH;

else S2 WITH
 pas.R = 1;
 ds.R = 1;
 ENDWITH;

state S2:
if (!reset # !dedlk) then S0 WITH
 pas.S = 1;      ”no strobe”
 ds.S= 1;        ”no strobe”
 rw.S = 1;       ”read”
 rwmem.S = 1;    ”flag for mem”
 rmc.S = 1;      ”no rmc”
 siz0.R = 1;     ”set for”
 siz1.R = 1;     ”32–bit xfers”
 fc1.R = 1;      ”set for super”
 fc2.S = 1;      ”data access”
 ENDWITH;

else if !mwb then S3;      ”wait for !mwb”

else S2;

state S3:
if (!reset # !dedlk) then S0 WITH
 pas.S = 1;      ”no strobe”
 ds.S= 1;        ”no strobe”
 rw.S = 1;       ”read”
 rwmem.S = 1;    ”flag for mem”
```

## Appendix C (Continued)

```
rmc.S = 1;       ″no rmc″
 siz0.R = 1;     ″set for″
 siz1.R = 1;     ″32–bit xfers″
 fc1.R = 1;      ″set for supervisory″
 fc2.S = 1;      ″data access″
 ENDWITH;

else if ((!dsack1 & !dsack0) # !lberr) then S4 WITH
 grdy1.R = 1
 ENDWITH;

else S3;

state S4:
goto S0 WITH
 pas.S = 1;
 ds.S = 1;
 ENDWITH;

@page
″Master Write″

state S8:
if (!reset # !dedlk) then S0 WITH
 pas.S = 1;      ″no strobe″
 ds.S= 1;        ″no strobe″
 rw.S = 1;       ″read″
 rwmem.S = 1;
 rmc.S = 1;      ″no rmc″
 siz0.R = 1;     ″set for″
 siz1.R = 1;     ″32–bit xfers″
 fc1.R = 1;      ″set for supervisory″
 fc2.S = 1;      ″data access″
 ENDWITH;

else S9 WITH
 pas.R = 1;
 ENDWITH;

state S9:
if (!reset # !dedlk) then S0 WITH
 pas.S = 1;      ″no strobe″
 ds.S= 1;        ″no strobe″
 rw.S = 1;       ″read″
 rwmem.S = 1;
 rmc.S = 1;      ″no rmc″
 siz0.R = 1;     ″set for″
 siz1.R = 1;     ″32–bit xfers″
 fc1.R = 1;      ″set for supervisory″
 fc2.S = 1;      ″data access″
 ENDWITH;

else S10 WITH
 ds.r = 1;
 ENDWITH;
```

## Appendix C (Continued)

```
state S10:
if (!reset # !dedlk) then S0 WITH
 pas.S = 1;      ”no strobe”
 ds.S= 1;        ”no strobe”
 rw.S = 1;       ”read”
 rwmem.S = 1;
 rmc.S = 1;      ”no rmc”
 siz0.R = 1;  ”set for”
 siz1.R = 1;  ”32–bit xfers”
 fc1.R = 1;   ”set for super”
 fc2.S = 1;   ”data access”
 ENDWITH;

else if !mwb then S11;

else S10;

state S11:
if (!reset # !dedlk) then S0 WITH
 pas.S = 1;   ”no strobe”
 ds.S= 1;     ”no strobe”
 rw.S = 1;    ”read”
 rwmem.S = 1;
 rmc.S = 1;   ”no rmc”
 siz0.R = 1;  ”set for”
 siz1.R = 1;  ”32–bit xfers”
 fc1.R = 1;   ”set for supervisory”
 fc2.S = 1;   ”data access”
 ENDWITH;

else if ((!dsack1 & !dsack0) # !lberr) then S12;

else S11;

state S12:
goto S0 WITH
 pas.S = 1;
 ds.S = 1;
 ENDWITH;

