# ANNEXE S5

# DOCUMENTATION du CIO Z8536

**Z8536 CIO**
**Counter/Timer and**
**Parallel I/O Unit**

**ZILOG**

## Features

- Two independent 8-bit, double-buffered bidirectional I/O ports plus a 4-bit special purpose I/O port. I/O ports feature programmable polarity, programmable direction (Bit mode), "pulse catchers," and programmable open-drain outputs.
- Four handshake modes, including 3-Wire (like the IEEE-488).
- REQUEST/WAIT signal for high-speed data transfer.
- Flexible pattern-recognition logic, programmable as a 16-vector interrupt controller.
- Three independent 16-bit counter/timers with up to four external access lines per counter/timer (count input, output, gate, and trigger), and three output duty cycles (pulsed, one-shot, and square-wave), programmable as retriggerable or non-retriggerable.
- Easy to use since all registers are read/write.

## General Description

The Z8536 CIO Counter/Timer and Parallel I/O element is a general-purpose peripheral circuit, satisfying most counter/timer and parallel I/O needs encountered in system designs. This versatile device contains three I/O ports and three counter/timers. Many programmable options tailor its configuration to specific applications. The use of the device is simplified by making all internal registers (command, status, and data) readable and (except for status bits) writable. In addition, each register is given its own unique internal address, so that any register can be accessed in two operations. All data registers can be directly accessed in a single operation. The CIO is easily interfaced to all popular microprocessors.

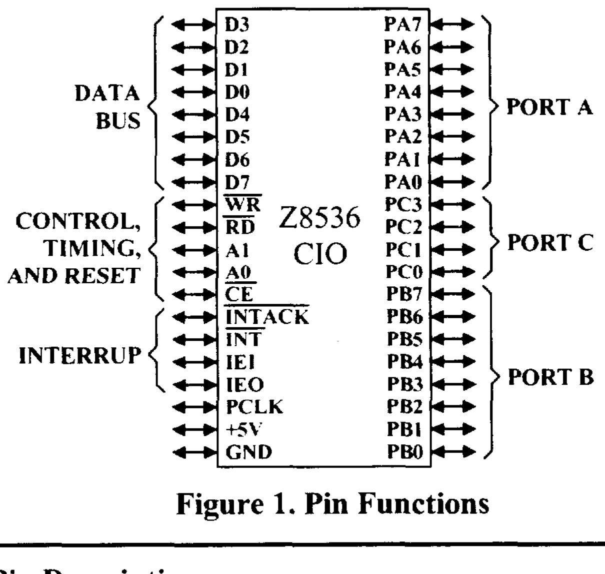


**Figure 1. Pin Functions**

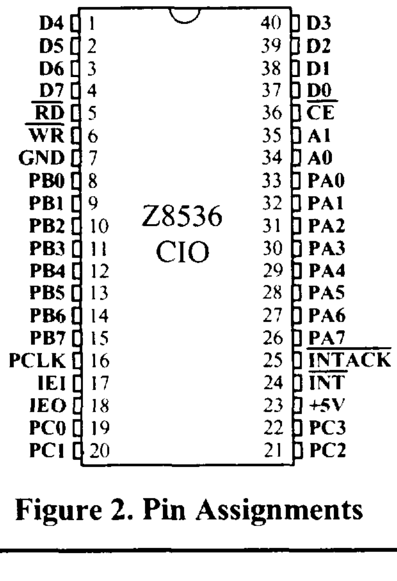


**Figure 2. Pin Assignments**

## Pin Description

**A0-A1.** *Address Lines* (input). These two lines are used to select the register involved in the CPU transaction: Port A's Data register, Port B's Data register, Port C's Data register, or a control register.

**$\overline{\text{CE}}$.** *Chip Enable* (input, active Low). A Low level on this input enables the CIO to be read from or written to.

**D0-D7.** *Data Bus* (Bidirectional 3-state). These eight data lines are used for transfers between the CPU and the CIO.

**IEI.** *Interrupt Enable In* (input, active high). IEI is used with IEO to form an interrupt daisy chain when there is more than one interrupt-driven device. A High IEI indicates that no other higher priority device has an interrupt under service or is requesting an interrupt.

**IEO.** *Interrupt Enable Out* (output, active high). IEO is High only if IEI is High and the CPU is not servicing an interrupt from the requesting CIO or is not requesting an interrupt (Interrupt Acknowledge cycle only). IEO is connected to the next lower priority device's IEI input and thus inhibits interrupts from lower priority devices.

**$\overline{\text{INT}}$.** *Interrupt Request* (output, open-drain, active low). This signal is pulled Low when the CIO requests an interrupt.

**$\overline{\text{INTACK}}$.** *Interrupt Acknowledge* (input, active Low). This input indicates to the CIO that an interrupt Acknowledge cycle is in progress. $\overline{\text{INTACK}}$ must be synchronized to PCLK, and it must be stable throughout the Interrupt Acknowledge cycle.

