



Università degli Studi dell'Aquila

Centro di Eccellenza DEWS

Design Methodologies for Embedded controllers, Wireless interconnect and System-on-chip

Wireless Sensor Networks

Examples nesC/TinyOS



Overview

- Examples nesC/TinyOS
 - Blink
 - BlinkTask
 - RadioCountToLeds
 - RadioSenseToLeds
 - Oscilloscope

Examples nesC/TinyOS

Blink (TinyOS 2.x)



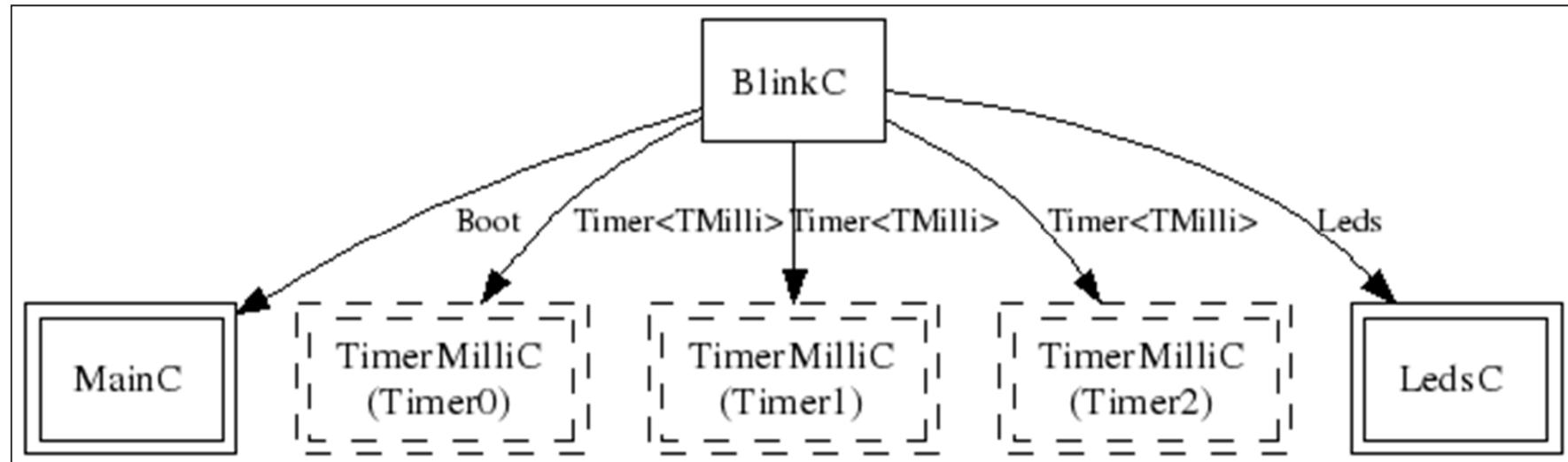
Blink

- This app lets 3 leds of a sensor node blink with frequencies of 1Hz, 2 Hz, e 4 Hz
 - Applicative components
 - *BlinkAppC (Configuration)*
 - *BlinkC (Module)*
 - System components
 - *MainC, LedsC, TimerMilliC*



Blink

- BlinkAppC: component graph



	Singleton	Generic
Module		
Configuration		



Blink

- BlinkAppC.nc

```
configuration BlinkAppC
{
}
implementation
{
    components MainC, BlinkC, LedsC;
    components new TimerMilliC() as Timer0;
    components new TimerMilliC() as Timer1;
    components new TimerMilliC() as Timer2;

    BlinkC -> MainC.Boot;

    BlinkC.Timer0 -> Timer0;
    BlinkC.Timer1 -> Timer1;
    BlinkC.Timer2 -> Timer2;
    BlinkC.Leds -> LedsC;
}
```



Blink

- BlinkC.nc

```
#include "Timer.h"

module BlinkC
{
    uses interface Timer<TMilli> as Timer0;
    uses interface Timer<TMilli> as Timer1;
    uses interface Timer<TMilli> as Timer2;
    uses interface Leds;
    uses interface Boot;
}
implementation
{
```



Blink

- BlinkC.nc

```
event void Boot.booted()
{
    call Timer0.startPeriodic( 250 );
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
}
```



Blink

- BlinkC.nc

```
event void Timer0.fired()
{
    dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
    call Leds.led0Toggle();
}

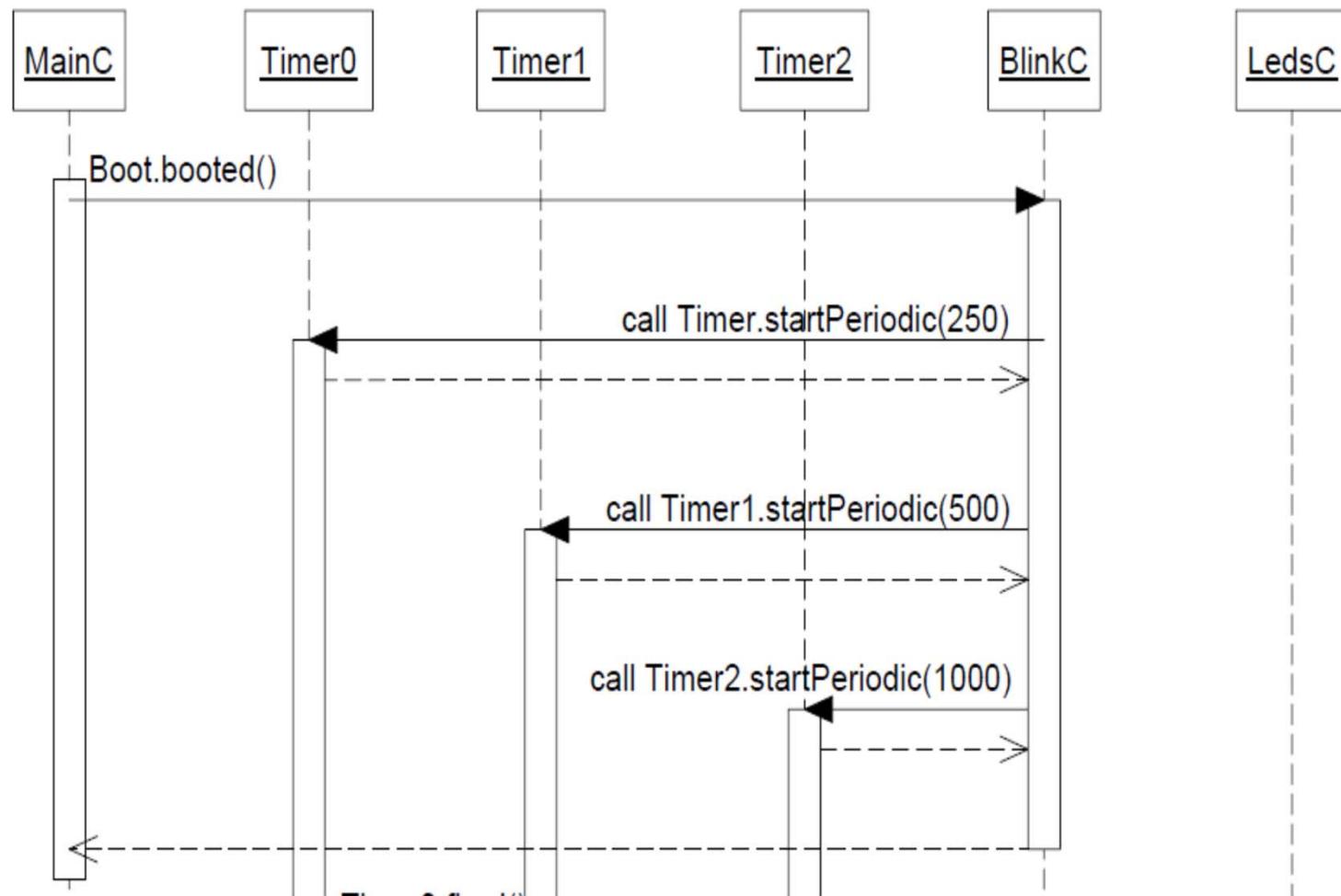
event void Timer1.fired()
{
    dbg("BlinkC", "Timer 1 fired @ %s \n", sim_time_string());
    call Leds.led1Toggle();
}

event void Timer2.fired()
{
    dbg("BlinkC", "Timer 2 fired @ %s.\n", sim_time_string());
    call Leds.led2Toggle();
}
```



