

LAMPIRAN - LAMPIRAN

LAMPIRAN A

PERANGKAT LUNAK

```
#INCLUDE "8051.H"
```

```
LED .EQU P3.7
```

```
SS .EQU P3.3
```

```
KODE .EQU P3.4
```

```
.ORG $30
```

```
DATANOHEX1 .BLOCK 1
```

```
DATANOHEX2 .BLOCK 1
```

```
DATANOHEX3 .BLOCK 1
```

```
DATANOHEX4 .BLOCK 1
```

```
DATADECHASIL1 .BLOCK 1
```

```
DATADECHASIL2 .BLOCK 1
```

```
DATADECHASIL3 .BLOCK 1
```

```
DATADECHASIL4 .BLOCK 1
```

```
DATADECHASIL5 .BLOCK 1
```

```
DATAPENAMBAH1 .BLOCK 1
```

```
DATAPENAMBAH2 .BLOCK 1
```

```
DATAPENAMBAH3 .BLOCK 1
```

```
DATAPENAMBAH4 .BLOCK 1
```

```
DATAPENAMBAH5 .BLOCK 1
```

```
DATAS1 .BLOCK 1
```

```
DATAS2 .BLOCK 1
```

```
DATAS3 .BLOCK 1
```

```
DATAS4 .BLOCK 1
```

```
DATAS5 .BLOCK 1
```

```
DATAS6 .BLOCK 1
```

```
DATAS7 .BLOCK 1
```

```
DATAS8 .BLOCK 1
```

```
BUFDATANOMOR .BLOCK 18
```

```
STARTDATA .BLOCK 1
```

```
DATANOMORASCII1 .BLOCK 1
```

```
DATANOMORASCII2 .BLOCK 1
```

```
DATANOMORASCII3 .BLOCK 1
DATANOMORASCII4 .BLOCK 1
DATANOMORASCII5 .BLOCK 1
DATANOMORASCII6 .BLOCK 1
DATANOMORASCII7 .BLOCK 1
DATANOMORASCII8 .BLOCK 1
DATANOMORASCII9 .BLOCK 1
DATANOMORASCII10 .BLOCK 1
ENDDATA      .BLOCK 1
STKIRIM      .BLOCK 1
DATAKIRIM    .BLOCK 1
```

```
      .ORG $0
      LJMP MULAI
      .ORG $23
      LJMP SERINT
      .ORG $100
MULAI:  MOV  SP,#$20
        MOV  PSW,#0
        MOV  R0,#BUFDATANOMOR
        SETB LED
        MOV  STKIRIM,#0
        MOV  STARTDATA,#$02
        MOV  ENDDATA,#$0D
        CLR  KODE
        LCALL INITSERIAL
        SETB ES
        SETB EA

LOOP:   MOV  A,STKIRIM
        CJNE A,#$1,KIRIMF
        LJMP LOOP
```


KIRIMF:

```
MOV  A,#$FF
MOV  DATAKIRIM,A
LCALL KIRIMKEPC
LJMP LOOP
```

```
SERINT:  JBC  RI,GETDATASERIAL
          RETI
```

GETDATASERIAL:

```
PUSH  ACC
MOV   A,SBUF
CLR   RI
CJNE  A,#$03,ISIKEBUFFER
LCALL PROSES
MOV   R0,#BUFDATANOMOR
POP   ACC
RETI
```

ISIKEBUFFER:

```
MOV   @R0,A
INC   R0

POP   ACC
RETI
```

PROSES:

```
MOV   R0,#BUFDATANOMOR
INC   R0
INC   R0
INC   R0
MOV   A,@R0
LCALL CEKA_F
```

```
ANL  A,#$0F
MOV  DATAS1,A
INC  R0
MOV  A,@R0
LCALL CEKA_F
ANL  A,#$0F
MOV  DATAS2,A
INC  R0
MOV  A,@R0
LCALL CEKA_F
ANL  A,#$0F
MOV  DATAS3,A
INC  R0
MOV  A,@R0
LCALL CEKA_F
ANL  A,#$0F
MOV  DATAS4,A
INC  R0
MOV  A,@R0
LCALL CEKA_F
ANL  A,#$0F
MOV  DATAS5,A
INC  R0
MOV  A,@R0
LCALL CEKA_F
ANL  A,#$0F
MOV  DATAS6,A
INC  R0
MOV  A,@R0
LCALL CEKA_F
ANL  A,#$0F
MOV  DATAS7,A
```

```
INC R0
MOV A,@R0
LCALL CEKA_F
ANL A,#$0F
MOV DATAS8,A
```

;-----PENGGABUNGAN

```
MOV A,DATAS1
SWAP A
ANL A,#$F0
MOV DATANOHEX1,A
MOV A,DATAS2
ANL A,#$0F
ORL A,DATANOHEX1
MOV DATANOHEX1,A
```

```
MOV A,DATAS3
SWAP A
ANL A,#$F0
MOV DATANOHEX2,A
MOV A,DATAS4
ANL A,#$0F
ORL A,DATANOHEX2
MOV DATANOHEX2,A
```

```
MOV A,DATAS5
SWAP A
ANL A,#$F0
MOV DATANOHEX3,A
MOV A,DATAS6
ANL A,#$0F
ORL A,DATANOHEX3
MOV DATANOHEX3,A
```

```
MOV  A,DATAS7
SWAP A
ANL  A,#$F0
MOV  DATANOHEX4,A
MOV  A,DATAS8
ANL  A,#$0F
ORL  A,DATANOHEX4
MOV  DATANOHEX4,A
```

```
LCALL HEXTODES
```

```
MOV  A,DATADECHASIL1
LCALL ANDF0
MOV  DATANOMORASCII1,A
MOV  A,DATADECHASIL1
LCALL AND0F
MOV  DATANOMORASCII2,A
```

```
MOV  A,DATADECHASIL2
LCALL ANDF0
MOV  DATANOMORASCII3,A
MOV  A,DATADECHASIL2
LCALL AND0F
MOV  DATANOMORASCII4,A
```

```
MOV  A,DATADECHASIL3
LCALL ANDF0
MOV  DATANOMORASCII5,A
MOV  A,DATADECHASIL3
LCALL AND0F
MOV  DATANOMORASCII6,A
```

```
MOV  A,DATADECHASIL4
LCALL ANDF0
```

```
MOV  DATANOMORASCII7,A
MOV  A,DATADECHASIL4
LCALL AND0F
MOV  DATANOMORASCII8,A
```

```
MOV  A,DATADECHASIL5
LCALL ANDF0
MOV  DATANOMORASCII9,A
MOV  A,DATADECHASIL5
LCALL AND0F
MOV  DATANOMORASCII10,A
```

```
MOV  STKIRIM,#1
```

```
CLR  LED
LCALL DELAY
SETB LED
LCALL DELAY
CLR  LED
LCALL DELAY
SETB LED
LCALL DELAY
```

```
;ADA DATA;
```

```
CLR  LED
SETB KODE
MOV  R0,#DATANOMORASCII1
MOV  R1,#11
```

```
KIRIMLAGI:
```

```
MOV  A,@R0
MOV  DATAKIRIM,A
LCALL KIRIMKEPC
INC  R0
DJNZ R1,KIRIMLAGI
```

```

SETB  LED
      MOV  STKIRIM,#0
      RET

KIRIMKEPC:

TG1:   JB  SS,TG1
      MOV  A,DATAKIRIM
      MOV  P1,A

TG2:   JB  SS,TG2
      MOV  A,DATAKIRIM
      RL  A
      RL  A
      RL  A
      RL  A
      ANL  A,#$F0
      MOV  P1,A
      LCALL DELAYKIRIM
      RET

ANDF0:  SWAP  A
      ANL  A,#$0F
      ADD  A,#$30
      RET

ANDOF:
      ANL  A,#$0F
      ADD  A,#$30
      RET

CEKA_F:
CEKA:   CJNE  A,#'A',CEKB
      MOV  A,#$0A
      ADD  A,#$30
      RET

CEKB:   CJNE  A,#'B',CEKC

```

```

MOV  A,#$0B
ADD  A,#$30
RET
CEKC:  CJNE  A,#'C',CEKD
MOV  A,#$0C
ADD  A,#$30
RET
CEKD:  CJNE  A,#'D',CEKE
MOV  A,#$0D
ADD  A,#$30
RET
CEKE:  CJNE  A,#'E',CEKF
MOV  A,#$0E
ADD  A,#$30
RET
CEKF:  CJNE  A,#'F',CEKPA
MOV  A,#$0F
ADD  A,#$30
CEKPA:
RET

HEXTODES:
MOV  DATADECHASIL1,#0
MOV  DATADECHASIL2,#0
MOV  DATADECHASIL3,#0
MOV  DATADECHASIL4,#0
MOV  DATADECHASIL5,#0

MOV  A,DATANOHEX4
KE1:  CLR  C
RRC  A
JNC  KE2
LCALL HITBIT1
LCALL HITUNG

```

```
KE2:    CLR  C
        RRC  A
        JNC  KE3
        LCALL HITBIT2
        LCALL HITUNG
KE3:    CLR  C
        RRC  A
        JNC  KE4
        LCALL HITBIT3
        LCALL HITUNG
KE4:    CLR  C
        RRC  A
        JNC  KE5
        LCALL HITBIT4
        LCALL HITUNG
KE5:    CLR  C
        RRC  A
        JNC  KE6
        LCALL HITBIT5
        LCALL HITUNG
KE6:    CLR  C
        RRC  A
        JNC  KE7
        LCALL HITBIT6
        LCALL HITUNG
KE7:    CLR  C
        RRC  A
        JNC  KE8
        LCALL HITBIT7
        LCALL HITUNG
KE8:    CLR  C
        RRC  A
        JNC  KE9
        LCALL HITBIT8
```


LCALL HITUNG

KE9:

MOV A,DATANOHEX3

CLR C

RRC A

JNC KE10

LCALL HITBIT9

LCALL HITUNG

KE10: CLR C

RRC A

JNC KE11

LCALL HITBIT10

LCALL HITUNG

KE11: CLR C

RRC A

JNC KE12

LCALL HITBIT11

LCALL HITUNG

KE12: CLR C

RRC A

JNC KE13

LCALL HITBIT12

LCALL HITUNG

KE13: CLR C

RRC A

JNC KE14

LCALL HITBIT13

LCALL HITUNG

KE14: CLR C

RRC A

JNC KE15

LCALL HITBIT14

LCALL HITUNG

KE15: CLR C

```
RRC  A
    JNC  KE16
    LCALL HITBIT15
    LCALL HITUNG
KE16:  CLR  C
    RRC  A
    JNC  KE17
    LCALL HITBIT16
    LCALL HITUNG
KE17:
    MOV  A,DATANOHEX2
    CLR  C
    RRC  A
    JNC  KE18
    LCALL HITBIT17
    LCALL HITUNG
KE18:  CLR  C
    RRC  A
    JNC  KE19
    LCALL HITBIT18
    LCALL HITUNG
KE19:  CLR  C
    RRC  A
    JNC  KE20
    LCALL HITBIT19
    LCALL HITUNG
KE20:  CLR  C
    RRC  A
    JNC  KE21
    LCALL HITBIT20
    LCALL HITUNG
KE21:  CLR  C
    RRC  A
    JNC  KE22
```

```
LCALL HITBIT21
    LCALL HITUNG

KE22:    CLR  C
        RRC  A
        JNC  KE23
        LCALL HITBIT22
        LCALL HITUNG

KE23:    CLR  C
        RRC  A
        JNC  KE24
        LCALL HITBIT23
        LCALL HITUNG

KE24:    CLR  C
        RRC  A
        JNC  KE25
        LCALL HITBIT24
        LCALL HITUNG

KE25:
        MOV  A,DATANOHEX1
        CLR  C
        RRC  A
        JNC  KE26
        LCALL HITBIT25
        LCALL HITUNG

KE26:    CLR  C
        RRC  A
        JNC  KE27
        LCALL HITBIT26
        LCALL HITUNG

KE27:    CLR  C
        RRC  A
        JNC  KE28
        LCALL HITBIT27
```

```

LCALL HITUNG
KE28:   CLR  C
        RRC  A
        JNC  KE29
        LCALL HITBIT28
        LCALL HITUNG
KE29:   CLR  C
        RRC  A
        JNC  KE30
        LCALL HITBIT29
        LCALL HITUNG
KE30:   CLR  C
        RRC  A
        JNC  KE31
        LCALL HITBIT30
        LCALL HITUNG
KE31:   CLR  C
        RRC  A
        JNC  KE32
        LCALL HITBIT31
        LCALL HITUNG
KE32:   CLR  C
        RRC  A
        JNC  KE33
        LCALL HITBIT32
        LCALL HITUNG
KE33:

        RET

HITUNG:  PUSH  ACC
        MOV  A,DATADECHASIL5
        ADD  A,DATAPENAMBAH5
        DA   A

```

```
MOV  DATADECHASIL5,A

MOV  A,DATADECHASIL4
ADDC A,DATAPENAMBAH4
DA   A
MOV  DATADECHASIL4,A

MOV  A,DATADECHASIL3
ADDC A,DATAPENAMBAH3
DA   A
MOV  DATADECHASIL3,A

MOV  A,DATADECHASIL2
ADDC A,DATAPENAMBAH2
DA   A
MOV  DATADECHASIL2,A

MOV  A,DATADECHASIL1
ADDC A,DATAPENAMBAH1
DA   A
MOV  DATADECHASIL1,A
POP  ACC
RET
```

;-----

HITBIT1:

```
MOV  DATAPENAMBAH5,#$01
MOV  DATAPENAMBAH4,#$00
MOV  DATAPENAMBAH3,#$00
MOV  DATAPENAMBAH2,#$00
MOV  DATAPENAMBAH1,#$00
RET
```

HITBIT2:

```
MOV DATAPENAMBAH5,#$02
MOV DATAPENAMBAH4,#$00
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT3:

```
MOV DATAPENAMBAH5,#$04
MOV DATAPENAMBAH4,#$00
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT4:

```
MOV DATAPENAMBAH5,#$08
MOV DATAPENAMBAH4,#$00
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT5:

```
MOV DATAPENAMBAH5,#$16
MOV DATAPENAMBAH4,#$00
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT6:

```
MOV DATAPENAMBAH5,#$32
MOV DATAPENAMBAH4,#$00
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
```

RET

HITBIT7:

```
MOV DATAPENAMBAH5,#$64
MOV DATAPENAMBAH4,#$00
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT8:

```
MOV DATAPENAMBAH5,#$28
MOV DATAPENAMBAH4,#$01
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

;-----

HITBIT9:

```
MOV DATAPENAMBAH5,#$56
MOV DATAPENAMBAH4,#$02
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT10:

```
MOV DATAPENAMBAH5,#$12
MOV DATAPENAMBAH4,#$05
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT11:

```
MOV DATAPENAMBAH5,#$24
```

```
MOV DATAPENAMBAH4,#$10
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT12:

```
MOV DATAPENAMBAH5,#$48
MOV DATAPENAMBAH4,#$20
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT13:

```
MOV DATAPENAMBAH5,#$96
MOV DATAPENAMBAH4,#$40
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT14:

```
MOV DATAPENAMBAH5,#$92
MOV DATAPENAMBAH4,#$81
MOV DATAPENAMBAH3,#$00
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT15:

```
MOV DATAPENAMBAH5,#$84
MOV DATAPENAMBAH4,#$63
MOV DATAPENAMBAH3,#$01
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```


HITBIT16:

```
MOV DATAPENAMBAH5,#$68
MOV DATAPENAMBAH4,#$27
MOV DATAPENAMBAH3,#$03
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

