Hardware-Software Interfaces

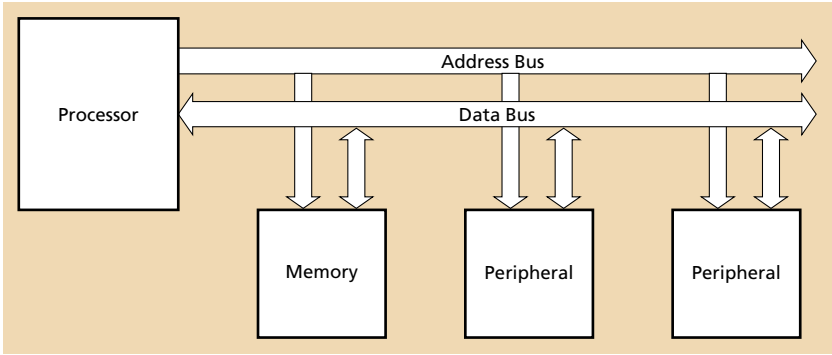
CSEE W4840

Prof. Stephen A. Edwards

Columbia University

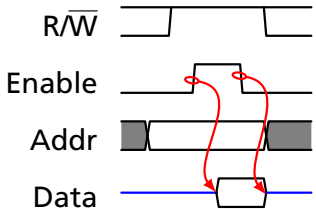
Spring 2026

Processor System Block Diagram

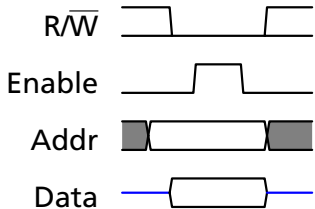


Simple Bus Timing

Read Cycle

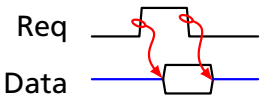


Write Cycle

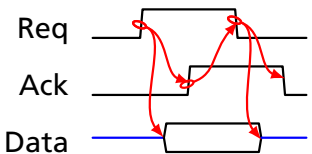


Strobe vs. Handshake

Strobe



Handshake

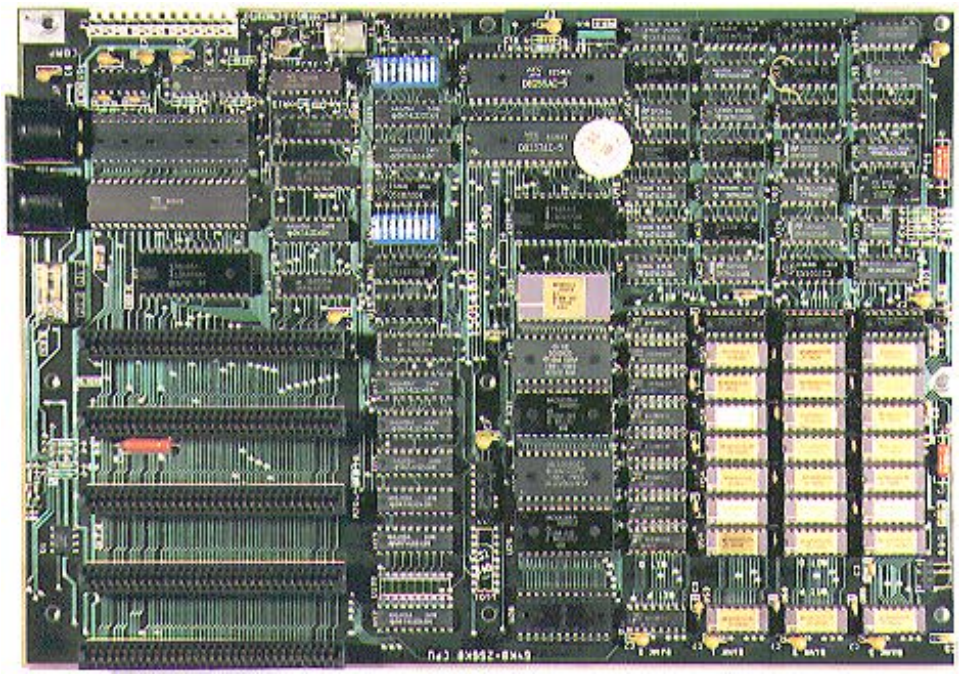


1982:

