



**KTH Electrical Engineering**

# Getting started with TinyOS

at the Automatic Control Lab

AITOR HERNÁNDEZ  
ALEJANDRO MARZINOTTO  
MANI AMOOZADEH

Stockholm November 9, 2012

---

TRITA-EE 2011:XXX

Version: 1.0



# Contents

<b>Contents</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Installing TinyOS on Ubuntu</b>	<b>3</b>
2.1 Ubuntu installation . . . . .	3
2.2 TinyOS installation . . . . .	3
2.2.1 Installation Steps . . . . .	4
<b>3 Installing TinyOS on Windows</b>	<b>7</b>
3.1 Introduction . . . . .	7
3.2 Requirements . . . . .	7
3.3 Installation . . . . .	7
3.3.1 Step 1: Installing Java Development Kit (JDK) . . . . .	7
3.3.2 Step 2: Installing Cygwin . . . . .	8
3.3.3 Step 3: Installing TinyOS packages . . . . .	9
3.3.4 Step 4: Fixing Files Post-Install . . . . .	10
<b>4 Your first TinyOS program</b>	<b>11</b>
<b>5 Examples and tutorials</b>	<b>13</b>
5.1 Basics for Automatic Control department . . . . .	13
5.1.1 printf . . . . .	13
5.1.2 Serial Communication . . . . .	14
5.1.3 Timers . . . . .	15
5.1.4 ADC . . . . .	17
5.1.5 DAC . . . . .	20
5.1.6 General purpose I/O . . . . .	21
5.1.7 User Button usage . . . . .	22



---

# Introduction

This document describes the installation procedure of TinyOS on Ubuntu and Windows.

In chapter 2 the installation procedure on Ubuntu is described. Section 2.1 provides some guidelines on installation of the Ubuntu OS. And TinyOS installation procedure is described in section 2.2.

Chapter 3 encompasses, some guidelines about the installation environment in section 3.1, the installation requirements in section 3.2, and the procedure for installing TinyOS on Windows in section 3.3.

Chapter 4 provides instructions for implementing a TinyOS program on a mote.

And chapter 5 describes the basic code examples for different components of TinyOS.



---

# Installing TinyOS on Ubuntu

This chapter describes the installation procedure of Ubuntu OS, and TinyOS from official repository.

**CAUTION:** *As many symbols may change or disappear when copying and pasting from the pdf file, resulting in wrong commands and thus defective installation, you are advised to write the commands yourself instead.*

## 2.1 Ubuntu installation

The image of Ubuntu can be downloaded from <http://www.ubuntu.com/download/ubuntu/download>. *Note:* It is preferable to download the recommended version of Ubuntu from the website to avoid vague problems in the future. So, don't worry if you would install a 32-bit version on a computer with a 64-bit processor. Once downloaded, follow the intallation documentation at <https://help.ubuntu.com/community/Installation> to install the operating system.

## 2.2 TinyOS installation

The official TinyOS documentation on istallation can be found at [http://docs.tinyos.net/tinywiki/index.php/Getting\\_started](http://docs.tinyos.net/tinywiki/index.php/Getting_started). Assuming Ubuntu OS installed by following previous chapter, you can follow the TinyOS installation guide using SVN/GIT repository described at [http://docs.tinyos.net/tinywiki/index.php/Installing\\_from\\_SVN/GIT](http://docs.tinyos.net/tinywiki/index.php/Installing_from_SVN/GIT) which is also explained in the following numbered steps.

Before starting the installation steps PLEASE NOTE that

- To execute any command, you need to open a terminal window and make sure you are in the home directory.
- You can always detect your current directory executing `$ pwd` command.
- You can always go to your home directory executing `$ cd` command.
- You have to confirm all of the continuation/verification prompts. Be aware of capital/small letters for yes and no.