@page
”Register Read”

state S16:
if !reset then S0 WITH
 pas.S = 1;   ”no strobe”
 ds.S= 1;     ”no strobe”
 rw.S = 1;    ”read”
 rwmem.S = 1;
 rmc.S = 1;   ”no rmc”
 siz0.R = 1;  ”set for”
 siz1.R = 1;  ”32–bit xfers”
 fc1.R = 1;   ”set for super”
 fc2.S = 1;   ”data access”
 ENDWITH;

else S17 WITH
 pas.R = 1;
 ds.R =1;
 ENDWITH;
```

## Appendix C (Continued)

```
state S17:
if !reset then S0 WITH
 pas.S = 1;   ”no strobe”
 ds.S= 1;     ”no strobe”
 rw.S = 1;    ”read”
 rwmem.S = 1;
 rmc.S = 1;   ”no rmc”
 siz0.R = 1;  ”set for”
 siz1.R = 1;  ”32–bit xfers”
 fc1.R = 1;   ”set for super”
 fc2.S = 1;   ”data access”
 ENDWITH;

else if !dsack1 then S18 WITH
 grdy1.R = 1
 ENDWITH;

else S17;

state S18:
goto S0 WITH
 pas.S = 1;
 ds.S = 1;
 ENDWITH;

@page
”Register Write”

state S24:
if !reset then S0 WITH
 pas.S = 1;   ”no strobe”
 ds.S= 1;     ”no strobe”
 rw.S = 1;    ”read”
 rwmem.S = 1;
 rmc.S = 1;   ”no rmc”
 siz0.R = 1;  ”set for”
 siz1.R = 1;  ”32–bit xfers”
 fc1.R = 1;   ”set for supervisory”
 fc2.S = 1;   ”data access”
ENDWITH;

else S25 WITH
 pas.R = 1;
 ENDWITH;

state S25:
if !reset then S0 WITH
 pas.S = 1;   ”no strobe”
 ds.S= 1;     ”no strobe”
 rw.S = 1;    ”read”
 rwmem.S = 1;
 rmc.S = 1;   ”no rmc”
 siz0.R = 1;  ”set for”
 siz1.R = 1;  ”32–bit xfers”
 fc1.R = 1;   ”set for supervisory”
 fc2.S = 1;   ”data access”
 ENDWITH;

else S26 WITH
 ds.r = 1;
 ENDWITH;
```

## Appendix C (Continued)

```
state S26:
if !reset then S0 WITH
 pas.S = 1;   ”no strobe”
 ds.S= 1;     ”no strobe”
 rw.S = 1;    ”read”
 rwmem.S = 1;
 rmc.S = 1;   ”no rmc”
 siz0.R = 1;  ”set for”
 siz1.R = 1;  ”32–bit xfers”
 fc1.R = 1;   ”set for supervisory”
 fc2.S = 1;   ”data access”
 ENDWITH;
else if !dsack1 then S27;

else S26;

state S27:
goto S0 WITH
 pas.S = 1;
 ds.S = 1;
 ENDWITH;

@page
”Power–Up”

state S31:
goto S0 WITH
 pas.S = 1;   ”no strobe”
 pas.R = 0;   ”error 6099 fix”
 ds.S= 1;     ”no strobe”
 ds.R= 0;     ”error 6099 fix”
 rw.S = 1;    ”read”
 rwmem.S = 1;
 rw.R = 0;    ”error 6099 fix”
 rmc.S = 1;   ”no rmc”
 rmc.R = 0;   ”error 6099 fix”
 siz0.R = 1;  ”set for”
 siz0.S = 0;  ”error 6099 fix”
 siz1.R = 1;  ”32–bit xfers”
 siz1.S = 0;  ”error 6099 fix”
 fc1.R = 1;   ”set for supervisory”
 fc1.S = 0;   ”error 6099 fix”
 fc2.S = 1;   ”data access”
 fc2.R = 0;   ”error 6099 fix”
ENDWITH;

@page
test_vectors

([clk,reset,gprom,rwr,rrd,mwr,mrd,lbr,mwb,
 dsack1,dsack0,dedlk,lberr,glock,oe] –>
[sreg,rwmem,fc2,fc1,siz1,siz0,rmc,rw,ds,pas])

