SUMMARY SLIDE

Interrupt example, Three interrupts (Interrupt 1 is highest priority)
Try again, with different priority (what was #3 is now #1)
There are no critical regions,
the longest instruction takes 1 ms to execute
Interrupts run with interrupts disabled
For interrupt #1, T_{P1} = 4 ms, T_1= 1.0 ms, T_{1+}= 2.5 ms
For interrupt #2, T_{P2} = 60 ms, T_2= 1.0 ms, T_{2+}= 2.5 ms
For interrupt #3, T_{P3} = 20 ms, T_3= 2.5 ms, T_{3+}= 1.0 ms

Step 1: Check interrupt density. $\frac{1}{4} + \frac{1}{60} + \frac{2.5}{20} = \frac{30}{120} + \frac{2}{120} + \frac{14}{120} = \frac{46}{120} < 1.000$ $\boxed{\text{OK}}$

Step 2: Find maximum latency for each interrupt, the T_{i+} values

Step 3: Find the interrupt interval constraint for each interrupt, start with highest priority. General formula is:

$$T_{i+} + \sum_{x=1}^{i} N(i,x)T_i \overset{?}{<} T_{Pi}$$

For interrupt #1 (i = 1) $T_{1+} + N(1{,}1)T_1 \overset{?}{<} T_{P1} \Longrightarrow 2.5 + (1)(1.0) \overset{?}{<} 4.0\,0 \Longrightarrow 3.5 < 4.0$ $\boxed{\text{OK}}$

For interrupt #2 (i = 2) $T_{2+} + N(2{,}2)T_2 + N(2{,}1)T_1 \overset{?}{<} T_{P2}$

But now I need $N(2{,}1) = \left\lceil \frac{T_{P2}-T_2}{T_{P1}} \right\rceil = \left\lceil \frac{60-1}{4} \right\rceil = \frac{59}{4} = 15$

$2.5 + (1)(1) + (15)1 \overset{?}{<} 60 \Longrightarrow 18.5 < 60$ $\boxed{\text{OK}}$

For interrupt #3 (i = 3) $T_{3+} + N(3{,}3)T_3 + N(3{,}2)T_2 + N(3{,}1)T_1 \overset{?}{<} T_{P3}$

But now I need $N(3{,}2) = \left\lceil \frac{T_{P3}-T_3}{T_{P2}} \right\rceil = \left\lceil \frac{20-2.5}{60} \right\rceil = 1$ and I need $N(3{,}1) = \left\lceil \frac{T_{P3}-T_3}{T_{P1}} \right\rceil = \left\lceil \frac{20-2.5}{4} \right\rceil = \left\lceil \frac{17.5}{4} \right\rceil = 5$

$1 + (1)(2.5) + (1)1 + (5)1 \overset{?}{<} 20 \Longrightarrow 9.5 < 20$ $\boxed{\text{OK}}$

All three interrupt interval constraints are satisfied. Interrupts will always get serviced on time.

Critical Regions

Shared resources need to be protected from interrupts.

A *critical region* is a section of code that accesses a hardware resource that is not capable of being concurrently accessed by more than one software process.

In a multiprocessor context the matter of critical regions is more complicated than discussed here. In this more complicated case semaphores are usually used to allocate access to critical regions.

The quintessential example of a critical region is a multiple word global variable such as a long integer on an eight-bit or sixteen-bit machine when this global integer variable is used to pass information to and from an ISR. (Global variables are the primary method of sharing information with an ISR.)

Suppose the long integer contains 4 bytes with the hexadecimal value 00 00 01 00 hex (+256 decimal).
Suppose the main program is in the process of reading this variable and has already read the first three bytes (so it has read 00 00 01). Suppose that during the machine instruction to read the third byte an interrupt is requested and the ISR decrements this shared variable to 00 00 00 FF. (255 decimal).

Critical Regions

Shared resources need to be protected from interrupts.

A *critical region* is a section of code that accesses a hardware resource that is not capable of being concurrently accessed by more than one software process.

In a multiprocessor context the matter of critical regions is more complicated than discussed here. In this more complicated case semaphores are usually used to allocate access to critical regions.

The quintessential example of a critical region is a <u>multiple word global variable</u> such as a long integer on an eight-bit or sixteen-bit machine when this global integer variable is used to pass information to and from an ISR. (Global variables are the primary method of sharing information with an ISR.)

Suppose the long integer contains 4 bytes with the hexadecimal value 00 00 01 00 hex (+256 decimal). Suppose the main program is in the process of reading this variable and has already read the first three bytes (so it has read 00 00 01). Suppose that during the machine instruction to read the third byte an interrupt is requested and the ISR decrements this shared variable to 00 00 00 FF. (255 decimal). Execution is now returned to the main program which now reads the final byte (FF). The main program will receive the value of 00 00 01 FF (511 decimal).

SUMMARY SLIDE

Critical Regions

Shared resources need to be protected from interrupts.

A *critical region* is a section of code that accesses a hardware resource that is not capable of being concurrently accessed by more than one software process.

In a multiprocessor context the matter of critical regions is more complicated than discussed here. In this more complicated case semaphores are usually used to allocate access to critical regions.

The quintessential example of a critical region is a <u>multiple word global variable</u> such as a long integer on an eight-bit or sixteen-bit machine when this global integer variable is used to pass information to and from an ISR. (Global variables are the primary method of sharing information with an ISR.)

