Embedded HW and OS – Wk3 Tiny OS Hands-On Assignment

Simulation of blinking LEDs on a simulated MICAz mote (MICAz datasheet) using TinyOS and the supplied Blink app. First we'll get the default Blink application running and then add another timer, Timer3.

Background

The MICAz mote has three (3) LEDs, red, green and yellow. The default Blink application supplied with TinyOS blinks (turns on and off) these LEDs at 250 (4Hz), 500 (2Hz), and 1000(1Hz) millisecond (Hz) intervals. Basically a binary counter.

Blink Application.

Blink is composed of a configuration and a module. All Tiny OS applications are composed of a configuration file and an implementation file. The configuration is used to assemble the needed components together. The implementation provides the functionality of the implementation.

The Blink Configuration – BlinkAppC.nc

The configuration section indicates this is a configuration file. No other clauses are used by the configuration.

The implementation section describes the set of components referenced, MainC, BlinkC, LedsC, TimerMilliC. The TimerMilliC are further qualified using the 'as' keyword. The MainC.Boot allow for the LEDs to be initialized as part of the boot sequence.

The Timer0-2 and LedsC lines connect interfaces between BlinkC and TimerMiliC and LedsC.

```
/**
 * Blink is a basic application that toggles a mote's LED periodically.
 * It does so by starting a Timer that fires every second. It uses the
 * OSKI TimerMilli service to achieve this goal.
 *
 * @author tinyos-help@millennium.berkeley.edu
 **/

configuration BlinkAppC
{
}
implementation
{
  components MainC, BlinkC, LedsC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
```

```
  components new TimerMilliC() as Timer2;


  BlinkC -> MainC.Boot;

  BlinkC.Timer0 -> Timer0;
  BlinkC.Timer1 -> Timer1;
  BlinkC.Timer2 -> Timer2;
  BlinkC.Leds -> LedsC;
}
```

Blink Implementation – BlinkC.nc

Implementation of the Blink application. The first section, module, indicates this is the BlinkC module and that it uses interfaces Timer, Leds, and Boot.

The implementation is coded to start the 3 periodic timers when the booted event is received. Event handlers for the timer event received and then toggle the Timer.

```
/**
 * Implementation for Blink application.  Toggle the red LED when a
 * Timer fires.
 **/

#include "Timer.h"
s indicate the timer
module BlinkC @safe()
{
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
}
implementation
{
  event void Boot.booted()
  {
    call Timer0.startPeriodic( 250 );
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
  }

  event void Timer0.fired()
  {
    dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
    call Leds.led0Toggle();
  }

  event void Timer1.fired()
  {
    dbg("BlinkC", "Timer 1 fired @ %s \n", sim_time_string());
```

```
    call Leds.led1Toggle();
  }

  event void Timer2.fired()
  {
    dbg("BlinkC", "Timer 2 fired @ %s.\n", sim_time_string());
    call Leds.led2Toggle();
  }
}
```

Compile and Run The Application

From the notes:

- Compile the application
  - make micaz sim
- Create a python script to run the application – runblink.py
  - ```
    #!/usr/bin/python
    from TOSSIM import *
    import sys
    t = Tossim([])
    r = t.radio()
    t.addChannel("BlinkC", sys.stdout)
    m = t.getNode(1)
    m.bootAtTime(100)
    print "while loop"
    while (m.isOn() == 0):
     t.runNextEvent()
    print "for loop"
    for i in range(0, 100):
     t.runNextEvent()
    ```
- Now run the application
    ```

```
user@instant-contiki:~/tinyos-release/apps/Blink$ python runblink.py
while loop
for loop
DEBUG (1): Timer 0 fired @ 0:0:0.244140645.
DEBUG (1): Timer 0 fired @ 0:0:0.488281270.
DEBUG (1): Timer 1 fired @ 0:0:0.488281280
DEBUG (1): Timer 0 fired @ 0:0:0.732421895.
DEBUG (1): Timer 0 fired @ 0:0:0.976562520.
DEBUG (1): Timer 1 fired @ 0:0:0.976562530
DEBUG (1): Timer 2 fired @ 0:0:0.976562540.
DEBUG (1): Timer 0 fired @ 0:0:1.220703145.
DEBUG (1): Timer 0 fired @ 0:0:1.464843770.
DEBUG (1): Timer 1 fired @ 0:0:1.464843780
DEBUG (1): Timer 0 fired @ 0:0:1.708984395.
DEBUG (1): Timer 0 fired @ 0:0:1.953125020.
DEBUG (1): Timer 1 fired @ 0:0:1.953125030
DEBUG (1): Timer 2 fired @ 0:0:1.953125040.
DEBUG (1): Timer 0 fired @ 0:0:2.197265645.
DEBUG (1): Timer 0 fired @ 0:0:2.441406270.
DEBUG (1): Timer 1 fired @ 0:0:2.441406280
user@instant-contiki:~/tinyos-release/apps/Blink$
```

Task 1 Running of Blink Application

Modify the Blink Application

The two following modifications should be made to the Blink application code:

1) Add another timer called “Timer3” and schedule it every 100ms.
   User this timer to print out the message “I am Timer 3 and I have the shortest period!”

2) Make the simulation run for 2000 events instead of 100

Modifications

The following modifications were made (see highlighted sections below).

BlinkAppC.nc

Add the new timer, Timer3, and associate with the BlinkC implementation.

