SAPIENZA
UNIVERSITÀ DI ROMA

Wireless Systems Laboratory

18 November 2013

A. Cammarano, A.Capossele, D. Spenza

SAPIENZA
UNIVERSITÀ DI ROMA

Contacts

Cammarano: cammarano@di.uniroma1.it
Capossele: capossele@di.uniroma1.it
Spenza: spenza@di.uniroma1.it

Tel: 06-49918430
Room: 333

Slides: *http://wwwusers.di.uniroma1.it/~spenza/lab2013.html*

Outline

- Oscilloscope
  http://tinyos.stanford.edu/tinyos-wiki/index.php/Sensing#The_Oscilloscope_application

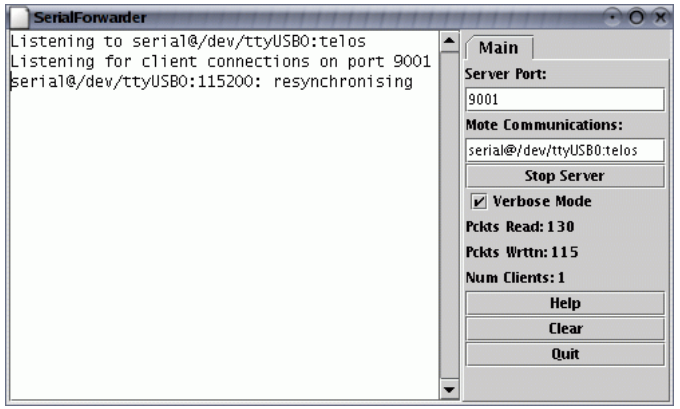
- Reading from ADC ***(Analog-to-Digital Converter)***
  http://www.tinyos.net/tinyos-2.x/doc/html/tep101.html

- Data Storage (***Flash Memory***)
  http://www.tinyos.net/tinyos-2.x/doc/html/tep103.html

- Final Projects

Last Episode

- Mote-PC serial communication
- BaseStation
- SerialForwarder



dest addr	link source addr	msg len	groupID	handlerID	source addr	counter
ff ff	00 00	04	22	06	00 02	00 0B

Homework

- Add packet acknowledgements to the modified BlinkToRadio application to fix the issue that causes blinking to freeze when a counter packet is lost.

- PacketLink: is a layer added to the CC2420 radio stack to help unicast packets get delivered successfully.
  - The first command, **setRetries(..)**, will specify the maximum number of times the message should be sent before the radio stack stops transmission.
  - The second command, **setRetryDelay(..)**, specifies the amount of delay in milliseconds between each retry.

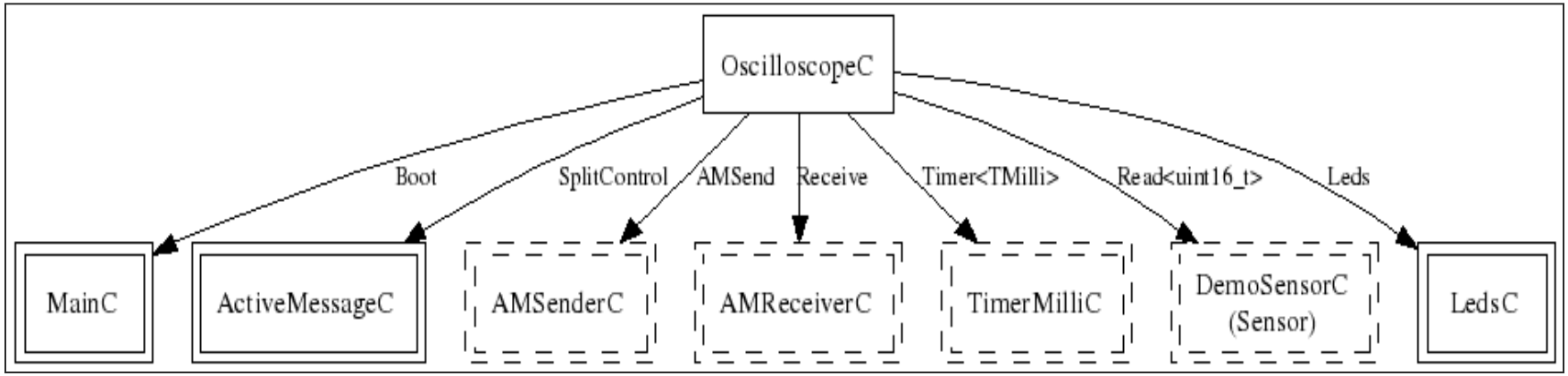
```
command void setRetries(message_t *msg, uint16_t maxRetries);
command void setRetryDelay(message_t *msg, uint16_t retryDelay);
```

