Wireless Sensor Network for Experiment

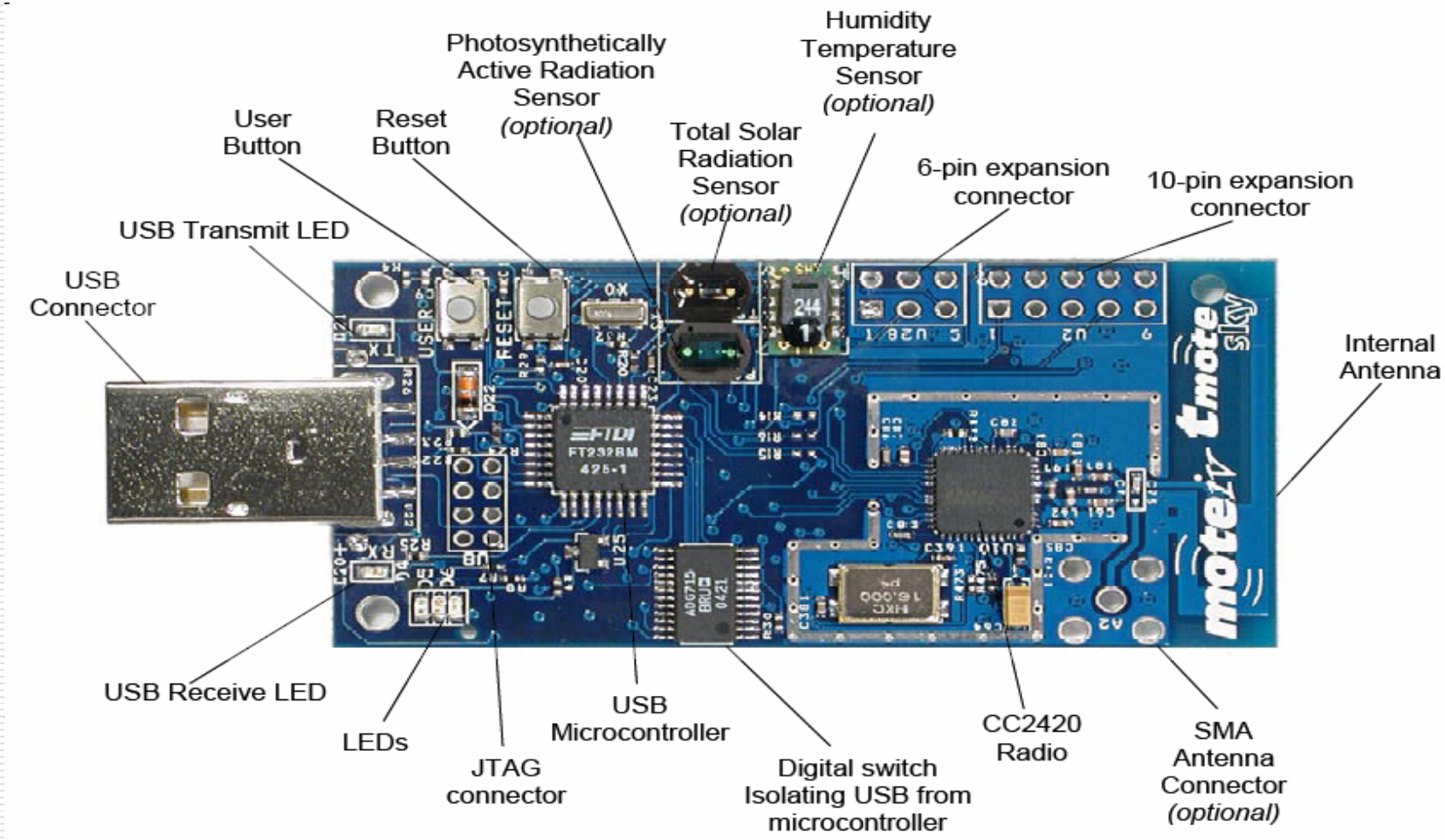
Speaker : Chyi Yih Pan

Outline

- Introduce Mote & Connect
- Introduce TinyOS & nesC
- How to program nesC
- Compiler and Download nesC program
- Generate program's structure documentation
- Conclusion

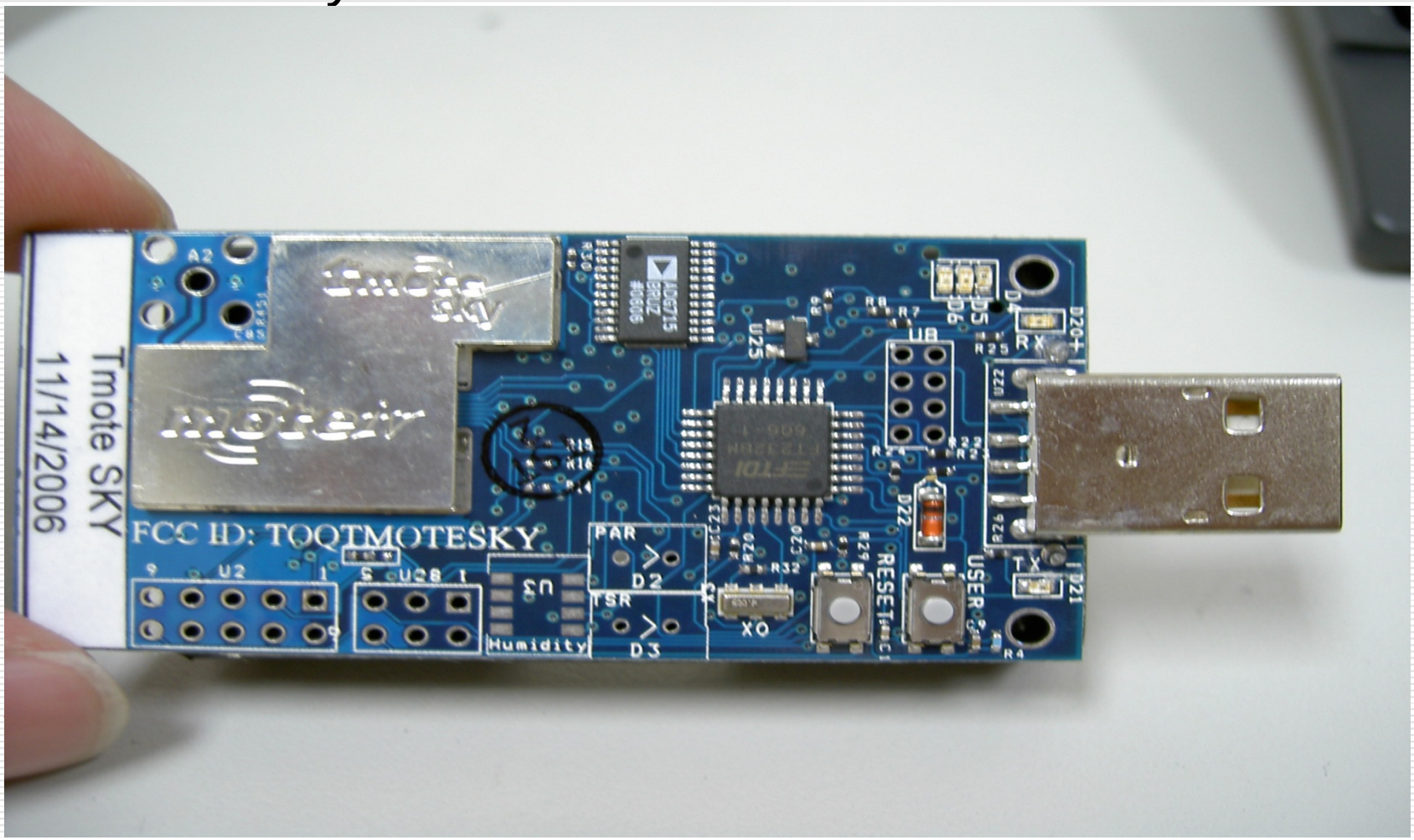
Mote Hardware

- Front of the Tmote Sky module :



