Connecting the Cypress VIC068/VAC068 to the TI TMS320C40: A Prototype Design

Introduction

The Cypress Semiconductor VIC068 VMEbus Interface Controller and its companion VAC068 VMEbus Address Controller provide a complete VMEbus interface including master and slave capability (Reference 2). As these components can be used in a wide variety of applications, it is natural to utilize the VIC068/VAC068 in a single- or multiple- TMS320C40 VMEbus card design.

This application note provides high-level as well as low-level details of interfacing VIC/VAC to TMS320C40. This allows for techniques to be implemented to minimize design time for subsequent efforts since this design has not been optimized for either size or speed. The Design Requisites section provides the design goals established prior to design as well as relevant background regarding devices involved. Hardware details, including schematics and programmable logic source code, represent the central focus of the paper. In addition, software initialization of the chip, set by the TMS320C40, is covered. Throughout this note, it is assumed that the reader is familiar with the TMS320C40 architecture (Reference 1), the basics of the VIC068A/VAC068A (Reference 2), and the VMEbus and its protocol(s) (References 5,6).

Design Requisites

Design Goals

This project began with the development of a set of design goals for the VME interface based on our particular needs. The main focus was on a TMS320C40 card providing both (VMEbus) master and slave capability for reads, writes, read-modify-writes, write posting, and slave block transfers. In terms of the address/data capability, a design was made 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 provides system controller capability. VMEbus interrupt support is not provided. The VAC068 is utilized to provide address control/mapping, two Universal Asynchronous Receiver/Transmitters (UARTs) (required for our application), and a general purpose parallel I/O. In addition to the VMEbus functionality, interface compatibility is required with both the existing TMS320C40-40, which has 50-nanosecond cycle time, and the faster TMS320C40, 40-nanosecond part.

Design Considerations

The 68020 User's Manual (Reference 7) was referenced extensively for this design, which covers a complete examination of the VIC068 and VAC068 and extends to review the 680x0 family bus signals and cycles. An examination of the VIC068 and VAC068 reveals a direct interface to the Motorola 680x0 family data, address, and control signals. The VIC068 and VAC068 are both driven with the familiar 680x0 address and data strobes (PAS*, DS*), which have an asynchronous transfer protocol implemented with the data transfer and size acknowledgment signals DSACK0* and DSACK1*. In addition to these signals, the transfer size signals, SIZ0 and SIZ1, are an integral part of the 680x0's dynamic bus sizing capability and, with the lower address lines, encode the size of the transfer in progress. During transfer, the function code signals (FC0–FC2) provide additional information of importance in multi-user environments. Bus arbitration is an integral part of the 680x0 via the bus request (BR*), bus grant (BG*), and bus grant acknowledge (BGACK*) signals and are used directly by the VIC068. Finally, like many other general purpose microprocessors, bus cycles for the 680x0 are several clock cycles long.

Although the VIC068 and VAC068 are driven by, and can drive, the familiar 680x0 bus signals, a quick examination of the TMS320C40 bus signals shows little similarity to the 680x0 family. The TMS320C40 provides a bus protocol 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 Reference 1, the TMS320C40 provides a 32-bit address range divided into two identical 31-bit address, 32-bit data buses termed local and global. The TMS320C40 executes single cycle instructions and relies on a multistage pipeline for execution speed. Detailed bus status is provided for each cycle via the STAT lines, which provide information regarding the type of instruction and access. Individual control lines are provided for three-stating the address, data, and control bus(es). A read-modify-write signal is not provided (as it is on the 680x0). However, an instruction-driven LOCK* signal is available. Each cycle is controlled by a strobe (STRBx*) signal in conjunction with the corresponding read/write (R/Wx*) strobe. One of the TMS320C40's many features is the range of configuration options for this external interface. The TMS320C40 has evolved from its earlier floating-point counterparts and provides a truly flexible interface via the local and global bus configuration control registers.

High-Level Architecture

The high-level architecture for the card places 20-nanosecond (ns), high-density, 4-megabit (128Kx32) Cypress CMOS SRAM modules on both local and global buses of the TMS320C40. (The size of the memory array should not impact the TMS320C40-to-VIC068/VAC068 interface design.) The global side is designated as program memory and the local side as data memory for the application. 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, the A24 VME spectrum is placed on the global (program) side, segmenting the local side I/O activity, the critical path for the application, from all VMEbus activity. (Note that the interface documented herein can be used on either side due to the symmetry of the

global and local buses.) In addition to programming SRAM on the global side, two 128Kx16 EPROMs for embedded program store are also placed, with the boot load feature of the TMS320C40 applied.

Because the design is limited to VMEbus A24 addressing, its spectrum fits nicely anywhere in the TMS320C40 global side address spectrum, from 08000 0000h to 0FFFF FFFFh. Therefore, VMEbus master access is memory mapped into the TMS320C40 global side address range. When an access occurs in this predefined A24 range, the TMS320C40 bus signals are transformed into 680x0 bus signals. These 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" (References 8,9) of the interface, it is known 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 TMS320C40 is flat in its byte endianess (32-bit word addressable only) and little-endian bit ordered. Therefore, no swapping is done on the interface as 32-bit word transfers (between processors) maintain D0 as the least significant bit. This forces the designer to tradeoff transfer speed with a wider range of transfers (byte, word, and three byte) than inherent to the TMS320C40 (longword). Transfers are limited to D32. In order to provide transfers of all sizes, additional set-up and/or decoding would have to be done prior to/during the transfer in progress.

Hardware Description

After examining the VIC068/VAC068 interface and capabilities and comparing them with the TMS320C40, a prototype design was initiated. Based on the above discussion, the strategy is to map from the given set of TMS320C40 bus signals to a set of 680x0-like signals driving their counterparts on the VIC068 and VAC068 for master cycles. (Note: the TMS320C40 is reading from or writing to the VMEbus.) Not only can the TMS320C40 initiate VMEbus cycles as a bus master, the card can also respond to slave cycles. Most often, slave access is used in terms of access to shared memory on the (slave) card. To accomplish this on the TMS320C40-based VME card, a set of signals is required to respond to bus requests from the VIC068/VAC068 and an additional set is required to "hold off" the TMS320C40 global side during such transfers.