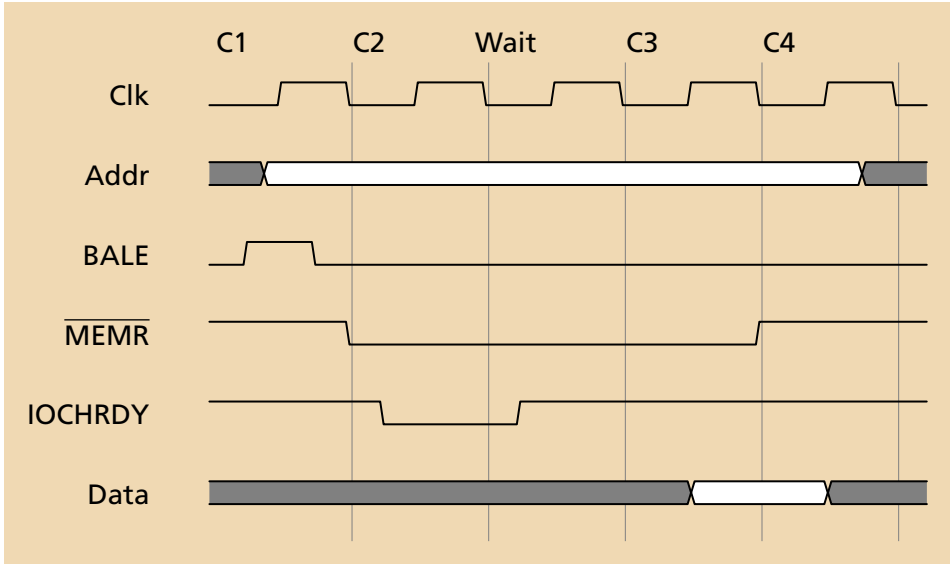
The IBM PC

Intel 8088 4.77 MHz

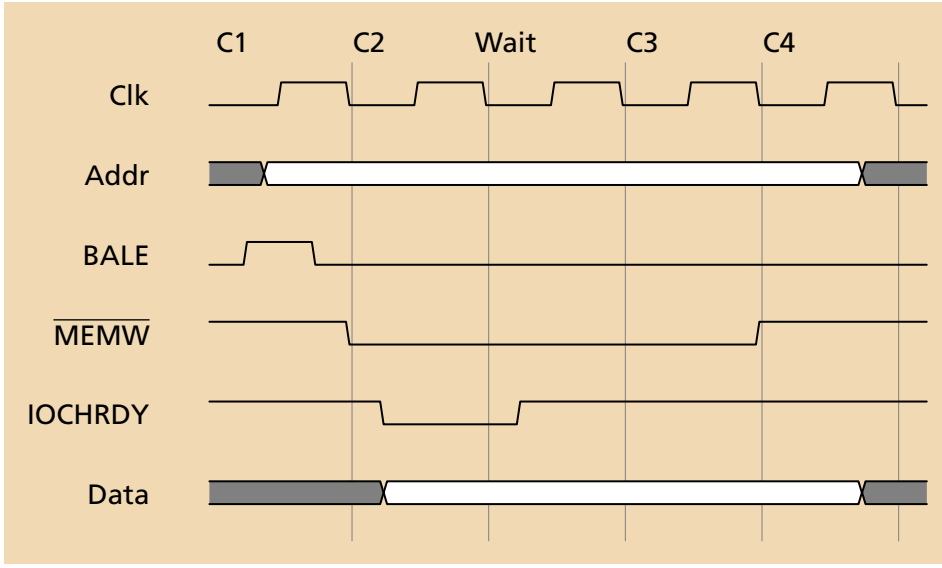
8-bit ISA bus



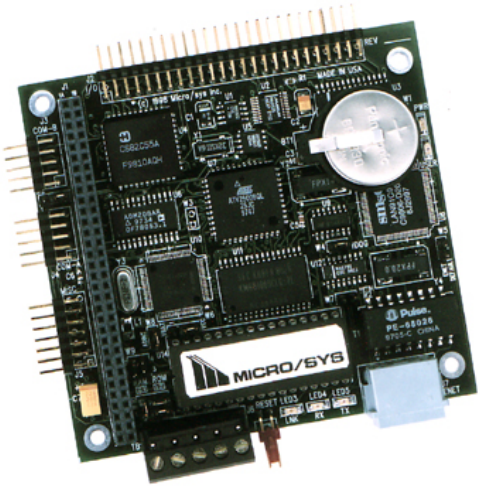
The ISA Bus: Memory Read



The ISA Bus: Memory Write



The PC/104 Form Factor: ISA Lives



Memory-Mapped I/O

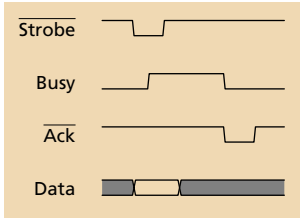
- To a processor, everything is memory.
- Peripherals appear as magical memory locations.
- Status registers: when read, report state of peripheral
- Control registers: when written, change state of peripheral

Typical Peripheral: The PC Parallel Port

Signal Name	Adapter Pin Number
– Strobe	1
+Data Bit 0	2
+Data Bit 1	3
+Data Bit 2	4
+Data Bit 3	5
+Data Bit 4	6
+Data Bit 5	7
+Data Bit 6	8
+Data Bit 7	9
– Acknowledge	10
+Busy	11
+P.End (out of paper)	12
+Select	13
– Auto Feed	14
– Error	15
– Initialize Printer	16
– Select Input	17
Ground	18-25

Printer

Printer Adapter



				Printer Adapter			
				Output to address hex 378			
Bit 7 Pin 9	Bit 6 Pin 8	Bit 5 Pin 7	Bit 4 Pin 6	Bit 3 Pin 5	Bit 2 Pin 4	Bit 1 Pin 3	Bit 0 Pin 2

				Printer Adapter			
				Input from address hex 379			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pin 11	Pin 10	Pin 12	Pin 13	Pin 15	—	—	—

IBM Personal Computer
Technical Reference
Manual, 1983

		Printer Adapter			
		Output to address hex 37A			
	Bit 4 IRQ Enable	$\overline{\text{Bit 3}}$ Pin 17	Bit 2 Pin 16	$\overline{\text{Bit 1}}$ Pin 14	$\overline{\text{Bit 0}}$ Pin 1

Table 2-2: Parallel port bits, arranged by register.