[1,X,X,X,X,X,X,X,X,X,X,X,X,X,0] –> [S31,X,X,X,X,X,X,X,X,X]; ”1 power up”
[0,X,X,X,X,X,X,X,X,X,X,X,X,X,0] –> [S31,X,X,X,X,X,X,X,X,X]; ”2 power up”
[C,0,X,X,X,X,X,X,X,X,X,X,X,X,0] –> [S0, 1,1,0,0,0,1,1,1,1]; ”3 reset state”
[C,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0] –> [S2, 1,1,0,0,0,1,1,0,0]; ”4 master read”
[C,1,1,1,1,1,1,0,1,0,1,1,1,1,1,0] –> [S3, 1,1,0,0,0,1,1,0,0]; ”5 mwb
asserted”
[C,1,1,1,1,1,1,0,1,0,0,0,1,1,1,0] –> [S4, 1,1,0,0,0,1,1,0,0]; ”6 data acked”
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0] –> [S0, 1,1,0,0,0,1,1,1,1]; ”7 ready for
nxt”
[C,1,1,1,1,0,1,1,1,1,1,1,1,1,0] –> [S8, 0,1,0,0,0,1,0,1,1]; ”8 master
```

## Appendix C (Continued)

```
write”
[C,1,1,1,1,0,1,1,1,1,1,1,1,1,0] -> [S9, 0,1,0,0,0,1,0,1,0]; ”9 assert pas”
[C,1,1,1,1,0,1,1,1,1,1,1,1,1,0] -> [S10,0,1,0,0,0,1,0,0,0]; ”10 assert ds”
[C,1,1,1,1,0,1,1,0,1,1,1,1,1,0] -> [S11,0,1,0,0,0,1,0,0,0]; ”11 mwb”
[C,1,1,1,1,0,1,1,0,0,0,1,1,1,0] -> [S12,0,1,0,0,0,1,0,0,0]; ”12 data ackd”
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S0, 0,1,0,0,0,1,0,1,1]; ”13 ready for
next”
[C,1,1,1,0,1,1,1,1,1,1,1,1,1,0] -> [S16,1,1,0,0,0,1,1,1,1]; ”14 reg read”
[C,1,1,1,0,1,1,1,1,1,1,1,1,1,0] -> [S17,1,1,0,0,0,1,1,0,0]; ”15 assert
strobes”
[C,1,1,1,0,1,1,1,1,0,1,1,1,1,0] -> [S18,1,1,0,0,0,1,1,0,0]; ”16 data ackd”
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S0, 1,1,0,0,0,1,1,1,1]; ”17 ready for
nxt”
[C,1,1,0,1,1,1,1,1,1,1,1,1,1,0] -> [S24,0,1,0,0,0,1,0,1,1]; ”18 reg write”
[C,1,1,0,1,1,1,1,1,1,1,1,1,1,0] -> [S25,0,1,0,0,0,1,0,1,0]; ”19 assert pas”
[C,1,1,0,1,1,1,1,1,1,1,1,1,1,0] -> [S26,0,1,0,0,0,1,0,0,0]; ”20 assert ds”
[C,1,1,0,1,1,1,1,1,0,1,1,1,1,0] -> [S27,0,1,0,0,0,1,0,0,0]; ”21 data ackd”
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,0] -> [S0, 0,1,0,0,0,1,0,1,1]; ”22 ready for
next”
end master
```

APPENDIX D

SLAVE CYCLE GENERATION SEQUENCER – ABEL SOURCE

```
module    slave
title     ’C40 Bus Control
Date         2 April 1992
Revision     1.0
Part         TIB82S105BC
Abel Version 4.00
Designer     Peter F. Siy
Company      MITRE Corp.
Location     Bedford, MA
Project      ’C40 I/O Card ’

U15 device ’F105’;

”Inputs”
clk, reset   pin 1,9;      ”clock, reset”
pas,ds       pin 8,7;      ”address,data strobe”
rw,rmc       pin 6,5;      ”read/write strobes”
siz0,siz1    pin 4,3;      ”bus sizing”
fc0,fc1,fc2  pin 2,27,26;  ”function codes”
lbg          pin 25;       ”local bus grant”
oe           pin 19;       ”output enable”

