Instituto Superior Técnico

Energia-based moteIST++ manual

Authors:
Pedro Gameiro, ist172617

November 4, 2016

Contents

List of Figures

Listings

1 Getting started

This manual will explain in a user perspective the steps required to install, configure, extend and use the MoteIST port for Energia. To fully use the MoteIST port three pieces of software are required. The Texas Instruments (TI) Energia, the MoteIST boards for Energia and the Mote Resident Monitor (MRM) client application. These will be explained throughout this document. Further information can be found in the project website[1] and in the MoteIST website[2].

1.1 Installation

First install the standard TI Energia package from the official website[3]. Instruction on how to do this are available in the website.
Before starting the installation, first identify the TI Energia sketchbook directory location. This is the directory where TI Energia saves its projects and can be configured (and identified) in the File->Preferences Energia menu.
After identifying the sketchbook directory location, download the project files to the directory. This can be done either by downloading a zip file from the project website[1], or by cloning the project git repository (the same URL is used for both the repository, and the website). The git cloning method is exemplified in the penultimate line of listing 2.
The resulting directory should look like the following:

Listing 1: Sketchbook structure

```
sketchbook
+-- examples
+-- hardware
+-- install.sh
+-- libraries
+-- mrm_memory.x
\-- README.md
```

If you restart Energia, in the Tools->Board menu, you should now be able to select a MoteIST board. This informs Energia that you will be developing for this board.
If you plan on using MRM, you will need to replace the original Energia memory mapping with one suitable for MRM. To do so, replace the *hardware/tools/msp430/msp430/lib/ldscripts/msp430f5438a/memory.x* file located inside Energia install directory, with the *mrm_memory.x* file displayed above.
If you have access to a bash terminal the installation process can be automatically performed by running the script from listing 2 after installing the original Energia package. Please do not forget to set the two variables from the script header. The *ENERGIA_INSTALL_DIR* variable should point to the TI Energia installation directory and the *SKETCHBOOK_DIR* to the sketchbook directory. For convenience, the script is included in the project files with the name *install.sh* like one can observe in listing 1.

[1] `https://github.com/pedrogameiro/energia_moteist`
[2] `http://leme.tagus.ist.utl.pt/gems/PmWiki/index.php/Projects/MoteIST`
[3] `http://energia.nu/`

Listing 2: Install script

```
# ************************ PLEASE EDIT ****************************#
# The directory where TI Energia is installed. Please change
# the value of this variable to your specific installation directory.
ENERGIA_INSTALL_DIR=
# Your sketchbook directory. Defaults to $HOME/sketchbook, and can be
# configured in the File->Preferences TI Energia Menu.
SKETCHBOOK_DIR=
#*****************************************************************#

# The MRM memory mapping file. The MRM will *not* work unless
# this file is copied into the TI Energia install directory.
MEMORY_X_DIR="hardware/tools/msp430/msp430/lib/ldscripts/msp430f5438a"
MEMORY_X="$MEMORY_X_DIR/memory.x"

cd "$SKETCHBOOK_DIR"
git clone https://github.com/pedrogameiro/energia_moteist "$PWD"
cp -v mrm_memory.x "$ENERGIA_INSTALL_DIR"/"$MEMORY_X"
```

1.2 Using Energia

After following the installation steps of section 1.1, the TI Energia editor is ready to be used. First go to the Tools->Board menu and select the MoteIST board to which you will be developing. Three options should be available, moteist_v0413, moteist_v1011, moteist_v1011_pepeonboard. This step is important for the correct detection of the board, so please to not forget to choose the right one. With the board selected, one can start developing programs for the MoteIST board. Several examples are available in the TI Energia official documentation[4] and in section 2 with detailed explanations.
In the next section (section 1.3) it is explained how to transfer the developed program to the actual MoteIST board.