Blink

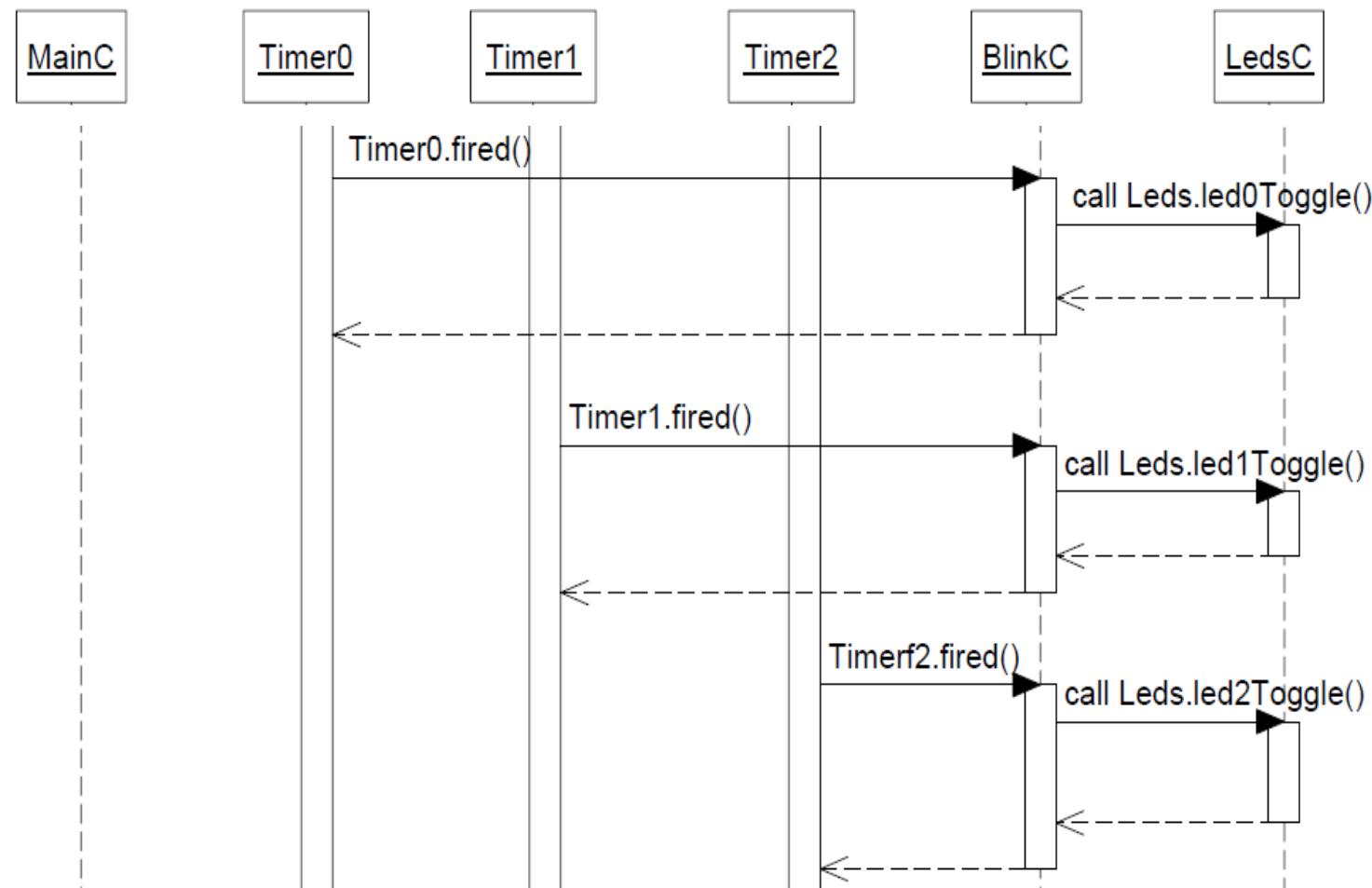
- Sequence diagram: INIT





Blink

- Sequence diagram: periodic activity



Examples nesC/TinyOS

BlinkTask (TinyOS 2.x)



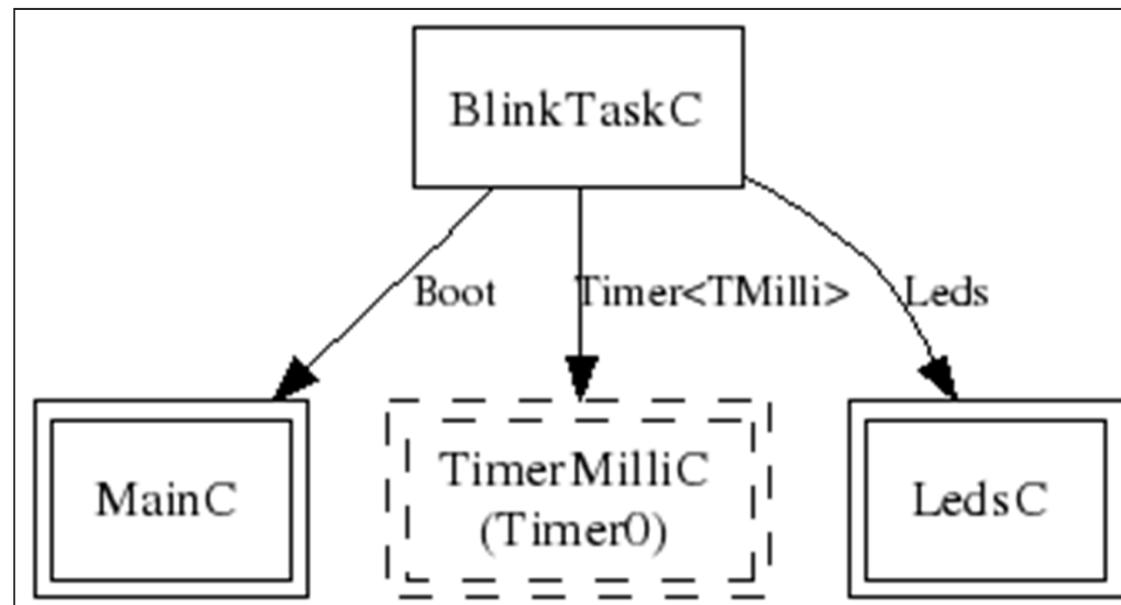
BlinkTask

- This app lets 1 led of a sensor node blink with frequencies of 1Hz by means of a TinyOS task
 - Applicative components
 - *BlinkTaskAppC (Configuration)*
 - *BlinkTaskC (Module)*
 - System components
 - *MainC, LedsC, TimerMilliC*