Oscilloscope

- Periodically samples the default sensor via (DemoSensorC) and broadcasts a message with 10 accumulated readings over the radio.
- Application Components
  - *OscilloscopeAppC (Configuration), OscilloscopeC (Module)*
    - *BaseStation*
- System Components
  - *MainC, LedsC, TimerMilliC, DemoSensorC*
  - *AMReceiverC, AMSenderC, ActiveMessageC*

Oscilloscope

- OscilloscopeAppC: components' 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

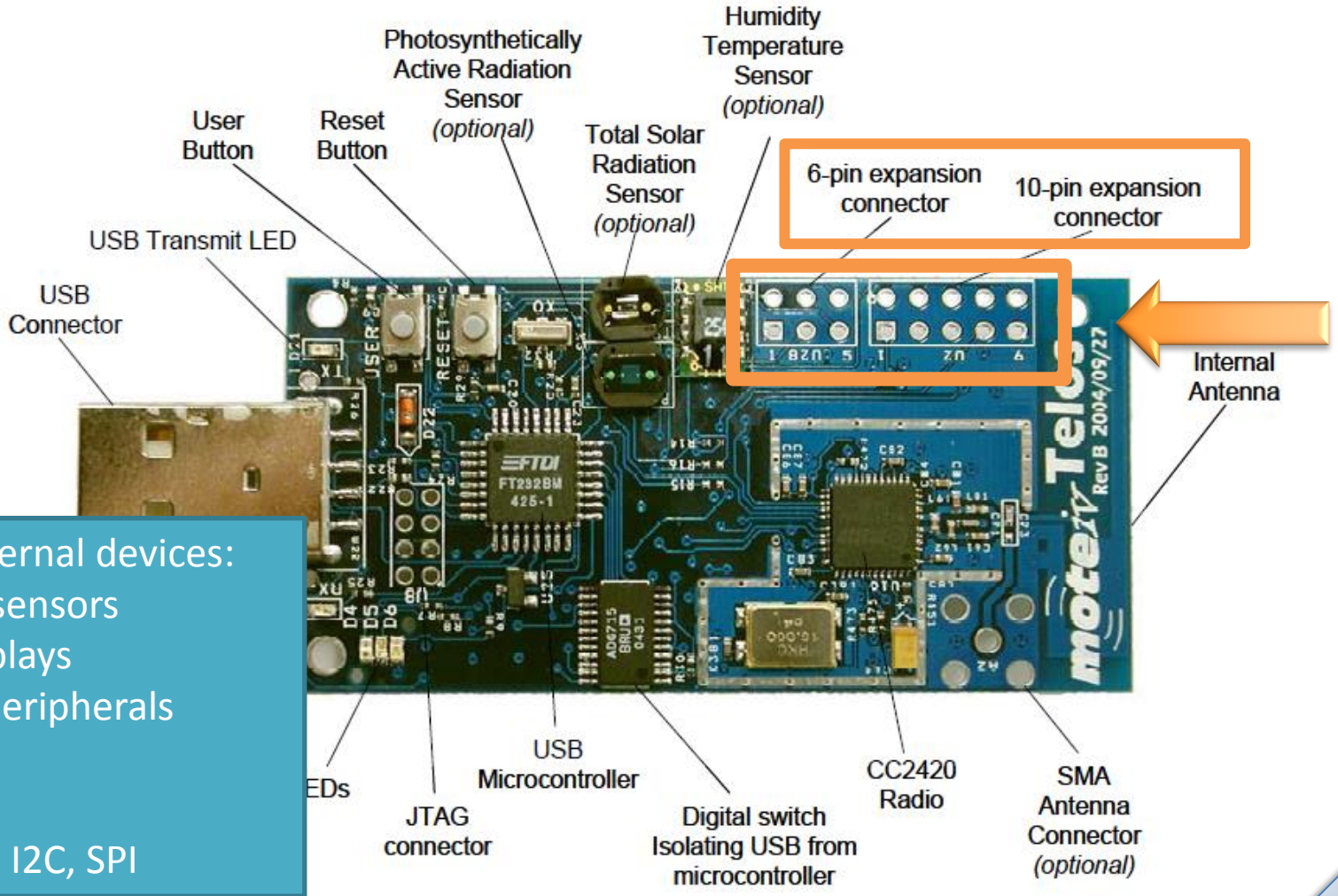
- OscilloscopeC.nc

```
 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;
  }
```

