Introduction to Real-Time Kernels

Signaling, Inter-Task Communications and Debugging

2013-07-19

Jean J. Labrosse

CEO, **Micriμm**

Outline

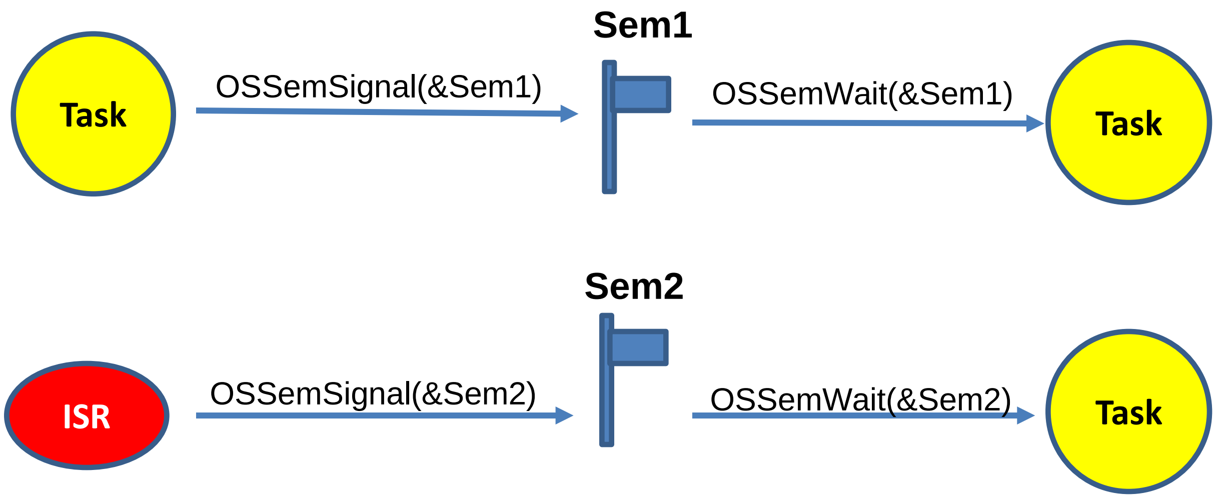
- **Signaling a Task**
  - Semaphores
  - Event Flags
- **Inter-task Communications**
- **Debugging kernel-based applications**
  - Output Port
  - DAC output
  - Kernel Aware Debuggers
  - Run-Time Kernel Awareness
  - Trace Tool
- **Summary**

Signaling a Task

(Semaphores)

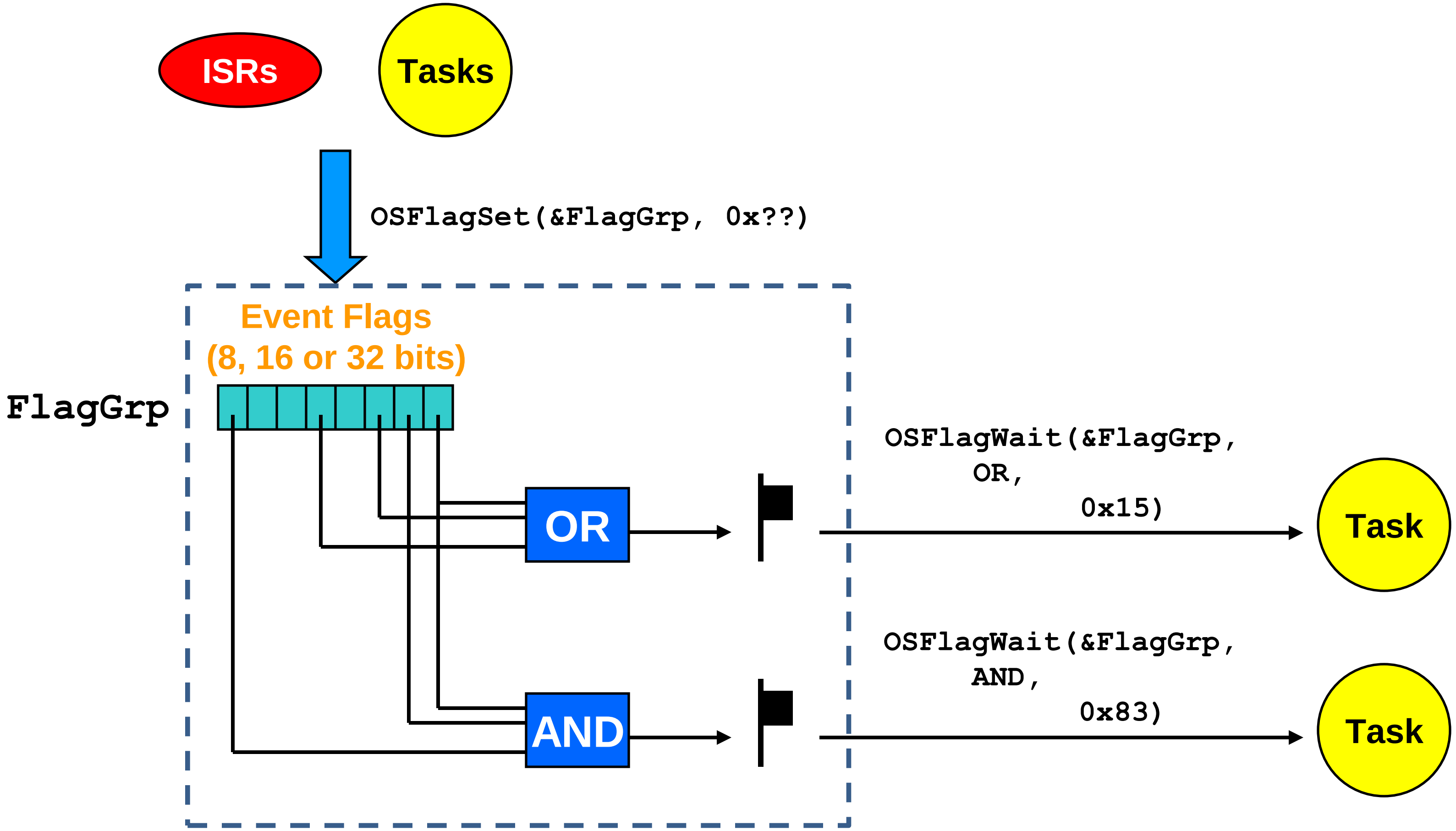
- **Semaphores are used to signal the occurrence of an event**
  - Either from an ISR or another task