”Outputs”
dsack0    pin 18 istype ’buffer,reg_RS’;    ”data ack 0”
dsack1    pin 17 istype ’buffer,reg_RS’;    ”data ack 1”
lberr     pin 16 istype ’buffer,reg_RS’;    ”bus error”
gstrb0    pin 15 istype ’buffer,reg_RS’;    ”C40 mem strobe”
grw0      pin 13 istype ’buffer,reg_RS’;    ”C40 read/write”

”Sets”
size = [siz1,siz0];     ”size”
func = [fc2,fc1,fc0];   ”function”
output = [grw0,gstrb0,lberr,dsack1,dsack0];

”State Description”
P3,P2,P1,P0   node 40,39,38,37    istype ’reg_RS’;
sreg = [P3,P2,P1,P0];
 S0 = [0,0,0,0];
 S1 = [0,0,0,1];
 S2 = [0,0,1,0];
```

## Appendix C (Continued)

```
 S3 = [0,0,1,1];
 S4 = [0,1,0,0];
 S5 = [0,1,0,1];
 S6 = [0,1,1,0];
 S7 = [0,1,1,1];
 S8 = [1,0,0,0];
 S9 = [1,0,0,1];
 S10 = [1,0,1,0];
 S11 = [1,0,1,1];
 S12 = [1,1,0,0];
 S13 = [1,1,0,1];
 S14 = [1,1,1,0];
 S15 = [1,1,1,1];
"Misc"
rwmemnode 42 istype 'reg_RS';    "r/w flag"
H,L,X,C,Z = 1,0,.X.,.C.,.Z.;
equations
output.OE = !oe;                 "set output enable"
output.CLK = clk;                "clock the output regs"
sreg.CLK = clk;                  "and state regs"
rwmem.CLK = clk;                 "and r/w store"
@page
state_diagram sreg
state S0:
if (!reset) then S0 WITH
 dsack0.S = 1;          "deassert"
 dsack1.S = 1;   "all"
 lberr.S = 1;    "strobes"
 gstrb0.S = 1;   "deassert C40"
 grw0.R = 1;     "strobe, read"
 rwmem.S = 1;    "set to read"