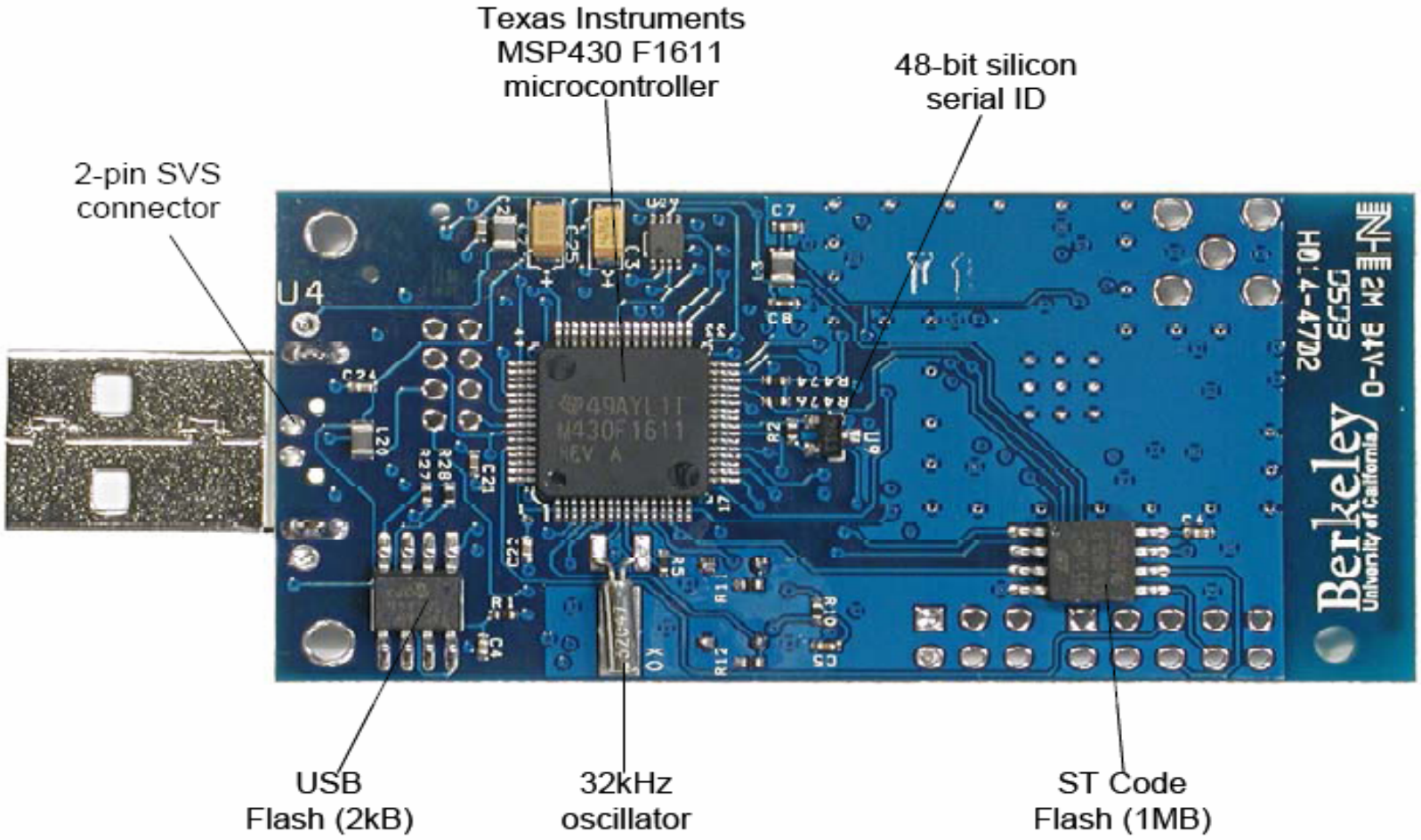
Mote Hardware

- Tmote Sky module：



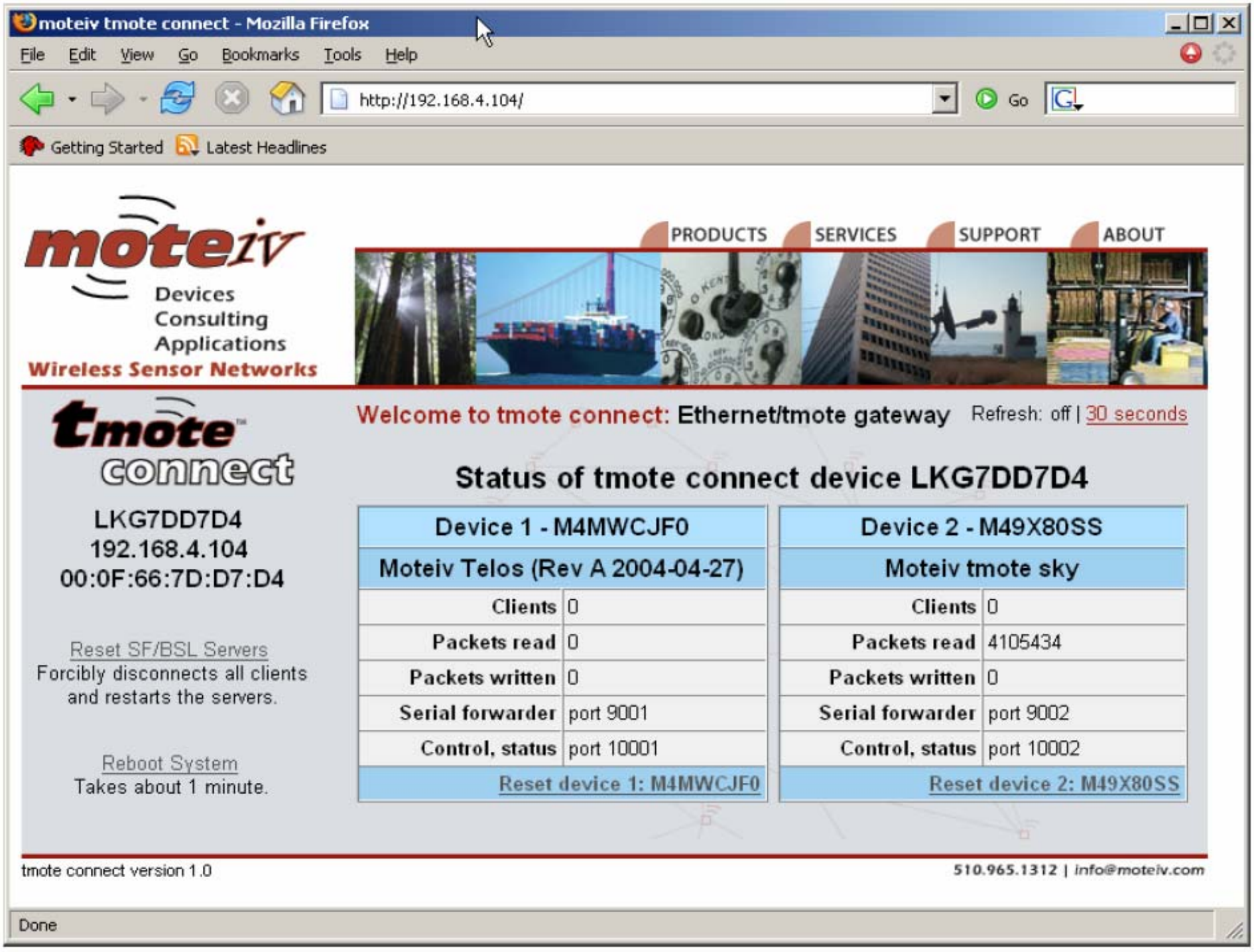
Mote Hardware

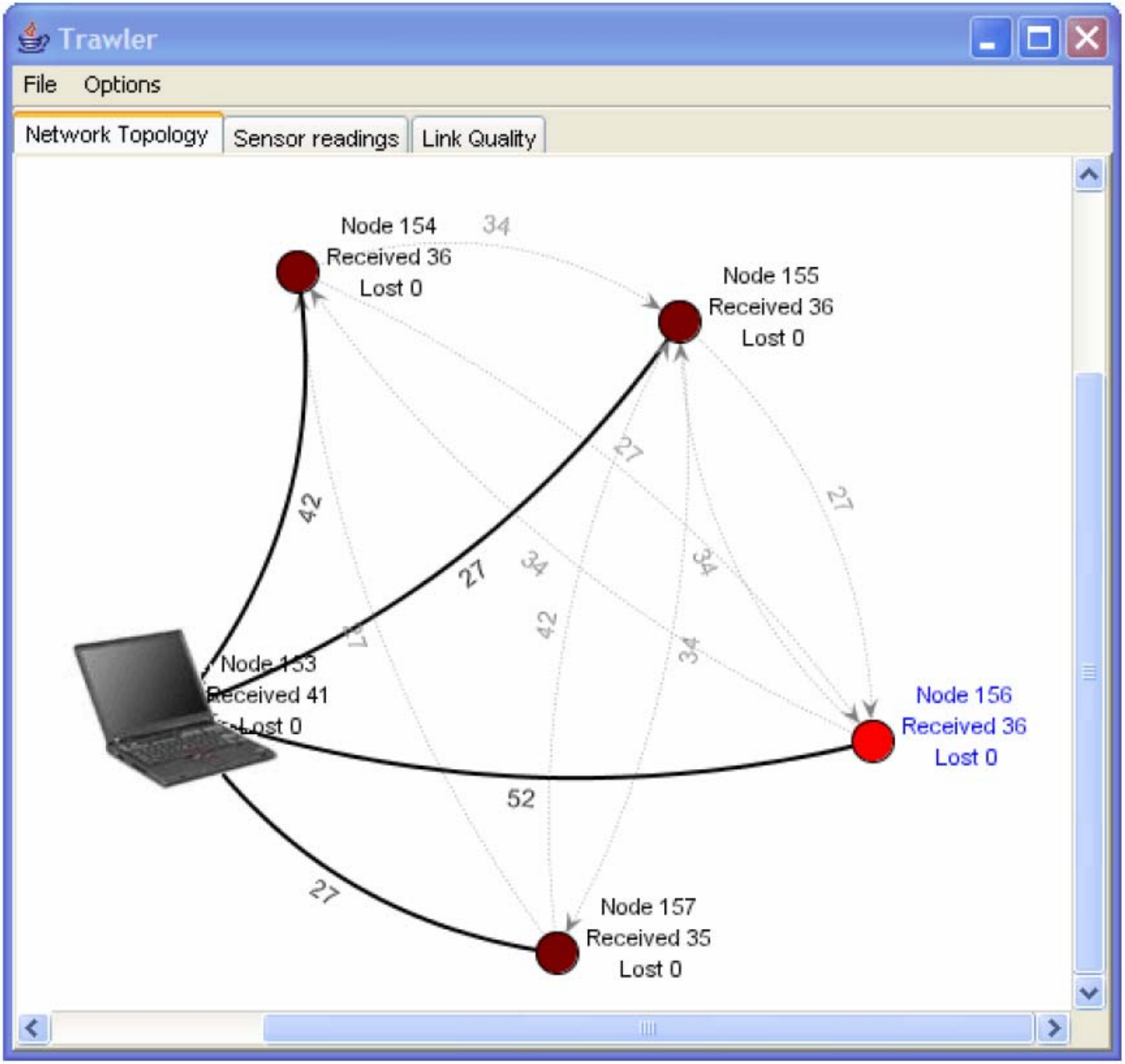
- Back of the Tmote Sky module :

