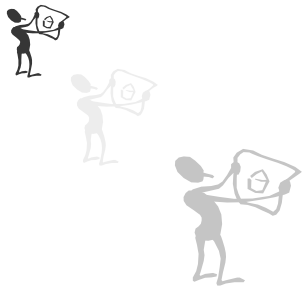
Chapter 7

Parallel Port & Peripherals

Process Control

Outline

- **Parallel Port Architecture (SPP)**
- **Parallel Port Architecture (EPP)**
- **SPP Driver**
- **EPP Driver**
- **IO-Drive 2000 Architecture**
- **Data Acquisition Functions**



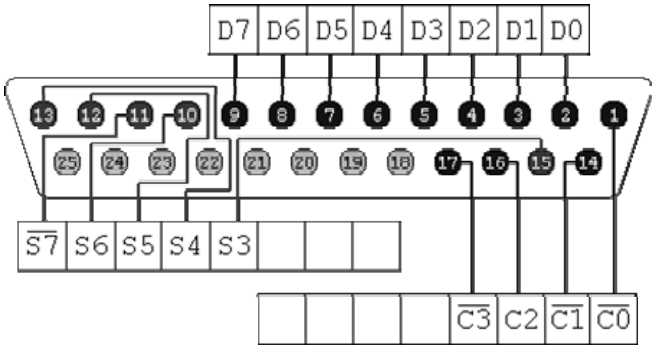
Parallel Port Architecture (SPP)

Base Address 0x378, 0x278, 0x3BC

Port 0x378	**Data Register**	**[7..0]**	**In / Out**
Port 0x379	**Status Register**	**[7..3]**	**In**
Port 0x37A	**Control Register**	**[3..0]**	**Out (O. C.)**

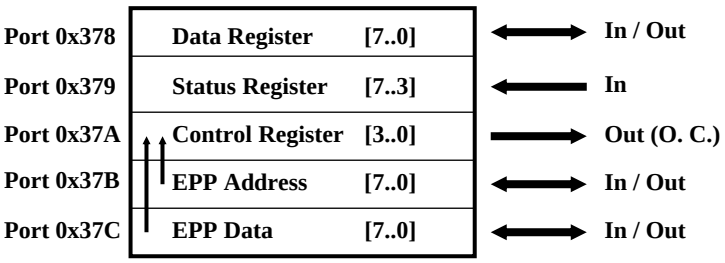
- Connector - 25 Pin D-Type (8 + 4 + 5 + 8G)
- Control Register has 2 virtual bits [5..4]:
  - •Bit-4 => Interrupt Enable.
  - •Bit-5 => Data Register direction (0-Out, 1-In).

Parallel Port 25 Pins Connector



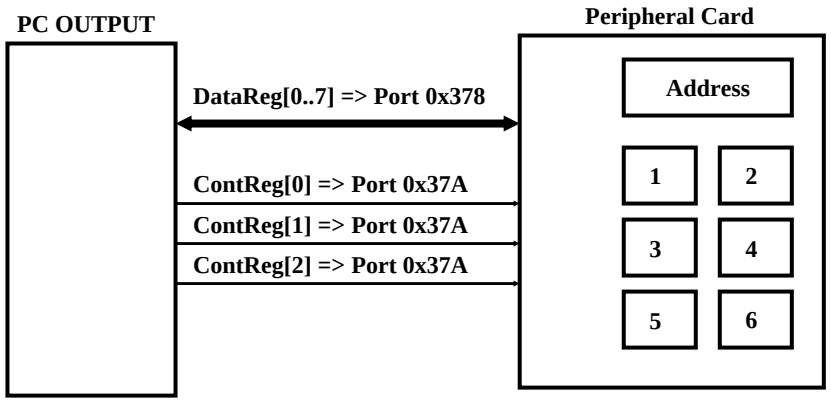
Parallel Port Architecture (EPP)

Base Address 0x378, 0x278, 0x3BC



- Control Register must init to 0x04 - outp(0x37A, 0x04).
- Write or Read to EPP ports generate full handshake ISA cycle, using the control register.
- The port direction is setting automatically.

