Qduino: A Multithreaded Arduino System for Embedded Computing

Zhuoqun Cheng, Ye Li, Richard West
Background

- Many Robotics, Internet of Things, Home Automation applications have been developed recently
  - Perform complicated computing tasks
  - Interact with the physical world
- Need an easy-to-use platform to develop applications
  - High processing capabilities
  - Straightforward hardware and software interface
Background

- Arduino
  - Digital and analog GPIOs
  - Simple API
  - Low processing capabilities
    - Arduino Uno: 16MHz 8-bit ATmega328P
More powerful Arduino-compatible boards emerge to meet the demands

- **Intel Galileo**: 400MHz Intel Quark X1000
- **Intel Edison**: 500MHz dual-core Atom
- **Arduino-compatible**: the same GPIO layout with the standard Arduino boards
Background

- The standard Arduino runs sketches (Arduino program) on the bare metal
- New boards are shipped with Linux
  - Able to afford the overhead of operating systems
  - To cope with the complexity of the hardware
  - Run sketches as Linux processes
Motivation

- Linux lacks predictability
  - Many embedded applications have real-time requirements
  - RTOS is needed
- The standard Arduino API designed for a single thread of execution
  - No multithreading or concurrency
  - Fails to utilize computing resources and hardware parallelism
Qduino: a programming environment that provides support for preemptive multithreading Arduino API that guarantees timing predictability of different control flows in a sketch

- Multithreaded sketches, and synchronization and communication between control flows
- Temporal isolation between different control flows and asynchronous system events, e.g., interrupts
- Predictable event delivery for I/O handling in sketches
## Arduino vs Qduino APIs

<table>
<thead>
<tr>
<th>Category</th>
<th>Standard APIs</th>
<th>New APIs (backward compatible)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure</strong></td>
<td>setup(), loop()</td>
<td>loop(id, C, T)</td>
</tr>
<tr>
<td><strong>Digital and Analog I/Os</strong></td>
<td>pinMode(), digitalWrite(), digitalRead(), analogWrite(), analogRead()</td>
<td></td>
</tr>
<tr>
<td><strong>Interrupts</strong></td>
<td>Interrupts(), noInterrupts(), attachInterrupt(pin, ISR, mode), detachInterrupt(pin)</td>
<td>interruptsVcpu(C, T), attachInterruptVcpu(pin, ISR, mode, C, T)</td>
</tr>
<tr>
<td><strong>Synchronization &amp; Communication</strong></td>
<td></td>
<td>spinlock, four-slot channel, ringbuffer</td>
</tr>
<tr>
<td><strong>Other Utility Functions</strong></td>
<td>micros(), delay(), min(), sqrt(), sin(), isLowerCase(), random(), bitset(), ...</td>
<td></td>
</tr>
</tbody>
</table>
Contributions

• Qduino:
  • Multithreaded sketches, and synchronization and communication between control flows
  • Temporal isolation between different control flows and asynchronous system events, e.g., interrupts
  • Predictable event delivery for I/O handling in sketch
Multithreaded Sketch

- Structure
  - `loop()`, `setup()`
  - `loop(id, C, T)`

- Standard API
  - Only one `loop()` is allowed
  - Blocking I/Os block the sketch

- Qduino:
  - Up to 32 `loop()` in one sketch
  - Each `loop()` function is assigned to a Quest thread
Multithreaded Sketch

• Benefits
  • Loop interleaving
    • Blocking I/Os won't block the entire sketch
    • increase CPU utilization
  • Easy to write sketches with parallel tasks
    • Example: toggle pin 9 every 2s, pin 10 every 3s
Multithreaded Sketch

//Sketch 1: toggle pin 9 every 2s
int val9 = 0;

void setup() {
    pinMode(9, OUTPUT);
}

void loop() {
    val9 = !val9; //flip the output value
digitalWrite(9, val9);
    delay(2000); //delay 2s
}

//Sketch 2: toggle pin 10 every 3s
int val10 = 0;

void setup() {
    pinMode(10, OUTPUT);
}

void loop() {
    val10 = !val10; //flip the output value
digitalWrite(10, val10);
    delay(3000); //delay 3s
}

Delay(?)

