Linux Application Development on TI Processors Using Linux-RT SDK

Agenda

- Key takeaways
- Introducing the Toggling GPIO Application to explore latency
- Linux kernel latency concepts with the GPIO application
- RT-patched Linux kernel latency of the GPIO application
- The Processor Linux-RT (PREEMPT) SDK
- Performance data on the patched Linux-RT TI kernel
- RT application development
- Summary of RT Linux within the TI SDK

Key Takeaways

- Linux-RT SDK is supported on AM335x, AM437x, and AM57x Sitara processors.
- The Linux-RT SDK kernel provides more determinism, but does not guarantee it.
- Recognize the effect of increasing processor load and thread latency between the non-RT and RT kernels.
- System design is critical; The developer must set latency targets and make specific efforts to determine whether the system design hits or misses the target latency.
- System analysis of throughput versus determinism is required.
- Know where to find the TI collateral for the Processor Linux-RT SDK

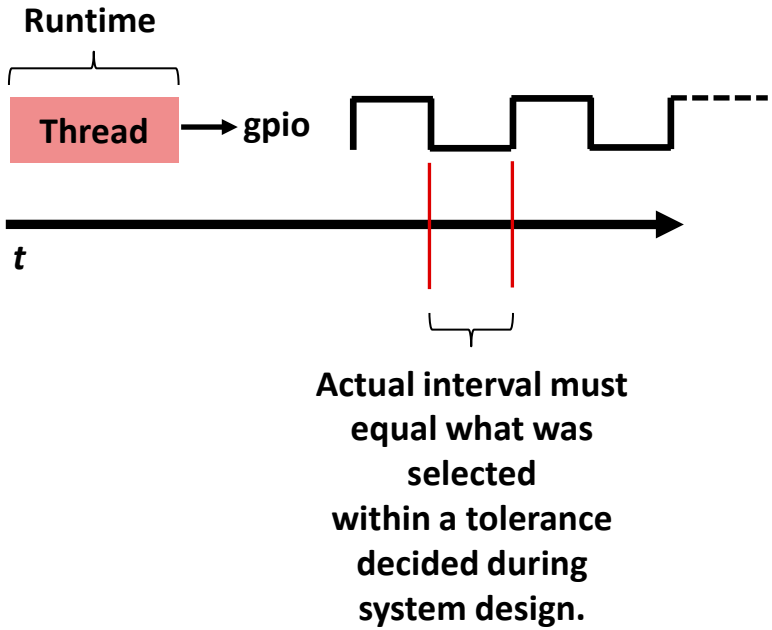
Linux Community RT Wiki
http://rt.wiki.kernel.org/index.php/Main_Page

Simple Application: Running a Single Thread that Toggles a GPIO

Linux Application Development on TI Processors Using Linux-RT SDK

Start with a Simple Thread Controlling a GPIO

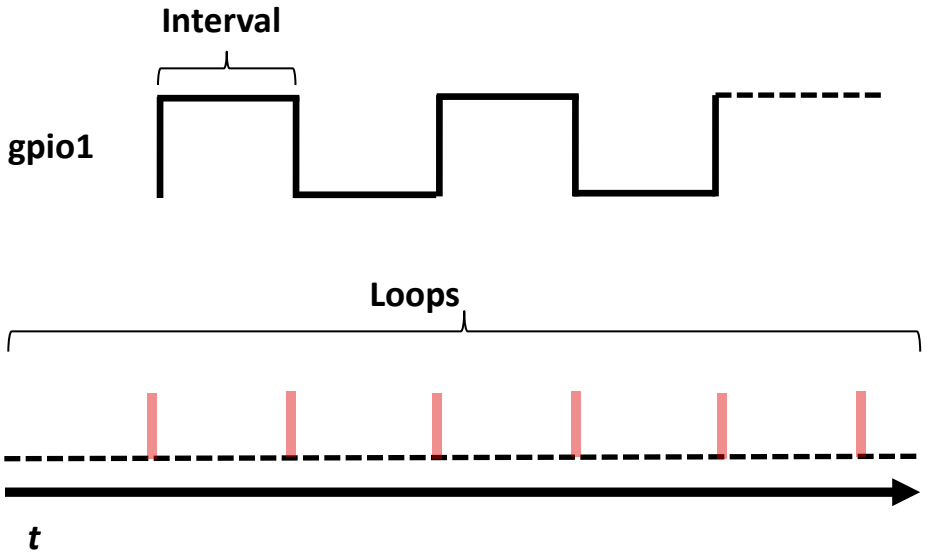
- For this investigation, consider a thread that toggles a GPIO in a POSIX thread at a 50% duty cycle.
- The requirement is that the GPIO must toggle within a tolerance of the interval selected. Otherwise, the application is considered to have failed.



Toggle GPIO Application: Main Loop

./app.out <gpio1> <gpio3> <interval> <max_latency> <loops> <priority>

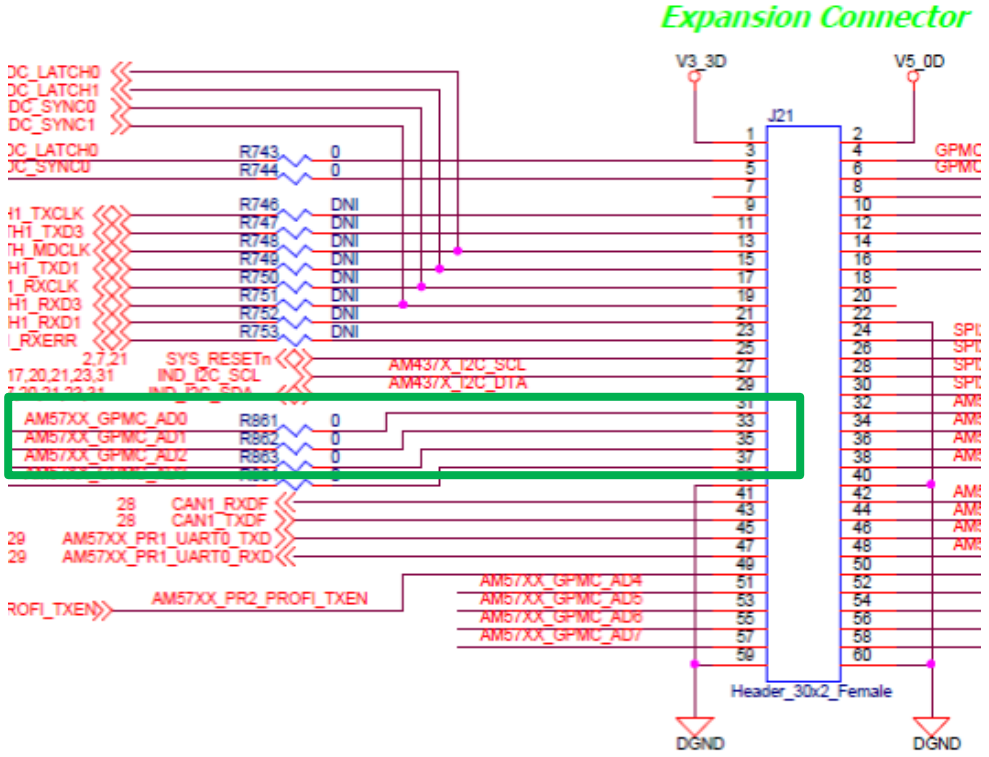
```
int main (int argc, char *argv[])
{
.
.
   for(i=0;i<loops;i++) {
      clock_nanosleep(,, &t,);
      clock_gettime(,&t_now);
      latency[i] = t_now.tv_nsec - t.tv_nsec;
      if ( i & 1) {
         write(fd_gpio1,"1",2);
      } else {
         write(fd_gpio1,"0",2);
      }
      t.tv_nsec+=interval;
.
.
}
```



Later slides show app.out = golt_rt.out

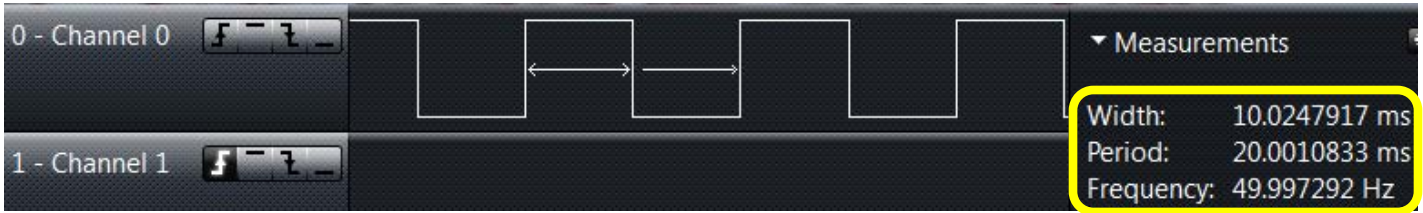
Experimentation Setup: AM572x IDK Rev 1.3A

