**Cpr E 546 Spring 2019**

**Programming Assignment #1**

**Due Date: xx/xx/2018 (Tue) by xx:xx PM**

**Submit via Canvas**

**Objectives**

The purpose of this assignment is to introduce you to programming Contiki-NG applications. You will set up a Contiki-NG development environment, compile Contiki-NG applications, and install them on TI SensorTags or TelosB/Sky motes. You will learn some fundamental concepts of programming Contiki-NG applications through exercises in LED control, wireless communication, and RSSI sampling.

# Notes

1. Googling returns a fair amount of information about [Contiki-NG](http://www.contiki-ng.org/). The community also maintains a [wiki](https://github.com/contiki-ng/contiki-ng/wiki) and a [cheat sheet](https://contiki-ng.github.io/resources/contiki-ng-cheat-sheet.pdf).
2. Please cleanly format and clearly comment submitted code. Use of the [Contiki-NG Code Style](https://github.com/contiki-ng/contiki-ng/wiki/Code-style) is encouraged but not required.
3. We will use $CONTIKI to refer to the root directory of the Contiki-NG source code on your computer.
4. For questions or issues, contact [mlwymore@iastate.edu](mailto:mlwymore@iastate.edu) and CC [daji@iastate.edu](mailto:daji@iastate.edu).

# Procedure

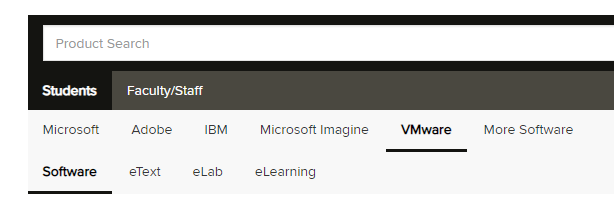
***Part I***

***What is Contiki-NG?***

Contiki-NG is a “next generation” fork of Contiki OS, and we will henceforth refer to Contiki-NG as simply “Contiki.” Contiki is an operating system for the IoT. More specifically, it is a set of open-source code libraries and a build system designed to simplify and standardize coding for embedded, networked devices. Contiki provides APIs for common embedded system tasks such as reading sensors, using timers, lighting up LEDs, and communicating over a wireless radio. Contiki even provides a thread/process framework. After you write your Contiki application, you use Contiki’s build system to *cross-compile* your code on your laptop or desktop. This gives you a binary image that can be loaded onto your target hardware. This image runs automatically when the microcontroller boots. Contiki comes packaged with a variety of support tools, including Cooja, a GUI-based WSN simulator that uses Contiki code. Contiki has an active developer community and is frequently used for WSN and IoT research.

***Set up your Contiki environment.***

You need a Linux environment to write Contiki applications. You can either use a Linux machine, or you can use a Linux virtual machine (VM) if you have a Windows or MacOS laptop/desktop. These steps will assume you are using a VM – if you’re instead using Linux on “bare metal,” you can skip the steps related to VMs. If you are going to install a VM, be aware that you will need some hard drive space – 20 GB at least. These steps will assume that you are using Ubuntu Linux as your Linux distribution, and may need to be modified if you are using a different distribution.