- **Only tasks can wait for events**



Signaling a Task

(Event Flags)



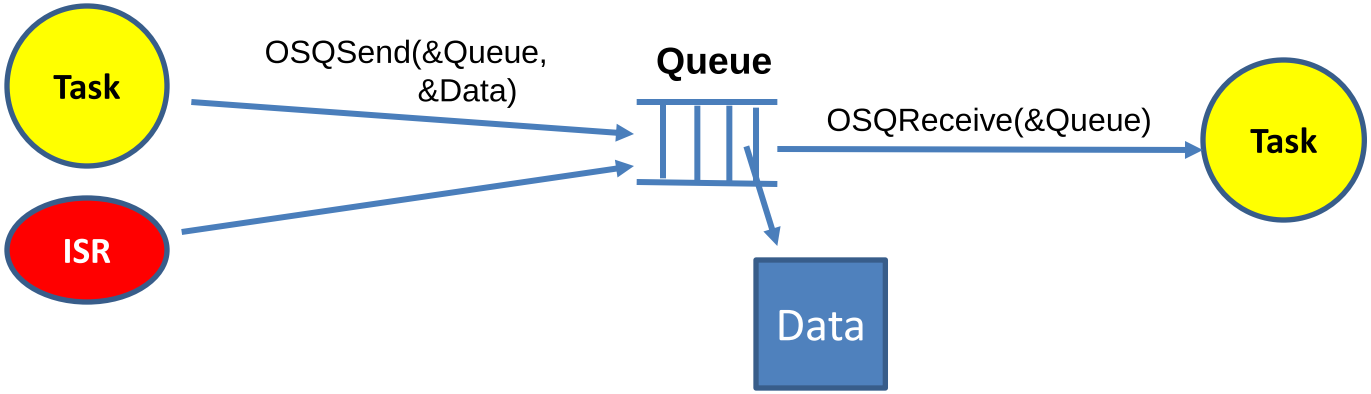
```
OSFlagSet(&FlagGrp, 0x??)
```

```
OSFlagWait(&FlagGrp,
     OR,
           0x15)
```

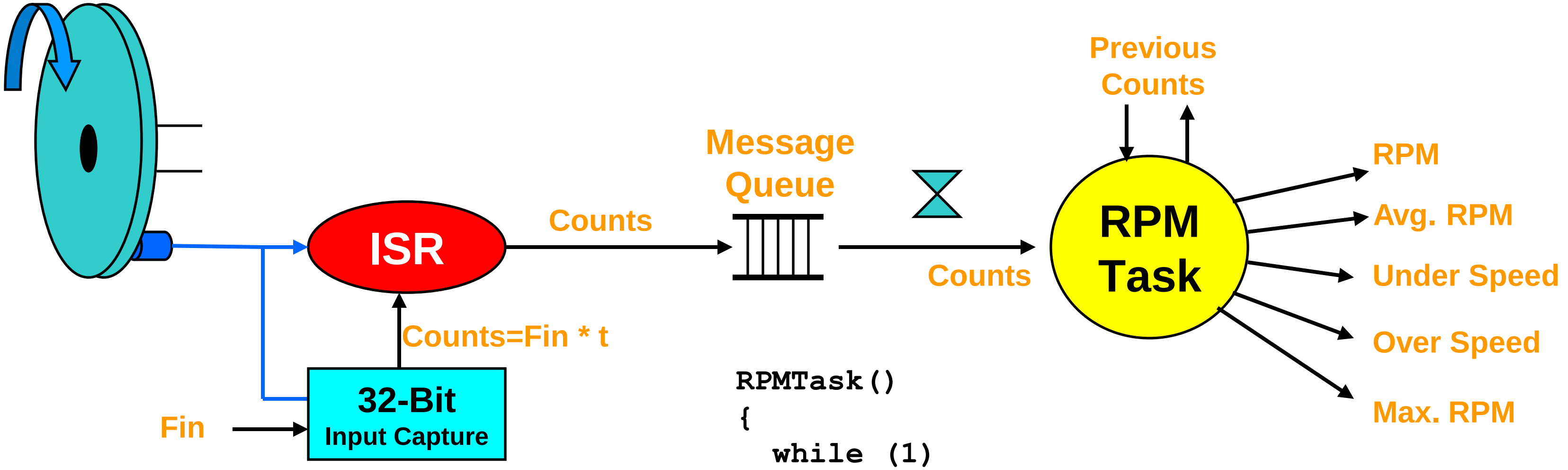
```
OSFlagWait(&FlagGrp,
     AND,
           0x83)
```

