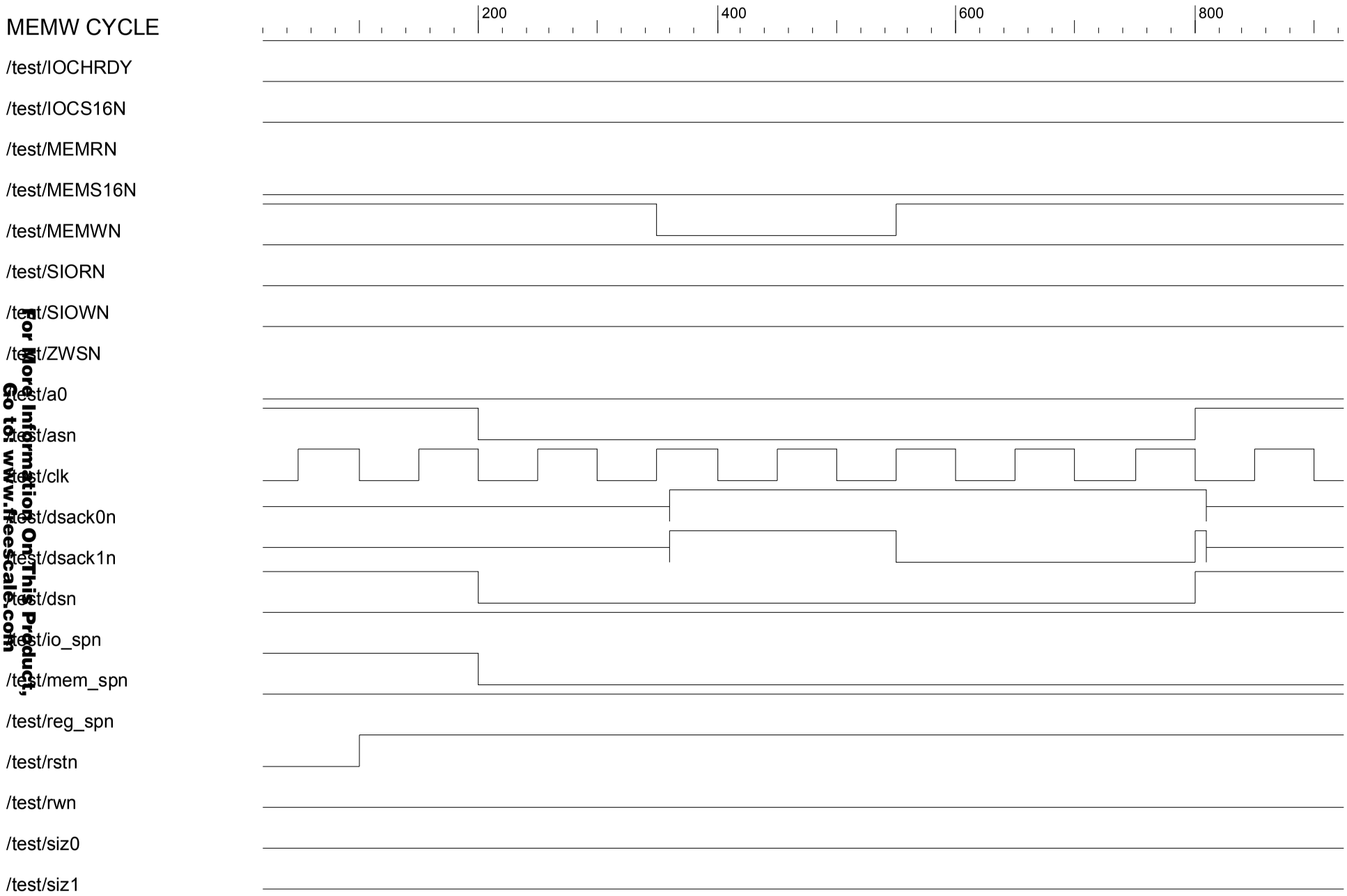
ut /home/mercedes/tomb/PCMCIA/HOST/top.dump - Version 5.2.2 Thu Feb 9 08:51:35 1995 Time0T0-T1Time1T1-T2Time2



EC020/683XX Interface to ISA Bus-type PCMCIA Controller

PRELIMINARY

by
Tom Balph
/602/ 413-2504
ASU-02

Revision History

1. Rev 0.2 3/9/95

1.0 BACKGROUND

Freescale does not presently supply a PCMCIA host controller targeted to the M68XXX family. As an alternative, an activity has been undertaken to evaluate the design problem presented by attaching an Intel-like, ISA Bus PCMCIA host controller to a common M68K bus. The 68K target is an EC020 or CPU32-like bus. This target was chosen in that the bus is common to our general purpose MCU's aimed at the embedded market.