### 2.2.1 Installation Steps

1. Specify "lucid" as the distribution you are going to use and update it by

```
$ sudo apt-add-repository "deb http://tinyos.stanford.edu/tinyos/dists/ubuntu
lucid main"
and
$ sudo apt-get update
```
2. Install the nesc compiler.

```
$ sudo apt-get install nesc
```
3. Install the cross tools, Debian MSP430

```
$ sudo apt-get install msp430-tinyos
and Debian AVR
$ sudo apt-get install avr-binutils-tinyos msp430-gcc-tinyos msp430-libc-tinyos
```
4. Install g++

```
$ sudo apt-get install g++
```
5. Checkout the TinyOS 2.x source tree or change the path to the directory where you want the TinyOS to be installed (`/opt` in this example):

```
$ cd /opt
$ mkdir -p local/src
$ cd local/src
```
6. Install svn

```
$ sudo apt-get install subversion
```
7. Run the following command to get the latest version of TinyOS from the official repository:

```
$ svn checkout http://tinyos-main.googlecode.com/svn/trunk/ tinyos-2.x
```



8. Compile the TinyOS tools

```
$ sudo apt-get install automake
$ cd tinyos-2.x/tools
$ ./Bootstrap
$ ./configure prefix=$HOME/local
$ make all
$ make install
```

9. Open your `.bashrc` by executing the following command:

```
$ cd
$ gedit .bashrc
and add following lines at the end of the file:
export PATH=$HOME/local/bin:$PATH
export TOSROOT=$HOME/local/src/tinyos-2.x
export TOSDIR=$TOSROOT/tos
export MAKERULES=$TOSROOT/support/make/Makerules
export CLASSPATH=$TOSROOT/support/sdk/java/tinyos.jar:.
export PYTHONPATH=.:$TOSROOT/support/sdk/python:$PYTHONPATH
export PATH=$TOSROOT/support/sdk/c:$PATH
```

10. Save the file and relaunch the terminal.
11. To check out the integrity of your installed TinyOS, execute the following commands to compile the file `BlinkAppC`.

```
$ cd
$ cd opt/local/src/tinyos-2.x/apps/Blink
$ make telosb
```



---

# Installing TinyOS on Windows

## 3.1 Introduction

The standard environment to program the motes is under Linux's installation of TinyOS, however all the LabVIEW code provided for communication with the motes works only under Windows. The following tutorial provides a method to install TinyOS programming tools under Windows, to make the process of debugging and testing code more efficient.

This tutorial is a compilation of different methods and tutorials provided by different people. It will provide a fully functional TinyOS development system in Windows.

## 3.2 Requirements

To follow this tutorial, it is recommended to run a 32 bit version of Windows 7; however it should work under the other Windows environments as well. You need internet connection to download the installers and to test if the installation was successful you will need a mote.

## 3.3 Installation

### 3.3.1 Step 1: Installing Java Development Kit (JDK)

To install JDK 1.6 (32 bits), copy and paste the following link into your browser.

<http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u25-download-346242.html>

Download and install the following *.exe* file from the table *Java SE Development Kit 6 Update 25*

Windows x86	76.66 MB	jdk-6u25-windows-i586.exe
-------------	----------	---------------------------

If you have a newer version of java already installed or you have a 64 bit version already installed, the installer will complain at the end but will install this version anyway.

You may need to manually add the location of the java bin folder `C:\Program Files\Java\jdk1.6.0_25\bin` to your Windows Path variable.

To do this, right-click on My Computer and click Properties. In the System Properties window, click on the Advanced tab. In the Advanced section, click the Environment Variables button. Finally, in the Environment Variables window, highlight the Path variable in the Systems Variable section and click the Edit button. Add the path to the Java bin folder to the end (don't delete what's in the variable, just add to it separating folders with semicolons).

### 3.3.2 Step 2: Installing Cygwin

Download the **setup.exe** file from this website: <http://cygwin.com/install.html> and run it. Click Next to proceed.

Choose, "Install from Internet". When it asks for "Root Directory" then select `c:\cygwin` (Note: make sure the path does not contain any space), and install for all users.

When it asks for "Local Package Directory" then select `c:\cygwin-files`.

Choose "Direct Connection" and click Next.

Choose a download site from the list and click Next. Setup will download a list of available packages, and you can see a windows like figure 3.1. Click on the "Default" field next to the "All" category to change it into "Install". By this the setup will download and install all the packages.