Inter-task Communications
(Message Queues)

- **Message queues are typically used to send actual data to tasks**
- **Messages are typically pointers to the actual data**
  - This avoids copying data
- **ISRs or Tasks can send messages to other Tasks**
- **Only Tasks can receive messages**



Inter-task Communications
(Message Queues)

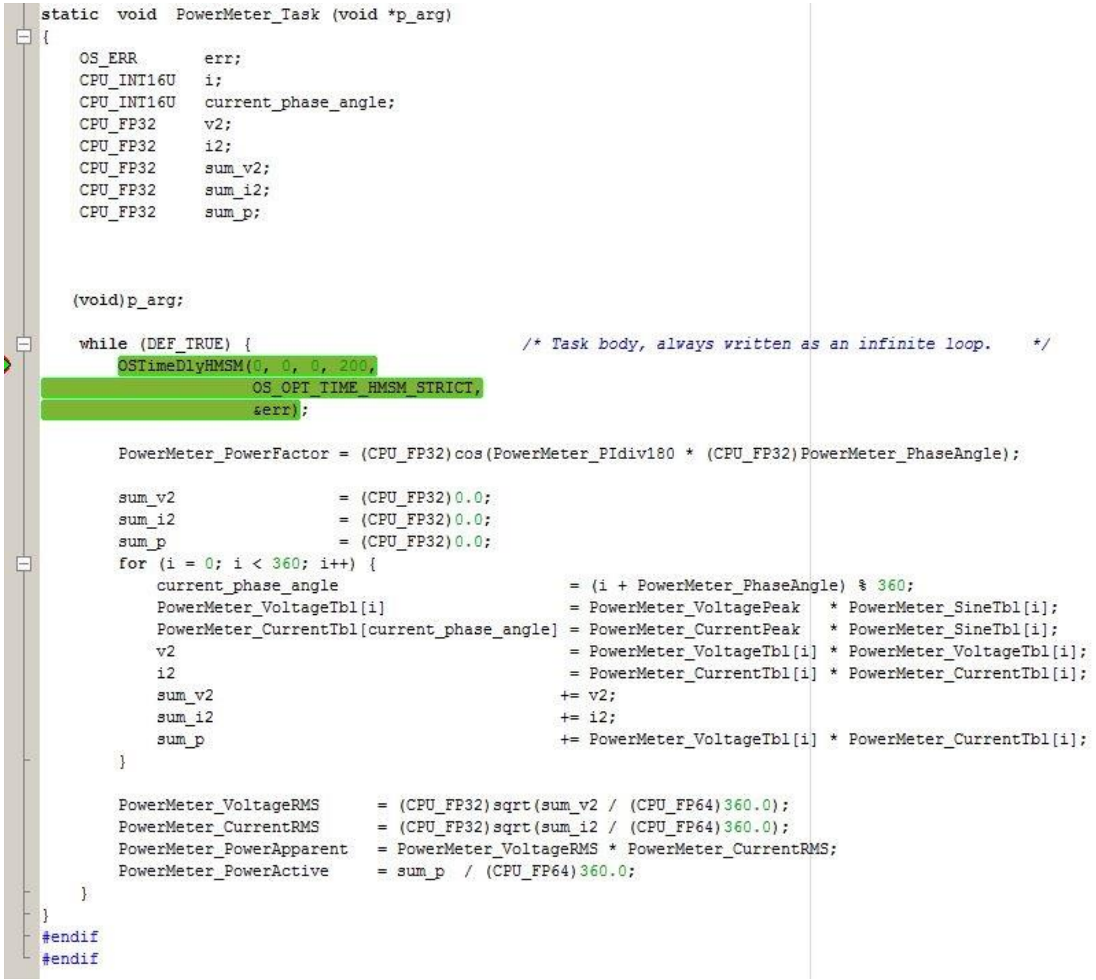


```
RPMTask()
{
  while (1)
    Wait for message from ISR (with timeout);

    if (timed out)
       RPM = 0;
    else {
       DeltaCounts     = Counts
                       - PreviousCounts;
       PreviousCounts = Count;
       RPM             = 60 * Fin / DeltaCounts;
    }
    Compute average RPM;
    Check for overspeed/underspeed;
    Keep track of peak RPM;
}
```

```
RPM_ISR()
{
  Read Input Capture;
  Post Counts;
}
```

