Debug software in Tinyos-v2

The Nesc code that one application must include to perform debug messages through serial port is based in a previous version from DIKU that was developed under Tinyos-v1. The adaptation of the code from Tinyos-v1 to Tinyos-v2 implies several changes in the code and the inclusion of new files in the distribution file tree. The files that come from the DIKU distribution are the following:

Interfaces are included in the folder **interfaces**:

- File **ConsoleInput.nc** defines interface ***ConsoleInput***
- File **ConsoleOutput.nc** defines interface ***ConsoleOutput***
- File **Debug.nc** defines interface ***Debug***

Files under the folder **lib** are:

- File **ConsoleDebugM.nc** provides the interface Debug and uses the interface ConsoleOutput. This module constitutes the first level in the debug code from DIKU. It accepts the debug commands coming from the application and calls the sequence of ConsoleOutput commands to send the debug information using a sequence of strings.
- File **ConsoleM.nc** provides interface ConsoleInput, ConsoleOutput and StdControl. The purpose of the first one is to signal the reception of debug commands through the serial port. The second allows the sending of debug messages through the serial port. The last interface is provided for the initialization of the serial port. On the other hand, this module uses the interface HPLUART that interacts directly with the hardware of the serial port. This module is in charge of changing the format of the incoming debug strings and the sending of the characters to the serial port.
- File **Debug.h** includes the declaration of the debug commands that the application can use.

Due to the fact that in Tinyos-v2 the files for the control of the serial port have been modified and the interface HPLUART is no longer supported, it is necessary to include the some files in the Tinyos-v2 distribution. These files add code that depends on the hardware of the mote, so different versions should be included for every type of platforms or motes. In this document we have centred our attention in the Tmote.

Interfaces that must be included:

- File **HPLUART.nc** includes the description of the ***HPLUART*** interface.

Besides the files for the declaration of the interfaces the following modules for the Uart control and programming must be included in the tree as well.

- File **HPLDBUART.nc** is the component that connects the modules and interfaces for the debugging with the modules of Tinyos-v2. It is a generic component that does not depend on the platform that it is finally used.

- File **HPLDBUARTM.nc** is the module that performs the translation from the interface HPLUART to the interface UartStream that is provided by Tinyos-v2 to send and receive data through the serial port.
- File **DebugC.nc** is a new component defined to integrate all the modules and interfaces that are necessary for debugging purposes. It is a wrapper that hides the rest of modules and interfaces to the application programmer. This component provides two interfaces Debug (that is the high level interface to send debug messages) and StdControl that is used to initialize the complete debug software stack. The main application will interact with the debug software only through this component. This is the only component that the programmer has to add to the application wiring. It does not depend on the platform for which we are compiling.

All these files must be inserted in the Tinyos tree in the following locations:

Folder: **tinyos-2.x\tos\interfaces**

- Files to be included:
  - **ConsoleInput.nc,**
  - **ConsoleOutput.nc,**
  - **Debug.nc,**
  - **HPLUART.nc,**

Folder: **tinyos-2.x\tos\system**

- Files to be included:
  - **DebugC.nc,**
  - **ConsoleDebugM.nc,**
  - **ConsoleM.nc**
  - **Debug.h**
  - **HPLDBUARTM.nc**
  - **HPLDBUART.nc**

Tmote Issues

The debug messages can make use of either serial port 1or serial port 0. The port can be changed modifying the code inside the file PlatformSerialC.nc in folder \tinyos-2.x\tos\platforms\telosa. This file defines the serial port 1 as the serial port that is used for Serial_messages by default:

```
components new Msp430Uart1C() as UartC;
```

But it can be changed by the corresponding serial port 0:

```
components new Msp430Uart0C() as UartC;
```

Another important issue is the initial configuration for the serial port that is defined in the file:

```
\tos\platforms\telosa\TelosSerialP.nc
```

The transmission speed established by default is 115200 bps.

Micaz Issues

In this case the serial port used is port 0. This property is declared in the file:

`\tos\platforms\mica\PlatformSerialC.nc`

And its configuration is established in the file:

`\tos\chips\atm128\HplAtm128UartP.nc`

The transmission speed by default is 57600 bps.

