Introduction

- Embedded systems often have many special I/O devices, so I/O interfacing is a critical task.
- I/O interfacing includes both physical connections and software routines that affect information exchange.
- Chapter 3 introduces basic interfacing methods.
- Chapter 4 introduces interfacing using interrupts.

Performance Measures

- Latency is the delay from when an I/O device needs service until the service is initiated.
  - It includes both hardware and software delays.
  - Real-time systems guarantee a worst-case latency.
  - Throughput or bandwidth is maximum rate data can be processed.
  - Can be limited by I/O device or the software.
  - Priority determines order of service when two or more requests are made at the same time.
  - Soft real time system is one that supports priority.

Synchronizing the Software with the State of the I/O

- Hardware is in 1 of 3 states: idle, busy, or done.
- When working, device alternates between busy and done.
- I/O devices usually much slower than software, so synchronization is required for proper transmission.
- When an I/O device is slower than software, it is I/O bound, otherwise it is CPU bound.
- Interface can be buffered or unbuffered.

Synchronizing Software with an Input Device

Synchronizing Software with an Output Device
Synchronization Mechanisms

- **Blind cycle** - software waits a fixed amount of time for the I/O to complete.
- **Gadfly or busy waiting** - software loops checking the I/O status waiting for the done state.
- **Interrupt** - I/O device causes special software to execute when required.
- **Periodic polling** - clock interrupts periodically to check I/O status.
- **Direct memory access** - I/O device directly transfers data to/from memory.

---

**Blind Cycle Printer Interface**

![Blind Cycle Printer Interface](image)

```c
void Init(void){
  DDRT = 0xFF; // outputs
  DDRM |= 0x01; // PM0 GO
  PTM |= 1; // GO=1
  Timer_Init();} // Program 2.6

void Out(unsigned char value){
  PTT = value;
  PTM&=~0x01; // GO=0
  PTM|=0x01; // GO=1
  Timer_Wait(10000);} // 10ms
```

---

**Blind Cycle ADC Interface**

![Blind Cycle ADC Interface](image)

```c
void Init(void){
  DDRT = 0x00; // input DATA
  DDRM |= 0x01; // PM0 GO
  PTM &=-0x01; // GO=0
  Timer_Init();} // Program 2.6

unsigned char In(void){
  PTM |= 0x01; // GO=1
  PTM &=~0x01; // GO=0
  Timer_Wait(10); // 10us
  return(PTT);}
```

---

**Gadfly or Busy Waiting Synchronization**

![Gadfly or Busy Waiting Synchronization](image)
Key Wakeup Interrupts in the 6812

- Allows active edge on an input to set a flag or generate an interrupt.
- Available on two pins of Port J (PJ7 and PJ6) and all 8 pins of Port P.
- Only PP5 is available on your module.
- Direction register, DDRJ and DDRP, sets input or output.
- Active input edge sets a flag in PIFJ and PIFP.
- Either edge can be configured to be active using PPSJ and PPSP.
- Each has a separate interrupt arm bit in PIEJ and PIEP.
- Key wakeup interrupt generated if flag bit set, arm bit set, and interrupts enabled (I=0).
- Ports also have built in pull up or pull down resistors which are configured using PPSJ/PPSP and PERJ/PERP registers.
- RDRJ and RDRP determine drive strength, if bit is 1 then uses 1/3 drive current to save power.

Gadfly Keyboard Interface Using Latched Input

```c
void Init(void){ // PJ7=STROBE
    DDRJ = 0x00; // PT6-0 DATA
    DDRT = 0x80; // PT7 unused output
    PPSJ = 0x80; // rise on PJ7
    PIFJ= 0x80;} // clear flag7
unsigned char In(void){
    while((PIFJ&0x80)==0); // wait
    PIFJ = 0x80; // clear flag7
    return(PTT);
}
```

Gadfly ADC Interface Using Simple Input
Initialize and Read from an ADC

```c
void Init(void) { // PJ7=DONE in
  DDRJ = 0x40; // PJ6=GO out
  PPSJ = 0x80; // rise on PJ7
  DDRT = 0x00; // PT7-0 DATA in
  PTJ &=~0x40; // GO=0

  unsigned char In(void){
    PIFJ=0x80; // clear flag7
    PTJ |= 0x40; // GO=1
    PTJ &=~0x40; // GO=0
    while((PIFJ&0x80)==0);
    return(PTT);
  }
```
**Initialization of the DS1620**

```c
void DS_Init(void) { // PT7=RST=0
    DDRT = 0xE0; // PT6=CLK=1
    PTT = 0x60; // PT5=DQ=1
}
```