Data Register (Base Address)					
Bit	**Pin: D-sub**	**Signal Name**	**Source**	**Inverted at connector?**	**Pin: Centronics**
0	2	Data bit 0	PC	no	2
1	3	Data bit 1	PC	no	3
2	4	Data bit 2	PC	no	4
3	5	Data bit 3	PC	no	5
4	6	Data bit 4	PC	no	6
5	7	Data bit 5	PC	no	7
6	8	Data bit 6	PC	no	8
7	9	Data bit 7	PC	no	9
Some Data ports are bidirectional. (See Control register, bit 5 below.)					
Status Register (Base Address +1)					
Bit	**Pin: D-sub**	**Signal Name**	**Source**	**Inverted at connector?**	**Pin: Centronics**
3	15	nError (nFault)	Peripheral	no	32
4	13	Select	Peripheral	no	13
5	12	PaperEnd	Peripheral	no	12
6	10	nAck	Peripheral	no	10
7	11	Busy	Peripheral	yes	11
Additional bits not available at the connector: 0: may indicate timeout (1=timeout). 1, 2: unused.					
Control Register (Base Address +2)					
Bit	**Pin: D-sub**	**Signal Name**	**Source**	**Inverted at connector?**	**Pin: Centronics**
0	1	nStrobe	PC[1]	yes	1
1	14	nAutoLF	PC[1]	yes	14
2	16	nInit	PC[1]	no	31
3	17	nSelectIn	PC[1]	yes	36
[1]When high, PC can read external input (SPP only). Additional bits not available at the connector: 4: Interrupt enable. 1=IRQs pass from *nAck* to system's interrupt controller. 0=IRQs do not pass to interrupt controller. 5: Direction control for bidirectional Data ports. 0=outputs enabled. 1=outputs disabled; Data port can read external logic voltages. 6,7: unused					

Jan Axelson, *Parallel Port Complete*, 2002

DATA PORT BASE ADDRESS + 00H
STATUS PORT BASE ADDRESS + 01H
CONTROL PORT BASE ADDRESS + 02H

The bit map of these registers is:

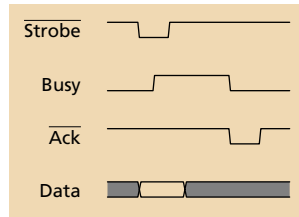
	D0	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**
DATA PORT	PD0	PD1	PD2	PD3	PD4	PD5	PD6	PD7
STATUS PORT	TMOUT	0	0	nERR	SLCT	PE	nACK	nBUSY
CONTROL PORT	STROBE	AUTOFD	nINIT	SLC	IRQE	PCD	0	0

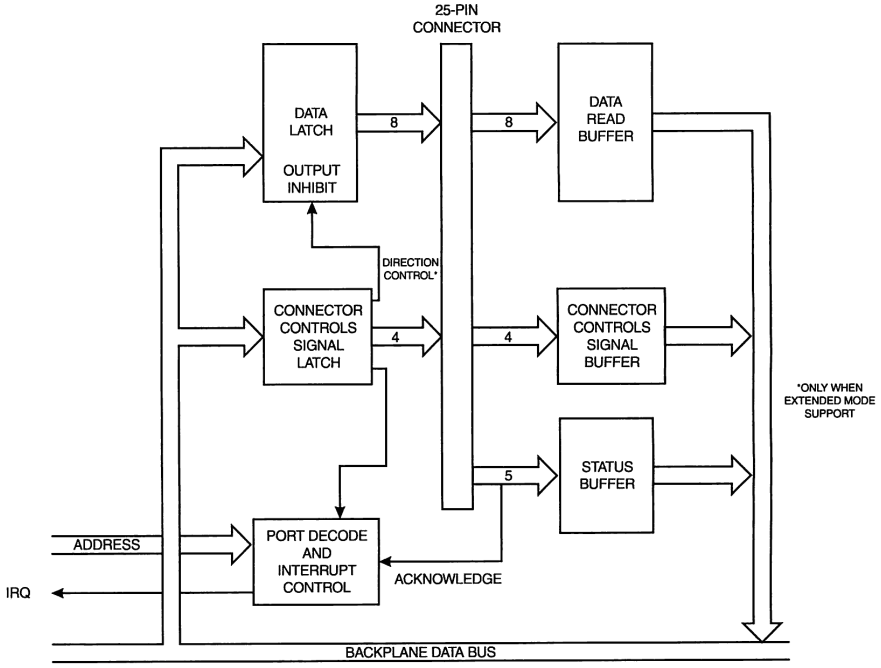
SMSC FDC37M81x PC98/99 Compliant Enhanced Super I/O Controller with Keyboard/Mouse Wakeup Datasheet, 2000

Parallel Port Registers

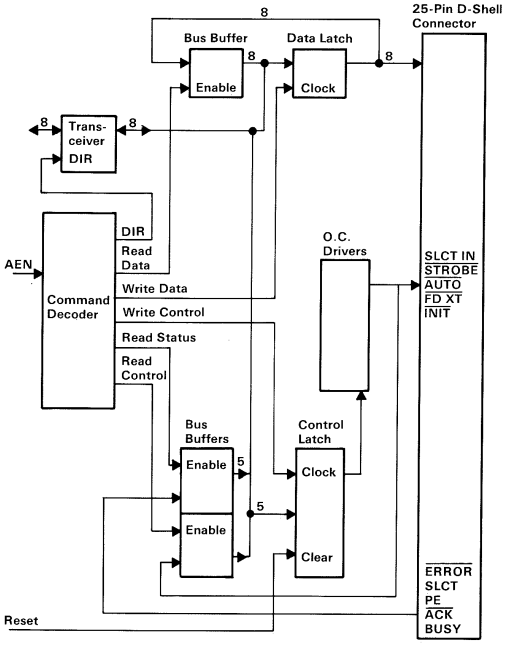
D7	D6	D5	D4	D3	D2	D1	D0	
$\overline{\text{Busy}}$	Ack	Paper	Sel	Err				0x378 Data
								0x379 Status
				$\overline{\text{Sel}}$	Init	$\overline{\text{Auto}}$	$\overline{\text{Strobe}}$	0x37A Control