**PA0-PA7.** *Port A I/O lines* (bidirectional, 3-state, or open-drain). These eight I/O lines transfer information between the CIO's Port A and external device.

**PB0-PB7.** *Port B I/O lines* (bidirectional, 3-state, or open-drain). These eight I/O lines transfer information between the CIO's Port B and external devices. May also be used to provide external access to Counter/Timers 1 and 2.

**PC0-PC3.** *Port C I/O lines* (bidirectional, 3-state, or open-drain). These four I/O lines are used to provide handshake, $\overline{\text{WAIT}}$, and REQUEST lines for Ports A and B or to provide external access to Counter/Timer 3 or access to the CIO's Port C.

**PCLK.** *Peripheral Clock* (input, TTL-compatible). This is the clock used by the internal control logic and the counter/timers in timer mode. It does not have to be the CPU clock.

**$\overline{\text{RD}}$*.** *Read* (input, active Low). This signal indicates that a CPU is reading from the CIO. During an Interrupt Acknowledge cycle, this signal gates the interrupt vector onto the data bus if the CIO is the highest priority device requesting an interrupt.

**$\overline{\text{WR}}$*.** *Write* (input, active Low). This signal indicates a CPU Write to the CIO.

* When $\overline{\text{RD}}$ and $\overline{\text{WR}}$ are detected Low at the same time (normally all illegal condition), the CIO is reset.

## Architecture

The CIO Counter/Timer and Parallel I/O element (figure 3) consists of a CPU interface, three I/O ports (two general-purpose 8-bit ports and one special-purpose 4-bit port), three 16-bit counter/timers, an interrupt-control logic block, and the internal-control logic block. An extensive number of programmable options allow the user to tailor the configuration to best suit the specific application.

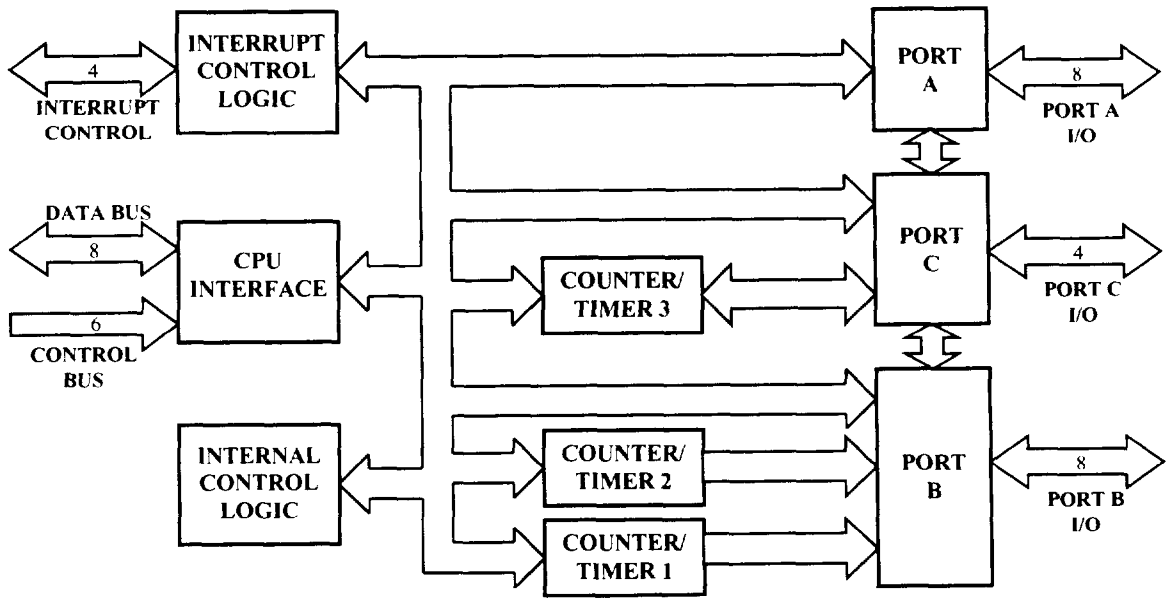


**Figure 3. CIO Block Diagram**

## Functional Description

The following describes the functions of the ports, pattern-recognition logic, counter/timers, and interrupt logic.

**I/O Port Operations.** Of the CIO's three I/O ports, two (Ports A and B) are general-purpose, and the third (Port C) is a special-purpose 4-bit port. Ports A and B can be configured as input, output, or bidirectional ports with handshake. (Four different handshakes are available.) They can also be linked to form a single 16-bit port. If they are not used as ports with handshake, they provide 16 input or output bits with the data direction programmable on a bit-by-bit basis. Port B also provides access for Counter/Timers 1 and 2. In all configurations, Port A and B can be programmed to recognize specific dada patterns and to generate interrupts when the pattern is encountered.

*Bit Port Operations.* In bit port operations, the port's Data Direction register specifies the direction of data flow for each bit. A 1 specifies an input bit, and a 0 specifies an output bit. If bits are used as I/O bits for a counter/timer, they should be set as input or output, as required.