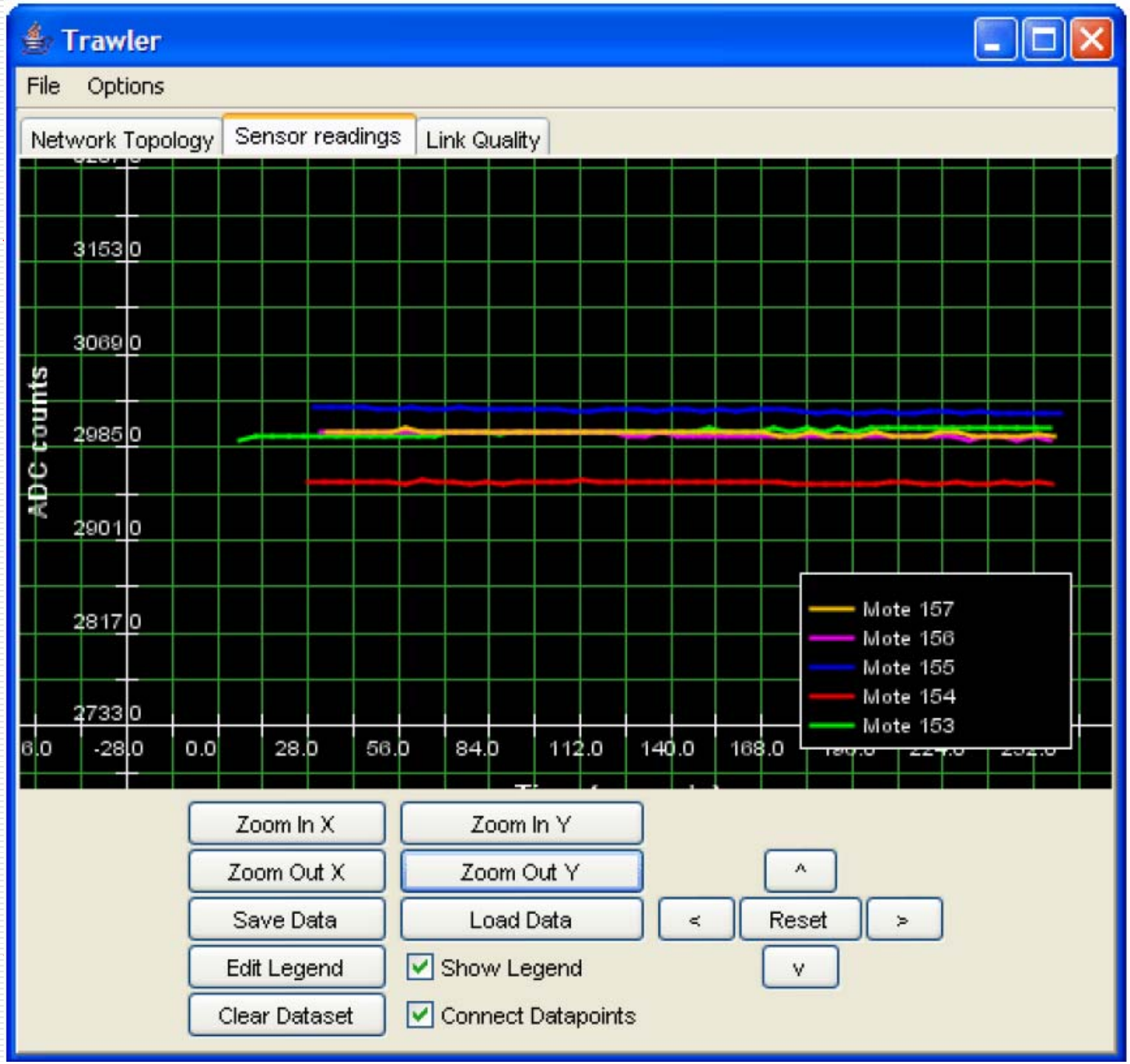


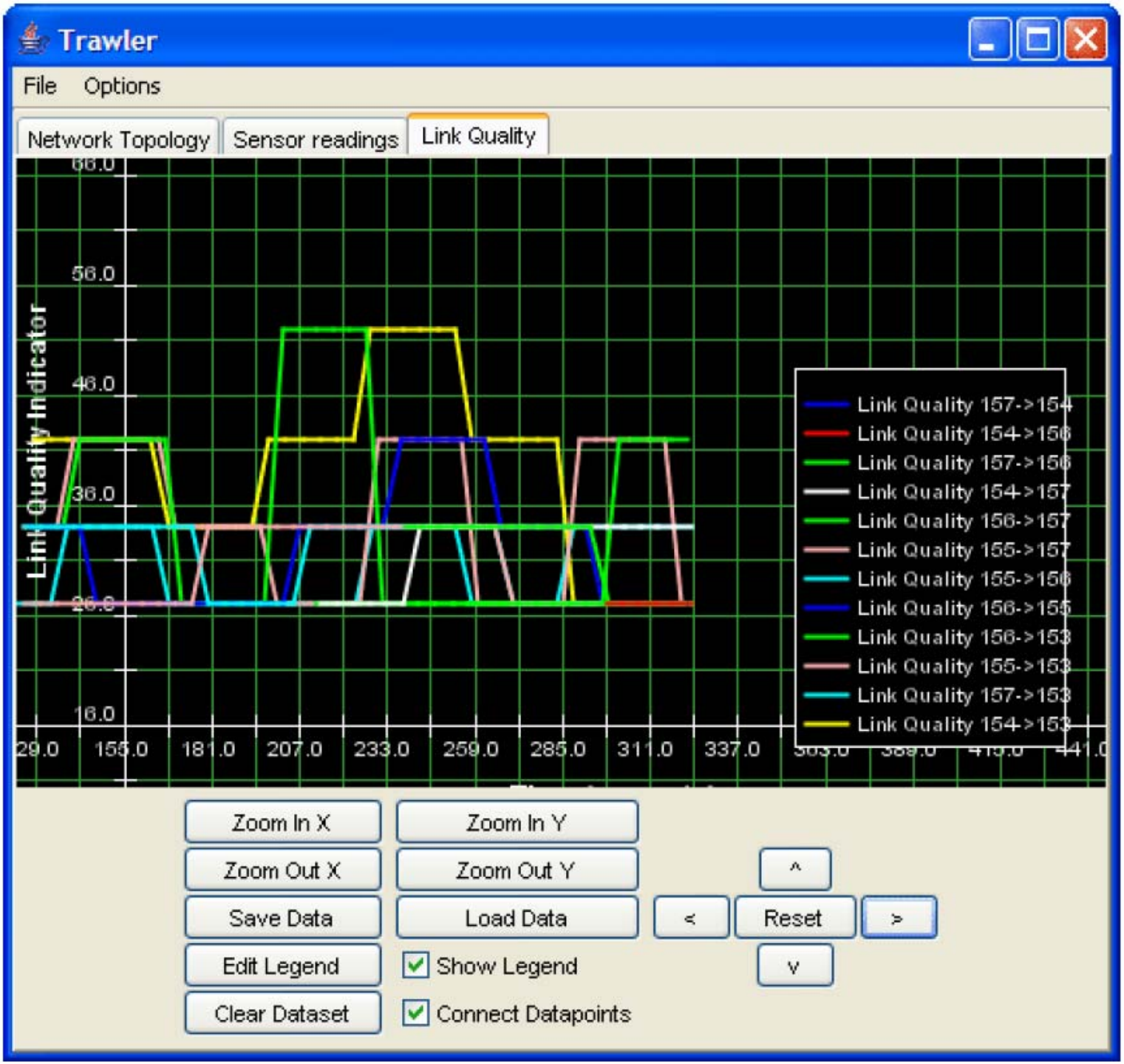
Connect Hardware

- Access Tmote wireless sensor modules through Ethernet with Moteiv's Tmote Connect gateway


moteiv tmote connect - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://192.168.4.104/
Go
Getting Started
Latest Headlines
moteiv
Devices
Consulting
Applications
Wireless Sensor Networks
PRODUCTS
SERVICES
SUPPORT
ABOUT
tmote connect
LKG7DD7D4
192.168.4.104
00:0F:66:7D:D7:D4
Reset SF/BSL Servers
Forcibly disconnects all clients and restarts the servers.
Reboot System
Takes about 1 minute.
Welcome to tmote connect: Ethernet/tmote gateway
Refresh: off | 30 seconds
Status of tmote connect device LKG7DD7D4
Device 1 - M4MWCJF0
Moteiv Telos (Rev A 2004-04-27)
Clients 0
Packets read 0
Packets written 0
Serial forwarder port 9001
Control, status port 10001
Reset device 1: M4MWCJF0
Device 2 - M49X80SS
Moteiv tmote sky
Clients 0
Packets read 4105434
Packets written 0
Serial forwarder port 9002
Control, status port 10002
Reset device 2: M49X80SS
tmote connect version 1.0
510.965.1312 | info@moteiv.com
Done


Trawler
File
Options
Network Topology
Sensor readings
Link Quality
Node 154
Received 36
Lost 0
Node 155
Received 36
Lost 0
Node 153
Received 41
Lost 0
Node 156
Received 36
Lost 0
Node 157
Received 35
Lost 0
34
42
27
34
27
34
42
27
34
52
27


Trawler
File
Options
Network Topology
Sensor readings
Link Quality
3153 0
3069 0
2985 0
2901 0
2817 0
2733 0
ADC counts
6.0
-28.0
0.0
28.0
56.0
84.0
112.0
140.0
168.0
Mote 157
Mote 156
Mote 155
Mote 154
Mote 153
Zoom In X
Zoom In Y
Zoom Out X
Zoom Out Y
Save Data
Load Data
Edit Legend
Show Legend
Clear Dataset
Connect Datapoints
^
<
Reset
>
v


Trawler
File
Options
Network Topology
Sensor readings
Link Quality
Link Quality Indicator
56.0
46.0
36.0
16.0
29.0
155.0
181.0
207.0
233.0
259.0
285.0
311.0
337.0
Link Quality 157->154
Link Quality 154->156
Link Quality 157->156
Link Quality 154->157
Link Quality 156->157
Link Quality 155->157
Link Quality 155->156
Link Quality 156->155
Link Quality 156->153
Link Quality 155->153
Link Quality 157->153
Link Quality 154->153
Zoom In X
Zoom In Y
Zoom Out X
Zoom Out Y
Save Data
Load Data
Edit Legend
Show Legend
Clear Dataset
Connect Datapoints
^
<
Reset
>
v

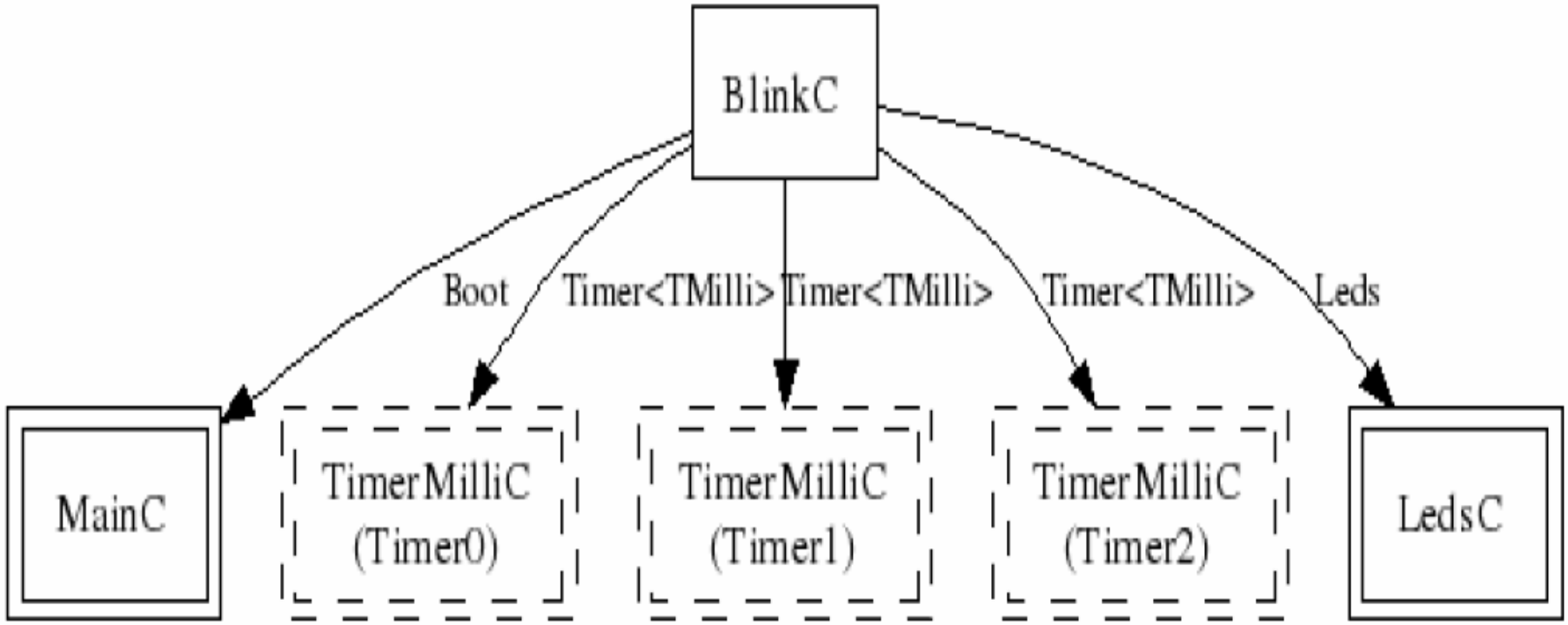
Introduce TinyOS & nesC

- Component-based architecture nesC
- C-like language
- nesC code 撰寫在副檔名爲 *. nc
- *. nc 的內容可能是定義
  - interface (定義存取元件的 function)
  - component (程式碼實作部份)

How to use Interface

- interface 內定義 function
- component藉由宣告介面和其他元件溝通
- 元件可對interface 宣告成兩種型態：
  - provides（外界使用該元件的唯一橋樑）
  - uses（該元件對外溝通的唯一通道）
- function 可定義為兩種類型：
  - Command : provides介面的元件實作function
  - Event: uses介面的元件實作function

How to use Interface(Cont.)