The following design aims at using a commonly available 82365SL register model compatible host controller. These devices are available in many flavors from Cirrus Logic, Vadem, Omega and other vendors. They are typically designed to attach directly to an ISA Bus-compatible interface. They are cost effective and the attachment cost to the 68K bus is not high.

Presently, this paper provides a modeled-only design that still needs hardware verification. The writeup is presented for critique and comment and as a starting point for preliminary evaluation.

2.0 OVERVIEW

The design focuses on generating the required ISA Bus control signals for controller accesses and interrupt request/service requirements. Some PCMCIA controllers support a form of DMA access and additional study is required to evaluate the ISA Bus DMA interface with relation to a device such as the 68340 which has DMA capability. Figure 2-1 shows a block diagram of the interface application. The bus interface logic should fit into a small PAL or PLD. Address decode logic would be very dependent on the system.

Special considerations of this interface include:

- Differences in bus byte ordering
- Use of dynamic bus sizing
- The ISA Bus supports separate I/O and memory address spaces
- The design is based on a nominal 8 MHz system clock
- The ISA Bus is basically a non-handshake bus, while the 68K uses the DSACK handshake
- The 68K read-modify-write cycle is not supported

BLOCK DIAGRAM

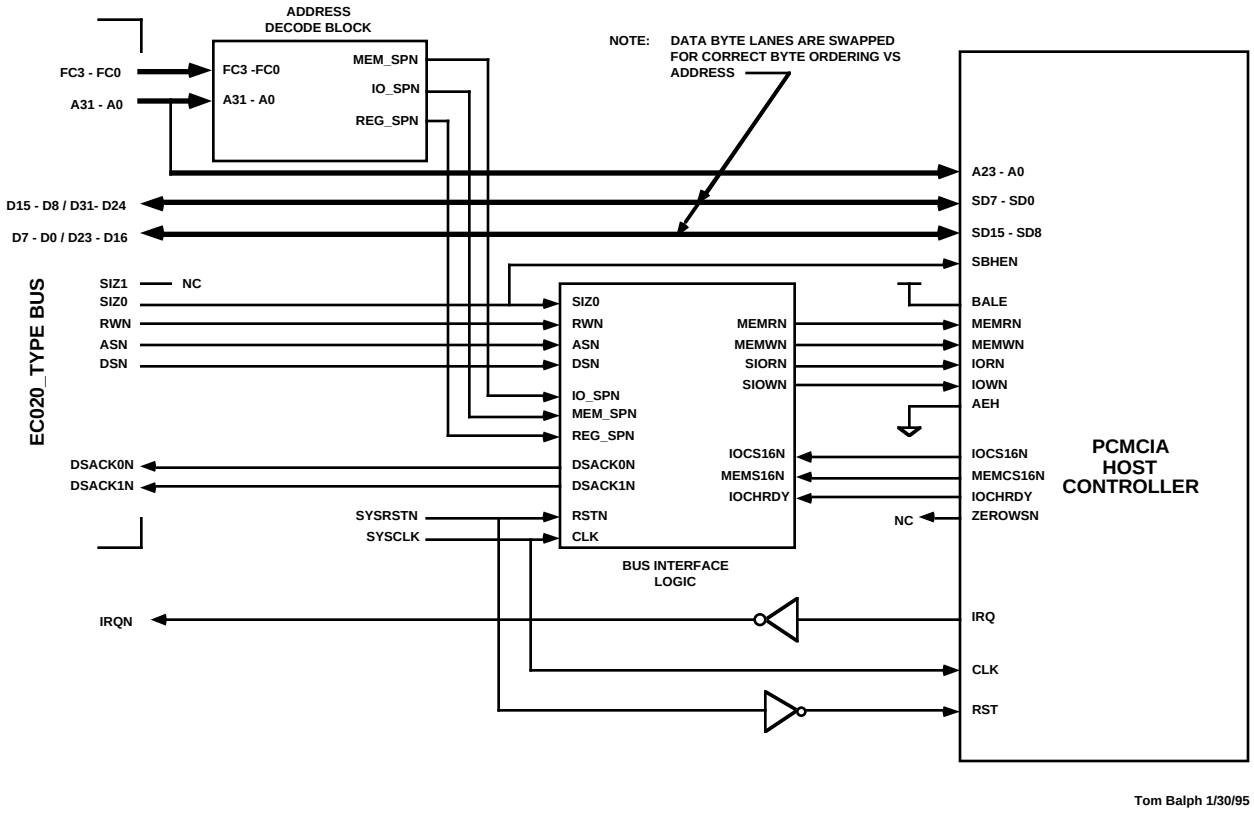


Figure 2-1. Block Diagram of Interface Application

3.0 ADDRESS DECODING

The ISA Bus supports two address spaces which include a 24-bit memory address space and a 16-bit I/O address space. However in reviewing Figure 2-1, one observes that there are three signals routed between the ADDRESS DECODE BLOCK and the BUS INTERFACE LOGIC. This allows for an I/O access to be mapped into one area of the 68K address map (IO_SPN), a memory access to be mapped into a second area of the 68K map (MEM_SPN), and the controller onboard registers to be mapped into a third area of the 68K map (REG_SPN) if desired. Note the following issues:

1. The ISA Bus only supports 24 address lines for memory accesses. Any additional upper 68K address lines must be included in the address decode logic.

2. The ISA Bus only supports 16 address lines for an I/O access (although A16 should be

forced to zero during an I/O access). Upper address lines should be included in the address decode. Typically, I/O accesses cannot target the programmed onboard register addresses because these are redundant with the register access, see Number 3 following this note.