**Send 8-bits Out to the DS1620**

```c
void out8(char code) { int n;
    for(n=0;n<8;n++) {
        PTT &= 0xBF; // PT6=CLK=0
        if(code&0x01)
            PTT |= 0x20; // PT5=DQ=1
        else
            PTT &= 0xDF; // PT5=DQ=0
        PTT |= 0x40; // PT6=CLK=1
        code = code>>1;
    }
}
```

**Start/Stop the DS1620**

```c
void DS_Start(void) {
    PTT |= 0x80; // PT7=RST=1
    out8(0xEE);
    PTT &= 0x7F; // PT7=RST=0
}
void DS_Stop(void) {
    PTT |= 0x80; // PT7=RST=1
    out8(0x22);
    PTT &= 0x7F; // PT7=RST=0
}
```

**Config the DS1620**

```c
void DS_Config(char data) {
    PTT |= 0x80; // PT7=RST=1
    out8(0x0C);
    out8(data);
    PTT &= 0x7F; // PT7=RST=0
}
```

**Send 9-bits Out to the DS1620**

```c
void out9(short code) { short n;
    for(n=0;n<9;n++){
        PTT &= 0xBF; // PT6=CLK=0
        if(code&0x01)
            PTT |= 0x20; // PT5=DQ=1
        else
            PTT &= 0xDF; // PT5=DQ=0
        PTT |= 0x40; // PT6=CLK=1
        code = code>>1;
    }
}
```

**Set Threshold Registers on the DS1620**

```c
void DS_WriteTH(short data) {
    PTT |= 0x80; // PT7=RST=1
    out8(0x01);
    out9(data);
    PTT &= 0x7F; // PT7=RST=0
}
void DS_WriteTL(short data) {
    PTT |= 0x80; // PT7=RST=1
    out8(0x02);
    out9(data);
    PTT &= 0x7F; // PT7=RST=0
}
```
Read 8-bits from the DS1620

```c
unsigned char in8(void){
    short n; unsigned char result;
    DDRT &= 0xDF; // PT5=DQ input
    for(n=0;n<8;n++){
        PTT &= 0xBF; // PT6=CLK=0
        result = result>>1;
        if(PTT&0x20) result |= 0x80; // PT5=DQ=1
        PTT |= 0x40; // PT6=CLK=1
    }
    DDRT |= 0x20; // PT5=DQ output
    return result;
}
```

Read 9-bits from the DS1620

```c
unsigned short in9(void){
    short n; unsigned short result=0;
    DDRT &= 0xDF; // PT5=DQ input
    for(n=0;n<9;n++){
        PTT &= 0xBF; // PT6=CLK=0
        result = result>>1;
        if(PTT&0x20) result |= 0x0100; // PT5=DQ=1
        PTT |= 0x40; // PT6=CLK=1
        DDRT |= 0x20; // PT5=DQ output
    }
    return result;
}
```

Read the Configuration Register on the DS1620

```c
unsigned char DS_ReadConfig(void){
    unsigned char value;
    PTT |= 0x80; // PT7=RST=1
    out8(0xAC);
    value = in8();
    PTT &= 0x7F; // PT7=RST=0
    return value;
}
```

Read the Temperatures from the DS1620

```c
unsigned short DS_ReadTH(void){
    unsigned short value;
    PTT |= 0x80; // PT7=RST=1
    out8(0xA1);
    value = in9();
    PTT &= 0x7F; // PT7=RST=0
    return value;
}
```

Read the Temperatures from the DS1620 (cont)

```c
unsigned short DS_ReadTL(void){
    unsigned short value;
    PTT |= 0x80; // PT7=RST=1
    out8(0xA2);
    value = in9();
    PTT &= 0x7F; // PT7=RST=0
    return value;
}
```

Read the Temperatures from the DS1620 (cont)

```c
unsigned short DS_ReadT(void){
    unsigned short value;
    PTT |= 0x80; // PT7=RST=1
    out8(0xAA);
    value = in9();
    PTT &= 0x7F; // PT7=RST=0
    return value;
}
```