Expansion I/O Connectors

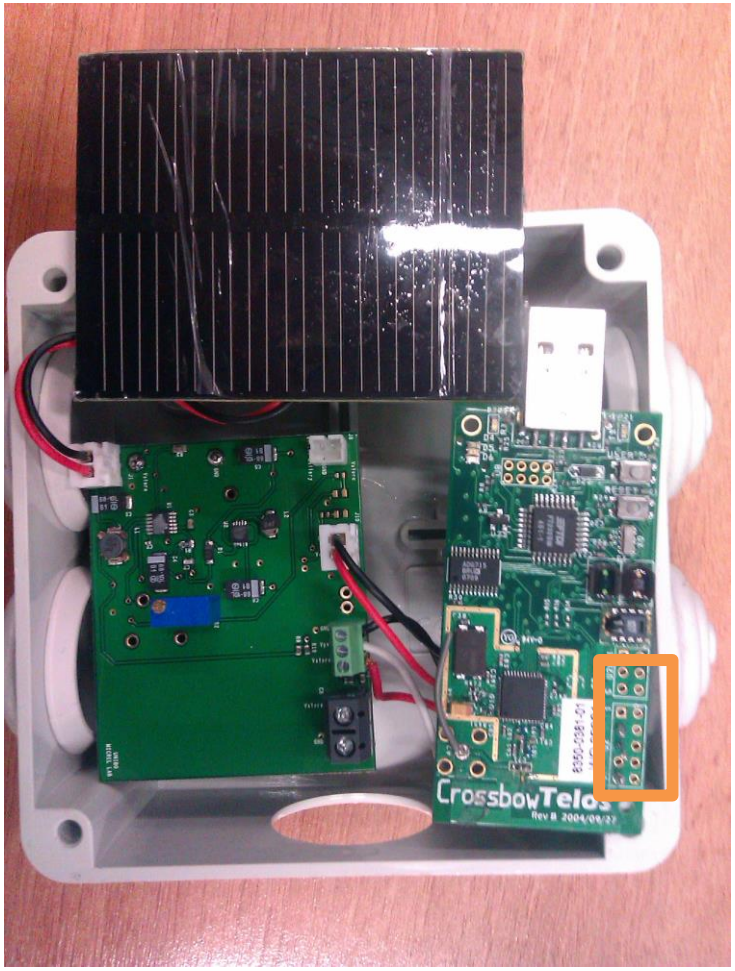


Control external devices:

- Analog sensors
- LCD displays
- Digital peripherals
- …

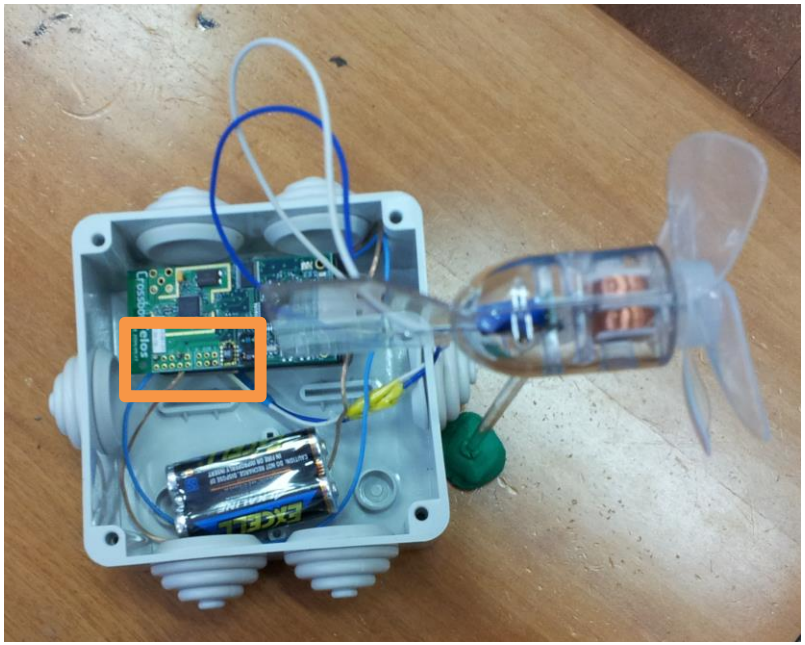
ADC, UART, I2C, SPI

Analog-to-Digital Converters (ADC)