1. Install a hypervisor, which is a program that runs VMs. The OS that runs the hypervisor is called the “host,” and the VM is called the “guest.” We recommend that you use one of the professional hypervisor programs available to ISU Engineering students.
   1. Go to [this site](https://it.engineering.iastate.edu/software/students/) and click on the link for VMWare Products. You will be prompted to log in with your ISU credentials.
   2. Under the “Students” tab, choose the “VMWare” subtab. 
   3. If your host OS is Windows (or Linux), choose the latest version of VMWare Workstation. If your host OS is MacOS, choose the latest version of VMWare Fusion.
   4. Follow the steps to download and install the software. You will need the Serial Number that shows up on the Downloads page. This is the product key that you will enter when you open the software for the first time.
2. Install a Linux VM (guest OS) on your hypervisor.
   1. Download an image at Ubuntu Desktop’s [download page](https://www.ubuntu.com/download/desktop). We recommend downloading the latest Long Term Support (LTS) release, which is currently 18.04. It should be a 64-bit image.
   2. Open your hypervisor program (VMWare Workstation or VMWare Fusion).
   3. Follow the steps to install a new VM in the hypervisor. When prompted, select the Ubuntu Desktop image (.iso file) that you downloaded. You will be prompted to choose a username and password, which you will use to log in to Linux and gain admin access. The process of installing Linux may take a while. Make sure your computer does not go to sleep.
3. Boot up your VM and install packages. These steps should all be performed on your Linux OS.
   1. We recommend resizing the VM window to be as large as possible, or running the VM in fullscreen mode. We also recommend choosing display settings in Linux so the VM fits in the window. This will save you a lot of pain, as you will be using this VM frequently.
   2. If you’re prompted to update software, we recommend doing so. You may need to restart Linux after updates are applied.
   3. Open Terminal. If you don’t know how to find Terminal, press the Windows/CMD key to bring up a search bar.
   4. In Terminal, install git, a version control tool, with the command: sudo apt install git  
      You will be prompted to enter a password – this is the password for your Linux user.
   5. Install gcc with the command: sudo apt install gcc
   6. Install make with the command: sudo apt install make
   7. Install ant with the command: sudo apt install ant
   8. Install srecord with: sudo apt install srecord
   9. Install python with: sudo apt install python
   10. ONLY IF you will be using TI SensorTags, install the ARM compiler.
       1. Open Firefox and go to the ARM GNU-RM [downloads page](https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads).
       2. Download the latest Linux 64-bit version of the GNU Arm Embedded Toolchain. Choose to save the file.
       3. We will install the toolchain in your home directory. In Terminal, navigate to your home directory with: cd
       4. Create a new directory called arm with: mkdir arm
       5. Navigate to the new directory with: cd arm
       6. Extract the toolchain to this location with:  
          tar xjf ~/Downloads/gcc-arm-none-eabi-XXX-linux.tar.bz2  
          The XXX should be replaced with the specific name of the toolchain tarball you downloaded. Note that you can use tab completion, e.g., type ~/Downloads/gcc- and then press tab and the rest of the name of the file will be filled in for you.
       7. The previous step created a new directory. The name of this directory depends on the version of the toolchain. Enter the new directory, and then enter the bin subdirectory. Use the command pwd to get the full path of this directory; it should be something similar to: /home/mlwymore/arm/gcc-arm-none-eabi-8-2018-q4-major/bin
       8. We need to add this path to our $PATH environment variable, so that our shell knows where to search for the executables in this directory. You can do this in your .profile file. Open your .profile for editing with: gedit ~/.profile &
       9. At the end of this text file, add the below line, replacing the path with your actual path.  
          PATH="/home/XXX/arm/gcc-arm-none-XXX/bin:$PATH"  
          Save and close this file.
       10. For changes to take effect, you need to log out and log back in. In the upper-right corner, click the power symbol, then your user name, and then Log Out. Log back in again. Open Terminal. If the changes worked, the command which arm-none-eabi-gcc should return the path to that program. If nothing is returned, double check that you used the correct path in your .profile.
   11. Regardless of which mote type you are using, install the MSP430 compiler using the command: sudo apt install gcc-msp430
4. On Linux, clone the Contiki-NG git repository.
   1. In Terminal, navigate to your home directory with the following command: cd
   2. Clone the git repository with the following command:  
      git clone https://github.com/contiki-ng/contiki-ng.git
   3. This creates a directory called contiki-ng. We will refer to this directory as the “root” directory of Contiki, or $CONTIKI. Navigate into this directory with: cd contiki-ng
   4. Use git to also download Contiki’s submodules with:  
      git submodule update --init --recursive

***Compile a Contiki application and run it on hardware.***

In Terminal, use the cd command to navigate to $CONTIKI/examples/hello-world. Also open the Files application and navigate to this directory. You should see that it contains a README, a Makefile, and hello-world.c.

In Terminal, compile the hello-world application. If you are using the SensorTag platform, compile it the following command:  
 make TARGET=cc26x0-cc13x0 BOARD=sensortag/cc2650 hello-world

If you are using TelosB/Sky motes, compile it instead with this command:

make TARGET=sky hello-world

Note that the values for TARGET (and BOARD) depend on which hardware platform you’re using. These variables tell the Contiki build system what platform-specific code to include in the binary image, allowing Contiki to support a wide variety of devices.

After you run make, you’ll see a new directory called build. If you go into this directory, you will eventually find the compiled image files for your target platform. They can have a variety of formats (.bin, .hex, .elf). Let’s upload one of these image files to your hardware. The process for this is currently different for SensorTags and TelosB/Sky motes.