;------

HITBIT17:

```
MOV DATAPENAMBAH5,#$36
MOV DATAPENAMBAH4,#$55
MOV DATAPENAMBAH3,#$06
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT18:

```
MOV DATAPENAMBAH5,#$72
MOV DATAPENAMBAH4,#$10
MOV DATAPENAMBAH3,#$13
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT19:

```
MOV DATAPENAMBAH5,#$44
MOV DATAPENAMBAH4,#$21
MOV DATAPENAMBAH3,#$26
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT20:

```
MOV DATAPENAMBAH5,#$88
MOV DATAPENAMBAH4,#$42
MOV DATAPENAMBAH3,#$52
```

```
MOV DATAPENAMBAH2,#$00
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT21:

```
MOV DATAPENAMBAH5,#$76
MOV DATAPENAMBAH4,#$85
MOV DATAPENAMBAH3,#$04
MOV DATAPENAMBAH2,#$01
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT22:

```
MOV DATAPENAMBAH5,#$52
MOV DATAPENAMBAH4,#$71
MOV DATAPENAMBAH3,#$09
MOV DATAPENAMBAH2,#$02
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT23:

```
MOV DATAPENAMBAH5,#$04
MOV DATAPENAMBAH4,#$43
MOV DATAPENAMBAH3,#$19
MOV DATAPENAMBAH2,#$04
MOV DATAPENAMBAH1,#$00
RET
```

HITBIT24:

```
MOV DATAPENAMBAH5,#$08
MOV DATAPENAMBAH4,#$86
MOV DATAPENAMBAH3,#$38
MOV DATAPENAMBAH2,#$08
MOV DATAPENAMBAH1,#$00
RET
```

;-----

HITBIT25:

```
MOV DATAPENAMBAH5,#$16
```

```
MOV  DATAPENAMBAH4,#$72
      MOV  DATAPENAMBAH3,#$77
      MOV  DATAPENAMBAH2,#$16
      MOV  DATAPENAMBAH1,#$00
      RET
```

HITBIT26:

```
MOV  DATAPENAMBAH5,#$32
      MOV  DATAPENAMBAH4,#$44
      MOV  DATAPENAMBAH3,#$55
      MOV  DATAPENAMBAH2,#$33
      MOV  DATAPENAMBAH1,#$00
      RET
```

HITBIT27:

```
MOV  DATAPENAMBAH5,#$64
      MOV  DATAPENAMBAH4,#$88
      MOV  DATAPENAMBAH3,#$10
      MOV  DATAPENAMBAH2,#$67
      MOV  DATAPENAMBAH1,#$00
      RET
```

HITBIT28:

```
MOV  DATAPENAMBAH5,#$28
      MOV  DATAPENAMBAH4,#$77
      MOV  DATAPENAMBAH3,#$21
      MOV  DATAPENAMBAH2,#$34
      MOV  DATAPENAMBAH1,#$01
      RET
```

HITBIT29:

```
MOV  DATAPENAMBAH5,#$56
      MOV  DATAPENAMBAH4,#$54
      MOV  DATAPENAMBAH3,#$43
      MOV  DATAPENAMBAH2,#$68
      MOV  DATAPENAMBAH1,#$02
      RET
```

HITBIT30:

```
MOV  DATAPENAMBAH5,#$12
      MOV  DATAPENAMBAH4,#$09
      MOV  DATAPENAMBAH3,#$87
      MOV  DATAPENAMBAH2,#$36
      MOV  DATAPENAMBAH1,#$05
      RET
```

HITBIT31:

```
MOV  DATAPENAMBAH5,#$24
MOV  DATAPENAMBAH4,#$18
MOV  DATAPENAMBAH3,#$74
MOV  DATAPENAMBAH2,#$73
MOV  DATAPENAMBAH1,#$10
      RET
```

HITBIT32:

```
MOV  DATAPENAMBAH5,#$48
MOV  DATAPENAMBAH4,#$36
MOV  DATAPENAMBAH3,#$48
MOV  DATAPENAMBAH2,#$47
MOV  DATAPENAMBAH1,#$21
      RET
```

;-----

INITSERIAL:

```
MOV  TMOD,#20H
MOV  TCON,#41H
MOV  TH1,#0FDH
MOV  SCON,#50H
      RET
```

DELAY: MOV R5,#\$01

DELAY1: MOV R6,#\$FF

DELAY2: MOV R7,#\$FF

DELAY3: DJNZ R7,DELAY3

 DJNZ R6,DELAY2

 DJNZ R5,DELAY1

RET

DELAYKIRIM:

MOV R6,#\$01

DELAYK1: MOV R7,\$FF

DELAYK2: DJNZ R7,DELAYK2

DJNZ R6,DELAYK1

RET

.END

```
unit FUnit1;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Buttons, ExtCtrls;
```

```
type
```

```
Tfutama = class(TForm)
```

```
  pjam: TPanel;
```

```
  ptanggal: TPanel;
```

```
  BitBtn1: TBitBtn;
```

```
  BitBtn2: TBitBtn;
```

```
  BitBtn3: TBitBtn;
```

```
  Panel1: TPanel;
```

```
  Label1: TLabel;
```

```
  Pnamamasuk: TPanel;
```

```
  Pkartumasuk: TPanel;
```

```
  Timer1: TTimer;
```

```
  procedure BitBtn1Click(Sender: TObject);
```

```
  procedure Timer1Timer(Sender: TObject);
```

```
  procedure BitBtn2Click(Sender: TObject);
```

```
  procedure FormCreate(Sender: TObject);
```

```
private
```

```
  { Private declarations }
```

```
public
```

```
  { Public declarations }
```

```
end;
```

```
var
```

```
  futama: Tfutama;
```

```
  databaca,datal1,datah1,databacaH,databacaL:integer;
```

```
  datal,datah:byte;
```

```
datakartu:string;
  datas1,
  datas2,
  datas3,
  datas4,
  datas5,
  datas6,
  datas7,
  datas8,
  datas9,
  datas10:string;
  datab1,
  datab2,
  datab3,
  datab4,
  datab5,
  datab6,
  datab7,
  datab8,
  datab9,
  datab10:byte;
  datastatus:byte;
  datakartumasuk:string;
  stketemu:boolean;
  Nama,NomorKartu:string;
```

implementation

```
uses Unittambahdata,unitbrowse;
```

```
{ $R *.dfm }
```

```
procedure Tfutama.BitBtn1Click(Sender: TObject);
```

```
begin
```

```
  Ftambahdata.show;
```

```
end;
```

```

procedure WritePortInit;
begin
  asm
    mov    dx,$37a;
    mov    al,$04;
    out    dx,al
  end;
end;
procedure WritePortHigh;
begin
  asm
    mov    dx,$378;
    mov    al,$03;
    out    dx,al
  end;
end;
procedure WritePortLow;
begin
  asm
    mov    dx,$378;
    mov    al,$00;
    out    dx,al;
  end;
end;

procedure bacaportLow;
begin
  asm
    mov dx,$37a;
    in al,dx;
    mov datal,al;
  end;
  datal:=datal and $0f;
end;

```



```

procedure bacaportHigh;
begin

    asm
        mov dx,$379;
        in al,dx;
        mov datah,al;
    end;

    datastatus:=datah and $08;
    datah:=(datah shr 4) and $0f;
end;

procedure Tfutama.Timer1Timer(Sender: TObject);
var delay1,delay2:longint;
    code:integer;
begin
    pjam.caption:=formatdatetime('hh:mm:ss',time);
    ptanggal.caption:=formatdatetime('dd/mm/yyyy',date);

    bacaportHigh;
    if datastatus=$08 then
    begin
        writeportlow;
        bacaportHIGH;
        writeporthigh;
        databacah:=datah;
        writeportlow;
        bacaporthigh;
        writeporthigh;
        databacal:=datah;

        databaca:=databacah+databacal;
        if (databacah=$0f) or (databacal=$0f) then databaca:=0;
    end;
end;

```

```

if ((databacah<>$0f) and (databacal<>$0f)) then
begin

datakartumasuk:=datakartumasuk+inttostr(databacah)+inttostr(databacal);
{ pkartumasuk.caption:=datakartumasuk;}
if length(datakartumasuk)=23 then
begin

    datakartu:=copy(datakartumasuk,3,20);
    datas1:=copy(datakartu,1,2);
    datas2:=copy(datakartu,3,2);
    datas3:=copy(datakartu,5,2);
    datas4:=copy(datakartu,7,2);
    datas5:=copy(datakartu,9,2);
    datas6:=copy(datakartu,11,2);
    datas7:=copy(datakartu,13,2);
    datas8:=copy(datakartu,15,2);
    datas9:=copy(datakartu,17,2);
    datas10:=copy(datakartu,19,2);

    val('$'+datas1,datab1,code);
    val('$'+datas2,datab2,code);
    val('$'+datas3,datab3,code);
    val('$'+datas4,datab4,code);
    val('$'+datas5,datab5,code);
    val('$'+datas6,datab6,code);
    val('$'+datas7,datab7,code);
    val('$'+datas8,datab8,code);
    val('$'+datas9,datab9,code);
    val('$'+datas10,datab10,code);

    datakartu:=chr(datab1)+chr(datab2)+chr(datab3)+chr(datab4)+chr(datab5)+chr(datab
6)+chr(datab7)+chr(datab8)+chr(datab9)+chr(datab10);
    pkartumasuk.caption:=datakartu;

```

```

with ftambahdata do
begin
tablekaryawan.first;
tablekaryawan.refresh;
stketemu:=false;
while not tablekaryawan.eof do
begin
if tablekaryawan.fieldbyname('NOMORKARTU').asstring=datakartu then
begin
stketemu:=true;
nama:=tablekaryawan.fieldbyname('NAMA').asstring;
nomorkartu:=tablekaryawan.fieldbyname('NOMORKARTU').asstring;
tablekaryawan.last;
end;
tablekaryawan.next;
end;
end;

if stketemu then
begin
pnamamasuk.caption:=na ma;
with fbrowsedata do
begin

tabledata.refresh;
tabledata.last;
tabledata.insert;
tabledata.fieldbyname('TANGGAL').asstring:=ptanggal.caption;
tabledata.fieldbyname('JAM').asstring:=pjam.caption;
tabledata.fieldbyname('NAMA').asstring:=nama;
tabledata.fieldbyname('NO_KARTU').asstring:=nomorkartu;
tabledata.refresh;
end;
end;

```

```

end else
begin
    pnamamasuk.caption:='TIDAK ADA';
    pkartumasuk.caption:='';
end;

datakartumasuk:='';
databaca:=0;
end;
end;
{CAPTION:=INTTOSTR(DATABacaH)+' '+INTTOSTR(DATABacal);}
end;

end;

procedure Tfutama.BitBtn2Click(Sender: TObject);
begin
    fbrowsedata.show;
end;

procedure Tfutama.FormCreate(Sender: TObject);
begin
    stketemu:=false;
    datakartumasuk:='';
    timer1.interval:=10;
    asm
        mov dx,$37a;
        in al,dx;
        mov datal,al;
    end;

    datal:=((datal and $f0) or $04);

```

asm

mov dx,\$37a;

mov al,data1;

out dx,al;

end;

writeporthigh;

end;

end.

LAMPIRAN B

DATA KOMPONEN KARTU

Contactless Card Specification

Model No.	ISO Card	Imm ISO Card	Clamshell Card
Reading Distance	80mm	100mm	160mm
Base on GK4001 chip, with EM9917 Reader			
Operating temperature	-20°C to 50°C	-20°C to 50°C	-20°C to 70°C
Storage temperature	-20°C to 50°C	-20°C to 50°C	-40°C to 80°C
Material	PVC/PET	PVC	ABS
Operating Frequency	125KHz / 13.56MHz	125KHz	125KHz / 13.56MHz
Antenna	Copper Coil	Copper Coil	Copper Coil
Dimensions	85.6mm+0.12x53.98mm+0.05x0.76mm+0.08	85.6mm+0.12x53.98mm+0.05x10mm+0.08	85.6mm+0.12x53.98mm+0.05x1.84mm+0.04
Technology			
GK4001 (125KHz, 64bit, R/O)	Y	Y	Y
H4102 (125KHz, 64bit, R/O)	Y	Y	Y
I-Code SLI (13.56MHz, 1024bit, R/W)	Y	Y	Y
Mifare S50 (13.56MHz, 1KByte, R/W)	Y	Y	Y
Temic E5557 (125KHz, 330bit, R/W)	Y	Y	Y
Hitag 1 (125KHz, 2048bit, R/W)	Y	Y	Y
Hitag 2 (125KHz, 256bit, R/W)	Y	Y	Y
Legic 256 (13.56MHz, 256Byte, R/W)	Y	Y	Y

Application: Door Access Control, Burglar Prevention, Patrol Management, Personal Identification, Car Park Control, Member Club Management, etc.

LAMPIRAN C

DATA KOMPONEN READER

ID - 2, ID - 12, ID - 20

(including ID-0, ID-10, ID-15)

Reader Specification

Low Cost – Short Range
RFID Compact Reader Modules



Advanced Reader Development
Advanced Digital Reader Technology

Manual Rev 0.2

PRODUCT DESCRIPTION

The ID-2, ID-12 and ID-20 are hybrid RFID reader modules that provide simple, consistent and flexible implementation of this technology into existing swipe card based ID equipment.

With the simple addition of an antenna and a capacitor to suit your specific design and needs, equipment such as access control units, time and attendance equipment or workstation logging equipment can be converted to the popular wireless ID designs.

The ID-2, ID-12 and ID-20 are readers for the popular EM4001 format 125KHz tags. Read

ranges of 1-15 cm are possible with ISO cards (25cm with a well tuned matched antenna on the ID-2). With CMOS/TTL Serial, Wiegand26 and NEW with this series, Magnetic ABA Track 2 format, swipe cards can be emulated. Furthermore, the readers are encapsulated for environmental protection and potential damage or changes during manufacture or handling.

The ID-2 requires an external antenna, ideal for custom cased installations. The ID-12 & ID-20 has an internal antenna, allowing fast integration, particularly when space is a premium.

ANTENNA DESIGN

Maximum coil sizes:-

ISO Card	-	15cm x 15cm for ISO Card
Glass Tag	-	10cm x 10cm for Glass Tag

ANTENNA DESIGN PRINCIPLES

- Generally the bigger the antenna the better, as long as there is enough field strength to excite the tag.
- It is possible to calculate the optimum size of an antenna, but there is always an element of try it and see.

Elements affecting reading range.

- In areas of interference reduction in reading range is unavoidable.
- Reducing the coil size will provide more concentrated field strength and coupling.