To accomplish this transformation of signals, programmable logic is applied. It is desirable to keep the design time to a minimum while maintaining the most flexible or programmable design. Based on this, Cypress's CY7C335 universal synchronous EPLDs (Reference 3) are used. These devices are field programmable and optimized for state machines. The 335 has 12 input macrocells, 12 output macrocells, 256 product terms, 4 buried registers, and operates to a maximum frequency of 100 megahertz (MHz). Development tools for these EPLDs include *Warp2*™, a VHDL compiler from Cypress, and Data I/O's ABEL™ version 4.0 or better, using fitters available on the Cypress Bulletin Board (408-943-2954).

Reset Circuitry

The VIC068 *must* receive a global reset in order to function correctly. The global reset should occur after the power supply and the 64-MHz (U4) oscillator have stabilized and before any interaction with the chip is attempted. The VIC requires both the leading and trailing edges of IRESET*/IPL0, as shown on page 14-32 of the Users Guide and as discussed in Chapter 12.

To implement this function, an R/C network along with a pair of Schmitt Trigger inverters are used to create a power-up signal. The R/C values are not given since they will be a function of the system power supply and oscillator power-up delay time. The power-up signal is supplied to U12, a 22V10 that is programmed as a state machine to create the required waveforms. (Part of U12 is also used for address decoding.) After the power up delay is complete, the power-up signal goes High, which causes U12 to drive IRESET Low. The state machine waits for the VIC to respond by driving RESET Low. It then drives IPL0 Low for two clock cycles indicating a global reset is to take place. IPL0 is returned to a High state followed by the IRESET signal. If the VIC is configured as a VMEbus system controller, a global reset will cause the VIC to drive the SYSRESET line for 200 ms, as required by the VMEbus specification. The RESET output on the VIC068 is used to reset both the TMS320C40 as well as the VAC068 and all programmable logic on-board.

Address Bus Decoding

The VIC068/VAC068 interface, and consequently the VMEbus itself, is mapped into the TMS320C40 global side at 0D000 0000h. In this application, the global side is divided into two halves via the STRB ACTIVE field in the TMS320C40 (global) memory control register. Zero-wait state devices (fast SRAM) are placed in the lower half and slower memory (EPROM) and peripherals (the VIC068/VAC068 pair) are placed in the upper half. Therefore, the TMS320C40 addresses program memory via STRB0 and addresses the VMEbus via STRB1. As shown in the accompanying schematics, U12 is a 22V10 PAL used for $\overline{\text{STRB1}}$ address bus decoding. In particular, a Cypress PAL22V10D-7 was used. This PAL is used to decode each (global) TMS320C40 $\overline{\text{STRB1}}$ bus cycle using the TMS320C40 H1 clock. Cycle type decoding is accomplished fully via the STAT lines (versus simply using the R/W strobe) and allows for future expansion/reconfiguration if required. As shown, the $\overline{\text{STRB1}}$ range is divided into 8 distinct segments via use of GA28–GA30. (GA31 is implicitly a logic 1.) Outputs of the decoding operation are (VMEbus) master write (MWR*), master read (MRD*), VIC068/VAC068 register write (RWR*), register read (RRD*) and EPROM read (GPROM*). As found in the VIC068/VAC068 documentation, the VAC068 is hard-wired, starting at address 0FFFD 0000h. It also provides for VIC068 selection, starting at address 0FFFC 0000h. A memory map for the global side, as decoded by the 22V10 PAL, 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 PAL, the sequencers shown provide the signals required for both master and slave cycles. U13 is the first of three sequencers and accomplishes overall bus control providing enable signals for global bus access by the TMS320C40 (GBE*), master cycle sequencer output (MOE*), slave cycle sequencer output (SOE*), VMEbus slave local bus grant (LBG*) and a ready signal for the TMS320C40 (GRDY1*). Notice that a full complement of inputs are presented to the bus control sequencer. This is done to accommodate all possible cycles and to allow reconfiguration without hardware changes. While the TMS320C40 H3 clock (20 MHz) is used here, this is not an absolute requirement as the array of sequencers operate asynchronously once a master or slave cycle begins. The use of H3 here, however, does simplify the sequencer code because the H3 clock serves as a convenient reference to the TMS320C40 cycle in progress.

A master cycle begins with U12 generating a master read or write signal or either 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. Termination of a master cycle ends with the assertion of the acknowledge signals DSACK0* and DSACK1* and/or the local bus error signal (LBERR*). All are generated by the VIC068 in response to acknowledge signals provided over the VMEbus. The sequencer responds to these signals by providing the ready signal for this TMS320C40 STRB1 access (RDY1*) by asserting GRDY1*. 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 called out in Reference 1: ready is High between accesses.

Slave cycles are initiated by the assertion of local bus request (LBR*) by the VIC068. With this asserted, U13 provides bus control by first disabling the TMS320C40 global bus (deasserting GBE*), then disabling the master bus cycle generation sequencer outputs (U14 MOE*), followed by enabling the outputs on the slave bus cycle signal generation sequencer (SOE*), U15. Given that the bus has been successfully "seized", the local bus grant signal (LBG*) is asserted. Slave cycles are terminated by deassertion of the local bus request input.

Master Bus Cycle Generation

The master bus cycle generation sequencer U14 runs in tandem with the bus control seqencer U13. The sequencer code found in U13 and U14 results from one common state diagram. It is necessary to split these functions up due to limitations of the number of outputs per sequencer. Therefore, the inputs to U14 are identical to those found on U13. Master bus cycles proceed according to the appropriate cycle (read or write) definition found in Reference 7. The function code lines are driven to a supervisory state, giving the widest possible audience, supervisory data. Termination of a master cycle ends with the assertion of the acknowledge signals DSACK0* and DSACK1* and/or the local bus error signal (LBERR*) as described above. Note that VIC068/VAC068 register accesses are also master accesses in the address range(s) specified above. While the sequencer code does not initiate read-modify-write cycles, it is easy to see how the use of the TMS320C40 GLOCK* input could be used to accomplish this.

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 (covered in the next section). As shown, 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 (LBG*) by U13 indicates the absence of the TMS320C40 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 GSTRB0* and GR/W0 are driven to provide access to the shared global SRAM. Following this, acknowledgment is provided via DSACK0* and DSACK1*, ending the slave cycle. Note that while VAC068 documentation states that its DSACK signals can be three-stated on the assertion of LAEN, this was not the case with this configuration. Therefore, U8A was required to artificially three-state those signals so that the slave sequencer could control the data acknowledgment.