Click Next again to start the download process. After the packages are downloaded, setup will automatically begin to install them. The process of downloading/installing takes some time. Be advised that this will download and install hundreds of megabytes to your computer.

In the last window, tick both "Create icon on Desktop", and "Add icon to Start Menu", and then click Finish to close the setup program.

Now you can see a shortcut icon called "Cygwin Terminal" on your desktop. Right click on this icon, and go to properties. Under the Compatibility Tab, check the box

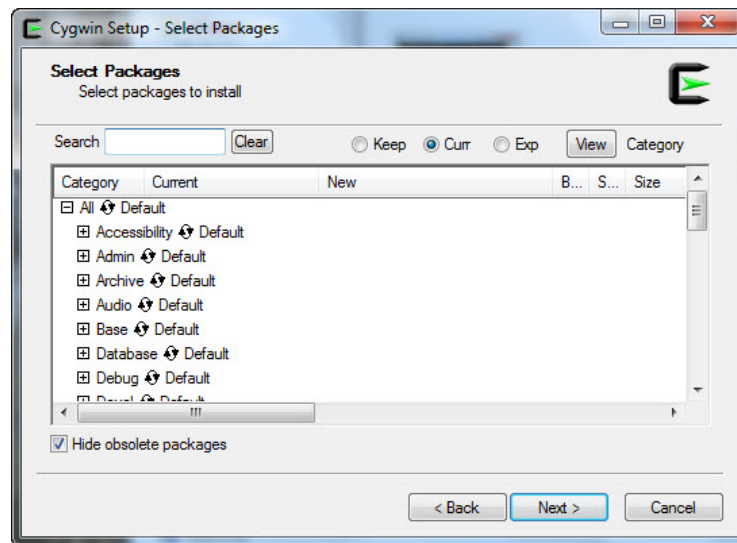


Figure 3.1: Select Packages Window in Cygwin

that says *Run this program as an administrator* and press OK. Start Cygwin as you normally would, notice that now it is going to run as administrator so it may ask for confirmation depending on your Windows security settings.

### 3.3.3 Step 3: Installing TinyOS packages

Download the **TinyOS\_Files** folder from SVN <sup>1</sup>. This folder contains two sub-folders (Driver and RPM), and two files (install.sh and variables). Copy the folder **TinyOS\_Files** into `C:\cygwin\home\username`. In the cygwin terminal, type `ls` and confirm that **TinyOS\_Files** folder is listed. Then type `cd TinyOS_Files` and then `ls` to check inside the folder. Look at figure 3.2.

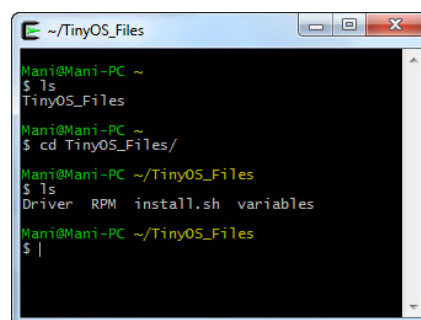


Figure 3.2: Checking the Contents of the Folder

<sup>1</sup>[http://kth-wsn.googlecode.com/svn/trunk/kth-wsn/apps.tutorials/GettingStarted/~TinyOS\\_Files/](http://kth-wsn.googlecode.com/svn/trunk/kth-wsn/apps.tutorials/GettingStarted/~TinyOS_Files/)

Now run the `install.sh` script by issuing the **sh install.sh** command, and wait for the script to install all the necessary RPM files.

Now in Windows explorer, navigate to `C:\cygwin\home\username\TinyOS_Files`, and open the `variables` file with a text editor (like notepad or notepad++). Copy the content of this file to the end of `.bash_profile` file which is in `C:\cygwin\home\username`. Close Cygwin terminal and relaunch it to load the new information contained in `.bash_profile`.

### 3.3.4 Step 4: Fixing Files Post-Install

There are a few problems with the default installation which will become apparent as soon as you walk through tutorials and try to compile code, run Java, etc. Follow the explained steps to fix such problems.