Choice Of Tuning Capacitor

Quality of capacitor can significantly affect the quality of your system. For quality and reliability we recommend the following capacitor types:-

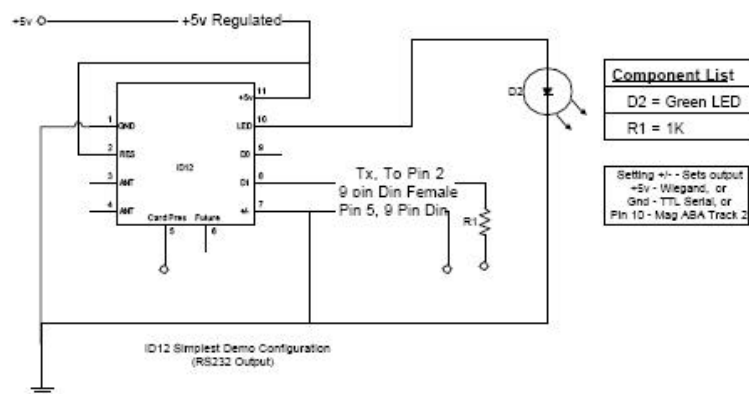
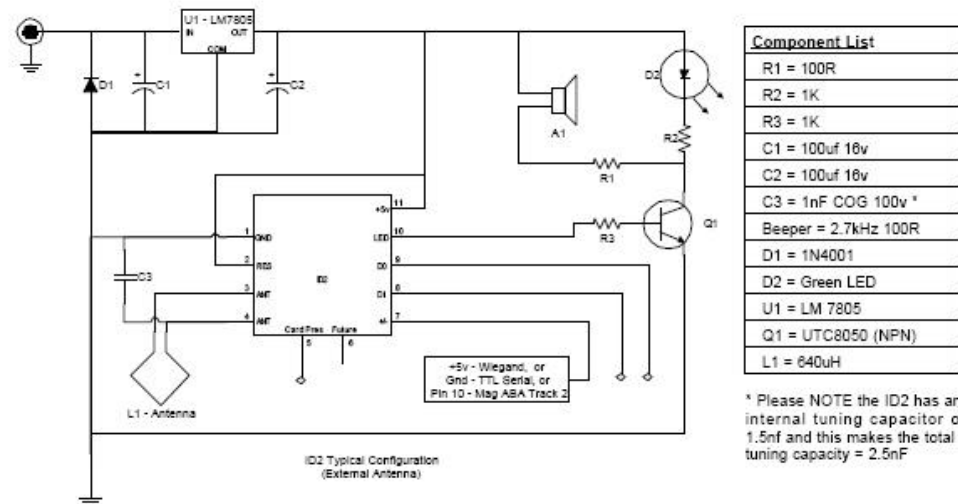
- Polypropylene
- COG/NPO
- Polyethylene Sulfide
- Mica
- Polycarbonate

Capacitors rated voltage (MUST cope with RMS voltage at 125kHz). It is recommend starting with 630v 1n5 Polypropylene. After experimentation, the voltage and capacitor type can come down to measured values. Do not go by DC voltage rating. The tolerance should be 2% preferred, 5% acceptable.

ANTENNA DESIGN STEPS

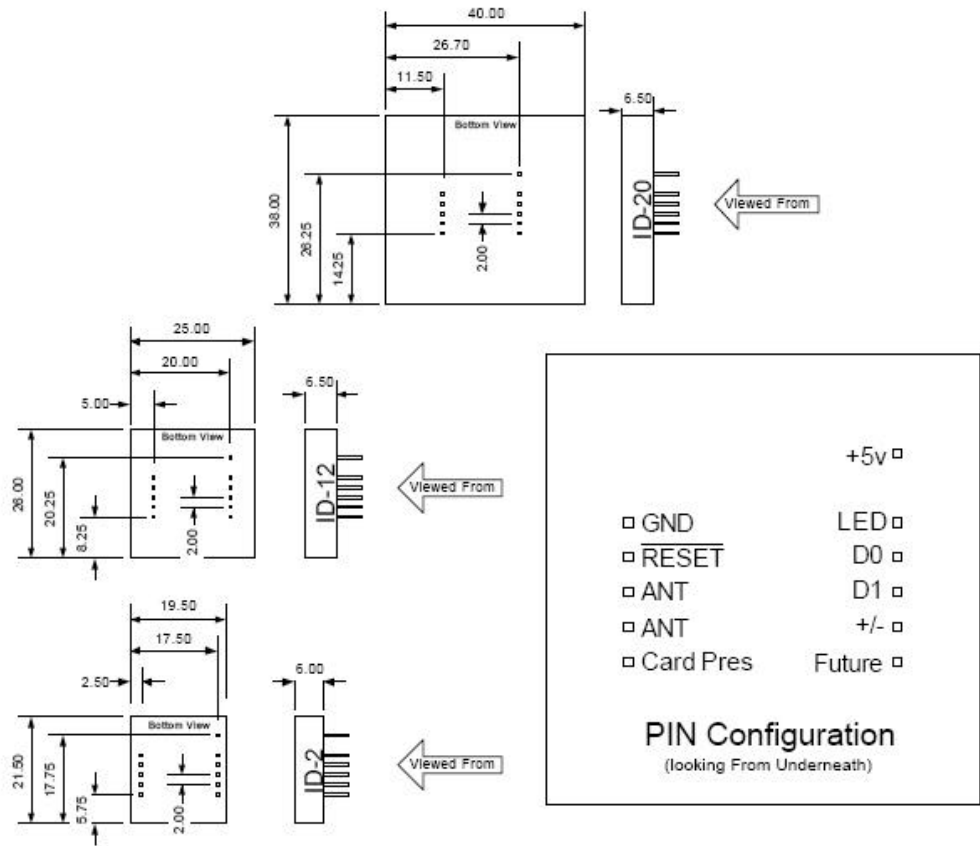
1. If antenna design is governed by enclosure size and is less than 4cmx4cm, wind the coil to 640uH and skip steps 2-4.
2. If target transducer is small, do not wind the coil more than 10cmx10cm
3. In heavy interference, start with a large coil and reduce the size to see if there is significant improvement.
4. If there is a large mass of metal in proximity, the inductance will be changed, usually reducing the original value. Start with a higher inductance. Using a smaller coil may help, by increasing the field strength and coupling.
5. The ID-2 has a maximum current allowed of 200mA. Corresponding to 100v Pk to Pk. If the antenna voltage is higher, the designer can risk it or lower the Q by adding a series resistor in the coil.

EXAMPLE CIRCUITS



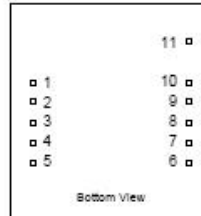
PIN LAYOUTS AND SPACING

Please note, the pin layouts and spacing are the same for the modules, the casing sizes differ to allow for internal antennas where applicable.



SPECIFICATIONS ID-2, ID-12, ID-20

Low Cost Short-Range Proximity Readers



The ID Series short-range readers come in three different sizes and read ranges. The ID-12 and ID-20 come with internal antennas, and have read ranges of 12+ cm and 16+ cm, respectively. With an external antenna, the ID-2 can deliver read ranges of up to 25 cm. All three readers support ASCII, Wiegand26 and Magnetic ABA Track2 data formats.



Operational and Physical Characteristics

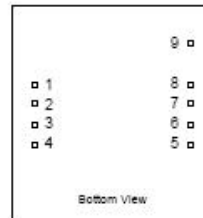
Parameters	ID-2	ID-12	ID-20
Read Range	N/A (no internal antenna)	12+ cm	15+ cm
Dimensions	21 mm x 19 mm x 6 mm	26 mm x 25 mm x 7 mm	40 mm x 40 mm x 9 mm
Frequency	125 kHz	125 kHz	125 kHz
Card Format	EM 4001 or compatible	EM 4001 or compatible	EM 4001 or compatible
Encoding	Manchester 64-bit, modulus 64	Manchester 64-bit, modulus 64	Manchester 64-bit, modulus 64
Power Requirement	5 VDC @ 13mA nominal	5 VDC @ 30mA nominal	5 VDC @ 55mA nominal
I/O Output Current	+/-200mA PK	-	-
Voltage Supply Range	+4.6V through +5.4V	+4.6V through +5.4V	+4.6V through +5.4V

Pin Description & Output Data Formats

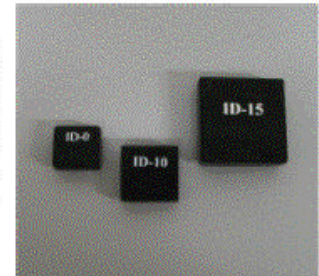
Pin No.	Description	ASCII	Magnet Emulation	Wiegand26
Pin 1	Zero Volts and Tuning Capacitor Ground	GND 0V	GND 0V	GND 0V
Pin 2	Strap to +5V	Reset Bar	Reset Bar	Reset Bar
Pin 3	To External Antenna and Tuning Capacitor	Antenna	Antenna	Antenna
Pin 4	To External Antenna	Antenna	Antenna	Antenna
Pin 5	Card Present	No function	Card Present	No function
Pin 6	Future	Future	Future	Future
Pin 7	Format Selector (+/-)	Strap to GND	Strap to Pin 10	Strap to +5V
Pin 8	Data 1	CMOS	Data	One Output
Pin 9	Data 0	TTL Data (inverted)	Clock	Zero Output
Pin 10	2.7 kHz Logic	Beeper / LED	Beeper / LED	Beeper / LED
Pin 11	DC Voltage Supply	+5V	+5V	+5V

SPECIFICATIONS ID-0, ID-10, ID-15 (NOT FOR NEW DESIGNS)

Low Cost Short-Range Proximity Readers



The ID Series short-range readers come in three different sizes and read ranges. Both the ID-10 and ID-15 come with internal antennas, and have read ranges of 12+ cm and 16+ cm, respectively. With an external antenna, the ID-0 Mk(ii) can deliver read ranges of up to 25 cm. All three readers support ASCII and Wiegand26 data formats.



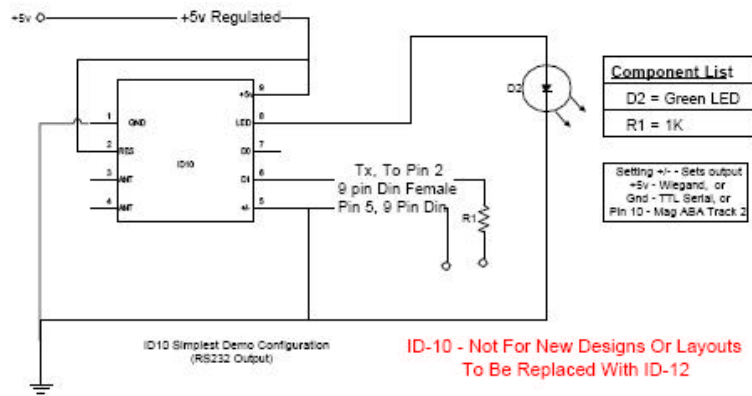
Operational and Physical Characteristics

Parameters	ID-0	ID-10	ID-15
Read Range	N/A (no internal antenna)	12+ cm	15+ cm
Dimensions	21 mm x 19 mm x 6 mm	26 mm x 25 mm x 7 mm	40 mm x 40 mm x 9 mm
Frequency	125 kHz	125 kHz	125 kHz
Card Format	EM 4001 or compatible	EM 4001 or compatible	EM 4001 or compatible
Encoding	Manchester 64-bit, modulus 64	Manchester 64-bit, modulus 64	Manchester 64-bit, modulus 64
Power Requirement	5 VDC @ 13mA nominal	5 VDC @ 30mA nominal	5 VDC @ 50mA nominal
I/O Output Current	+/-200mA PK	-	-
Voltage Supply Range	+4.6V through +5.4V	+4.6V through +5.4V	+4.6V through +5.4V

Pin Description & Output Data Formats

Pin No.	Description	ASCII	Wiegand26
Pin 1	Zero Volts and Tuning Capacitor Ground	GND 0V	GND 0V
Pin 2	Strap to +5V	Reset Bar	Reset Bar
Pin 3	To External Antenna and Tuning Capacitor	Antenna	Antenna
Pin 4	To External Antenna	Antenna	Antenna
Pin 5	Format Selector (+/-)	Strap to GND	Strap to +5V
Pin 6	Data 1	CMOS	One Output
Pin 7	Data 0	TTL Data (inverted)	Zero Output
Pin 8	2.7 kHz Logic	Beeper / LED	Beeper / LED
Pin 9	DC Voltage Supply	+5V	+5V

EXAMPLE CIRCUITS



DATA FORMATS

Output Data Structure – ASCII (TTL Level RS2323)

STX (02h)	DATA (10 ASCII)	CHECKSUM	CR	LF	ETX (03h)
-----------	-----------------	----------	----	----	-----------

The checksum is the result of the XOR of the 5 binary Data bytes (the 10 ASCII data characters)

Output Data Structure – Wiegand26

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
P	E	E	E	E	E	E	E	E	E	E	E	E	O	O	O	O	O	O	O	O	O	O	O	O	P
Even parity (E)													Odd parity (O)												

P = Parity start bit and stop bit

Specifications subject to change. ARD reserves the right to change its products and the specifications given here at any time without notice.

LAMPIRAN D

DATA KOMPONEN MIKRO AT89C51

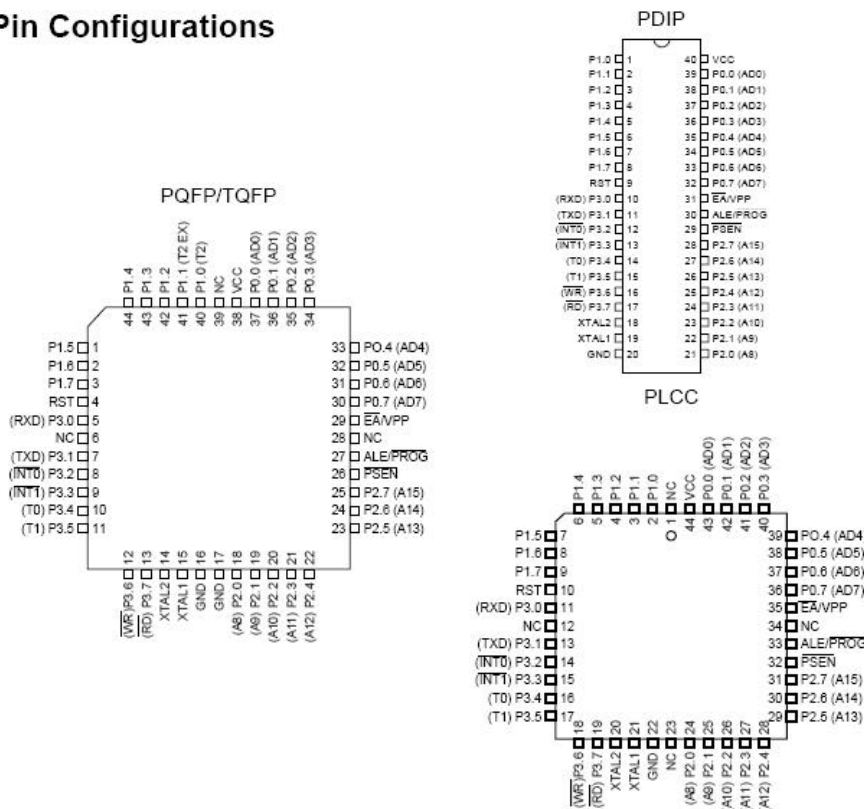
Features

- Compatible with MCS-51™ Products
- 4K Bytes of In-System Reprogrammable Flash Memory
 - Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-level Program Memory Lock
- 128 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial Channel
- Low-power Idle and Power-down Modes

Description

The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4K bytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard MCS-51 instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications.