3. Register accesses create an I/O access similar to a standard I/O access. The additional enable is only to let the controller registers reside in a different 68K memory page if desired (if not, the REG_SPN signal can be eliminated). All the onboard registers for this type of PCMCIA controller are indirectly accessed via two 8-bit direct read/write registers in the I/O space. The first register called the Index Register is first written with the index number of the desired onboard register. Once the Index Register is valid, the desired register is accessed through the Data I/O register. (This was done to minimize usage of the I/O address space while still accessing a large number of registers).

   Although different vendors' devices offer variations on a programmable chip select signal, most commonly the above two registers are strapped to be accessed at addresses $3E0_h/3E1_h$ or $3E2_h/3E3_h$. For this reason, no PCMCIA I/O accesses should be normally programmed for addresses on top of these register addresses.

If an integrated MCU such as the 68331 or 68340 is used, the address decode logic may be replaced by programmed chip selects. In this way the additional logic of the ADDRESS DECODE BLOCK can be eliminated.

The user should become familiar with the onboard windowing technique used by these controllers. The PC Card bus (Rev 2.1) is very similar in definition to the ISA Bus in that separate memory and I/O address spaces are supported. The address mapping windows onboard the controller support I/O > I/O accesses and memory > memory accesses between the buses. It is not possible to access PC Card I/O registers via an memory access to the chip. The user in a 68K environment should do their memory mapping accordingly. Also become familiar with the memory page sizes available onboard the controller.

4.0 BYTE ORDERING AND BUS SIZING

The external bus interface common to the EC020, 300 series MCU's, and other processors supports either a 32-bit or a 16-bit data bus and bus sizing. When interfacing into a 16-bit ISA Bus interface, byte ordering and bus sizing must be considered. With byte ordering, the "little endian" (ISA) versus "big endian" (Freescale) dilemma is encountered. As it turns out, simple byte lane swapping solves this problem, and it is accomplished by tying the upper ISA Bus [D15 : D8] byte to the corresponding lower Freescale byte and tying the lower ISA Bus [D7 : D0] byte to the corresponding upper Freescale byte respectively. For all types of accesses, this solution works if the DSACK signals are handled properly.

The byte lane swapping is also affected by the width of the Freescale bus. For an external Freescale 16-bit data bus, the byte lanes are merely swapped. For a 32-bit external Freescale bus, the lower ISA Bus byte lane is wired to the most significant Freescale byte lane [D31 : D24] and the upper ISA Bus byte lane is wired to the next most significant byte [D23 : D16].