#### ⇒ The -32.dll problem

- In `C:\cygwin\lib\tinyos`, make copies of `getenv.dll` and `toscomm.dll` and name them `getenv-32.dll` and `toscomm-32.dll` respectively.
- Copy these four DLL files into `C:\Program Files\Java\jdk1.6.0_25\bin`

#### ⇒ The printf.h problem

- In `C:\cygwin\opt\tinyos-2.x\tos\lib\printf`
- make a copy of `printf.h` and name it `generic_printf.h`

#### ⇒ Compiling Tinyos Java

- In cygwin, Navigate to `C:\cygwin\opt\tinyos-2.x\support\sdk\java` and Type **make** to compile the Java environment.
- If you get an error about *javac* not found, double check your Windows PATH variable in System Properties. On 64-bit systems, if you install the 32-bit JDK, you will be in “Program Files (x86)” not in “Program Files”.

#### ⇒ Installing JNI library

- In any directory, type **tos-install-jni** (make sure everything is successful. some operations require admin privileges in Windows).
- If there is an error on Line 23 about “*file: command not found*”, be sure to select and install “*file*” in the Cygwin installer.

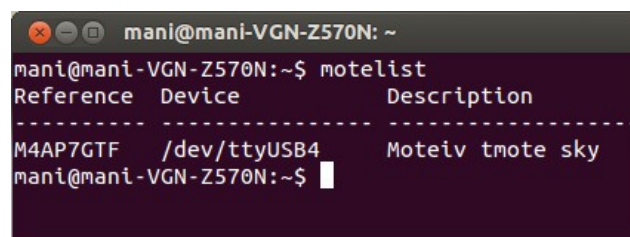
## Your first TinyOS program

Make your first program in TinyOS installing Blink application on a mote. This is done in two steps as following:

1. First of all, you should connect the mote into a USB port and identify the USB port address to which your mote is attached. To do so, enter the following command into the Ubuntu/Cygwin terminal:

```
$ motelist
```

The result in Ubuntu will be a table in which the USB port address is a term starting with `/dev` under the `Device` column. Figure 4.1 shows this.



```
man1@man1-VGN-Z570N: ~  
man1@man1-VGN-Z570N:~$ motelist  
Reference Device Description  
-----  
M4AP7GTF /dev/ttyUSB4 Moteiv tmote sky  
man1@man1-VGN-Z570N:~$
```

Figure 4.1: motelist Command in Ubuntu

In Windows the mote's ID and COM port number will be displayed. Figure 4.2 shows this. You may also detect the mote's USB port in windows via Control Panel-> Device Manager-> Ports->USB Serial Port (COMXX).

**Note:** USB port driver is installed on first use of the sensor node, using FTDI USB to serial chip. If you get `error, could not open key HKLM\SYSTEM\CurrentControlSet\Enum\FTDIBUS`, it means, that FTDI USB driver has not been loaded. To load it, plug in the sensor node, and the driver will be loaded automatically.

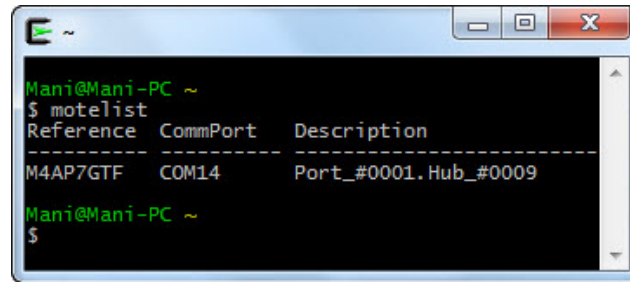


Figure 4.2: motelist Command in Cygwin

2. Go to the Blink application folder:

```
$ cd /opt/tinyos-2.x/apps/Blink/
```

and issue the following command where `<Port>` should be replaced by the USB port address term in Ubuntu or the serial port number "XX" minus 1 in Windows. The number "1" in the command is the address the mote will use in all it's communication and can be thought of as analogous to an IP address.

```
$ make telosb install.1 bsl, <Port>
```

For instance in Ubuntu to program the mote that is attached to USB4 in figure 4.1, the following command can be used:

```
$ make telosb install.1 bsl,/dev/ttyUSB4
```

and in Cygwin to program the mote that is attached to COM14 in figure 4.2, the following command can be used:

```
$ make telosb install.1 bsl,13
```

Once the above command is invoked, the application is first compiled to the telosb architecture (using a cross compiler), and the resulting files are stored in the current directory.