Pin Configurations



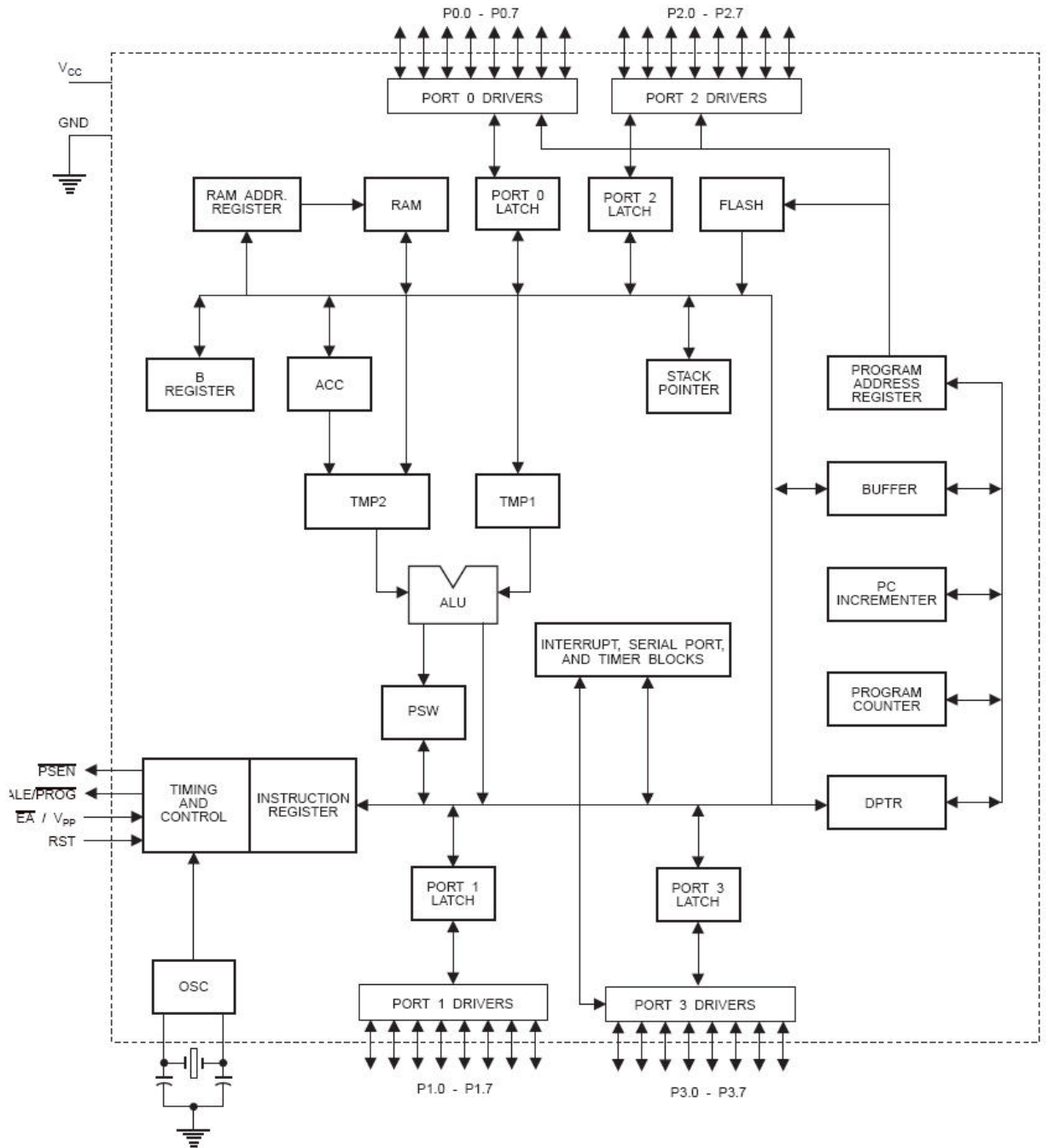
**8-bit
Microcontroller
with 4K Bytes
Flash**

AT89C51

**Not Recommended
for New Designs.
Use AT89S51.**



Block Diagram



The AT89C51 provides the following standard features: 4K bytes of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timer/counters, a five vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator and clock circuitry. In addition, the AT89C51 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. The Power-down Mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

Pin Description

VCC

Supply voltage.

GND

Ground.

Port 0

Port 0 is an 8-bit open-drain bi-directional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 may also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode P0 has internal pullups.

Port 0 also receives the code bytes during Flash programming, and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1

Port 1 is an 8-bit bi-directional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 1 also receives the low-order address bytes during Flash programming and verification.

Port 2

Port 2 is an 8-bit bi-directional I/O port with internal pullups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins they are pulled high by the internal pullups and can be used as inputs. As inputs,

Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, it uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3

Port 3 is an 8-bit bi-directional I/O port with internal pullups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C51 as listed below:

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt 0)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

Port 3 also receives some control signals for Flash programming and verification.

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/PROG

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input (PROG) during Flash programming.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE

pulse is skipped during each access to external Data Memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

PSEN

Program Store Enable is the read strobe to external program memory.

When the AT89C51 is executing code from external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external data memory.

EA/VPP

External Access Enable. EA must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, EA will be internally latched on reset.

EA should be strapped to VCC for internal program executions.

This pin also receives the 12-volt programming enable voltage (VPP) during Flash programming, for parts that require 12-volt VPP.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting oscillator amplifier.

Oscillator Characteristics

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left

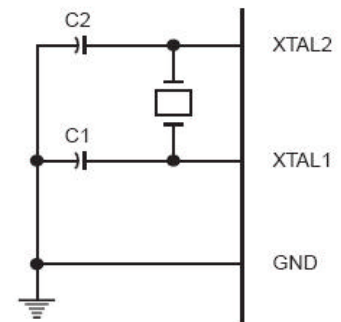
unconnected while XTAL1 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

It should be noted that when idle is terminated by a hardware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

Figure 1. Oscillator Connections

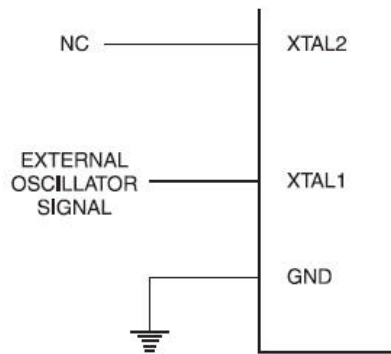


Note: C1, C2 = 30 pF ± 10 pF for Crystals
= 40 pF ± 10 pF for Ceramic Resonators

Status of External Pins During Idle and Power-down Modes

Mode	Program Memory	ALE	PSEN	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power-down	Internal	0	0	Data	Data	Data	Data
Power-down	External	0	0	Float	Data	Data	Data

Figure 2. External Clock Drive Configuration



Power-down Mode

In the power-down mode, the oscillator is stopped, and the instruction that invokes power-down is the last instruction executed. The on-chip RAM and Special Function Regis-

ters retain their values until the power-down mode is terminated. The only exit from power-down is a hardware reset. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before V_{CC} is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

Program Memory Lock Bits

On the chip are three lock bits which can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the table below.

When lock bit 1 is programmed, the logic level at the \overline{EA} pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value, and holds that value until reset is activated. It is necessary that the latched value of \overline{EA} be in agreement with the current logic level at that pin in order for the device to function properly.

Lock Bit Protection Modes

	Program Lock Bits			Protection Type
	LB1	LB2	LB3	
1	U	U	U	No program lock features
2	P	U	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, \overline{EA} is sampled and latched on reset, and further programming of the Flash is disabled
3	P	P	U	Same as mode 2, also verify is disabled
4	P	P	P	Same as mode 3, also external execution is disabled

Programming the Flash

The AT89C51 is normally shipped with the on-chip Flash memory array in the erased state (that is, contents = FFH) and ready to be programmed. The programming interface accepts either a high-voltage (12-volt) or a low-voltage (V_{CC}) program enable signal. The low-voltage programming mode provides a convenient way to program the AT89C51 inside the user's system, while the high-voltage programming mode is compatible with conventional third-party Flash or EPROM programmers.

The AT89C51 is shipped with either the high-voltage or low-voltage programming mode enabled. The respective top-side marking and device signature codes are listed in the following table.

	$V_{PP} = 12V$	$V_{PP} = 5V$
Top-side Mark	AT89C51 xxxx yyww	AT89C51 xxxx-5 yyww
Signature	(030H) = 1EH (031H) = 51H (032H) = FFH	(030H) = 1EH (031H) = 51H (032H) = 05H

The AT89C51 code memory array is programmed byte-by-byte in either programming mode. *To program any non-blank byte in the on-chip Flash Memory, the entire memory must be erased using the Chip Erase Mode.*

Programming Algorithm: Before programming the AT89C51, the address, data and control signals should be set up according to the Flash programming mode table and Figure 3 and Figure 4. To program the AT89C51, take the following steps.

1. Input the desired memory location on the address lines.
2. Input the appropriate data byte on the data lines.
3. Activate the correct combination of control signals.
4. Raise \overline{EA}/V_{PP} to 12V for the high-voltage programming mode.
5. Pulse ALE/\overline{PROG} once to program a byte in the Flash array or the lock bits. The byte-write cycle is self-timed and typically takes no more than 1.5 ms. Repeat steps 1 through 5, changing the address

and data for the entire array or until the end of the object file is reached.

Data Polling: The AT89C51 features \overline{Data} Polling to indicate the end of a write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written datum on PO.7. Once the write cycle has been completed, true data are valid on all outputs, and the next cycle may begin. Data Polling may begin any time after a write cycle has been initiated.

Ready/Busy: The progress of byte programming can also be monitored by the RDY/BSY output signal. P3.4 is pulled low after ALE goes high during programming to indicate BUSY. P3.4 is pulled high again when programming is done to indicate READY.

Program Verify: If lock bits LB1 and LB2 have not been programmed, the programmed code data can be read back via the address and data lines for verification. The lock bits cannot be verified directly. Verification of the lock bits is achieved by observing that their features are enabled.

Chip Erase: The entire Flash array is erased electrically by using the proper combination of control signals and by holding ALE/\overline{PROG} low for 10 ms. The code array is written with all "1"s. The chip erase operation must be executed before the code memory can be re-programmed.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 030H, 031H, and 032H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows.

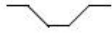

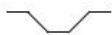


- (030H) = 1EH indicates manufactured by Atmel
- (031H) = 51H indicates 89C51
- (032H) = FFH indicates 12V programming
- (032H) = 05H indicates 5V programming

Programming Interface

Every code byte in the Flash array can be written and the entire array can be erased by using the appropriate combination of control signals. The write operation cycle is self-timed and once initiated, will automatically time itself to completion.

All major programming vendors offer worldwide support for the Atmel microcontroller series. Please contact your local programming vendor for the appropriate software revision.

Flash Programming Modes

Mode	RST	PSEN	ALE/PROG	EA/V _{PP}	P2.6	P2.7	P3.6	P3.7
Write Code Data	H	L		H/12V	L	H	H	H
Read Code Data	H	L	H	H	L	L	H	H
Write Lock	H	L		H/12V	H	H	H	H
				H/12V	H	H	L	L
				H/12V	H	L	H	L
Chip Erase	H	L	 (1)	H/12V	H	L	L	L
Read Signature Byte	H	L	H	H	L	L	L	L

Note: 1. Chip Erase requires a 10 ms PROG pulse.

Figure 3. Programming the Flash

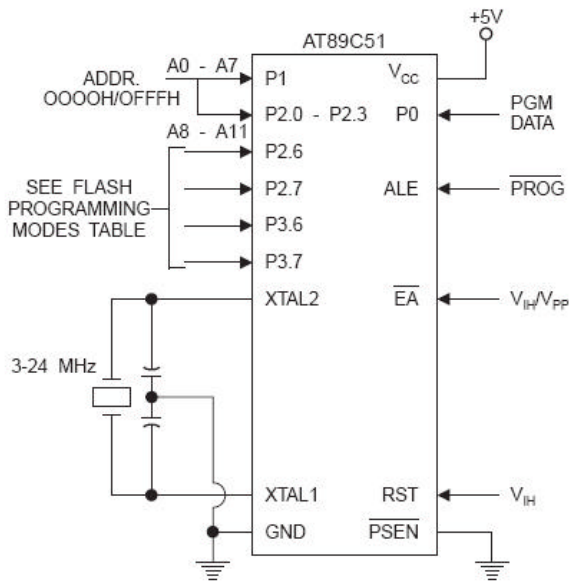
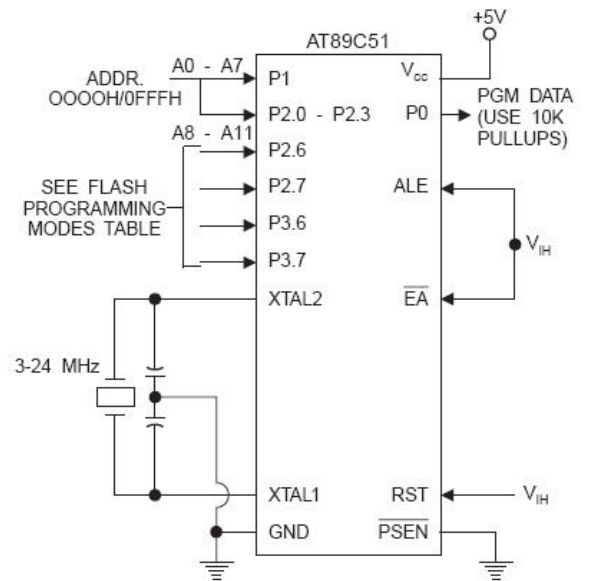
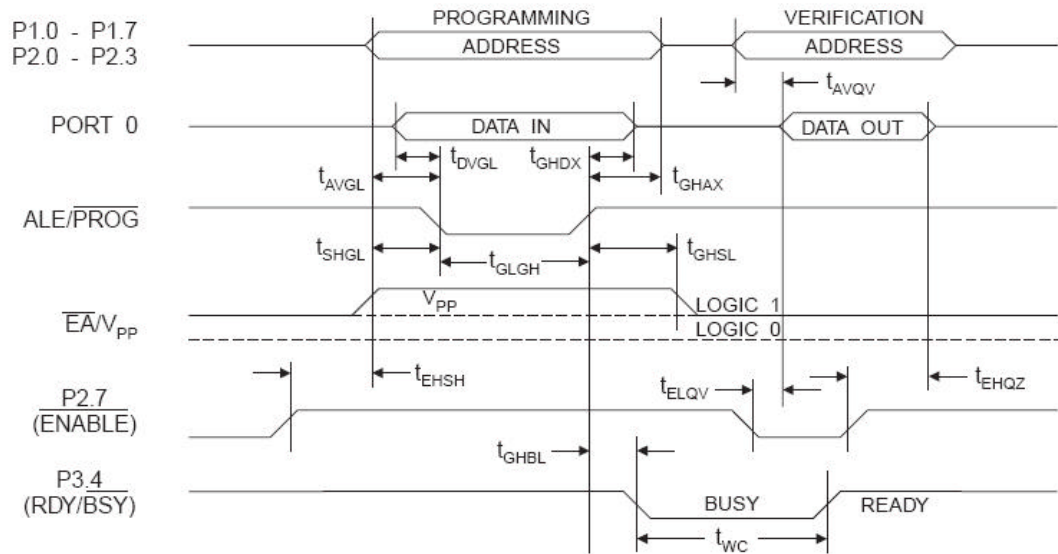


Figure 4. Verifying the Flash

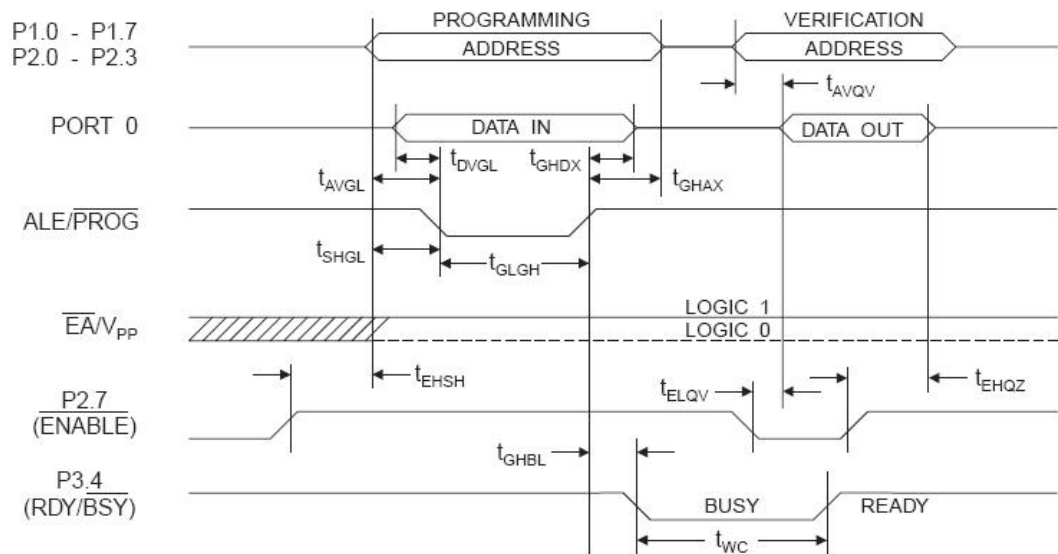




Flash Programming and Verification Waveforms - High-voltage Mode ($V_{PP} = 12V$)