The Data Path Polarity Register provides the capability of inverting the data path. A 1 specifies inverting, and a 0 specifies non-inverting. All discussions of the port operations assume that the path in non-inverting.

The value returned when reading an input bit reflects the state of the input just prior the input data to the read. A 1's catcher can be inserted into the input data path by programming a 1 to the corresponding bit position of the port's Special I/O Control register. When a 1 is detected at the 1's catcher input, its outputs is set to 1 until it is cleared. The 1's catcher is cleared by writing a 0 to the bit. In all other cases, attempted writes to input bits are ignored.

When Ports A and B include output bits, reading the Data register returns the value being output. Reads of Port C return the state of the pin. Outputs can be specified as open-drain by writing a 1 to the corresponding bit of the port's Special I/O Control register. Port C has the additional feature of bit-addressable writes. When writing to Port C, the four most significant bits are used as a write protect mask for the least significant bits (0-4, 1-5, 2-6, and 3-7). If the write protect bit is written with a 1, the state of the corresponding output bit is not changed.

## Programming

The data registers within the CIO are directly accessed by address lines $A_0$ and $A_1$ (Table 3). All other internal registers are accessed by the following two-step sequence, with the address lines specifying a control operation. First, write the address of the target register to an internal 6-bit Pointer Register; then read from or write to the target register. The Data registers can also be accessed by this method.

An internal state machine determines if accesses with $A_0$ and $A_1$ equaling 1 are to the Pointer Register or to an internal control register (Figure 11). Following any control read operation, the state machine is in State 0 (the next control access is to the Pointer Register). This can be used to force the state machine into a known state. Control reads in State 0 return the contents of the last register pointed to. Therefore, a register can be read continuously without writing to the Pointer. While the CIO is in State 1 (next control access is to the register pointed to), many internal operations are suspended- no Ips are set and internal status is frozen. Therefore, to minimize interrupt latency and to allow continuous status updates, the CIO should not be left in State 1.

The CIO is reset by forcing $\overline{RD}$ and $\overline{WR}$ Low simultaneously (normally an illegal condition) or by writing a 1 to the Reset bit. Reset disables all functions except a read from or write to the Reset bit; writes to all others bits are ignored, and all reads return $01_H$. In this state, all control bits are forced to 0 and may be programmed only after clearing the Reset bit (by writing a 0 to it).

| $A_0$ | $A_1$ | Register |
|---|---|---|
| 0 | 0 | Port C's Data Register |
| 0 | 1 | Port B's Data Register |
| 1 | 0 | Port A's Data Register |
| 1 | 1 | Control Register |

**Table 3. Register Selection**

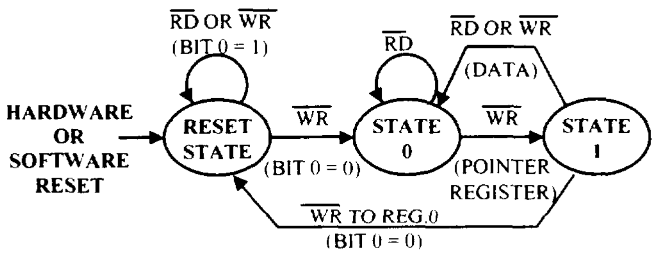

$\overline{RD}$ OR $\overline{WR}$ (BIT 0 = 1)
HARDWARE OR SOFTWARE RESET
RESET STATE
$\overline{WR}$ (BIT 0 = 0)
$\overline{RD}$
STATE 0
$\overline{RD}$ OR $\overline{WR}$ (DATA)
$\overline{WR}$ (POINTER REGISTER)
STATE 1
$\overline{WR}$ TO REG.0 (BIT 0 = 0)

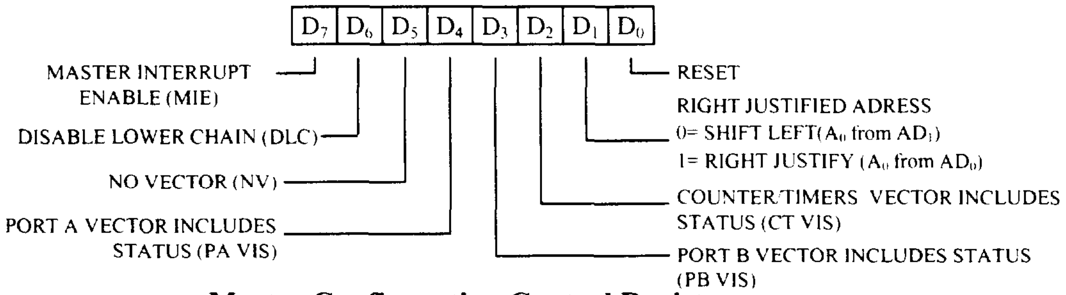
**Figure 11. State Machine Operation**

## Registers

### Master Interrupt Control Register

Addresses: 000000

(Read/Write)