```
/**
 * Blink is a basic application that toggles a mote's LED periodically.
 * It does so by starting a Timer that fires every second. It uses the
 * OSKI TimerMilli service to achieve this goal.
 *
 * @author tinyos-help@millennium.berkeley.edu
 **/
```

```
configuration BlinkAppC
{
}
implementation
{
  components MainC, BlinkC, LedsC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;
  components new TimerMilliC() as Timer3;


  BlinkC -> MainC.Boot;

  BlinkC.Timer0 -> Timer0;
  BlinkC.Timer1 -> Timer1;
  BlinkC.Timer2 -> Timer2;
  BlinkC.Timer3 -> Timer3;
  BlinkC.Leds -> LedsC;
```

BlinkC.nc

Access the new timer, Timer3, and establish its period as 100 milliseconds or 10Hz. Set the Timer3 event to print the requested message.

```
/**
 * Implementation for Blink application.  Toggle the red LED when a
 * Timer fires.
 **/

#include "Timer.h"

module BlinkC @safe()
{
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Timer<TMilli> as Timer3;
  uses interface Leds;
  uses interface Boot;
}
implementation
{
  event void Boot.booted()
  {
    call Timer0.startPeriodic( 250 );
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
    call Timer3.startPeriodic( 100 );
  }

  event void Timer0.fired()
  {
    dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
```

```
    call Leds.led0Toggle();
  }

  event void Timer1.fired()
  {
    dbg("BlinkC", "Timer 1 fired @ %s \n", sim_time_string());
    call Leds.led1Toggle();
  }

  event void Timer2.fired()
  {
    dbg("BlinkC", "Timer 2 fired @ %s.\n", sim_time_string());
    call Leds.led2Toggle();
  }

  event void Timer3.fired()
  {
    dbg("BlinkC", "I am Timer 3 and I have the shortest period!\n");
  }
}
```

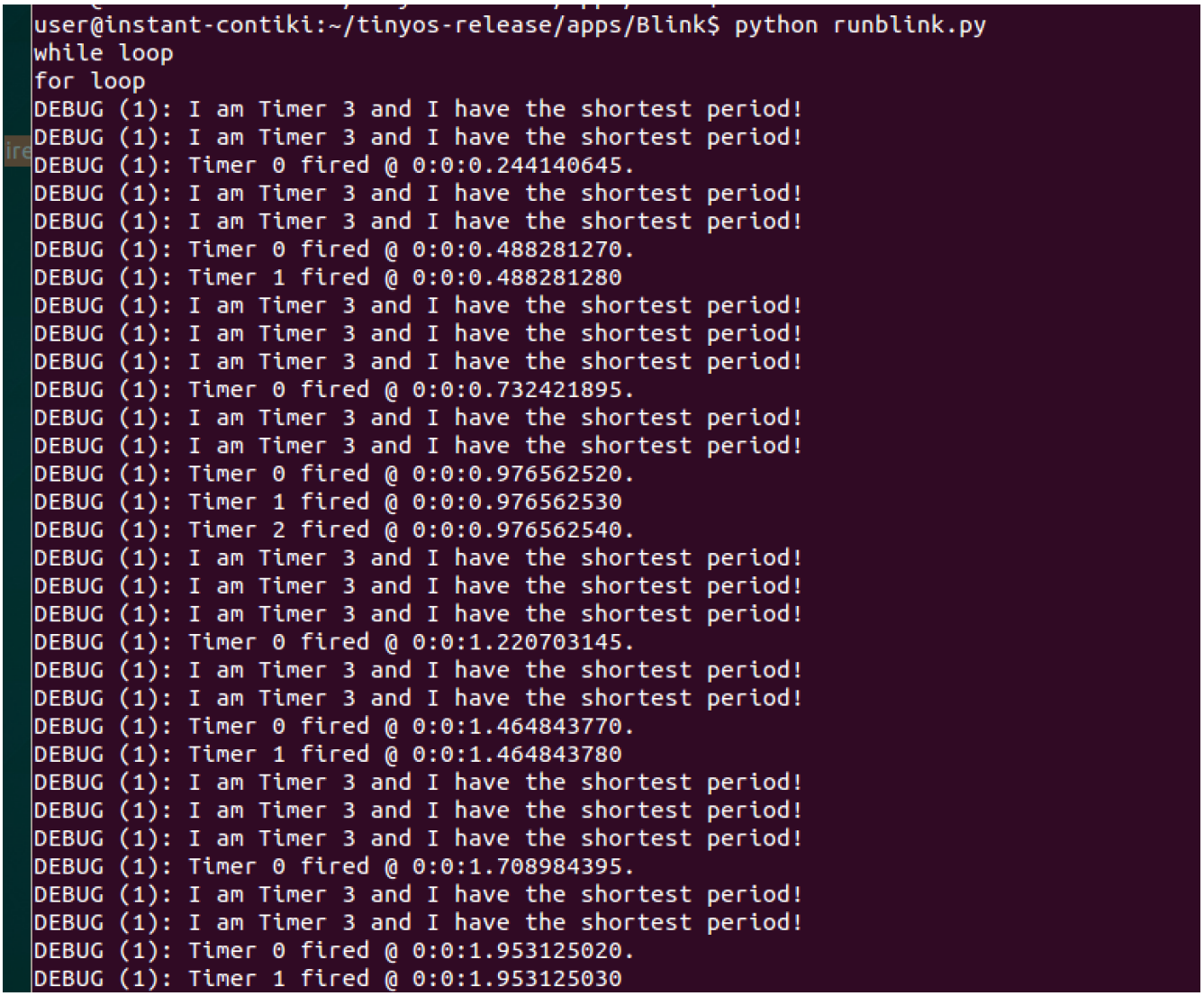
runblink.py

Have the simulation run for 2000 events by changing the runNextEvent range from 0 – 100 to 0 – 2000.