1. Write Data
2. Assert Strobe
3. Wait for Busy to clear
4. Wait for Acknowledge



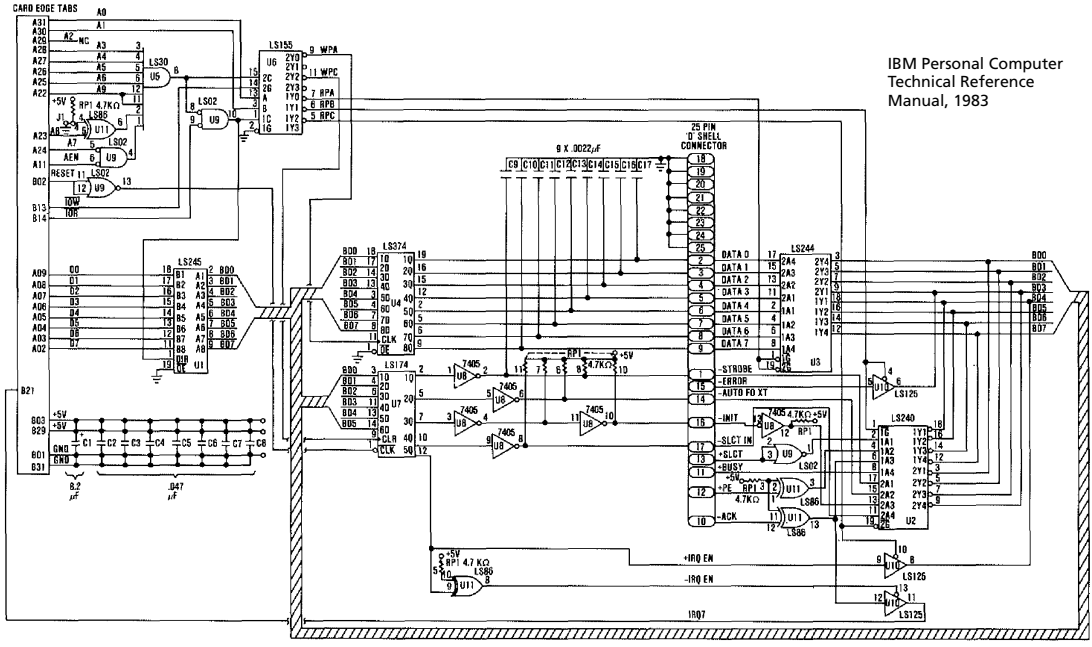


Frank van Gilluwe,
The Undocumented PC,
1997

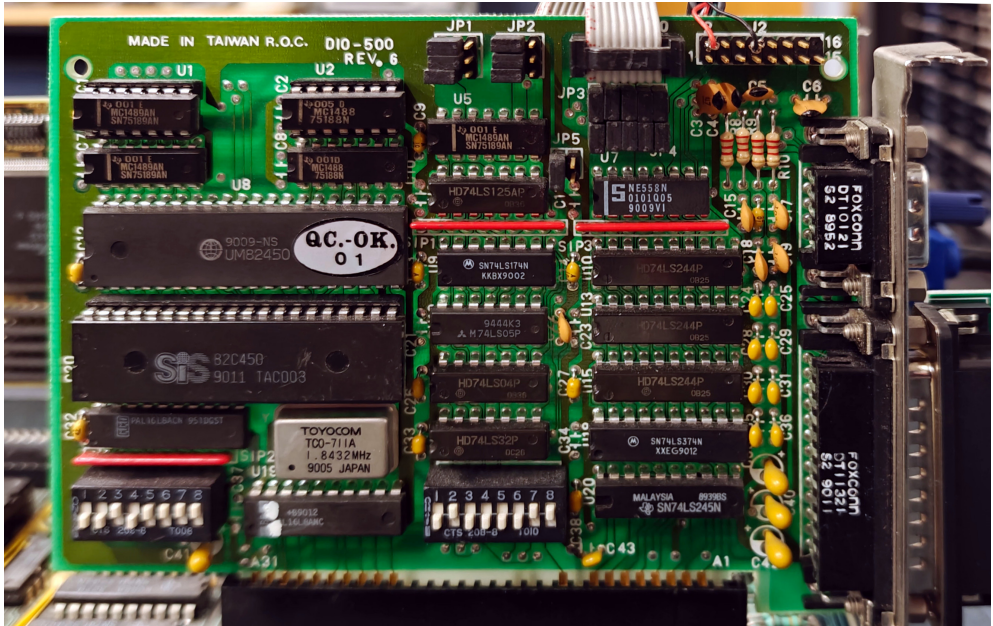


Printer Adapter Block Diagram

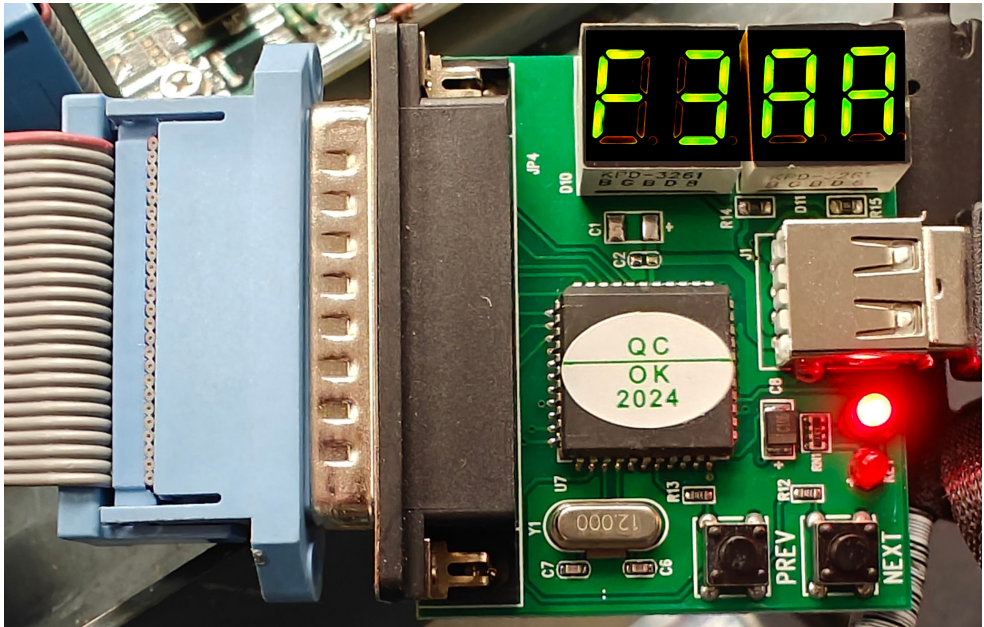
IBM Personal Computer Technical Reference Manual, 1983


IBM Personal Computer
Technical Reference
Manual, 1983
CARD EDGE TABS
25 PIN
'D' SHELL
CONNECTOR
9 X .0022μF
LS155
9 WPA
11 WPC
7 RPA
6 RPB
5 RPC
LS30
LS02
LS86
LS245