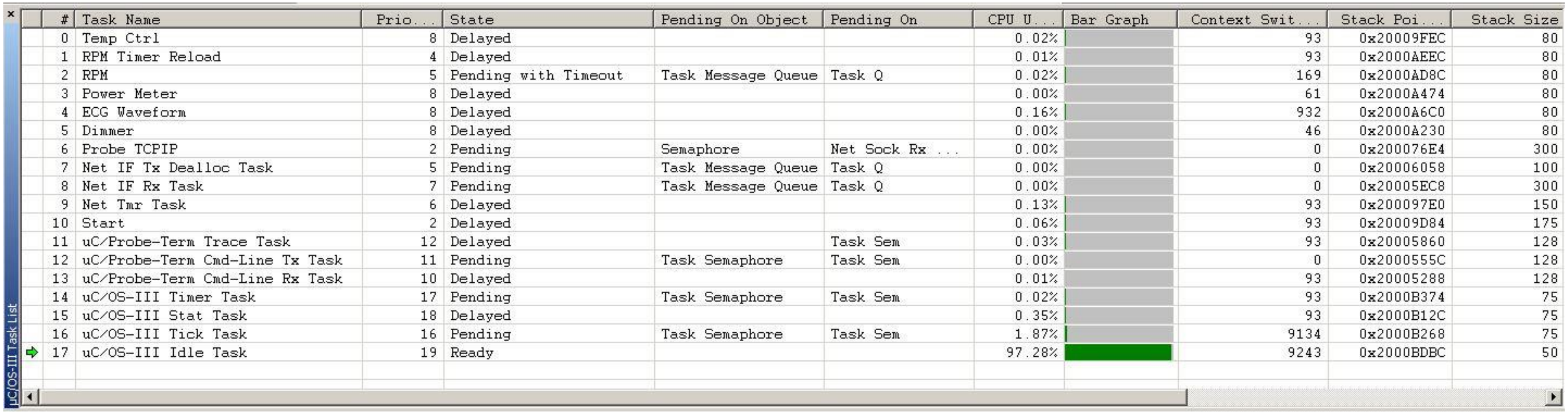
Debugging with Kernels

(Debugger)