BlinkTask

- BlinkTaskAppC: component graph





BlinkTask

- BlinkTaskAppC.nc

```
configuration BlinkAppC
{
}
implementation
{
    components MainC, BlinkTaskC, LedsC;
    components new TimerMilliC() as Timer0;

    BlinkTaskC -> MainC.Boot;
    BlinkTaskC.Timer0 -> Timer0;
    BlinkTaskC.Leds -> LedsC;
}
```



BlinkTask

- BlinkTaskC.nc

```
#include "Timer.h"

module BlinkTaskC
{
    uses interface Timer<TMilli> as Timer0;
    uses interface Leds;
    uses interface Boot;
}
implementation
{
    task void toggle()
    {
        call Leds.led0Toggle();
    }
}
```



BlinkTask

- BlinkTaskC.nc

```
event void Boot.booted()
{
    call Timer0.startPeriodic( 1000 );
}

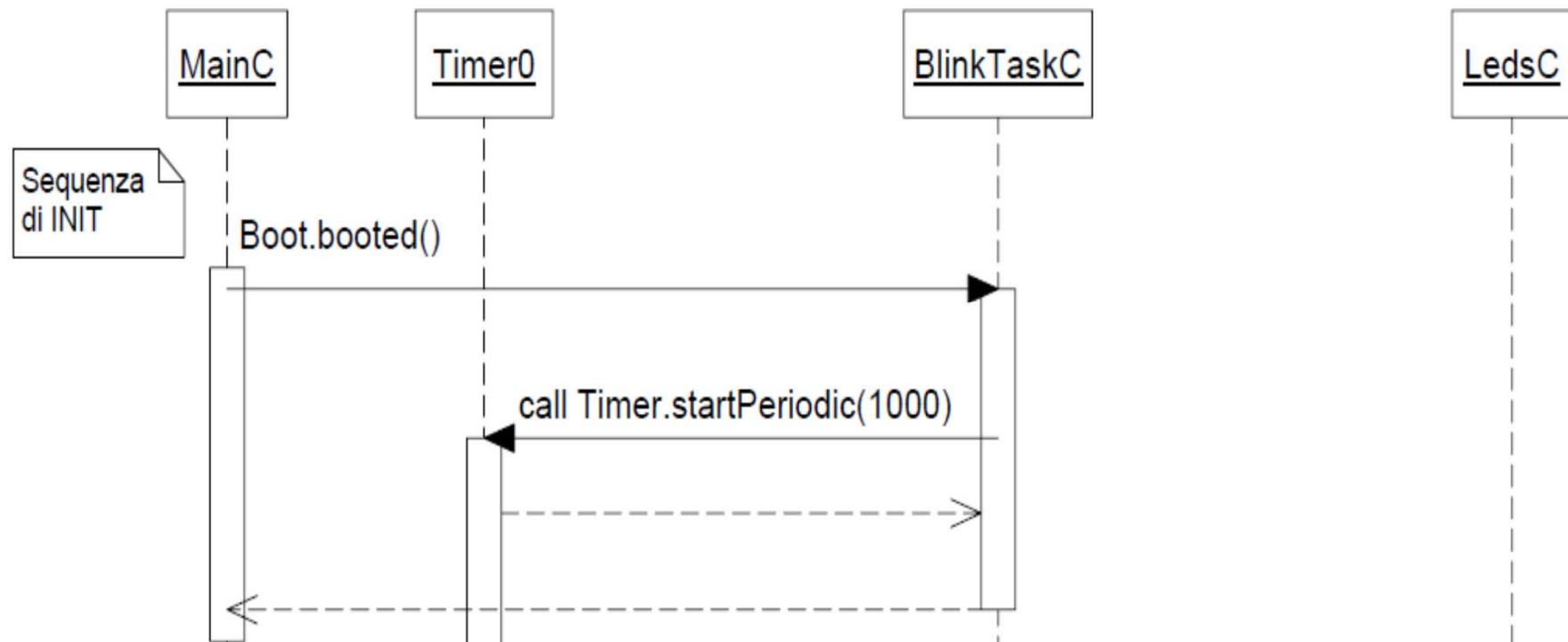
event void Timer0.fired()
{
    post toggle();
}

}
```



BlinkTask

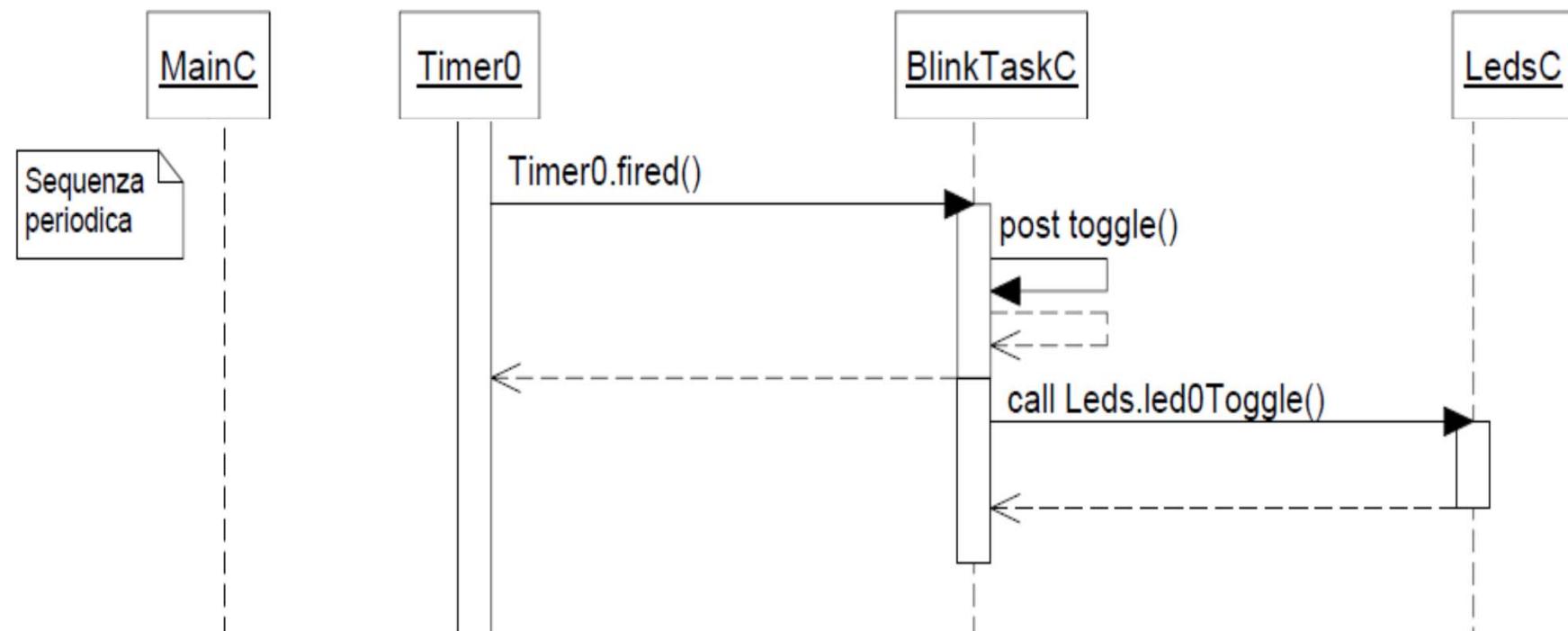
- Sequence diagram: INIT





BlinkTask

- Sequence diagram: periodic activity



Examples nesC/TinyOS

RadioCountToLeds (TinyOS 2.x)