No way to merge them!
Multithreaded Sketch

- Inefficient
- Do scheduling by hand
- Hard to scale

```cpp
int val9, val10 = 0;
int next_flip9, next_flip10 = 0;

void setup() {
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
}

void loop() {
  if (millis() >= next_flip9) {
    val9 = !val9; /*flip the output value*/
    digitalWrite(9, val9);
    next_flip9 += 2000;
  }
  if (millis() >= next_flip10) {
    val10 = !val10; /*flip the output value*/
    digitalWrite(10, val10);
    next_flip10 += 3000;
  }
```
void setup() {
    pinMode(9, OUTPUT);
    pinMode(10, OUTPUT);
}

void loop1, C, T) {
    val9 = !val9; //flip the output value
    digitalWrite(9, val9);
    delay(2000);
}

void loop2, C, T) {
    val10 = !val10; //flip the output value
    digitalWrite(10, val10);
    delay(3000);
}
Communication & Synchronization

- Loops – threads
  - Communication via global variables
- Serialized global variable access
  - Explicit: spinlock
  - Implicit: channel, ring buffer

<table>
<thead>
<tr>
<th>Function Signatures</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>spinlockInit(lock)</td>
<td>Spinlock</td>
</tr>
<tr>
<td>spinlockLock(lock)</td>
<td></td>
</tr>
<tr>
<td>spinlockUnlock(lock)</td>
<td></td>
</tr>
<tr>
<td>channelWrite(channel,item)</td>
<td>Four-slot</td>
</tr>
<tr>
<td>item channelRead(channel)</td>
<td></td>
</tr>
<tr>
<td>ringbufInit(buffer,size)</td>
<td>Ring buffer</td>
</tr>
<tr>
<td>ringbufWrite(buffer,item)</td>
<td></td>
</tr>
<tr>
<td>ringbufRead(buffer,item)</td>
<td></td>
</tr>
</tbody>
</table>
Contributions

• Qduino:
  • Multithreaded sketches, and synchronization and communication between control flows
  • Temporal isolation between different control flows and asynchronous system events, e.g., interrupts
  • Predictable event delivery for I/O handling in sketch
Temporal Isolation

- Real-time Virtual CPU (VCPU) Scheduling
  - VCPU: kernel objects for time accounting and scheduling
  - Two classes:
    - Main VCPU – conventional thread
    - I/O VCPU – threaded interrupt handler
Temporal Isolation

- Real-time Virtual CPU (VCPU) Scheduling
  - Each VCPU has a max budget $C$, a period $T$ and a utilization $U = C / T$
  - Integrate the scheduling of tasks & I/O interrupts
    - Extension to rate-monotonic scheduling
    - Ensure temporal isolation if the Liu-Layland utilization bound is satisfied
### Temporal Isolation

<table>
<thead>
<tr>
<th>Structure</th>
<th>loop(), setup()</th>
<th>loop(id, C, T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts</td>
<td>interrupts()</td>
<td>interruptsVcpu(C, T)</td>
</tr>
</tbody>
</table>

- **Loop** – thread – Main VCPU
  - Specify loop timing requirements

- **GPIO interrupt handler** – I/O VCPU
  - Control # of interrupts to handle

- Balance CPU time between tasks, as well as tasks and interrupts

---

**Sketch**

```
Quest Native App ...
```

**Qduino Libs**

```
Sketch

loop1 ... loopN
```

**VCPU Scheduler**

- **GPIO Driver**
- **SPI Driver**
- **I2C Driver**

**User Kernel**

- **Galileo**
- **Edison**
- **Minnowboard**

**x86 SoC**
Contributions

- **Qduino:**
  - Multithreaded sketches, and synchronization and communication between control flows
  - Temporal isolation between different control flows and asynchronous system events, e.g., interrupts
  - Predictable event delivery for I/O handling in sketch
Event delivery time: the time interval between the invocation of the ISR and the invocation of the user-level interrupt handler

Predictable end-to-end event delivery

attachInterruptVcpu(..., C, T), interruptsVcpu(C, T)
Predictable Events

- I/O VCPU \((C_{io}, T_{io})\) – threaded interrupt bottom half
- Main VCPU \((C_h, T_h)\) – threaded user interrupt handler