 ENDWITH;
else if (!lbg) then S1;
else S0 WITH
 dsack0.S = 1;   "deassert"
 dsack1.S = 1;   "all"
 lberr.S = 1;    "strobes"
 gstrb0.S = 1;   "deassert C40"
 grw0.R = 1;     "strobe, read"
 rwmem.S = 1;    "set to read"
 ENDWITH;
@page
"Sort Slave Request"
state S1:
"Reset"
if (!reset) then S0 WITH
 dsack0.S = 1;   "deassert"
 dsack1.S = 1;   "all"
 lberr.S = 1;    "strobes"
 gstrb0.S = 1;   "deassert C40"
 grw0.R = 1;     "strobe, read"
 rwmem.S = 1;    "set to read"
 ENDWITH;
"32-Bit Read"
else if (!pas & !ds & rw & !rwmem & !siz0 & !siz1) then S2 WITH
 grw0.S = 1;
 rwmem.S = 1;
 ENDWITH;
```

## Appendix C (Continued)

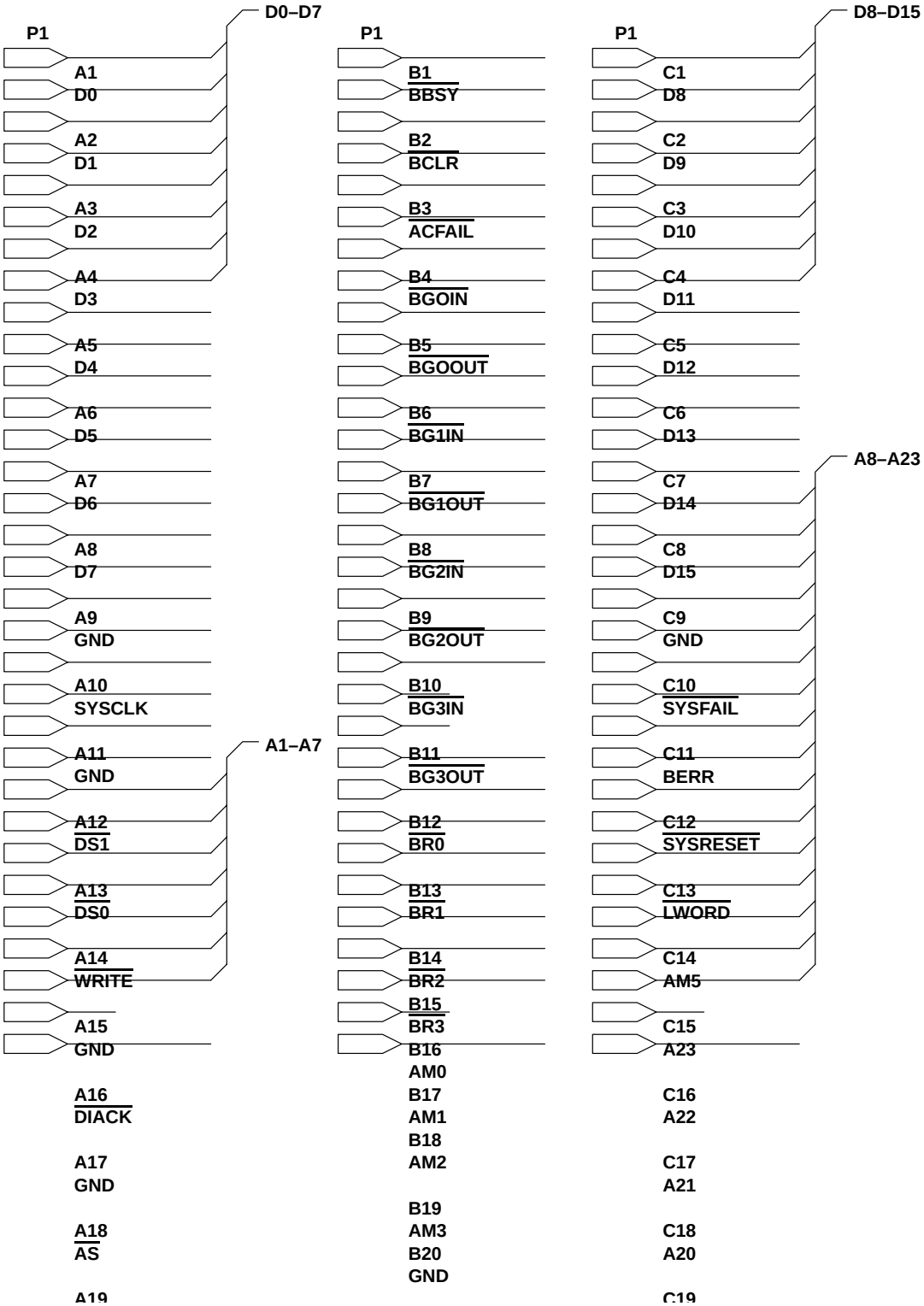
```
else if (!pas & !ds & rw & rwmem & !siz0 & !siz1) then S3 WITH
 gstrb0.R = 1;
 ENDWITH;

"32–Bit Write"
else if (!pas & !ds & !rw & rwmem & !siz0 & !siz1) then S2 WITH
 grw0.R = 1;
 rwmem.R = 1;
 ENDWITH;

else if (!pas & !ds & !rw & !rwmem & !siz0 & !siz1) then S3 WITH
 gstrb0.R = 1;
 ENDWITH;

"Illegal Access (non–32 bit access)"
 else if (!pas & !ds & (rw # !rw) & (siz0 # siz1)) then S9 WITH
 lberr.R = 1;
 ENDWITH;

else S1;

@page
"32–Bit Read/Write"

state S2:
goto S3 WITH
 gstrb0.R = 1;
 ENDWITH;

state S3:
goto S4 WITH
 dsack0.R = 1;
 dsack1.R = 1;
 ENDWITH;

state S4:
if pas then S0 WITH
 dsack0.S = 1;
 dsack1.S = 1;
 gstrb0.S = 1;
 ENDWITH;

else S4;