Example

Now, we can see an example of an application that includes debug commands to send to the serial port the same packets that it is broadcasting through the wireless link. The application is the Oscilloscope that comes with Tinyos-v2 distribution. Now we can see the main changes that it must be introduced in the original code.

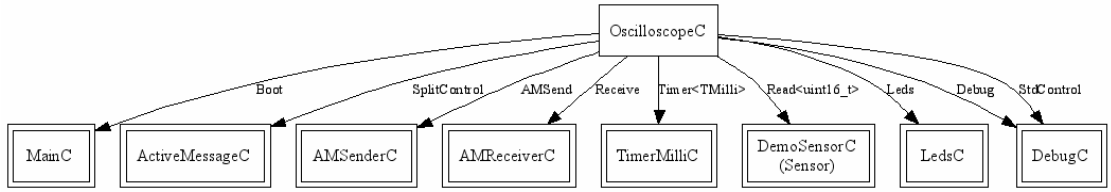
```
configuration OscilloscopeAppC { }
implementation
{
  components OscilloscopeC, MainC, ActiveMessageC, LedsC,
    new TimerMilliC(), new DemoSensorC() as Sensor,
    new AMSenderC(AM_OSCILLOSCOPE), new AMReceiverC(AM_OSCILLOSCOPE),
    DebugC;

  OscilloscopeC.Boot -> MainC;
  OscilloscopeC.RadioControl -> ActiveMessageC;
  OscilloscopeC.AMSend -> AMSenderC;
  OscilloscopeC.Receive -> AMReceiverC;
  OscilloscopeC.Timer -> TimerMilliC;
  OscilloscopeC.Read -> Sensor;
  OscilloscopeC.Leds -> LedsC;
  OscilloscopeC.Debug -> DebugC;
  OscilloscopeC.StdControl -> DebugC;
}
```

The text in bold constitutes the new lines that have been introduced in the original file. The module has also to be modified in following way:

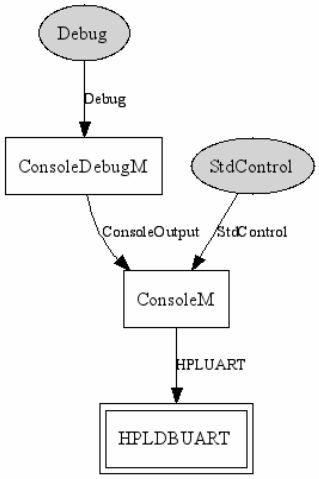
```
module OscilloscopeC
{
  uses {
    interface Boot;
    interface SplitControl as RadioControl;
    interface AMSend;
    interface Receive;
    interface Timer<TMilli>;
    interface Read<uint16_t>;
    interface Leds;
    interface Debug;
    interface StdControl;
  }
}
```

```
implementation
{
....

event void Boot.booted() {
    local.interval = DEFAULT_INTERVAL;
    local.id = TOS_NODE_ID;
    if (call RadioControl.start() != SUCCESS)
      report_problem();
    call StdControl.start();
  }
}
....

event void AMSend.sendDone(message_t* msg, error_t error) {
    if (error == SUCCESS)
      report_sent();
    else
      report_problem();
      DBG_DUMP((uint8_t*) msg, sizeof(sendbuf), 2);
    sendbusy = FALSE;
  }
}
```

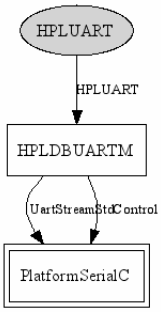
The line call StdControl.start(); inside the Boot.booted() event is included to initialize the debug code. And the line DBG_DUMP((uint8_t*) msg, sizeof(sendbuf), 2); inside AMSend.sendDone event performs the sending of the message msg through the serial port. Finally, the detailed diagrams of the complete wiring inside the Oscilloscope application and the debugC module are:



Oscilloscope wiring



DebugC wiring



HPLDBUART wiring