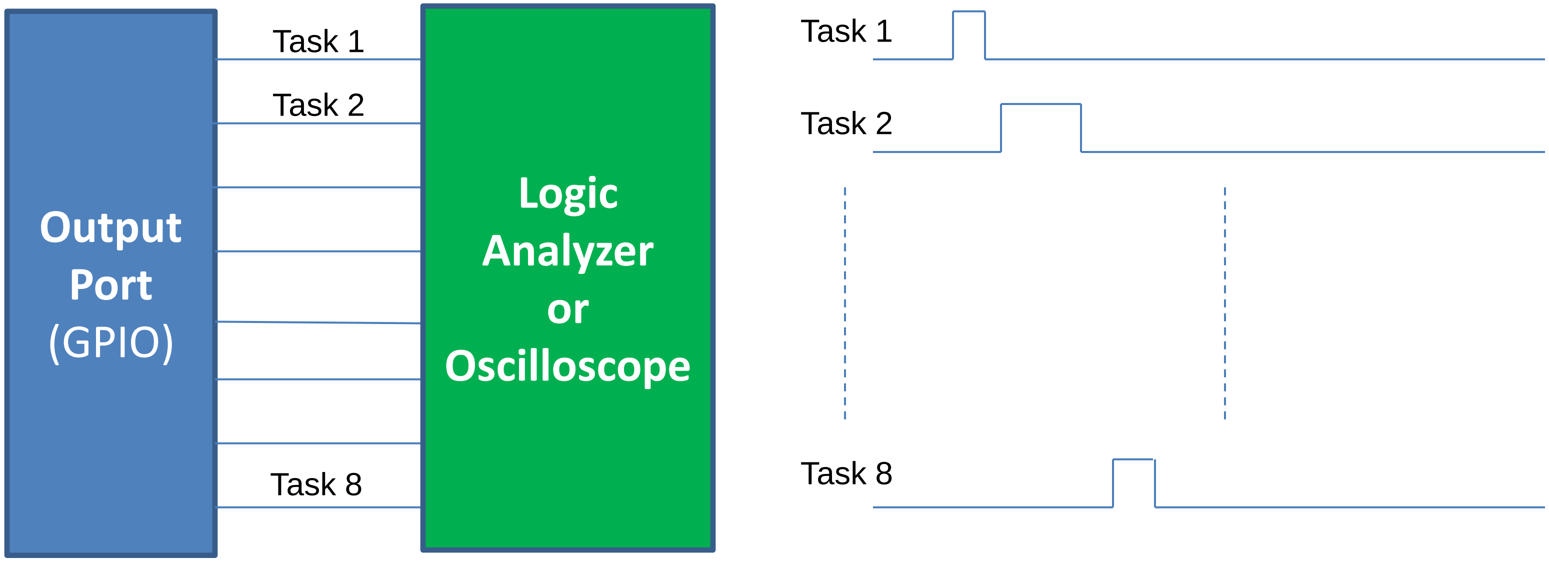
```
static  void  PowerMeter_Task (void *p_arg)
{
    OS_ERR        err;
    CPU_INT16U    i;
    CPU_INT16U    current_phase_angle;
    CPU_FP32      v2;
    CPU_FP32      i2;
    CPU_FP32      sum_v2;
    CPU_FP32      sum_i2;
    CPU_FP32      sum_p;



   (void)p_arg;

    while (DEF_TRUE) {                                        /* Task body, always written as an infinite loop.    */
        OSTimeDlyHMSM(0, 0, 0, 200,
                      OS_OPT_TIME_HMSM_STRICT,
                      &err);

        PowerMeter_PowerFactor = (CPU_FP32)cos(PowerMeter_PIdiv180 * (CPU_FP32)PowerMeter_PhaseAngle);

        sum_v2                 = (CPU_FP32)0.0;
        sum_i2                 = (CPU_FP32)0.0;
        sum_p                  = (CPU_FP32)0.0;
        for (i = 0; i < 360; i++) {
            current_phase_angle                     = (i + PowerMeter_PhaseAngle) % 360;
            PowerMeter_VoltageTbl[i]                = PowerMeter_VoltagePeak   * PowerMeter_SineTbl[i];
            PowerMeter_CurrentTbl[current_phase_angle] = PowerMeter_CurrentPeak   * PowerMeter_SineTbl[i];
            v2                                      = PowerMeter_VoltageTbl[i] * PowerMeter_VoltageTbl[i];
            i2                                      = PowerMeter_CurrentTbl[i] * PowerMeter_CurrentTbl[i];
            sum_v2                                 += v2;
            sum_i2                                 += i2;
            sum_p                                  += PowerMeter_VoltageTbl[i] * PowerMeter_CurrentTbl[i];
        }

        PowerMeter_VoltageRMS     = (CPU_FP32)sqrt(sum_v2 / (CPU_FP64)360.0);
        PowerMeter_CurrentRMS     = (CPU_FP32)sqrt(sum_i2 / (CPU_FP64)360.0);
        PowerMeter_PowerApparent  = PowerMeter_VoltageRMS * PowerMeter_CurrentRMS;
        PowerMeter_PowerActive    = sum_p  / (CPU_FP64)360.0;
    }
}
#endif
#endif
```

Debugging with Kernels
(Kernel Aware Debugger)

