

Lecture 6 objectives

- Software latency and real-time systems
- Blind-cycle versus busy-wait synchronization
- Output compare interrupts

Software Latency is the time between when the I/O device needs service, and the time when service is initiated.

For an input device,
latency is the time between new input data ready,
and the software reading the data.

For an output device,
latency is the time between output device idle,
and the software giving the device new data to output.

Periodic events (sampling ADC, outputting to DAC).
latency is the time between when it is supposed to be run,
and when it is actually run.

A **real time** system is one that can guarantee a worst case software latency. In other words, there is an upper bound on the software response time.

Hardware Latency or **device latency** is the time between when an I/O device is given a command, and the time when command is completed.

bandwidth is maximum data flow that can be processed
Sometimes the bandwidth is limited by the I/O device,
while other times it is limited by computer software.
can be reported as an overall average or a short-term maximum.

Priority determines the order of service when two or more requests are made simultaneously.
a "soft-real-time" system supports priority.

Five ways to synchronize

| | |
|-------------------------|--|
| Blind cycle | Fixed time delay |
| Busy Waiting | Software checks flag over and over |
| Interrupt | Flag causes ISR to run |
| Periodic Polling | Software check flag in periodic interrupt |
| DMA | I/O device reads/writes directly into RAM |

I/O bound is defined as
Bandwidth is limited by speed of I/O device
Making the I/O device faster will increase bandwidth
Making the software run faster will not increase bandwidth
Software often waits for the I/O device

CPU bound is defined as

- Bandwidth is limited by speed of executing software
- Making the I/O device faster will not increase bandwidth
- Making the software run faster will increase bandwidth
- Software does not have to wait for the I/O device

Synchronizing the software with an input device

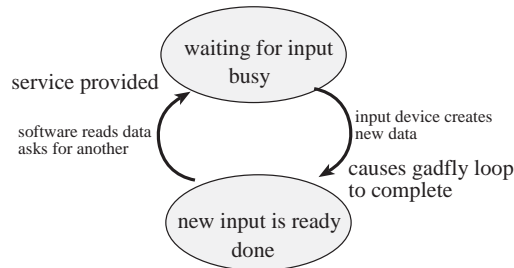
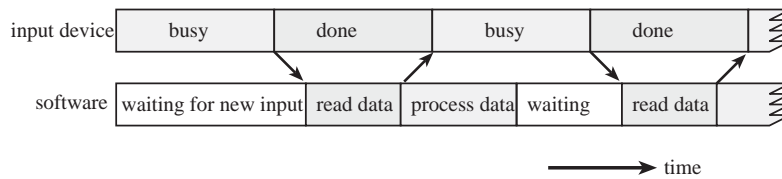
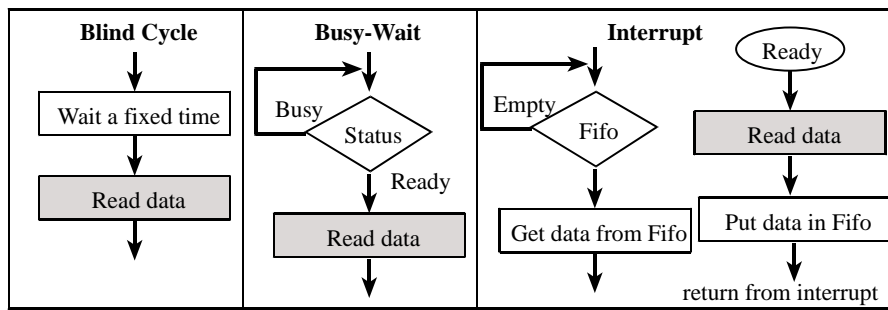


Figure 3.1. The input device sets a flag when it has new data.

Synchronizing with an input device



Is this I/O bound or CPU bound?

Answer: a little bit of both because in the done state hardware waits for software. In the busy state software waits for hardware.

Synchronizing the software with an output device

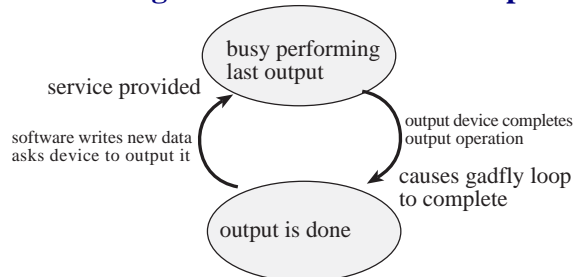


Figure 3.3. The output device sets a flag when it has finished outputting the last data.