ONLY IF you have TelosB/Sky motes, do the following:

1. Plug your mote into a USB port on your development machine.
2. If you are using a VM and you get a popup asking if you want to connect the mote to the host or the guest, choose to connect the mote to the guest (Linux). Otherwise, use the hypervisor’s dropdown menus to virtually connect the mote to the VM.
3. In Terminal on Linux, you should now be able to see a USB device for your mote. Verify this using the command: ls -l /dev/ttyUSB\*
4. Open up permissions for the USB device using this command: sudo chmod 666 /dev/ttyUSB\*
5. Navigate to $CONTIKI/examples/hello-world.
6. Upload the program to the mote with: make TARGET=sky hello-world.upload

ONLY IF you have SensorTags, you will need to use a separate flash programming tool to upload the image to the SensorTag and must do the following:

1. You currently need a Windows machine for these steps. If your host OS is not Windows or you do not have access to another Windows machine, you can install a Windows VM on your hypervisor. You can obtain a Windows image by going to [this site](https://it.engineering.iastate.edu/software/students/) and selecting “Other Microsoft Products.” After you log in with your ISU credentials, on the Microsoft tab, pick Windows 10. After checking out, you will arrive at a page titled "Upgrading to Windows 10 Education." From here, select the link for Microsoft's ISO Software Download Site. From here, choose your language and select the 64-bit download. This is the Windows image (.iso file). Install this image on a new VM in your hypervisor. The Product Key that you will need is all the way back on one of the original download pages. The Windows version is Windows 10 Education.
2. On Windows, go to [TI’s cloud dev tools site](https://dev.ti.com/). This site seems to work fairly well in Chrome. Somewhere during this process, you will need to log in using a myTI account. Create one if you don’t already have one.
3. If you have not been here before, in the device detection sidebar on the left, you will be prompted to install TI’s cloud agent and browser extension. Follow the steps to install these things.
4. Plug a SensorTag into the Windows machine.
   1. To do this, you must take the SensorTag out of its red case and attach a Debugger DevPack. There are two connectors on the SensorTag board that line up with the connectors on the DevPack, making the boards stack together. Be careful not to damage the pins when connecting/disconnecting a DevPack.
   2. The DevPack has a micro USB port. Connect a cable from that port to the Windows machine. The DevPack will show up on the OS as an XDS110 debugger.
   3. If the Windows machine is a VM, you will need to virtually connect the USB device to the VM. If there is no popup when you plug in the SensorTag, you can accomplish this through the hypervisor’s dropdown menus.
5. The debugger should now show up in the device detection sidebar on the TI cloud tools site. Open the cloud app called UniFlash. The debugger should also show up in Detected Devices here.
6. In UniFlash, select the debugger. If you have to manually select a “device,” choose CC2650F128 and select the XDS110 for the “connection.” Click Start to begin a flashing session.
7. On the Program tab, click Browse to upload an image to flash. The image you want is hello-world.bin, so you will need to copy that over to the Windows machine from the build directory in $CONTIKI/examples/hello-world on your Linux machine. If either Linux or Windows is a VM, you should be able to copy the file between the host and guest using copy/paste. Another easy solution is to email the file to yourself.
8. After you’ve selected the correct Flash Image, click Load Image. This will upload the image to the SensorTag.

***Obtain the serial output from the device.***

After you've uploaded the image to the device, the Contiki application will immediately begin to run. To view the output printed by the device over the serial port (via USB), you can use the "make login" command provided by Contiki. Connect your device to Linux. Then, open up permissions for the USB device.

For SensorTag: sudo chmod 666 /dev/ttyACM\*

For TelosB/Sky: sudo chmod 666 /dev/ttyUSB\*

Then, from the application directory (e.g. $CONTIKI/examples/hello-world/), make the "login" application.

For SensorTag:

make TARGET=cc26x0-cc13x0 BOARD=sensortag/cc2650 PORT=/dev/ttyACM0 login

For TelosB/Sky:

make TARGET=sky login

For the hello-world application, you should see the device repeatedly printing the string "Hello world" to the terminal. Take a screenshot of this output. Note that "make login" is a shortcut to the serialdump application, which can be found in $CONTIKI/tools/serial-io/. Instead of "make login," you can use serialdump directly. You can also get the serial output with many other programs, such as PuTTY for Windows or CoolTerm for MacOS.

***Run an example application in the Cooja simulator.***

Cooja is a WSN simulator packaged with Contiki. Cooja uses the compiled Contiki code in its simulations, making it easy to develop an application with Cooja and then deploy it to hardware.