- Added U-Boot Pin Mux definitions:
  - GPMC_AD0 (gpio1_6) – Output
  - GPMC_AD1 (gpio1_7) – Input
  - GPMC_AD2 (gpio1_8) – Output
- Added nodes to the AM572-IDK.DTS to cover GPIO-LED
- U-Boot Pin mux was updated



Non-RT Kernel: No Load almost 100% Idle

- Pre-built kernel from TI Processors SDK Linux AM57xx
- Run the GPIO application: “Channel 0” is gpio1



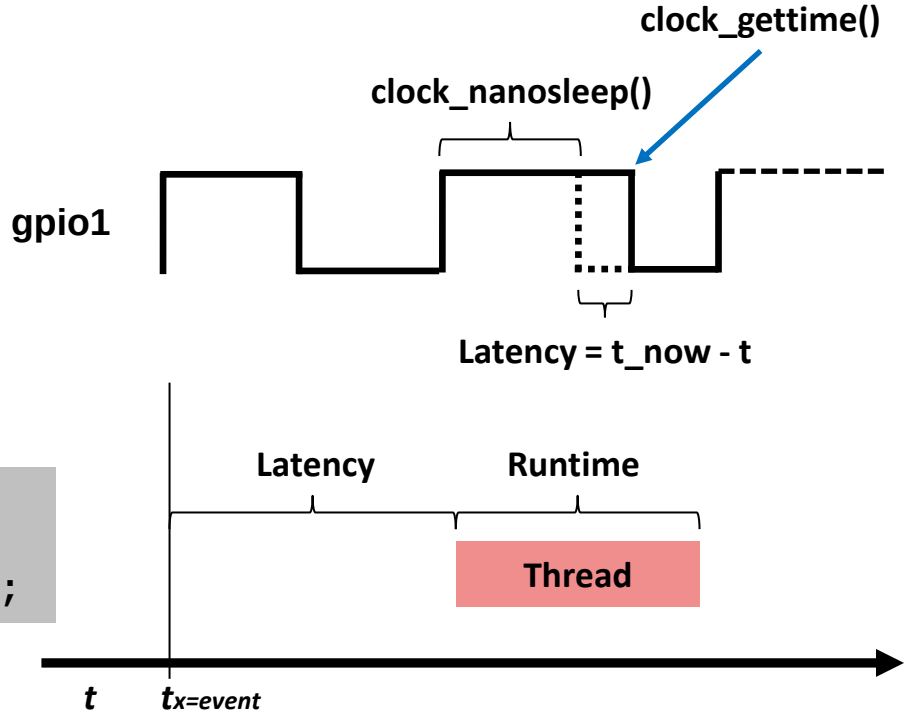
- Run with 10mS intervals
- Looks pretty good… so far….
- From a system design perspective, does it matter if the square wave produced is ever out of tolerance and does not have a 50/50 duty cycle?

Thread Latency

./app.out <gpio1> <gpio3> <interval> <max_latency> <loops> <priority>

- Thread latency is the difference in time between when a thread should have run and when it actually started to run.
- Is there a latency that negatively impacts the system?

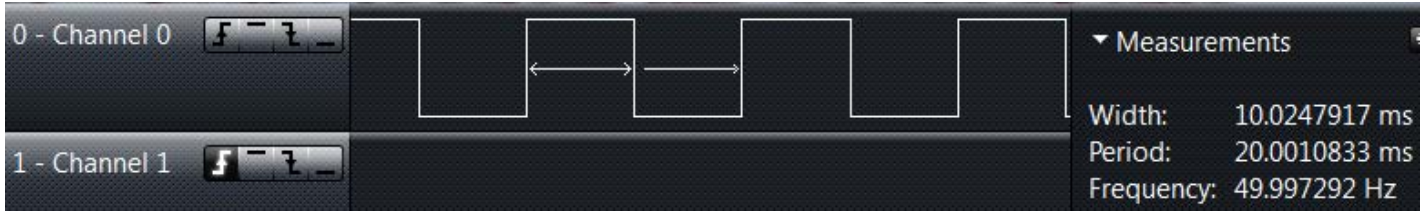
```
clock_nanosleep(,, &t, );
clock_gettime(,&t_now);
latency[i] = t_now.tv_nsec - t.tv_nsec;
```



Non-RT Kernel: Measuring Thread Latency

./app.out <gpio1> <gpio3> <interval> <max_latency> <loops> <priority>

- Run the GPIO application: “Channel 0” is gpio1, “Channel 1” is gpio3



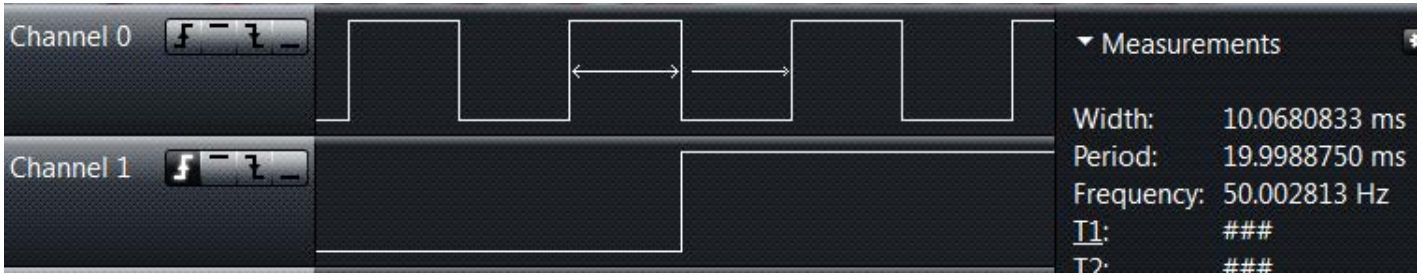
- Run with 10mS intervals, looking for > 1mS latency, start with 100 loops
- Code from application: If latency > max latency, application toggles gpio3 .

```
if (latency[i] > latency_max)
{
   write(fd_gpio3,“1",2);
} else {
   write(fd_gpio3,“0",2);
}
```

- No latency was detected by the application. So gpio3 did not toggle.

Non-RT Kernel: No Load almost 100% Idle

- 10mS interval, > 80uS latency in 100 Samples of clock_nanosleep




```
root@am57xx-evm:~# ./golt_rt.out 38 40 10000000 80000

Look through 100 samples looking for > latency max
80000

latency sample[  82] 108397
```

- Question: Is this result good enough for the system design?

Unloaded System is Unrealistic: Go for 0% Idle

- To achieve 0%, iperf was run along with cpuloadgen @ 100 % on both cores.
- There are probably other ways to load the ARM cores.

```
root@am57xx-evm:~# iperf -c 192.168.20.128 -t 7200 &
```

