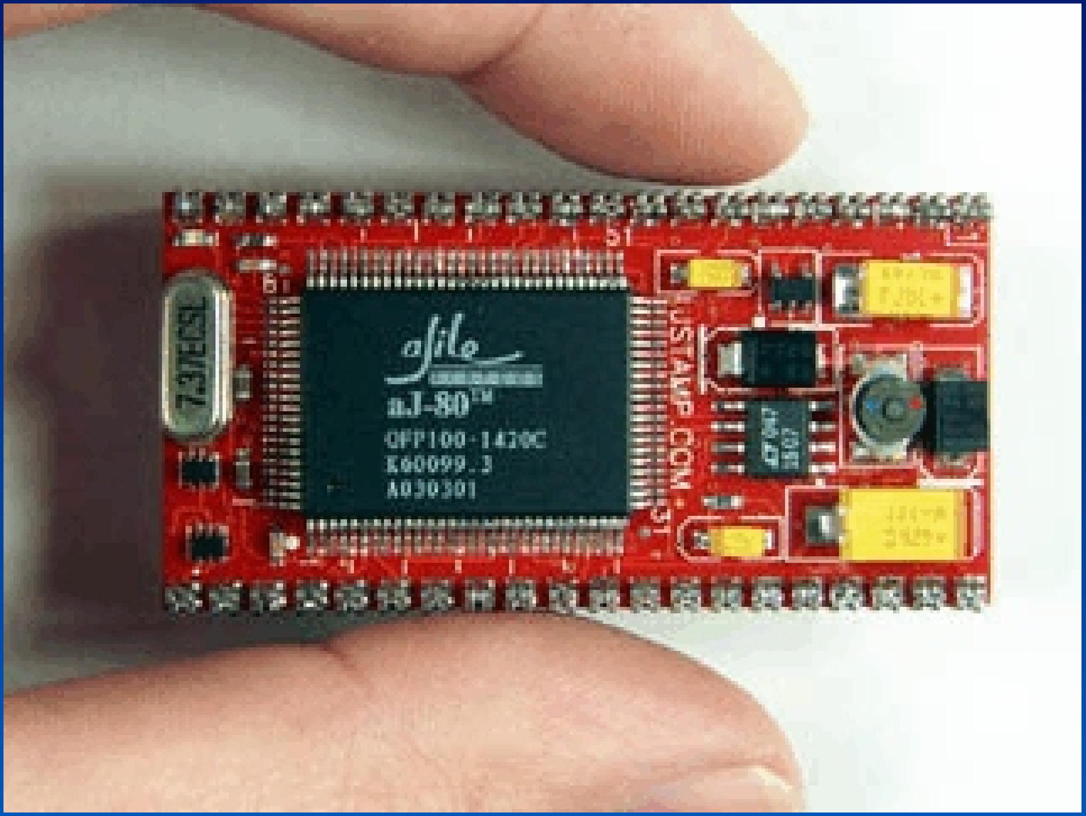
JStamp Periodic Threads

Using real-time periodic threads on JStamp/JStik/SaJe



Periodic Threads are not portable

real-time behavior is highly implementation-dependent

- aJile classes PianoRoll and PeriodicThread may not have equivalent classes on other JVMs
- With JStamp given a performance index of 1, JStik is about 5 and SaJe about 6. So there is no way all these platforms can execute the same code in the same amount of time or meet the same task deadline. This means periodic thread code may not be portable.
- Other JVMs' thread management may be completely different

The big picture

PianoRoll is a list of scheduled PeriodicThreads, like a player piano roll

Imagine a circular list with entries [0..n] and which wraps around, so that entry '0' appears again as the next entry after entry 'n'. There is a potential entry at each "beat" interval. 'N' Is the "duration" of the piano roll. For example if the beat is 1 msec and duration is 100 msec, there are 100 possible entries.

Now imagine a group of PeriodicThreads PT[A,B..Z] which each have a desired period of execution. PTA with a 2 msec period would have table entries every 2 beats. PTB with 10 msec period would have entries every 10 beats. Note the period must be an integer multiple of the beat.

PeriodicThreads can be placed in the PianoRoll table with a starting delay so that they are spaced out across the available entries.