1.3 MRM

The MRM client is the piece of software responsible for transferring the resulting, compiled, project from the Energia editor to the MoteIST board, where it will actually be executed. Throughout the rest of this document the MRM host will be the name used to reference the software installed in the MoteIST board responsible for receiving and executing the compiled projects (this component should be more or less invisible for the users activities) and MRM guest will be the name used for the Energia compiled projects that will be transferred to the host.
To use the MRM, it is necessary to change the Energia default memory mapping. The instruction on how to do this can be found in section 1.1. It is also necessary to program the MoteIST board with a MRM host[5], however this guide assumes that this has already been done.
The MRM client (named MRM.jar) program can be downloaded from the project website[5]. This

[4] http://energia.nu/guide/tutorial_bareminimum/
[5] http://leme.tagus.ist.utl.pt/gems/PmWiki/index.php/Projects/MoteIST

guide was written for the MRM version 1, since version 3 does not currently work with this project due to a software incompatibility.
To retrieve the MRM guest program (the resulting hex file) of a Energia project, after compiling the project, click on the *Sketch->Copy Hex File as Path* Energia menu, and execute the MRM client program with the retrieved path.
The MRM client syntax is displayed in listing 3 with the */dev/ttyUSB0* corresponding to the MoteIST connect serial port and the *firmware.hex* value corresponding to the MRM guest, retrieved from the Energia interface.

Listing 3: Boards structure

```
java -jar MRM.jar connect /dev/ttyUSB0
java -jar MRM.jar program firmware.hex /dev/ttyUSB0
java -jar MRM.jar run /dev/ttyUSB0
```

More information about the MRM, and how to use it, can be found in its project webpage [5].

2 Examples and core functions

This section will present several examples designed to instruct the user on how to use the main functions and libs that are available for use in the MoteIST boards. These are presented with detailed explanations.

2.1 GPIO Access

The MoteIST GPIO values can be changed through the same interface as the standard Energia package, as ports. An example is available in the *File->Sketchbook->MoteIST->blink* menu and displayed in listing 4. In this example three ports are accessed. Each one is connect to a led whose value is read and inverted in each iteration of the program, thus blinking the three leds. An explanation of the program at the instruction level is available in its source code in commentary form.

Listing 4: GPIO example

```
/*
  Blink
  The basic Energia example.

  Turns on the three leds of MoteIST for one second,
  then off for one second, repeatedly.

*/

int leds[3] = { LED1, LED2, LED3 };
int ledsnum = sizeof(leds) / sizeof(leds[0]);
int i;

// the setup routine runs once when you press reset:
```

```
void setup() {

    // initialize the digital pin as an output.
    for (i=0; i<ledsnum; i++)
        pinMode(leds[i], OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {

    // loop through the three leds.
    for (i=0; i<ledsnum; i++){

        // toogle the led state. aka Blinky blink! =)
        if (digitalRead(leds[i]) == HIGH) // read the led state
            digitalWrite(leds[i],LOW);     // write low state
        else if (digitalRead(leds[i]) == LOW)
            digitalWrite(leds[i],HIGH);
    }

    // wait for 1 second (1000 milliseconds), forcing
    // a blink frequency of approximately 2Hz.
    delay(1000);
}
```

All the available ports are listed and can be modified in the corresponding board *pins_energia.h* file, in a key value fashion as illustrated in the listing 8. An explanation of this file and how it can be used is available in section3.2.
More information can be found in the Energia official documentation [6].

2.2 libcc2420

To access the functionalities of the cc2420 chip you will need to include the libcc2420 library. An example is available in the *File->Sketchbook->MoteIST->libcc2420_example* menu option and is displayed in listing 5. In this example, the board broadcasts a dummy packet in channel 26 with the PAN address 0x0022 and blinks led 2. After this it checks for a received packet, and if one exists, blinks led 3. It should be noted that if two boards are programmed with this example, they will communicate with each other, and the correct transmission and reception can be observed by the blinking of the leds.