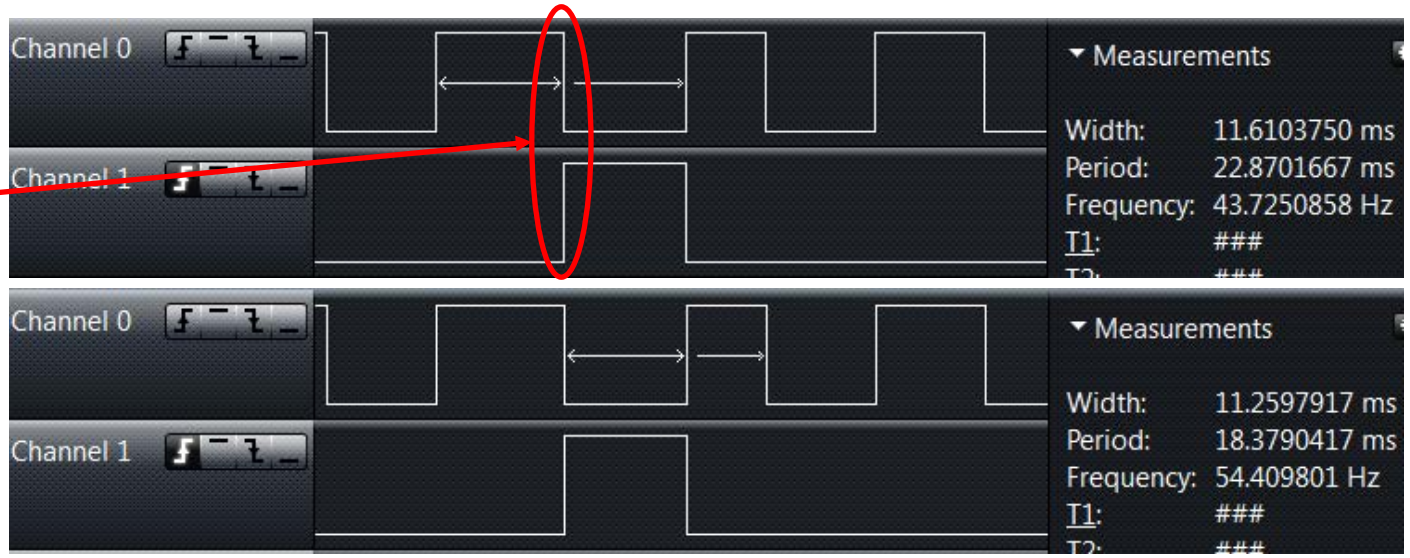
```
root@am57xx-evm:~# cpuloadgen 100 100 &
```

```
root@am57xx-evm:~# mpstat -P ALL 2
Linux 4.1.18-gcb4c79c797 (am57xx-evm)   04/02/16        _armv7l_        (2 CPU)
16:50:35     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest   %idle
16:50:37     all   40.70    0.00   14.32    0.00    0.00   44.97    0.00    0.00    0.00
16:50:37       0    8.04    0.00    2.01    0.00    0.00   89.95    0.00    0.00    0.00
16:50:37       1   73.37    0.00   26.63    0.00    0.00    0.00    0.00    0.00    0.00
```

Non-RT Kernel: Fully loaded No Idle

- 10mS interval, > 1mS max latency in 100 loops of clock_nanosleep

```
sample[ 6] 1664493
sample[ 7] 2926394
sample[15] 2713581
sample[23] 2736634
sample[32] 2173532
sample[33] 3309041
sample[48] 2441516
sample[64] 1814995
sample[65] 3090886
sample[73] 2916299
sample[81] 2654197
sample[89] 1472541
sample[90] 2553069
```

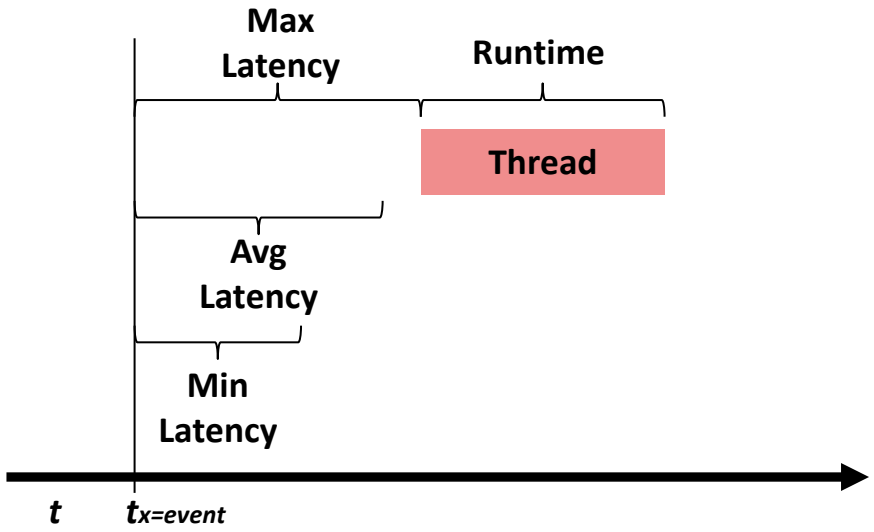


- Max latency target was missed 13 times in this particular run of just 100 loops.
- Not a positive development for the system design viability.

Thread Latency is Not Consistent Over Time

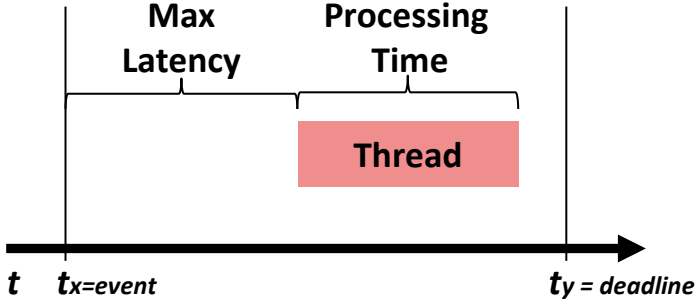
- Thread must be able to respond deterministically to an event despite system latency.
- Which latency would be good to design a system around?

```
sample[ 6] 1664493
sample[ 7] 2926394
sample[15] 2713581
sample[23] 2736634
sample[32] 2173532
sample[33] 3309041
sample[48] 2441516
sample[64] 1814995
sample[65] 3090886
sample[73] 2916299
sample[81] 2654197
sample[89] 1472541
sample[90] 2553069
```



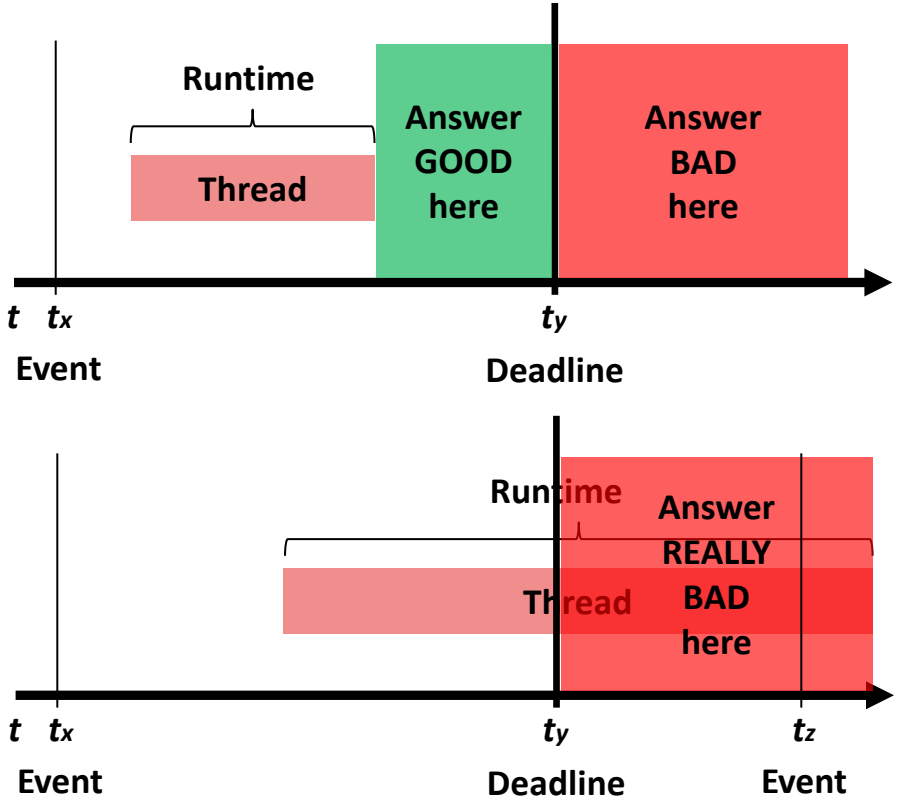
Max Thread Latency + Processing Time < Deadline

- This is a simplified diagram for discussion purposes.
- Deadline is the point in time by which a processor thread must have completed a task.
- Thread must be able to respond deterministically to an event despite system latency.
- The application that is missing target latencies in this example has a negligible processing time per loop.