@page
"Illegal Access"

state S9:
if pas then S0 WITH
 lberr.S = 1;
 ENDWITH

@page
"Power–Up"

state S15:
goto S0 WITH
 dsack0.S = 1;   "no ack"
 dsack0.R = 0;   "error 6099 fix"
 dsack1.S = 1;   "no ack"
 dsack1.R = 0;   "error 6099 fix"
 rwmem.S = 1;    "r/w mem"
 rwmem.R = 0;    "error 6099 fix"
 lberr.S = 1;    "no bus error"
 lberr.R = 0;    "error 6099 fix"
 gstrb0.S = 1;   "no strobe"
 grw0.S = 1;     "read"
 ENDWITH;
```

## Appendix C (Continued)

```
@page
test_vectors

([clk,reset,pas,ds,rw,rmc,siz0,siz1,fc0,fc1,fc2,lbg,oe] ->
[sreg,rwmem,dsack0,dsack1,lberr,gstrb0,grw0])

[1,X,X,X,X,X,X,X,X,X,X,0] -> [S15,X,X,X,X,X,X]; "1 power up"
[0,X,X,X,X,X,X,X,X,X,X,0] -> [S15,X,X,X,X,X,X]; "2 power up"
[C,0,X,X,X,X,X,X,X,X,X,0] -> [S0, 1,1,1,1,1,1]; "3 reset state"
[C,1,1,1,1,1,1,1,X,X,X,0,0] -> [S1, 1,1,1,1,1,1]; "4 slave read,lbg"
[C,1,0,1,1,1,0,0,X,X,X,0,0] -> [S1, 1,1,1,1,1,1]; "5 pas asserted"
[C,1,0,0,1,1,0,0,X,X,X,0,0] -> [S3, 1,1,1,1,0,1]; "6 and ds, strobe"
[C,1,0,0,1,1,0,0,X,X,X,0,0] -> [S4, 1,0,0,1,0,1]; "7 ack"
[C,1,0,0,1,1,0,0,X,X,X,0,0] -> [S4, 1,0,0,1,0,1]; "8 wait for pas release"
[C,1,1,1,1,1,1,1,X,X,X,1,0] -> [S0, 1,1,1,1,1,1]; "9 done, release gstrb"
[C,1,1,1,0,1,1,1,X,X,X,0,0] -> [S1, 1,1,1,1,1,1]; "10 slave write,lbg"
[C,1,0,1,0,1,0,0,X,X,X,0,0] -> [S1, 1,1,1,1,1,1]; "11 pas asserted"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S2, 0,1,1,1,1,0]; "12 and ds"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S3, 0,1,1,1,0,0]; "13 assert strobe"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S4, 0,0,0,1,0,0]; "14 ack"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S4, 0,0,0,1,0,0]; "15 wait for pas release"
[C,1,1,1,1,1,1,1,X,X,X,1,0] -> [S0, 0,1,1,1,1,0]; "16 done,relase gstrb"
[C,1,1,1,1,1,1,1,X,X,X,0,0] -> [S1, 0,1,1,1,1,0]; "17 slave read,lbg"