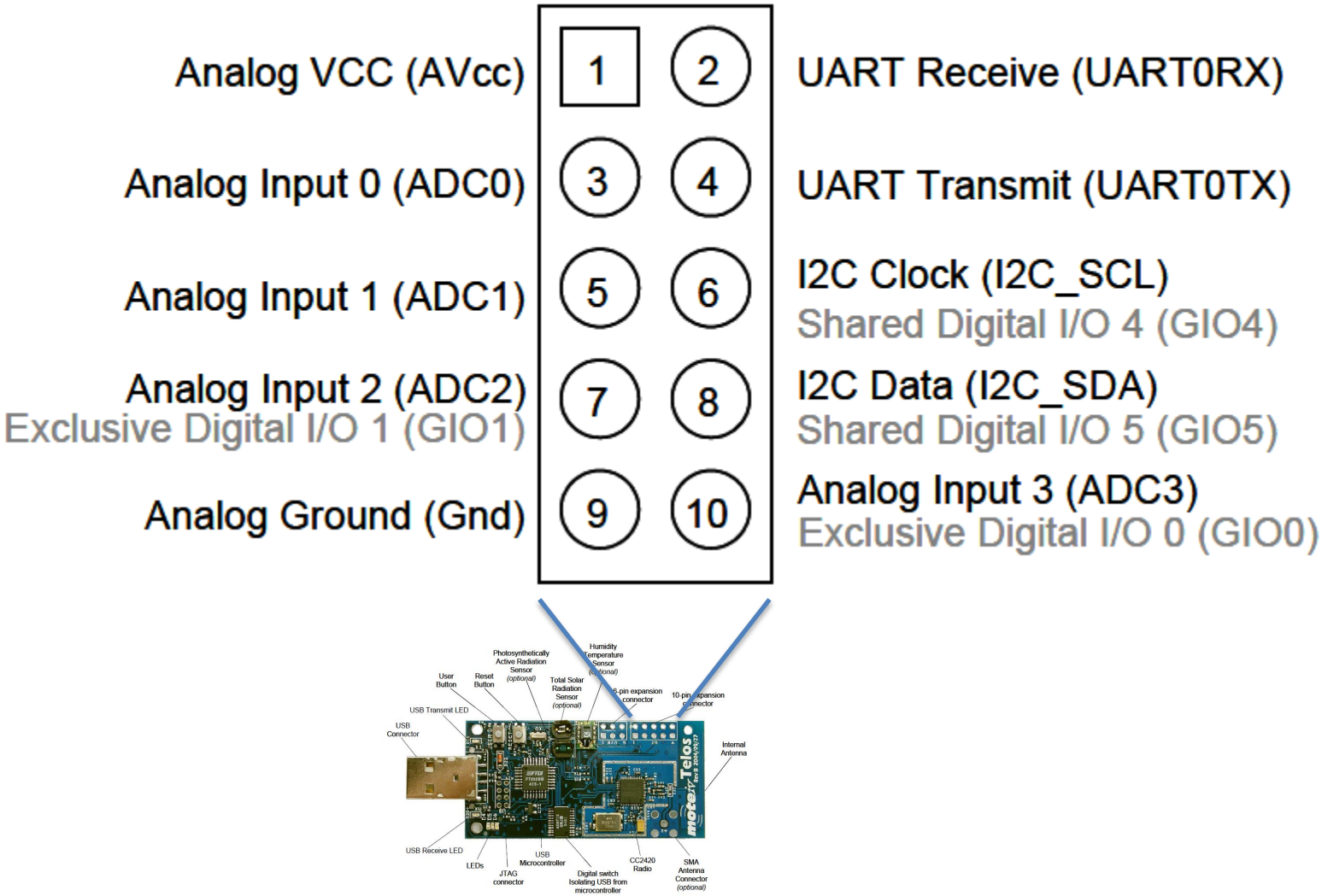


Analog-to-digital converter: converts input analog voltage to a digitial value

Example Collect data from external analog devices: solar cells, micro-turbines.



10-pin TelosB expansion connector



Msp430 ADC12

- Converts an analog input to its 12-bit digital representation

Input voltage (signal to convert)

ADC result:

- Full scale (0FFFh) if $V_{in} >= V_{R+}$
- 0 if $V_{in} <= V_{R-}$

$$N_{ADC} = 4095 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

Upper limit of conversion

Lower limit of conversion

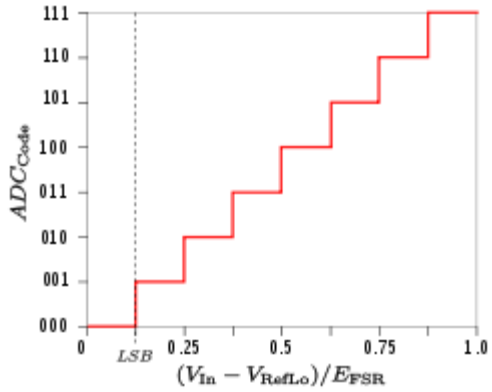
More details: Texas Instruments MSP430 F1611 Datasheet, Chapter 17

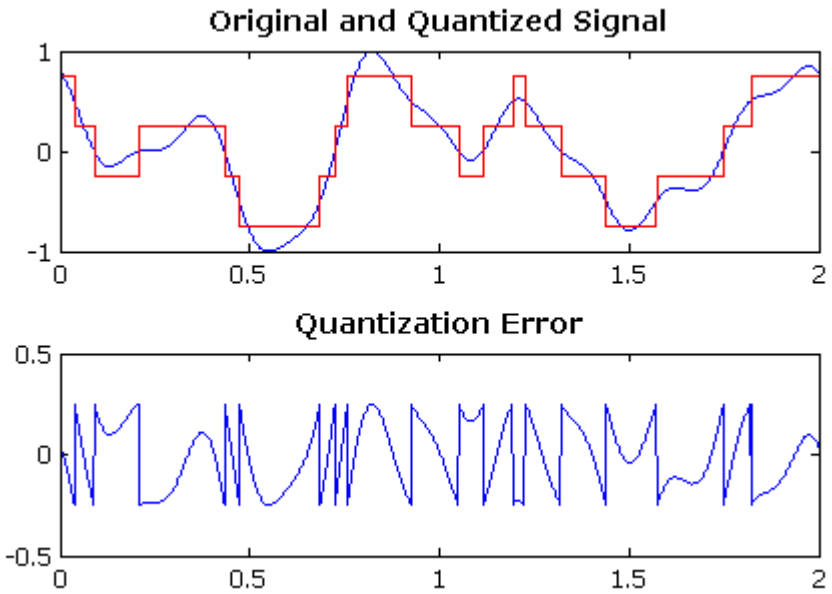
ADC input is quantized

Resolution: number of discrete values that can be produced over the range of analog values

Quantization error: rounding error between the analog input voltage to the ADC and the output digitized value.