LS374
LS174
LS244
LS125
LS240
7405
4.7KΩ
RP1
DATA 0
DATA 1
DATA 2
DATA 3
DATA 4
DATA 5
DATA 6
DATA 7
-STROBE
-ERROR
-AUTO FD XT
-INIT
-SLCT IN
+SLCT
+BUSY
+PE
-ACK
+IRQ EN
-IRQ EN
IRQ7
+5V
GND
8.2 μF
.047 μF

Diamond Flower DIO-500 Parallel + Serial ISA card c. 1986

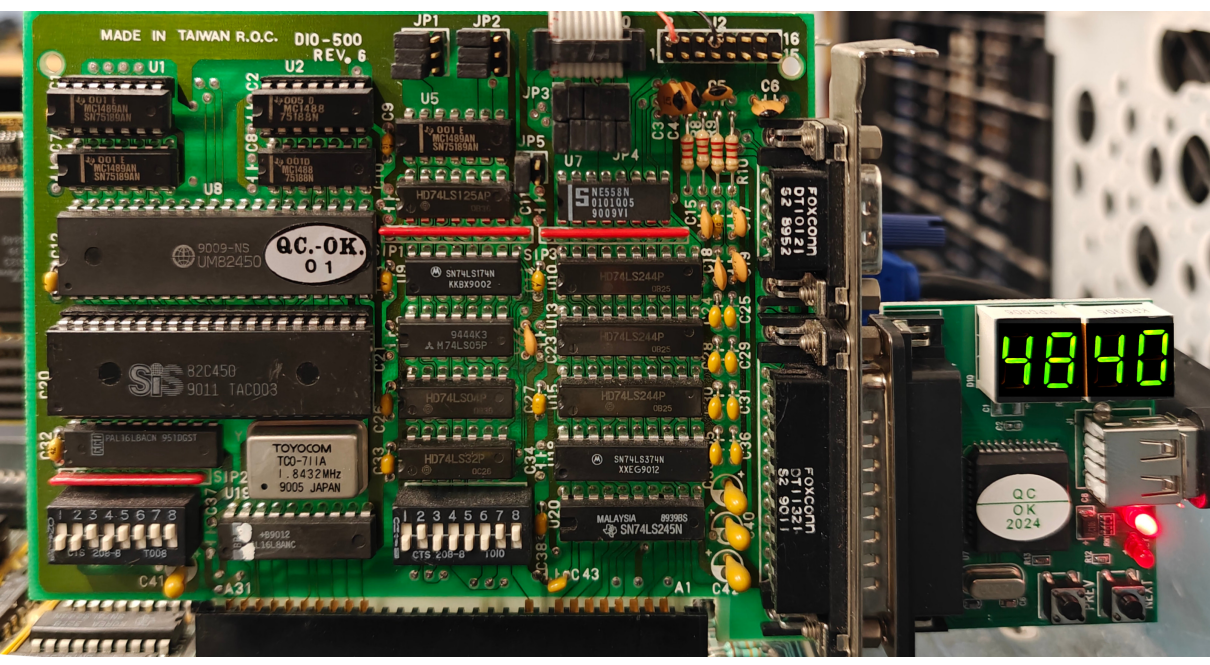


Parallel Port Monitor



Write to the Parallel Port in BASIC

```
list
10 OUT &H378, &H48
20 FOR I = 1 TO 100    : NEXT I
25 PRINT "."
30 OUT &H378, &H40
40 FOR I = 1 TO 100    : NEXT I
50 GOTO 10
Ok
```