Next, the loader program is invoked (bsl), and the compiled binary is loaded to the mote attached to `<Port>`. While loading, you will see a host of lights blinking on the mote. Once the loading completes, the mote is reset automatically and the application loaded starts running on the mote.



---

## Examples and tutorials

This chapter describes the basic code examples for different components of TinyOS.

### 5.1 Basics for Automatic Control department

The source code for the examples described below can be found both on the KTH-WSN repository and in the `apps` folder of TinyOS installation. To obtain the latest version of the code, use the following command:

```
svn checkout http://kth-wsn.googlecode.com/svn/ kth-wsn-read-only
```

Also, some of the examples' source code is located under the `apps.tutorials/GettingStarted` folder.

#### 5.1.1 `printf`

The official tutorial on usage of `printf` library can be found at [http://docs.tinyos.net/tinywiki/index.php/The\\_TinyOS\\_printf\\_Library](http://docs.tinyos.net/tinywiki/index.php/The_TinyOS_printf_Library).

In order to use `printf` library add the following line to your program source code:

```
#include <printf.h>
```

and the following line to its Makefile:

```
CFLAGS += -I$(TOSDIR)/lib/printf
```

Now `printf()` can be called at any place in the code:

```
event void Boot.booted() {
```



the sending was successful, we toggle the red LED on the mote. We also set the `busy` variable to `TRUE` to block the serial port while it's busy sending the packet:

**NOTE:** The "--" symbol in the following code pieces represents continuation to the next line.

```
event void MilliTimer.fired() {
    if(busy == FALSE){
        TestSerialCommMsg* btrpkt = (TestSerialCommMsg*)--
        (call Packet.getPayload(&pkt, --
        sizeof(TestSerialCommMsg)));
        counter++;
        btrpkt->counter = counter;
        if (call UartSend.send[AM_TESTSERIALCOMMMSG]--
        (AM_BROADCAST_ADDR, &pkt, --
        sizeof(TestSerialCommMsg)) == SUCCESS) {
            call Leds.led0Toggle();
        }
        busy = TRUE;
    }
}
```

And finally, we check if the packet was successfully sent and set the `busy` to `FALSE` if so:

```
event void UartSend.sendDone[am_id_t id]--
(message_t* msg, error_t err) {
    if (&pkt == msg) {
        busy = FALSE;
    }
}
```

### 5.1.3 Timers

General description of different types of timers can be found at <http://www.tinyos.net/tinyos-2.x/doc/html/tep102.html>. In this example we will discuss timer interfaces. TinyOS provides following interfaces:

```
interface Counter<precision_tag, size_type>
interface Alarm<precision_tag, size_type>
interface BusyWait<precision_tag, size_type>
interface LocalTime<precision_tag>
interface Timer<precision_tag>
```

Standard `Timer.h` header file contains following precision tags:

```
typedef struct { int notUsed; } TMilli;
```

```
typedef struct { int notUsed; } T32khz;  
typedef struct { int notUsed; } TMicro;
```

## Timers

Timer is a synchronous interface meaning that all commands and events of this interface are synchronous. For below example, please refer to folder

`apps.tutorials/GettingStarted/TestTimers/`

First of all, we should include the `Timer.h` header file to be able to use timers:

```
#include <Timer.h>
```

and include the `TimerMilliC` components in our wiring file (`TestTimersC.nc`):

```
components new TimerMilliC();
```

and wire two timers:

```
App.MilliTimer -> TimerMilliC;
```

```
App.MilliTimer2 -> TimerMilliC;
```

Now, in our main program file (`TestTimersP.nc`), we start two timers. One periodic:

```
call MilliTimer2.startPeriodic(1000);
```

and one one-shot:

```
call MilliTimer.startOneShot(1000);
```

Now, if you program a mote with this program, you should see the red LED on the mote to turn on after one second from the start, and green LED turning on and off every second.