Flash Programming and Verification Waveforms - Low-voltage Mode ($V_{PP} = 5V$)



Flash Programming and Verification Characteristics

$T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5.0 \pm 10\%$

Symbol	Parameter	Min	Max	Units
$V_{PP}^{(1)}$	Programming Enable Voltage	11.5	12.5	V
$I_{PP}^{(1)}$	Programming Enable Current		1.0	mA
$1/t_{CLCL}$	Oscillator Frequency	3	24	MHz
t_{AVGL}	Address Setup to $\overline{\text{PROG}}$ Low	$48t_{CLCL}$		
t_{GHAX}	Address Hold after $\overline{\text{PROG}}$	$48t_{CLCL}$		
t_{DVGL}	Data Setup to $\overline{\text{PROG}}$ Low	$48t_{CLCL}$		
t_{GHDX}	Data Hold after $\overline{\text{PROG}}$	$48t_{CLCL}$		
t_{EHS}	P2.7 ($\overline{\text{ENABLE}}$) High to V_{PP}	$48t_{CLCL}$		
t_{SHGL}	V_{PP} Setup to $\overline{\text{PROG}}$ Low	10		μs
$t_{GHSL}^{(1)}$	V_{PP} Hold after $\overline{\text{PROG}}$	10		μs
t_{GLGH}	$\overline{\text{PROG}}$ Width	1	110	μs
t_{AVQV}	Address to Data Valid		$48t_{CLCL}$	
t_{ELQV}	$\overline{\text{ENABLE}}$ Low to Data Valid		$48t_{CLCL}$	
t_{EHQZ}	Data Float after $\overline{\text{ENABLE}}$	0	$48t_{CLCL}$	
t_{GHBL}	$\overline{\text{PROG}}$ High to $\overline{\text{BUSY}}$ Low		1.0	μs
t_{WC}	Byte Write Cycle Time		2.0	ms

Note: 1. Only used in 12-volt programming mode.



Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-1.0V to +7.0V
Maximum Operating Voltage	6.6V
DC Output Current.....	15.0 mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

$T_A = -40^\circ\text{C}$ to 85°C , $V_{CC} = 5.0\text{V} \pm 20\%$ (unless otherwise noted)

Symbol	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low-voltage	(Except \overline{EA})	-0.5	$0.2 V_{CC} - 0.1$	V
V_{IL1}	Input Low-voltage (\overline{EA})		-0.5	$0.2 V_{CC} - 0.3$	V
V_{IH}	Input High-voltage	(Except XTAL1, RST)	$0.2 V_{CC} + 0.9$	$V_{CC} + 0.5$	V
V_{IH1}	Input High-voltage	(XTAL1, RST)	$0.7 V_{CC}$	$V_{CC} + 0.5$	V
V_{OL}	Output Low-voltage ⁽¹⁾ (Ports 1,2,3)	$I_{OL} = 1.6\text{ mA}$		0.45	V
V_{OL1}	Output Low-voltage ⁽¹⁾ (Port 0, ALE, PSEN)	$I_{OL} = 3.2\text{ mA}$		0.45	V
V_{OH}	Output High-voltage (Ports 1,2,3, ALE, PSEN)	$I_{OH} = -60\ \mu\text{A}$, $V_{CC} = 5\text{V} \pm 10\%$	2.4		V
		$I_{OH} = -25\ \mu\text{A}$	$0.75 V_{CC}$		V
		$I_{OH} = -10\ \mu\text{A}$	$0.9 V_{CC}$		V
V_{OH1}	Output High-voltage (Port 0 in External Bus Mode)	$I_{OH} = -800\ \mu\text{A}$, $V_{CC} = 5\text{V} \pm 10\%$	2.4		V
		$I_{OH} = -300\ \mu\text{A}$	$0.75 V_{CC}$		V
		$I_{OH} = -80\ \mu\text{A}$	$0.9 V_{CC}$		V
I_{IL}	Logical 0 Input Current (Ports 1,2,3)	$V_{IN} = 0.45\text{V}$		-50	μA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1,2,3)	$V_{IN} = 2\text{V}$, $V_{CC} = 5\text{V} \pm 10\%$		-650	μA
I_{LI}	Input Leakage Current (Port 0, \overline{EA})	$0.45 < V_{IN} < V_{CC}$		± 10	μA
RRST	Reset Pull-down Resistor		50	300	$\text{K}\Omega$
C_{iO}	Pin Capacitance	Test Freq. = 1 MHz, $T_A = 25^\circ\text{C}$		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz		20	mA
		Idle Mode, 12 MHz		5	mA
	Power-down Mode ⁽²⁾	$V_{CC} = 6\text{V}$		100	μA
		$V_{CC} = 3\text{V}$		40	μA

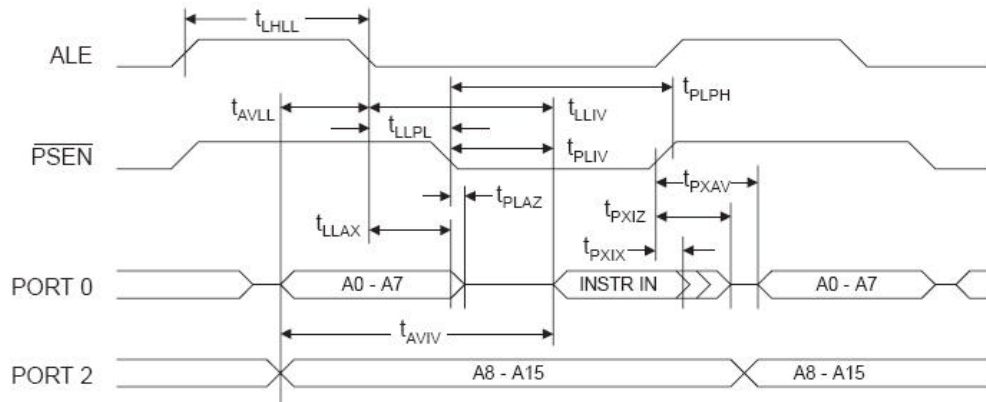
Notes: 1. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:

- Maximum I_{OL} per port pin: 10 mA
- Maximum I_{OL} per 8-bit port: Port 0: 26 mA
- Ports 1, 2, 3: 15 mA
- Maximum total I_{OL} for all output pins: 71 mA

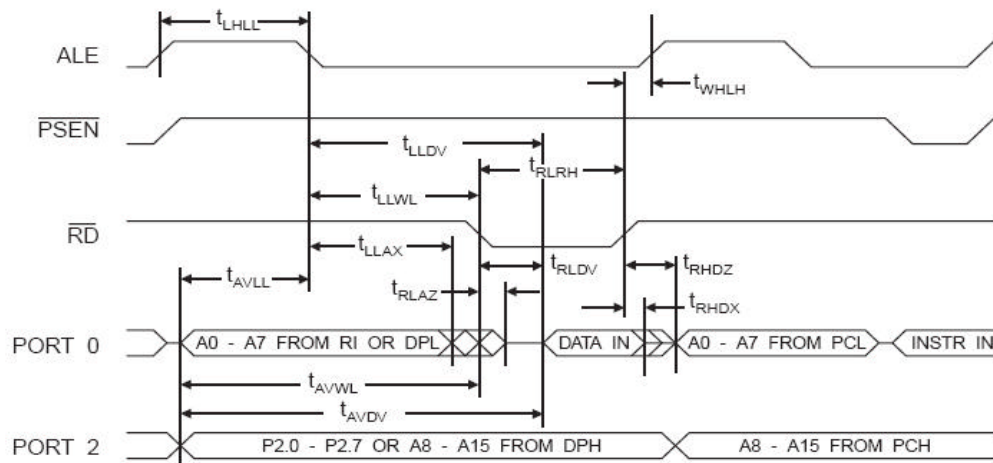
If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.

- 2. Minimum V_{CC} for Power-down is 2V.

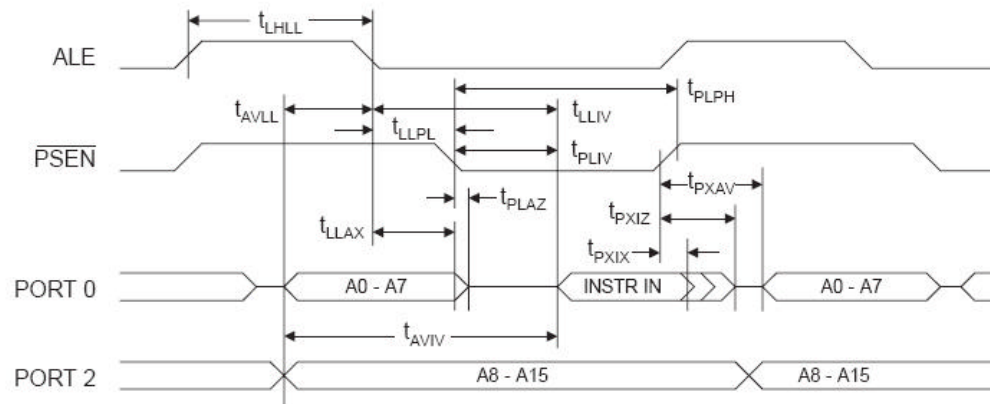
External Program Memory Read Cycle



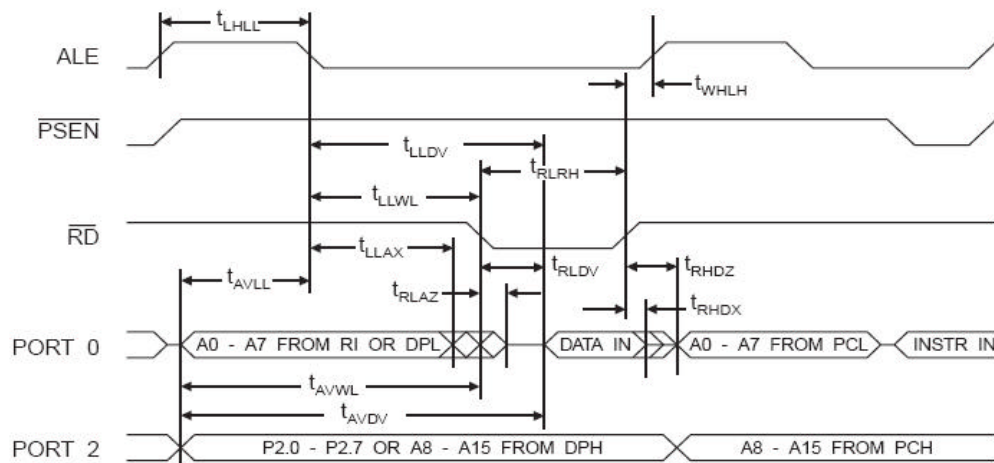
External Data Memory Read Cycle



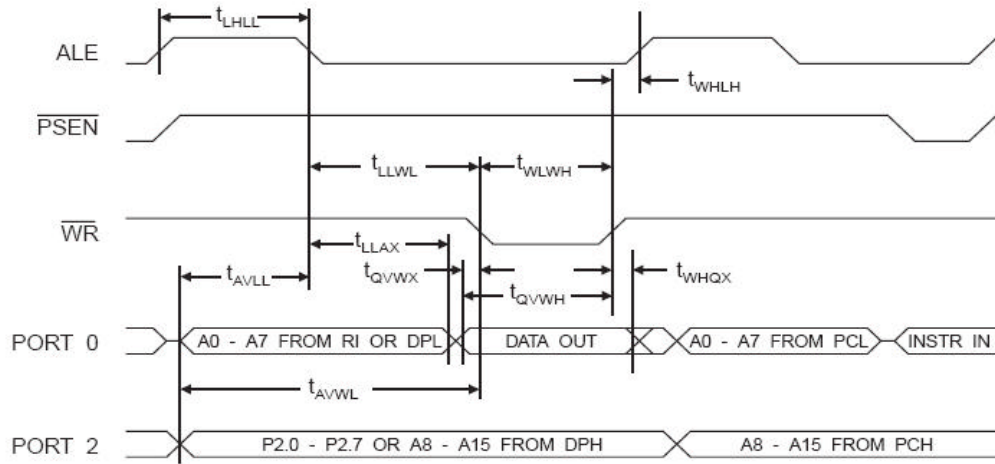
External Program Memory Read Cycle



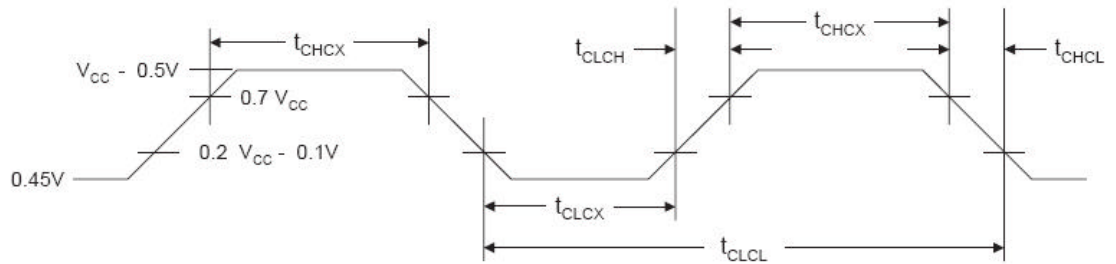
External Data Memory Read Cycle



External Data Memory Write Cycle



External Clock Drive Waveforms



External Clock Drive

Symbol	Parameter	Min	Max	Units
$1/t_{CLCL}$	Oscillator Frequency	0	24	MHz
t_{CLCL}	Clock Period	41.6		ns
t_{CHCX}	High Time	15		ns
t_{CLCX}	Low Time	15		ns
t_{CLCH}	Rise Time		20	ns
t_{CHCL}	Fall Time		20	ns

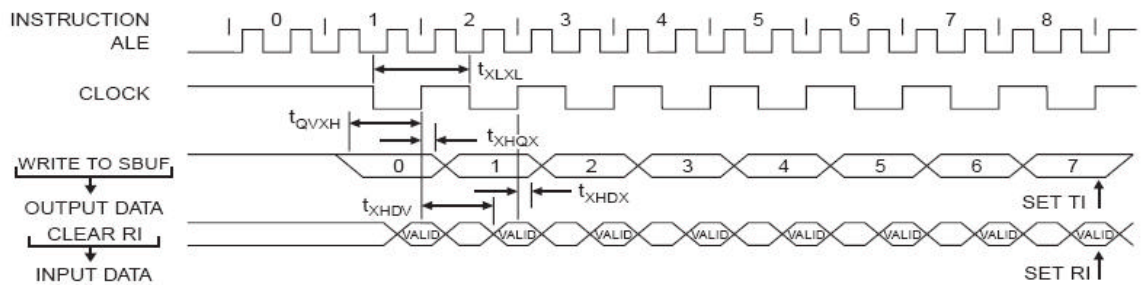


Serial Port Timing: Shift Register Mode Test Conditions

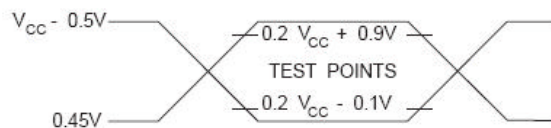
($V_{CC} = 5.0\text{ V} \pm 20\%$; Load Capacitance = 80 pF)

Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
t_{XLXL}	Serial Port Clock Cycle Time	1.0		$12t_{CLCL}$		μs
t_{QVXH}	Output Data Setup to Clock Rising Edge	700		$10t_{CLCL}-133$		ns
t_{XHGX}	Output Data Hold after Clock Rising Edge	50		$2t_{CLCL}-117$		ns
t_{XHDX}	Input Data Hold after Clock Rising Edge	0		0		ns
t_{XHDV}	Clock Rising Edge to Input Data Valid		700		$10t_{CLCL}-133$	ns

Shift Register Mode Timing Waveforms



AC Testing Input/Output Waveforms⁽¹⁾



Note: 1. AC Inputs during testing are driven at $V_{CC} - 0.5V$ for a logic 1 and $0.45V$ for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

Float Waveforms⁽¹⁾



Note: 1. For timing purposes, a port pin is no longer floating when a 100 mV change from load voltage occurs. A port pin begins to float when 100 mV change from the loaded V_{OH}/V_{OL} level occurs.

