Bezvadu Sensoru Tīklu operētājsistēma TinyOS

Uldis Bojārs
Datorikas fakultāte
Latvijas Universitāte
12.sep.2011.

Iesākumā

- Jautājumi
- Kas jauns?
- Praktiskie darbi (= “mājas darbi”)
  - šodien sākam 1. PD – “Morzes kods”
  - sīkāka info lekcijas beigās

Jautājumi

- Jautājumi, kas radās iepriekšējā reizē:
  - Kursa projekts Android vidē?
  - Linux: nav iepriekš strādāts ar to?
  - ...
- Papildus “bonusi” par derīgu info, kas noder visiem
  - skat. praktiskā darbā “bonusu” par TOSSIM

Komunikācija

- Wiki
  - http://selavo.lv/wiki/index.php/LU-BST-b12
- Google Groups
  - links BST wiki lapas sākumā
  - ja vēl neesat grupā, rakstiet un piesakieties!
- Vieta sarunām un derīgam info:
  - collab. editing: http://tinyurl.com/bst-b12-notes

Saturs

- Ievads
- TinyOS ideoloģija: komponenti un interfeisi
- Hello World
- Uzdevumi (tasks)
- Radio komunikācija (Mote-to-Mote)
- Seriālā porta komunikācija (Mote-to-PC)
- Sensoru lasīšana
- Vingrinājumi

Kas ir operētājsistēma?

Kas ir operētājsistēma?

- Programma, kas darbojas kā lietotāja un aparatūras starpnieks
- Uzdevumi:
  - Izpildīt lietotāja programmas
  - Piedāvāt ērtu un efektīvu piekļuvi resursiem
- Sensoru tīklos lietotājs ir "pētnieks" → jāpiedāvā ērta sensoru mezglu programmēšana

Kas vēl varētu būt sensoru tīklu lietotāji?

Kas ir TinyOS?

- Paredzēta sensoru mezgliem
- Efektīvs resursu izmantojums
- Notikumu bāzētas aplikācijas

- Tipiskajiem uzdevumiem (taimeri, radio un seriālā komunikācija, sensoru (ADC) lasīšana) tiek piedāvāti gatavi risinājumi

TinyOS: OS vai bibliotēkas?

- Lietotāja izejas kods tiek savienots ar TinyOS kodu, veidojot vienotu aplikāciju

- Vai TinyOS ir operētājsistēma?

Lūdzu, neaizmiedziet!

- Būs daudz tehniskas informācijas

http://www.vajazzling.com/files/sorry.jpg

skat. otras prezentācijas slaidus:

TinyOSTutorial-10-31-2011.pdf

TinyOS uzbūve, nesC valoda

Komponentu sajūgšana

Split-phase pieeja operācijām

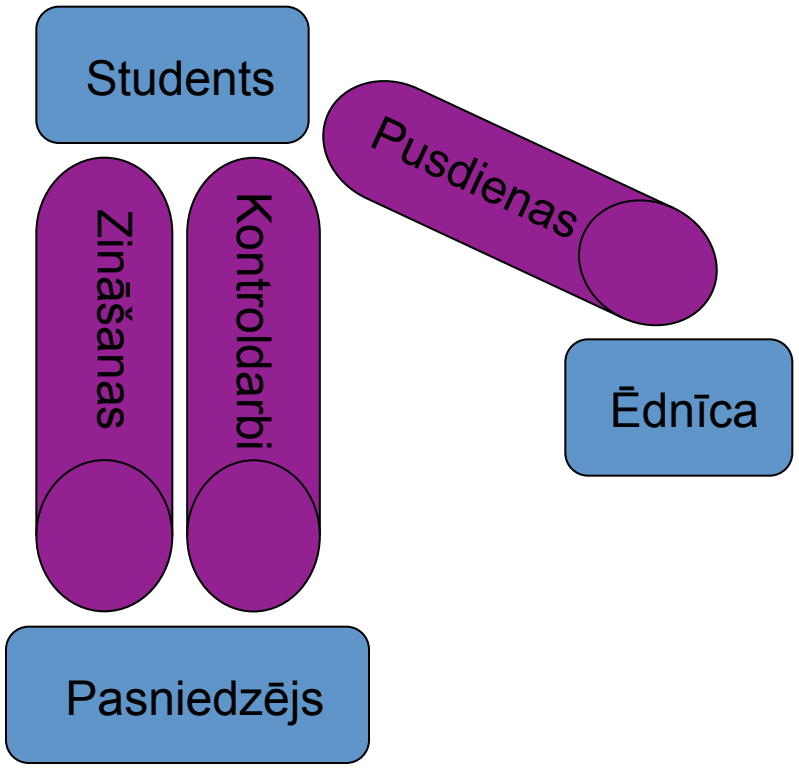
TinyOS ideoloģija = komponenti

- Aplikācijas sastāv no komponentiem
- Komponents = izmantotie interfeisi
  + piedāvātie interfeisi
- Interfeisi = komponentu sadarbības valoda
- Divu veidu komponenti:
  - Moduļi = interfeisu implementācija
  - Konfigurācija = komponentu saistības

Komponentu piemērs

```
module Students {
  uses interface Pusdienas;
  uses interface Zināšanas;
  provides interface Kontroldarbi;
}
module Ēdnīca {
  provides interface Pusdienas;
}
module Pasniedzējs {
  uses interface Kontroldarbi;
  provides interface Zināšanas;
}

configuration Universitāte {}
implementation {
  components Students, Pasniedzējs, Ēdnīca;
  Students.Zināšanas -> Pasniedzējs;
  Students.Kontroldarbi -> Pasniedzējs;
  Students.Pusdienas -> Ēdnīca;
}
```

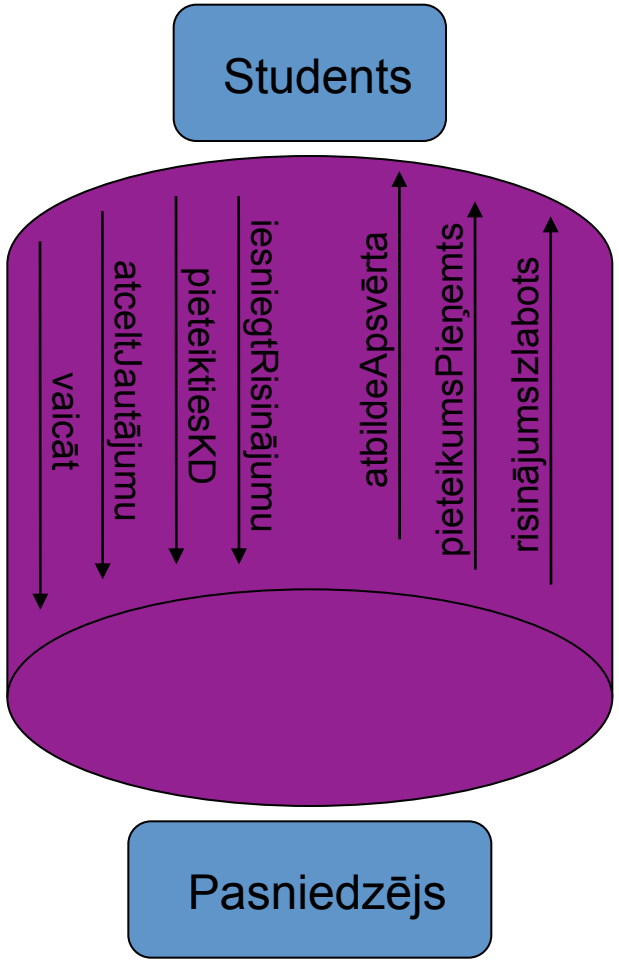


Interfeisi

- Interfeisi apraksta abpusēju komunikāciju:
  - **Komandas**, kas **jāimplementē piedāvātājam** un kuras var izsaukt izmantotājs
  - **Notikumi**, kurus var izsaukt piedāvātājs un kurus **jāimplementē izmantotājam**