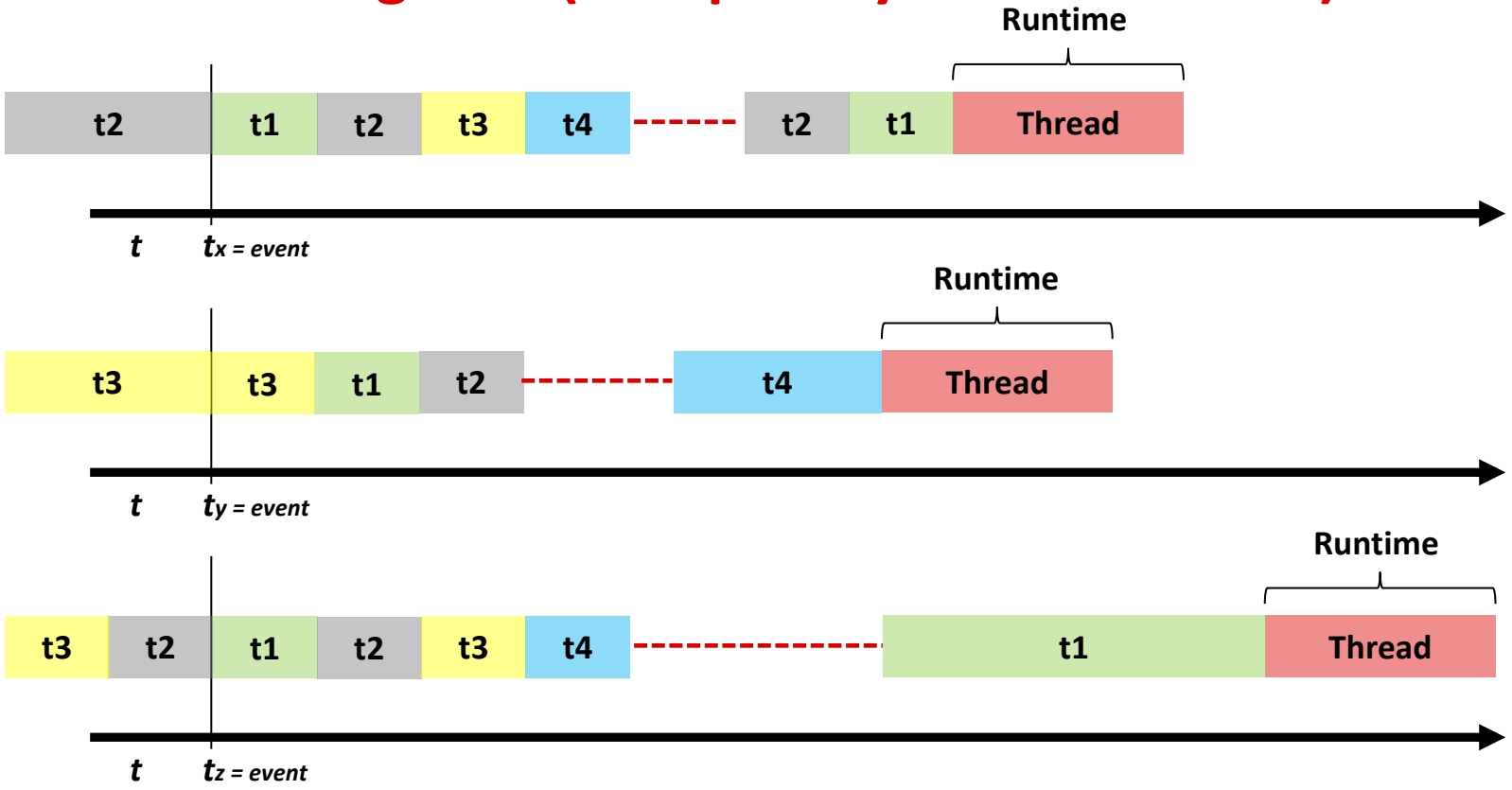


Responding to an Event by a Deadline

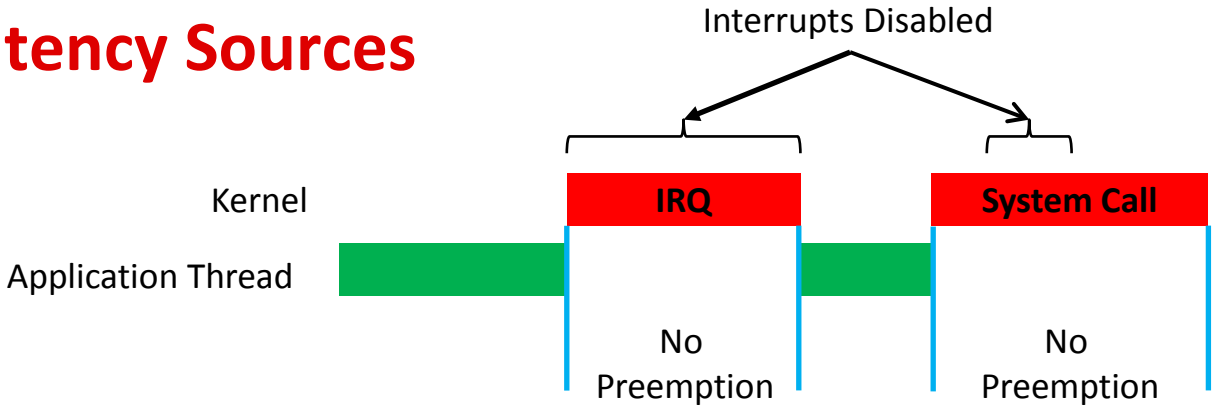
- A thread is waiting on an event to happen.
- The time period it takes for a thread to process an event is known.
- The time taken to start running the waiting thread is not known.
- Relative to the event there is an expiration date on the answer; Right answer at the right time.
- What happens when the next event shows up before the previous answer?



Linux Scheduling: CFS (Completely Fair Scheduler) * Simplified



Linux: Scheduling Latency Sources



- Linux kernel contexts that are non-preemptive:
  - IRQ Handling
  - System Calls
  - Kernel Threads
- Linux also disables interrupts in critical sections of code.
- During the period of interrupts being disabled and no preemption, portions of the kernel code undermine the deterministic performance of the system.

Section Summary

- What is the event/deadline relationship?
- When is the result needed?
- What is the impact of loading a processor on latency?
- What is the effect of the Completely Fair Scheduler (CFS)?
- What are the sources of scheduling latency?

GPIO Application Running on the RT Kernel and non-RT Kernel

Linux Application Development on TI Processors Using Linux-RT SDK

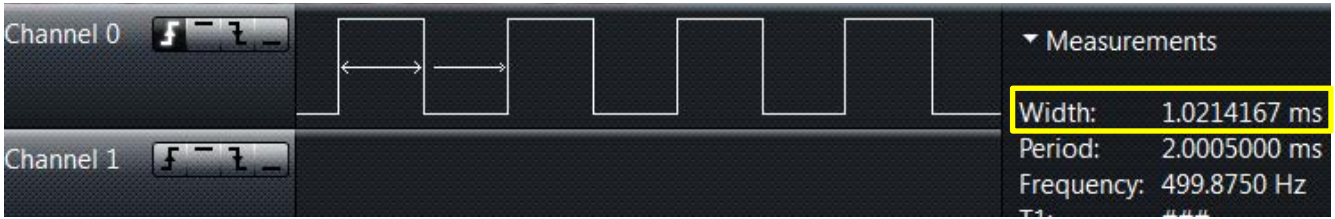
Section Overview

- Introduce the application using the priority parameter.
- Show the result of a test run that has a tighter interval and latency target.
- Introduce the FIFO scheduling class and how the priority is used.
- Discuss what happens if priority is used on the non-RT kernel.

RT Kernel: Fully Loaded 0% Idle

./app.out <gpio1> <gpio3> <interval> <max_latency> <loops> <priority>

- 1mS interval, > 70uS latency, 100K Samples, FIFO_SCHED with priority 75
- No missed deadlines



- How many loops should be run?

Toggle GPIO Application: Main Loop

./app.out <gpio1> <gpio3> <interval> <max_latency> <loops> <priority>

```
int main (int argc, char *argv[])
{
.
.
   param.sched_priority = atoi(argv[6]);
   if(sched_setscheduler(0, SCHED_FIFO,&param)==-1){
     perror("sched_setscheduler failed");
     exit(-1);
  }
.
   for(i=0;i<loops;i++) {
.
   sleep/toggle loop ...
.
}
```

Non-RT Kernel: Fully Loaded using Thread Priority

- The Non-RT SDK Kernel has preemption enabled within configuration. This means the RT scheduler is built into the kernel. However, the preemption is not full.
- Uses the same fully loaded 0% Idle scenario, except adding the number of missed latencies can be reduced.

```
root@am57xx-evm:~# ./rolt_rt.out 38 40 10000000 1000000 100
gpio1 38
pulse width = 10000000 nS latency trigger 1000000 loops 100
gpio1_path /sys/bus/platform/devices/leds/leds/gpio38/brightness
gpio3_path /sys/bus/platform/devices/leds/leds/gpio40/brightness
t.tv_sec 8005 t.tv_nsec 552319327

Look through 100 samples looking for > latency max 1000000

latency sample[  27] 2300095
latency sample[  42] 1791900
latency sample[  44] 1641137
latency sample[  45] 1072875
latency sample[  61] 1486283
latency sample[  77] 1109619
```

The application has a priority parameter that, if passed, invokes the RT scheduler. This can add latency determinism on the non-RT SDK.

```
root@am57xx-evm:~# ./rolt_rt.out 38 40 10000000 1000000 100 75
gpio1 38
pulse width = 10000000 nS latency trigger 1000000 loops 100
priority 75
gpio1_path /sys/bus/platform/devices/leds/leds/gpio38/brightness
gpio3_path /sys/bus/platform/devices/leds/leds/gpio40/brightness
t.tv_sec 8011 t.tv_nsec 393066707

Look through 100 samples looking for > latency max 1000000
```

GPIO App Summary Running on the Two Kernels

This simple application demonstrates how thread determinism is greatly improved with the TI SDK-RT Kernel.