Synchronizing with an output device

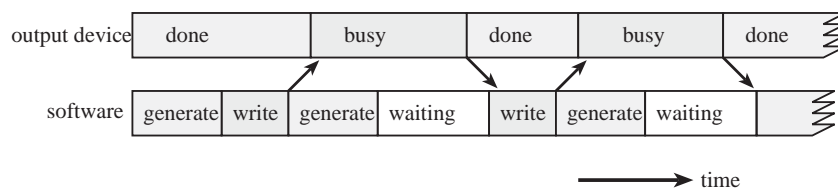
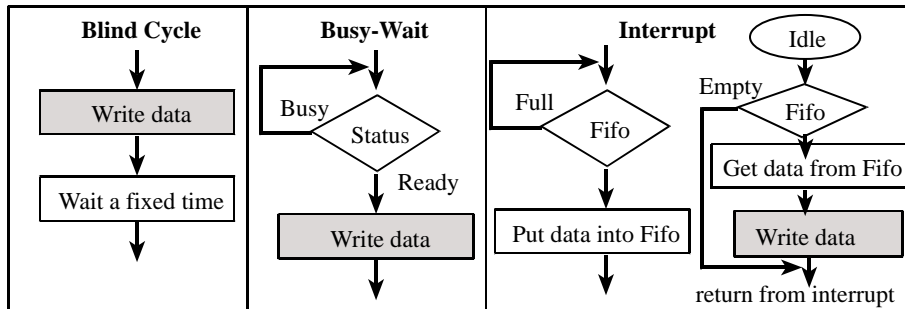


Figure 3.4. The software must wait for the output device to finish the previous operation.

3.2. Blind Cycle Counting Synchronization

Blind cycle counting is appropriate when the I/O delay is fixed and known. This type of synchronization is blind because it provides no feedback from the I/O back to the computer. Example

```
//-----wait-----
// time delay
// Input: time in 0.667usec
// Output: none
void static wait(unsigned short cycles){
  unsigned short startTime = TCNT;
  while((TCNT-startTime) <= cycles){}
}
//-----outCsr-----
// sends one command code to the LCD
// Input: command is 8-bit function
// Output: none
static void outCsr(unsigned char command){
  PTM = 0x0F&(command>>4); // ms nibble
  PTM |= 0x10;             // E goes 0,1
  PTM &= ~0x10;           // E goes 1,0
  PTM = command&0x0F;    // ls nibble
  PTM |= 0x10;           // E goes 0,1
  PTM &= ~0x10;         // E goes 1,0
  wait(135);             // blind cycle 90us wait
}
```

3.3. Gadfly or Busy Waiting Synchronization

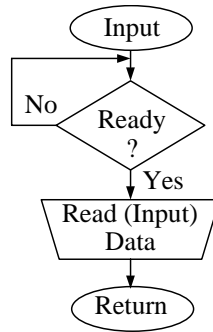


Figure 3.7. A software flowchart for gadfly input.

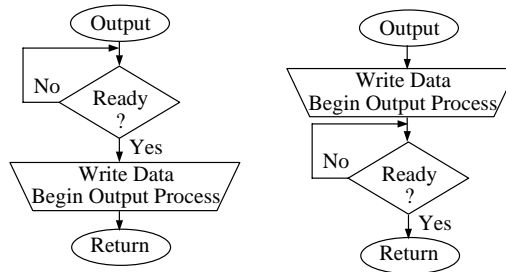


Figure 3.8. Two software flowcharts for gadfly output.

Busy Waiting with three devices

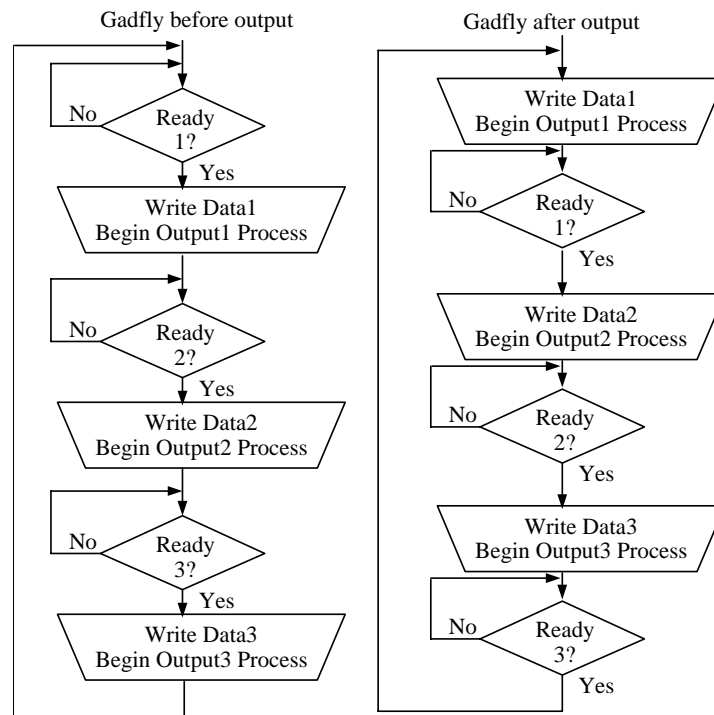


Figure 3.9. Two software flowcharts for multiple gadfly outputs.