Thread Priorities

- JStamp has 32 thread priorities
  - So does the RealTime Java spec...
  - 0 is low, 31 is high
- generic (not RTJ) Java has 10 priorities
  - The default Java level of 5 equates to the JStamp level of 10
  - 10 Java priorities are mapped to JStamp even priorities from 2 to 20
- JStamp periodic threads use more (& odd) priorities
  - setPriority(5) becomes internal priority of (5*2)+1 = 11
  - you can set priorities higher than 10
  - periodic threads can pre-empt the GC
    - set priority to 12, this becomes 25 internally (GC is 24)
    - Don't generate garbage in such a thread or you will run out of memory!
    - 25 is also the TCP/IP thread priority on JStik and SaJe (not JStamp)

System Threads

- Some threads setup by the JStamp runtime system [UC]:
  - Executive thread (priority 31) - Interrupts always disabled.
  - Serial thread for unloading internal UART FIFOs (priority 30).
    - Only present if serial I/O classes are used.
  - TCP/IP support threads (priority 25)
    - Only present if networking classes are used (so never on JStamp)
  - GC thread (priority 24)
  - Heap monitor threads (priority 12)
  - Idle thread (priority -1) - This priority is special and not available to users.

Threads vs Interrupts

Thread and interrupt priorities are completely independent

- There's no relationship between Thread and Interrupt priorities
- Any interrupt can interrupt any thread -- an arbitrarily high thread priority does not block interrupts.
  - It's easy to get this confused and think that a thread of priority 'n' will block an interrupt of lower priority < 'n'. It can't.
    - The only exception to this is if a thread disables interrupts

PeriodicThread Class

public class PeriodicThread
extends java.lang.Thread

- makePeriodic() sets up a periodic thread, given:
  - desired period in msec or msec and nsec
    - on JStik this should be 100 usec or greater
    - on JStamp this should be 500 usec or greater
  - initial delay before starting (can be zero)
  - desired priority
  - the Thread Object you wish to make Periodic
- run(), start() etc are inherited from Thread class
- cycle()
  - causes the thread to sleep not for a specified time, like wait(), but for a period of time since the last time it returned from cycle(). In other words, the remainder of its period

PeriodicThread Priority

PeriodicThreads can interact in complex ways

- PeriodicThreads with different priorities
  - can pre-empt other PeriodicThreads of lower priority
  - Pre-emption only can occur at the PianoRoll beat time
  - A currently active thread yields to other threads when it executes cycle()
- Periodic threads of equal priority
  - are round-robinned, if their periods and initial delays are equal
  - can be “sequenced”, one thread per PianoRoll beat, if each thread’s initial delay is different by one beat time. See the example which follows
  - can get complicated if periods are different, especially if they are multiples of each other

PeriodicThread cautions (1/3)

at the moment it's a simple construct with little error handling

- Consider PeriodicThread parameters carefully
  - period, initial delay, and priority interact with the PianoRoll beat to determine if a given PeriodicThread will ever pre-empt or round-robin with another PeriodicThread.
- A PeriodicThread should have less than its period's worth of tasks to perform
  - otherwise you may get cycle slippage with no indication of it. There is no deadline parameter or metric to alert you to this.
  - A much shorter task than the PT period is OK
- You can pause or end a PeriodicThread
  - do this yourself in your run() method
  - terminated PeriodicThreads will be GC'ed (they are Objects)

PeriodicThread cautions (2/3)

Periodic Threads of the same priority and period can be round-robinned

- Pick a time granularity or Quantum Q
  - This should be the smallest useful time interval for your system. On JStamp, this should be no smaller than 500 usec.
- Create a PianoRoll with a beat of B and duration D
  - B is a positive integer multiple of Q
  - B must be less than 65535 microseconds
  - D must be a positive integer multiple of B, and greater than the (period + initial delay) of all threads. It's simplest to make D the Least Common Multiple of both B and (2 x longest PeriodicThread period)
- Create T PeriodicThreads
  - Each PeriodicThread has the same initial delay of (B x n).
    - initial delay must be less than the thread's period
  - Each PeriodicThread has the same priority
  - Each PeriodicThread has the same period P
- Each thread is then active at a period of (T x P)
  - all such periodic threads 'share' the same PianoRoll beat
  - each time P, the next thread takes its turn

PeriodicThread cautions (2/3)