VIC068/VAC068 Software Initialization

The VIC068/VAC068 pair register set can be overwhelming at first glance, but very few registers require attention prior to using the pair for either master or slave operations. The VAC068 should be initialized first as this controls both master and slave address mapping. With that complete, the VIC068 is programmed. Fine tuning of the interface can be performed using the programmable delay registers for the interface after initial capability is verified. As the VIC068/VAC068 was programmed using C, vic.h and vac.h header files were developed which provide base and offset definitions for the complete register set of each device.

Before programming the VIC068/VAC068 pair, the VAC068 must be brought out of its initial "Force EPROM" mode which asserts EPROMCS* for all accesses. This is accomplished by reading from the EPROM space beginning at 0FF00 0000h. The address decoding PAL U12 does not provide for access to this range. However, a dummy access to this region can be initiated by manipulating the TMS320C40 (global) memory interface control register. The register is set to provide zero-wait state, internal ready dependent (only) accesses to the appropriate strobe (STRB1 for this case). With this set in the SWW and WTCNT fields, a read from address 0FF00 0000h is performed. Immediately following this read, the SWW field to external ready accesses is reset and a second read to the VAC068 is performed, this time at the VAC068 register base, 0FFFD 0000h. This read provides the required access to "snap" the sequencers back to their default states.

After the "Force EPROMCS" mode is exited, it is verified that the VAC068 can be addressed by reading the device ID via the VAC068 ID register. With that established, the (slave) SLSEL0 Base Address register and the SLSEL0 Address Mask register followed by the (master) A24 Base Address register are programmed. 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 and, following the successful read of that register, the Address Modifier Source register and the Slave Select 0 Control 0 register are set. This completes the initial programming of the pair. At this time, the SYSFAIL LED, if applicable, may be extinguished by writing to the Interprocessor Communication 7 register. The initial register settings for this 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 EnableVAC
0FFFC 00B4h	VIC Address Modifier	8	03Dh
0FFFC 00C0h	VIC Slave Select 0 Control 0	8	014h

Conclusion

The development of a prototype interface between the TMS320C40 DSP and the Cypress VIC068/VAC068 was accomplished with a minimum amount of programmable logic in the form of simple PALs and sequencers. The result is a reconfigurable, programmable interface for A24/D32 VMEbus master/slave capability. While the initial transfer speed is low, speed gains can be made by increasing the sequencer's clock speed and eliminating unnecessary states in the prototype sequencer code. Read-modify-write cycles can be accomplished with the existing hardware via the use of the TMS320C40 LOCK instruction group. Ultimately, the design documented herein could be encapsulated in an FPGA for both speed and size gains.

References

1. *TMS320C4x Users Guide*. Texas Instruments, 1991.
2. *VIC068A/VAC068A Users Guide*. Cypress Semiconductor, 1992.
3. *Cypress Semiconductor CMOS/BiCMOS Data Book*. Cypress Semiconductor, 1992.
4. Siy, P. F., and W. T. Ralston, "Application of the TI TMS320C40 in Satellite Modem Technology," to be published, to be presented at the Third Annual International Conference on Signal Processing Applications and Technology, November, 1992, Boston, MA.
5. *IEEE Standard for a Versatile Backplane Bus*: *VMEbus*. IEEE, 1987, New York, NY: Wiley-Interscience.
6. Peterson, W. D., *The VMEbus Handbook*, Scottsdale, AZ: VFEA International Trade Association.
7. *MC68020 32-Bit Microprocessor User's Manual*. Motorola, Inc., 1984
8. Henessey, J. L., and D. A. Patterson, 1990, *Computer Architecture A Quantitative Approach*, San Mateo, CA: Morgan Kaufmann Publishers, Inc.
9. Dewar, R. B. K, and M. Smosna, 1990, *Microprocessors A Programmers View*, New York, NY: McGraw-Hill, Inc.

Appendix A. Address Bus Decoder - ABEL Source

```
module          U12
title           'Global Bus Decode
Revision        1.0
Part            Cypress PAL22V10D-7
Abel Version    4.3
Project         C40 I/O Board'

     U12 device 'P22V10';

"Inputs"
clk, reset                pin 1,2;    "clock, reset"
gstat0,gstat1             pin 3,4 ;   "C40 status"
gstat2,gstat3             pin 5,6;    "C40 status"
ga28,ga29,ga30            pin 7,8,9;  "C40 address"
gstrb1                    pin 10;     "C40 strobe 1"
oute                      pin 11;     "output enable
pureset                   pin 13;     "output enable"

"Outputs"
ipl0,ireset,tmp1          pin 17,18,16 istype 'reg,invert';
mrd,mwr                   pin 19,20;  "master read & write"
rrd,rwr                   pin 21,22;  "register read & write"
gprom                     pin 23;     "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"
power_up = [ireset,ipl0,tmp1];          "reset

" state definations for power up reset sequence.
"  [ireset,ipl0,tmp1]
s0 = [1,1,1] ; "initial state
s1 = [0,1,1] ; "ireset asserted
s2 = [0,0,1] ; "ireset/ipl0 asserted
s3 = [0,0,0] ; "hold for extra clock
s4 = [0,1,0] ; "de-assert ipl0
s5 = [1,1,0] ; "de-assert ireset
s6 = [1,0,0] ; "should never get here
s7 = [1,0,1] ; "should never get here

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

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

Appendix A. Address Bus Decoder - ABEL Source (continued)

```
“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;

“ power up reset state equations
state_diagram power_up

“ Initial power up state
state s0:
if (pureset) then s1
else s0;

“ assert ireset
state s1:
if (!pureset) then s0
else if (!reset) then s2
else s1;

“assert ipl0
state s2:
if (!pureset) then s0
else s3;

“keep both asserted for extra clock
state s3:
if (!pureset) then s0
else s4;

“de-assert ipl0
state s4:
if (!pureset) then s0
else s5;

“de-assert ireset and stop
state s5:
if (!pureset) then s0
else s5;

“check on indeterminate states
state s6:
if (!pureset) then s0
else s6;

“check on indeterminate states
state s7:
if (!pureset) then s0
else s7;
```

Appendix A. Address Bus Decoder - ABEL Source (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 U12
```