How to use function

- command:

A元件對B元件下命令

故B元件要實作command

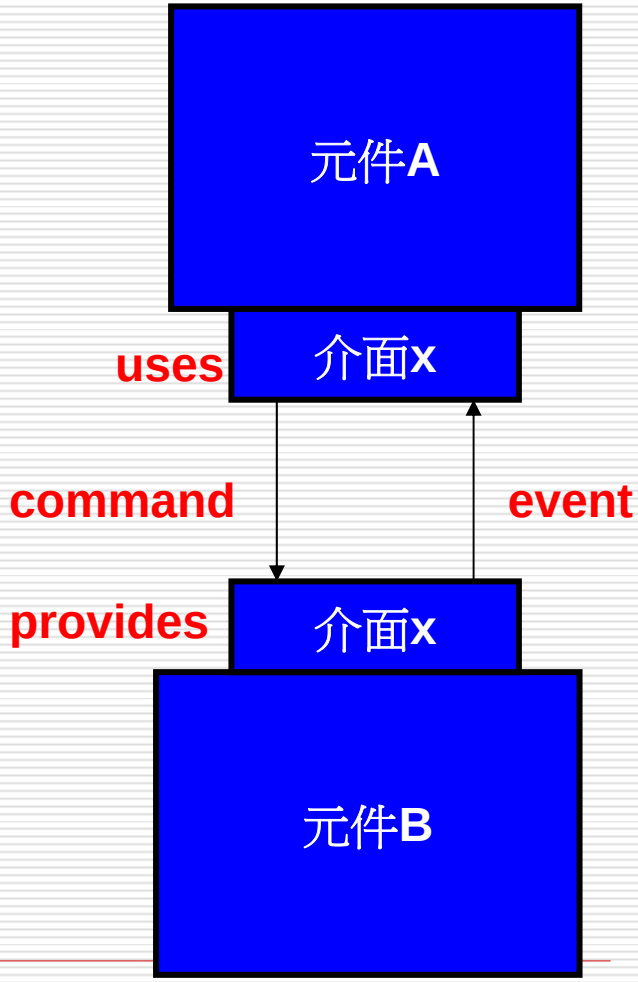
- event:

B元件做完A元件的命令

後，可選擇是否回報事件

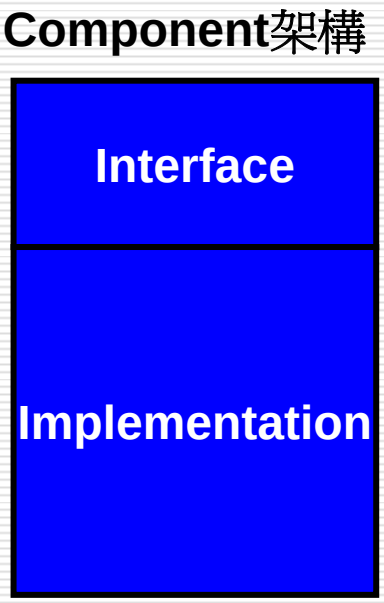
給A，故A元件要實作

event



How to use Component

- Component 分兩種形式：
  - module
    - module為元件的最小單位
    - 實作介面定義function的程式碼
  - configuration
    - 負責元件和元件之間的溝通



nesC code example

```
// Component-name . nc
module Component-name{
  provides{
    interface Interface A;
  }
  uses{
    interface Interface B;
  }
}
implementation{
  //…
}
```

How to program nesC

- 假設現在撰寫一個程式欲

  控制Sensor node上的LED紅燈每一秒閃一次

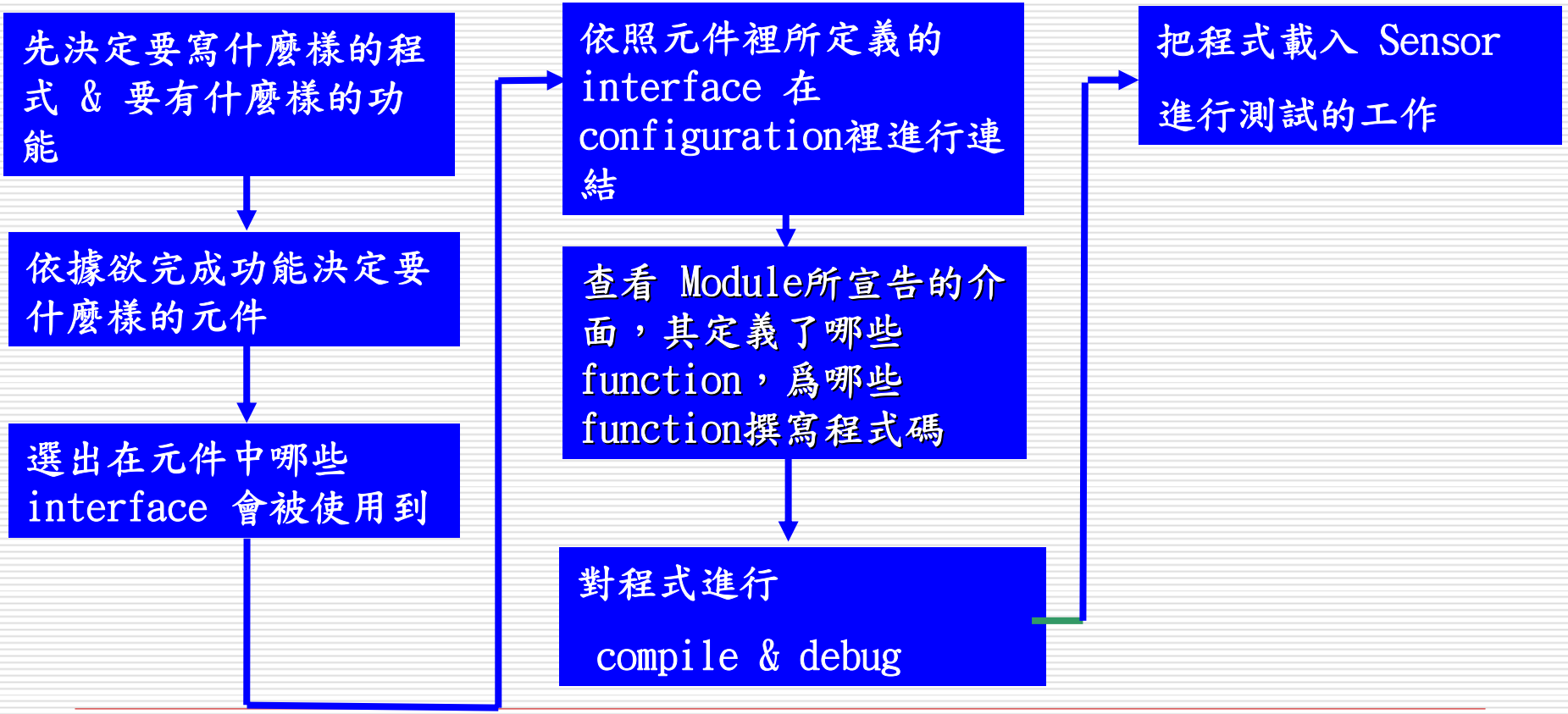
- 我們已知下列元件可用：
  - Main
  - SingleTimer（計時器 計時最小單位為ms）
  - LedsC（控制Led燈的元件）

How to program nesC

- 欲得知有何元件可使用，可查詢以下資料夾，有詳細解說各元件
  - **C:\cygwin\opt\tinyos-1.x\tos\system**
- 欲得知該元件如何存取，要看該元件的介面如何宣告，可查詢以下資料夾
  - **C:\cygwin\opt\tinyos1.x\tos\interfaces**

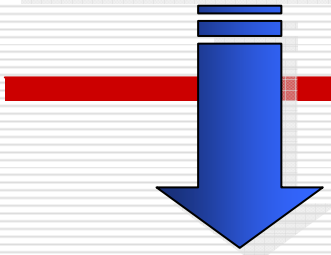
How to program nesC

- 流程圖



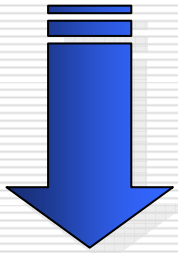
先決定要寫什麼樣的程式 & 要有什麼樣的功能

假設現在撰寫一個程式欲控制 Sensor上的LED紅燈每一秒閃一次



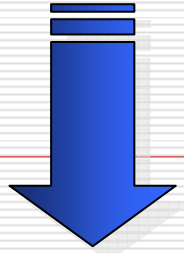
依據欲完成功能決定要什麼樣的元件

Main **BlinkM**
SingleTimer **LedsC**



選出在元件中哪些 interface 會被使用到

implementation 內容要先查看**BlinkM**所宣告的介面，其定義了哪些**function**，才能得知　你將要實際爲哪些**function**撰寫程式碼



依照元件裡所定義的 interface 在configuration裡進行連結

把 該連結的 uses interface 與 provides interface 進行連結

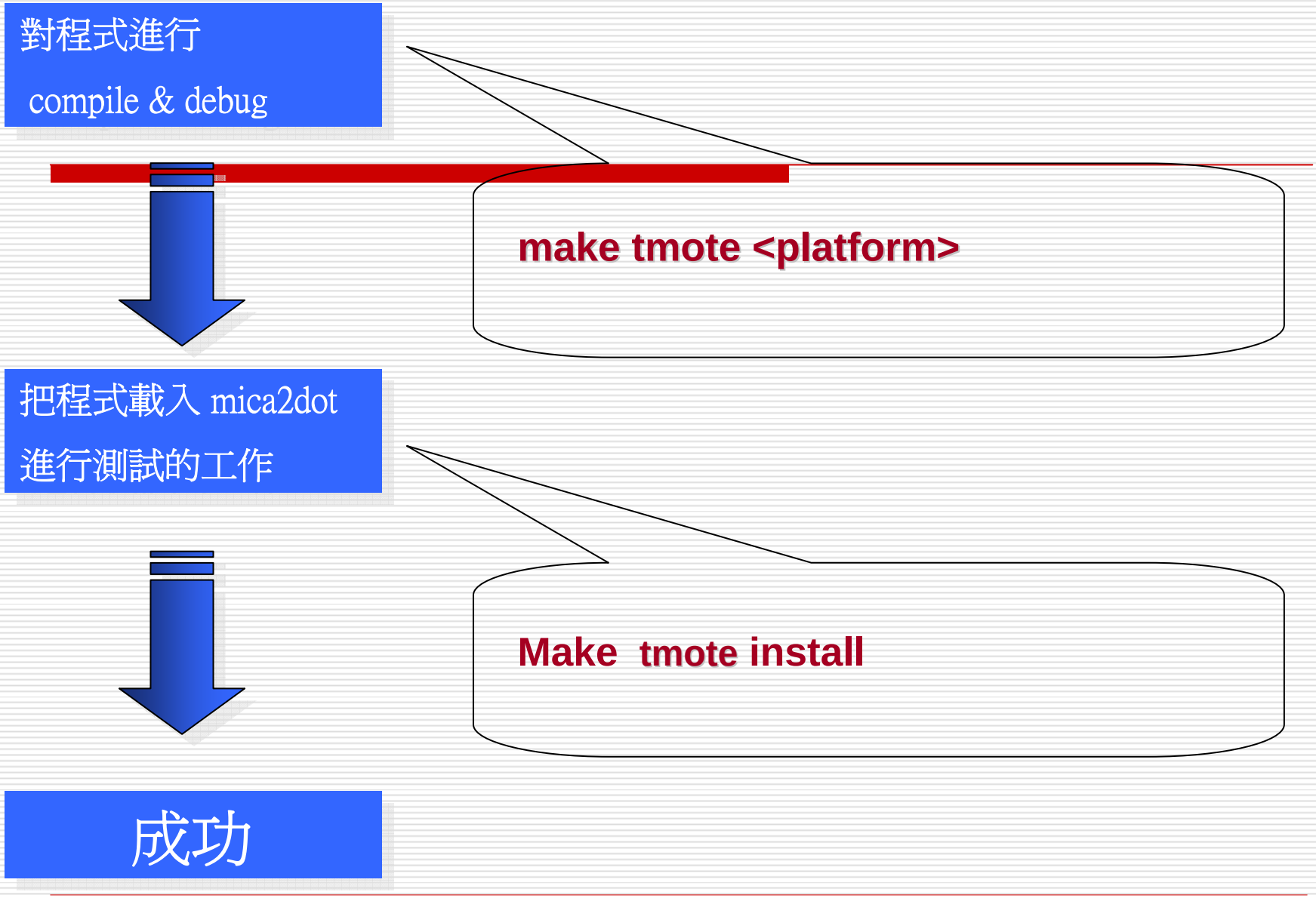
查看 Mudule所宣告的介面，其定義了哪些function，爲哪些function撰寫程式碼

