Problem Overview

The Mismatch between Arduino Hardware and Software:
- Emerging Arduino-compatible devices
- Faster processors and more complicated I/O architectures
- Increasingly complicated physical computing applications
- The standard Arduino API
- Missing support for multitreaded programs, or specification of real-time requirements
- Restricted to the capabilities found on less powerful devices

Qduino
- An operating system and programming environment
- Adds support for real-time, multithreading extensions to the standard Arduino API
- Runs on Quest RTOS for Intel Galileo + future Arduino-compatible boards

Architecture
- Driver interfaces exposed to user level through system calls.
- GPIO system calls wrapped by user level APIs in libqduino.
- Sketches run as Quest user processes, linked with libqduino.

Qduino Programming
- Allows Up to 32 `loop()` functions.
- Each `loop()` function is assigned to a Quest thread and scheduled by the Quest scheduler.
- Makes it easier to write sketches with parallel tasks.
- Experiments show up to 28% performance increase over the single-loop version.

New APIs

<table>
<thead>
<tr>
<th>Function Signatures</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>loop(loop_id, C, T)</code></td>
<td>Structure</td>
</tr>
<tr>
<td><code>interruptsVcpu(C, T)</code></td>
<td>Interrupt</td>
</tr>
<tr>
<td><code>attachInterruptVcpu(pin, ISR, node, C, T)</code></td>
<td>Spinlock</td>
</tr>
<tr>
<td><code>spinlockInit(lock)</code></td>
<td>Four-slot</td>
</tr>
<tr>
<td><code>spinlockLock(lock)</code></td>
<td>Ring buffer</td>
</tr>
<tr>
<td><code>spinlockUnlock(lock)</code></td>
<td></td>
</tr>
<tr>
<td><code>channelWrite(channel, item)</code></td>
<td></td>
</tr>
<tr>
<td><code>item channelRead(channel)</code></td>
<td></td>
</tr>
<tr>
<td><code>ringbufInit(buffer, size)</code></td>
<td></td>
</tr>
<tr>
<td><code>ringbufWrite(buffer, item)</code></td>
<td></td>
</tr>
<tr>
<td><code>ringbufRead(buffer, item)</code></td>
<td></td>
</tr>
</tbody>
</table>

Sample Sketch - Blinking LEDs

```c
int led1 = 13, led2 = 9; // connect LEDs to pin 13 and 9
int brightness = 0; // how bright the LED is
int fadeAmount = 5; // how many points to fade the LED by

// the LED is on
digitalWrite(led1, HIGH);
// wait for a second
delay(1000);
// the LED is off
digitalWrite(led1, LOW);
// wait for a second
delay(1000);
void loop(1, 40, 100) { // loop 1 with VCPU (40, 100) blinks led1
    digitalWrite(led1, HIGH);
    delay(1000);
    digitalWrite(led1, LOW);
    delay(1000);
}

// the LED is on
digitalWrite(led2, HIGH);
// the LED is off
digitalWrite(led2, LOW);
void loop(2, 20, 100) { // loop 2 with VCPU (20, 100) fades led2
    analogWrite(led2, brightness);
    // change the brightness
    if (brightness < 0) brightness += fadeAmount;
    // reverse the direction of the fading at the ends of the fade
    if (brightness < 255) {
        fadeAmount = -fadeAmount;
    }
    delay(30);
    // wait for 30 milliseconds to see the dimming effect
}

void setup() {
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);
}
```

Temporal Isolation

- The execution of one loop is guaranteed not to interfere with the timely execution of others.
- Interrupts are handled in threads so that they do not unduly interfere with the execution of loops.

Predictable Events

- User level interrupt handling threads bind to Main VCPUs
- The Main VCPUs are invoked by wake-up events generated by the bottom half.
- Kernel level threaded bottom half binds to an I/O VCPU
- The I/O VCPU are invoked by hardware interrupt handler
- The above process is bounded by its worst-case delay (WCD):
  \[ \Delta_{WCD} = (T_h - C_h) + \Delta_{bh} = (T_h - C_h) + (T_i - C_i) + \left( \frac{\delta_{bh}}{C_i} - 1 \right) T_i + \delta_{bh} \mod C_i \]
- Notation:
  - \((C_h, T_h)\) - parameters of the Main VCPU associated with the user level interrupt handler
  - \((C_i, T_i)\) - parameters of the I/O VCPU associated with the bottom half
  - \(\Delta_{bh}\) and \(\delta_{bh}\) - the wall-clock time and the required CPU time to execute the bottom half

Qduino Website: www.cs.bu.edu/fac/richwest/Qduino.php