Interrupt Polling Using Linked Lists

- ISR using polled interrupts must check status of all devices that may have caused the interrupt.
- Must poll when two devices share the same interrupt vector (ex., IRQ and STAF, SCI).
- Sometimes poll anyway to verify status of the device to help detect software or hardware errors.
- Polling using a linked list makes it easier to debug, change the polling order, add devices, or subtract devices.

Linked List Data Structure for Polling

```
const struct Node {
    unsigned char *StatusPt; /* Ptr to status reg */
    unsigned char Amask; /* And Mask */
    unsigned char Cmask; /* Compare Mask */
    void (*Handler)(void); /* Handler */
    const struct Node* NextPt; /* Link to Next Node */
};

unsigned char CLdata, PIAAdata, PIABdata;
void STRAHan(void) { // regular functions that
    CLdata=PORTCL;} // return (rts) when done
void PIAnHanA(void){
    PIAAdata=ADATA;}
void PIAnHanB(void){
    PIAbdata=BDATA;}
```
Interrupt Polling Using Linked Lists in C

typedef const struct Node 
{
    NodePtr NextPt
    } NodeType;
NodePtr sys[3]={
    {&PIOC, 0xFF, 0xC0, STRAHan, &sys[1]},
    {&ACNT, 0x87, 0x85, PIAHanA, &sys[2]},
    {&BCNT, 0x7C, 0x4C, PIAHanB, 0} 
};

#pragma interrupt_handler IRQHan()
void IRQHan(void){
    NodePtr Pt;
    unsigned char Status;
    Pt=sys[0];
    while(Pt){// executes device handlers
        Status=*(Pt->StatusPt);
        if((Status&(Pt->Amask))==(Pt->Cmask)){
            (*Pt->Handler)();
        } /* Execute handler */
        Pt=Pt->NextPt;
    } // returns after all devices polled
}

void main(void){
    while(1);
}

Interrupt Polling Using Linked Lists in Assembly

start fdb 11STAF place to start polling
Sreg equ 0 Index to Status Register
Amask equ 2 and mask
Cmask equ 3 compare mask
DevHan equ 4 device handler
NextPt equ 6 next pointer
num fcb 3 number of devices
11STAF fdb $1002 address of PIOC
fcb $ff look at all the bits in PIOC
fcb $C0 expect exact match with $C0
fdb STAHan device handler
fdb 11CA1 ptr to next device to poll

Interrupt Polling Using Linked Lists in Assembly

IrqHan 1dx start Reg X points to linked list
ldab num number of possible devices
next 1dy Sreg,x Reg Y points to status reg
ldaa ,y read status
anda Amask,x clear indeterminate bits
cmpa Cmask,x expected value if active
bne Notyet skip if not requesting
1dy DevHan,x Reg Y points to handler
jsr ,y call device handler
Notyet 1dx NextPt,x Reg X points to next entry
decb device counter
bne next check next device
rti
Fixed Priority Using One Interrupt Line

- Device 1
- Device 2

Fixed Priority Implemented Using XIRQ

- Device 1
- Device 2

Round-Robin Polling

- Sometimes we want to have no priority.
- This gives a guarantee of service under a heavy load to equally important devices.
- Round-robin polling rotates the polling order to allow all devices an equal chance of getting service.
- Does not apply to vectored interrupts.

- Example sequence of events:
  - Interrupt, poll A, B, C
  - Interrupt, poll B, C, A
  - Interrupt, poll C, A, B
  - Interrupt, poll A, B, C, etc.

Round-Robin Polling in C

```c
NodeType sys[3]=
{&PIOC, 0xFF, 0xC0, STRAHan, &sys[1]},
{&ACNT, 0x87, 0x85, PIAHanA, &sys[2]},
{&BCNT, 0x7C, 0x4C, PIAHanB, &sys[0]};
NodePtr Pt=&sys[0]; // polled first last interrupt
#pragma interrupt_handler IRQHan()
void IRQHan(void){
  unsigned char Counter,Status;
  Counter=3; // quit after 3 devices checked
  Pt=Pt->NextPt; // rotates polling orders
  while(Counter--){
    Status=*Pt->StatusPt;
    if((Status&(Pt->Amask))==Pt->Cmask){
      (*Pt->Handler)();} /* Execute handler */
    Pt=Pt->NextPt; }
}
Real-Time Interrupts and Periodic Polling

- A real-time interrupt (RTI) is one that is requested on a fixed time basis.
- Required for data acquisition and control systems because servicing must be performed at accurate time intervals.
- RTIs are also used for intermittent or periodic polling.
- In gadfly, I/O devices polled continuously.
- With periodic polling, I/O devices polled on regular basis.
- If no device needs service, interrupt simply returns.
- Use periodic polling if the following conditions apply:
  1. The I/O hardware cannot generate interrupts directly.
  2. We wish to perform I/O functions in the background.

Periodic Polling

Real Time Interrupt Using a 6811 STRA

Periodic Interrupt Using External Clock in Assembly

- TIME rmb 2 incremented every 1ms
- RITAL sel disable interrupts during RITAL
  ldaa #$42 PASC STAI=1, HNDS=0, EGA=1
  staa $1002 Arm interrupt on rise of STRA
  ldd #0
  std TIME initialize variable
  ldaa $1005 initially clear STRA
  cli enable
  rts
- IRQHAN ldaa $1002 11000010 if interrupting
  cmpa #$C2
  beq CLKHAN
  swi
- CLKHAN ldaa $1005 Acknowledge
  ldx TIME
  inx
  stx TIME
  rti
### Periodic Interrupt Using RTI in Assembly

```c
RITUAL sei       // disable interrupts during RITUAL
ldaa #3         // Set RTR1,RTR0 = 11
staa $102       // Interrupt period = 32.768ms
ldaa #$40       // Set RTII=1
staa $1024      // Enable IRQ interrupts
cli             // Make ritual atomic
RITIHAN ldaa $1025  // expect=X1XX0000
                   // ignore TOF, PAOVF, and PAIF
                   // cmpa #$40 // RTIF should equal 1
                   // beq OK
                   // swi // Error
OK ldaa #$40     // RTIF is cleared by writing to TFLG2
staa $1025      // with bit 6 set
                   // * service occurs every 32.768ms or about 30.517Hz
rti             // rti
```

### Periodic Interrupt Using RTI in C

```c
unsigned int Time;
#pragma interrupt_handler RTIHan()
void RTIHan(void){
    if((TFLG2&0x4F)!=0x40) asm("swi"); /* Illegal */
    TFLG2=RTIF; /* Acknowledge by clearing RTIF */
    Time++;
}
void Ritual(void){
    asm("sei"); /* Make ritual atomic */
    PACTL=(0xFC&TMSK2)|2; /* Set RTR to 2, 61.035Hz */
    TMSK2|RTII; /* Set RTI */
    Time=0; /* Initialize global data */
    asm("cli");
}
```

---

**6811 RTI**

<table>
<thead>
<tr>
<th>RTR1</th>
<th>RTR0</th>
<th>Divide by</th>
<th>Period (µs)</th>
<th>Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$2^{15}$</td>
<td>4.096</td>
<td>244.14</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$2^{14}$</td>
<td>8.192</td>
<td>122.07</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$2^{16}$</td>
<td>16.384</td>
<td>61.035</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$2^{16}$</td>
<td>32.768</td>
<td>30.517</td>
</tr>
</tbody>
</table>