ADC12: 2^{12} = 4096 different levels





Oscilloscope: read from ADC0

Modify the Oscilloscope app to read from ADC0

```
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;
}
```

Change to ADC0SensorC

ADC0SensorC.nc

```
generic configuration ADC0SensorC() {
	provides interface Read<uint16_t>;
}

implementation {
	components new AdcReadClientC();
	Read = AdcReadClientC;

	components ADC0SensorP;
	AdcReadClientC.AdcConfigure -> ADC0SensorP;
}
```

ADC0SensorP.nc

```
module ADC0SensorP {
        provides interface AdcConfigure<const msp430adc12_channel_config_t*>;
}
implementation {
        msp430adc12_channel_config_t config = {
                inch: INPUT_CHANNEL_A0,
                sref: REFERENCE_VREFplus_AVss,
                ref2_5v: REFVOLT_LEVEL_2_5,
                adc12ssel: SHT_SOURCE_ACLK,
                adc12div: SHT_CLOCK_DIV_1,
                sht: SAMPLE_HOLD_4_CYCLES,
                sampcon_ssel: SAMPCON_SOURCE_SMCLK,
                sampcon_id: SAMPCON_CLOCK_DIV_1
        };

        async command const msp430adc12_channel_config_t* AdcConfigure.getConfiguration() {
                return &config;
        }
}
```

chips/msp430/adc12/Msp430Adc12.h

Data Storage (Flash memory)

Debugging - Printf

TinyOS provides a printing functionality (`printf` library) to help developers during first debugging phase.

1. C syntax
2. Include PrintfC component
3. Include ‘’printf.h’’
4. Supported on msp430 and atmega128x based platforms
5. Library directory: tos/lib/printf/2_0_2

Example: tinyos-2.x/apps/tutorials/Printf

To read printf output start the PrintfClient application:
java net.tinyos.tools.PrintfClient -comm serial@/dev/ttyUSBXXX:telosb

Data Storage

TinyOS provides:

- 3 basic storage abstractions: **small objects, circular logs, and large objects**
- *interfaces* to abstract the underlying storage services
- *components* that *provide* (implement) these interfaces

Interfaces and types:

- *BlockRead – BlockWrite*
- *Mount*
- *ConfigStorage*
- *LogRead – LogWrite*
- *Storage.h*

Components:

- ConfigeStorageC
- LogStorageC
- BlockStorageC

chip-specific implementations for stm25p

Large Objects

- A large object ranges from a few kilobytes upwards (e.g., 100b to 100Kb)
- **Random** Read and Write
- A large object is erased before the first write
- A sync ensures that a large object survives a reboot or crash
- Reads are unrestricted
- Each byte can only be written once between two erases
- Fault tolerance ⟶ Validity Check (CRC)

Applications:

I. Network Reprogramming
II. Large data-packets in a data-transmission system

Large Objects - Interface

```
generic configuration BlockStorageC (volume_id_t volid) {
    provides {
        interface BlockWrite;
        interface BlockRead;
    }
}
```

- **BlockWrite.erase**: erase the volume. After a reboot or a commit, a volume MUST be erased before it can be written to.
- **BlockWrite.write**: write some bytes starting at a given offset. Each byte MUST NOT be written more than once between two erases.
- **BlockWrite.sync**: ensure all previous writes are present on a given volume. Sync MUST be called to ensure written data survives a reboot or crash.
- **BlockRead.read**: read some bytes starting at a given offset.
- **BlockRead.computeCrc**: compute the CRC of some bytes starting at a given offset.
- **BlockRead.getSize (*)**: return bytes available for large object storage in volume.