D7 D6 D5 D4 D3 D2 D1 D0
MASTER INTERRUPT ENABLE (MIE)
DISABLE LOWER CHAIN (DLC)
NO VECTOR (NV)
PORT A VECTOR INCLUDES STATUS (PA VIS)
RESET
RIGHT JUSTIFIED ADRESS
0= SHIFT LEFT(A0 from AD1)
1= RIGHT JUSTIFY (A0 from AD0)
COUNTER/TIMERS VECTOR INCLUDES STATUS (CT VIS)
PORT B VECTOR INCLUDES STATUS (PB VIS)

### Master Configuration Control Register

Addresses: 000001

(Read/Write)

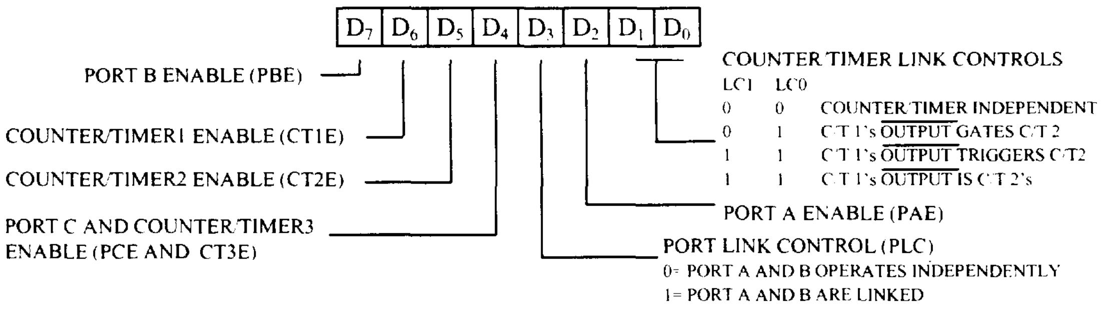

D7 D6 D5 D4 D3 D2 D1 D0
PORT B ENABLE (PBE)
COUNTER/TIMER1 ENABLE (CT1E)
COUNTER/TIMER2 ENABLE (CT2E)
PORT C AND COUNTER/TIMER3 ENABLE (PCE AND CT3E)
COUNTER/TIMER LINK CONTROLS
LC1 LC0
0 0 COUNTER/TIMER INDEPENDENT
0 1 C/T 1's OUTPUT GATES C/T 2
1 1 C/T 1's OUTPUT TRIGGERS C/T2
1 1 C/T 1's OUTPUT IS C/T 2's
PORT A ENABLE (PAE)
PORT LINK CONTROL (PLC)
0= PORT A AND B OPERATES INDEPENDENTLY
1= PORT A AND B ARE LINKED

**Figure 12. Master Control Register**

### Port Mode Specification Registers

Addresses: 100000 Port A
101000 Port B

(Read/Write)

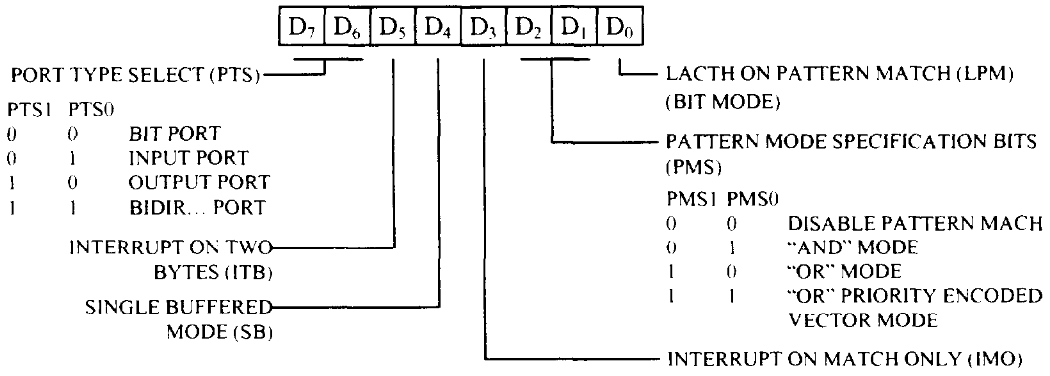

D7 D6 D5 D4 D3 D2 D1 D0
PORT TYPE SELECT (PTS)
PTS1 PTS0
0 0 BIT PORT
0 1 INPUT PORT
1 0 OUTPUT PORT
1 1 BIDIR... PORT
INTERRUPT ON TWO BYTES (ITB)
SINGLE BUFFERED MODE (SB)
LACTH ON PATTERN MATCH (LPM) (BIT MODE)
PATTERN MODE SPECIFICATION BITS (PMS)
PMS1 PMS0
0 0 DISABLE PATTERN MACH
0 1 "AND" MODE
1 0 "OR" MODE
1 1 "OR" PRIORITY ENCODED VECTOR MODE
INTERRUPT ON MATCH ONLY (IMO)