MADE IN TAIWAN R.O.C.
DIO-500
REV. 6
JP1
JP2
JP3
JP4
JP5
U1
U2
U5
U7
U8
MC1488AN
SN75189AN
MC1488
75188N
HD74LS125AP
NE558N
0101Q05
9009VI
9009-NS
UM82450
Q.C.-OK.
01
SN74LS174N
KKBX9002
HD74LS244P
9444K3
M74LS05P
HD74LS04P
HD74LS32P
SiS
82C450
9011 TAC003
PAL16L8ACN 951DGST
TOYOCOM
TCO-711A
1.8432MHz
9005 JAPAN
SN74LS374N
XXEG9012
MALAYSIA
8939BS
SN74LS245N
FOXCONN
DT-1012I
S2 8952
FOXCONN
DT-1321
S2 9011
4840
QC
OK
2024
PREV
NEXT

A Parallel Port Driver

```
#define DATA    0x378
#define STATUS  0x379
#define CONTROL 0x37A

#define NBSY 0x80
#define NACK 0x40
#define OUT  0x20
#define SEL  0x10
#define NERR 0x08
#define STROBE 0x01

#define INVERT  (NBSY | NACK |       SEL | NERR)
#define MASK    (NBSY | NACK | OUT | SEL | NERR)
#define NOT_READY(x) ((inb(x)^INVERT)&MASK)

void write_single_character(char c) {
  while (NOT_READY(STATUS)) ;
  outb(DATA, c);
  outb(CONTROL, control | STROBE); /* Assert STROBE */
  outb(CONTROL, control ); /* Clear STROBE */
}
```

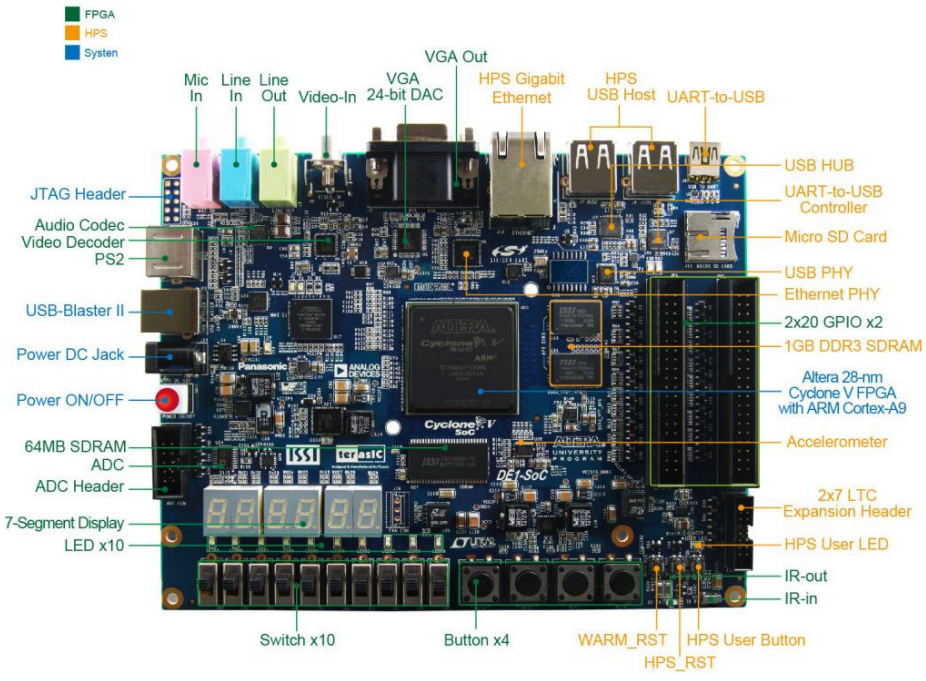

FPGA
HPS
Systen
Mic In
Line In
Line Out
Video-In
VGA 24-bit DAC
VGA Out
HPS Gigabit Ethernet
HPS USB Host
UART-to-USB
JTAG Header
Audio Codec
Video Decoder
PS2
USB-Blaster II
Power DC Jack
Power ON/OFF
64MB SDRAM
ADC
ADC Header
7-Segment Display
LED x10
Switch x10
Button x4
WARM_RST
HPS_RST
HPS User Button
IR-out
IR-in
HPS User LED
2x7 LTC Expansion Header
Accelerometer
Altera 28-nm Cyclone V FPGA with ARM Cortex-A9
1GB DDR3 SDRAM
2x20 GPIO x2
Ethernet PHY
USB PHY
Micro SD Card
UART-to-USB Controller
USB HUB

Table 3-23 Pin Assignment of LEDs, Switches and Push-buttons

Signal Name	*HPS GPIO*	*Register/bit*	*Function*
HPS_KEY	GPIO54	GPIO1[25]	I/O
HPS_LED	GPIO53	GPIO1[24]	I/O