| Time(ms) | Gadfly Before Event | Gadfly After Event |
|-------------|---------------------|--------------------|
| 0 | Start 1,2,3 | Start1 |
| from 0 to 1 | Wait for 1 | Wait for 1 |
| 1 | Start 1,2,3 | Start 2 |
| from 1 to 2 | Wait for 1 | Wait for 2 |
| 2 | Start 1,2,3 | Start 3 |
| from 2 to 3 | Wait for 1 | Wait for 3 |
| 3 | Start 1,2,3 | Start1 |
| from 3 to 4 | Wait for 1 | Wait for 1 |
| 4 | Start 1,2,3 | Start 2 |
| from 4 to 5 | Wait for 1 | Wait for 2 |

Performance Tip: Whenever we can establish concurrent I/O operations, we can expect an improvement in the overall system bandwidth.

Busy Waiting with multiple devices

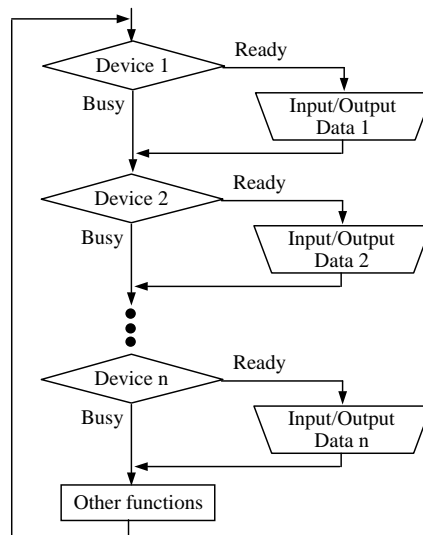


Figure 3.10. A software flowchart for multiple gadfly inputs and outputs.

Example of a three thread system

- 1) The foreground thread usually is running
- 2) The "key" background thread is invoked whenever a key is touched
- 3) The "time" background thread is invoked every 1ms in a periodic fashion.

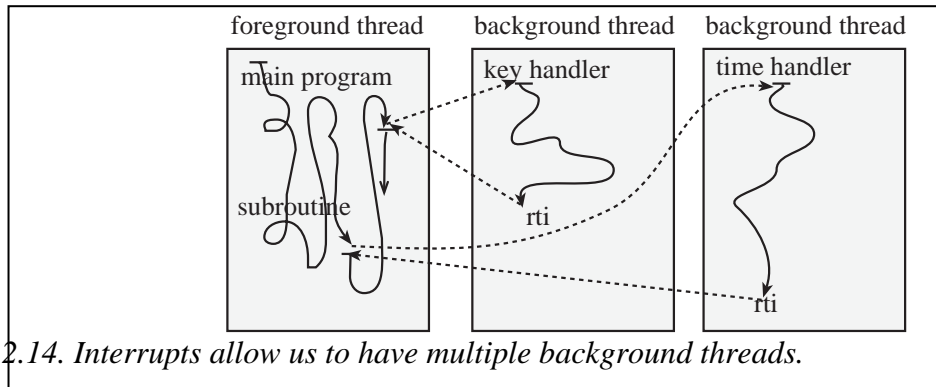


Figure 2.14. Interrupts allow us to have multiple background threads.

The running thread must be suspended in order to execute another thread.

save registers on stack

set PC to beginning of the interrupt handler

created with an "empty" stack and uninitialized registers.

When the **keyhandler** is done it executes **rti** to relinquish control

original stack and registers of the main program will be restored

We can think of each thread as having its own registers and its own stack area.

Three conditions must be true for an interrupt to occur:

- 1) Initialization software will set the arm bit, E.g., **C3I**
individual control bit for each possible flag that can interrupt
- 2) When it is convenient, the software will enable, I=0
asm cli
- 3) Hardware action (busy to done) sets a flag E.g., **C3F**
C3F is set at a periodic rate by an output compare internal clock

What happens when an interrupt is processed?