Interfeisu piemērs

```
interface Zināšanas {
  // atgriež true, ja vaicātājam ir cerības uz atbildes saņemšanu
  // kā parametrs tiek nodots jautājuma teksts
  command bool vaicāt(const char *jautājums);
  // jautājums zaudējis aktualitāti un tiek atcelts
  command void atceltJautājumu(const char *jautājums);
  // kā parametru nodod atbildes tekstu vai 0, ja atbilde nav atrasta
  event void atbildeApsvērta(const char *atbilde);
}
interface Pusdienas {
  // pieprasot ēdienu nekas nav zināms par saņemšanas iespējām,
  // arī atcelt nav iespējams
  command void prasītĒdienu();
  // nodod pointeri uz porciju vai 0, ja ēdiena nav
  event void pusdienasGatavas(void *porcija);
  // uz izdrukāto čeku arī obligāti jāreaģē
  event void čeksIzdrukāts(uint8_t summa);
}
interface Kontroldarbi {
  command void pieteiktiesKD(uint8_t kdNr);
  // ja atļauts piedalīties KD, nodod pointeri uz uzdevumiem,
  // pretējā gadījumā 0
  event void pieteikumsPieņemts(void *uzdevumi);
  command void iesniegtRisinājumu(uint8_t kdNr, char *risinajums);
  // atzīme 10 baļļu sistēmā. 11: neesi pieteicies; 12: nošpikots
  event void risinājumsIzlabots(uint8_t kdNr, uint8_t atzime);
}
```



Hello, World!

- Print the words “`hello, world`”

— Kernighan and Ritchie,

The C Programming Language (2nd edition)

- Kā izdrukāt “Hello, world” uz motes?

The World is Blinking

// Konfigurācija

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

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

// Modulis

```
#include "Timer.h"

module BlinkC {
  uses interface Timer<TMilli> as Timer0;
  uses interface Leds;
  uses interface Boot;
}
implementation {
  event void Boot.booted() {
    call Timer0.startPeriodic(250);
  }
  event void Timer0.fired() {
    call Leds.led0Toggle();
  }
}
```

// Makefile

```
COMPONENT=BlinkAppC
include $(MAKERULES)
```

Programmas kompilācija

- **`motelist`** – izvada sarakstu ar datoram pieslēgtajām motēm
- **`make telosb`** – uzbūvē programmu telosb (TMote Sky) platformai
- **`make telosb reinstall`** – uzkopē programmu uz motes
- **`make telosb install`** – gan uzbūvē, gan uzkopē
- **`make clean`** – iztīra visus uzbūvētos failus, reizēm palīdz
- **`make telosb install,3 bsl,/dev/ttyUSB2`** – uzbūvē un uzinstalē programmu uz motes (kurai piešķir ID=3), kas pieslēgta pie /dev/ttyUSB2
- **`make telosb docs`** – uzbūvē programmu un dokumentāciju (pieejama tinyosPath/doc/nesdoc/telosb/) par visiem izmantotajiem interfeisiem un komponentiem

Nosaukumu piešķiršanas likumi

- Faila vārdam jāsakrīt ar tajā aprakstīto komponentu vai interfeisu
- Publiskajiem komponentiem nosaukumā sufikss “C”, privātajiem: “P”
- *CamelCase* nosaukumi: KatrsVardsArLieloBurtu
- Failiem paplašinājums “.nc”
- Pilnais likumu saraksts atrodams TEP 3 (tinyosPath/doc/html/tep3.html)

Uzdevumi (tasks)

- Uzdevumi paredzēti lielu (ilgu) aprēķinu veikšanai
- Ieteicams garus uzdevumus dalīt mazākos
- Uzdevumi darbojas viena komponenta ietvaros
- Uzdevumi tiek likti FIFO rindā, kur nākošais sāk izpildi, kad iepriekšējais pabeidzis
- Notikumi var pārtraukt uzdevumus (un arī citus notikumus), bet ne otrādi
- Viens uzdevums nevar atrasties rindā vairākos eksemplāros vienlaikus (aktīvais uzdevums var sevi ielikt rindā)

Uzdevumi, notikumi un komandas

- Pazīmējam uz tāfeles:
  - Sākas uzdevums T1
  - Tas izsauc post(T2)
  - Ierodas notikums E1
  - Tam pa vidu notikums E2
  - Utt, improvizējam

Deklarāciju un izsaukumu sintakse

- Uzdevums:
  - `task uint8_t uzdevums();`
  - `uint8_t res = post uzdevums();`
- Komanda:
  - `command void startSending();`
  - `call startSending();`
- Notikums:
  - `event void sendDone(uint8_t res);`
  - `signal sendDone(RES_OK);`

Radio komunikācija

- Tiek izmantota abstrakta ziņojuma (paketes) struktūra **message_t**
- TinyOS definēti interfeisi, kas iegūst informāciju no message_t un prot to apstrādāt
- Katra platforma definē komponentus, kas piedāvā attiecīgos paketes apstrādes interfeisus

message_t datu struktūra

- Pa tiešo to labot nedrīkst, jāizmanto funkcijas
- Satur platformas specifisku *header* un *footer*
- Satur *payload*: derīgo datu apgabals, maksimāli **28 baitus** garš. Arī šim apgabalam izmantot struktūras, neaiztikt baitus pa tiešo! Kāpēc?