- Non-RT Kernel:
  - 10mS interval, > 1mS max latency in 100 loops
  - 13 missed deadlines for this particular run
- Non-RT Kernel:
  - 10mS interval, > 1mS max latency in 100 loops, FIFO_SCHED with priority 75
  - No missed deadlines for this particular run
- RT Kernel:
  - 1mS interval, > 70uS latency, 100K Loops, FIFO_SCHED with priority 75
  - No missed deadlines in this example

Linux-RT SDK for AM3x/4x/5x

Linux Application Development on TI Processors Using Linux-RT SDK

RT-PREEMPT SDK Overview

- There are two separate Linux SDKs: RT and Non-RT
- The RT-PREEMPT Linux Kernel has the RT-Patch applied, tuned, and tested
- Same development eco-system: toolchain, filesystem, installation, etc,
- Same POSIX API; No special APIs for RT support
- Increases deterministic responses. But the tradeoff is performance lowering or degradation
- There are kernel configuration differences from the standard Processor SDK, including disabling power management, CPU governor, etc.

RT Patch: Impact on the Linux Kernel

- The RT Patch is applied to the Linux Kernel to make kernel-level operations pre-emptible.
- TI has tuned the SoC drivers for lower latency.
- The information block shown here is taken from the RT Wiki site and summarizes the effects of the RT Patch.

http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

The RT-Preempt patch converts Linux into a fully preemptible kernel. The magic is done with:

- Making in-kernel locking-primitives (using spinlocks) preemptible though reimplementation with rtmutexes:
- Critical sections protected by i.e. spinlock_t and rwlock_t are now preemptible. The creation of non-preemptible sections (in kernel) is still possible with raw_spinlock_t (same APIs like spinlock_t)
- Implementing priority inheritance for in-kernel mutexes, spinlocks and rw_semaphores. For more information on priority inversion and priority inheritance please consult Introduction to Priority Inversion
- Converting interrupt handlers into preemptible kernel threads: The RT-Preempt patch treats soft interrupt handlers in kernel thread context, which is represented by a task_struct like a common userspace process. However it is also possible to register an IRQ in kernel context.
- Converting the old Linux timer API into separate infrastructures for high resolution kernel timers plus one for timeouts, leading to userspace POSIX timers with high resolution.

Kernel Config Differences: Non-RT and RT-Linux SDKs

Consider the differences between the TI Processor SDK RT and non-RT Kernel configurations.

Removed from SDK RT Kernel configuration:

- CPU Freq/Performance gov
- CPU Idle/PM/Suspend/Sleep
- Thermal (BYOF,bring your own fan...☺)
- BT
- Oprofile
- KEXEC
- Hotplug
- Transparent Hugepage

Added from SDK RT Kernel configuration:

- Preempt RT Full

- Note that Preempt is enabled in the non-RT SDK.

Processor OPPs for RT-Linux AM57x

- cpufreq, performance governors are not enabled on the PREEMPT_RT kernels
- The highlighted entries on the table shown indicate the maximum frequency of the SoC for the Linux Kernel that is set by U-boot.
- If a higher frequency is desired, additional work by the end user is required.

ONLY OPP

Table 5-8. Supported OPP vs Max Frequency (2)

DESCRIPTION	OPP_NOM	OPP_OD	OPP_HIGH
	Max Freq. (MHz)	Max Freq. (MHz)	Max Freq. (MHz)
VD_MPU			
MPU_CLK	1000	1176	1500
VD_DSPEVE			
DSP_CLK	600	700	750
VD_IVA			
IVA_GCLK	388.3	430	532
VD_GPU			
GPU_CLK	425.6	500	532
VD_CORE			
CORE_IPUx_CLK	212.8	N/A	N/A
L3_CLK	266	N/A	N/A
DDR3 / DDR3L	532 (DDR3-1066)	N/A	N/A
VD_RTC			
RTC_FCLK	0.034	N/A	N/A

(1) N/A in this table stands for Not Applicable.

(2) Maximum supported frequency is limited according to the Device Speed Grade (see Table 5-5).

Performance Difference between Non-RT and RT Kernels

Using network performance as an example of possible throughput lowering

Non-RT Kernel

AM57XX TCP Performance

Ethernet Port0 TCP - 1000Mbps Mode Performance

TCP Window Size (in KBytes)	Bandwidth (without interrupt pacing, in Mbits/sec)	CPU Load (without interrupt pacing, in %)	Bandwidth (with interrupt pacing, in Mbits/sec)	CPU Load (with interrupt pacing, in %)
16	621.60	66.09	456.00	25.30
32	992.80	80.05	754.40	44.40
64	1082.80	87.50	1188.00	78.33
128	1024.00	70.80	1184.00	64.80
256	906.16	94.63	1169.60	61.42

RT Kernel

AM572X-IDK TCP Performance

Ethernet Port0 TCP - 1000Mbps Mode Performance

TCP Window Size (in KBytes)	Bandwidth (without interrupt pacing, in Mbits/sec)	CPU Load (without interrupt pacing, in %)	Bandwidth (with interrupt pacing, in Mbits/sec)	CPU Load (with interrupt pacing, in %)
16	412.00	88.00	700.00	90.30
32	414.40	87.00	754.40	93.40
64	425.00	87.50	781.00	90.33
128	443.00	88.80	844.00	93.80
256	523.00	86.32	873.00	94.00

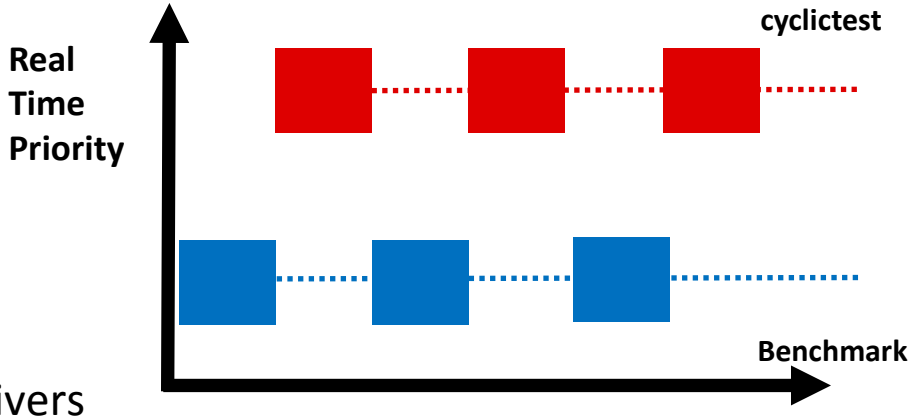
Latency Measurements on the RT-patched Linux TI Kernel

Linux Application Development on TI Processors Using Linux-RT SDK

Latency Measurement Overview

- The basic measurement tool for RT Linux is cyclictest.
- The example GPIO application is based on cyclictest with some tweaks.
- Look at how cyclictest proves the statistically maximum latency for a system.
- Compare cyclictest results between the RT and Non-RT kernel.
- Examine the system latency with different peripherals running on a loaded system

System RT Latency Tests

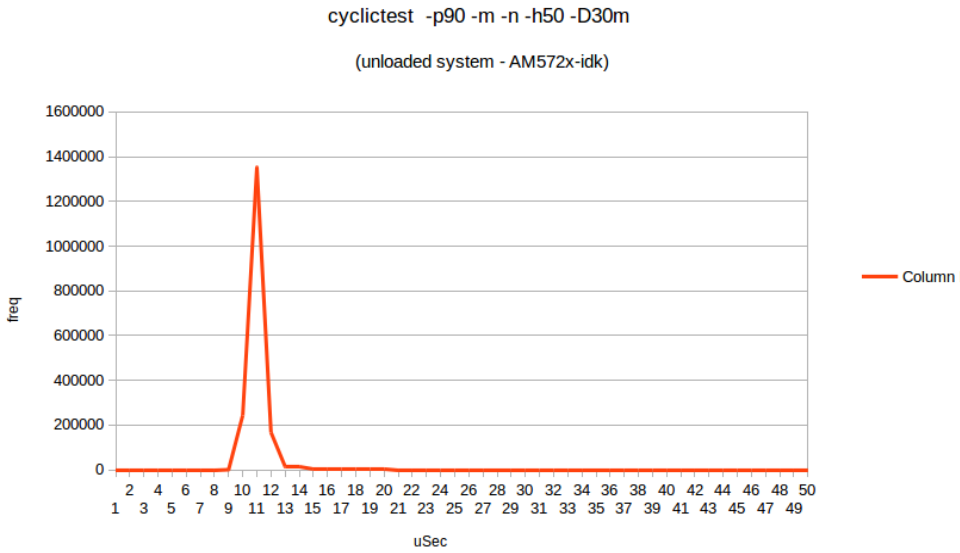


- The basic measurement tool for RT Linux is cyclictest.
- System test runs the cyclictest test at a higher priority than the subject benchmark test.
- These tests ensure that the underlying drivers are not impacting the overall system latency for a particular benchmark.
- The example GPIO application is based on cyclictest with some tweaks.
- If given enough time to run, cyclictest proves the statistically max latency for a system.