Listing 5: libcc2420 example

```
/*

    libcc2420_example
    A simple send-receive example loop
```

[6] http://energia.nu/Guide_MSP430LaunchPad.html

```
    In channel 26, with pan address 0x0022, broadcasts
    the value 0x3ff0 and checks for a received packet.

    If two boards are programmed with this example, they
    will transmit and receive from one another.

*/

#include <libcc2420.h>


void transmit_test_packet(int seq){

        int pkt_len = 9;
        char pkt[pkt_len];

        pkt[0] = seq;
        pkt[1] = 0x22;          //PAN ID low
        pkt[2] = 0x00;          //PAN ID high
        pkt[3] = 0xFF;          //dest addr low
        pkt[4] = 0xFF;          //dest addr high

        pkt[5] = 0x01;          //src addr low
        pkt[6] = 0x00;          //src addr high

        // -- payload -- //
        pkt[7] = 0x3f;
        pkt[8] = 0xf0;

        cc2420_send(pkt, pkt_len); // send the actual packet

}

// Toggles leds value
void toogled(int led){

    int ledstate;

    // read led value
    ledstate = digitalRead(led);

    // toogle value
    if (ledstate == LOW)
        digitalWrite(led,HIGH);
    else
        digitalWrite(led,LOW);
```

```
}

// the setup routine runs once when you press reset:
void setup() {

    // Init the chip.
    // Channel = 26; Panaddr = 0x0022
    cc2420_init(26,0x0022);

    // init the led2 as output for debuging.
    pinMode (LED2, OUTPUT);

    // init the led3 as output for debuging.
    pinMode (LED3, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {

        unsigned char seq_num = 0;
        char rxbuf[128]; // receive buffer
        int bytes_count;

        /*
        +-------+-----+
        | level | dbm |
        | 0     |  0  |
        | 1     | -1  |
        | 2     | -3  |
        | 3     | -5  |
        | 4     | -7  |
        | 5     | -10 |
        | 6     | -25 |
        +-------+-----+ */
        cc2420_set_txpower(0); // max tx power.

        //transmission-reception cycle
        //when transmitting, LED2 toggles its state
        //when a packet is received, LED3 toggles its state
        while (1) {

                // wait for a second (1000 milliseconds)
                delay(1000);

                // transmit the test packet
                transmit_test_packet(seq_num++);
                toogled(LED2);
```

```
                // check for a received packet in cc2420
                // the internal buffer. This function
                // does *not* block.
                bytes_count = cc2420_recv(&rxbuf,128);

                // if received bytes > 0, then a message
                // was received. Blink led3.
                if (bytes_count > 0)
                        toogled(LED3);

        }

}
```

The libcc2420 header files can be found in the libraries directory of the project and expose to the user the following operations:

- void cc2420_init(int channel,int panid)
  It performs all the initializations required to start using the cc2420 including starting and calibrating the transmission oscillator, selecting a PAN address and a communication channel.

- void cc2420_set_channel(int channel)
  It changes the communication channel, if one desires, after the initialization. The cc2420 can operate in channels from 11 through 26.

- int cc2420_get_channel(void)
  Returns the value of the currently operating channel.

- void cc2420_set_pan(int panid)
  Changes the operating PAN address, if one desires, after the initialization.

- int cc2420_recv(void *buf, unsigned short bufsize)
  Checks if the CC2420 has a packet in its internal buffer and, if so, retrieves it and fills *buf* with its value. *buffsize* is the size of the *buf* buffer. The returned *int* value contains the number of bytes retrieved. This function is non-blocking.

- void cc2420_send(const char *payload, unsigned short pkt_len)
  Delivers the *payload* array of bytes to the CC2420 for transmission. *pkt_len* is the size in bytes of the message to transmit.

- void cc2420_set_txpower(uint8_t power)
  Changes the amount of power that will be used in the Radio Frequency (RF) transmissions. The possible values for *power* are the ones displayed in the *PA_LEVEL* column of the figure 1.

- int cc2420_get_txpower(void)
  Retrieves the current value of the amount of power being used in transmissions.

PA_LEVEL	TXCTRL register	Output Power [dBm]	Current Consumption [mA]
31	0xA0FF	0	17.4
27	0xA0FB	-1	16.5
23	0xA0F7	-3	15.2
19	0xA0F3	-5	13.9
15	0xA0EF	-7	12.5
11	0xA0EB	-10	11.2
7	0xA0E7	-15	9.9
3	0xA0E3	-25	8.5

Table 9. Output power settings and typical current consumption @ 2.45 GHz

Figure 1: CC2420 power modes

2.3 SPI

The MoteIST SPI functionalities are available through the same interface as the standard Energia package. An example is available in the *File->sketchbook->moteist_pepeonboard->Spi7Display* menu, and is displayed in listing 6. It serves the simple purpose of demonstrating the use of the SPI interface while using the MoteIST PepeOnBoard seven segment display as a SPI slave. The example program, in each iteration, prints a debug message over the serial interface, selects the seven segment display SS (slave select) pin, thus reserving the channel for a transmission, and increments the values of the display.
A lot of detailed SPI examples can be found in the official Energia documentation [7].