Ordering Information

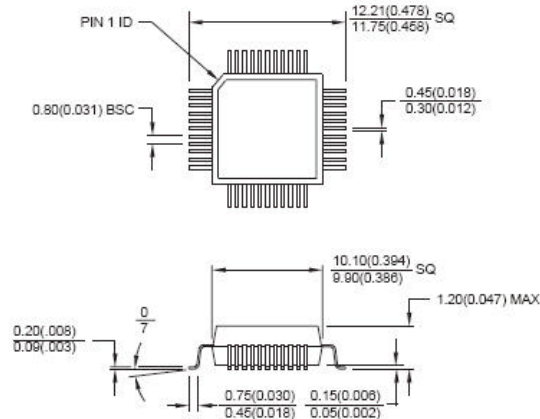
Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
12	5V ±20%	AT89C51-12AC	44A	Commercial (0° C to 70° C)
		AT89C51-12JC	44J	
		AT89C51-12PC	40P6	
		AT89C51-12QC	44Q	
		AT89C51-12AI	44A	Industrial (-40° C to 85° C)
		AT89C51-12JI	44J	
		AT89C51-12PI	40P6	
		AT89C51-12QI	44Q	
16	5V ±20%	AT89C51-16AC	44A	Commercial (0° C to 70° C)
		AT89C51-16JC	44J	
		AT89C51-16PC	40P6	
		AT89C51-16QC	44Q	
		AT89C51-16AI	44A	Industrial (-40° C to 85° C)
		AT89C51-16JI	44J	
		AT89C51-16PI	40P6	
		AT89C51-16QI	44Q	
20	5V ±20%	AT89C51-20AC	44A	Commercial (0° C to 70° C)
		AT89C51-20JC	44J	
		AT89C51-20PC	40P6	
		AT89C51-20QC	44Q	
		AT89C51-20AI	44A	Industrial (-40° C to 85° C)
		AT89C51-20JI	44J	
		AT89C51-20PI	40P6	
		AT89C51-20QI	44Q	
24	5V ±20%	AT89C51-24AC	44A	Commercial (0° C to 70° C)
		AT89C51-24JC	44J	
		AT89C51-24PC	40P6	
		AT89C51-24QC	44Q	
		AT89C51-24AI	44A	Industrial (-40° C to 85° C)
		AT89C51-24JI	44J	
		AT89C51-24PI	40P6	
		AT89C51-24QI	44Q	

Package Type	
44A	44-lead, Thin Plastic Gull Wing Quad Flatpack (TQFP)
44J	44-lead, Plastic J-leaded Chip Carrier (PLCC)
40P6	40-lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
44Q	44-lead, Plastic Gull Wing Quad Flatpack (PQFP)



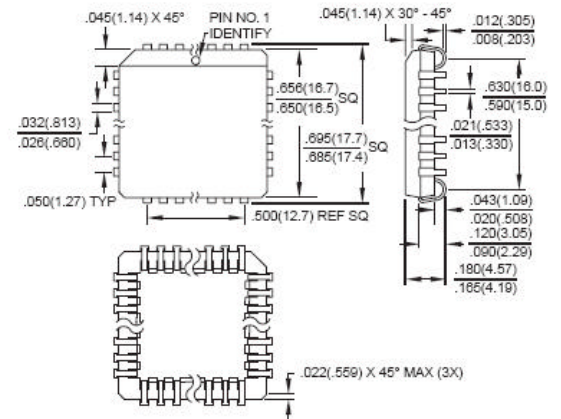
Packaging Information

44A, 44-lead, Thin (1.0 mm) Plastic Gull Wing Quad Flatpack (TQFP)
 Dimensions in Millimeters and (Inches)*
 JEDEC STANDARD MS-026 ACB

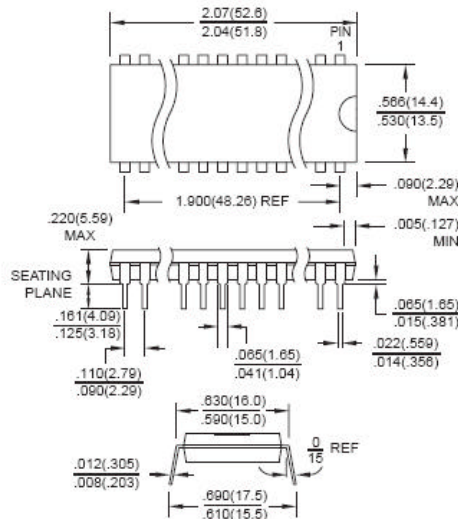


Controlling dimension: millimeters

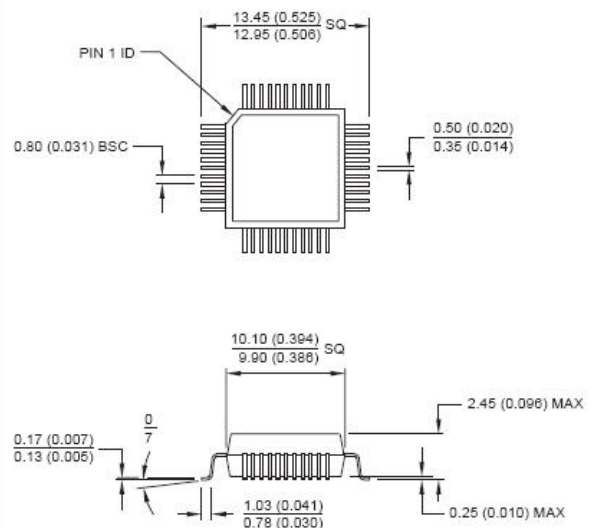
44J, 44-lead, Plastic J-leaded Chip Carrier (PLCC)
 Dimensions in Inches and (Millimeters)
 JEDEC STANDARD MS-018 AC



40P6, 40-lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
 Dimensions in Inches and (Millimeters)



44Q, 44-lead, Plastic Quad Flat Package (PQFP)
 Dimensions in Millimeters and (Inches)*
 JEDEC STANDARD MS-022 AB



Controlling dimension: millimeters



Atmel Headquarters

Corporate Headquarters

2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe

Atmel U.K., Ltd.
Coliseum Business Centre
Riverside Way
Camberley, Surrey GU15 3YL
England
TEL (44) 1276-686-677
FAX (44) 1276-686-697

Asia

Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan

Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs

1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset

Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Fax-on-Demand

North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail

literature@atmel.com

Web Site

<http://www.atmel.com>

BBS

1-(408) 436-4309

© Atmel Corporation 2000.

Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

Marks bearing ® and/or ™ are registered trademarks and trademarks of Atmel Corporation.

Terms and product names in this document may be trademarks of others.



Printed on recycled paper.

0265G-02/00/xM

LAMPIRAN E

DATA KOMPONEN PARALLEL PORT

Interfacing the Standard Parallel Port

Disclaimer : While every effort has been made to make sure the information in this document is correct, the author can not be liable for any damages whatsoever for loss relating to this document. Use this information at your own risk.

Table of Contents

Introduction to Parallel Ports	Page 1
Hardware Properties	Page 2
Centronics?	Page 4
Port Addresses	Page 4
Software Registers - Standard Parallel Port (SPP)	Page 6
Bi-directional Ports	Page 8
Using The Parallel Port to Input 8 Bits.	Page 9
Nibble Mode	Page 11
Using the Parallel Port's IRQ	Page 12
Parallel Port Modes in BIOS	Page 14
Parallel Port Modes and the ECP's Extended Control Register	Page 15

Introduction to Parallel Ports

The Parallel Port is the most commonly used port for interfacing home made projects. This port will allow the input of up to 9 bits or the output of 12 bits at any one given time, thus requiring minimal external circuitry to implement many simpler tasks. The port is composed of 4 control lines, 5 status lines and 8 data lines. It's found commonly on the back of your PC as a D-Type 25 Pin female connector. There may also be a D-Type 25 pin male connector. This will be a serial RS-232 port and thus, is a totally incompatible port.

Newer Parallel Port's are standardized under the IEEE 1284 standard first released in 1994. This standard defines 5 modes of operation which are as follows,

1. Compatibility Mode.
2. Nibble Mode. (*Protocol not Described in this Document*)
3. Byte Mode. (*Protocol not Described in this Document*)
4. EPP Mode (*Enhanced Parallel Port*).
5. ECP Mode (*Extended Capabilities Port*).

The aim was to design new drivers and devices which were compatible with each other and

also backwards compatible with the Standard Parallel Port (SPP). Compatibility, Nibble & Byte modes use just the standard hardware available on the original Parallel Port cards while EPP & ECP modes require additional hardware which can run at faster speeds, while still being downwards compatible with the Standard Parallel Port.

Compatibility mode or "Centronics Mode" as it is commonly known, can only send data in the forward direction at a typical speed of 50 kbytes per second but can be as high as 150+ kbytes a second. In order to receive data, you must change the mode to either Nibble or Byte mode. Nibble mode can input a nibble (*4 bits*) in the reverse direction. E.g. from device to computer. Byte mode uses the Parallel's bi-directional feature (*found only on some cards*) to input a byte (*8 bits*) of data in the reverse direction.

Extended and Enhanced Parallel Ports use additional hardware to generate and manage handshaking. To output a byte to a printer (or anything in that matter) using compatibility mode, the software must.

1. *Write the byte to the Data Port.*
2. *Check to see if the printer is busy. If the printer is busy, it will not accept any data, thus any data which is written will be lost.*
3. *Take the Strobe (Pin 1) low. This tells the printer that there is the correct data on the data lines. (Pins 2-9)*
4. *Put the strobe high again after waiting approximately 5 microseconds after putting the strobe low. (Step 3)*

This limits the speed at which the port can run at. The EPP & ECP ports get around this by letting the hardware check to see if the printer is busy and generate a strobe and /or appropriate handshaking. This means only one I/O instruction need to be performed, thus increasing the speed. These ports can output at around 1-2 megabytes per second. The ECP port also has the advantage of using DMA channels and FIFO buffers, thus data can be shifted around without using I/O instructions.

Hardware Properties

On the next page is a table of the "Pin Outs" of the D-Type 25 Pin connector and the Centronics 34 Pin connector. The D-Type 25 pin connector is the most common connector found on the Parallel Port of the computer, while the Centronics Connector is commonly found on printers. The IEEE 1284 standard however specifies 3 different connectors for use with the Parallel Port. The first one, 1284 Type A is the D-Type 25 connector found on the back of most computers. The 2nd is the 1284 Type B which is the 36 pin Centronics Connector found on most printers.

IEEE 1284 Type C however, is a 36 conductor connector like the Centronics, but smaller. This connector is claimed to have a better clip latch, better electrical properties and is easier to assemble. It also contains two more pins for signals which can be used to see whether the other device connected,

has power. 1284 Type C connectors are recommended for new designs, so we can look forward on seeing these new connectors in the near future.

Pin No (D-Type 25)	Pin No (Centronics)	SPP Signal	Direction In/out	Register	Hardware Inverted
1	1	nStrobe	In/Out	Control	Yes
2	2	Data 0	Out	Data	
3	3	Data 1	Out	Data	
4	4	Data 2	Out	Data	
5	5	Data 3	Out	Data	
6	6	Data 4	Out	Data	
7	7	Data 5	Out	Data	
8	8	Data 6	Out	Data	
9	9	Data 7	Out	Data	
10	10	nAck	In	Status	
11	11	Busy	In	Status	Yes
12	12	Paper-Out PaperEnd	In	Status	
13	13	Select	In	Status	
14	14	nAuto-Linefeed	In/Out	Control	Yes
15	32	nError / nFault	In	Status	
16	31	nInitialize	In/Out	Control	
17	36	nSelect-Printer nSelect-In	In/Out	Control	Yes
18 - 25	19-30	Ground	Gnd		

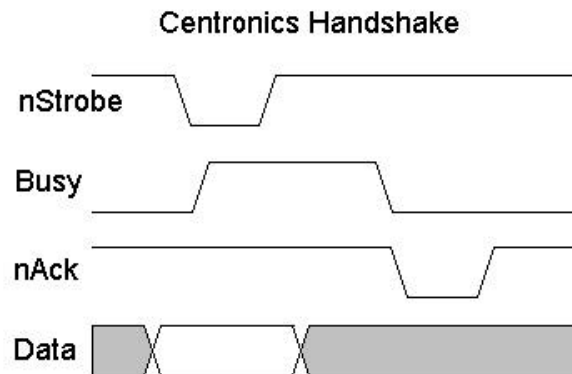
Table 1. Pin Assignments of the D-Type 25 pin Parallel Port Connector.

The above table uses "n" in front of the signal name to denote that the signal is active low. e.g. nError. If the printer has occurred an error then this line is low. This line normally is high, should the printer be functioning correctly. The "Hardware Inverted" means the signal is inverted by the Parallel card's hardware. Such an example is the Busy line. If +5v (Logic 1) was applied to this pin and the status register read, it would return back a 0 in Bit 7 of the Status Register.

The output of the Parallel Port is normally TTL logic levels. The voltage levels are the easy part. The current you can sink and source varies from port to port. Most Parallel Ports implemented in ASIC, can sink and source around 12mA. However these are just some of the figures taken from Data sheets, Sink/Source 6mA, Source 12mA/Sink 20mA, Sink 16mA/Source 4mA, Sink/Source 12mA. As you can see they vary quite a bit. The best bet is to use a buffer, so the least current is drawn from the Parallel Port.

Centronics?

Centronics is an early standard for transferring data from a host to the printer. The majority of printers use this handshake. This handshake is normally implemented using a Standard Parallel Port under software control. Below is a simplified diagram of the 'Centronics' Protocol.



Data is first applied on the Parallel Port pins 2 to 7. The host then checks to see if the printer is busy. i.e. the busy line should be low. The program then asserts the strobe, waits a minimum of $1\mu\text{S}$, and then de-asserts the strobe. Data is normally read by the printer/peripheral on the rising edge of the strobe. The printer will indicate that it is busy processing data via the Busy line. Once the printer has accepted data, it will acknowledge the byte by a negative pulse about $5\mu\text{S}$ on the nAck line.

Quite often the host will ignore the nAck line to save time. Later in the Extended Capabilities Port, you will see a Fast Centronics Mode, which lets the hardware do all the handshaking for you. All the programmer must do is write the byte of data to the I/O port. The hardware will check to see if the printer is busy, generate the strobe. Note that this mode commonly doesn't check the nAck either.

Port Addresses

The Parallel Port has three commonly used base addresses. These are listed in table 2, below. The 3BCh base address was originally introduced used for Parallel Ports on early Video Cards. This address then disappeared for a while, when Parallel Ports were later removed from Video Cards. They has now reappeared as an option for Parallel Ports integrated onto motherboards, upon which their configuration can be changed using BIOS.