Cyclictest: Latency Analysis

```
root@am57xx-evm:~# cyclictest -p90 -m -n –h50 –D30m
# /dev/cpu_dma_latency set to 0us
WARN: Running on unknown kernel version...YMMV
T: 0 ( 1657) P:90 I:1000 C:1799998 Min:      8 Act:   10 Avg:    10 Max:      25
```

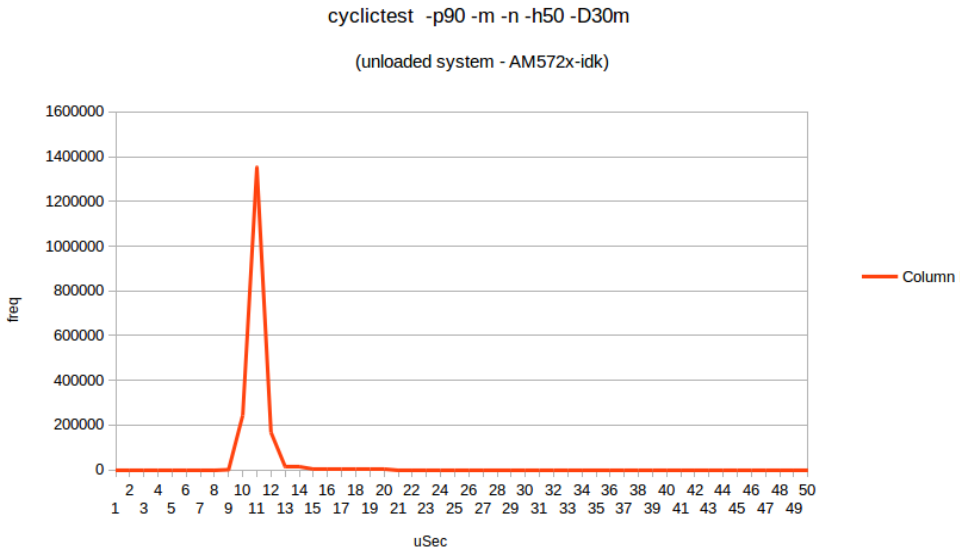
- The histogram output demonstrates where the outliers of excessive latency exist.
- This test is used to provide statistical data that the latency of a platform is below a target latency.
- NOTE: The results **do not guarantee** the latency measurement will always be below a target.
- The longer the test is run, the better the statistical data will be (e.g., run this test for months).



Cyclictest: Histogram Data

```
root@am57xx-evm:~# cyclictest  -p90 -m -n -h50 -D30m
# /dev/cpu_dma_latency set to 0us
WARN: Running on unknown kernel version...YMMV
policy: fifo: loadavg: 0.01 0.02 0.05 1/136 1781

T: 0 ( 1657) P:90 I:1000 C:1799998 Min:8 Act:10 Avg:10 Max:25
# Histogram

000007 000000
000008 000244
000009 243188
000010 1352108
000011 167423
000012 016242
000013 013071
000014 001712
000015 001307
000016 000819
000017 000586
000018 002389
000019 000658
000020 000083
000021 000122
000022 000037
000023 000008
000024 000002
000025 000001
000026 000000
```

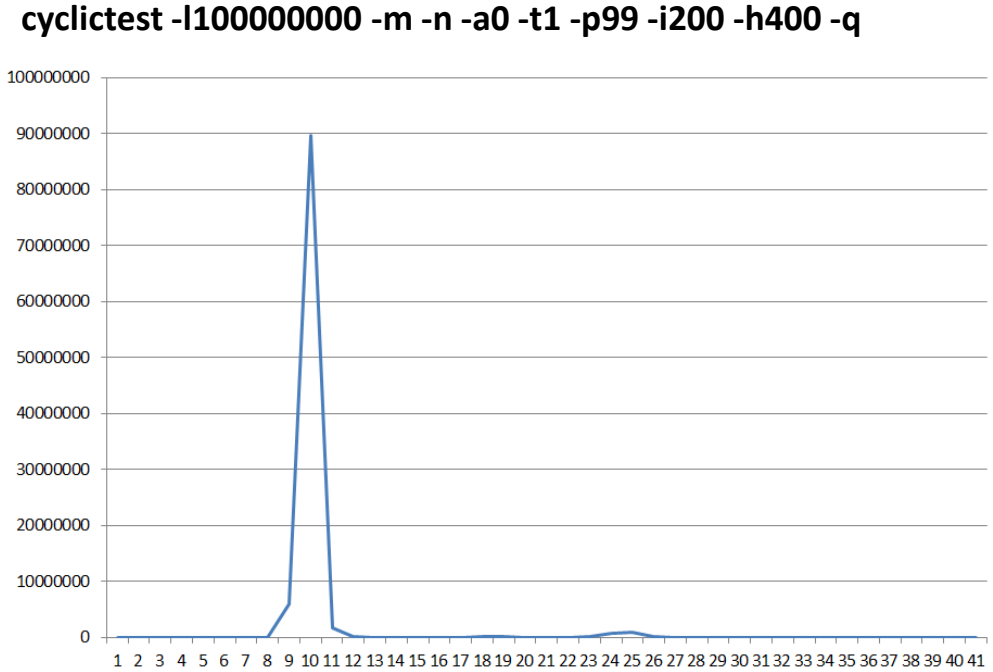
- To find frequency of latency outliers, review the histogram data from cyclictest.
- System is meeting latency target when there are no outliers beyond the tolerance specified by the system design.



OSADL: Cyclictest Benchmark

Total: 100000000
Min Latencies: 00008
Avg Latencies: 00009
Max Latencies: 00038
Histogram Overflows: 00000

- Latency test used by Open Source Automation Development Lab: http://www.osadl.org
- Graph taken from the RT SDK performance datasheet.



http://ap-fpdsp-swapps.dal.design.ti.com/index.php/Sdk_2015.03_(4.1-rt)

Latency Improvement over Non-RT Kernel

- This block represents a test combining iperf and cyclictest on a pre-built non-RT kernel from the Processor SDK. (NOTE: cyclictest output simplified to fit onto slide)

```
root@am57xx-evm:~# iperf -c  192.168.20.128 -d -t 2400 &

cyclictest -p98 -m -n -q -D 10m
# /dev/cpu_dma_latency set to 0us
WARN: Running on unknown kernel version...YMMV
T: 0 ( 1057) P:98 I:1000 C: 600000 Min:      7 Act:   16 Avg: 29 Max:    1019
```

- This block represents the same test, but demonstrates the much lower Avg and Max latencies from the Linux-RT Processor SDK.

```
root@am57xx-evm:~#  iperf -c  192.168.20.128 -d -t 2400 &

root@am57xx-evm:~# cyclictest -p98 -m -n -q -D 10m
# /dev/cpu_dma_latency set to 0us
WARN: Running on unknown kernel version...YMMV
T: 0 ( 1099) P:98 I:1000 C: 600000 Min:      8 Act:   17 Avg: 17 Max:      61
```

Latest AM572x-idk RT Latency Test Results

- Worst-case latency with various loads on isolated core by using cyclictest.
- Measure latency under various loads on isolated core using cyclictest.

```
opt/ltp/runtest/ddt/
.
realtime_net_load
realtime_net_load-smp
.
.
```

