ComputerScience

Boston University

Qduino: A Multithreaded Arduino System for Embedded Computing

Zhuoqun Cheng, Ye Li, Richard West

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 multithreaded 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.

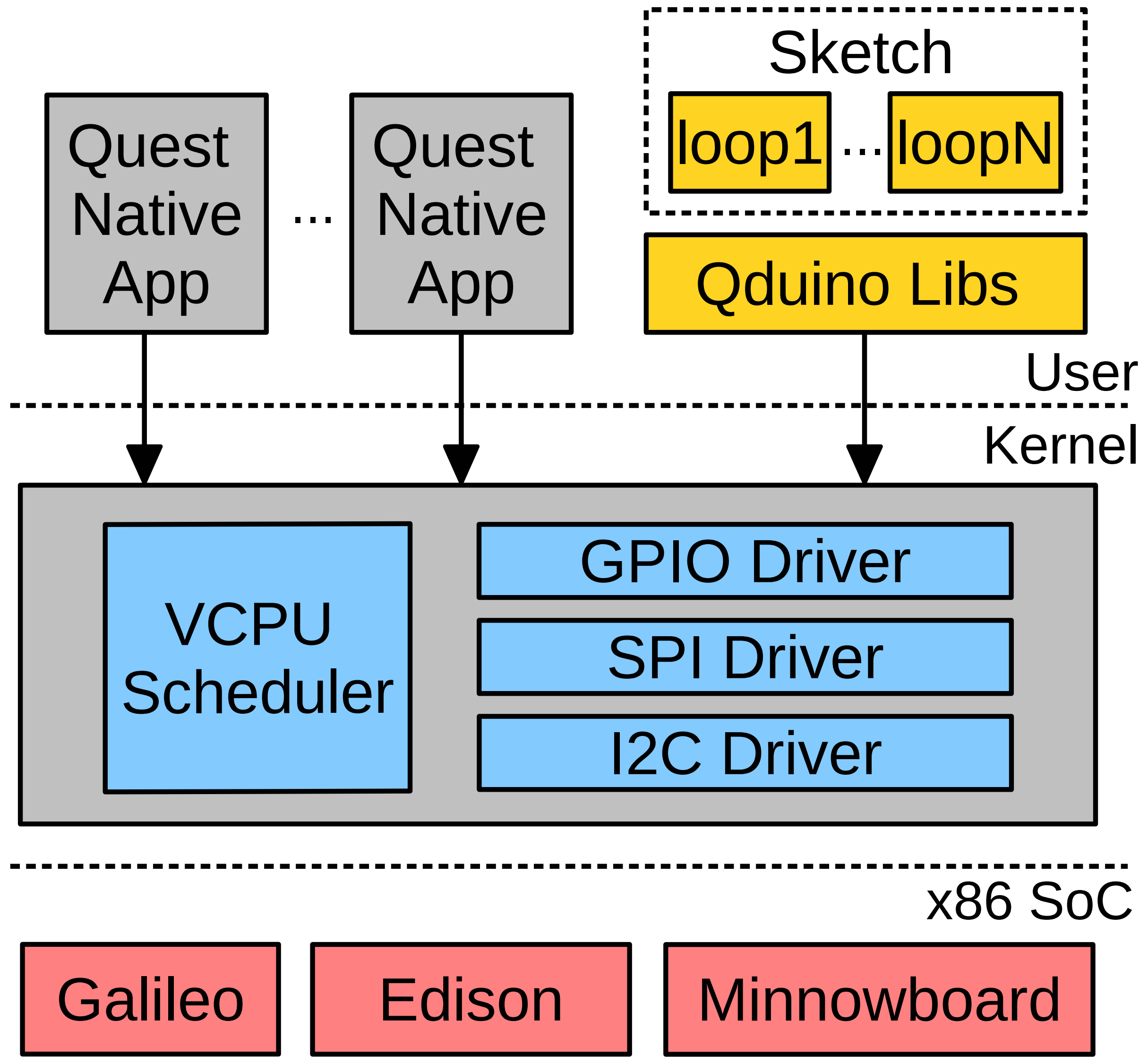

Figure 1: Qduino Architecture

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

Function Signatures	Category
`loop(loop_id, C, T)`	Structure
`interruptsVcpu(C, T)`, `attachInterruptVcpu(pin, ISR, mode, C, T)`	Interrupt
`spinlockInit(lock)`, `spinlockLock(lock)`, `spinlockUnlock(lock)`	Spinlock
`channelWrite(channel, item)`, `item channelRead(channel)`	Four-slot
`ringbufInit(buffer, size)`, `ringbufWrite(buffer, item)`, `ringbufRead(buffer, item)`	Ring buffer