EX :

command result_t StdControl.init();
command result_t StdControl.start();
command result_t StdControl.stop();
event result_t Timer.fired();

使紅燈閃動

async command result_t leds.redToggle();


對程式進行
compile & debug
make tmote <platform>
把程式載入 mica2dot
進行測試的工作
Make tmote install
成功

How to program nesC

- 可想像程式架構雛爲：

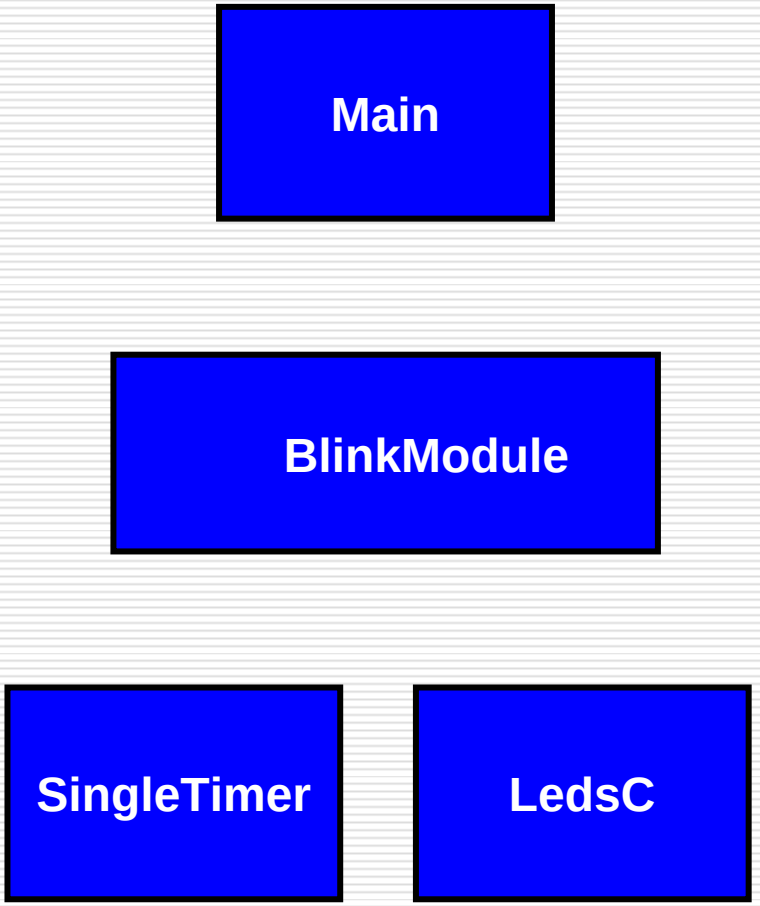
  注意撰寫程式碼 Main

  要在最上層

- 接下來考慮interface

  因爲元件互相連結溝通

  要透過interface



Start programming nesC

- 首先要做的是撰寫元件和元件如何溝通的程式碼，也就是之前提起的
  - configuration（負責元件和元件之間的溝通）
- 接下來就是實際撰寫控制元件的程式碼
  - module（實作介面定義function的程式碼）

- 現在假設configuration 檔名為 Blink.nc
  假設module 檔名為 BlinkM.nc

Introduce – Example (Blink.nc)

```
configuration Blink {
}
implementation {
components Main, BlinkM, SingleTimer, LedsC;
Main.StdControl -> BlinkM.StdControl;
Main.StdControl -> SingleTimer.StdControl;
BlinkM.Timer -> SingleTimer.Timer;
BlinkM.Leds -> LedsC;
}
```

Specify uses and provides

Specify component

Connecting interface

Program configuration

- configuration file-name{..} (括號內可定義介面)
- implementation{..} (撰寫元件溝通的程式碼)
- component (宣告你將使用哪些元件)
- A.x -> B.x;
  - A元件宣告的uses x介面連接到B元件宣告的provides x介面
  - A透過x對B下命令，B也透過x回傳A事件

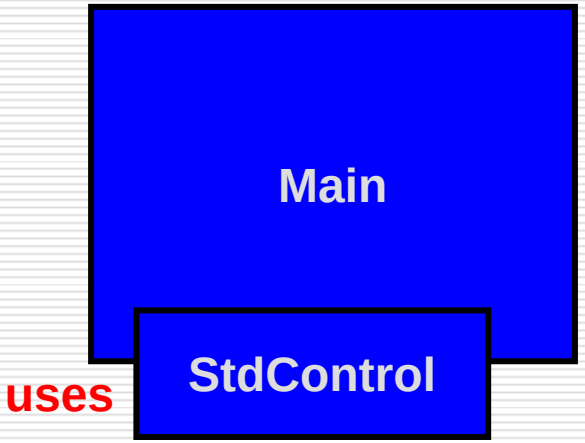
Program configuration

- **Blink.nc**

```
configuration Blink {
  }
  implementation {

    components Main, BlinkM, SingleTimer,LedsC;

    Main.StdControl -> BlinkM.StdControl;
    Main.StdControl -> SingleTimer.StdControl;
    BlinkM.Timer -> SingleTimer.Timer;
    BlinkM.Leds -> LedsC;
  }
```