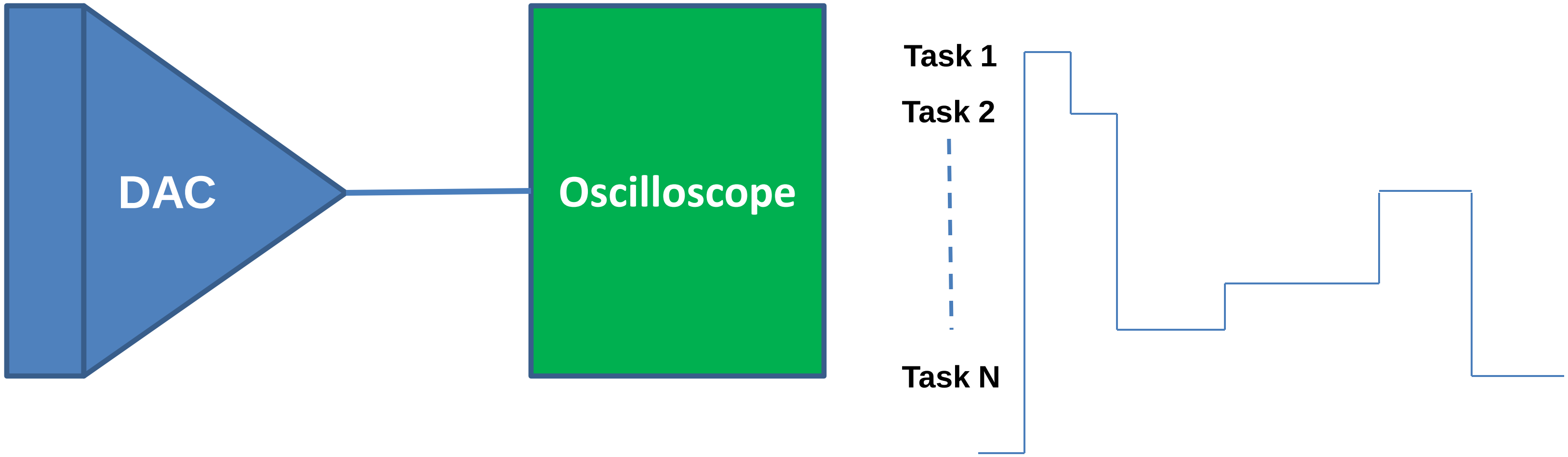


	#	Task Name	Prio...	State	Pending On Object	Pending On	CPU U...	Bar Graph	Context Swit...	Stack Poi...	Stack Size
	0	Temp Ctrl	8	Delayed			0.02%		93	0x20009FEC	80
	1	RPM Timer Reload	4	Delayed			0.01%		93	0x2000AEEC	80
	2	RPM	5	Pending with Timeout	Task Message Queue	Task Q	0.02%		169	0x2000AD8C	80
	3	Power Meter	8	Delayed			0.00%		61	0x2000A474	80
	4	ECG Waveform	8	Delayed			0.16%		932	0x2000A6C0	80
	5	Dimmer	8	Delayed			0.00%		46	0x2000A230	80
	6	Probe TCPIP	2	Pending	Semaphore	Net Sock Rx ...	0.00%		0	0x200076E4	300
	7	Net IF Tx Dealloc Task	5	Pending	Task Message Queue	Task Q	0.00%		0	0x20006058	100
	8	Net IF Rx Task	7	Pending	Task Message Queue	Task Q	0.00%		0	0x20005EC8	300