Periodic Threads of the same priority and period can execute sequentially

- Create T PeriodicThreads
  - Think of each thread having an index I, where I = [0..T-1]
  - Each PeriodicThread has a different initial delay of (I x B)
  - Each PeriodicThread has the same priority
  - Each PeriodicThread has the same period
- The PianoRoll duration D
  - must be at least as long as the LCM of B and (2 x P)
  - if longer, make it an integer multiple of the LCM of B and (2 x P)
- Each thread is then active at a period of P
  - each beat B, the next periodic thread becomes active
  - within a time P, all such threads have executed once

PianoRoll Class

public class PianoRoll extends java.lang.Object

- PianoRoll sets up a list of PeriodicThreads given:
  - desired beat in msec or msec and nsec
    - beat timer has 1 usec resolution and max of 65536 usec (16 bit timer)
    - on JStik this should be 100 usec or greater
    - on JStamp this should be 500 usec or greater
    - PeriodicThread period must be a multiple of the beat
  - duration (msec or msec + nsec)
    - must be greater than or equal to the beat
    - must be the least common multiple (LCM) of all PeriodicThread periods (if they have no initial delay), or the LCM of twice the period of each periodic thread which has non-zero initial delay.
  - You don't use markNotes() method or PeriodicTimer or PianoRoll_Timer classes to set up PianoRoll & PeriodicThreads
    - these appear to be used internally by PianoRoll

Periodic Thread execution

- Periodic threads can interrupt non-periodic threads
  - The periodic thread's priority must be higher
  - Threads are examined every PianoRoll timer tick
  - The JSI value is not used by periodic threads as it is by non-periodic threads
- Every PianoRoll beat the PeriodicThread priority mask is checked.
  - If there is a periodic thread ready to run which is of higher priority than the current periodic thread, that periodic thread is enabled.
- If the currently active PeriodicThread runs too long...
  - a beat will be quietly missed, and any Thread at that entry will be skipped over until the next time through the table.

Cautions

Don’t rely on implementation specifics – and engineer your system!

- There are few checks on periodic threads
  - you can attempt to use nonsensical values, and no exceptions will be thrown. Behavior will probably surprise you.
- Debugging periodic thread timing
  - Start simple
  - Never, ever, use System.out.println() to the Charade console. It destroys real time performance and can make execution non-deterministic
  - Set and clear GPIO pins and watch them on a ‘scope
  - Or increment variables, which you can then poll within Charade (don’t System.out.print() them!)
  - Sleep, wait, or yield within your periodic thread run method can cause drastic changes in execution, so beware.

Summary

Here’s how to set up periodic threads

- makePeriodic() one or more Threads
  - each must have a run() method
  - each must cycle() at end of run method
  - run() should loop forever (or as long as desired)
- create a PianoRoll
- start the PianoRoll
- start the PeriodicThreads

Example 1

Needed: 8 each 700 usec to 2.3 msec pulses every 20 msec

- ServoTest
  - PianoRoll with 2.5 msec beat, 2.5 msec duration (1 entry)
  - InterruptEventListener for timer interrupt
  - Timer which will create a servo pulse width
  - PeriodicThread with 2.5 msec period
    - run method steps through array of servos to service
    - run method starts a timer
    - run method cycle()
    - priority is higher than the GC so it will preempt the GC if needed
  - Now main() or other threads run
  - Timer causes interrupt event, listener handles it
    - listener turns off the current servo pulse
    - increments the array index to the next servo location, wraps if at last
  - PianoRoll beat fires again and it all repeats

Example 2

Needed: 10 Threads with a 10 msec period, executing 1 msec apart, each thread with less than 1 msec of work per active period

- PianoRoll
  - 20 msec duration (2 x Thread period)
  - 1 msec beat
- Periodic Threads
  - Create 10 Threads, almost identical
  - same 10 msec period
  - same priority
  - different initial delays of [0,1,2,3...9]
  - Each thread must cycle() after running less than 1 msec, or it will slip into the next beat

References

[AJTRM] aJ-100TM Reference Manual Version 2.1 - Dec 06, 2001

[UC] *Unpublished correspondence* - Systronix and aJile 2002, 2003

[RTS] *aJile Runtime API* - version 3.16.09 aJile Systems 2002

[CLDC] CLDC Library API Specification 1.0 - Sun Microsystems 2000