Appendix B. Bus Control Sequencer - ABEL Source

```
module          U13C
title           'C40 Bus Control
Revision        1.0
Part            CY7C335
Abel Version    4.3 using Cypress fitter
Project         TMS320C40 I/O Card '

U13C device 'p335';

"Inputs"
clk, reset                pin 1,13;          "clock, reset"
!mrd,mwr,rrd,rwr,gprom     pin 24,11,10,9,12; "decoded cycle"
mwb,lbr                   pin 7,6;           "master/slave requests"
dedlk                     pin 5;             "m/s deadlock"
dsack0,!dsack1,!lberr     pin 4,28,15;       "cycle responses"
!glock                    pin 26;            "C40 lock"
oe                        pin 14;            "output enable"

"Outputs"
lbg       pin 27      istype 'reg_RS,invert' ;"slave grant"
gbe       pin 16      istype 'reg_RS,invert' ;"C40 g bus enable"
soe,moe   pin 17,18   istype 'reg_RS,invert' ;"pls oe(s)"
grdy1     pin 19      istype 'reg_RS,invert' ;"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 node 34,33,32,31;
P0   pin  25 istype 'reg,invert';
P4,P3,P2,P1  istype 'reg';

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];
 S20 = [1,0,1,0,0];
 S21 = [1,0,1,0,1];
```

Appendix B. Bus Control Sequencer - ABEL Source (continued)

```
 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"
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"

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;
```

Appendix B. Bus Control Sequencer - ABEL Source (continued)

```
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. Bus Control Sequencer - ABEL Source (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;  “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;  “not ready”
 ENDWITH;

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

Appendix B. Bus Control Sequencer - ABEL Source (continued)

```
@page
“Local Bus Request”

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;
```

Appendix B. Bus Control Sequencer - ABEL Source (continued)

```
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;

@page
“make sure there are no undefined states”
state S21: goto S0;
state S22: goto S0;
state S23: goto S0;
state S24: goto S0;
state S25: goto S0;
state S26: goto S0;
state S27: goto S0;
state S28: goto S0;
state S29: goto S0;
state S30: goto S0;

“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 will not work with beta version of
“ Abel 4.3
test_vectors

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

Appendix B. Bus Control Sequencer - ABEL Source (continued)

```
[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 resetstate”
[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 rdy 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 rdy 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 rdy for nxt”
[C,1,1,1,1,1,1,1,0,1,1,1,1,1,1,0]->[S16,1,1,1,1,1];“21slaverequest’
[C,1,1,1,1,1,1,1,0,1,1,1,1,1,1,0]->[S17,1,0,1,1,1];“22 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];“23 slave grant”
[C,1,1,1,1,1,1,1,0,1,1,1,1,1,1,0]->[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];“25rescnd grant”
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0]->[S20,1,1,1,1,1];“26disbl sl pls”
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0]->[S0, 1,1,0,0,1];“27end sl acces”
[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];“33rescnd grant”
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0]->[S20,1,1,1,1,1];“34disbl sl pls”
[C,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0]->[S0, 1,1,0,0,1];“35end sl acces”

end U13C
```

Appendix C. Master Cycle Generation Sequencer - ABEL Source

```
module          u14a
title           'C40 Bus Control
Revision        1.0
Part            CY7C335
Abel Version    4.3 using Cypress fitter
Project         TMS320C40 I/O Card '

U14a device 'p335';

"Inputs"
clk, reset                pin 1,13;       "clock, reset"
mrd,mwr,rrd,rwr           pin 12,11,10,9; "decoded cycle"
mwb,lbr,gprom             pin 7,6,5 ;     "master/slave requests"
dedlk                     pin 4;        "m/s deadlock"
!dsack0,!dsack1,lberr     pin 28,27,2;  "cycle responses"
glock                     pin 3;        "C40 lock"
oe                        pin 14;        "output enable"

"Outputs"
pas  pin 18    istype 'invert,reg_RS';"68K address strobe"
ds   pin 16    istype 'invert,reg_RS';"68K data strobe"
rw   pin 17    istype 'invert,reg_RS';"68K read/write bar"
rmc  pin 15    istype 'invert,reg_RS';"68K read-mod-write"
siz0 pin 19    istype 'invert,reg_RS';"68K size 0"
siz1 pin 20    istype 'invert,reg_RS';"68K size 1"
fc1  pin 23    istype 'invert,reg_RS';"68K function 1"
fc2  pin 24    istype 'invert,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  node 34,33,32,31 istype 'reg';
P0           pin  25          istype 'reg,invert';
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];
 S20 = [1,0,1,0,0];
```

Appendix C. Master Cycle Generation Sequencer - ABEL Source (continued)

```
 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"
rwmem  pin 26 istype 'reg_RS,invert'; "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;
```

Appendix C. Master Cycle Generation Sequencer - ABEL Source (continued)

```
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;

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;
```

Appendix C. Master Cycle Generation Sequencer - ABEL Source (continued)

```
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”
 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;
```

Appendix C. Master Cycle Generation Sequencer - ABEL Source (continued)

```
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;

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;
```

Appendix C. Master Cycle Generation Sequencer - ABEL Source (continued)

```
@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;

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”
```

Appendix C. Master Cycle Generation Sequencer - ABEL Source (continued)

```
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;

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;
```

Appendix C. Master Cycle Generation Sequencer - ABEL Source (continued)

```
@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 power up”
[1,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”
[0,X,X,X,X,X,X,X,X,X,X,X,X,0] -> [S31,X,X,X,X,X,X,X,X,X];
“3 reset state”
[C,0,X,X,X,X,X,X,X,X,X,X,X,0] -> [S0, 1,1,0,0,0,1,1,1,1];
“4 master read”
[C,1,1,1,1,1,0,1,1,1,1,1,1,1,0] -> [S2, 1,1,0,0,0,1,1,0,0];
“5 mwb  asserted”
[C,1,1,1,1,1,1,0,1,0,1,1,1,1,0] -> [S3, 1,1,0,0,0,1,1,0,0];
“6 data acked”
[C,1,1,1,1,1,1,0,1,0,0,0,1,1,0] -> [S4, 1,1,0,0,0,1,1,0,0];
“7 ready for  nxt”
[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];
“8 master  write”
[C,1,1,1,1,1,0,1,1,1,1,1,1,1,0] -> [S8, 0,1,0,0,0,1,0,1,1];
“9 assert pas”
[C,1,1,1,1,1,0,1,1,1,1,1,1,1,0] -> [S9, 0,1,0,0,0,1,0,1,0];
```

Appendix C. Master Cycle Generation Sequencer - ABEL Source (continued)

```
"10 assert ds"
[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];
"11 mwb"
[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];
"12 data ackd"
[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];
"13 ready for  next"
[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];
"14 reg read"
[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];
"15 assert  strobes"
[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];
"16 data ackd"
[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];
"17 ready for  nxt"
[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];
"18 reg write"
[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];
"19 assert pas"
[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];
"20 assert ds"
[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];
"21 data ackd"
[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];
"22 ready for  next"
[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];

end u14a
```