Sample Sketch - Blinking LEDs

```
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

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

void loop(2,20,100) {  // loop 2 with VCPU (20,100) fades led2
  analogWrite(led2, brightness);         // set the brightness of pin 9
  brightness = brightness + fadeAmount; // change the brightness
  // reverse the direction of the fading at the ends of the fade
  if (brightness == 0 || 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.

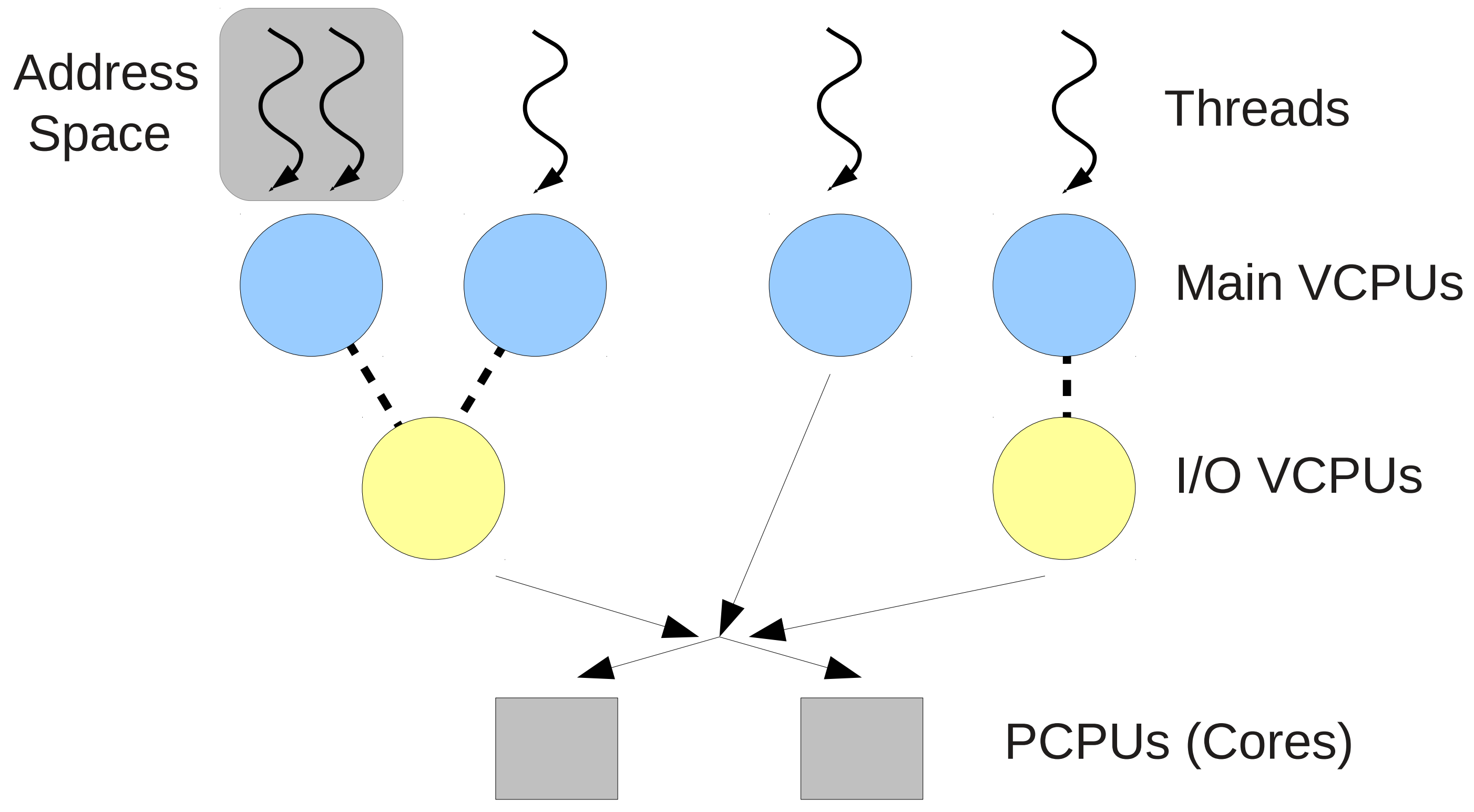

Figure 2: Quest VCPU Hierarchy

Predictable Events

- User level interrupt handling threads bind to Main VCPUs
- The Main VCPUs are invoked by wakeup 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):

$$\Delta_{WCD} = (T_h - C_h) + \Delta_{bh} = (T_h - C_h) + (T_{io} - C_{io}) + \left\lceil \frac{\delta_{bh}}{C_{io}} - 1 \right\rceil \cdot T_{io} + \delta_{bh} \bmod C_{io}$$

- Notation:
  - (C_h, T_h) - parameters of the Main VCPU associated with the user level interrupt handler
  - (C_{io}, T_{io}) - parameters of the I/O VCPU associated with the bottom half
  - Δ_{bh} and δ_{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

Advisor: Richard West