Small Objects

Sensor network applications need to store configuration data:

- Assumes that configuration data is relatively small (< 100B)
- Allows **random** reads and writes
- Has simple transactional behaviour: each read is a separate transaction, all writes up to a commit form a single transaction
- At reboot, the volume contains the data as of the most recent successful commit
- Fault tolerance: **atomic** write ⟶ failure between/during writes does not compromise data

Small Objects - Interface

```
generic configuration ConfigStorageC(volume_id_t volid) {
    provides {
        interface Mount;
        interface ConfigStorage;
    }
}
```

- **Mount.mount**: mount the volume.
- **ConfigStorage.valid**: return TRUE if the volume contains a valid small object.
- **ConfigStorage.read**: read some bytes starting at a given offset. Fails if the small object is not valid. Note that this reads the data as of the last successful commit.
- **ConfigStorage.write**: write some bytes to a given offset.
- **ConfigStorage.commit**: make the small object contents reflect all the writes since the last commit.
- **ConfigStorage.getSize**: return the number of bytes that can be stored in the small object.

Circular Logs

TinyOS provides a functionality to realiably (atomic) logging events and small data items:

- Linear or Circular
- Sequential Read and Write
- Fault tolerance: user-specified commit points

```
generic configuration LogStorageC(volume_id_t volid, bool circular)
{
    provides {
        interface LogWrite;
        interface LogRead;
    }
}
```

Circular Logs - Interface

- **LogWrite.erase**: erase the log
- **LogWrite.append**: append some bytes to the log. In a circular log, this may overwrite the current read position
- **LogWrite.sync**: guarantee that data written so far will not be lost to a crash or reboot. (*may require some memory*)
- **LogWrite.currentOffset**: return cookie representing current append position (for use with LogRead.seek)
- **LogRead.read**: read some bytes from the current read position in the log and advance the read position.
- **LogRead.currentOffset**: return cookie representing current read position
- **LogRead.seek**: set the read position to a value returned by a prior call to LogWrite.currentOffset or LogRead.currentOffset, or to the special SEEK_BEGINNING value
- **LogRead.getSize**: return an approximation of the log's capacity in bytes. Uses of sync and other overhead may reduce this number

Volume

Flash chips are divided into separate volumes providing a single storage abstraction.

- Different numerical identifiers for each volume
- fdisk like allocation
  - init volume table
  - allocate volumes
  - commit volume table
- volume must be mounted before use
- no unmount operation

```
<volume_table>
 <volume name="DELUGE0" size="65536" />
 <volume name="CONFIGLOG" size="65536" />
 <volume name="DATALOG" size="131072" />
 <volume name="GOLDENIMAGE" size="65536" base="983040" />
</volume_table>
```

Volume

- name and size parameters are required
- base is optional
- name is a string in [a-zA-Z0-9_]
- size in bytes
- volumes is specified by an XML file that is placed in the application's directory volumes-CHIPNAME.xml

The storage abstractions are accessed by instantiating generic components that take the volume macro as argument:
components new BlockStorageC(VOLUME_DELUGE0);

Log Application

apps/tests/storage/Log

Log abstraction test in linear mode.

Program the node id with a value ID = T * 100 + k, where **k** is a random seed and **T** specifies the type of test to perform.

T = 0 perform a full test
T = 1 erase the log
T = 2 read the log
T = 3 write some data to the log

LED 1 on -> test succeded
LED 0 on -> test failed

Final projects

Coding standards

http://www.tinyos.net/tinyos-2.x/doc/html/tep3.html

The default packages in a ut which are not viewed as essential to its operation. TinyOS distribution are:

- ***tos/system/*** Core TinyOS components.
- ***tos/interfaces/*** Core TinyOS interfaces, including hardware-independent abstractions.
- ***tos/platforms/*** Contains code specific to mote platforms, but chip-independent.
- ***tos/chips/*** Contains code specific to particular chips and to chips on particular platforms.
- ***tos/libs/*** Contains interfaces and components which extend the usefulness of TinyOS b
- ***apps/, apps/demos, apps/tests, apps/tutorials*** Contain applications with some division by purpose. Applications may contain subdirectories.

Final projects

- Adaptive sampling/compressive sensing
- Energy predictions
- Energy-harvesting-aware routing
- Energy-harvesting-aware scheduling
- Security (DTLS)
- Wind energy-harvesting systems