Listing 6: SPI example

```
/*

    Spi7Display
    A simple SPI 7 segment display manipulation example.

    The two digit display value will loop from 00 to FF
    in hexadecimal values, in intervals of 1 second.

*/

#include <SPI.h>

// P2.4 7Segment Display
const int SS = CBC1_8;

// 7 seg. display translation table. The N position
// of this table contains the hex value that will
// draw the said value on the display.
char decode[] = {0x3F, 0x6, 0x5B, 0x4F, 0x66,
                 0x6D, 0x7D, 0x7, 0x7F, 0x67,
```

[7] http://energia.nu/reference/spi/

```
                0x77, 0x7C, 0x39, 0x5E, 0x79,
                0x71};

// Current number on display.
int num=0;

// Toggles leds value
void toogled(int led){

    int ledstate;

    // read led value
    ledstate = digitalRead(led);

    // toogle value
    if (ledstate == LOW)
        digitalWrite(led,HIGH);
    else
        digitalWrite(led,LOW);
}


// the setup routine runs once when you press reset:
void setup()
{
    // init the SPI slave select pin as output.
    pinMode (SS, OUTPUT);
    // init the led2 as output for debuging.
    pinMode (LED2, OUTPUT);

    // init the Serial bus with 9600 baud rate.
    // also for debuging.
    Serial.begin(9600);

    // init the spi bus.
    SPI.begin();

}


// the loop routine runs over and over again forever:
void loop()
{

    // print debug info
    Serial.print("Changing display to "+num);
```

```
    // Send the two digits according to the display
    // SPI protocol.
    digitalWrite(SS,LOW);           // Select SPI slave
    SPI.transfer(decode[num]);      // Second digit of display
    SPI.transfer(decode[num++]);    // First digit of display
    digitalWrite(SS,HIGH);          // Release SPI slave

    // toogle led state
    toogled(LED2);

    // wait for 1 second (1000 milliseconds)
    delay(1000);

    // if overflow, we reached 0xff. Reseting to 0x00
    if(num==15)
        num=0;
}
```

2.4 Serial

Once more, the MoteIST serial functionalities are available through the original serial interface. An example is available in the *File->sketchbook->moteist->serial* menu and displayed in listing 7. It exemplifies the use of the interface by writing a message to both available serial ports (through the MoteIST UART0 and UART3). In the case of the older versions of the MoteIST (version 1011 with the exception of the MoteIST PepeOnBoard) the usb driver has been discontinued, and so, it may be very hard to use the serial port.

Listing 7: Serial example

```
/*

    Serial
    A simple Serial port write example

    The example runs an infinite loop, printing
    a message to both the debug and auxilary serial
    ports. LED1 toogles on every loop.

*/

// the setup routine runs once when you press reset:
void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  Serial1.begin(9600);
```

```
  // set LED1 as an output pin.
  pinMode(LED1, OUTPUT);

}

// the loop routine runs over and over again forever:
void loop() {

  // Write message to serial port
  Serial.println("Hello I'm talking to Serial");
  Serial1.println("Hello I'm talking to Serial1");


  // wait 1 second (1000 milliseconds)
  delay(1000);
  // turn off led 1.
  digitalWrite(LED1,LOW);
  // wait another second
  delay(1000);
  // turn on led 1
  digitalWrite(LED1,HIGH);

}
```

The pin mapping of the serial ports is defined in the boards corresponding *pins_energia.h* file in a key value fashion, as illustrated in the listing 8. To change the used UARTS one only needs to change this mapping.