Parallel Port and Peripherals



Parallel Port Protocol (Centronics)

Parallel Port Write Protocol (EPP & SPP)

Parallel Port Read Protocol (EPP & SPP)

SPP Drivers (p1)

```
#define nWriteBit       0x01        // ~C0
#define nDstBit         0x02        // ~C1
#define nInitBit        0x04        //  C2
#define nAstBit         0x08        // ~C3

#define IRQenBit        0x10        //  C5
#define BiDir           0x20        //  C6

static unsigned short Base  = 0x378;
static unsigned short Cport = 0x37A;

static const int idle = nWriteBit & ~nDstBit & ~nAstBit | nInitBit;  // 0x05

static int init = 0x01;
static int nAst = 0x0D;
static int nDst = 0x07;
static int inPst= 0x24;
```

SPP Drivers (p2)

```
/*******************************************************/
void setLptBaseAddr(unsigned addr)
  {
  Base = addr;
  Cport = Base+2;
  inItControlVar();
  }
/*******************************************************/
unsigned getLptBaseAddr()
  {
  return(Base);
  }
/*******************************************************/
```


SPP Drivers (p3)

```
/*******************************************************/
void setAddr(int addr)
  {
  outp(Base, addr);
  outp(Cport, nAst);
  outp(Cport, idle);
  }
/*******************************************************/
void writeData(int data)
  {
  outp(Base, data);
  outp(Cport, nDst);
  outp(Cport, idle);
  }
/*******************************************************/
```

SPP Drivers (p4)

```
/*******************************************************/
int readData()
  {
  int data;
  outp(Cport, inPst);
  outp(Cport, inPst | nDstBit );
  data = inp(Base);
  outp(Cport, inPst & ~nDstBit );
  outp(Cport, idle);
  return(data);
  }
/*******************************************************/
```


EPP Drivers (p1)

```
#define nWriteBit 0x01          // ~C0
#define nDstBit   0x02          // ~C1
#define nInitBit  0x04          //  C2
#define nAstBit   0x08          // ~C3

#define IRQenBit  0x10          //  C5
#define BiDir     0x20          //  C6


static unsigned Base  = 0x378;
static unsigned Cport = 0x37A;


static const int idle = 0x04;
```

EPP Drivers (p2)

```
/*********************************************************/
void setBaseAddr(unsigned addr)
  {
  Base = addr;
  Cport = Base+2;
  }
/*********************************************************/
unsigned getBaseAddr()
  {
  return(Base);
  }
/*********************************************************/
void resetCard()
  {
  outp(Cport, 0x00);
  outp(Cport, idle);
  }
/ /*********************************************************/
```


EPP Drivers (p3)

```
/*********************************************************/
void setAddr(int addr)
  {
  outp(Base+3, addr);
  }
/*********************************************************/
void writeData(int data)
  {
  outp(Base+4, data);
  }
/*********************************************************/
int readData()
  {
   return(inp(Base+4));
   }
/*********************************************************/
```

ECP Modes

```
/*********************************************************/
void setECPPortMode(int P, int M)
  {
  ECPport = P;
  /*
  000 - STANDARD MODE
  001 - SPP BIDIR MODE
  010 - PP FIFO MODE
  011 - ECP FIFO MODE
  100 - EPP MODE
  */
  outp(ECPport+2, M<<5);
 }
/*********************************************************/
 char getECPPort()
  {  return(ECPport);  }
/*********************************************************/
```


Addresses of IO-Drive 2000

0 = ADCL	- Read low byte	;ADC all Channel
1= ADCH	- Read high byte	;ADC all Channel
2= ADCS	- Write = Control & start conversion, Read = INT end of conversion.	
3= RESERV	- Optional	
4 = DAC0L	- Load low byte	;Analog Out Chan 0
5 = DAC0H	- Load high byte and update	;Analog Out Chan 0
6 = DAC1L	- Load low byte	;Analog Out Chan 1
7 = DAC1H	- Load high byte and update	;Analog Out Chan 1
8 = DIGITAL	- Write = Digital Out, Read=Digital In.	