For every timer used in a program, there should be `<TimerName>.fired()` event implemented. In our case, we have:

```
event void MilliTimer.fired() {  
    call Leds.led0Toggle();  
}  
  
event void MilliTimer2.fired() {  
    call Leds.led1Toggle();  
}
```

## Alarms

Alarm interface, unlike the Timer interface, is asynchronous which means that it can interrupt the main execution routine. For this example, please refer to the code

at `$TOSROOT/apps/tests/TestAlarm`. Again, we include the header:

```
#include <Timer.h>
```

and wire the components (`BlinkC.nc`):

```
components new AlarmMilliC() as AlarmC;

BlinkM.Alarm -> AlarmC;
```

Now in the main program file (`BlinkM.nc`), we define a variable for delaying the alarm, turn on the green LED and start the alarm with the defined delay:

```
enum {DELAY_MILLI = 512};

event void Boot.booted(){
    atomic{
        call Leds.led1On();
        call Alarm.start( DELAY_MILLI );
    }
}
```

Now once the alarm is fired, we delay the alarm once again and toggle the LED:

```
async event void Alarm.fired(){
    atomic{
        call Alarm.startAt( call Alarm.getAlarm(), DELAY_MILLI );
        call Leds.led0Toggle();
    }
}
```

If you program a mote with this program, you should see the green LED turning on and off every ~500ms.

#### 5.1.4 ADC

Analog-to-Digital Converter (denoted further as *ADC*) is an interface that converts an input analog voltage or current to a digital number proportional to the magnitude of the voltage or current. General information about Analog-to-Digital Converters (ADCs) implementation in TinyOS can be found at <http://www.tinyos.net/tinyos-2.x/doc/html/tep101.html>.

Tmote Sky is based on Texas Instruments MSP430 microprocessor and 5 pins on the mote's extension header can be used as ADC inputs with 12-bit precision.

For this example, please use the code at `apps.tutorials/GettingStarted/TestADC`. In this example program, we will use a periodic timer which will fire every second, call ADC to read values from 4 channels and send the captured values to the serial port. First of all we include necessary headers:

```
#include <Timer.h>
```

```
#include <Serial.h>
```

Now we wire necessary components in the wiring file (`TestAdcC.nc`):

```
components new Msp430Adc12ClientAutoRVGC() as AutoAdc;
App.Resource -> AutoAdc;
AutoAdc.AdcConfigure -> App;
App.MultiChannel -> AutoAdc.Msp430Adc12MultiChannel;
```

Since ADC uses the shared resources, we have to take care of resource arbitration (more info can be found at <http://www.tinyos.net/tinyos-2.x/doc/html/tep108.html>). This can be achieved by using `Resource` component.

Now we declare necessary variables:

```
message_t pkt; - the packet which will be sent to the serial port;
bool busy = FALSE; - boolean variable used to block the serial port;
uint16_t buffer[4]; - buffer containing the ADC measure data.
```

The ADC configuration is contained in a special variable of type

```
msp430adc12_channel_config_t:
```

```
const msp430adc12_channel_config_t config = {
    INPUT_CHANNEL_A0, REFERENCE_VREFplus_AVss, REFVOLT_LEVEL_1_5,
    SHT_SOURCE_SMCLK, SHT_CLOCK_DIV_1, SAMPLE_HOLD_64_CYCLES,
    SAMPCON_SOURCE_SMCLK, SAMPCON_CLOCK_DIV_1
};
```

In this example, we use channel A0 (more channels will be added later), Vss as the reference voltage and 1.5V reference voltage level. For the full list of options refer to the header file `Msp430Adc12.h` found at `$TOSDIR/chips/msp430/adc12`. Now we request the shared resource for the ADC:

```
event void Boot.booted() {
    ...
    call Resource.request();
}
```

and once the resource has been granted, we define more channels to be used to read values from, call ADC configure procedure and start the periodic timer:

```
event void Resource.granted(){
    atomic {
        adc12memctl_t memctl[] = { {INPUT_CHANNEL_A1, REFERENCE_VREFplus_AVss},{IN
        if (call MultiChannel.configure(&config, memctl, 3, buffer, 4, 0) != SUCC
        call Leds.led0Toggle();
    }
}
```

```

    call MilliTimer.startPeriodic(1000);
}