LPT1 is normally assigned base address 378h, while LPT2 is assigned 278h. However this may not always be the case as explained later. 378h & 278h have always been commonly used for Parallel Ports. The lower case h denotes that it is in hexadecimal. These addresses may change from machine to machine.

Address	Notes:
3BCh - 3BFh	Used for Parallel Ports which were incorporated in to Video Cards and now, commonly an option for Ports controlled by BIOS. - Doesn't support ECP addresses.
378h - 37Fh	Usual Address For LPT 1
278h - 27Fh	Usual Address For LPT 2

Table 2 Port Addresses

When the computer is first turned on, BIOS (Basic Input/Output System) will determine the number of ports you have and assign device labels LPT1, LPT2 & LPT3 to them. BIOS first looks at address 3BCh. If a Parallel Port is found here, it is assigned as LPT1, then it searches at location 378h. If a Parallel card is found there, it is assigned the next free device label. This would be LPT1 if a card wasn't found at 3BCh or LPT2 if a card was found at 3BCh. The last *port of call*, is 278h and follows the same procedure than the other two ports. Therefore it is possible to have a LPT2 which is at 378h and not at the expected address 278h.

What can make this even confusing, is that some manufacturers of Parallel Port Cards, have jumpers which allow you to set your Port to LPT1, LPT2, LPT3. Now what address is LPT1? - On the majority of cards LPT1 is 378h, and LPT2, 278h, but some will use 3BCh as LPT1, 378h as LPT1 and 278h as LPT2. *Life wasn't meant to be easy.*

The assigned devices LPT1, LPT2 & LPT3 should not be a worry to people wishing to interface devices to their PC's. Most of the time the base address is used to interface the port rather than LPT1 etc. However should you want to find the address of LPT1 or any of the **Line PrinTer** Devices, you can use a lookup table provided by BIOS. When BIOS assigns addresses to your printer devices, it stores the address at specific locations in memory, so we can find them.

Start Address	Function
0000:0408	LPT1's Base Address
0000:040A	LPT2's Base Address
0000:040C	LPT3's Base Address
0000:040E	LPT4's Base Address (Note 1)

Table 3 - LPT Addresses in the BIOS Data Area

Note 1 : Address 0000:040E in the BIOS Data Area may be used as the Extended Bios Data Area in PS/2 and newer Bioses, and thus this field may be invalid.

The above table, table 3, shows the address at which we can find the Printer Port's addresses in the BIOS Data Area. Each address will take up 2 bytes. The following sample program in C, shows how you can read these locations to obtain the addresses of your printer ports.

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
    unsigned int far *ptraddr; /* Pointer to location of Port Addresses */
    unsigned int address;     /* Address of Port */
    int a;

    ptraddr=(unsigned int far *)0x00000408;

    for (a = 0; a < 3; a++)
    {
        address = *ptraddr;
        if (address == 0)
            printf("No port found for LPT%d \n",a+1);
        else
            printf("Address assigned to LPT%d is %Xh\n",a+1,address);
        *ptraddr++;
    }
}
```

Software Registers - Standard Parallel Port (SPP)

Offset	Name	Read/Write	Bit No.	Properties
Base + 0	Data Port	Write (Note-1)	Bit 7	Data 7 (Pin 9)
			Bit 6	Data 6 (Pin 8)
			Bit 5	Data 5 (Pin 7)
			Bit 4	Data 4 (Pin 6)
			Bit 3	Data 3 (Pin 5)
			Bit 2	Data 2 (Pin 4)
			Bit 1	Data 1 (Pin 3)
			Bit 0	Data 0 (Pin 2)

Table 4 Data Port

Note 1 : If the Port is bi-directional then Read and Write Operations can be performed on the Data Register.

The base address, usually called the Data Port or Data Register is simply used for outputting data on the Parallel Port's data lines (Pins 2-9). This register is normally a write only port. If you read from the port, you should get the last byte sent. However if your port is bi-directional, you can receive data on this address. See *Bi-directional Ports* for more detail.

Base + 1	Status Port	Read Only	Bit 7	Busy
			Bit 6	Ack
			Bit 5	Paper Out
			Bit 4	Select In
			Bit 3	Error
			Bit 2	IRQ (Not)
			Bit 1	Reserved
			Bit 0	Reserved

Table 5 Status Port

The Status Port (base address + 1) is a read only port. Any data written to this port will be ignored. The Status Port is made up of 5 input lines (Pins 10,11,12,13 & 15), a IRQ status register and two reserved bits. Please note that Bit 7 (Busy) is a active low input. E.g. If bit 7 happens to show a logic 0, this means that there is +5v at pin 11. Likewise with Bit 2. (nIRQ) If this bit shows a '1' then an interrupt has **not** occurred.

Base + 2	Control Port	Read/Write	Bit 7	Unused
			Bit 6	Unused
			Bit 5	Enable bi-directional Port
			Bit 4	Enable IRQ Via Ack Line
			Bit 3	Select Printer
			Bit 2	Initialize Printer (Reset)
			Bit 1	Auto Linefeed
			Bit 0	Strobe

Table 6 Control Port

The Control Port (base address + 2) was intended as a write only port. When a printer is attached to the Parallel Port, four "controls" are used. These are Strobe, Auto Linefeed, Initialize and Select Printer, all of which are inverted except Initialize.

The printer would not send a signal to initialize the computer, nor would it tell the computer to use auto linefeed. However these four outputs can also be used for inputs. If the computer has placed a pin high (e.g. +5v) and your device wanted to take it low, you would effectively short out the port, causing a conflict on that pin. Therefore these lines are "open collector" outputs (*or open drain for CMOS devices*). This means that it has two states. A low state (0v) and a high impedance state (open circuit).

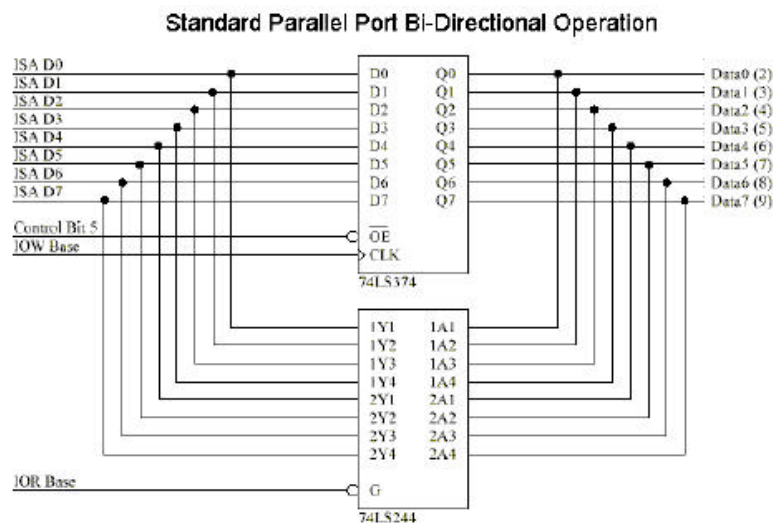
Normally the Printer Card will have internal pull-up resistors, but as you would expect, not all will. Some may just have open collector outputs, while others may even have normal totem pole outputs. In order to make your device work correctly on as many Printer Ports as possible, you can use an external resistor as well. Should you already have an internal resistor, then it will act in Parallel with it, or if you have Totem pole outputs, the resistor will act as a load.

An external 4.7k resistor can be used to pull the pin high. I wouldn't use anything lower, just in case you do have an internal pull up resistor, as the external resistor would act in parallel giving effectively, a lower value pull up resistor. When in high impedance state the pin on the Parallel Port is high (+5v). When in this state, your external device can pull the pin low and have the control port change read a different value. This way the 4 pins of the Control Port can be used for bi-directional data transfer. However the Control Port must be set to xxxx0100 to be able to read data, that is all pins to be +5v at the port so that you can pull it down to GND (logic 0).

Bits 4 & 5 are internal controls. Bit four will enable the IRQ (*See Using the Parallel Ports IRQ*) and Bit 5 will enable the bi-directional port meaning that you can input 8 bits using (DATA0-7). This mode is only possible if your card supports it. Bits 6 & 7 are reserved. Any writes to these two bits will be ignored.

Bi-directional Ports

The schematic diagram below, shows a simplified view of the Parallel Port's Data Register. The original Parallel Port card's implemented 74LS logic. These days all this is crammed into one ASIC, but the theory of operation is still the same.



The non bi-directional ports were manufactured with the 74LS374's output enable tied permanent low, thus the data port is always output only. When you read the Parallel Port's data register, the data comes from the 74LS374 which is also connected to the data pins. Now if you can overdrive the '374 you can effectively have a Bi-directional Port. (*or a input only port, once you blow up the latches output!*)

What is very concerning is that people have actually done this. I've seen one circuit, a scope connected to the Parallel Port distributed on the Internet. The author uses an ADC of some type, but finds the ADC requires transistors on each data line, to make it work! No wonder why. Others have had similar trouble, the 68HC11 cannot sink enough current (30 to 40mA!)

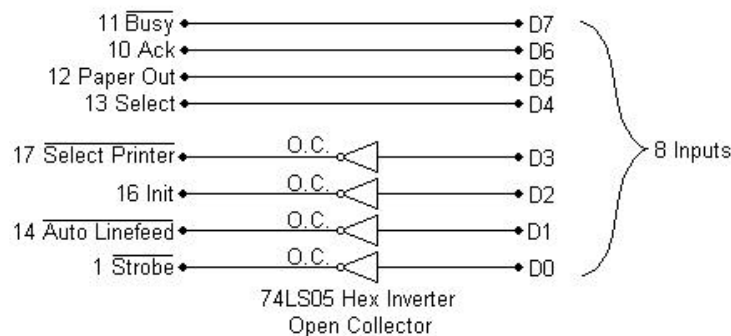
Bi-directional ports use Control Bit 5 connected to the 374's OE so that it's output drivers can be turned off. This way you can read data present on the Parallel Port's Data Pins, without having bus conflicts and excessive current drains.

Bit 5 of the Control Port enables or disables the bi-directional function of the Parallel Port. This is only available on true bi-directional ports. When this bit is set to one, pins 2 to 9 go into high impedance state. Once in this state you can enter data on these lines and retrieve it from the Data Port (base address). Any data which is written to the data port will be stored but will not be available at the data pins. To turn off bi-directional mode, set bit 5 of the Control Port to '0'.

However not all ports behave in the same way. Other ports may require setting bit 6 of the Control Port to enable Bi-directional mode and setting of Bit 5 to dis-enable Bi-directional mode. Different manufacturers implement their bi-directional ports in different ways. If you wish to use your Bi-directional port to input data, test it with a logic probe or multimeter first to make sure it is in bi-directional mode.

Using The Parallel Port to Input 8 Bits.

If your Parallel Port doesn't support bi-directional mode, don't despair. You can input a maximum of 9 bits at any one given time. To do this you can use the 5 input lines of the Status Port and the 4 inputs (open collector) lines of the Control Port.



The inputs to the Parallel Port has been chosen as such, to make life easier for us. Busy just happens to be the MSB (Bit 7) of the Status Port, then in ascending order comes Ack, Paper Out and Select, making up the most significant nibble of the Control Port. The Bars are used to represent which inputs are Hardware inverted, i.e. +5v will read 0 from the register, while GND will read 1. The Status Port only has one inverted input.

The Control port is used to read the least significant nibble. As described before, the control port has open collector outputs, i.e. two possible states, high impedance and GND. If we connect our inputs directly to the port (*For example an ADC0804 with totem pole outputs*), a conflict will result if the input is high and the port is trying to pull it down. Therefore we use open collector inverters.

However this is not always entirely necessary. If we were connecting single pole switches to the port with a pull up resistor, then there is no need to bother with this protection. Also if your software initializes the control port with xxxx0100 so that all the pins on the control port are high,

then it may be unnecessary. If however you don't bother and your device is connected to the Parallel Port before your software has a chance to initialize then you may encounter problems.

Another problem to be aware of is the pull up resistors on the control port. The average pull-up resistor is 4.7k. In order to pull the line low, your device will need to sink 1mA, which some low powered devices may struggle to do. Now what happens if I suggest that some ports have 1K pull up resistors? Yes, there are such cards. Your device now has to sink 5mA. More reason to use the open collector inverters.

Open collector inverters were chosen over open collector buffers as they are more popular, and thus easier to obtain. There is no reason, however why you can't use them. Another possibility is to use transistors.

The input, D3 is connected via the inverter to Select Printer. Select Printer just happens to be bit 3 of the control port. D2, D1 & D0 are connected to Init, Auto linefeed and strobe, respectively to make up the lower nibble. Now this is done, all we have to do is assemble the byte using software. The first thing we must do is to write xxxx0100 to the Control Port. This places all the control port lines high, so they can be pulled down to input data.

```
outportb(CONTROL, inportb(CONTROL) & 0xF0 | 0x04);
```

Now that this is done, we can read the most significant nibble. This just happens to be the most significant nibble of the status port. As we are only interested in the MSnibble we will AND the results with 0xF0, so that the LSnibble is clear. Busy is hardware inverted, but we won't worry about it now. Once the two bytes are constructed, we can kill two birds with one stone by toggling Busy and Init at the same time.

```
a = (inportb(STATUS) & 0xF0); /* Read MSnibble */
```

We can now read the LSnibble. This just happens to be LSnibble of the control port - How convenient! This time we are not interested with the MSnibble of the port, thus we AND the result with 0x0F to clear the MSnibble. Once this is done, it is time to combine the two bytes together. This is done by OR'ing the two bytes. This now leaves us with one byte, however we are not finished yet. Bits 2 and 7 are inverted. This is overcome by XOR'ing the byte with 0x84, which toggles the two bits.

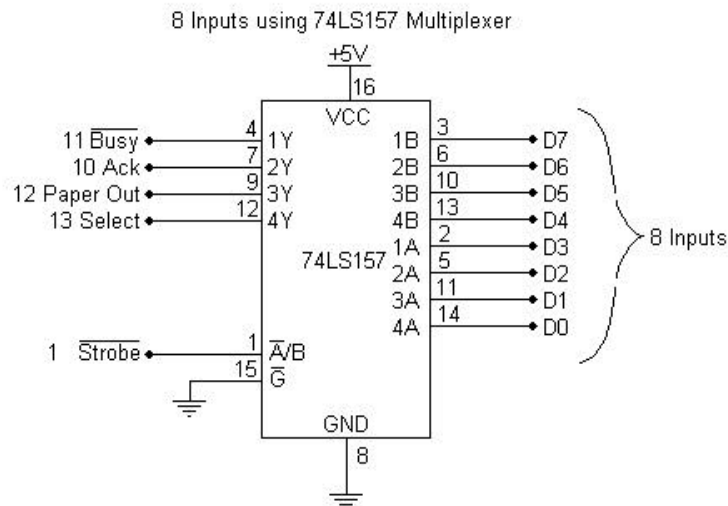
```
a = a | (inportb(CONTROL) & 0x0F); /* Read LSnibble */
```

```
a = a ^ 0x84; /* Toggle Bit 2 & 7 */
```

Note: Some control ports are not open collector, but have totem pole outputs. This is also the case with EPP and ECP Ports. Normally when you place a Parallel Port in ECP or EPP mode, the control port becomes totem pole outputs only. Now what happens if you connect your device to the Parallel Port in this mode? Therefore, in the interest of portability I recommend using the next circuit, reading a nibble at a time.

Nibble Mode

Nibble mode is the preferred way of reading 8 bits of data without placing the port in reverse mode and using the data lines. Nibble mode uses a Quad 2 line to 1 line multiplexer to read a nibble of data at a time. Then it "switches" to the other nibble and reads its. Software can then be used to construct the two nibbles into a byte. The only disadvantage of this technique is that it is slower. It now requires a few I/O instructions to read the one byte, and it requires the use of an external IC.



The operation of the 74LS157, Quad 2 line to 1 line multiplexer is quite simple. It simply acts as four switches. When the A/B input is low, the A inputs are selected. E.g. 1A passes through to 1Y, 2A passes through to 2Y etc. When the A/B is high, the B inputs are selected. The Y outputs are connected up to the Parallel Port's status port, in such a manner that it represents the MSnibble of the status register. While this is not necessary, it makes the software easier.

To use this circuit, first we must initialize the multiplexer to switch either inputs A or B. We will read the LSnibble first, thus we must place A/B low. The strobe is hardware inverted, thus we must set Bit 0 of the control port to get a low on Pin 1.