Suppose the long integer contains 4 bytes with the hexadecimal value 00 00 01 00 hex (+256 decimal). Suppose the main program is in the process of reading this variable and has already read the first three bytes (so it has read 00 00 01). Suppose that during the machine instruction to read the third byte an interrupt is requested and the ISR decrements this shared variable to 00 00 00 FF. (255 decimal). Execution is now returned to the main program which now reads the final byte (FF). The main program will receive the value of 00 00 01 FF (511 decimal). This is wildly wrong. The main program should receive either the before-interrupt or after-interrupt value (256 or 255 decimal), but not 511 decimal. This error happened because a shared resource was being *concurrently* accessed. Any code that accesses a shared resource is a *critical region* of code.

In a single-processor environment critical regions can be protected by disabling interrupts before beginning execution of a critical region and re-enabling interrupts afterward.

Critical Regions

Shared resources need to be protected from interrupts.

Copied from previous slide

A *critical region* is a section of code that accesses a hardware resource that is not capable of being concurrently accessed by more than one software process.

In a single-processor environments critical regions can be protected by disabling interrupts before beginning execution of a critical region and re-enabling interrupts afterward.

Other common examples of critical regions are access to the stack (or other memory structure such as a heap), peripheral hardware, or access to a network connection (if shared with an ISR).

A disadvantage of disabling interrupts is that a critical region then prevents ALL interrupts when in reality it only needs to typically prevent the one unique interrupt source that causes access to the shared resource when non-ISR code needs access. Also, in multiple processor environments disabling interrupts on one CPU is not adequate to protect the resource from the other CPUs. *Semaphores* provide a more sophisticated level of access control. You should know that this technique exists. We will not cover semaphores in this introductory course. These types of topics are usually covered in depth in the more advanced parts of a course on operating systems.

Scheduling Tasks

Two Types of Tasks

1) Preemtive
   Runs at scheduled time with interrupts disabled
   Runs as a procedure within the tic-clock ISR
   Must Be Short BECAUSE

2) Cooperative
   Starts running at about the scheduled time but all premptive tasks go ahead of it.
   Tick clock marks this task as something that should run now but does not actually call it.
   Main loop polls to see what cooperative tasks should be running & calls them in some order (often round-robin style)

The Schedule Table

A shared resource in global memory

A row of data for each Task

PREMPTIVE OR COOP	scheduled Start Time	Task Procedure Name	Possible other data

USUALLY AN ADDRESS TO THE PROCEDURE

Flag values are possible
e.g 0 MEANS "NOT SCHEDULED," NEVER RUN IT
e.g. < 0 MEANS "AS SOON AS POSSIBLE", NEXT CLOCK TIC

Example Turn on an LED For 10 s STARTING 5s FROM now

Read the current time
Add 5s To it
Write it into the schedule table as the start time
for the "LED on" Procedure

At the appropriate time the LED start procedure runs
Turns the LED on
Also reads start time, adds 10 s and writes that
back into the table as the start time for the
"LED off" procedure

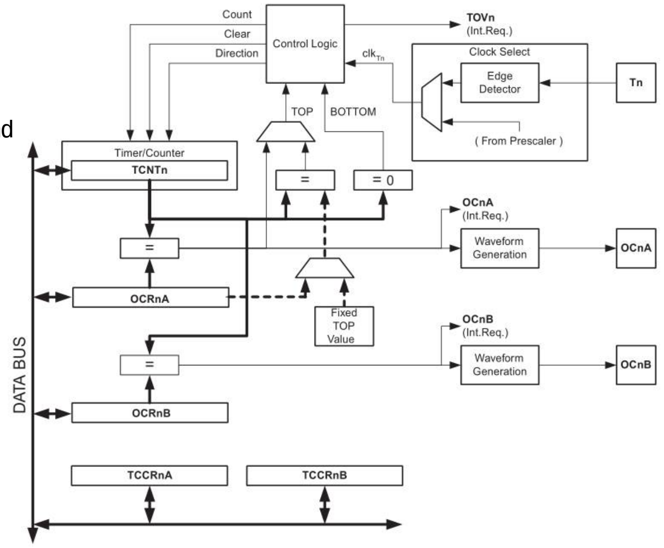
Once LED TURNS ON, return to main loop & do other good work

The "LED off" procedure will run when scheduled
Turns LED off

("Delay" is NO LONGER EVER USED)

Task Scheduling is software dependent
∴ runs at relatively slow rates with resolution of 0.1 or 1 s or so,

Figure 14-1. 8-bit Timer/Counter Block Diagram



Most microcontrollers have a counter/timer system—special support hardware for higher speed operations.

Example, PWM with an analogWrite command on the arduino.

Illustration is from the AVR datasheet

SUMMARY SLIDE

Using the hardware counter/timer support systems—faster, higher resolution than task scheduler.

1.) Input capture event

When did an input pin change? Capture that information in a register from one of the high-speed timers

Examples of use:

Log the real time of an event to a higher precision than the tic clock gives.

Set up a pin to do an input capture and simultaneously an interrupt.

The ISR will read the real time to the resolution of the real-time clock, typically about 1 s.

The input capture also stored the timer register data when the pin changed. This can be used to interpolate between the real-time-clock's increments.

Could measure a short period or a frequency—can now deal with frequencies of 1000 Hz or so.

2.) Output compare event

Make something happen at a particular time. (With more resolution than tic clock system can deliver)
You can make the tic-clock itself with an output compare event.
Pulse width modulation
Any other pulse type applications.

3.) Combine input capture and output compare techniques to do indirect period or frequency measurements.

e.g. indirect period. (measure freq. and take reciprocal)
Set up an output compare to establish a time interval.
Set up an input capture to count pulses during that interval.