[C,1,0,1,1,1,0,0,X,X,X,0,0] -> [S1, 0,1,1,1,1,0]; "18 pas asserted"
[C,1,0,0,1,1,0,0,X,X,X,0,0] -> [S2, 1,1,1,1,1,1]; "19 and ds, r/w asserted"
[C,1,0,0,1,1,0,0,X,X,X,0,0] -> [S3, 1,1,1,1,0,1]; "20 and strobe"
[C,1,0,0,1,1,0,0,X,X,X,0,0] -> [S4, 1,0,0,1,0,1]; "21 ack"
[C,1,0,0,1,1,0,0,X,X,X,0,0] -> [S4, 1,0,0,1,0,1]; "22 wait for pas release"
[C,1,1,1,1,1,1,1,X,X,X,1,0] -> [S0, 1,1,1,1,1,1]; "23 done,relase gstrb"
[C,1,1,1,1,1,1,1,X,X,X,0,0] -> [S1, 1,1,1,1,1,1]; "24 bad access,lbg"
[C,1,0,1,1,1,0,1,X,X,X,0,0] -> [S1, 1,1,1,1,1,1]; "25 pas asserted"
[C,1,0,0,1,1,0,1,X,X,X,0,0] -> [S9, 1,1,1,0,1,1]; "26 and ds, error"
[C,1,0,0,1,1,0,1,X,X,X,0,0] -> [S9, 1,1,1,0,1,1]; "27 wait for pas release"
[C,1,1,1,1,1,1,1,X,X,X,1,0] -> [S0, 1,1,1,1,1,1]; "28 done, release lberr"
[C,1,1,1,0,1,1,1,X,X,X,0,0] -> [S1, 1,1,1,1,1,1]; "29 slave write,lbg"
[C,1,0,1,0,1,0,0,X,X,X,0,0] -> [S1, 1,1,1,1,1,1]; "30 pas asserted"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S2, 0,1,1,1,1,0]; "31 and ds"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S3, 0,1,1,1,0,0]; "32 assert strobe"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S4, 0,0,0,1,0,0]; "33 ack"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S4, 0,0,0,1,0,0]; "34 wait for pas release"
[C,1,1,1,1,1,1,1,X,X,X,1,0] -> [S0, 0,1,1,1,1,0]; "35 done,relase gstrb"
[C,1,1,1,0,1,1,1,X,X,X,0,0] -> [S1, 0,1,1,1,1,0]; "36 slave write,lbg"
[C,1,0,1,0,1,0,0,X,X,X,0,0] -> [S1, 0,1,1,1,1,0]; "37 pas asserted"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S3, 0,1,1,1,0,0]; "38 and ds,assert strobe"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S4, 0,0,0,1,0,0]; "39 ack"
[C,1,0,0,0,1,0,0,X,X,X,0,0] -> [S4, 0,0,0,1,0,0]; "40 wait for pas release"
[C,1,1,1,1,1,1,1,X,X,X,1,0] -> [S0, 0,1,1,1,1,0]; "41 done,relase gstrb"