```
outportb(CONTROL, inportb(CONTROL) | 0x01); /* Select Low Nibble (A)*/
```

Once the low nibble is selected, we can read the LSnibble from the Status Port. Take note that the Busy Line is inverted, however we won't tackle it just yet. We are only interested in the MSnibble of the result, thus we AND the result with 0xF0, to clear the LSnibble.

```
a = (inportb(STATUS) & 0xF0); /* Read Low Nibble */
```

Now it's time to shift the nibble we have just read to the LSnibble of variable a,

```
a = a >> 4; /* Shift Right 4 Bits */
```

We are now half way there. It's time to get the MSnibble, thus we must switch the multiplexer to select inputs B. Then we can read the MSnibble and put the two nibbles together to make a byte,


```

outportb(CONTROL, inportb(CONTROL) & 0xFE); /* Select High Nibble (B)*/
a = a |(inportb(STATUS) & 0xF0); /* Read High Nibble */
byte = byte ^ 0x88;

```

The last line toggles two inverted bits which were read in on the Busy line. It may be necessary to add delays in the process, if the incorrect results are being returned.

Using the Parallel Port's IRQ

The Parallel Port's interrupt request is not used for printing under DOS or Windows. Early versions of OS-2 used them, but don't anymore. Interrupts are good when interfacing monitoring devices such as high temp alarms etc, where you don't know when it is going to be activated. It's more efficient to have an interrupt request rather than have the software poll the ports regularly to see if something has changed. This is even more noticeable if you are using your computer for other tasks, such as with a multitasking operating system.

The Parallel Port's interrupt request is normally IRQ5 or IRQ7 but may be something else if these are in use. It may also be possible that the interrupts are totally disabled on the card, if the card was only used for printing. The Parallel Port interrupt can be disabled and enabled using bit 4 of the control register, *Enable IRQ Via Ack Line*. Once enabled, an interrupt will occur upon a low to high transition (rising edge) of the nACK. However like always, some cards may trigger the interrupt on the high to low transition.

The following code is an Interrupt Polarity Tester, which serves as two things. It will determine which polarity your Parallel Port interrupt is, while also giving you an example for how to use the Parallel Port's Interrupt. It checks if your interrupt is generated on the rising or falling edge of the nACK line. To use the program simply wire **one** of the Data lines (Pins 2 to 9) to the Ack Pin (Pin 10). The easiest way to do this is to bridge some solder from DATA7 (Pin 9) to ACK (Pin 10) on a male DB25 connector.

```

/* Parallel Port Interrupt Polarity Tester                               */
/* 2nd February 1998                                                    */
/* Copyright 1997 Craig Peacock                                         */
/* WWW      - http://www.senet.com.au/~cpeacock                         */
/* Email    - cpeacock@senet.com.au                                     */
#include <dos.h>

#define PORTADDRESS 0x378 /* Enter Your Port Address Here */
#define IRQ 7 /* IRQ Here */

#define DATA PORTADDRESS+0
#define STATUS PORTADDRESS+1
#define CONTROL PORTADDRESS+2

#define PIC1 0x20
#define PIC2 0xA0

int interflag; /* Interrupt Flag */
int picaddr; /* Programmable Interrupt Controller (PIC) Base Address */

```

```

void interrupt (*oldhandler)();

void interrupt parisr() /* Interrupt Service Routine (ISR) */
{
    interflag = 1;
    outportb(picaddr,0x20); /* End of Interrupt (EOI) */
}

void main(void)
{
    int c;
    int intno; /* Interrupt Vector Number */
    int picmask; /* PIC's Mask */

    /* Calculate Interrupt Vector, PIC Addr & Mask. */

    if (IRQ >= 2 && IRQ <= 7) {
        intno = IRQ + 0x08;
        picaddr = PIC1;
        picmask = 1;
        picmask = picmask << IRQ;
    }
    if (IRQ >= 8 && IRQ <= 15) {
        intno = IRQ + 0x68;
        picaddr = PIC2;
        picmask = 1;
        picmask = picmask << (IRQ-8);
    }

    if (IRQ < 2 || IRQ > 15)
    {
        printf("IRQ Out of Range\n");
        exit();
    }

    outportb(CONTROL, inportb(CONTROL) & 0xDF); /* Make sure port is in Forward Direction */
    outportb(DATA,0xFF);
    oldhandler = getvect(intno); /* Save Old Interrupt Vector */
    setvect(intno, parisr); /* Set New Interrupt Vector Entry */
    outportb(picaddr+1,inportb(picaddr+1) & (0xFF - picmask)); /* Un-Mask Pic */
    outportb(CONTROL, inportb(CONTROL) | 0x10); /* Enable Parallel Port IRQ's */

    clrscr();
    printf("Parallel Port Interrupt Polarity Tester\n");
    printf("IRQ %d : INTNO %02X : PIC Addr 0x%X : Mask 0x%02X\n",IRQ,intno,picaddr,picmask);
    interflag = 0; /* Reset Interrupt Flag */
    delay(10);
    outportb(DATA,0x00); /* High to Low Transition */
    delay(10); /* Wait */
    if (interflag == 1) printf("Interrupts Occur on High to Low Transition of ACK.\n");
    else
    {
        outportb(DATA,0xFF); /* Low to High Transition */
        delay(10); /* wait */
        if (interflag == 1) printf("Interrupts Occur on Low to High Transition of ACK.\n");
        else printf("No Interrupt Activity Occurred. \nCheck IRQ Number, Port Address "
            "and Wiring.");
    }

    outportb(CONTROL, inportb(CONTROL) & 0xEF); /* Disable Parallel Port IRQ's */
    outportb(picaddr+1,inportb(picaddr+1) | picmask); /* Mask Pic */
    setvect(intno, oldhandler); /* Restore old Interrupt Vector Before Exit */
}

```


At compile time, the above source may generate a few warnings, *condition always true*, *condition always false*, *unreachable code* etc. These are perfectly O.K. They are generated as some of the condition structures test which IRQ you are using, and as the IRQ is defined as a constant some outcomes will never change. While they would of been better implemented as a preprocessor directive, I've done this so you can cut and paste the source code in your own programs which may use command line arguments, user input etc instead of a defined IRQ.

To understand how this example works, the reader must have an assumed knowledge and understanding of Interrupts and Interrupt Service Routines (ISR). If not, see *Using Interrupts*¹ for a quick introduction.

The first part of the mainline routine calculates the Interrupt Vector, PIC Addr & Mask in order to use the Parallel Port's Interrupt Facility. After the Interrupt Service Routine (ISR) has been set up and the Programmable Interrupt Controller (PIC) set, we must enable the interrupt on the Parallel Port. This is done by setting bit 4 of the Parallel Port's Control Register using `outportb(CONTROL, inportb(CONTROL) | 0x10);`

Before enabling the interrupts, we wrote 0xFF to the Parallel Port to enable the 8 data lines into a known state. At this point of the program, all the data lines should be high. The interrupt service routine simply sets a flag (*interflag*), thus we can determine when an IRQ occurs. We are now in a position to write 0x00 to the data port, which causes a high to low transition on the Parallel Port's Acknowledge line as it's connected to one of the data lines.

If the interrupt occurs on the high to low transition, the interrupt flag (*interflag*) should be set. We now test this, and if this is so the program informs the user. However if it is not set, then an interrupt has not yet occurred. We now write 0xFF to the data port, which will cause a low to high transition on the nAck line and check the interrupt flag again. If set, then the interrupt occurs on the low to high transition.

However if the interrupt flag is still reset, then this would suggest that the interrupts are not working. Make sure your IRQ and Base Address is correct and also check the wiring of the plug.

Parallel Port Modes in BIOS

Today, most Parallel Ports are mulimode ports. They are normally software configurable to one of many modes from BIOS. The typical modes are,

Printer Mode (Sometimes called Default or Normal Modes))

Standard & Bi-directional (SPP) Mode

EPP1.7 and SPP Mode

EPP1.9 and SPP Mode

ECP Mode

ECP and EPP1.7 Mode

ECP and EPP1.9 Mode

Printer Mode is the most basic mode. It is a Standard Parallel Port in forward mode only. It has no bi-directional feature, thus Bit 5 of the Control Port will not respond. *Standard & Bi-directional (SPP) Mode* is the bi-directional mode. Using this mode, bit 5 of the Control Port will reverse the direction of the port, so you can read back a value on the data lines.

EPP1.7 and SPP Mode is a combination of EPP 1.7 (Enhanced Parallel Port) and SPP Modes. In this mode of operation you will have access to the SPP registers (Data, Status and Control) and access to the EPP Registers. In this mode you should be able to reverse the direction of the port using bit 5 of the control register. EPP 1.7 is the earlier version of EPP. This version, version 1.7, may not have the time-out bit. See *Interfacing the Enhanced Parallel Port*² for more information.

EPP1.9 and SPP Mode is just like the previous mode, only it uses EPP Version 1.9 this time. As in the other mode, you will have access to the SPP registers, including Bit 5 of the control port. However this differs from EPP1.7 and SPP Mode as you should have access to the EPP Timeout bit.

ECP Mode will give you an Extended Capabilities Port. The mode of this port can then be set using the ECP's Extended Control Register (ECR). However in this mode from BIOS the EPP Mode (100) will not be available. We will further discuss the ECP's Extended Control Register in this document, but if you want further information on the ECP port, consult *Interfacing the Extended Capabilities Port*³.

ECP and EPP1.7 Mode & ECP and EPP1.9 Mode will give you an Extended Capabilities Port, just like the previous mode. However the EPP Mode in the ECP's ECR will now be available. Should you be in *ECP and EPP1.7 Mode* you will get an EPP1.7 Port, or if you are in *ECP and EPP1.9 Mode*, an EPP1.9 Port will be at your disposal.

The above modes are configurable via BIOS. You can reconfigure them by using your own software, but this is **not recommended**. These software registers, typically found at 0x2FA, 0x3F0, 0x3F1 etc are only intended to be accessed by BIOS. There is no set standard for these configuration registers, thus if you were to use these registers, your software would not be very portable. With today's multitasking operating systems, its also not a good idea to change them when it suits you.

A better option is to select *ECP and EPP1.7 Mode* or *ECP and EPP1.9 Mode* from BIOS and then use the ECP's Extended Control Register to select your Parallel Port's Mode. The EPP1.7 mode had a few problems in regards to the Data and Address Strokes being asserted to start a cycle regardless of the wait state, thus this mode if not typically used now. Best set your Parallel Port to *ECP and EPP1.9 Mode*.

Parallel Port Modes and the ECP's Extended Control Register

As we have just discussed, it is better to set the Parallel Port to *ECP and EPP1.9 Mode* and use the ECP's Extended Control Register to select different modes of operation. The ECP Registers are standardized under Microsoft's **Extended Capabilities Port Protocol and ISA Interface Standard**, thus we don't have that problem of every vendor having their own register set.

When set to ECP Mode, a new set of registers become available at Base + 0x400h. A discussion of these registers are available in *Interfacing the Extended Capabilities Port*³. Here we are only interested in the Extended Control Register (ECR) which is mapped at Base + 0x402h. It should be stated that the ECP's registers are not available for port's with a base address of 0x3BCh.

Bit	Function
7:5	Selects Current Mode of Operation
000	Standard Mode
001	Byte Mode
010	Parallel Port FIFO Mode
011	ECP FIFO Mode
100	EPP Mode
101	Reserved
110	FIFO Test Mode
111	Configuration Mode
4	ECP Interrupt Bit
3	DMA Enable Bit
2	ECP Service Bit
1	FIFO Full
0	FIFO Empty

Table 7 ECR - Extended Control Register

The table above is of the Extended Control Register. We are only interested in the three MSB of the Extended Control Register which selects the mode of operation. There are 7 possible modes of operation, but not all ports will support all modes. The EPP mode is one such example, not being available on some ports.

Modes of Operation

Standard mode	Selecting this mode will cause the ECP port to behave as a Standard Parallel Port, without bi-directional functionality.
Byte Mode / PS/2 mode	Behaves as a SPP in bi-directional mode. Bit 5 will place the port in reverse mode.
Parallel Port FIFO mode	In this mode, any data written to the Data FIFO will be sent to the peripheral using the SPP Handshake. The hardware will generate the handshaking required. Useful with non-ECP devices such as printers. You can have some of the features of ECP like FIFO buffers and hardware generation of handshaking but with the existing SPP handshake (Centronics) instead of the ECP Handshake.
ECP FIFO mode	Standard mode for ECP use. This mode uses the ECP Handshake described in <i>Interfacing the Extended Capabilities Port</i> ³ <i>When in ECP Mode through BIOS, and the ECR register is set to ECP FIFO Mode (011), the SPP registers may disappear.</i>
EPP mode/Reserved	This will enable EPP Mode, if available. Under BIOS, if <i>ECP mode</i> is set then it's more than likely, this mode is not an option. However if BIOS is set to <i>ECP and EPP1.x Mode</i> , then EPP 1.x will be enabled. <i>Under Microsoft's Extended Capabilities Port Protocol and ISA Interface Standard this mode is Vendor Specified.</i>

Reserved	Currently Reserved. <i>Under Microsoft's Extended Capabilities Port Protocol and ISA Interface Standard this mode is Vendor Specified.</i>
FIFO Test Mode	While in this mode, any data written to the Test FIFO Register will be placed into the FIFO and any data read from the Test FIFO register will be read from the FIFO buffer. The FIFO Full/Empty Status Bits will reflect their true value, thus FIFO depth, among other things can be determined in this mode.
Configuration Mode	In this mode, the two configuration registers, cnfgA & cnfgB become available at their designated Register Addresses.

If you are in *ECP Mode* under BIOS, or if your card is jumpered to use ECP then it is a good idea to initialize the mode of your ECP port to a pre-defined state before use. If you are using SPP, then set the port to Standard Mode as the first thing you do. Don't assume that the port will already be in Standard (SPP) mode.

Under some of the modes, the **SPP registers may disappear or not work correctly**. If you are using SPP, then set the ECR to Standard Mode. This is one of the most common mistakes that people make.

Notes

Note¹ Using Interrupts is available in PDF from
<http://www.geocities.com/SiliconValley/Bay/8302/interrupt.pdf> (62kb)

Note² Interfacing the Enhanced Parallel Port is available in PDF from
<http://www.geocities.com/SiliconValley/Bay/8302/epp.pdf> (33kb)

Note³ Interfacing the Extended Capabilities Port is available in PDF from
<http://www.geocities.com/SiliconValley/Bay/8302/ecp.pdf> (53kb)

Craig Peacock's Interfacing the PC

<http://www.senet.com.au/~cpeacock>

<http://www.geocities.com/SiliconValley/Bay/8302/>

Copyright February 1998 Craig Peacock.

Any errors, ideas, criticisms or problems, please contact the author at cpeacock@senet.com.au