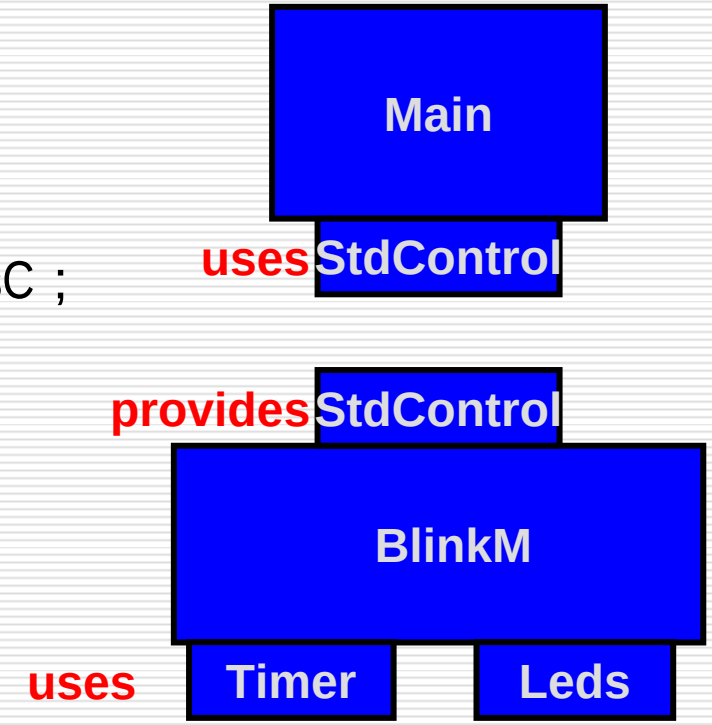
Program configuration

Blink.nc

```
configuration Blink {
}
implementation {

  components Main, BlinkM,SingleTimer, LedsC ;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

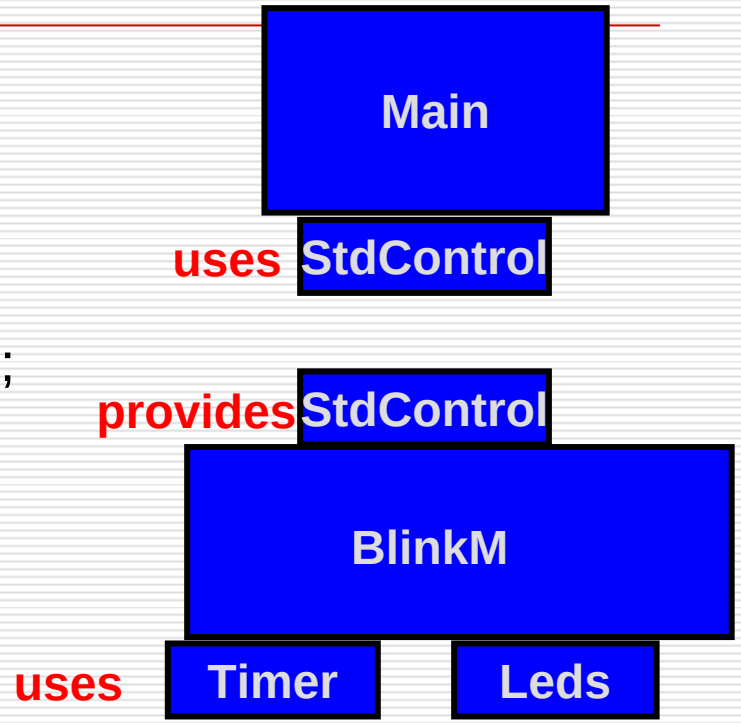


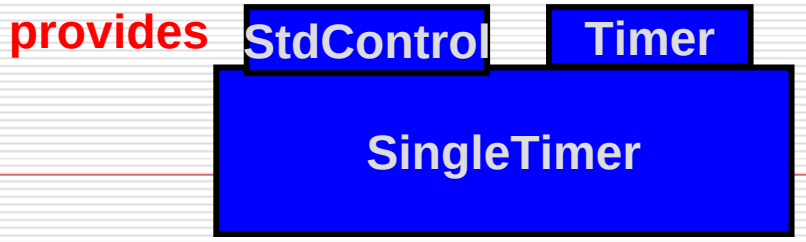
Program configuration

Blink.nc

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```



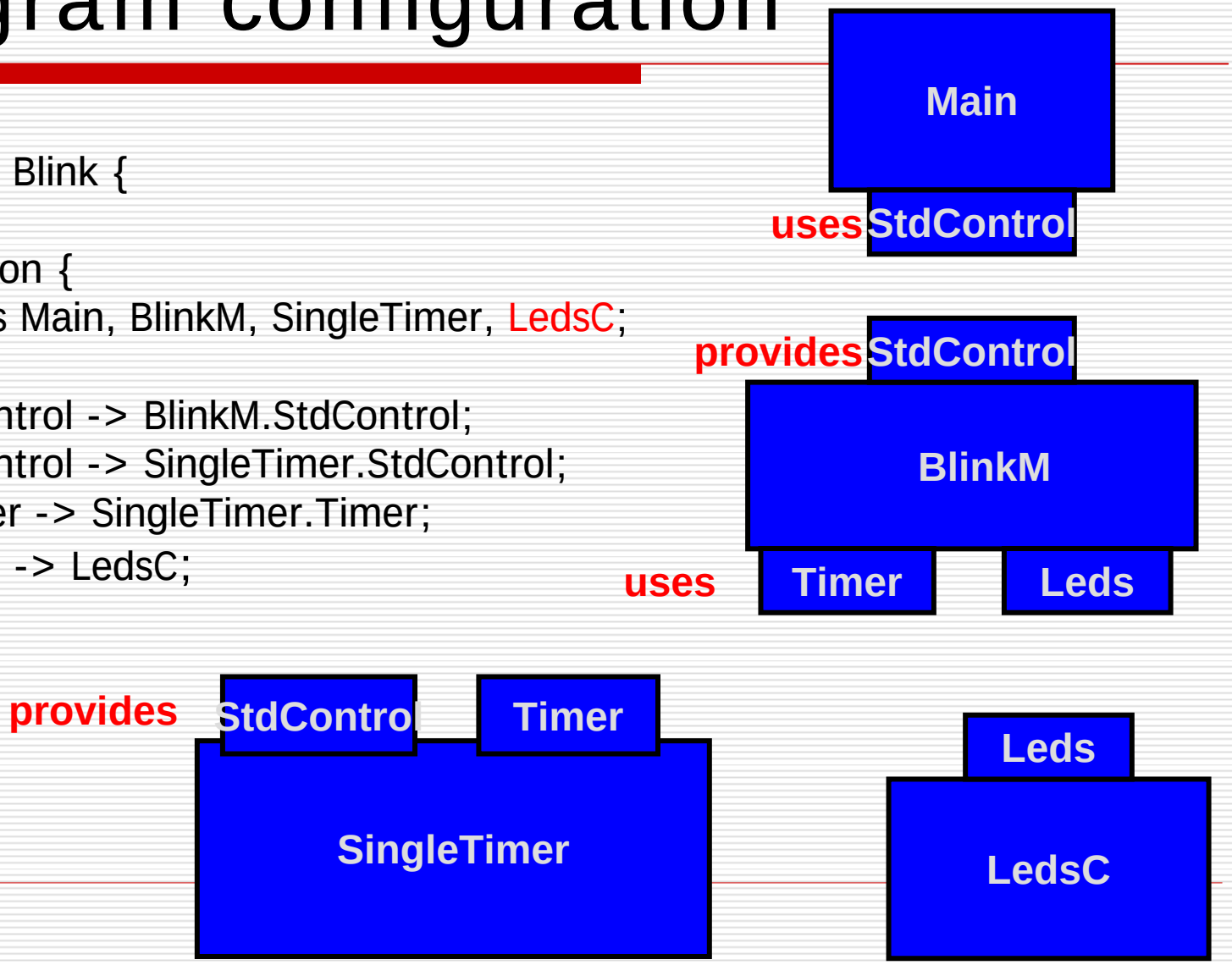


Program configuration

Blink.nc

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

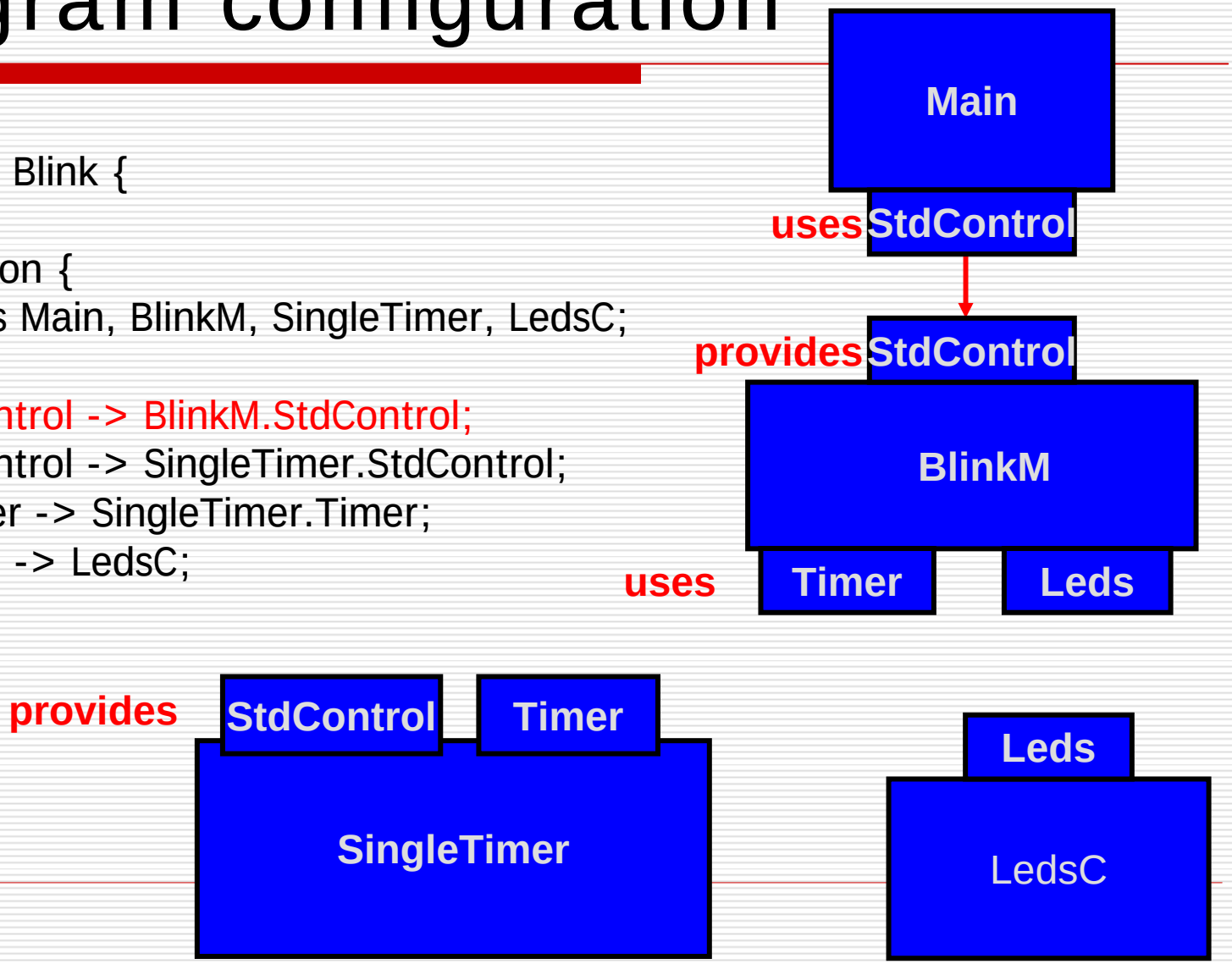


Program configuration

Blink.nc

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

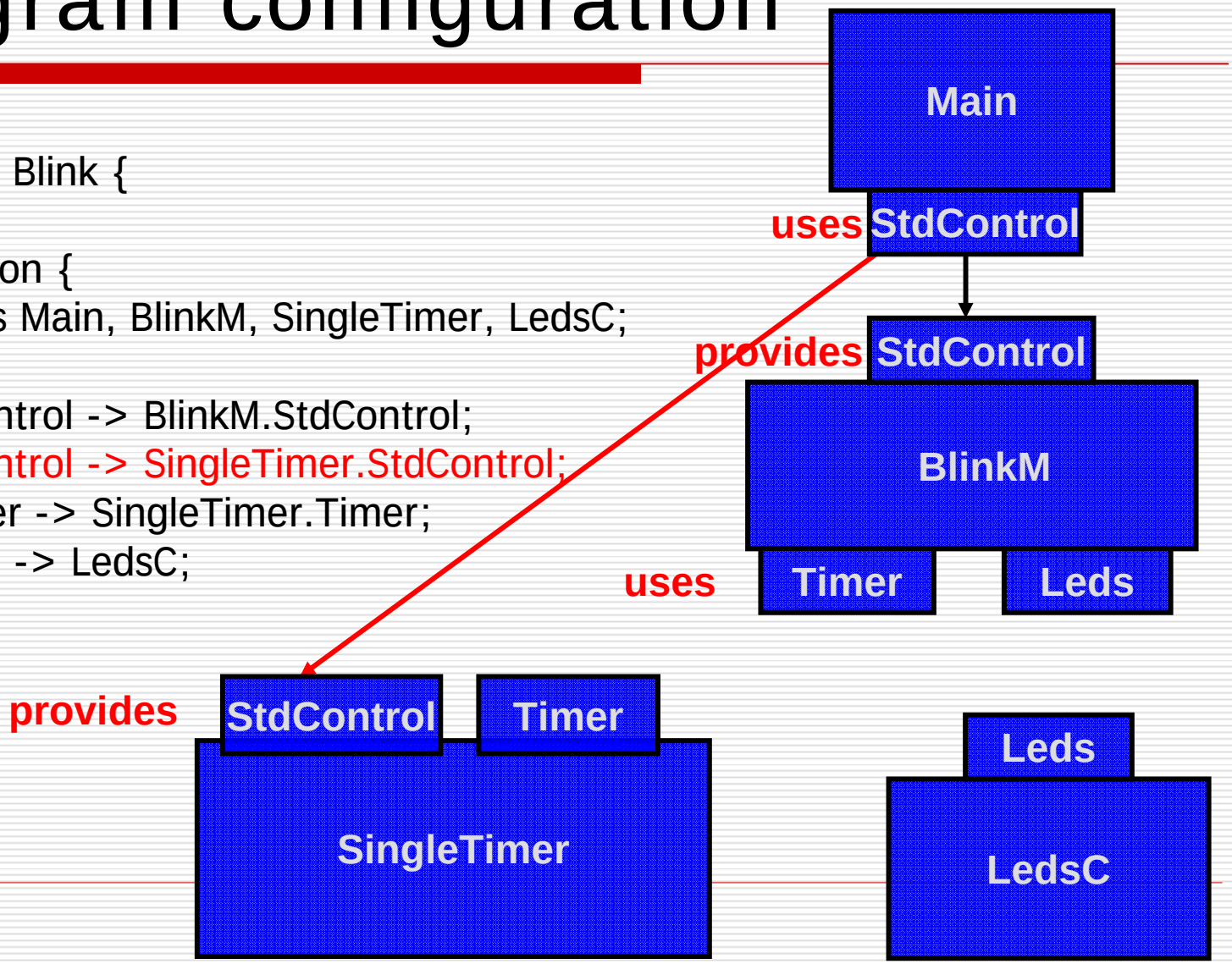


Program configuration

Blink.nc

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

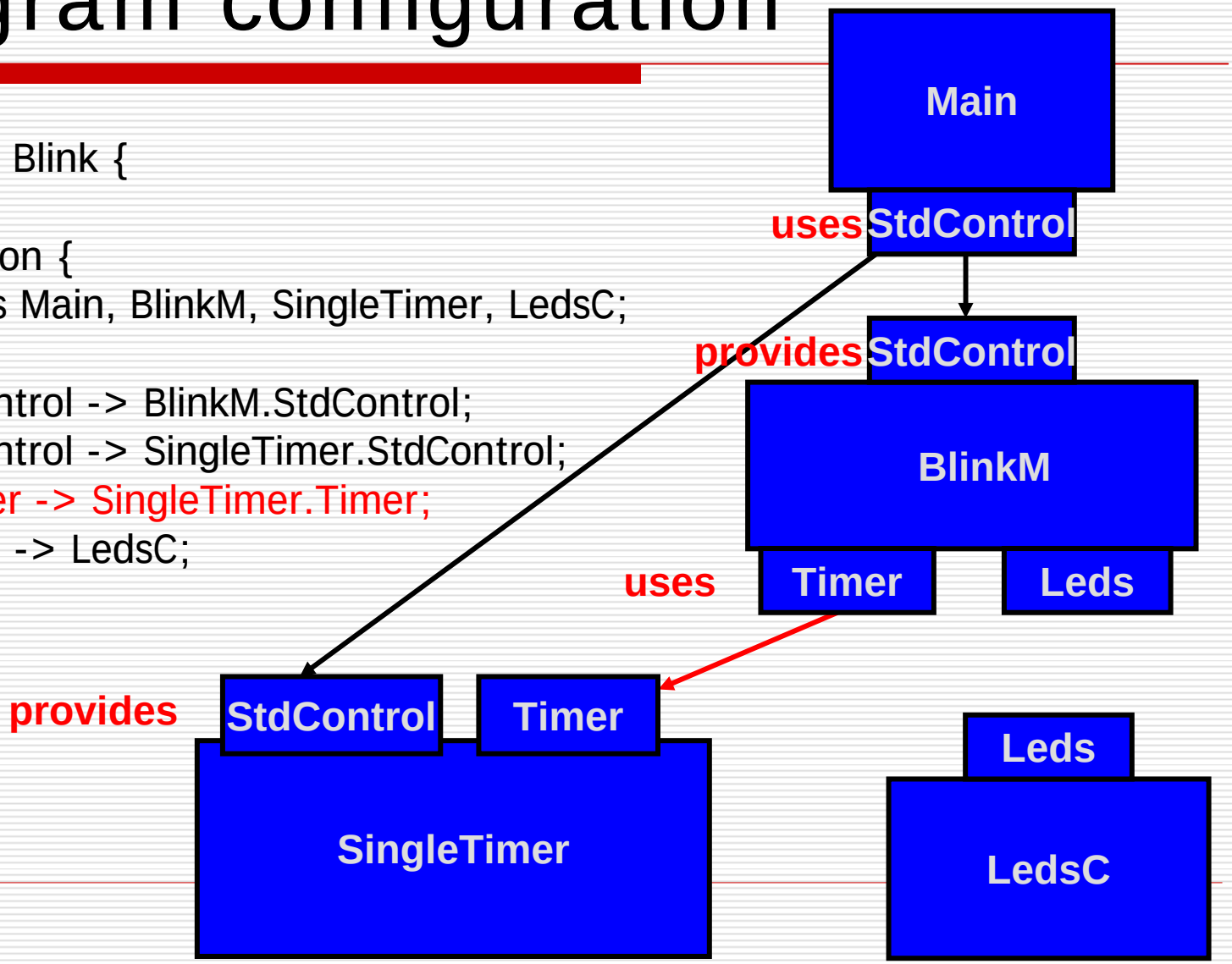


Program configuration

Blink.nc

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

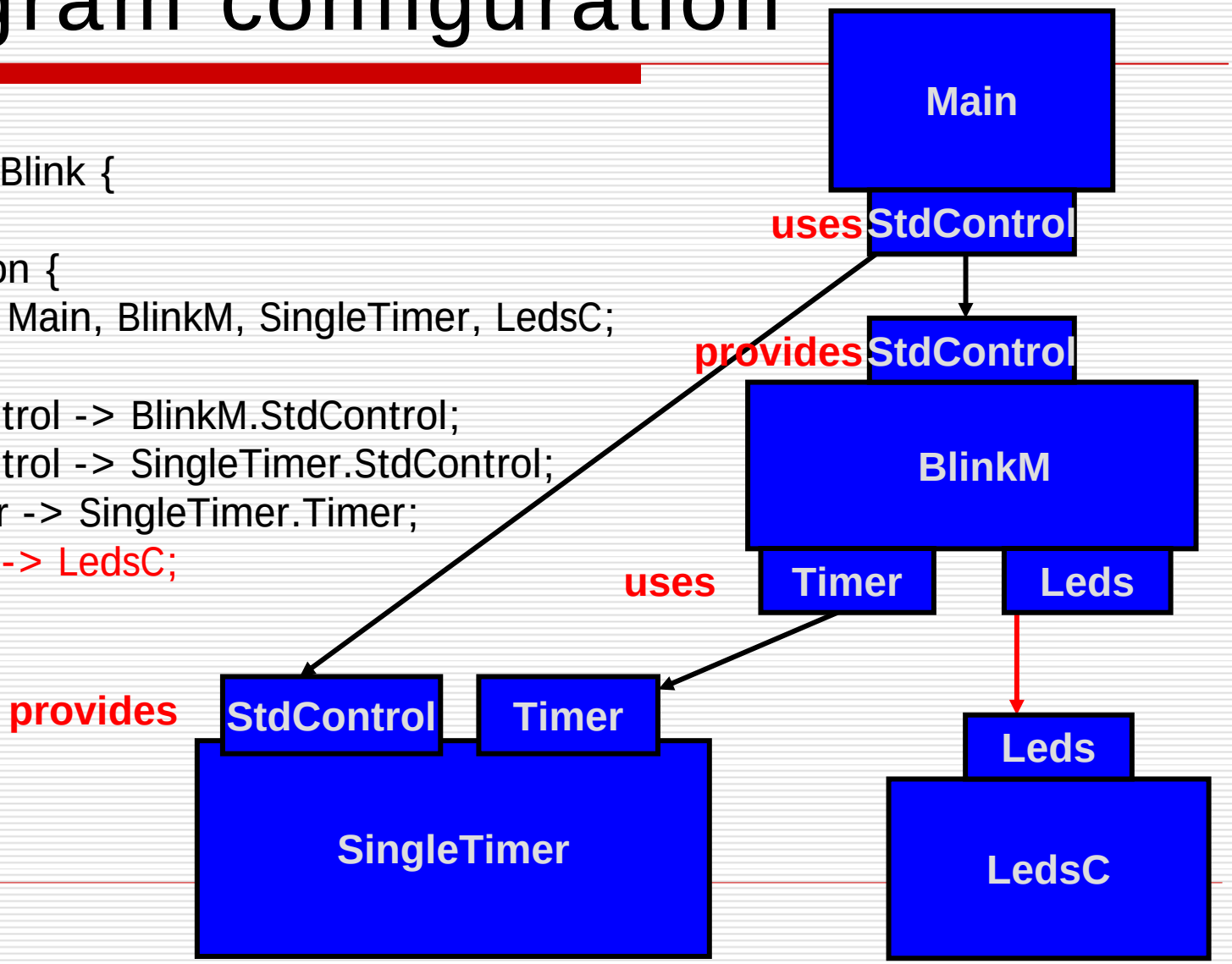


Program configuration

Blink.nc

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```



Program module

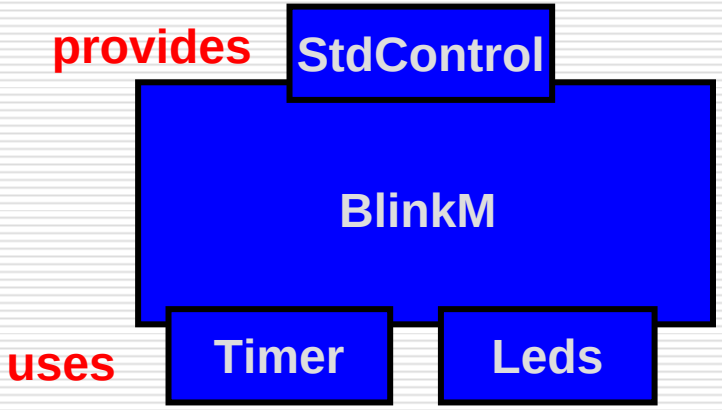
- configuration完成後，著手 BlinkM.nc
  - 開始撰寫如何實際操作你在configuration所宣告的物件的程式碼
- module{..} 內容宣告要和其他元件溝通的
- implementation 內容要先查看 BlinkM所宣告
  的介面，其定義了哪些function，才能得知你將要實際爲哪些function撰寫程式碼

Program module

BlinkM.nc

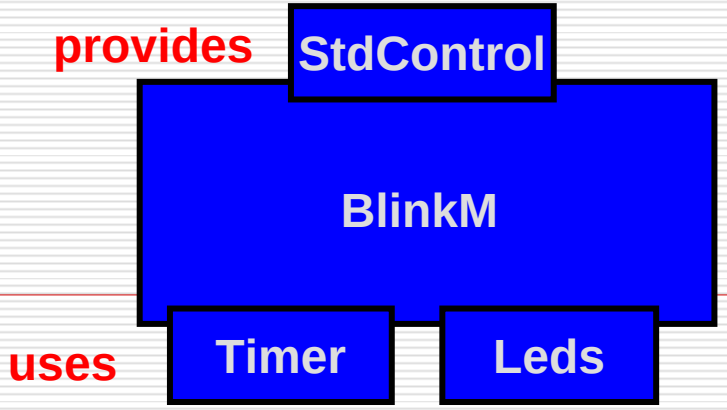
```
module BlinkM {
  provides {
    interface StdControl;
  }