Appendix D. Slave Cycle Generation Sequencer - ABEL Source

```
module          u15a
title           'C40 Bus Control
Revision        1.0
Part            CY7C335
Abel Version    4.30
Project         TMS320C40 I/O Card '

U15a device 'p335';

"Inputs"
clk, reset      pin 1,13;  "clock, reset"
pas,ds          pin 12,11;  "address,data strobe"
rw,rmc          pin 10,9;  "read/write strobes"
siz0,siz1       pin 7,6;  "bus sizing"
fc0,fc1,fc2     pin 5,4,3;"function codes"
lbg             pin 2;   "local bus grant"
oe              pin 14;   "output enable"

"Outputs"
dsack0    pin 15    istype 'invert,reg_RS'; "data ack 0"
dsack1    pin 17    istype 'invert,reg_RS'; "data ack 1"
lberr     pin 19    istype 'invert,reg_RS'; "bus error"
gstrb0    pin 23    istype 'invert,reg_RS'; "C40 mem strobe"
grw0      pin 25    istype 'invert,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 34,33,32,31    istype 'reg';
sreg = [P3,P2,P1,P0];
 S0 = [0,0,0,0];
 S1 = [0,0,0,1];
 S2 = [0,0,1,0];
 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"
!rwmem pin  27 istype 'reg_RS,invert'; "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"
```

Appendix D. Slave Cycle Generation Sequencer - ABEL Source (continued)

```
@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;

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;
```

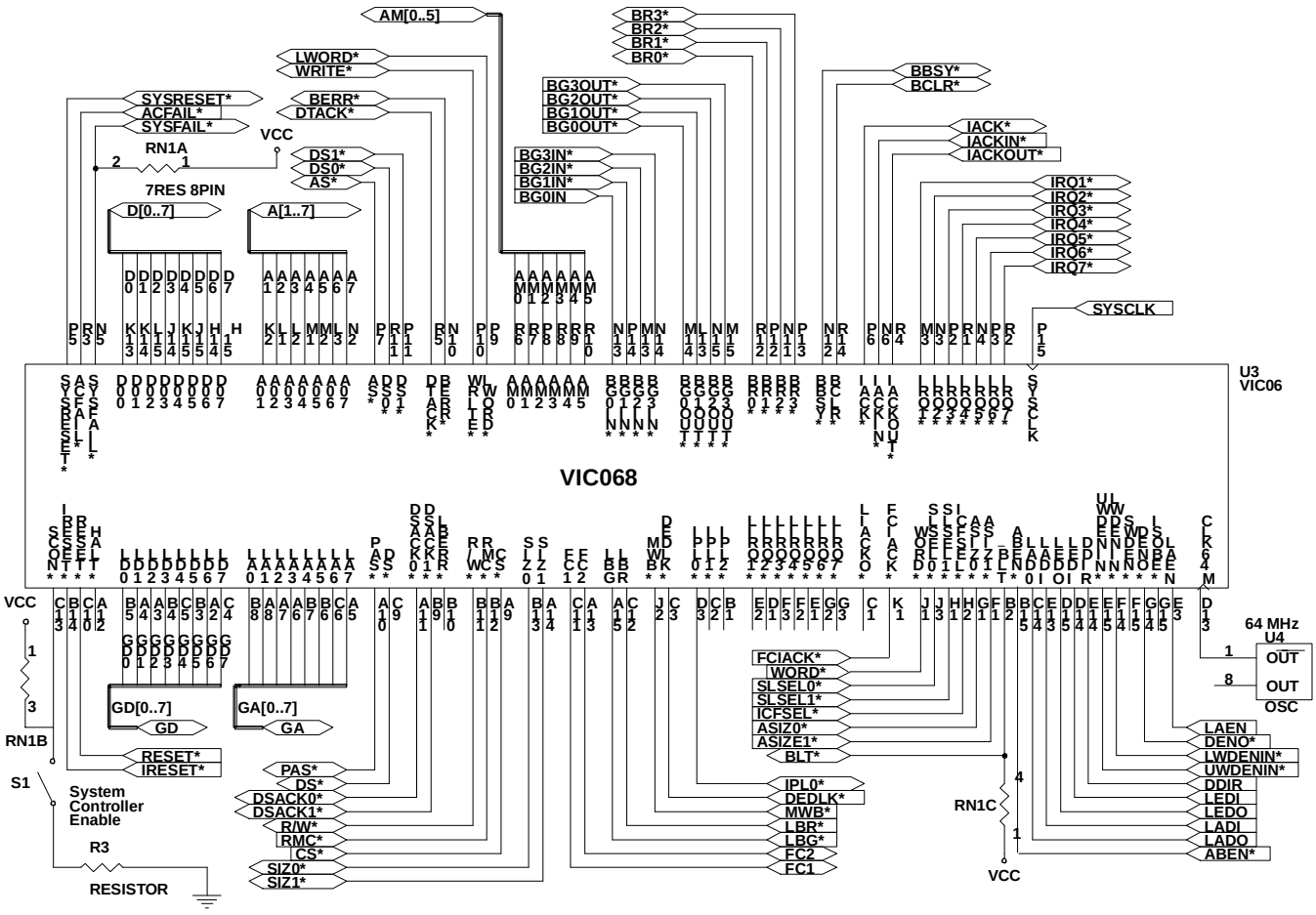
Appendix D. Slave Cycle Generation Sequencer - ABEL Source (continued)

```
“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;

@page
test_vectors

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

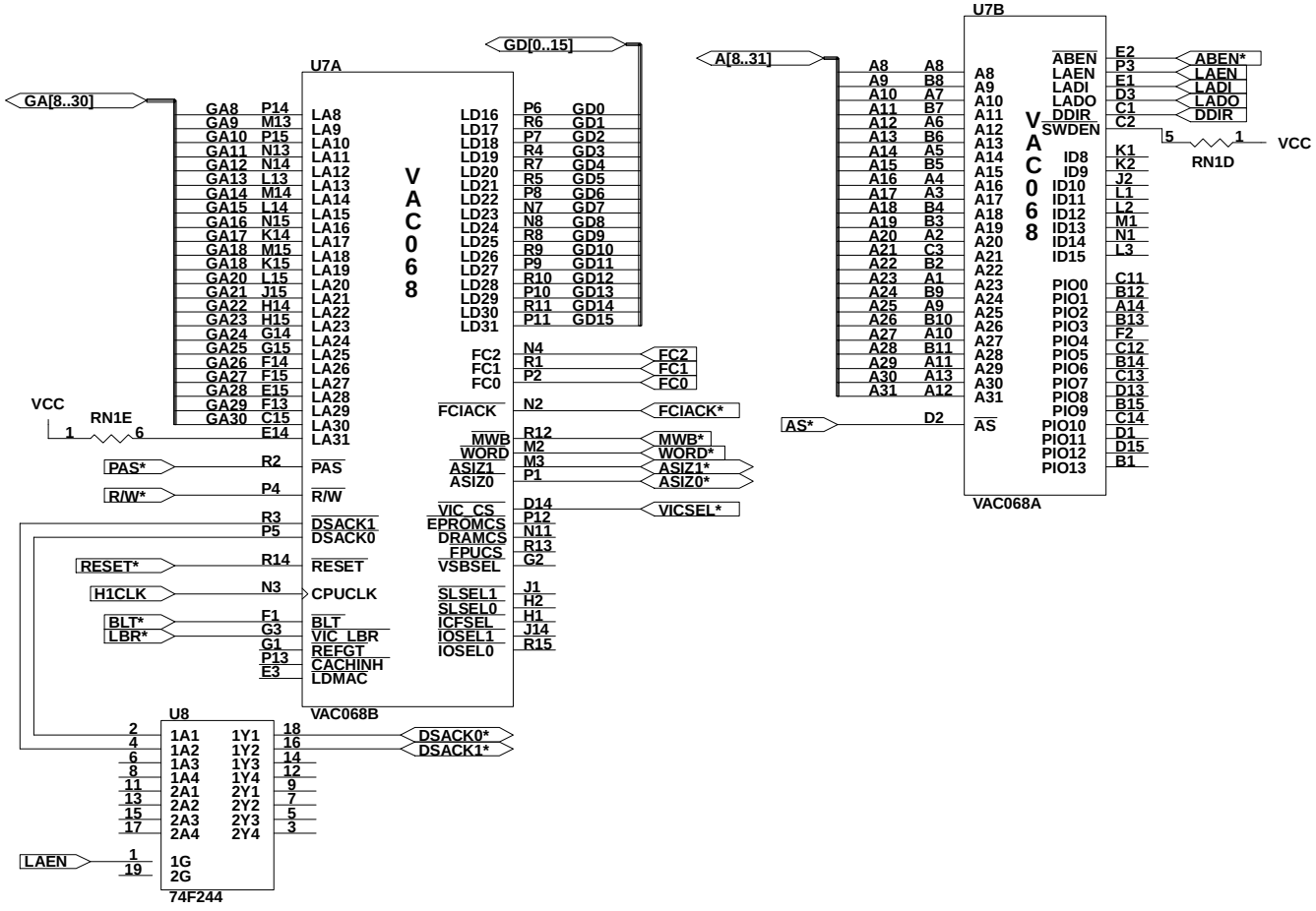
Appendix D. Slave Cycle Generation Sequencer - ABEL Source (continued)

