Real-Time Multitasking in Arduino

Pasquale Buonocunto, Alessandro Biondi, Pietro Lorefice
Who I am

- **Pasquale Buonocunto**
  PhD Fellow at Scuola Superiore Sant'Anna

- Graduated cum Laude (Oct 2012) in Computer Engineering at University of Pisa

- Nov 2012, now – PhD Fellow on Embedded Systems at TeCIP Institute, Scuola Superiore Sant'Anna, Pisa.

**Research Interests**

- Real-time operating systems: design and implementation;
- Real-time wireless communication
- Wearable, low power e-Health devices
Who I am

- **Alessandro Biondi**
  PhD Fellow at Scuola Superiore Sant'Anna

- Graduated cum Laude in 2013 in **Computer Engineering** at University of Pisa, within the excellence program;

- Aug 2011, Oct 2011 – **Visiting student** @ San Diego State University, California.

**Research Interests**

- Real-time operating systems: design and implementation;
- Real-time schedulability analysis;
- Hierarchical systems;
- Synchronization protocols.
Arduino Framework

“Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer.”

Very popular

- 2013: 700,000+ official boards were in users' hands (not counting unofficial clones...)
Arduino Framework

- **Very simple!**
  - Simple programming interface;
  - Easy firmware loading;
Arduino Framework

- **Low cost**
  - Arduino Uno ~20€;
  - Arduino Due ~35€.
Arduino Framework

- **Very popular**
  - 2013: 700,000+ official boards were in users' hands

- **Very simple!**
  - Simple
  - Easy firmware loading

- **Low cost**
  - ~20$ official

“Embedded system programming for everyone”
Arduino Framework

...Very simple!

```c
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.

 * This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```
Arduino Framework

void setup() {
  <instructions here>
}

void loop() {
  <instructions here>
  delay(1000);
}

One-shot execution at startup

Cyclically executed "until power off"!
**Arduino Framework**

A lot of libraries...  

### Standard Libraries

- EEPROM
- Ethernet
- GSM
- LiquidCrystal
- SD
- Servo
- SPI
- SoftwareSerial
- Stepper
- TFT
- WiFi
- Wire

### Contributed Libraries

- Messenger
- NewSoftSerial
- OneWire
- PS2Keyboard
- Simple Message System
- SSerial2Mobile
- Webduino
- X10
- XBee
- SerialControl
- Capacitive Sensing
- Debounce
- GLCD

- Improved LCD
- LedControl
- LedDisplay
- Matrix
- PCD8544
- Sprite
- ST7735
- FFT
- Tone
- TLC5940
- DateTime
- Metro
- MsTimer2
- PString
- Streaming
Limitation of the Arduino Framework

- **No** support for concurrency;

- Execution limited to a single loop of instructions;

- **No** period can be expressed.

```c
void loop() {
    <instructions here>
    delay(1000);
}
```
Existing Solutions

- **Scheduler Library**
  - Support for multiple “concurrent” loops;
  - Cooperative Scheduler: each task is responsible to “pass the baton”;
  - No periodic activities can be expressed;
  - Experimental Library.
void setup() {
    Scheduler.startLoop(func1);
    Scheduler.startLoop(func2);
}

void func1() {
    digitalWrite(led2, HIGH);
    delay(100);
    digitalWrite(led2, LOW);
    delay(100);
}

void func2() {
    // <instructions here>
    yield(); // Pass control to other tasks.
}
Existing Solutions

Scheduler Library

```cpp
void setup() {
    Scheduler.startLoop(func1);
    Scheduler.startLoop(func2);
}

void func1() {
    digitalWrite(led2, HIGH);
    delay(100);
    digitalWrite(led2, LOW);
    delay(100);
}

void func2() {
    <instructions here>
    yield(); // Pass control to other tasks.
}
```

No Scheduling Policy

The scheduling pattern is established by explicit call to `yield()` or `delay()`
Existing Solutions

**Scheduler Library**

void setup() {
    Scheduler.startLoop(func1);
    Scheduler.startLoop(func2);
}

void func1() {
    digitalWrite(led2, HIGH);
    delay(100);
    digitalWrite(led2, LOW);
    delay(100);
}

void func2() {
    <instructions here>
    yield(); // Pass control to other tasks.
}
Existing Solutions

- Other Solutions:
  - Arduino Simple Task Scheduler;
  - Looper;
  - WrapOS;
  - ...

Real-Time Systems Laboratory
Existing Solutions

```c
void put_leds_into_cycle()
{
    // Pulse all the LEDs in sequence...
    scheduler.createSchedule(20, 7, true, toggle_02_oclock_led_r);
    scheduler.delaySchedule(scheduler.createSchedule(20, 7, true, toggle_04_oclock_led_r), 7*20);
    scheduler.delaySchedule(scheduler.createSchedule(20, 7, true, toggle_08_oclock_led_r), 7*20*2);
    scheduler.delaySchedule(scheduler.createSchedule(20, 7, true, toggle_10_oclock_led_r), 7*20*3);
}
```