```

The `MultiChannel.configure` command has following format:

```

async command error_t configure(const msp430adc12_channel_config_t *config,

```

where:

- `config` - main ADC12 configuration and configuration of the first channel;
- `memctl` - list of additional channels and respective reference in format `{INPUT_CHANNEL_NUMBER, REFERENCE_VOLTAGE}`;
- `numMemctl` - number of entries in the list;
- `buffer` - buffer to store the conversion results, it must have `numSamples` entries. Results will be stored in the order the channels were specified;
- `numSamples` - total number of samples. Note: `numSamples % (numMemctl+1)` must be zero. For example, to sample every channel twice use `numSamples = (numMemctl+1) * 2`;
- `jiffies` - sampling period in terms of clock ticks of "sampcon\_ssel" and input divider "sampcon\_id". Samples are taken equally-spaced in time iterating round-robin over the channels (different channels are not sampled simultaneously but one after another).

Now, once the timer is fired, we call the `getData()` command which will enforce the ADC to read the data:

```

event void MilliTimer.fired() {
    ...
    call MultiChannel.getData();
}

```

Once the data is ready, `dataReady()` event is fired where we can handle the data read by ADC. In our example, we copy the measure values to our `buffer` variable:

```

async event void MultiChannel.dataReady(uint16_t *buf, uint16_t numSamples){
    buffer[0] = buf[0];
    buffer[1] = buf[1];
    buffer[2] = buf[2];
    buffer[3] = buf[3];
    post sendData();
}

```

### 5.1.5 DAC

A digital-to-analog converter (DAC) is a device that converts a digital (usually binary) code to an analog signal (current, voltage, or electric charge). It's operation is opposite to ADC. For this example, please use the code at `apps.tutorials/GettingStarted/TestDAC`.

Tmote Sky mote has two 12-bit DAC channels which are connected to pin on small extension header. The pin allocation can be found in [?] on page 21 (pins marked as DAC0 and DAC1). Since there is no unified interface for TinyOS for DAC, we use MSP430's instructions to work with the DAC. The output voltage from the DAC channels can be calculated by following formula:

$$V = \frac{2.5}{4096} \cdot DAC$$

where DAC - the number in the range 0...4095.

`TestDacP.nc` contains a sample program to test Tmote Sky's DAC. In this program, we run a periodic timer which fires every second and increases voltage on the DAC0 pin. First, we initialize the DAC pin, set it's value to 0 and start the timer:

```
event void Boot.booted() {
  atomic {
    DAC12_OCTL = DAC12IR + DAC12AMP_5 + DAC12ENC;
  }
  DAC12_ODAT = 0;
  call sMilliTimer.startPeriodic(1000);
}
```

here:

- DAC12\_OCTL is the DAC pin control register;
- DAC12IR is the DAC input range register. DAC12IR = 0 multiplies the input value by 3, DAC12IR = 1 by 1 (default value);
- DAC12AMP\_X is the DAC amplifier register. Its value specifies voltage settling time vs power consumption. As higher the value as faster is the voltage is settled and as higher is the current consumption. In our example, the register value is set to 5 (by using DAC12AMP\_5 register);
- DAC12ENC is the DAC enable conversion register. DAC12ENC = 0 disables DAC, DAC12ENC = 1 enables DAC (default value);
- DAC12\_XDAT is the DAC value register. In our case, we set the value of the DAC0 pin to 0 by using DAC12\_ODAT register.

Next, every time the timer is fired, we increase the DAC0 pins value by 128, check we exceed the 4095 value limit and set the new value to the pin:



```

event void MilliTimer.fired() {
    ...
    counter += 128;
    counter = counter % 4095;
    DAC12_0DAT = counter;
}

```

If you program a mote with this program, you should see the red LED turning on and off every second, and the voltage on pin DAC0 slowly increasing from 0V to 2.5V.