Listing 8: MoteIST 0413 serial mapping

```
                                                // Port / MoteIST Label
static const uint8_t DEBUG_UARTRXD = 11; // P3.5 / CB1_UCA0(RXD/SOMI)
static const uint8_t DEBUG_UARTTXD = 13; // P3.4 / CB1_UCA0(TXD/SIMO)
static const uint8_t AUX_UARTRXD = 31;   // P10.5 / CB2_UCA3(RXD/SOMI)
static const uint8_t AUX_UARTTXD = 33;   // P10.4 / CB2_UCA3(TXD/SIMO)
```

More information about the Energia serial lib can be found in the Energia official documentation[8].

3 Extending Energia

The TI Energia software offers a lot of potential for extending its features. In particular, support for detecting new boards and for creating new libraries can be very useful. This section will explain these operations.

[8] `http://energia.nu/reference/serial/`

3.1 How to add new libraries

The most straight forward way of adding new functionalities to Energia is in the form of libraries. A full explanation on how to add libraries to Energia can be found in the Energia website [9].

3.2 Energia configuration files

Each board in the Energia framework correspond to an entry in the *boards.txt*, and to a *pins_energia.h* file, as displayed bellow:

Listing 9: Boards structure

```
sketchbook/hardware/
+-- msp430
| +-- boards.txt
| +-- cores
| \-- variants
|       \-- moteist_v0413
|       +-- moteist_v1011
|       | +-- Board.mk
|       | \-- pins_energia.h
|       \-- moteist_v1011_pepeonboard
```

The *pins_energia.h* is the most important file for configuring a board, as it contains almost all of the information.
Three types of values as stored in the file:

- A couple of C type vectors that translate the msp430 digital pin notation (i.e. a pin is represented by $R.B$ where R is a register and B a bit of the register) to the Energia notation (a port typed access).
  To perform the translation, one vector is filled with the registers, the other with the bit numbers. When Energia wants to use a pin with its port type interface, it access the N position of each table and changes the respective bit of the respective register. An example of this is displayed in listing 10.

- C language constants will be available in the Energia framework. Since the *pins_energia.h* file is included by Energia framework, one can declare constants for the convenience of the user.

- Configurations for the available libs. This is defined, again, by the use of C constants, and their values are lib dependent. By retrieving board dependent information from the *pins_energia.h* the libs are made board independent.

It is worth noting that MoteIST **pins_energia.h** contains in its source code vim style folds, to separate and title the different sections of the file, and so, the use of a vim fold compatible editor will simplify the task of reading the file.

[9] `http://energia.nu/Tutorial_Library.html`

Listing 10: pin_energia.h

```
// Raise the X bit. i.e. BV(1) == 0010
#define BV(x) (1 << (x))

/*
* Map the Energia ports with the
* msp430 pins (identified by register.pin).
*/

// Registers from the msp430 pins
const uint8_t digital_pin_to_port[] = {
    NOT_A_PIN,    /* dummy */
    NOT_A_PIN,    /* 1 RST_NMI_SBWTDOI */ // CBC_1
    P2,           /* 2 P2 register from P2.1 */
    P4,           /* 3 P4 register from P4.1 */
    P2,           /* 4 P2 register from P2.2 */
    P4,           /* 5 P4 register from P4.0 */
    ...
};

// Bits from the msp430 pins
const uint8_t digital_pin_to_bit_mask[] = {
    NOT_A_PIN, /* 0,  pin count starts at 1 */
    NOT_A_PIN, /* 1 RST_NMI_SBWTDOI */ // CBC_1
    BV(1),     /* 2 0010 (bit 1) from P2.1 */
    BV(1),     /* 3 0010 (bit 1) from P4.1 */
    BV(2),     /* 4 0100 (bit 2) from P2.2 */
    BV(0),     /* 5 0001 (bit 0) from P4.0 */
    ...
};
```

3.3 How to add new boards

To add a new board, with a new pin-mapping configuration, just add a new entry to the *boards.txt* file and add a new *pins_energia.h* file to the variants directory, has explained in the section 3.2 and displayed in Listing 9.

3.4 CodeComposer integration

CodeComposer Integrated Development Environment (IDE) offers a great more deal of additional functionalities than the standard Energia editor. Because of this, it may be desirable to integrate the simplicity of Energia with a full-fledged IDE. The full instructions on how to do this can be found in the TI Energia official documentation [10]

[10] `http://energia.nu/guide/import-energia-sketch-to-ccsv6/`