- 1) execution of main program is suspended (current instruction is finished),
pushes registers on the stack,
sets the I bit,
gets the vector address from high memory, E.g., **0xFFE8**
- 2) the interrupt service routine, or background thread is executed,
clears the flag that requested the interrupt, **C0F** cleared by writing a 1 to it
performs necessary operations
communicates using global variables
- 3) the main program is resumed when the interrupt service routine executes **rti**.
pulls the registers from the stack

9S12C32 interrupts we will be using

| | |
|--------|------------------------------|
| 0xFFD6 | interrupt 20 SCI |
| 0xFFDE | interrupt 16 timer overflow |
| 0xFFE0 | interrupt 15 timer channel 7 |
| 0xFFE2 | interrupt 14 timer channel 6 |
| 0xFFE4 | interrupt 13 timer channel 5 |

| | |
|---------------|-------------------------------------|
| 0xFFE6 | interrupt 12 timer channel 4 |
| 0xFFE8 | interrupt 11 timer channel 3 |
| 0xFFEA | interrupt 10 timer channel 2 |
| 0xFFEC | interrupt 9 timer channel 1 |
| 0xFFEE | interrupt 8 timer channel 0 |
| 0xFFFF0 | interrupt 7 RTI real time interrupt |
| 0xFFFF6 | interrupt 4 SWI software interrupt |
| 0xFFFF8 | interrupt 3 trap software interrupt |
| 0xFFFFE | interrupt 0 reset |

MC9S12C32 Periodic Interrupt using Output Compare 3

| address | msb | | | | | | | | | | | | | | | lsb | Name |
|---------------|-----------|-----------|-----------|-----------|-----------|-----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|-------------|
| \$0044 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TCNT |
| \$0050 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TC0 |
| \$0052 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TC1 |
| \$0054 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TC2 |
| \$0056 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TC3 |
| \$0058 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TC4 |
| \$005A | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TC5 |
| \$005C | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TC6 |
| \$005E | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TC7 |

| Address | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 | Name |
|---------|------------|-------|-------|-------|-------------|------------|------------|------------|-------|
| \$0046 | TEN | TSWAI | TSBCK | TFFCA | 0 | 0 | 0 | 0 | TSCR1 |
| \$004D | TOI | 0 | 0 | 0 | TCRE | PR2 | PR1 | PR0 | TSCR2 |
| \$0040 | IOS7 | IOS6 | IOS5 | IOS4 | IOS3 | IOS2 | IOS1 | IOS0 | TIOS |
| \$004C | C7I | C6I | C5I | C4I | C3I | C2I | C1I | C0I | TIE |
| \$004E | C7F | C6F | C5F | C4F | C3F | C2F | C1F | C0F | TFLG1 |
| \$004F | TOF | 0 | 0 | 0 | 0 | 0 | 0 | 0 | TFLG2 |

Table MC9S12C32 registers used to configure periodic interrupts.

The rate is dependent of the **PLL**, **TSCR2**, and **1000** in **ISR**

Things you must do in every interrupt service routine

Acknowledge (clear flag that requested interrupt)

Things you must do in an OC interrupt service routine

Acknowledge (make C3F become zero)

Set the timer to specify when to interrupt next

```
interrupt 11 void TC3handler(void){
// executes at 1000 Hz
    TFLG1 = 0x08;    // acknowledge OC3
    TC3 = TC3+1000; // 1 ms
// do something
    PTT ^= 0x01;    // toggle LED
}
```

Things you must do in a ritual

Arm (specify a flag may interrupt)

Enable (allow all interrupts on the 6812)

Things you must do in an OC ritual

Turn on TCNT (TEN=1)

Set channel to output compare (TIOS)

Specify TCNT rate (TSCR2, PACTL, PLL)

Arm (C3I=1)

When to generate first interrupt

Enable (I=0)

```
//-----OC3_Start-----
// arm output compare 3 for 1000Hz interrupt
// Input: none
// Output: none
void OC3_Start(void){
    asm sei          // make ritual atomic
    TSCR1 = 0x80;    // Enable TCNT
    TIOS |= 0x08;    // TC3 as output compare
    TSCR2 = 0x02;    // divide by 4
    PACTL = 0;       // timer prescale used
/* three bits of TSCR2 (PR2,PR1,PR0) determine TCNT period
    divide  FastMode(24MHz)      Slow Mode (4MHz)
000  1      42ns  TOF  2.73ms  250ns TOF 16.384ms
001  2      84ns  TOF  5.46ms  500ns TOF 32.768ms
010  4     167ns  TOF 10.9ms   1us  TOF 65.536ms
011  8     333ns  TOF 21.8ms   2us  TOF 131.072ms
100 16     667ns  TOF 43.7ms   4us  TOF 262.144ns
101 32    1.33us  TOF 87.4ms   8us  TOF 524.288ms
110 64    2.67us  TOF 174.8ms 16us  TOF 1.048576s
111 128   5.33us  TOF 349.5ms 32us  TOF 2.097152s */
    TIE  |= 0x08;    // arm OC3
    TC3  = TCNT+50;  // first int right away
    asm cli
}
void main(void) {
    DDRT |= 0x03;    // PTT bits 1,0 out to LEDs
    OC3_Start();    // interrupt every 1ms
    for(;;) {
        PTT ^= 0x02; // toggle LED
    }
}
```