Figure 22-1: Cyclone V SoC GPIO

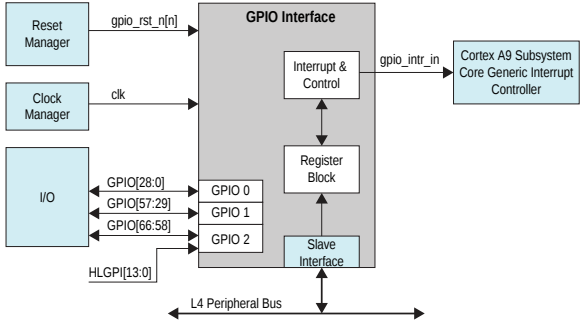


Table 22-1: GPIO Interface pin table

Pin Mux Name	Mapped to signal name	Comments
GPIO [28:0]	GPIO 0 [28:0]	Input / Output
GPIO [57:29]	GPIO 1 [28:0]	Input / Output
GPIO [66:58]	GPIO 2 [8:0]	Input / Output
HLGPI [13:0]	GPIO 2 [26:13]	Input only

GPIO Module Address Map

Registers in the GPIO module

Module Instance	Base Address
gpio0	0xFF708000
gpio1	0xFF709000
gpio2	0xFF70A000

GPIO Module

Register	Offset	Width	Access	Reset Value	Description
gpio_swporta_dr	0x0	32	RW	0x0	Port A Data Register
gpio_swporta_ddr	0x4	32	RW	0x0	Port A Data Direction Register
gpio_inten	0x30	32	RW	0x0	Interrupt Enable Register
gpio_intmask	0x34	32	RW	0x0	Interrupt Mask Register
gpio_inttype_level	0x38	32	RW	0x0	Interrupt Level Register
gpio_int_polarity	0x3C	32	RW	0x0	Interrupt Polarity Register
gpio_intstatus	0x40	32	RO	0x0	Interrupt Status Register
gpio_raw_intstatus	0x44	32	RO	0x0	Raw Interrupt Status Register
gpio_debounce	0x48	32	RW	0x0	Debounce Enable Register
gpio_porta_eoi	0x4C	32	WO	0x0	Clear Interrupt Register
gpio_ext_porta	0x50	32	RO	0x0	External Port A Register
gpio_ls_sync	0x60	32	RW	0x0	Synchronization Level Register
gpio_id_code	0x64	32	RO	0x0	ID Code Register
gpio_ver_id_code	0x6C	32	RO	0x3230382A	GPIO Version Register
gpio_config_reg2	0x70	32	RO	0x39CFC	Configuration Register 2
gpio_config_reg1	0x74	32	RO	0x1FF0F2	Configuration Register 1

gpio_ver_id_code

GPIO Component Version

Module Instance	Base Address	Register Address
gpio0	0xFF708000	0xFF70806C
gpio1	0xFF709000	0xFF70906C
gpio2	0xFF70A000	0xFF70A06C

Offset: 0x6C

Access: RO

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
gpio_ver_id_code RO 0x3230382A															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
gpio_ver_id_code RO 0x3230382A															

gpio_ver_id_code Fields

Bit	Name	Description	Access	Reset
31:0	gpio_ver_id_code	ASCII value for each number in the version, followed by *. For example. 32_30_31_2A represents the version 2.01	RO	0x3230382A

gpio_swporta_dr

This GPIO Data register is used to output data on the GPIO signals. Check the GPIO chapter in the handbook for details on how GPIO2 is implemented.

Module Instance	Base Address	Register Address
gpio0	0xFF708000	0xFF708000
gpio1	0xFF709000	0xFF709000
gpio2	0xFF70A000	0xFF70A000

Offset: 0x0

Access: RW

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			gpio_swporta_dr RW 0x0												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
gpio_swporta_dr RW 0x0															

gpio_swporta_dr Fields

Bit	Name	Description	Access	Reset
28:0	gpio_swporta_dr	Values written to this register are output on the I/O signals of the GPIO Data Register, if the corresponding data direction bits for GPIO Data Direction Field are set to Output mode. The value read back is equal to the last value written to this register. Note that only bits[26:0] are implemented for gpio2.	RW	0x0

Start the serial console before powering on the board

```
screen /dev/ttyUSB0 115200
```

Interrupt the boot process by quickly pressing a key

```
U-Boot SPL 2013.01.01 (Jan 12 2019 - 19:40:48)
BOARD : Altera SOCFPGA Cyclone V Board

...

Net:   mii0
Warning: failed to set MAC address

Hit any key to stop autoboot:  0
SOCFPGA_CYCLONE5 #
```

This is the U-Boot command line

```
SOCFPGA_CYCLONE5 # help
md      - memory display
mw      - memory write (fill)
```

Read the GPIO Version Register *gpio_ver_id_code*

```
SOCFPGA_CYCLONE5 # md ff70906c 1
ff70906c: 3230382a    *802
```

Read the state of the HPS USER BUTTON *gpio_ext_porta*

```
SOCFPGA_CYCLONE5 # md ff709050 1
ff709050: 1df7ffff    ....

SOCFPGA_CYCLONE5 # md ff709050 1
ff709050: 1ff7ffff    ....
```

Set the data direction register to output for HPS USER LED *gpio_swporta_ddr*

```
SOCFPGA_CYCLONE5 # mw ff709004 1000000
```

Blink the LED *gpio_swporta_dr*

```
SOCFPGA_CYCLONE5 # mw ff709000 1000000
SOCFPGA_CYCLONE5 # mw ff709000 0
```