	9	Net Tmr Task	6	Delayed			0.13%		93	0x200097E0	150
	10	Start	2	Delayed			0.06%		93	0x20009D84	175
	11	uC/Probe-Term Trace Task	12	Delayed		Task Sem	0.03%		93	0x20005860	128
	12	uC/Probe-Term Cmd-Line Tx Task	11	Pending	Task Semaphore	Task Sem	0.00%		0	0x2000555C	128
	13	uC/Probe-Term Cmd-Line Rx Task	10	Delayed			0.01%		93	0x20005288	128
	14	uC/OS-III Timer Task	17	Pending	Task Semaphore	Task Sem	0.02%		93	0x2000B374	75
	15	uC/OS-III Stat Task	18	Delayed			0.35%		93	0x2000B12C	75
	16	uC/OS-III Tick Task	16	Pending	Task Semaphore	Task Sem	1.87%		9134	0x2000B268	75
➡	17	uC/OS-III Idle Task	19	Ready			97.28%	🟩	9243	0x2000BDBC	50

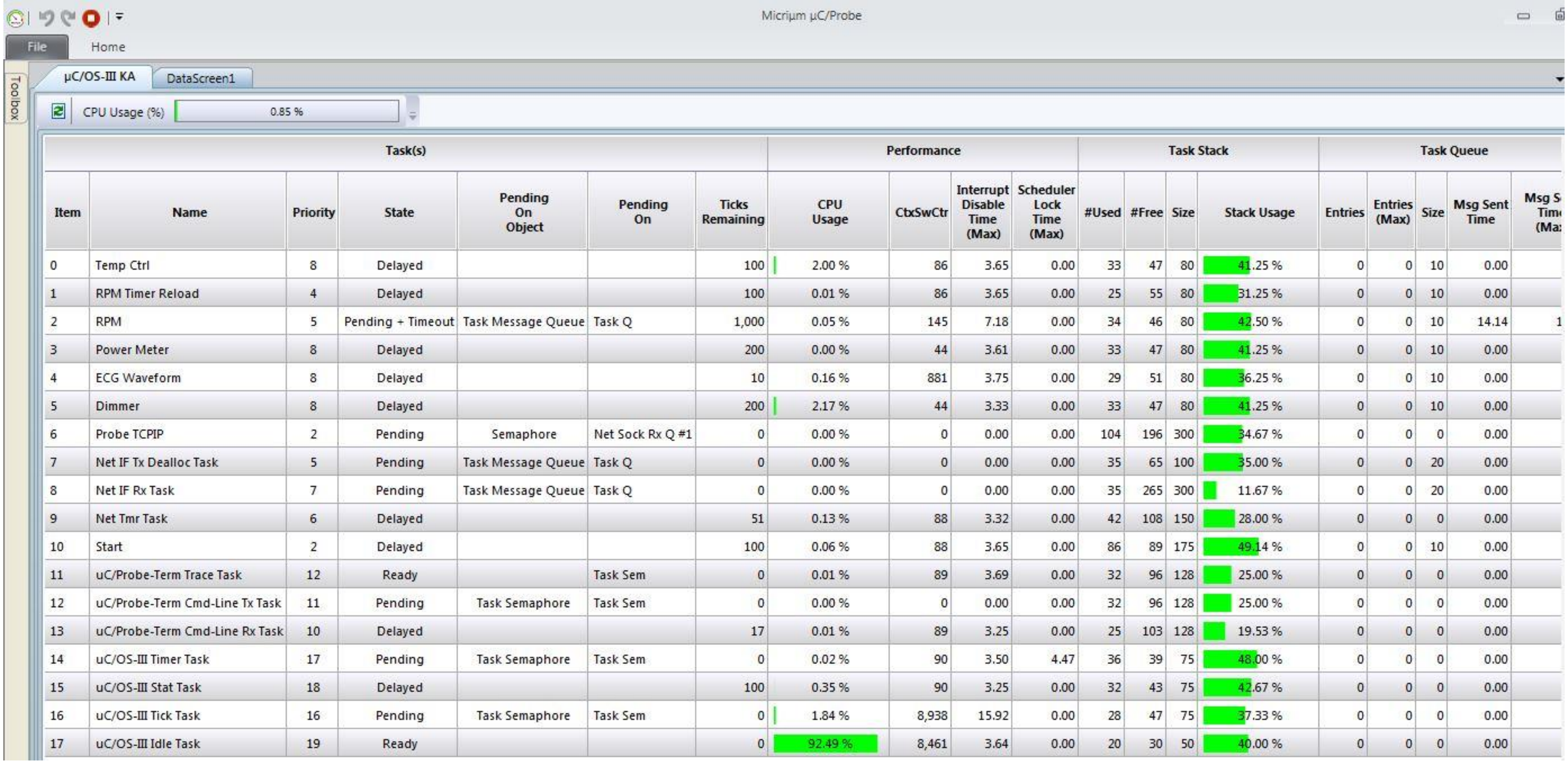
Debugging with a Kernel
(Output Port)