end slave
```

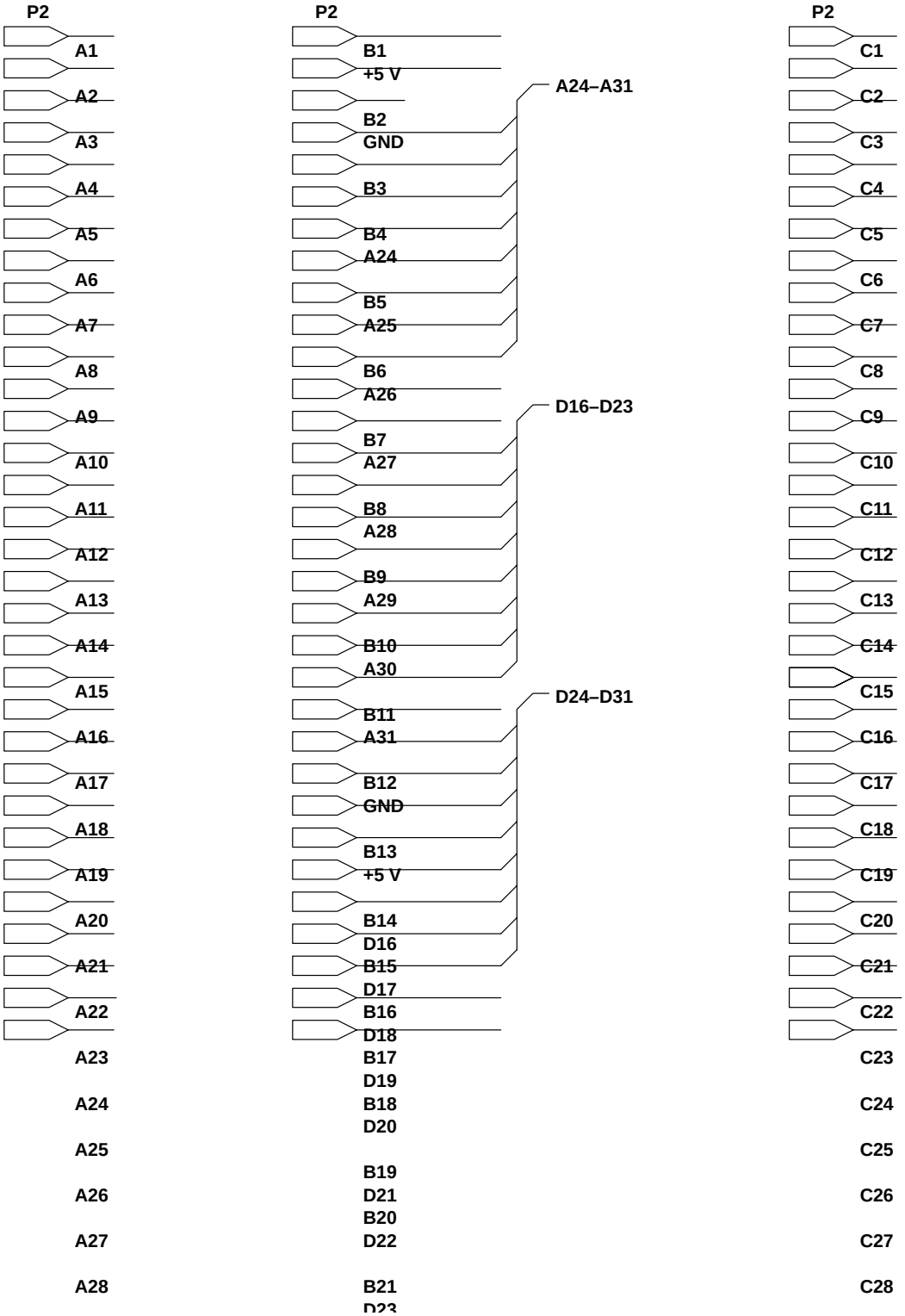
## Appendix D: Schematics

**Figure 10. TMS320C40 – VIC/VAC Prototype VMEbus P1 Connector**


P1
A1
D0
A2
D1
A3
D2
A4
D3
A5
D4
A6
D5
A7
D6
A8
D7
A9
GND
A10
SYSCLK
A11
GND
A12
DS1
A13
DS0
A14
WRITE
A15
GND
A16
DIACK
A17
GND
A18
AS
A19
D0–D7
A1–A7
P1
B1
BBSY
B2
BCLR
B3
ACFAIL
B4
BGOIN
B5
BGOOUT
B6
BG1IN
B7
BG1OUT
B8
BG2IN
B9
BG2OUT
B10
BG3IN
B11
BG3OUT
B12
BR0
B13
BR1
B14
BR2
B15
BR3
B16
AM0
B17
AM1
B18
AM2
B19
AM3
B20
GND
P1
C1
D8
C2
D9
C3
D10
C4
D11
C5
D12
C6
D13
C7
D14
C8
D15
C9
GND
C10
SYSFAIL
C11
BERR
C12
SYSRESET
C13
LWORD
C14
AM5
C15
A23
C16
A22
C17
A21
C18
A20
C19
D8–D15
A8–A23

## Appendix D: Schematics (Continued)

**Figure 11. TMS320C40 – VIC/VAC Prototype VMEbus P2 Connector**


P2
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
A16
A17
A18
A19
A20
A21
A22
A23
A24
A25
A26
A27
A28
P2
B1
+5 V
B2
GND
B3
B4
A24
B5
A25
B6
A26
B7
A27
B8
A28
B9
A29
B10
A30
B11
A31
B12
GND
B13
+5 V
B14
D16
B15
D17
B16
D18
B17
D19
B18
D20
B19
D21
B20
D22
B21
D23
A24–A31
D16–D23
D24–D31
P2
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14
C15
C16
C17
C18
C19
C20
C21
C22
C23
C24
C25
C26
C27
C28

## IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.