```c
void Main(void) // entry point
{
    SEM_ID sem1;
    MSGQ_ID msgQ1;

    semCreate(sem1); // initial it.
    msgQCreate(msgQ1, 100, 10);

    // create and start tasks.
    taskCreate("task1", 240, 0x7fff, (FUNCPTR)task1);
    taskCreate("task2", 250, 0x7fff, (FUNCPTR)task2);
}
```

```c
void put_leds_into_cycle()
{
    // Pulse all the LEDs in sequence...
    scheduler.createSchedule(20, 7, true, toggle_02_oclock_led_r);
    scheduler.delaySchedule(scheduler.createSchedule(20, 7, true, toggle_04_oclock_led_r), 7*20);
    scheduler.delaySchedule(scheduler.createSchedule(20, 7, true, toggle_08_oclock_led_r), 7*20*2);
    scheduler.delaySchedule(scheduler.createSchedule(20, 7, true, toggle_10_oclock_led_r), 7*20*3);
}
```

```c
void Main(void) // entry point
{
    SEM_ID sem1;
    MSGQ_ID msgQ1;

    semCreate(sem1); // initial it.
    msgQCreate(msgQ1, 100, 10);

    // create and start tasks.
    taskCreate("task1", 240, 0x7fff, (FUNCPTR)task1);
    taskCreate("task2", 250, 0x7fff, (FUNCPTR)task2);
}
```

```c
void task1(void)
{
    while (!done) // main loop
    {
        semTake(sem1, 10); // wait for the semaphore

        // Do task1 stuff here
    }
}
```

```c
void task2(void)
{
    while (!done) // main loop
    {
        // wait for a message to come.
        msgQReceive(msgQ1, buf, size, WAIT_FOREVER);

        // Do task2 stuff here.
    }
}
```
Existing Solutions

```c
scheduler.createSchedule(7*20*4, -1, false, put_leds_into_cycle);

uint32_t pid = scheduler.createSchedule(50, -1, false, callback_function);
scheduler.beginProfiling(pid);
scheduler.stopProfiling(pid);
scheduler.clearProfilingData(pid);

char * temp = dumpProfilingData(void);
if (temp != NULL) {
    Serial.println(temp);
    free(temp);
}

/* Fires callback_function() after 790ms, and then cleans itself up. */
scheduler.createSchedule(790, 0, true, callback_function);

/* Blinks an LED for (20*9)ms. */
scheduler.createSchedule(20, 9, true, toggle_led);

/* Makes a 500Hz beep that lasts for 0.5 seconds. */
scheduler.createSchedule(1, 1000, true, toggle_speaker_pin);

void put_leds_into_cycle() {
    // Pulse all the LEDs in sequence...
    scheduler.createSchedule(20, 7, true, toggle_02_oclock_led_r);
    scheduler.delaySchedule(scheduler.createSchedule(20, 7, true, toggle_02_oclock_led_r), 7*20);
    scheduler.delaySchedule(scheduler.createSchedule(20, 7, true, toggle_08_oclock_led_r), 7*20*2);
    scheduler.delaySchedule(scheduler.createSchedule(20, 7, true, toggle_10_oclock_led_r), 7*20*3);
}
```

void Main(void) // entry point
{
    SEM_ID sem1;
    MSGQ_ID msgQ1;
    semCreate(sem1);  //  initial it.
    msgQCreate(msgQ1, 100, 10);
    // create and start tasks.
    taskCreate("task1", 240, 0x7fff, (FUNCPTR)task1);
    taskCreate("task2", 250, 0x7fff, (FUNCPTR)task2);

    void task1(void) {
        while (!done) { // main loop
            semTake(sem1, 10);  //  wait for the semaphore
            //  Do task1 stuff here
        }
    }

    void task2(void) {
        while(!done) {  // main loop
            //  wait for a message to come.
            msgQReceive(msgQ1, buf, size, WAIT_FOREVER);
            //  Do task2 stuff here.
        }
    }
}
```
ArTe: Arduino Real-Time Extension
**ARTE:** Arduino Real-Time Extension

**GOAL**

- Support for **real-time multitasking** and **periodic activities**;
- Maintain a **very simple programming interface** compliant with the Arduino philosophy;
- **Minimum amount of differences** with respect to the original Arduino programming model.
What is Real-Time Multitasking?

- **Scheduling Algorithm**: Rules dictating the task that needs to be executed on the CPU at each time instant.
What is Real-Time Multitasking?

- **Preemptive** & **Priority driven** scheduling

  - Task have **priorities** (static);
  
  - At each time, the **highest priority** task is executed;

- **Preemption**: the execution of the running task can be interrupted by an **higher priority** task, and the CPU given to the new task.
What is Real-Time Multitasking?

- **Periodic Activities**

- **Standard Real-time programming model:**
  - Periodic activities without explicit delay/suspensions;
  - Be predictable: polling preferred to event reaction.
What is Real-Time Multitasking?