  uses {
    interface Timer;
    interface Leds;
  }
}
```



Program module

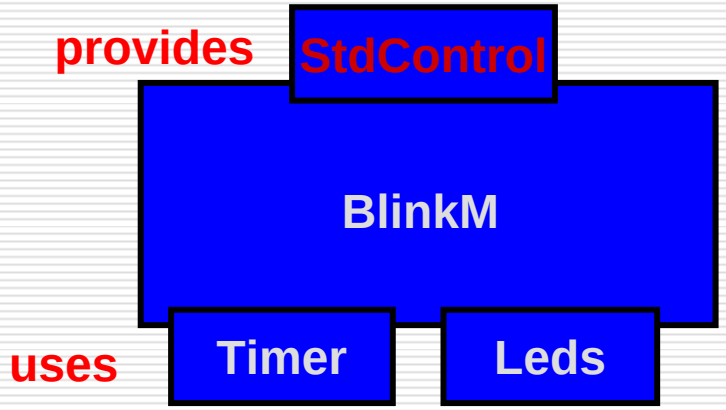
- 之前提到介面定義的function 可分為兩類：
  - command
  - event
- 所以我們要實作function的部份是
  - uses interface 的 event function
  - provides interface 的 command function



Program module

StdControl.nc

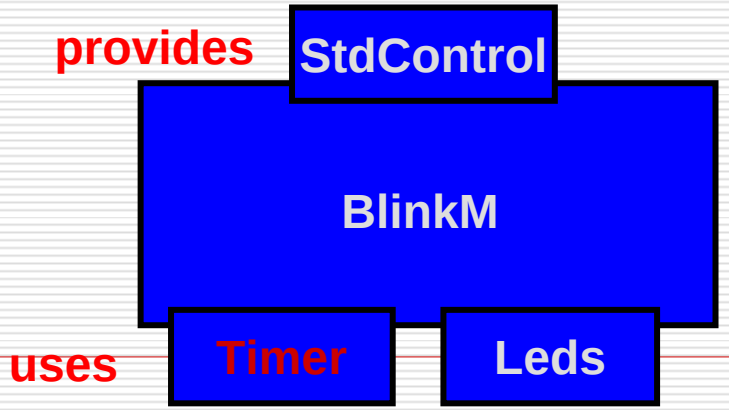
```
interface StdControl{
  command result_t init();
  command result_t start();
  command result_t stop();
}
```



Program module

Timer.nc

```
interface Timer{
 command result_t start(char type, uint32_t interval);
 command result_t stop();
 event result_t fired();
}
```

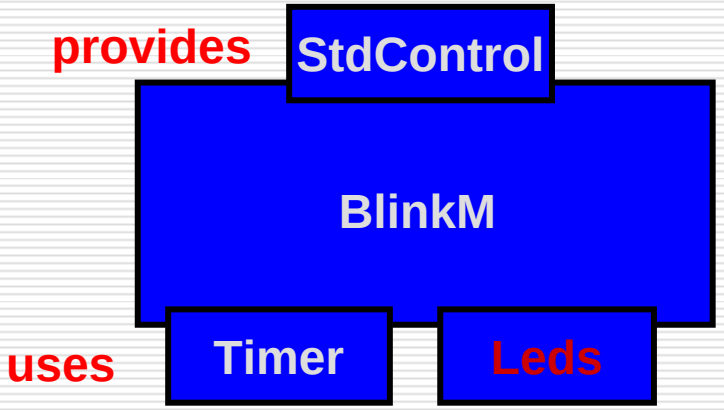


Program module

- command result_t start(char type, uint32_t interval)
  - Type有兩種:
    - TIMER_ONE_SHOT：經過一個interval單位，回傳fired()並暫停跳動
    - TIMER_REPEAT：不斷跳動一個interval單位，每回傳fired()，再跳動一個interval單位
  - Interval:啓動計時器 ，每interval單位 隨即回傳fired()，interval單位 ms

Program module