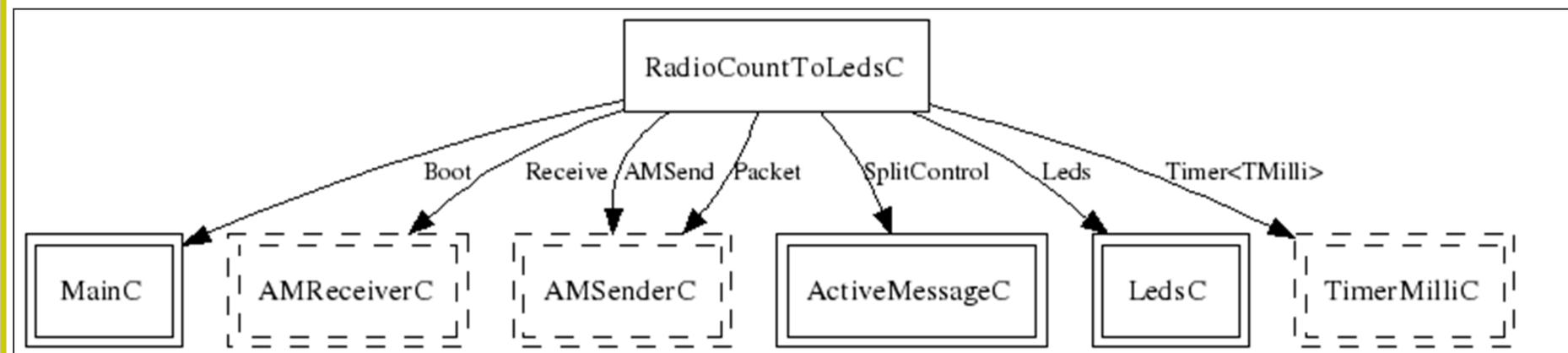
RadioCountToLeds

- This app periodically (4Hz) increments a counter while sending each new value by radio (broadcast)
 - Receiving nodes visualize received value 3 LSB by leds
 - Applicative component
 - *RadioCountToLedsAppC (Configuration)*
 - *RadioCountToLedsC (Module)*
 - System component
 - *MainC, LedsC, TimerMilliC*
 - *AMReceiverC, AMSenderC, ActiveMessageC*



RadioCountToLeds

- RadioCountToLedsAppC: component graph





RadioCountToLeds

- RadioCountToLeds.h

```
typedef nx_struct radio_count_msg {
    nx_uint16_t counter;
} radio_count_msg_t;

enum {
    AM_RADIO_COUNT_MSG = 6,
};
```



RadioCountToLeds

- RadioCountToLedsAppC.nc

```
#include "RadioCountToLeds.h"

configuration RadioCountToLedsAppC {}
implementation {
    components MainC, RadioCountToLedsC as App, LedsC;
    components new AMSenderC(AM_RADIO_COUNT_MSG);
    components new AMReceiverC(AM_RADIO_COUNT_MSG);
    components new TimerMilliC();
    components ActiveMessageC;

    App.Boot -> MainC.Boot;
    App.Receive -> AMReceiverC;
    App.AMSend -> AMSenderC;
    App.AMControl -> ActiveMessageC;
    App.Leds -> LedsC;
    App.MilliTimer -> TimerMilliC;
    App.Packet -> AMSenderC;
}
```



RadioCountToLeds

- RadioCountToLedsC.nc

```
#include "Timer.h"
#include "RadioCountToLeds.h"

module RadioCountToLedsC {
    uses {
        interface Leds;
        interface Boot;
        interface Receive;
        interface AMSend;
        interface Timer<TMilli> as MilliTimmer;
        interface SplitControl as AMControl;
        interface Packet;
    }
}
implementation {
    message_t packet;
    bool locked;
    uint16_t counter = 0;
```



RadioCountToLeds

- RadioCountToLedsC.nc

```
event void Boot.booted() {
    call AMControl.start();
}

event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
        call MilliTimmer.startPeriodic(250);
    }
    else {
        call AMControl.start();
    }
}

event void AMControl.stopDone(error_t err) {
    // do nothing
}
```



RadioCountToLeds

- RadioCountToLedsC.nc

```
event void MilliTimmer.fired() {
    counter++;
    if (locked) return;
    else
    {
        radio_count_msg_t* rcm =
        (radio_count_msg_t*) call Packet.getPayload(&packet, NULL);

        if (call Packet.maxPayloadLength() < sizeof(radio_count_msg_t))
            return;

        rcm->counter = counter;

        if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
            sizeof(radio_count_msg_t)) == SUCCESS)
            locked = TRUE;
    }
}
```



RadioCountToLeds

- RadioCountToLedsC.nc

```
event void AMSend.sendDone(message_t* bufPtr, error_t error) {  
    if (&packet == bufPtr) {locked = FALSE;}  
}  
}
```