1. Navigate to $CONTIKI/tools/cooja/.
2. Build and run Cooja with the command: ant run
3. ONLY IF you get an error about java.xml.bind, this is because Cooja currently relies on some deprecated Java APIs. To get around this:
   1. Install Java 8 with the command: sudo apt install openjdk-8-jdk
   2. Switch to using Java 8 with the command: sudo update-alternatives --config java  
      When prompted, choose the option for Java 8.
4. Once in Cooja, go to File->New simulation…
5. Click "Create."
6. First you need to create a "mote type." There is one mote type for each hardware/software combination you will simulate. Go to Motes->Add motes->Create new mote type->Sky mote…
7. Browse for a Contiki process / Firmware. Go to $CONTIKI/examples/rpl-udp and select udp-server.c.
8. Click "Compile."
9. After compilation (which should be successful), click "Create." Then click "Add motes."
10. This adds one new mote of this type in a random location. You can see the mote in the Network view. Click on it to see its communication range. Drag it around to change its location.
11. Now create another Sky mote type. For the Firmware, use udp-client.c from the same directory. Create several of these motes, e.g. three. These motes will self-organize into a RPL routing tree with the server mote at the root.
12. In the Simulation Control view, click Start to begin the simulation. Click Pause to pause the simulation, and Reload to return the simulation to its initial state.
13. Serial output from the motes is shown in the Mote Output view. You should see the motes print boot information, and then you should see request/response messages between the UDP server mote (i.e. mote ID:1) and the client motes. Take a screenshot of this output.
14. Radio activity can be seen in the Timeline view. Solid gray indicates the radio is on. A blue box indicates packet transmission. A green box indicates packet reception. A red box indicates radio interference (i.e. a failed reception).

**Exercise 1**

Submit two screenshots: the mote output from hello-world.c, and the Mote Output view from your Cooja simulation.

***Part II***

***What are processes?***

As in a desktop OS, a *process* in Contiki represents a running instance of code. An application may use multiple processes, but in Contiki, these processes are run in a single thread. The programmer must tell the application where the process begins, where it ends, and where the process pauses, such as to wait for an event. While a process is paused, other processes can run. Contiki maintains a process queue that determines which process runs next. Processes can interact with each other using the events API. Processes can also be interrupted for real-time events; when the interrupt is finished, control returns to the process. Read more about Contiki processes [here](https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-Processes-and-events).

***Run the leds-example.c application.***

ONLY IF you are using TelosB/Sky motes, before doing this section, perform the following steps:

1. Open $CONTIKI/arch/platform/sky/sky-def.h for editing.
2. After the definition of LEDS\_CONF\_YELLOW, add the following new line:  
   #define LEDS\_CONF\_ALL 0x7F
3. Save and exit the file.

Using what you learned in Part I, compile the leds-example.c application in $CONTIKI/examples/dev/leds/. Upload the application to a mote. Observe the behavior – what pattern do you see? How often do the LEDs change? Now open leds-example.c and examine the code. There is a process here called “leds\_example.” Look at the code and figure out the following: Where is the process declared? Where is the process implemented? Which code belongs to this process? This process automatically starts on boot – which line makes this happen?

The timing of the pattern is achieved using the event timer (etimer). Read about Contiki’s timers [here](https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-Timers). The etimer is set for CLOCK\_SECOND, which is the number of clock ticks in a second. This means that one second after the timer is set, a timer event is fired. The call to PROCESS\_YIELD() pauses the process and allows Contiki to run any other processes on the queue. After the timer event fires, Contiki delivers the event to the leds\_example process. This means that control returns to the process at the PROCESS\_YIELD() call, with the variable ev populated with the event type (PROCESS\_EVENT\_TIMER) and the variable data populated with a pointer to the etimer struct. The process then checks the value of a counter to determine what LEDs to light, increments the counter, and resets the etimer before yielding control again. Before continuing, you should understand this control flow.

*Note: If you are using TelosB/Sky motes, LEDS\_LEGACY\_API will be 1, so the code between #if and #endif will not be included in your application.*

***What is a HAL?***

Contiki makes extensive use of hardware abstraction layers (HALs). A HAL gives the programmer an API to use hardware components, hiding the messy details of exactly how the hardware is accessed. This makes life easier for the programmer, and also allows Contiki to support many hardware platforms. In leds-example.c, the LEDs are accessed using the LED HAL, which is documented [here](https://contiki-ng.readthedocs.io/en/latest/_api/group__leds.html). To use a HAL, the application must #include the appropriate header file – in this case, dev/leds.h.

**Exercise 2**

Write a description of the control flow of the leds-example.c application. You may draw a figure if helpful. Your description should answer the following questions:

1. Which code in leds-example.c executes first?
2. When is leds\_set() first called?
3. When is leds\_set() called for the second time?
4. What is the purpose of the PROCESS\_YIELD() statement?
5. What event causes control to return to the process after PROCESS\_YIELD() has been called?

# What to Turn In

**For Exercise 1 (50 pts):** Your two screenshots, as .jpgs, .pngs, or PDFs.

**For Exercise 2 (50 pts):** A PDF with your description, called **pa1\_ex2.pdf**.