```
[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 wtfor pas rel”
[C,1,1,1,1,1,1,1,X,X,X,1,0]->[S0, 1,1,1,1,1,1];“9 dne, rel gstrb”
[C,1,1,1,0,1,1,1,X,X,X,0,0]->[S1, 1,1,1,1,1,1];“10 slav wrte,lbg”
[C,1,0,1,0,1,0,0,X,X,X,0,0]->[S1, 1,1,1,1,1,1];“11 pas assert”
[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 asert strob”
[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 wtfor pas rel”
[C,1,1,1,1,1,1,1,X,X,X,1,0]->[S0, 0,1,1,1,1,0];“16done,rel gstrb”
[C,1,1,1,1,1,1,1,X,X,X,0,0]->[S1, 0,1,1,1,1,0];“17 slav 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 & ds,r/w asrt”
[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 wtfor pas rel”
[C,1,1,1,1,1,1,1,X,X,X,1,0]->[S0, 1,1,1,1,1,1];“23done,rel gstrb”
[C,1,1,1,1,1,1,1,X,X,X,0,0]->[S1, 1,1,1,1,1,1];“24 bad acess,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 & ds, error”
[C,1,0,0,1,1,0,1,X,X,X,0,0]->[S9, 1,1,1,0,1,1];“27 wtfor pas rel”
[C,1,1,1,1,1,1,1,X,X,X,1,0]->[S0, 1,1,1,1,1,1];“28done,rel lberr”
[C,1,1,1,0,1,1,1,X,X,X,0,0]->[S1, 1,1,1,1,1,1];“29 slv 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 wtfor pas rel”
[C,1,1,1,1,1,1,1,X,X,X,1,0]->[S0, 0,1,1,1,1,0];“35done,rel gstrb”
[C,1,1,1,0,1,1,1,X,X,X,0,0]->[S1, 0,1,1,1,1,0];“36 slav wrte,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 & ds,asrt str”
[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 wtfor pas rel”
[C,1,1,1,1,1,1,1,X,X,X,1,0]->[S0, 0,1,1,1,1,0];“41done,rel gstrb”