RadioCountToLeds

- RadioCountToLedsC.nc

```
event message_t*
Receive.receive(message_t* bufPtr, void* payload, uint8_t len){
    if (len != sizeof(radio_count_msg_t)) {return bufPtr;}
    else{
        radio_count_msg_t* rcm = (radio_count_msg_t*)payload;

        if (rcm->counter & 0x1) call Leds.led0On();
        else call Leds.led0Off();

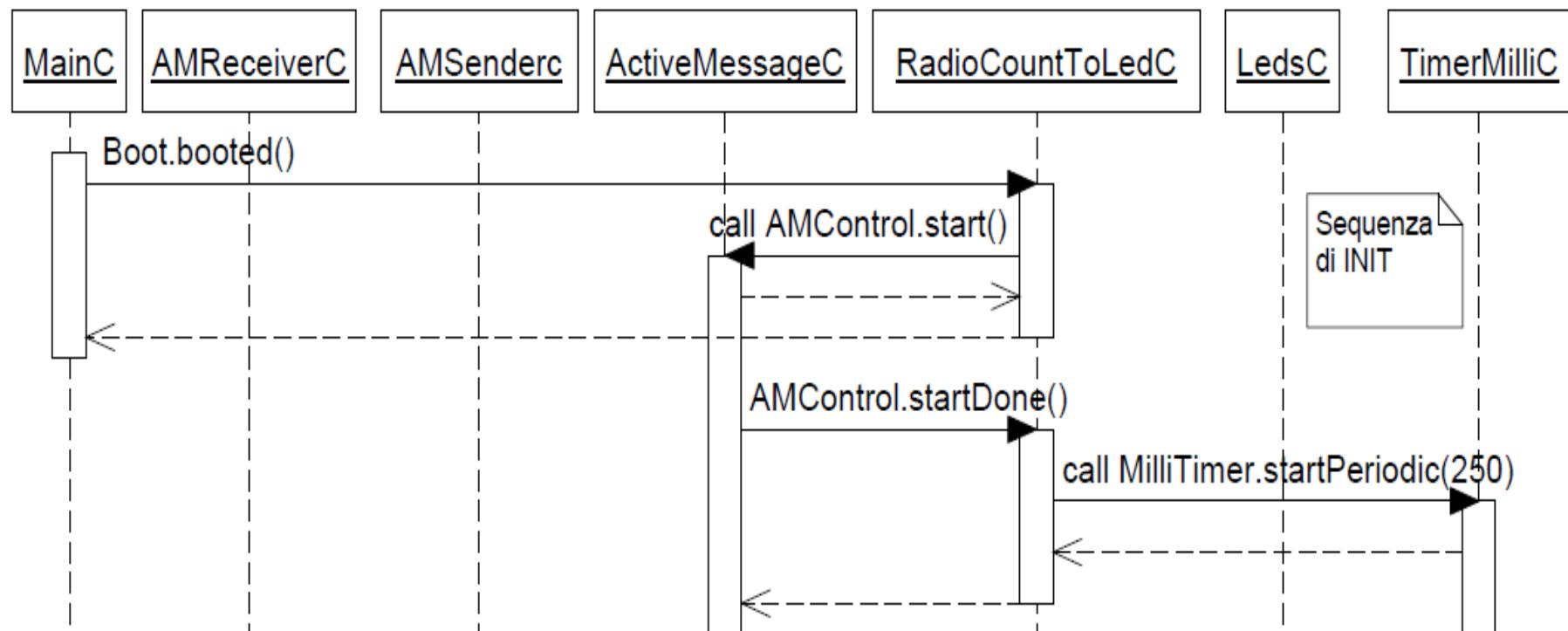
        if (rcm->counter & 0x2) call Leds.led1On();
        else call Leds.led1Off();

        if (rcm->counter & 0x4) call Leds.led2On();
        else call Leds.led2Off();
    }
    return bufPtr;
}
```