Table 4-1 below lists the required signal decoding from the EC020 interface to the ISA type interface as well as the DSACK signal responses for the simple byte swapping to function properly. The DSACK signals are correct for either 16- or 32-bit Freescale buses and misaligned accesses are not allowed:

TRANSFER TYPE	SIZ1	SIZ0	A0	SBEHN	IOCS16N/MEMCS16N	DSACK1	DSACK0
BYTE > BYTE	0	1	0	1	X	1	0
	0	1	1	1	X	1	0
WORD > WORD	1	0	0	0	0	0	1
	1	0	1	NOT ALLOWED			
WORD > BYTE	1	0	0	0	1	1	0
LWORD > WORD	0	0	0	0	0	0	1
	0	0	1	NOT ALLOWED			

TABLE 4-1. Decoded Bus Control Signals for Byte Swapping

SBEHN is active low to the ISA interface to indicate a desired 16-bit access; after observing the table, this signal can be driven from the SIZ0 signal. The two signals IOCS16N and MEMCS16N indicate if the peripheral is capable of a 16-bit access if one is requested (SBEHN asserted low); with IOCS16N asserted for I/O accesses and MEMCS16N asserted for memory accesses. If the 68K requests a word or long word access and the appropriate peripheral signal is not active, the response is a byte access. A word access normally gets a word response, and a long word access normally gets a word response (with a 16-bit data bus).

5.0 BUS INTERFACE LOGIC

The 68K and ISA buses are similar in that both are asynchronous, i.e., not synchronized to a clock. The 68K bus however in normal operation expects a handshake signal (DSACK's) to complete the cycle, whereas the ISA bus normally uses a predetermined cycle time (depending on cycle type and bus width) and gets "stretched" if the peripheral needs a longer access time.

The BUS INTERFACE LOGIC shown in Figure 2-1 generates the required signals and timing to interface the 68K bus control into the ISA Bus control. Timing generation is based upon a nominal 8 MHz system clock which is also used by the controller. The ISA signals generated are the active low cycle command lines MEMRN, MEMWN, IORN, and IOWN. In turn, the ISA signals IOCS16N (active low), MEMCS16N (active low), and IOCHRDY (active high) are inputs to the logic and are used to return the appropriate DSACK signals and stretch the access cycle as needed.

The logic is not designed as a state machine although a serial shift register is used as a time base (125 ns nominal clock period). A bus cycle proceeds as follows:

1. ASN is asserted low - cycle starts.

2. After a one clock wait min (dependent on ASN setup time with respect to clock) to allow address decode settle time -

If	[(IO_SPN asserted or REG_SPN asserted) and RWN not asserted and DSN asserted]	SIORN is asserted
else if	[MEM_SPN asserted and RWN not asserted and DSN asserted]	MEMRN is asserted
else if	[(IO_SPN asserted or REG_SPN asserted) and RWN asserted and DSN asserted]	SIOWN is asserted
else if	[MEM_SPN asserted and RWN asserted and DSN asserted]	MEMWN is asserted
else	no signal asserted	

3. After a two clock wait (sets basic command signal cycle time to 250 ns min) -

If	[IOCHRDY asserted and (IORN asserted or MEMRN asserted)]	DSACK1N and DSACK0N asserted per Table 4-1
else if	[IOCHRDY asserted and (IOWN asserted or MEMWN asserted)]	IOWN or MEMWN negated and DSACK1N and DSACK0N asserted per Table 4-1

   else wait for IOCHRDY asserted (sampled by clock); then go to above “if”.

4. Wait for DSN and ASN negated; then negate DSACK1N and DSACK0N followed by tristate condition and negate IORN or MEMRN as required.

Additional notes on the design include:

1. The ISA Bus normally only holds address lines A23 - A17 stable at the beginning of the cycle and therefore latches these lines onboard chip. All address lines on the 68K bus are stable for the entire cycle so latching is not required. As a result, ISA Bus latch signal BALE is tied “high” always allowing the address line state to pass through.

2. The ISA Bus signal AEN is used to disable I/O resources during a DMA cycle. This signal is

normally tied low when DMA is not supported.

3. SBEN is driven directly from SIZ0.

4. Data bus Data-In Valid time may require some special attention -

   On a memory or I/O read cycle in which the cycle has been stretched (IOCHRDY was not always active), the data setup time to the 68K bus must be checked. Some PCMCIA host controllers have on board buffering and some use external bidirectional buffers. The user must insure that the Data-In Valid time is met with respect to DSACKXN asserted. If the Data-In Valid time is too long after DSACKXN are asserted, then the DSACK signals must be delayed additional time (usually one clock) to allow the data to settle.