Worst Case Event Delivery Time:

\[
\Delta_{WCD} = \Delta_{bh} + (T_h - C_h) = (T_{io} - C_{io}) + \left\lfloor \frac{\delta_{bh}}{C_{io}} - 1 \right\rfloor \cdot T_{io} + \delta_{bh} \mod C_{io} + (T_h - C_h)
\]
Evaluation

- Experiment Setup
  - Intel Galileo board Gen 1
  - Qduino vs. Clanton
    - Clanton Linux 3.8.7 is shipped with the Galileo board
Evaluation

- Multithreaded Sketch
  - Computation-intensive: find all prime numbers smaller than 80000
  - I/O-intensive: 2000 digital write
  - Reduce 30% CPU Cycles

<table>
<thead>
<tr>
<th>Case #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>Single-loop digitalWrite()</td>
</tr>
<tr>
<td>Case 2</td>
<td>Single-loop findPrime</td>
</tr>
<tr>
<td>Case 3</td>
<td>Single-loop digitalWrite() + findPrime</td>
</tr>
<tr>
<td>Case 4</td>
<td>Multi-loop digitalWrite() + findPrime</td>
</tr>
</tbody>
</table>
Predictable loop execution

1 Foreground loop increments a counter during its loop period

2/4 background loops act as potential interference

Result interpretation
- Overlapped – temporal isolation
- Straight line – timing guarantee
Temporal Isolation between loops and interrupts

- Use an external device to toggle pin 2 of Galileo
- Run findPrime at the same time
- Execution time of findPrime and # of interrupts handled

<table>
<thead>
<tr>
<th>Case #</th>
<th>I/O VCPU</th>
<th>External Interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>10/100</td>
<td>OFF</td>
</tr>
<tr>
<td>Case 2</td>
<td>0/100</td>
<td>ON</td>
</tr>
<tr>
<td>Case 3</td>
<td>5/100</td>
<td>ON</td>
</tr>
<tr>
<td>Case 4</td>
<td>10/100</td>
<td>ON</td>
</tr>
<tr>
<td>Case 5</td>
<td>Disabled</td>
<td>ON</td>
</tr>
</tbody>
</table>
Autonomous Vehicle

- Collision avoidance using ultrasonic sensor
- Two tasks:
  - A sensing task detects distance to an obstacle - delay(200)
  - An actuation task controls the motors - delay(100)
- **Autonomous Vehicle**

  - Measure the time interval between two consecutive calls to the motor actuation code
  - Clanton single loop
    - delay from both sensing and actuation task
  - Qduino multi-loop
    - No delay from the sensing loop
    - No delay from sensor timeout
  - The shorter the worst case time interval, the faster the vehicle can drive

**Evaluation**

- **Clanton Single-loop**
- **Qduino Multi-loop**
- **Qduino Single-loop**
- **Clanton Interrupt**

![Graph showing time intervals for different loop types](image)
Conclusions

- Supported Quest RTOS on Intel Arduino-compatible boards

- Designed and implemented an extension to the Arduino API for Quest on new powerful Arduino-compatible boards
  
  - Multi-loop sketches
  
  - Real-time guarantee
Thank you!

- Questions?

- More information can be found at:
Future Work

- Conditional loops
- Communication between loops with loop IDs
- Multi-sketches
## Memory Footprint

<table>
<thead>
<tr>
<th></th>
<th>Text (Bytes)</th>
<th>Data (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qduino kernel</td>
<td>953358</td>
<td>321516</td>
</tr>
<tr>
<td>Clanton kernel</td>
<td>4390436</td>
<td>336104</td>
</tr>
<tr>
<td>Qduino autonomous vehicle sketch</td>
<td>4832</td>
<td>2360</td>
</tr>
<tr>
<td>Clanton autonomous vehicle sketch</td>
<td>26249</td>
<td>27652</td>
</tr>
</tbody>
</table>
### GPIOs

<table>
<thead>
<tr>
<th>Category</th>
<th>Standard APIs</th>
<th>Newly added APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital and Analog I/Os</td>
<td>PinMode(), digitalWrite(), digitalRead(),</td>
<td>anlogWrite(), anlogRead()</td>
</tr>
</tbody>
</table>

- **Complicated I/O Architecture on new boards**

![Diagram of GPIO architecture and board support](image)