RadioCountToLeds

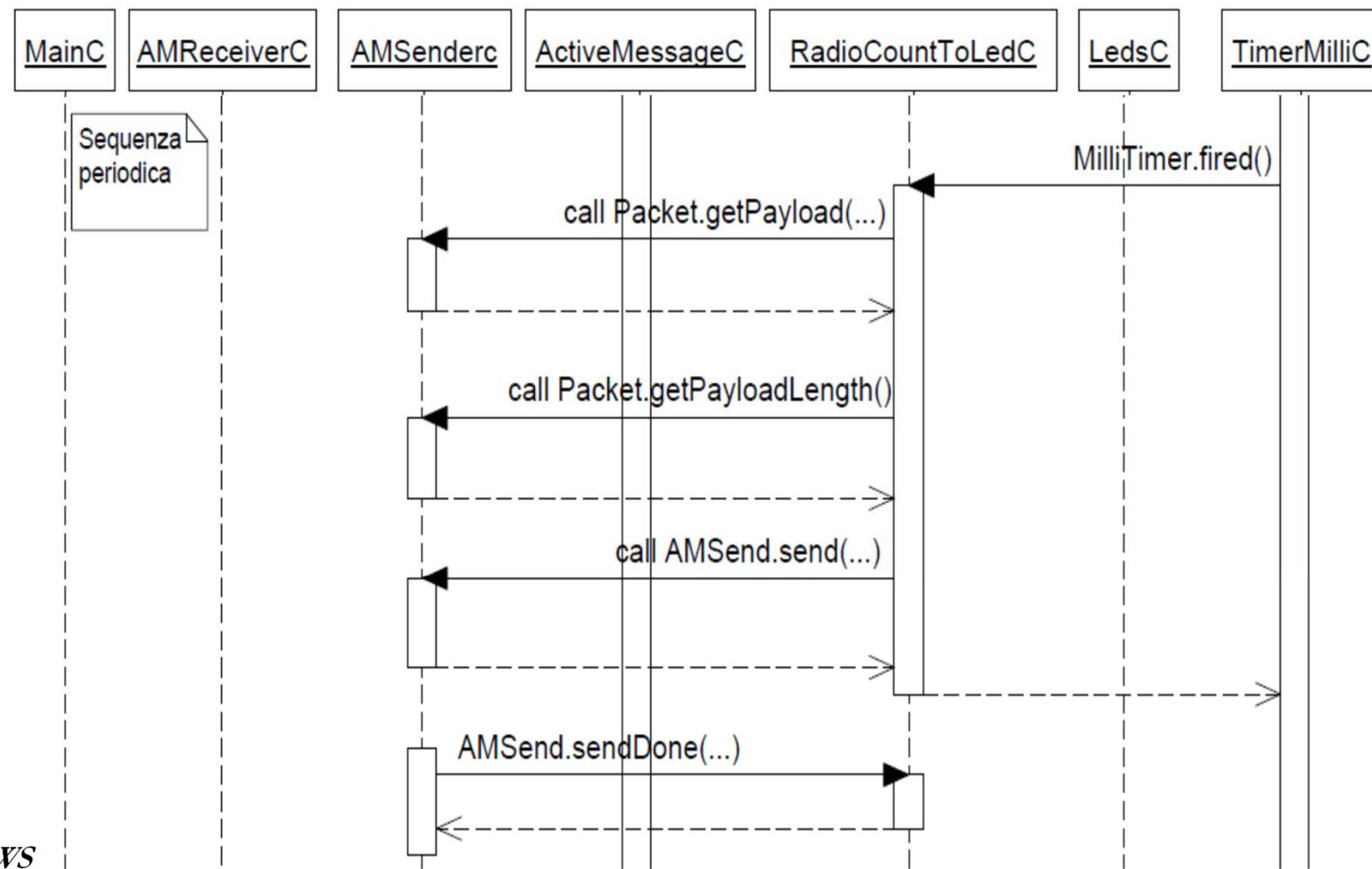
- Sequence diagram: INIT





RadioCountToLeds

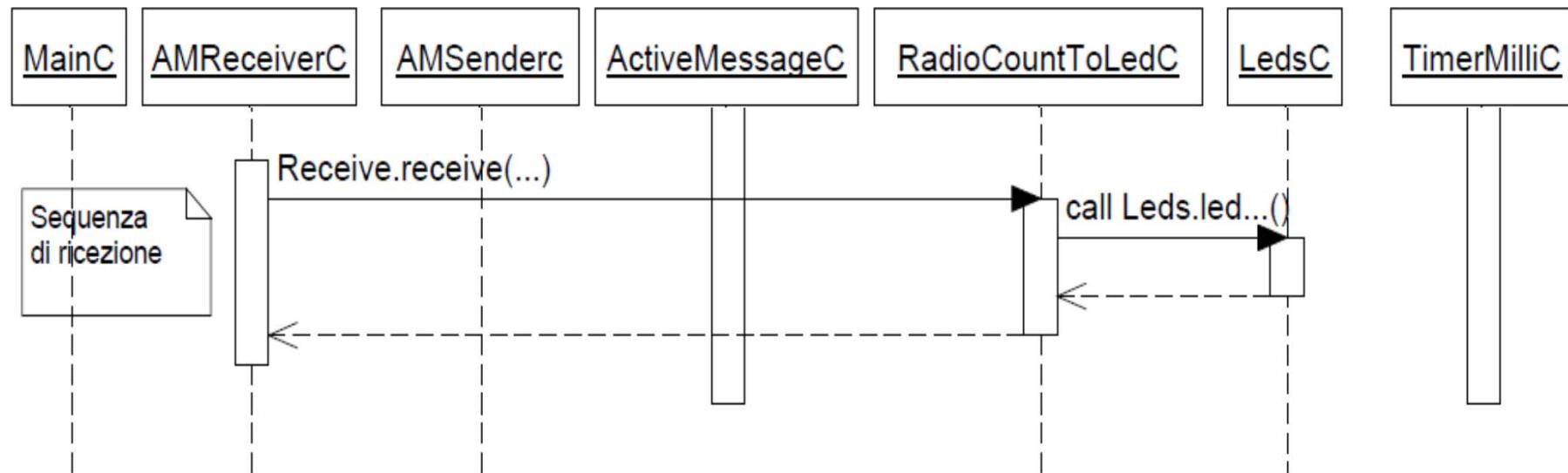
- Sequence diagram: periodic activity





RadioCountToLeds

- Sequence diagram: message reception



Examples nesC/TinyOS

RadioSenseToLeds (TinyOS 2.x)



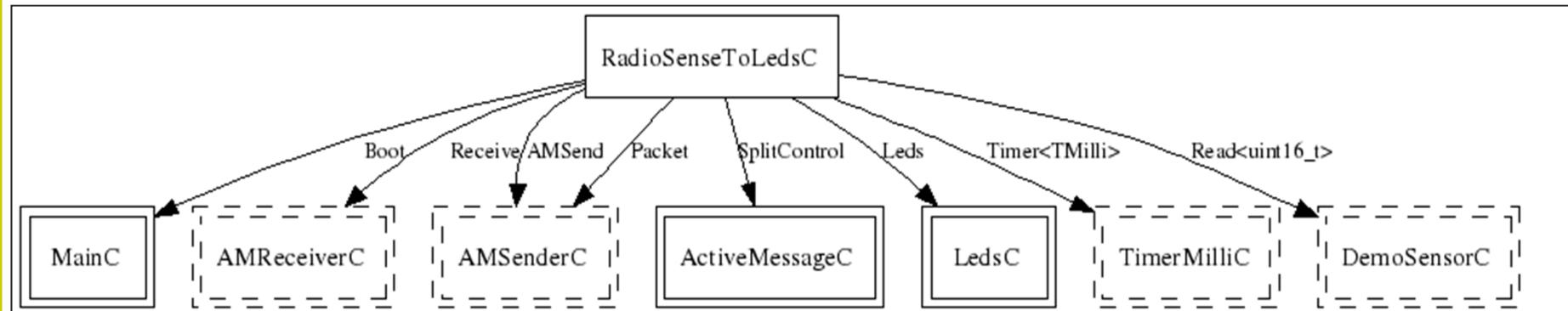
RadioSenseToLeds

- This app periodically (4Hz) samples a sensor while sending each read value by radio (broadcast)
 - Receiving nodes visualize received value 3 LSB by leds
 - Applicative component
 - *RadioSenseToLedsAppC (Configuration)*
 - *RadioSenseToLedsC (Module)*
 - System components
 - *MainC, LedsC, TimerMilliC, DemoSensorC*
 - *AMReceiverC, AMSenderC, ActiveMessageC*



RadioSenseToLeds

- RadioSenseToLedsAppC: component graph





RadioSenseToLeds

- RadioSenseToLeds.h

```
typedef nx_struct radio_sense_msg {
    nx_uint16_t error;
    nx_uint16_t data;
} radio_sense_msg_t;

enum {
    AM_RADIO_SENSE_MSG = 7,
};
```



RadioSenseToLeds

- RadioSenseToLedsAppC.nc

```
#include "RadioSenseToLeds.h"
configuration RadioSenseToLedsAppC {}
implementation {
    components MainC, RadioSenseToLedsC as App, LedsC, new DemoSensorC();
    components ActiveMessageC;
    components new AMSenderC(AM_RADIO_SENSE_MSG);
    components new AMReceiverC(AM_RADIO_SENSE_MSG);
    components new TimerMilliC();

    App.Boot -> MainC.Boot;
    App.Receive -> AMReceiverC;
    App.AMSend -> AMSenderC;
    App.RadioControl -> ActiveMessageC;
    App.Leds -> LedsC;
    App.MilliTimer -> TimerMilliC;
    App.Packet -> AMSenderC;
    App.Read -> DemoSensorC;
}
```



RadioSenseToLeds

- RadioSenseToLedsC.nc

```
#include "Timer.h"
#include "RadioSenseToLeds.h"
module RadioSenseToLedsC {
    uses {
        interface Leds;
        interface Boot;
        interface Receive;
        interface AMSend;
        interface Timer<TMilli> as MilliTimmer;
        interface Packet;
        interface Read<uint16_t>;
        interface SplitControl as RadioControl;
    }
}
implementation {
    message_t packet;
    bool locked = FALSE;
```



RadioSenseToLeds

- RadioSenseToLedsC.nc

```
event void Boot.booted() {
    call RadioControl.start();
}

event void RadioControl.startDone(error_t err) {
    if (err == SUCCESS) {
        call MilliTimmer.startPeriodic(250);
    }
}

event void RadioControl.stopDone(error_t err) {}
```



RadioSenseToLeds

- RadioSenseToLedsC.nc

```
event void MilliTimmer.fired() {call Read.read();}

event void Read.readDone(error_t result, uint16_t data) {
    if (locked) return;
    else
    { radio_sense_msg_t* rsm;

        rsm = (radio_sense_msg_t*)call Packet.getPayload(&packet, NULL);
        if (call Packet.maxPayloadLength() < sizeof(radio_sense_msg_t))
            return;

        rsm->error = result;
        rsm->data = data;
        if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
            sizeof(radio_sense_msg_t)) == SUCCESS) locked = TRUE;
    }
}
```



RadioSenseToLeds

- RadioSenseToLedsC.nc

```
event void AMSend.sendDone(message_t* bufPtr, error_t error) {  
    if (&packet == bufPtr) {locked = FALSE;}  
}  
}
```



RadioSenseToLeds

- RadioSenseToLedsC.nc

```
event message_t*
Receive.receive(message_t* bufPtr, void* payload, uint8_t len){
    if (len != sizeof(radio_sense_msg_t)) {return bufPtr;}
    else{
        radio_sense_msg_t* rcm = (radio_sense_msg_t*)payload;

        if (rcm->data & 0x0001) call Leds.led0On();
        else call Leds.led0Off();

        if (rcm->data & 0x0002) call Leds.led1On();
        else call Leds.led1Off();

        if (rcm->data & 0x0004) call Leds.led2On();
        else call Leds.led2Off();
    }
    return bufPtr;
}
```