**Figure 13. Port Specification Register**

### Data Path Polarity Registers

Addresses: 100010 Port A
101010 Port B
000101 Port C (4 LSBs only)
(Read/Write)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |

DATA PATH POLARITY (DPP)
0 = NON-INVERTING
1 = INVERTING

### Data Direction Registers

Addresses: 100011 Port A
101011 Port B
000110 Port C (4 LSBs only)
(Read/Write)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |

DATA DIRECTION (DD)
0 = OUTPUT BIT
1 = INPUT BIT

### Special I/O Control Registers

Addresses: 100100 Port A
101100 Port B
000111 Port C (4 LSBs only)
(Read/Write)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |

SPECIAL INPUT/OUTPUT (SIO)
0 = NORMAL INPUT OR OUTPUT
1 = OUTPUT WITH OPEN-DRAIN
INPUT WITH 1's CATCHER

**Figure 14. Bit Path Definition Registers**

### Port Data Registers

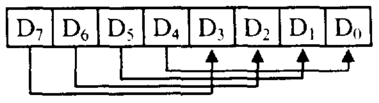
Addresses: 001101 Port A*
001110 Port B*
(Read/Write)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |

*These registers can be addressed directly.

### Port C Data Register

Addresses: 001111 *
(Read/Write)



4 MSBs
0 = WRITING OF CORRESPONDING LSB ENABLED
1 = WRITING OF CORRESPONDING LSB INHIBITED
(READ RETURNS 1)

**Figure 15. Port Data Registers**

### Main Control Register

| Address | Register name |
|---|---|
| 000000 | Master Interrupt Control |
| 000001 | Master Configuration Control |
| 000010 | Port A's Interrupt Vector |
| 000011 | Port B's Interrupt Vector |
| 000100 | Counter/Timer's Interrupt Vector |
| 000101 | Port C's Data Path Polarity |
| 000110 | Port C's Data Direction |
| 000111 | Port C's Special I/O Control |

### Port A specification Register

| Address | Register name |
|---|---|
| 100000 | Port A's Mode Specification |
| 100001 | Port A's Handshake Specification |
| 100010 | Port A's Data Path Polarity |
| 100011 | Port A's Data Direction |
| 100100 | Port A's Special I/O Control |
| 100101 | Port A's Pattern Polarity |
| 100110 | Port A's Pattern Transition |
| 100111 | Port A's Pattern Mask |

### Most Often Accessed Registers

| Address | Register name |
|---|---|
| 001000 | Port A's Command and Status |
| 001001 | Port B's Command and Status |
| 001010 | Counter/Timer 1's Command and Status |
| 001011 | Counter/Timer 2's Command and Status |
| 001100 | Counter/Timer 3's Command and Status |
| 001101 | Port A's Data (can be accessed directly) |
| 001110 | Port B's Data (can be accessed directly) |
| 001111 | Port C's Data (can be accessed directly) |

### Port B specification Register

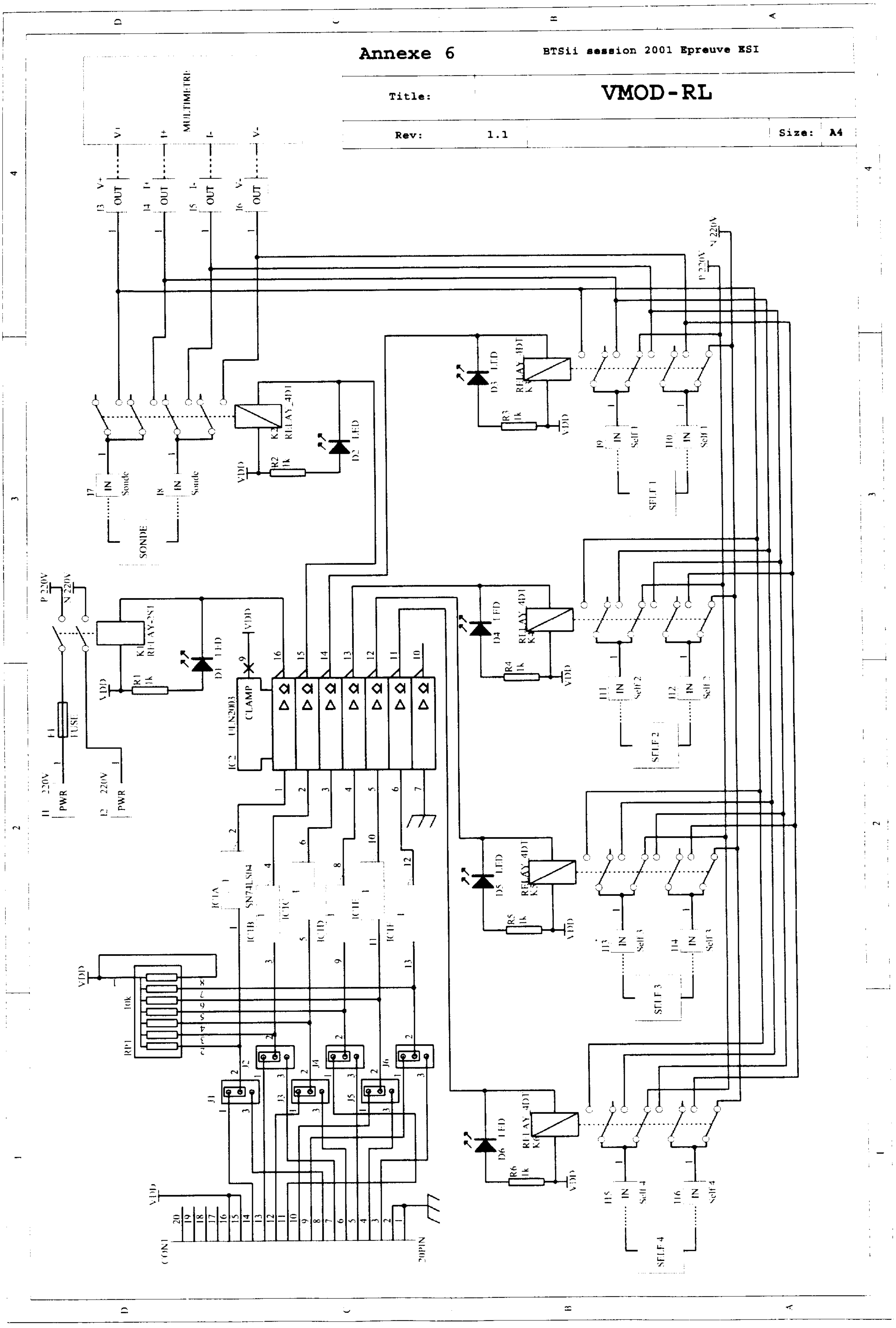
| Address | Register name |
|---|---|
| 101000 | Port A's Mode Specification |
| 101001 | Port A's Handshake Specification |
| 101010 | Port A's Data Path Polarity |
| 101011 | Port A's Data Direction |
| 101100 | Port A's Special I/O Control |
| 101101 | Port A's Pattern Polarity |
| 101110 | Port A's Pattern Transition |
| 101111 | Port A's Pattern Mask |