Max Latency for shielded use cases

Use Case	Core-SDK-2015.03
	am572x-idk
	Latency
L_PERF_SHIELD_SMP_1080ENC	33.0
L_PERF_SHIELD_SMP_1080ENCDEC	45.0
L_PERF_SHIELD_SMP_GRAPHICS	35.0
L_PERF_SHIELD_SMP_HACKBENCH	31.0
L_PERF_SHIELD_SMP_MULTIMEDIA	47.0
L_PERF_SHIELD_SMP_NET	24.0
L_PERF_SHIELD_SMP_STRESS_LOAD	34.0
L_PERF_SHIELD_SMP_UART	39.0
L_PERF_SHIELD_SMP_USB	47.0
L_PERF_SHIELD_SMP_V4L2CAP	35.0

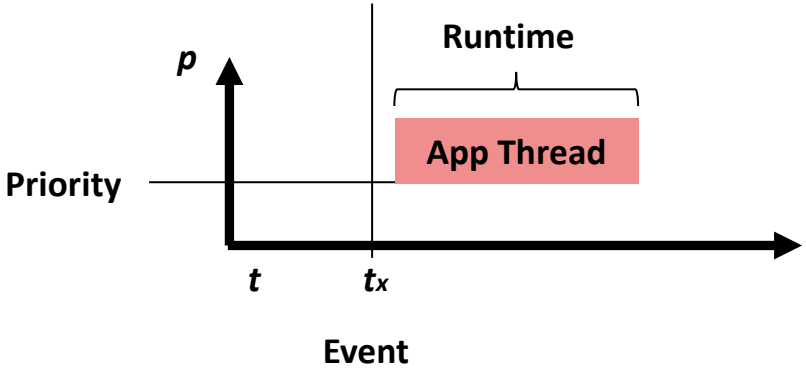
http://ap-fpdsp-swapps.dal.design.ti.com/index.php/Sdk_2015.03_(4.1-rt)

RT Application Development

Linux Application Development on TI Processors Using Linux-RT SDK

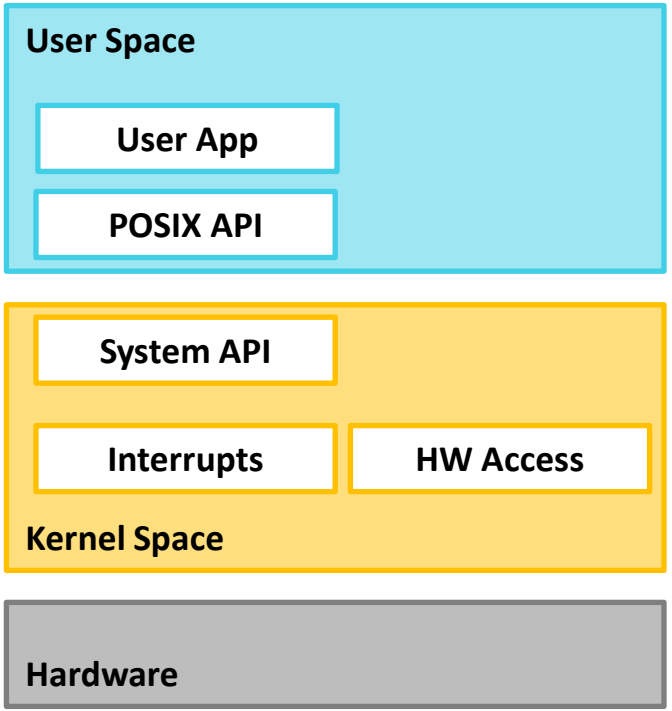
RT Application: Basic Questions

- Is there a specific API set for RT?
- How do you build it?
- Is there a basic structure to a RT App?
- How is the app scheduled to run?
- How is the priority set?



RT Application: API, Basic Structure, Background

- Basic Linux application rules are the same; Use the POSIX API.
- There is still a division of Kernel Space and User Space.
- Linux applications run in User Space .



RT Application: How do you build it?

- GCC-based cross-compiler toolchain; The one that comes with the TI Processors Linux-RT SDK.
- Link with the toolchain RT library'
- If experimenting on the target EVM using a TI Processors file system, use GCC natively.
- Using the cross compiler example:

```
~$ arm-linux-gnueabihf-gcc <filename>.c -o <filename>.out -lrt –Wall
```

- Using the native compiler on a target example:

```
~$ gcc <filename>.c -o <filename>.out -lrt –Wall
```

RT Application: API, Basic Structure, Hello World

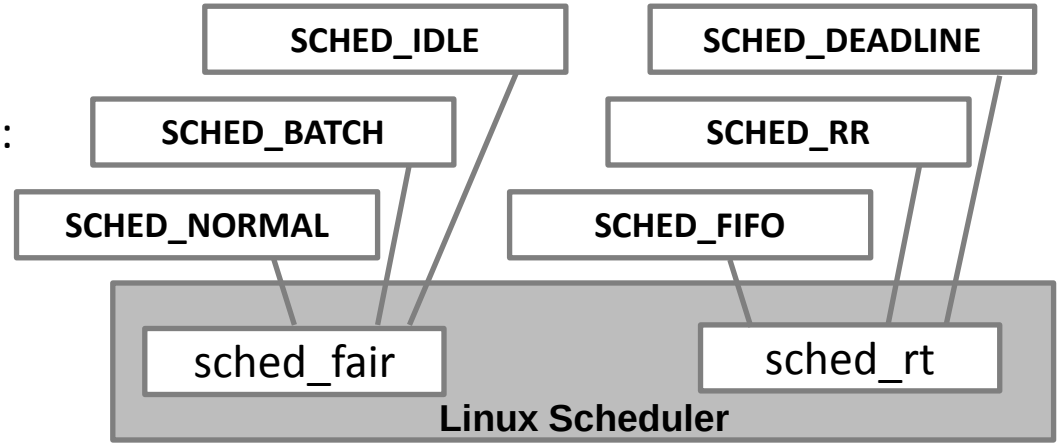
- Example app from the RT wiki, simplified for discussion here.
- POSIX API are used; Not any different than non-Linux RT Apps.
- Note how the priority is set with sched_setscheduler()
- Memory pages are locked into RAM with mlockall()
- This app is scheduled to run after the interval passed to clock_nanosleep expires.

```
int main(int argc, char* argv[]) {
   struct timespec t; struct sched_param param;
   int interval = 50000; /* 50us*/
   param.sched_priority = 49;

   if(sched_setscheduler(0, SCHED_FIFO, &param) == -1) {
     perror("sched_setscheduler failed");
     exit(-1);
   }
   if(mlockall(MCL_CURRENT|MCL_FUTURE) == -1) {
   perror("mlockall failed");
   }
   /* Pre-fault our stack, and get initial time */
   while(1) {
      clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &t, NULL);
      /* do the stuff */
      /* calculate next shot */
      t.tv_nsec += interval;
    }
   }
}
```

http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

Linux Scheduling Policies

- Each thread has an associated scheduling policy.
- Scheduling policies have two classes:
- Completely Fair Scheduling (CFS) policies:
  - SCHED_NORMAL
  - SCHED_BATCH
  - SCHED_IDLE
- RT policies:
  - SCHED_FIFO
  - SCHED_RR
  - SCHED_DEADLINE

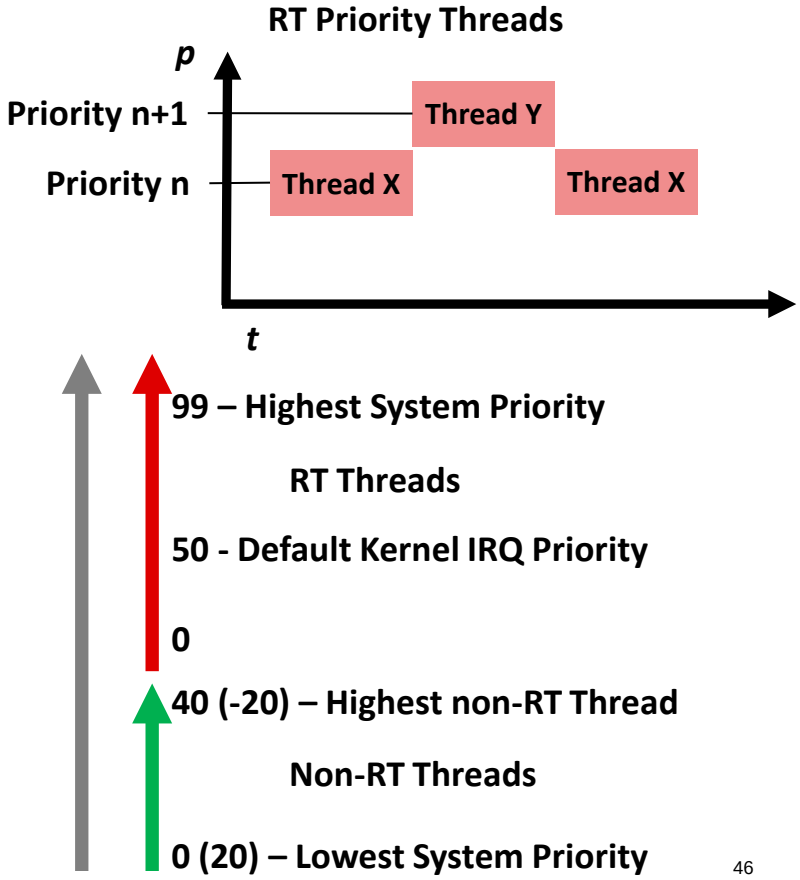


- Scheduling policy can be changed using API, either within or external to the thread.

```
if(sched_setscheduler(0, SCHED_FIFO, &param) == -1) {
     perror("sched_setscheduler failed");
     exit(-1);
   }
```