RadioSenseToLeds

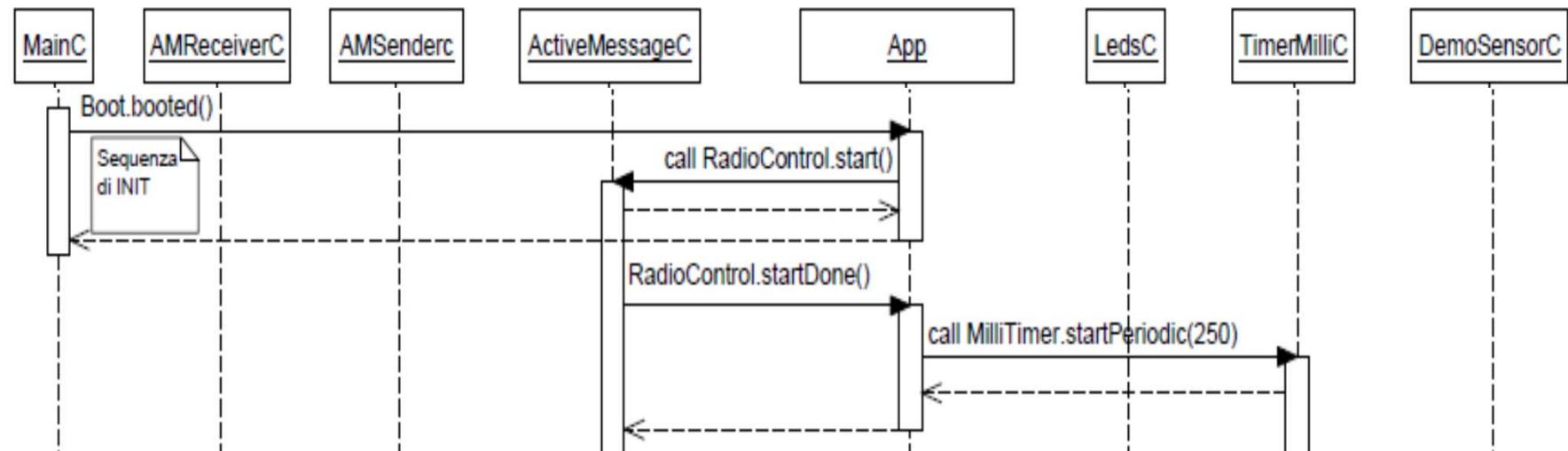
- DemoSensorC.nc

```
generic configuration DemoSensorC()
{
    provides interface Read<uint16_t>;
}
implementation
{
    components new VoltageC() as DemoChannel;
    Read = DemoChannel;
}
```