**Figure 16. Register Address Summary**

**Annexe 6** BTSii session 2001 Epreuve ESI

Title: **VMOD-RL**

Rev: 1.1 Size: A4



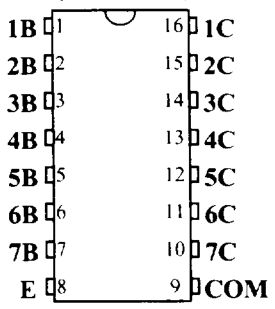
# ANNEXE S7
## DOCUMENTATION du ULN2003

**ULN2001A, ULN2002A, ULN2003A, ULN2004A
DARLINGTON TRANSISTOR ARRAYS**

## HIGH-VOLTAGE HIGH-CURRENT DARLINGTON TRANSISTOR ARRAYS

- **500-mA Rated Collector Current (Single Output)**
- **High-Voltage Outputs ... 50 V**
- **Output Clamp Diodes**
- **Inputs Compatible With Various Types of Logic**
- **Relay Driver Applications**
- **Designed to Be Interchangeable With Sprague ULN2001A Series**

**D or N PACKAGE
(TOP VIEW)**

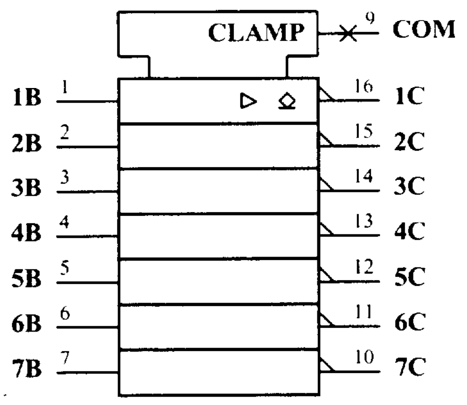


## Description

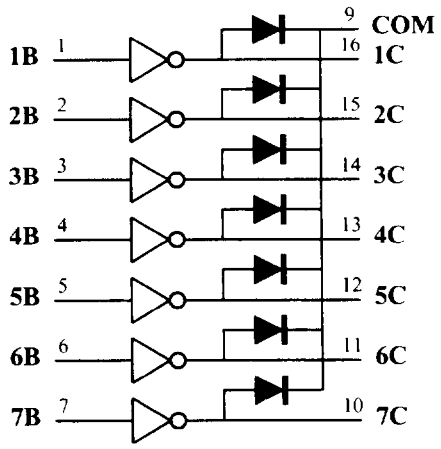
The ULN2001A, ULN2002A, ULN2003A, and ULN2004A are monolithic high-voltage, high-current Darlington transistor arrays. Each consists of seven NPN Darlington pairs that feature high-voltage outputs with common-cathode clamp diodes for switching inductive loads. The collector-current rating of a single Darlington pair is 500 mA. The Darlington pairs may be paralleled for higher current capability. Applications include relay drivers, hammer drivers, lamp drivers, display drivers (LED and gas discharge), line drivers, and logic buffers.
For 100-V (otherwise interchangeable) versions, see the SN75465 through SN75469.