Debugging with a Kernel
(DAC Output)



Debugging with Kernels
(Run-Time Kernel Awareness)

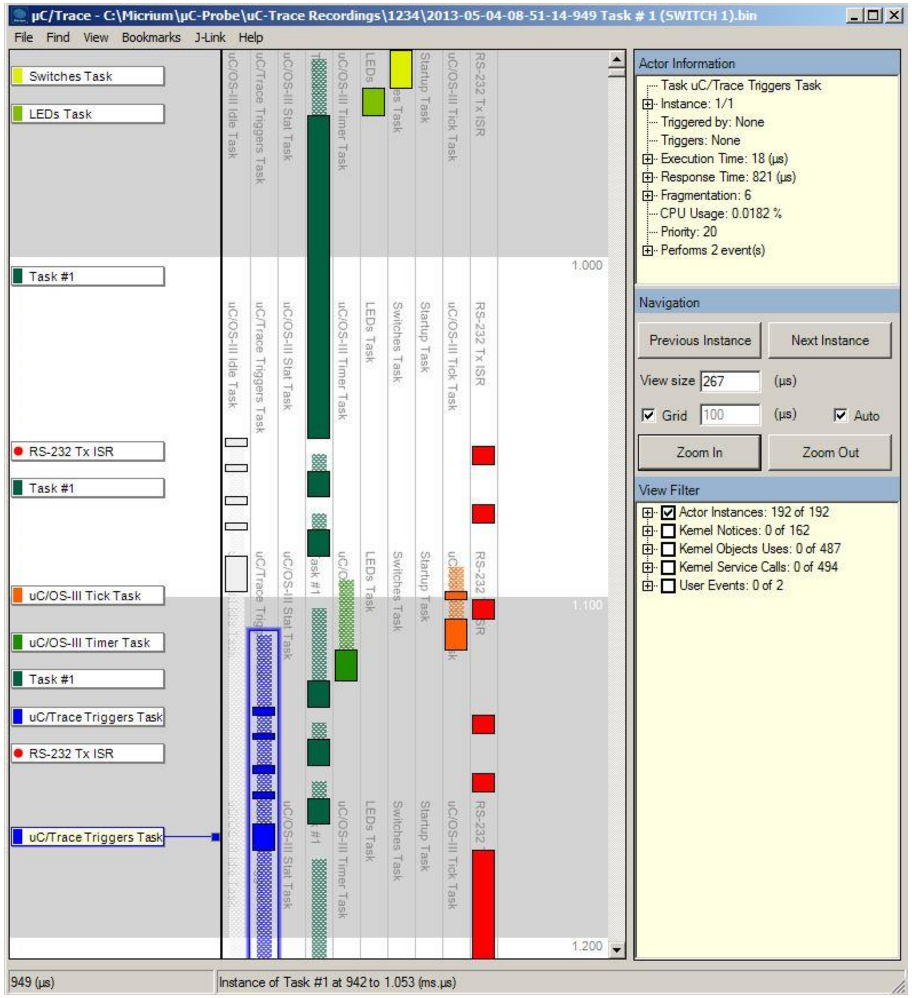


CPU Usage (%) 0.85 %

Task(s)							Performance				Task Stack				Task Queue				
Item	Name	Priority	State	Pending On Object	Pending On	Ticks Remaining	CPU Usage	CtxSwCtr	Interrupt Disable Time (Max)	Scheduler Lock Time (Max)	#Used	#Free	Size	Stack Usage	Entries	Entries (Max)	Size	Msg Sent Time	Msg S Tim (Ma
0	Temp Ctrl	8	Delayed			100	2.00 %	86	3.65	0.00	33	47	80	41.25 %	0	0	10	0.00	
1	RPM Timer Reload	4	Delayed			100	0.01 %	86	3.65	0.00	25	55	80	31.25 %	0	0	10	0.00	
2	RPM	5	Pending + Timeout	Task Message Queue	Task Q	1,000	0.05 %	145	7.18	0.00	34	46	80	42.50 %	0	0	10	14.14	1
3	Power Meter	8	Delayed			200	0.00 %	44	3.61	0.00	33	47	80	41.25 %	0	0	10	0.00	
4	ECG Waveform	8	Delayed			10	0.16 %	881	3.75	0.00	29	51	80	36.25 %	0	0	10	0.00	
5	Dimmer	8	Delayed			200	2.17 %	44	3.33	0.00	33	47	80	41.25 %	0	0	10	0.00	
6	Probe TCPIP	2	Pending	Semaphore	Net Sock Rx Q #1	0	0.00 %	0	0.00	0.00	104	196	300	34.67 %	0	0	0	0.00	
7	Net IF Tx Dealloc Task	5	Pending	Task Message Queue	Task Q	0	0.00 %	0	0.00	0.00	35	65	100	35.00 %	0	0	20	0.00	
8	Net IF Rx Task	7	Pending	Task Message Queue	Task Q	0	0.00 %	0	0.00	0.00	35	265	300	11.67 %	0	0	20	0.00	
9	Net Tmr Task	6	Delayed			51	0.13 %	88	3.32	0.00	42	108	150	28.00 %	0	0	0	0.00	
10	Start	2	Delayed			100	0.06 %	88	3.65	0.00	86	89	175	49.14 %	0	0	10	0.00	
11	uC/Probe-Term Trace Task	12	Ready		Task Sem	0	0.01 %	89	3.69	0.00	32	96	128	25.00 %	0	0	0	0.00	
12	uC/Probe-Term Cmd-Line Tx Task	11	Pending	Task Semaphore	Task Sem	0	0.00 %	0	0.00	0.00	32	96	128	25.00 %	0	0	0	0.00	
13	uC/Probe-Term Cmd-Line Rx Task	10	Delayed			17	0.01 %	89	3.25	0.00	25	103	128	19.53 %	0	0	0	0.00	
14	uC/OS-III Timer Task	17	Pending	Task Semaphore	Task Sem	0	0.02 %	90	3.50	4.47	36	39	75	48.00 %	0	0	0	0.00	
15	uC/OS-III Stat Task	18	Delayed			100	0.35 %	90	3.25	0.00	32	43	75	42.67 %	0	0	0	0.00	
16	uC/OS-III Tick Task	16	Pending	Task Semaphore	Task Sem	0	1.84 %	8,938	15.92	0.00	28	47	75	37.33 %	0	0	0	0.00	
17	uC/OS-III Idle Task	19	Ready			0	92.49 %	8,461	3.64	0.00	20	30	50	40.00 %	0	0	0	0.00	

Debugging with Kernels

(Kernel Trace Tool)



µC/Trace - C:\Micrium\µC-Probe\uC-Trace Recordings\1234\2013-05-04-08-51-14-949 Task # 1 (SWITCH 1).bin

Actor Information

- Task uC/Trace Triggers Task
- Instance: 1/1
- Triggered by: None
- Triggers: None
- Execution Time: 18 (µs)
- Response Time: 821 (µs)
- Fragmentation: 6
- CPU Usage: 0.0182 %
- Priority: 20
- Performs 2 event(s)

Navigation

Previous Instance | Next Instance

View size 267 (µs)

☑ Grid 100 (µs) ☑ Auto

Zoom In | Zoom Out

View Filter

- ☑ Actor Instances: 192 of 192
- □ Kernel Notices: 0 of 162
- □ Kernel Objects Uses: 0 of 487
- □ Kernel Service Calls: 0 of 494
- □ User Events: 0 of 2

949 (µs) | Instance of Task #1 at 942 to 1.053 (ms.µs)

Summary

- **A kernel is software that manages the time of a CPU**
  - A kernel is a ‘Subset’ of an RTOS
  - Allows multitasking – you split your application into ‘Tasks’
  - Each task is assigned a ‘Priority’
  - Provides services to your application
    - Semaphores, Queues, Timers, Time Management and so on
- **Most kernels are ‘Preemptive’**
  - The kernel will always run the highest-priority task that is ready-to-run
- **A Task is an Infinite Loop**
  - Each task needs to wait for an event to occur
  - Each task has its own stack, can access data and I/O devices

Summary

- **ISRs are more important than tasks**
  - ISRs can be kernel or non-kernel aware
- **Kernels typically require a Tick ISR**
  - Provides time delays and timeouts
  - This is NOT mandatory
- **Your application might share resources**
  - You need to protect those with Mutexes

Conclusion

Thank you for attending

Hope you found this class useful

www.micrium.com