RadioSenseToLeds

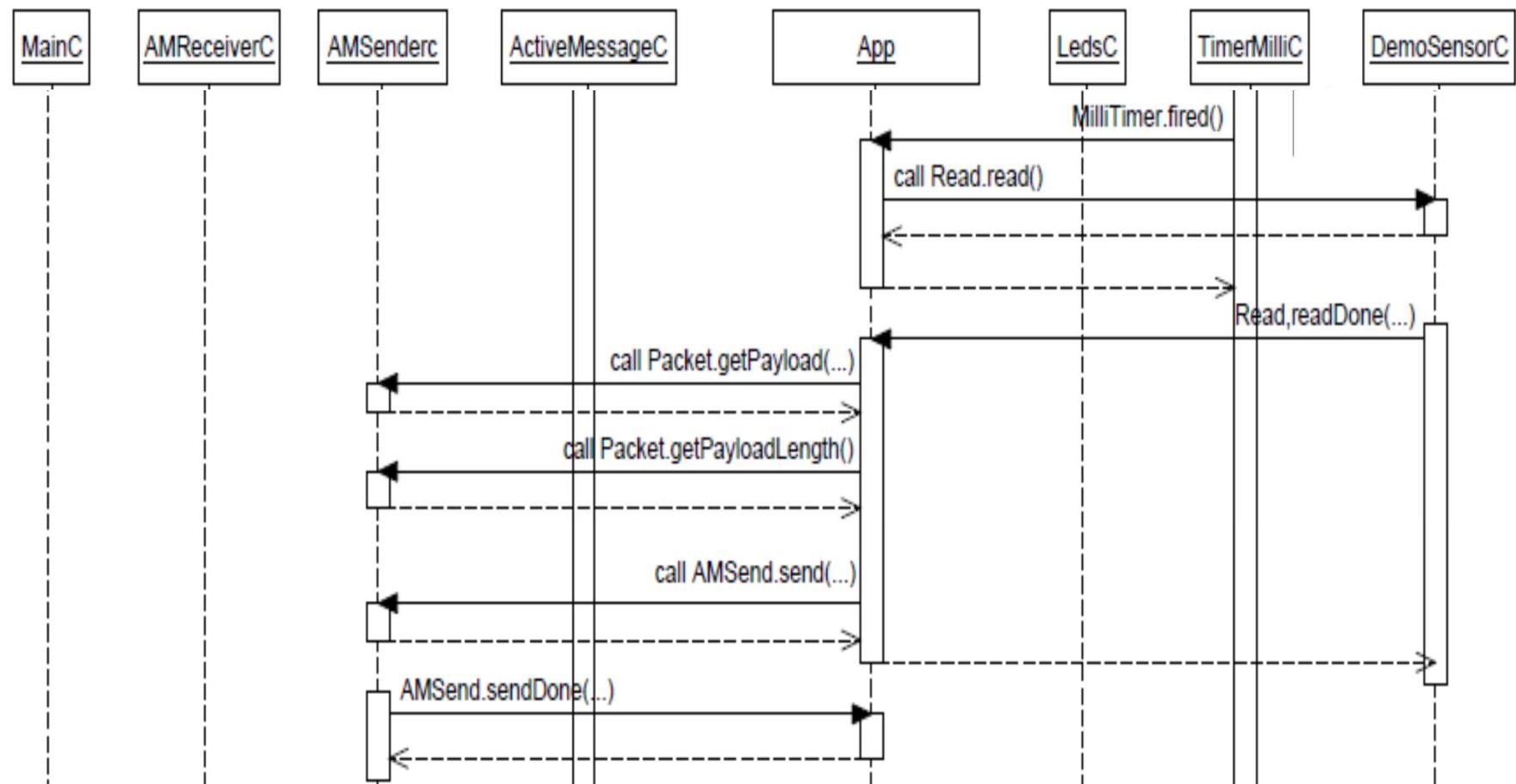
- Sequence diagram: INIT





RadioSenseToLeds

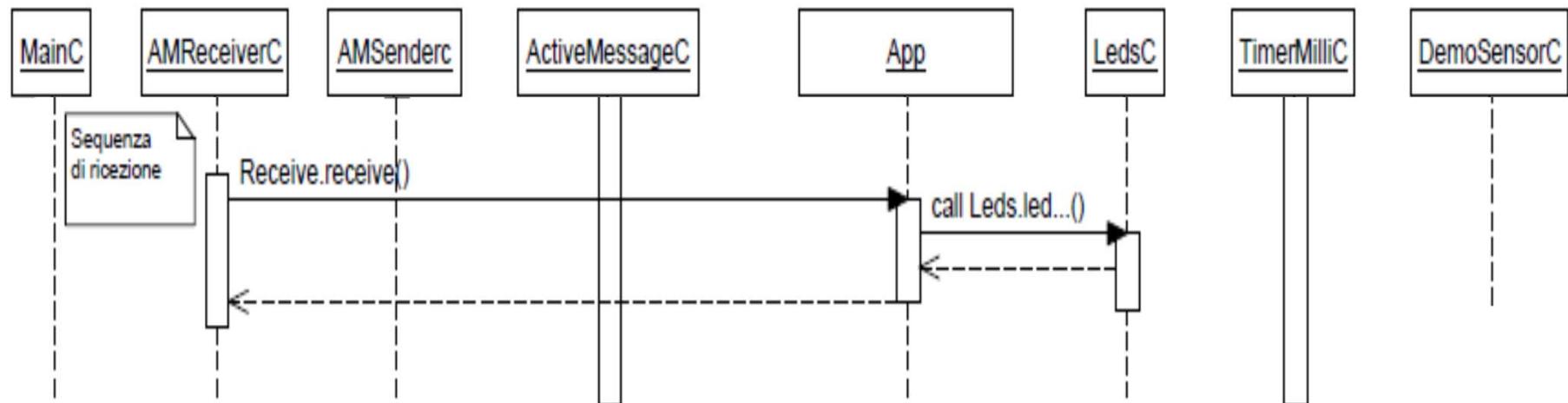
- Sequence diagram: periodic activity





RadioSenseToLeds

- Sequence diagram: message reception



Examples nesC/TinyOS

Oscilloscope (TinyOS 2.x)



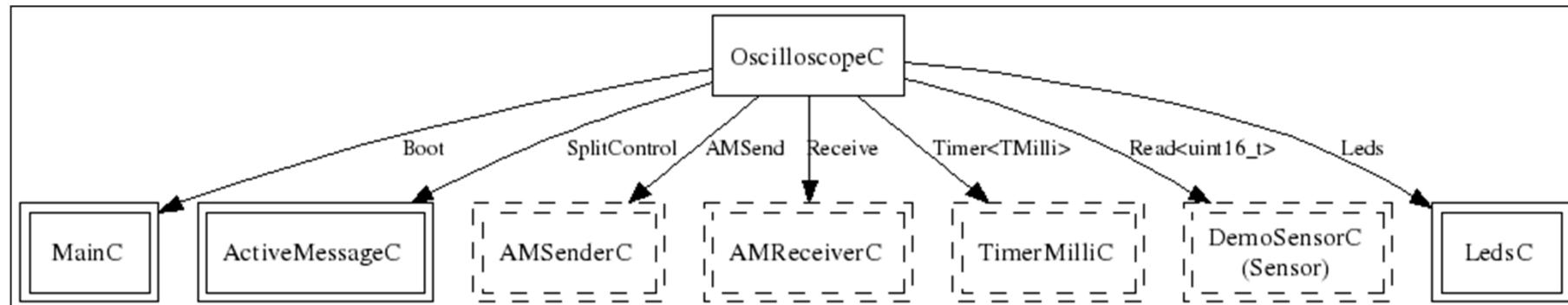
Oscilloscope

- This app periodically (4Hz) samples a sensor while sending a message each 10 read values
 - *Sink node* received data are shown by PC GUI
 - Applicative components
 - *OscilloscopeAppC (Configuration), OscilloscopeC (Module)*
 - *BaseStation*
 - System components
 - *MainC, LedsC, TimerMilliC, DemoSensorC*
 - *AMReceiverC, AMSenderC, ActiveMessageC*



Oscilloscope

- OscilloscopeAppC: component graph





Oscilloscope

- Oscilloscope.h

```
enum
{
    NREADINGS = 10,
    DEFAULT_INTERVAL = 256,
    AM_OSCILLOSCOPE = 0x93
};

typedef nx_struct oscilloscope
{
    nx_uint16_t version;          /* Version of the interval. */
    nx_uint16_t interval;         /* Sampling period. */
    nx_uint16_t id;               /* id of sending mote. */
    nx_uint16_t count;            /* The readings are samples count *
                                 * NREADINGS onwards */
    nx_uint16_t readings[NREADINGS];
} oscilloscope_t;
```



Oscilloscope

- OscilloscopeAppC.nc

```
#include "Oscilloscope.h"

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

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



Oscilloscope

- OscilloscopeC.nc

```
#include "Timer.h"
#include "Oscilloscope.h"
module OscilloscopeC{
    uses {
        interface Boot;
        interface SplitControl as RadioControl;
        interface AMSend;
        interface Receive;
        interface Timer<TMilli>;
        interface Read<uint16_t>;
        interface Leds;
    }
    implementation{
        message_t sendBuf;
        bool sendBusy;
        oscilloscope_t local;
        uint8_t reading;
        bool suppressCountChange;
    }
}
```



Oscilloscope

- OscilloscopeC.nc

```
// Utility C-style functions
void report_problem() { call Leds.led0Toggle(); }
void report_sent() { call Leds.led1Toggle(); }
void report_received() { call Leds.led2Toggle(); }
void startTimer()
{call Timer.startPeriodic(local.interval); reading = 0;}

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

event void RadioControl.startDone(error_t error) {
    startTimer();
}
event void RadioControl.stopDone(error_t error) {}
```



Oscilloscope

- OscilloscopeC.nc

```
event void Timer.fired(){
    if (reading == NREADINGS)
    {
        if (!sendBusy&&sizeof local <= call AMSend.maxPayloadLength())
        {
            memcpy(call AMSend.getPayload(&sendBuf,
                sizeof(local)), &local, sizeof local);
            if (call AMSend.send(AM_BROADCAST_ADDR, &sendBuf,
                sizeof local) == SUCCESS) sendBusy = TRUE;
        }

        if (!sendBusy) report_problem();
        reading = 0;
        if (!suppressCountChange) local.count++;
        suppressCountChange = FALSE;
    }
    if (call Read.read() != SUCCESS) report_problem();
}
```



Oscilloscope

- OscilloscopeC.nc

```
event void Read.readDone(error_t result, uint16_t data)
{
    if (result != SUCCESS)
    {
        data = 0xffff;
        report_problem();
    }
    local.readings[reading++] = data;
}
```



Oscilloscope

- OscilloscopeC.nc

```
event void AMSend.sendDone(message_t* msg, error_t error)
{ if (error == SUCCESS) report_sent();
  else report_problem();
  sendBusy = FALSE;}

event message_t*
Receive.receive(message_t* msg, void* payload, uint8_t len)
{ oscilloscope_t *omsg = payload;
  report_received();

  if (omsg->version > local.version)
    {local.version = omsg->version;local.interval = omsg->interval;
     startTimer();}
  if (omsg->count > local.count)
    {local.count = omsg->count; suppressCountChange = TRUE;}
  return msg;
}
```



Oscilloscope

- BaseStation
 - App to be installed on *sink node* (connected by wired serial to PC)
 - This app forward by radio data received by serial (i.e. by PC) and viceversa
 - Exchanged data have to follow AM packets structure