- Periodic Activities

- Typical Real-Time Applications:
  - Sampling sensors and actuating (e.g., control loop,...);
  - Multimedia and transmissions;
What is Real-Time Multitasking?

- **Periodic Activities**

- **Deadline**: we want the task finishing before the next activation.
What is Real-Time Multitasking?

1) Preemption

2) Preemption

3) Preemption
Real-Time Multitasking and Arduino

- **Idea**: Exploit the power of Real-Time Multitasking with the simple and effective Arduino Programming Model;

- Extension to the Arduino Framework allowing concurrent, multiple, loops each one scheduled with a **Real-Time OS**
ERIKA Enterprise

- ERIKA Enterprise is an OSEK/VDX certified RTOS;
- ERIKA Enterprise implements an API inspired by the AUTOSAR OS API;
- Offers a suitable open-source license allowing the static linking of closed source code;
- Typical footprint around 2-4KB Flash;
- Used by several automotive/white goods companies.
Example

- Make blinking three different LEDs, each one at a different frequency.
  - **Led1**: 3s
  - **Led2**: 7s
  - **Led3**: 11s
int led1 = 13;
int led2 = 14;
int led3 = 15;
int count = 0;

void loop() {
    if (count%3 == 0)
        digitalToggle(led1);

    if (count%7 == 0)
        digitalToggle(led2);

    if (count%11 == 0)
        digitalToggle(led3);

    if (count == 3*7*11)
        count = 0;
    count++;
    delay(1000);
}
**Example**

With **Arduino** Real-Time Extension

`int led1 = 13;`  
`int led2 = 14;`  
`int led3 = 15;`  

```c
void loop1(3000) {
  digitalToggle(led1);
}

void loop2(7000) {
  digitalToggle(led2);
}

void loop3(11000) {
  digitalToggle(led3);
}
```

<table>
<thead>
<tr>
<th>Led1</th>
<th>Led2</th>
<th>Led2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3s</td>
<td>7s</td>
<td>11s</td>
</tr>
</tbody>
</table>
Arduino Real-Time Extension

- **ARTE** is a real extension to the Arduino IDE, not an external library!

- It provides a code pre/post-processor that avoids tiring and error-prone user configurations;

- **Automatic generation** of the application configuration and skeleton code;

- The user is focused on the application logic.
Arduino Real-Time Extension

- **ARTE** has a **real** RTOS under the hood
```c
int led1 = 13;
int led2 = 14;
int led3 = 15;

void loop1(3000) {
    digitalToggle(led1);
}

void loop2(7000) {
    digitalToggle(led2);
}

void loop3(11000) {
    digitalToggle(led3);
}
```
ERIKA Enterprise

- ERIKA Enterprise is a **static** RTOS:
  - All the RTOS configuration (*tasks, resources, counters, hardware configuration*) is decided at compiling time;
  - **Minimal** impact on the RAM;
  - **OIL** language is used to configure the RTOS.
Example

- Mapping to an OSEK application

```c
int led1 = 13;
int led2 = 14;
int led3 = 15;

void loop1(3000) {
    digitalToggle(led1);
}

void loop2(7000) {
    digitalToggle(led2);
}

void loop3(11000) {
    digitalToggle(led3);
}
```

```c
TASK loop1 {
    PRIORITY = 0x01;
    SCHEDULE = FULL;
    STACK = SHARED;
};

ALARM Alarmloop1 {
    COUNTER = TaskCounter;
    ACTION = ACTIVATETASK {
        TASK = loop1;
    };}

TASK loop2 {
    PRIORITY = 0x02;
    SCHEDULE = FULL;
    STACK = SHARED;
};

ALARM Alarmloop2 {
    COUNTER = TaskCounter;
    ACTION = ACTIVATETASK {
        TASK = loop2;
    };}

TASK loop3 {
    PRIORITY = 0x03;
    SCHEDULE = FULL;
    STACK = SHARED;
};

ALARM Alarmloop3 {
    COUNTER = TaskCounter;
    ACTION = ACTIVATETASK {
        TASK = loop3;
    };}
```
Build Process

**Core of the Arduino Real-Time Extension**

Generates an ERIKA application code starting from the Arduino code

Parse the sketch an automatically generates the RTOS configuration
Build Process

**OIL** (OSEK Implementation Language)
Build Process

Arduino build process as ERIKA application

Link all together!
Build Process

Flashing of the binary on the Arduino board
DEMO
Arduino Real-Time Extension

- Additional possible usage:
  - **Fast prototyping**: an ERIKA application is automatically generated from the sketch; it can be used for extending the development by directly working with the RTOS.
Conclusion

- **ARTE** is an extension for the Arduino framework to support Real-Time Multitasking;

- Simple programming interface with minimal impact on the Arduino programming model;

- Relies on an OSEK/VDX RTOS (ERIKA Enterprise);
Thank you!

Alessandro Biondi
alessandro.biondi@sssup.it

Pasquale Buonocunto
p.buonocunto@sssup.it