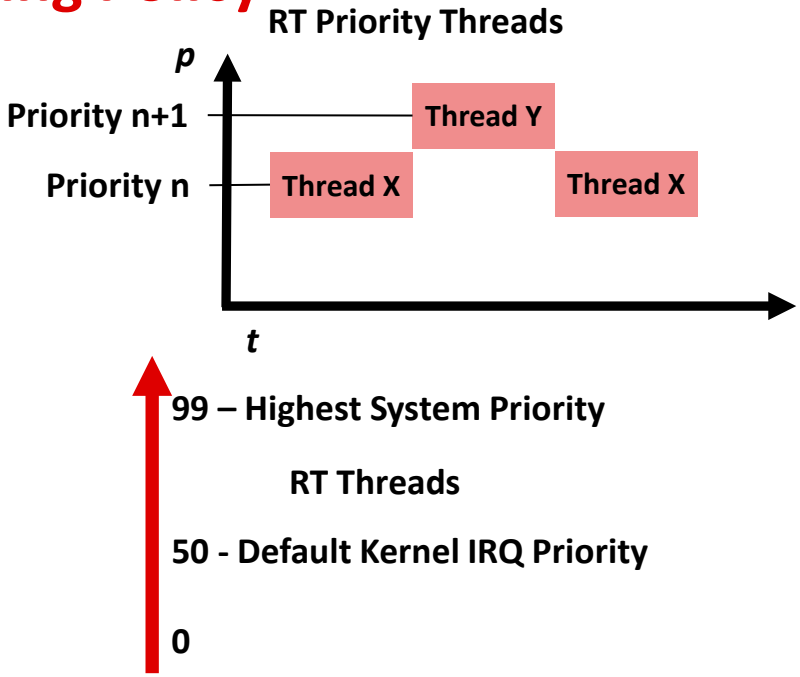
RT Application: Thread Priority Levels

- 0-99 are RT priority threads, SCHED_FIFO and SCHED_RR policies
- -20 to 20 are non-RT threads (statiscally set to 0)
- In the diagram shown:
  - Thread X is running at Priority n.
  - Thread X is preempted by Thread Y.
  - Upon completion of Thread Y, execution returns to Thread x until finished.
- Threads allocated by the Kernel start at priority 50.



RT Application: SCHED_FIFO Scheduling Policy

- FIFO (First In, First Out): Once it starts, the thread runs to completion or to a yield point, unless preempted by a higher thread.
- In the diagram shown:
  - Thread X is running at Priority n.
  - Thread X is preempted by Thread Y.
  - Upon completion of Thread Y, execution returns to Thread x until finished.
- In this example, it does not matter if Thread X was scheduled as SCHED_FIFO. Thread Y was higher priority and therefore preempted Thread X.
- Be careful in assigning priority to this thread, as it can be set to run higher than kernel threads.
- Is it possible to hang the processor?



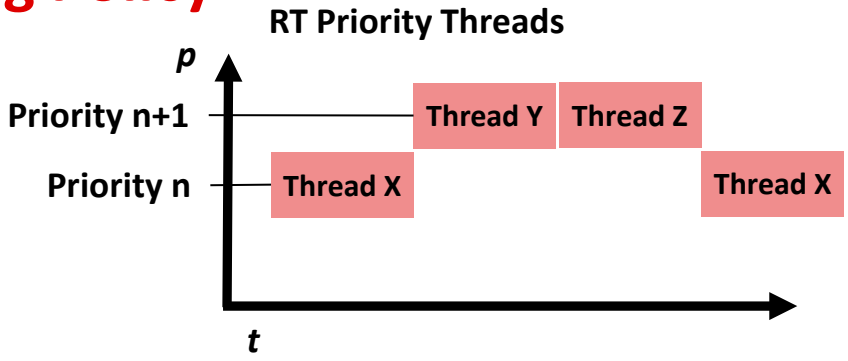
RT Application: SCHED_FIFO Scheduling Policy

- Code example from Hello World
- The SCHED_FIFO policy runs until preempted or yields.
- If the priority is high enough, it does not yield and other threads could be starved.

```
int main(int argc, char* argv[]) {
   struct timespec t; struct sched_param param;

/* important, running above kernel RT or below? */
param.sched_priority = ???;

if(sched_setscheduler(0, SCHED_FIFO, &param) == -1) {
     perror("sched_setscheduler failed");
     exit(-1);
   }

while(1) {
/* do the stuff */

/* Thread yield points */

/* watch for APIs that block */

}
```

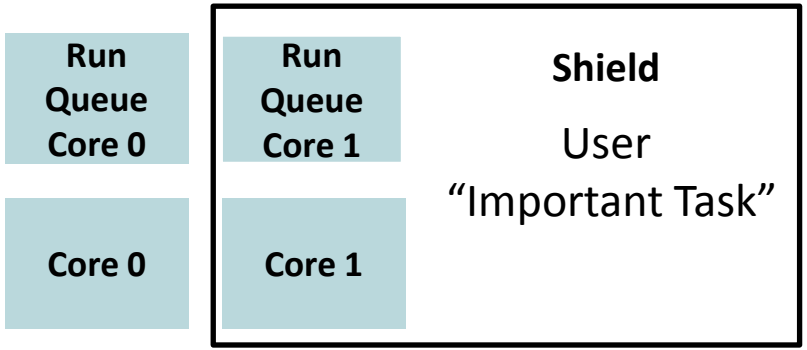
RT Application: SCHED_RR Scheduling Policy

- SCHED_RR (Schedule Round Robin): Once the thread starts, it runs for the time quanta designated or to a yield point, unless preempted by a higher thread.
- In the diagram shown:
  - Thread X is running at Priority n.
  - Thread X is preempted by Thread Y with Priority n+1.
  - Thread Y runs until one of the following is met:
    - The time quanta is exhausted
    - The thread completes
    - It yields to the next thread that is scheduled; In the case, Thread Z.
  - Upon completion of Thread Z, execution returns to Thread X until finished, assuming there are no other higher-priority threads.



Scheduling Multicore CPU Affinity

- On ARM Cortex-A15 devices, each processor has a separate, independently-run queue. Threads are scheduled per CPU.
- Higher priority threads on one CPU will not preempt lower priority threads on another CPU.
- Typically, Core 0 is used for interrupt processing.
- Key Tip: Lock down a CPU for the thread with shielding.



RT-Linux Application Development Summary

- Differences between writing a non-RT application from an RT application:
  - Scheduling Policy
  - Priority
  - Memory Control
- RT applications require tuning to the system.

It may appear that the RT Patch fixes all that is wrong as a single step for improving a casually-written application and system. RT Linux is NOT going to avoid or prevent the upfront work of a performing a detailed problem requirement analysis of the application and system priorities; In fact, the opposite is true.

Presentation Summary

- **Processor SDK with the RT Patch increases but does not guarantee deterministic behavior.**
- Linux-RT SDK is available now on AM3x/AM4x/AM5x.
- Recognize the effect of increasing processor load and thread latency between the non-RT and RT kernels.
- System design is critical; The developer must set latency targets and make specific efforts to determine whether the system design hits or misses the target latency.
- System analysis of throughput versus determinism is required.

For More Information

- Linux Community RT Wiki http://rt.wiki.kernel.org/index.php/Main_Page

- For questions regarding topics covered in this training, visit the support forums at the TI E2E Community website: http://e2e.ti.com