5. ISA bus cycle timing -

   The ISA Bus supports several different command cycle times dependent on access type and the data width. The PCMCIA does not have this requirement, so the interface timing was based on a standard minimum cycle time of two clocks or 250 ns. Extending the cycle time to the PCMCIA bus is done onboard the controller (through the controller stretching the cycle via the IOCHRDY signal). As a result of this design approach, the ZEROWSN signal from the controller is not used.

6.0 INTERRUPTS

ISA Bus interrupt requests are normally active high and therefore should be inverted for the 68K environment. The ISA Bus peripherals cannot support a 68K IACK cycle so autovectored interrupts should be used. Also, these PCMCIA controllers usually have several interrupt lines that are intended to be tied directly the ISA Bus IRQ lines. Any of these can be used for one or two interrupts to the bus (as example, one per PC Card). Be sure that the IRQ lines are configured for level-mode to be compatible with 68K requirements.

6.0 DMA (Presently not supported)

An option to the 16-bit Rev 2.1 of the PCMCIA spec includes DMA. The DMA is very similar to the ISA bus DMA with request, acknowledge, and terminal count signals. The MC68340 supports DMA on the external bus and it may be possible to provide this functionality. This is an issue under study.

A.0 BUS INTERFACE LOGIC Verilog HDL Listing

```
// source file:   bus_ctrl.v
//
// Description :  This module defines control logic for interfacing
//                an ISA PCMCIA controller to an EC020/3XX bus
//
// Author:        Tom Balph, Freescale, Phoenix Technology Center
//                Copywrite Freescale, Inc. 2/1/95
// History:       2/1/95
//

module bus_ctrl (SIORN, SIOWN, MEMRN, MEMWN, dsack0n, dsack1n,
                    siz0, asn, dsn, rwn, mem_spn, io_spn, reg_spn,
                         IOCS16N, MEMS16N, IOCHRDY, rstn, clk);

    output  SIORN,  // Active low I/O read cycle command signal
            SIOWN,  // Active low I/O write cycle command signal
            MEMRN,  // Active low memory read cycle command signa
            MEMWN,  // Active low memory write cycle command signa
            dsack0n,  // Active low 68K data and size acknowledge lsb
            dsack1n;  // Active low 68K data and size acknowledge msb

    input   siz0,     // Data transfer size lsb
            asn,      // Active low address strobe
            dsn,      // Active low data strobe
            rwn,      // Read / Write
            mem_spn,  // Active low memory space enable
            io_spn,   // Active low I/O space enable
            reg_spn,  // Active low control register space enable
            IOCS16N,  // Active low 16_bit I/O mode chip select
            MEMS16N,  // Active low 16_bit memory mode chip select
            IOCHRDY,  // Active high I/O channel ready (wait is active low)
            rstn,     // Active low hardware asyn reset
            clk;      // Clock (nominally 8 MHz)

reg q0, q1, q2, q3, cyc_end;

// DELAY GENERATORS ************************************

always @ (posedge clk or negedge rstn) begin
```

```
    if (!rstn) begin
        q0 <= 1'b0;
        q1 <= 1'b0;
        q2 <= 1'b0;
        q3 <= 1'b0;
        end

    else begin
        q0 <= ~asn;
        q1 <= q0 & ~asn;
        q2 <= q1 & ~asn;
        q3 <= q2 & ~asn;
        end
    end

always @ (posedge clk or negedge rstn) begin

    if (!rstn) cyc_end <= 1'b0;
    else cyc_end <= q2 & IOCHRDY;
    end

// GENERATE AT CONTROL SIGNALS *********************************

    wire SIORN = ~((~io_spn | ~reg_spn) & q1 & rwn & ~dsn);
    wire MEMRN = ~(~mem_spn & q1 & rwn & ~dsn);
    wire SIOWN = ~((~io_spn | ~reg_spn) & q1 & ~rwn & ~dsn & ~cyc_end);
    wire MEMWN = ~(~mem_spn & q1 & ~rwn & ~dsn & ~cyc_end);

// GENERATE DSACK SIGNALS***************************

    wire S16N = (mem_spn)? IOCS16N : MEMS16N;
    wire dsack1n_int = ~( ~(siz0 | S16N) & cyc_end & ~dsn);
    wire dsack0n_int = ~( (siz0 | S16N) & cyc_end & ~dsn);

    wire #10 dsack_enb = ~(mem_spn & io_spn & reg_spn) & q1 & ~dsn;

    wire dsack0n = (dsack_enb)? dsack0n_int : 1'bz;
    wire dsack1n = (dsack_enb)? dsack1n_int : 1'bz;

endmodule  // Main module
```