Addresses of IO-Drive 2000

9 = PWM0	- Load Duty Cycle byte for PWM Channel 0 Bits [1,2].
10 = PWM1	- Load Duty Cycle byte for PWM Channel 1 Bits [3,4].
11= CONTROL	- Enable or Disable the PWM0[1,2], PWM1[3,4], CNT1, CNT2 and Timer.
12 = TIMER	- Write 4 Timer Divition Nibbel.
13 = CNTC	- Write Latch & Reser Counters.
14 = LATCH_L	- Read Latch low byte.
15 = LATCH_H	- Read Latch High byte.

Addresses of IO-Drive 2000

Read from ADC L is analog to digital channel (8 low bit).

Read from ADC H is analog to digital channel (4 high bit).

Write to ADC S (control byte) is to start conversion A/D.

Read from ADC S (bit 0) is INT- end of conversion A/D.

Write to DACx L is digital to analog channel x 8 low bit.

Write to DACx H is digital to analog channel x 4 high bit + UpDate.

Addresses of IO-Drive 2000

CONTROL									
R/W	7	6	5	4	3	2	1	0	
W	BUZ	CNT3e	CNT2e	CNT1e	PWM4e	PWM3e	PWM2e	PWM1e	
CNTC									
R/W	7	6	5	4	3	2	1	0	
W					LAC2	LAC1	XRST2	XRST1	

IO-Drive 2000 Definitions

```
#define READ_TIME_OUT       1000
#define TIME_OUT_ERROR      -1.0
#define MAX_VOLT            9.995       // 10*2047/2048
#define MIN_VOLT            -10.000     // 10*2048/2048
#define  MAX_VOLT_IN        10.000
```

IO-Drive 2000 Definitions

```
#define ADCL        0x00
#define ADCH        0x01
#define ADCS        0x02
#define RESERV      0x03
#define DAC0L       0x04
#define DAC0H       0x05
#define DAC1L       0x06
#define DAC1H       0x07
#define DIGITAL     0x08
#define PWM0        0x09
#define PWM1        0x0A
#define CONTROL     0x0B
#define TIMER       0x0C
#define CNTC        0x0D
#define LATCH_L     0x0E
#define LATCH_H     0x0F
```


IO-Drive 2000 Digital Out / In

```
/*************************************************************/
void initCard()
  {
  resetCard();
  }
/*************************************************************/
void DigitalOut(byte data)
  {
  setAddr(DIGITAL);
  writeData(data);
  }
/*************************************************************/
byte DigitalIn()
  {
  setAddr(DIGITAL);
  return(readData());
  }/*************************************************************/
```

DAC Example

```
void AnalogOut(byte Chan, double Volt)
 {
 int Temp;
 byte Lo, Hi;
 byte outChan = (Chan == 0) ? DAC0L : DAC1L;
 if (Volt > MAX_VOLT)
  Volt = MAX_VOLT;
 else if (Volt < MIN_VOLT)
  Volt = MIN_VOLT;
  Temp = (int)(Volt / 10.0 * 2048);
 Lo = Temp;
 Hi = ((Temp >> 8) ^ 0x08) & 0x0F;
 setAddr(outChan);
 writeData(Lo);
 setAddr(outChan+1);
 writeData(Hi);
 }
```


ADC Example

```
/*****************************************************/
double AnalogIn(byte Chan)
 {
 #define BIP     1       // 1 - Bipolar      0 - Unipolar
 #define RNG     1       // 1 - 10V          0 - 5V
 #define ACQ     0       // 0 - Int Acquisition      1 - Ext Acquisition
 #define PD10    1       // Normal Operation & Internal Clock
```

ADC Example

```
int i;
  short temp;              // must be short for << operation
  byte lo, hi;             // must be byte unless the the sign of lo will expanded
  int time = READ_TIME_OUT;
  setAddr(ADCS);
  writeData( (PD10 << 6) | (ACQ << 5) | (RNG << 4) | (BIP << 3) | (Chan & 0x07) );

  while (readData() & 0x01)
  if (time-- == 0) return (TIME_OUT_ERROR);
  setAddr(ADCL);
  lo = readData();
  setAddr(ADCH);
  hi = readData();
  temp = (short)((hi<<8) | lo);
  temp <<= 4; temp >>= 4;  // store the sign bit
  return( (double)(temp * MAX_VOLT_IN / 2048.0) );
  }
```