```
#!/usr/bin/python
from TOSSIM import *
import sys
t = Tossim([])
r = t.radio()
t.addChannel("BlinkC", sys.stdout)
m = t.getNode(1)
m.bootAtTime(100)
print "while loop"
while (m.isOn() == 0):
  t.runNextEvent()
print "for loop"
for i in range(0, 2000):
  t.runNextEvent()
```

Sample Run

Only shows beginning and end of run.



Task 2 Running of Blink Application with 2000 Events - Part 1

...

```
DEBUG (1): Timer 1 fired @ 0:0:20.507812530
DEBUG (1): Timer 2 fired @ 0:0:20.507812540.
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): Timer 0 fired @ 0:0:20.751953145.
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): Timer 0 fired @ 0:0:20.996093770.
DEBUG (1): Timer 1 fired @ 0:0:20.996093780
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): Timer 0 fired @ 0:0:21.240234395.
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): Timer 0 fired @ 0:0:21.484375020.
DEBUG (1): Timer 1 fired @ 0:0:21.484375030
DEBUG (1): Timer 2 fired @ 0:0:21.484375040.
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): Timer 0 fired @ 0:0:21.728515645.
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): I am Timer 3 and I have the shortest period!
DEBUG (1): Timer 0 fired @ 0:0:21.972656270.
DEBUG (1): Timer 1 fired @ 0:0:21.972656280
DEBUG (1): I am Timer 3 and I have the shortest period!
user@instant-contiki:~/tinyos-release/apps/Blink$
```

Task 2 Running of Blink Application with 2000 Events - Part 2

Environment

The environment for this test was MacOS Mojave, Version 10.14.4. VMWare Fusion Pro Version 10.1.6. The simulation was run in an Instant Contiki image (Ubuntu 14.04) downloaded from sourceforge. Setup of Contiki is as described in the class assignment.

Copyright

The original TinyOS Blink application is covered under the following copyright.

```
/*                                                      tab:4
 * Copyright (c) 2000-2005 The Regents of the University  of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
```

```
* modification, are permitted provided that the following conditions
* are met:
*
* - Redistributions of source code must retain the above copyright
*   notice, this list of conditions and the following disclaimer.
* - Redistributions in binary form must reproduce the above copyright
*   notice, this list of conditions and the following disclaimer in the
*   documentation and/or other materials provided with the
*   distribution.
* - Neither the name of the University of California nor the names of
*   its contributors may be used to endorse or promote products derived
*   from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
* FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
* THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
* INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
*
* Copyright (c) 2002-2003 Intel Corporation
* All rights reserved.
*
* This file is distributed under the terms in the attached INTEL-LICENSE
* file. If you do not find these files, copies can be found by writing to
* Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
* 94704.  Attention:  Intel License Inquiry.
*/
```

Summary

This concludes the report on the assignment.