The ULN2001A is a general-purpose array and can be used with TTL and CMOS technologies. The ULN2002A is specifically designed for use with 14- to 25-V PMOS devices. Each input of this device has a zener diode and resistor in series to control the input current to a safe limit. The ULN2003A has a 2.7-kΩ series base resistor for each Darlington pair for operation directly with TTL or 5-V CMOS devices. The ULN2004A has a 10.5-kΩ series base resistor to allow its operation directly from CMOS devices that use supply voltages of 6 to 15 V. The required input current of the ULN2004A is below that of the ULN2003A, and the required voltage is less than that required by the ULN2002A.

**logic symbol†**



**logic diagram**



† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.

# ANNEXE S8

# DOCUMENTATION DU DRIVER GPIB

**NAME**        mgpib - VMOD-GPIB driver

## SYNOPSIS

```
mgpibDrv()          - GPIB driver initialization routine
mgpibDevCreate()    - create a device for the VMOD-GPIB

STATUS mgpibDrv(void)
STATUS mgpibDevCreate(char * name, int brdflag, int modno, UINT baddr, int
vector, int level)
```

## DESCRIPTION

This is the driver for the VMOD-GPIB IEEE-488 interface module. The driver support only master functionality.

## USER CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. Two routines, however, must be called directly:

```
    mgpibDrv() to initialize the driver, and
    mgpibDevCreate() to create devices.
```

Before the driver can be used, it must be initialized by calling gpibDrv(). This routine should be called exactly once, before any reads, writes, or calls to mgpibDevCreate().

Before the VMOD-GPIB module can be used, it must be created by mgpibDevCreate().

## MgpibDevCreate()

### NAME

mgpibDevCreate() - create a device for the VMOD-GPIB

### SYNOPSIS

```
STATUS mgpibDevCreate
        (
        char *  name,     /* name to use for this device      */
        int     brdflag,  /* board flag                       */
        int     modno,    /* module number                    */
        UINT    baddr,    /* MGPIB base address               */
        int     vector,   /* MGPIB vector                     */
        int     level     /* MGPIB level                      */
        )
```

### DESCRIPTION

This routine creates a device for the **GPIB** module port. Each port to be used should have exactly one device associated with it by calling this routine. The maximum number of GPIB modules that is supported is <MAX_MGPIB_SLOTS> as defined in <mgpib.h>.

<name> will be the device name, this can then be used to open the device after it is created.

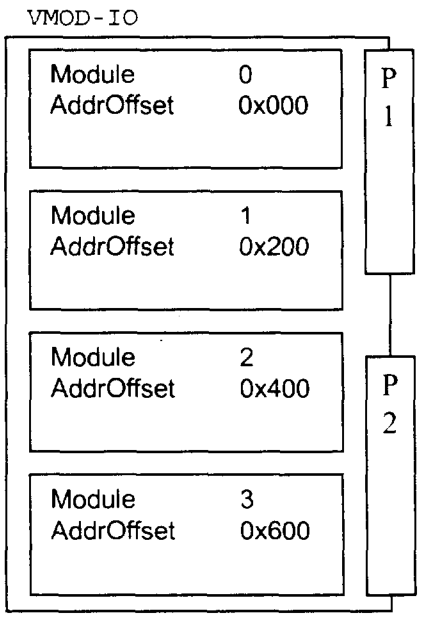
The integer number <brdflag> specifies the carrier board type, on which your module is installed. Currently the following carrier board types are supported:

| | |
|---|---|
| 0 | VMOD-IO, the nonintelligent 6U VMEbus MODULbus carrier board. |
| 1 | VMOD-40/60, the 68k VMEbus processor board with MODULbus. |
| 2 | MOD-AT/MOD-ATS, the nonintelligent carrier boards for ISAbus. |
| 3 | MOD-PCI, the nonintelligent carrier board for PCIbus. |
| 4 | LKMOD, a portable library for usage of CPU local MODULbus slots.currently supported by VMOD-P5 local slots. |

Some of the board types might not be available, depending on the compile time configuration of the driver.

## VMOD-IO

The VMOD-IO can take up to four modules, so the <modno> parameter will range from <0..3> (where <0> is the topmost module). For the <baddr> parameter, the access address and address space of the VMOD-IO on the VMEbus as well as the host CPU address map must be taken into consideration.



For further need to specify the base interrupt vector that you have jumpered on your carrier board. Note that <vector> will be equal for all devices on the same VMOD-IO.
By <level> you specify the hardware interrupt level that you have jumpered on the VMOD-IO. This information is used to enable VMEbus interrupts on the CPU board (by using sysIntEnable()).

The VMOD-IO has a strange interrupt vector generation: It might produce 15 different interrupt vectors if a board is equipped with 4 interrupt generating modules.