end u15a
```

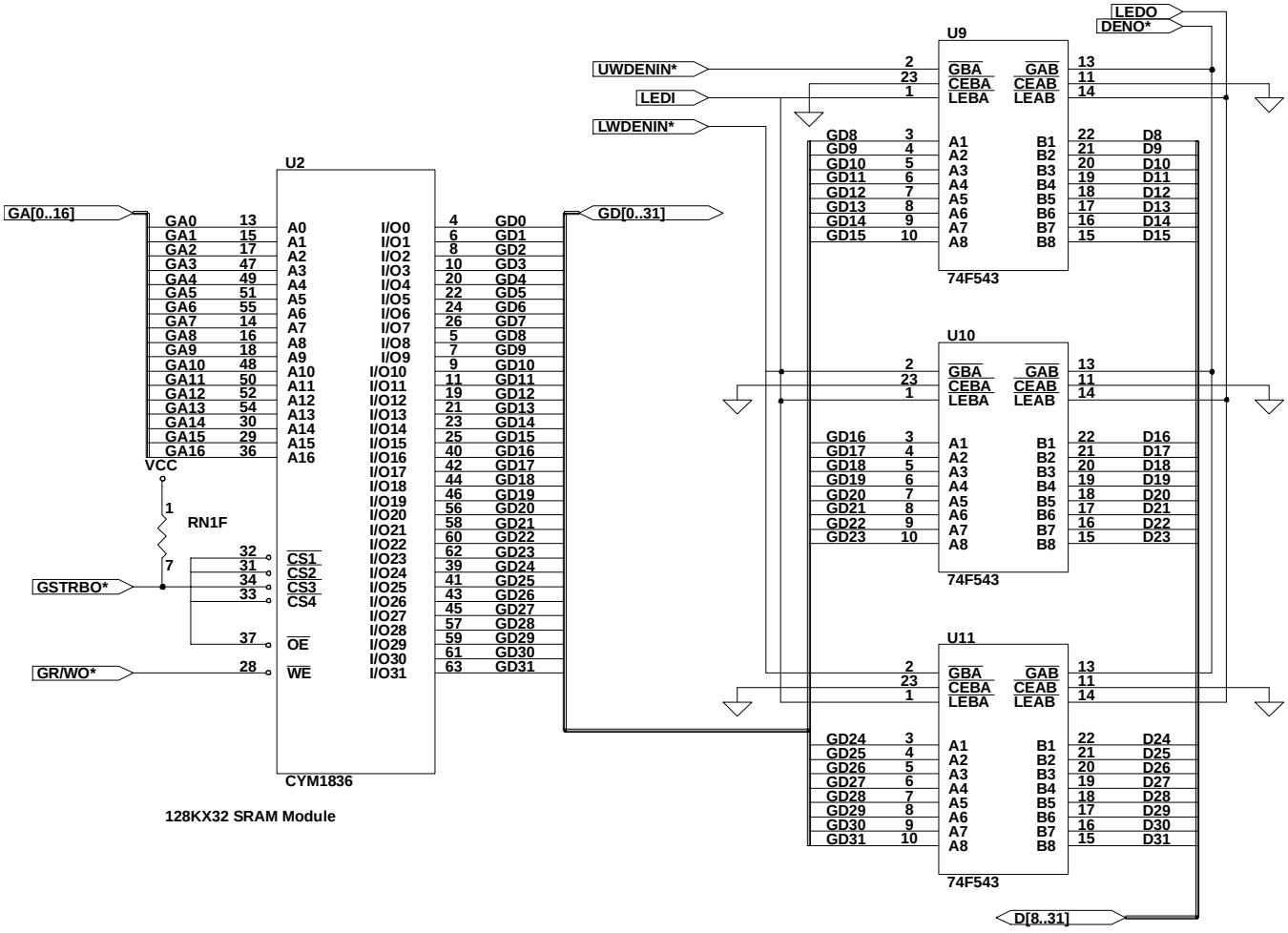
Appendix E. Schematics



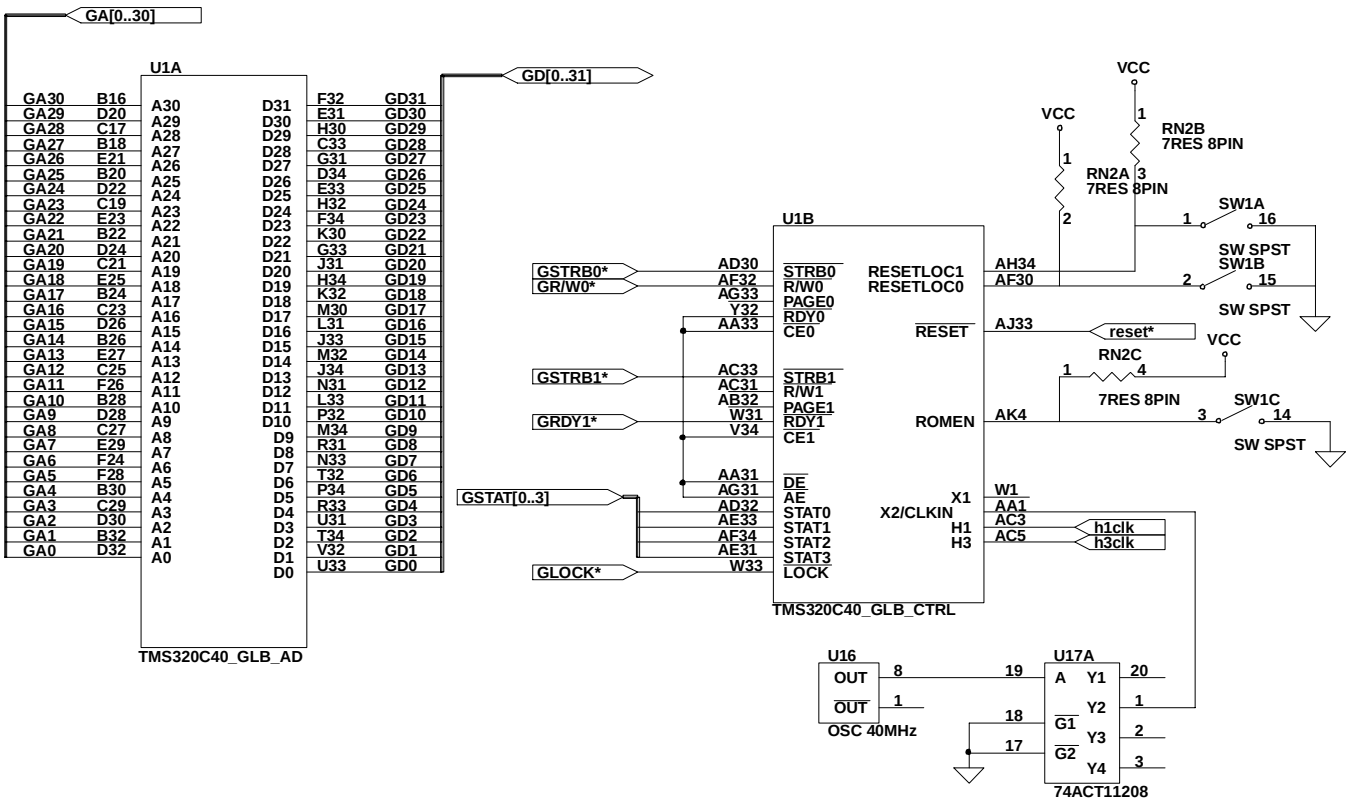
Appendix E. Schematics (continued)



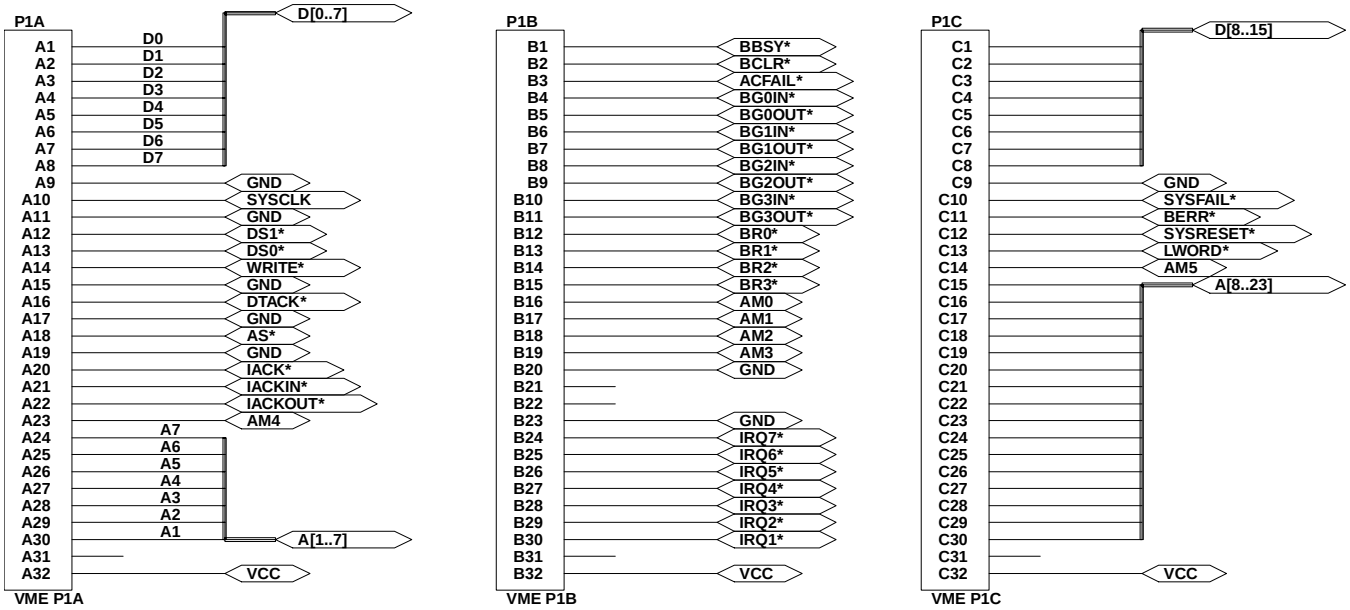
Appendix E. Schematics (continued)



Appendix E. Schematics (continued)

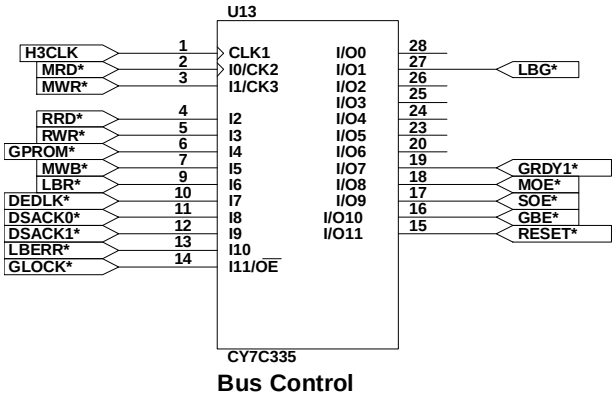