Interrupts and Polling

Two ways to get data from a peripheral:

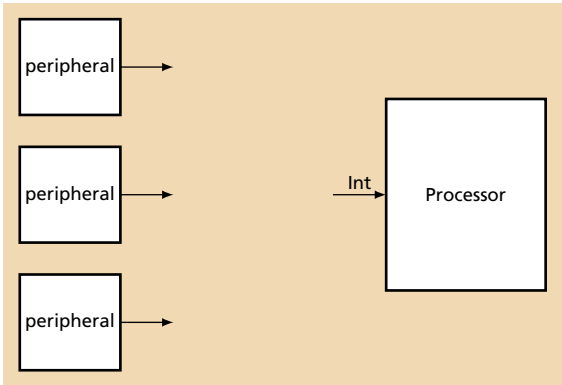
- Polling: “Are we there yet?”
- Interrupts: Ringing Telephone

Interrupts

Basic idea:

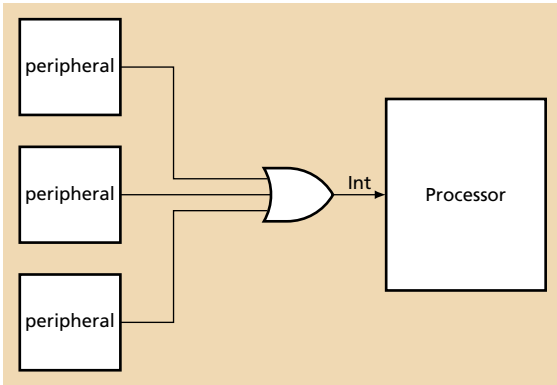
1. Peripheral asserts a processor's interrupt input
2. Processor temporarily transfers control to interrupt service routine
3. ISR gathers data from peripheral and acknowledges interrupt
4. ISR returns control to previously-executing program

Many Different Interrupts



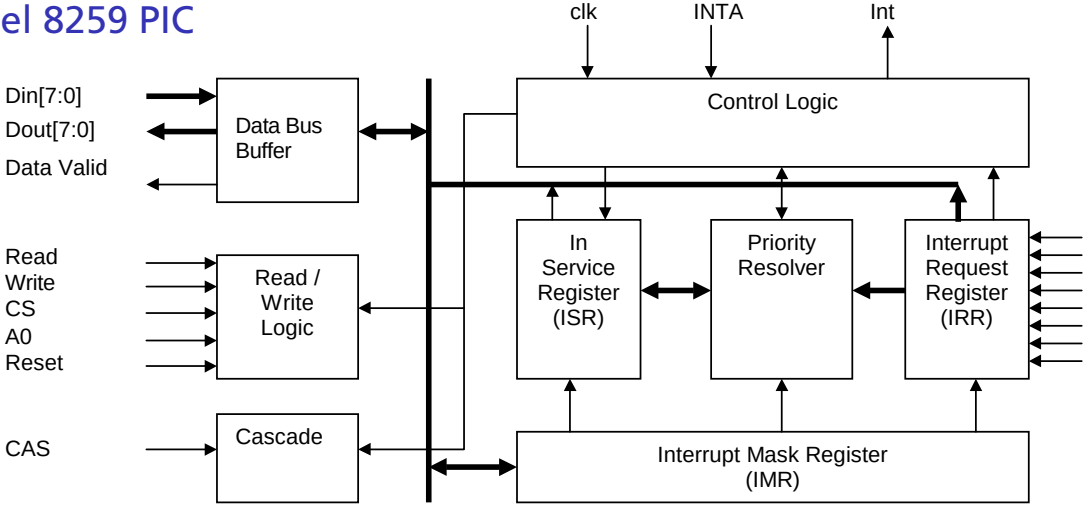
What's a processor to do?

Many Different Interrupts



What's a processor to do?
ISR polls all potential interrupt sources, then dispatches handler.

Intel 8259 PIC



Prioritizes incoming requests & notifies processor

ISR reads 8-bit interrupt vector number of winner

IBM PC/AT: two 8259s; became standard

Interrupts on the Cyclone V

Interrupt controller is part of the Cortex-A9 MPCore subsystem

The *Generic Interrupt Controller* supports 180 interrupt sources, including 64 from FPGA peripherals

Described in ARM's *Cortex-A9 MPCore Technical Reference Manual* (124 pages) and the *ARM Generic Interrupt Controller Architecture Specification* (952 pages)

Complicated by multiple cores, protection levels, and virtual machines

Many, many configuration registers. Each interrupt has a priority, enable flag, status, and control

A perfect thing to let the operating system manage:

`https://docs.kernel.org/core-api/genericirq.html`

The NoCallerID Project, 2023

Asking the kernel to route interrupts from the device to our code:

```
// Get the interrupt number for our device from the Device Tree
int irq = irq_of_parse_and_map(pdev->dev.of_node, 0);
dev.irq_num = irq;

// Claim the interrupt and register our ISR
ret = request_irq(irq, (irq_handler_t) irq_handler, 0,
                  "csee4840_audio", NULL);
```

Code the kernel will call when the interrupt occurs:

```
// Our Interrupt Service Routine, called by the kernel
irq_handler_t irq_handler(int irq, void *dev_id,
                          struct pt_regs *reg) {
 ...
 wake_up_interruptible(&wq); // Unblock the program waiting on us

 return IRQ_RETVAL(1);
}
```