All the possible interrupt vectors need to be connected to proper interrupt service routines when the devices are created, and therefore some interrupt routines need to be connected to several interrupt vectors to cover all combinations. The driver does so automatically when a device is created (This is the reason why you only need to specify the base interrupt vector when you call mgpibDevCreate()).

The assignment of interrupt vectors to interrupt routines is done as follows:

| **vector offset (binary)** | **interrupt routine** |
|---|---|
| 0000 | module 3 |
| ... | |
| 0111 | module 3 |
| 1000 | module 2 |
| ... | |
| 1011 | module 2 |
| 1100 | module 1 |
| 1101 | module 1 |
| 1110 | module 0 |
| 1111 | not assigned |

With this assignment, module 3 has the highest priority (in case of simultaneously pending interrupts from several modules). Note that each module has it's unique interrupt service routine, and that no negotiation between these interrupt routines is necessary. It does not matter if slots were not connected to any device (and no interrupt routine is inserted into the vector-table at a slots positions), or if all the devices are of a different type.

All VxWorks drivers `made by JANZ' follow this assignment system. If you want to mix your drivers with ours, then you need to implement the same scheme.

**VMOD-IO EXAMPLE**
Suppose that you have a CPU that accesses the VMEbus short IO range at the local address <0xffff0000> (which is a common assumption), and that you have configured your VMOD-IO to reside on address <0xa800> within short IO range, then a device for a VMOD-ICAN in the second-topmost slot would be created as follows:

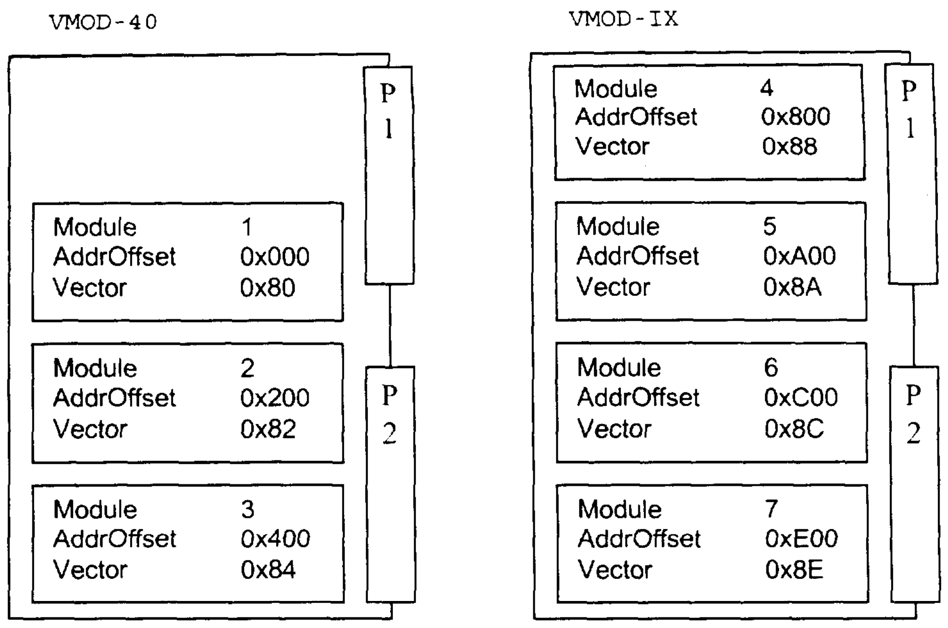
```
    mgpibDevCreate("/mgpib_io_1", 0, 1, 0xffffaa00, 0xa0, 3);
```

This example further assumes that the VMOD-IO uses IRQ-level 3 and the interrupt vector is configured to be <0xax>.

## VMOD-40

The VMOD-IO can take up to seven modules, the <modno> parameter will range from <1..7>. Module <1> is the topmost module on the base board, and <4> is the topmost module on the iX extension board. The <baddr> parameter will always be <0xfe400000 + AddrOffset>. This would be <0xfe400400> for the third module.

The MODULbus sockets on the VMOD-40 are always assigned to interupt level 5 (you still you must specify this). When you have not modified the BSP, interrupt vector numbers starting at 0x80 are used for MODULbus interrupts.



**VMOD-40 EXAMPLE**
To create a device that operates on the topmost MODULbus slot of a VMOD-40, you need to say:

```
      mgpibDevCreate("/mgpib_1", 1, 1, 0xfe400000, 0x80, 5);
```

---

## mgpibDrv()

### NAME

mgpibDrv() - GPIB driver initialization routine

### SYNOPSIS

```
STATUS mgpibDrv (void)
```

### DESCRIPTION

This routine initializes the driver, and performs hardware initialization of the GPIB module.

This routine should be called exactly once, before any reads, writes, or calls to mgpibDevCreate(). Normally, it is called by usrRoot() is usrConfig.c.

### RETURNS

OK, or ERROR if the driver cannot be installed.

### SEE ALSO

mgpib, mgpibDevCreate()