Appendix E. Schematics (continued)

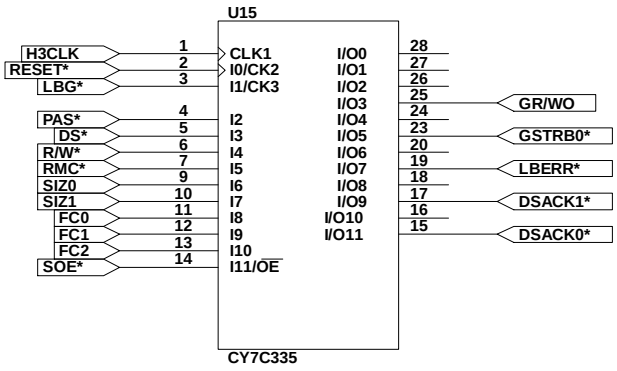


VME P1 CONNECTOR

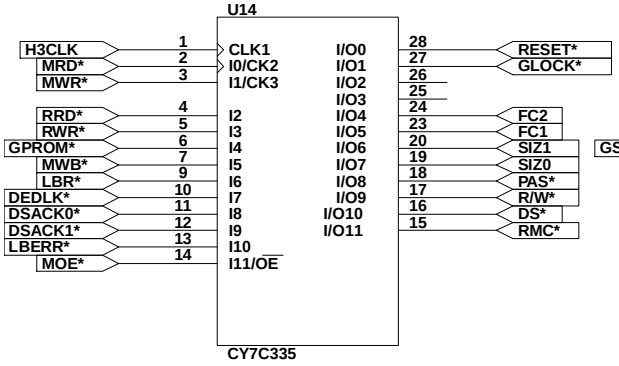
Appendix E. Schematics (continued)



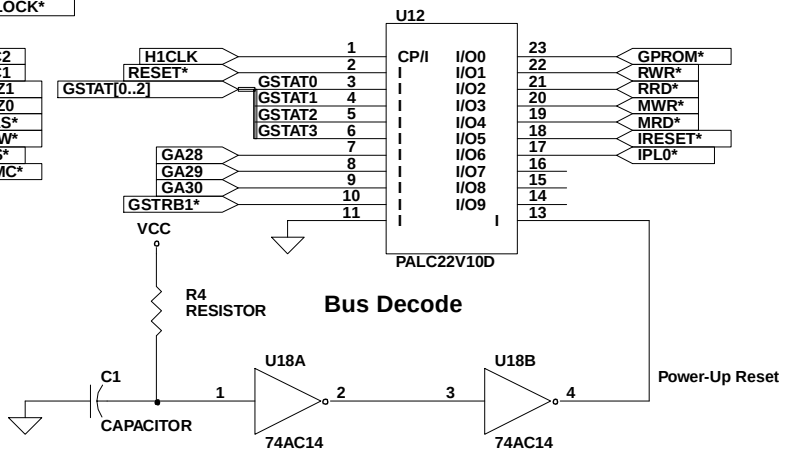
Bus Control



Slave Bus Cycle Generation



Master Bus Cycle Generation



Bus Decode