### 5.1.6 General purpose I/O

Tmote Sky has several pins which can be used as general purpose I/O. The pin allocation can be found in [?] on page 21 (pins marked as GI00 to GI05). These pins can be used both as digital inputs and outputs. For this example, please use the code at `apps.tutorials/GettingStarted/TestGPIO`.

First, we wire the necessary components. For this purpose, we use `Msp430GpioC` component (`TestGpioC.nc`):

```

components HplMsp430GeneralIO;
components new Msp430GpioC() as PinA;
PinA -> HplMsp430GeneralIO.Port26;
App.PinA -> PinA;

```

```

components new Msp430GpioC() as PinB;
PinB -> HplMsp430GeneralIO.Port23;
App.PinB -> PinB;

```

In this example, we use two pins, GI02 and GI03, and one will be configured as an input and another as an output. The pins are connected to the port on the MSP430 microcontroller and the appropriate connections can be found in [?] on page 7.

Next, in our main program file (`TestGpioP.nc`), we initialize the pins. The pin `PinA` (GI02) is configured to be an input and the pin `PinB` (GI03) is configured to be an output:

```

event void Boot.booted() {
    call PinA.makeInput();
    call PinB.makeOutput();
}

```

Now, every time the timer is fired, we toggle the output pin (from HI level to LO and vice versa), check the state of the input pin: if the pin is in HI state, we turn on the red LED; if the pin is in LO state - we turn off the LED:

```

event void MilliTTimer.fired() {
    call PinB.toggle();
    if (call PinA.get()) {
        call Leds.led0On();
    } else {
        call Leds.led0Off();
    }
}

```

If you program a mote with this program, you should be able to see the voltage on the output pin to change from 0V to 3.3V every second. It is not easy to test the input pin; for this purpose, we recommend to use a power supply, set its output voltage to 3.3V and connect its negative probe (usually black cable) with Tmote Sky's GND pin and the positive probe (usually red cable) with the input pin to set the pin in HI state. Remove the positive probe to set the pin to LO state. Now, every time you set the pin in HI state, you should see the red LED turning on and turning off when you remove the probe.

### 5.1.7 User Button usage

The Tmote Sky mote is equipped with a button which can be used as an interrupt input. This button is labeled as **USER** on the board. For this example, please use code at [apps.tutorials/GettingStarted/TestUserButton](http://apps.tutorials/GettingStarted/TestUserButton).

In order to be able to use the user button, we should first of all include the appropriate header file:

```
#include <UserButton.h>
```

First, we wire necessary components (**TestUserButtonC.nc**):

```

components UserButtonC;
App.Get -> UserButtonC;
App.Notify -> UserButtonC;

```

The **UserButtonC** component provides **Get** interface which allows to get the state of the user button, and **Notify** event which is fired when the state of the user button is changed. Next, in our main program file (**TestUserButtonP.nc**), we enable the **Notify** event and start the periodic timer:

```

event void Boot.booted() {
    call Notify.enable();
    call MilliTTimer.startPeriodic(1000);
}

```

Once the timer is fired, we check if the button is pressed by using the **Get** interface, and if the button is pressed we turn on the red LED, if not we turn it off:

```
event void MilliTTimer.fired() {  
    if ( call Get.get() == BUTTON_PRESSED){  
        call Leds.led0On();  
    } else {  
        call Leds.led0Off();  
    }  
}
```

Once the state of the user button is changed, the `Notify` event is fired where we check the state of the button as turn on the green LED if the button is pressed, and turn it off if the button is released:

```
event void Notify.notify( button_state_t state ) {  
    if ( state == BUTTON_PRESSED ) {  
        call Leds.led1On();  
    } else {  
        call Leds.led1Off();  
    }  
}
```

Now, if you program a mote with this program, you should see the green LED turn on as soon as you press the user button and turn off as soon as you release it. On the other hand, the red LED will turn on or turn off only when the timer is fired.