```
interface Leds {
  async command result_t init();
  async command result_t redOn();
  async command result_t redOff();
  async command result_t redToggle();
  async command result_t greenOn();
  async command result_t greenOff();
  async command result_t greenToggle();
  async command result_t yellowOn();
  async command result_t yellowOff();
  async command result_t yellowToggle();
  async command uint8_t get();
  async command result_t set(uint8_t value);
}
```



Program module

- 所以可以得知以下function需實作：

  command result_t StdControl.init();
  command result_t StdControl.start();
  command result_t StdControl.stop();
  event result_t Timer.fired();

- 要用到以下的command，使紅燈閃動

  async command result_t Leds.redToggle();

Program module

- **BlinkM.nc, continued**

```
implementation {
  command result_t StdControl.init() {
   call Leds.init();
  return SUCCESS;
  }
  command result_t StdControl.start() {
  return call Timer.start(TIMER_REPEAT, 1000) ;
   }

command result_t StdControl.stop() {
   return call Timer.stop()
  }

event result_t Timer.fired(){
   call Leds.redToggle();
  return SUCCESS;
  }
 }
```

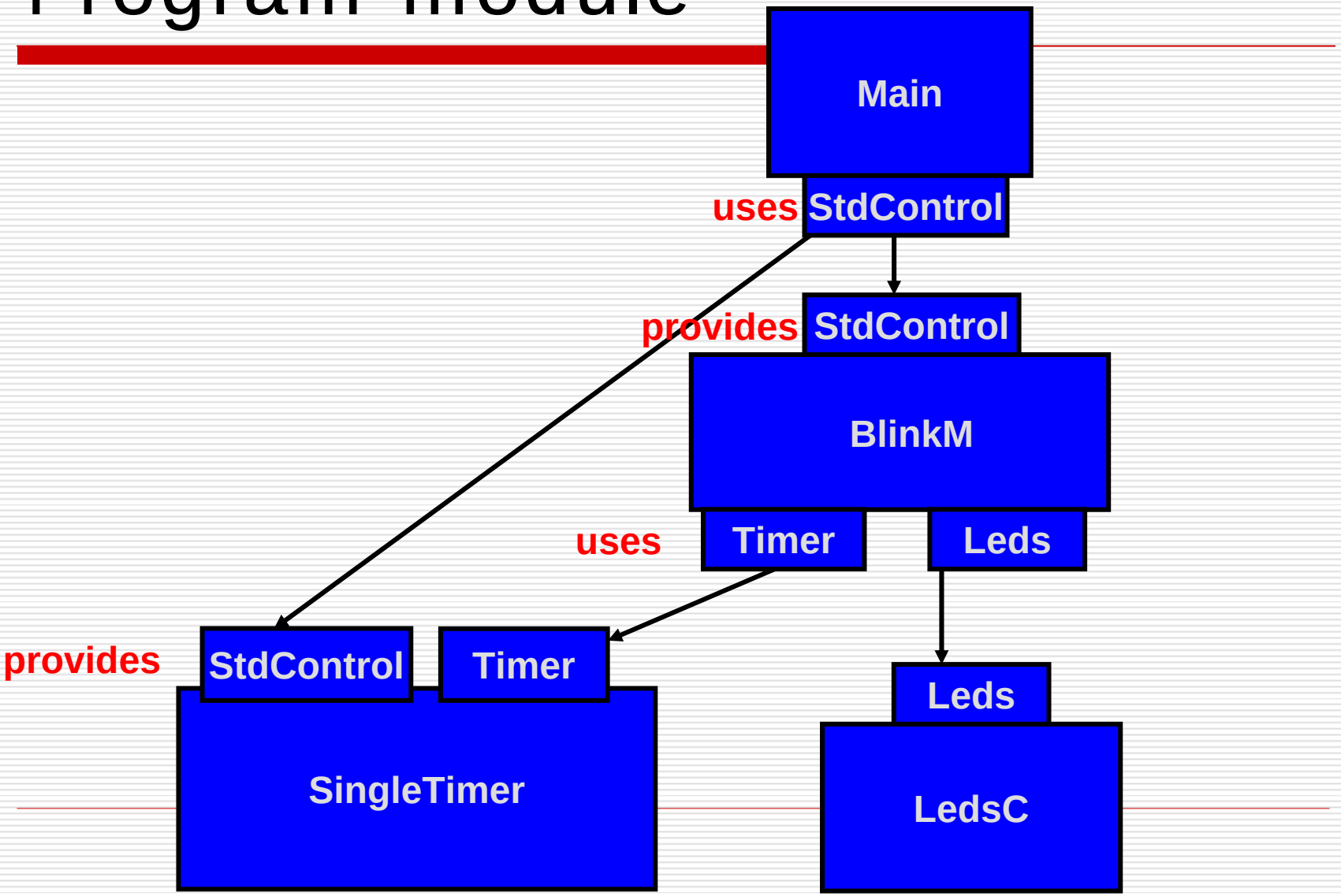
初始化LED燈

啟動計時器每1000ms跳動一次

停止計時器

閃紅燈

Program module



How to compiler nesC

make tmote <platform>

```
WMNLAB@WMNLAB /opt/tinyos-1.x/apps/blink
$ make tmote blink
mkdir -p build/tmote
    compiling Blink to a tmote binary
ncc -o build/tmote/main.exe -O -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-all
 -target=tmote -fnesc-cfile=build/tmote/app.c -board= -mdisable-hwmul -I/opt/mot
eiv/tos/platform/tmote -I/opt/moteiv/tos/platform/tmote/util/uartdetect -I/opt/m
oteiv/tos/platform/msp430/adc -I/opt/moteiv/tos/platform/msp430/dac -I/opt/motei
v/tos/platform/msp430/dma -I/opt/moteiv/tos/platform/msp430/resource -I/opt/mote
iv/tos/platform/msp430/timer -I/opt/moteiv/tos/platform/msp430 -I/opt/moteiv/tos
/lib/util/pool -I/opt/moteiv/tos/lib/util/button -I/opt/moteiv/tos/lib/util/null
 -I/opt/moteiv/tos/lib/util -I/opt/moteiv/tos/lib/MultiHopLQI -I/opt/moteiv/tos/
lib/netsync -I/opt/moteiv/tos/lib/sp -I/opt/moteiv/tos/lib/sp/cc2420 -I/opt/mote
iv/tos/lib/timer -I/opt/moteiv/tos/lib/resource -I/opt/moteiv/tos/lib/sched -I/o
pt/moteiv/tos/lib/Deluge -I/opt/moteiv/tos/lib/Flash/STM25P -I/opt/moteiv/tos/li
b/Flash -I/opt/moteiv/tos/lib/Spram -I/opt/moteiv/tos/interfaces -I/opt/moteiv/t
os/lib/CC2420Radio -I/opt/moteiv/tos/system -I/opt/moteiv/tinyos-1.x/tos/lib/CC2
420Radio -I/opt/moteiv/tinyos-1.x/tos/lib/Drip -fnesc-scheduler=TinySchedulerC,T
inySchedulerC.TaskBasic,TaskBasic,TaskBasic,runTask,postTask -Wl,--section-start
=.text=0x4800,--defsym=_reset_vector__=0x4000 -DLIB_DELUGE -DDELUGE_NUM_IMAGES=6
 -I%T/lib/Deluge -Wl,--section-start=.text=0x4800,--defsym=_reset_vector__=0x400
0 -DIDENT_PROGRAM_NAME=\"Blink\" -DIDENT_USER_ID=\"WMNLAB\" -DIDENT_HOSTNAME=\"W
MNLAB\" -DIDENT_USER_HASH=0xe5bd5188L -DIDENT_UNIX_TIME=0x457ca882L -DIDENT_UID_
HASH=0x22bee59eL Blink.nc -lm
    compiled Blink to build/tmote/main.exe
            2812 bytes in ROM
              43 bytes in RAM
msp430-objcopy --output-target=ihex build/tmote/main.exe build/tmote/main.ihex
    writing TOS image
```

Download program into mote

- Make tmote install

```
WMNLAB@WMNLAB /opt/tinyos-1.x/apps/blink
$ make tmote install
mkdir -p build/tmote
    compiling Blink to a tmote binary
ncc -o build/tmote/main.exe -O -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-all
 -target=tmote -fnesc-cfile=build/tmote/app.c -board= -mdisable-hwmul -I/opt/mot
eiv/tos/platform/tmote -I/opt/moteiv/tos/platform/tmote/util/uartdetect -I/opt/m
oteiv/tos/platform/msp430/adc -I/opt/moteiv/tos/platform/msp430/dac -I/opt/motei
v/tos/platform/msp430/dma -I/opt/moteiv/tos/platform/msp430/resource -I/opt/mote
iv/tos/platform/msp430/timer -I/opt/moteiv/tos/platform/msp430 -I/opt/moteiv/tos
/lib/util/pool -I/opt/moteiv/tos/lib/util/button -I/opt/moteiv/tos/lib/util/null
 -I/opt/moteiv/tos/lib/util -I/opt/moteiv/tos/lib/MultiHopLQI -I/opt/moteiv/tos/
lib/netsync -I/opt/moteiv/tos/lib/sp -I/opt/moteiv/tos/lib/sp/cc2420 -I/opt/mote
iv/tos/lib/timer -I/opt/moteiv/tos/lib/resource -I/opt/moteiv/tos/lib/sched -I/o
pt/moteiv/tos/lib/Deluge -I/opt/moteiv/tos/lib/Flash/STM25P -I/opt/moteiv/tos/li
b/Flash -I/opt/moteiv/tos/lib/Spram -I/opt/moteiv/tos/interfaces -I/opt/moteiv/t
os/lib/CC2420Radio -I/opt/moteiv/tos/system -I/opt/moteiv/tinyos-1.x/tos/lib/CC2
420Radio -I/opt/moteiv/tinyos-1.x/tos/lib/Drip -fnesc-scheduler=TinySchedulerC,T
inySchedulerC.TaskBasic,TaskBasic,TaskBasic,runTask,postTask -Wl,--section-start
=.text=0x4800,--defsym=_reset_vector__=0x4000 -DLIB_DELUGE -DDELUGE_NUM_IMAGES=6
 -I%T/lib/Deluge -Wl,--section-start=.text=0x4800,--defsym=_reset_vector__=0x400
0 -DIDENT_PROGRAM_NAME=\"Blink\" -DIDENT_USER_ID=\"WMNLAB\" -DIDENT_HOSTNAME=\"W
MNLAB\" -DIDENT_USER_HASH=0xe5bd5188L -DIDENT_UNIX_TIME=0x457cab41L -DIDENT_UID_
HASH=0x176a00a2L Blink.nc -lm
    compiled Blink to build/tmote/main.exe
            2812 bytes in ROM
              43 bytes in RAM
```

Generate program's structure documentation

```
msp430-objcopy --output-target=ihex build/tmote/main.exe build/tmote/main.ihex
    writing TOS image
cp build/tmote/main.ihex build/tmote/main.ihex.out
    found mote on COM5 (using bsl,auto)
    installing tmote bootloader using bsl
tos-bsl --telosb -c 4 -r -e -I -p C:/cygwin/opt/moteiv/tos/lib/Deluge/TOSBoot/bu
ild/tmote/main.ihex
MSP430 Bootstrap Loader Version: 1.39-telos-8
Mass Erase...
Transmit default password ...
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
Program ...
1774 bytes programmed.
Reset device ...
    installing tmote binary using bsl (without mass erase)
tos-bsl --telosb -c 4 -r -I -p build/tmote/main.ihex.out
MSP430 Bootstrap Loader Version: 1.39-telos-8
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
Program ...
2844 bytes programmed.
Reset device ...
rm -f build/tmote/main.exe.out build/tmote/main.ihex.out
```

Conclusion

- C:\tinyos\cygwin\opt\tinyos1.x\doc\nesc\ref.pdf
  - nesC 1.1 Language Reference Manual