Atkāpe

Atšķirīgi “indiāņi” = problēmas

- 16-bitu mainīgos “pa tiešo” atmiņā pa baitam neaiztikt!

- Pārsvarā visi ir mazie indiāņi, tai skaitā, MSP430, ATMega. Arī x86.
- PowerPC ir lielais indiānis

Bitu operācijas valodā C

- Dots: 16bitu skaitlis (0x1234h)
- Kā no tā iegūt:
  - Svarīgākos 8 bitus (0x12h)?
  - Mazākos 8 bitus (0x34h)?
- Kā apgriezt baitus vietām (0x3412h)?
- Kā n-to bitu baitā:
  - Nolasīt
  - Ierakstīt tajā 1/0

Atpakaļ pie tēmas

message_t apstrādes interfeisi

- tos/interfaces:
- **Packet**: informācija par paketi, pointeris uz payload, paketes satura iztīrīšana
- **Send**, **Receive**: pakešu sūtīšanas un saņemšanas interfeisi, ērtuma labad māk arī atrast payload pointeri un izmēru
- **PacketAcknowledgements**: iespēja ieslēgt/izslēgt paziņojumus par paketes nosūtīšanu
- **RadioTimeStamping**: informācija par paketes nosūtīšanas un saņemšanas laiku

Otrā līmeņa pakešu abstrakcija

- Virs iepriekš minētajiem interfeisiem atrodas *AM-paketes* (*Active Message*) interfeisi: **AMPacket** un **AMSend**
- AM paketes satur arī saņēmēja adresi (2B) un servisa tipu (1B)
- Pēc servisa tipa nosaka, kam nodot saņemto paketi apstrādei – radio raidītāja dalīšana starp vairākiem servisiem
- Motes AM adresi var uzstādīt instalācijas un arī izpildes laikā

AM paketē nosūtītie dati

- AM identifikators: 1B (= 00, vienmēr);
- Saņēmēja adrese: 2B
- Sūtītāja adrese: 2B (= 00 seriālam portam)
- Datu garums: 1B
- Grupas ID: 1B
- Servisa tips: 1B
- **Dati (Payload):** <= **28B**

Kopā: 28B dati + 8B meta = **36B**

AM komponenti

- **AMReceiverC**: Packet, AMPacket un Receive
- **AMSenderC**: Packet, AMPacket, AMSend un PackageAcknowledgements

- Jautājums: kā notiek AM komponentu sasiešana ar platformas realizāciju?

- Jautājums: kā notiek AM interfeisu(Packet, Send,...) sasiešana ar platformas specifisku realizāciju?

- Ar *wrapper* komponentu **ActiveMessageC**, kas katrai platformai savs

```
// tos/platforms/telosa/ActiveMessageC.nc

configuration ActiveMessageC { provides {
    interface AMSend[am_id_t id];
    interface Receive[am_id_t id];
    interface Packet; // …
}  } implementation {
  components CC2420ActiveMessageC as AM;
  AMSend        = AM;
  Receive       = AM.Receive;
  Packet        = AM;
  // …
}
```

Radio komunikācijas piemērs

```
// RadioCountToLeds.h

// Definējam payload struktūru
typedef nx_struct radio_count_msg {
    // nx_* tipi tiek galā ar indiāņiem
    nx_uint16_t counter;
} radio_count_msg_t;

// AM servisa tipa konstante
enum {
    AM_RADIO_COUNT_MSG = 6
};
```

Radio komunikācijas piemērs

```
// RadioCountToLedsAppC.nc: konfigurācija
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;
}
```

Radio komunikācijas piemērs

```
// RadioCountToLedsC.nc
module RadioCountToLedsC {
  uses {
    interface Leds; interface Boot;
    interface Receive; interface AMSend;
    interface Timer<TMilli> as MilliTimer;
    interface SplitControl as AMControl;
    interface Packet;
  }
}
implementation {
  message_t packet; // buferis sūtāmajai paketei
  bool locked;
  uint16_t counter = 0;

    // ...
```

Radio komunikācijas piemērs

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

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

event void AMControl.stopDone(error_t err) {
}

// ...
```

Radio komunikācijas piemērs

```
event void MilliTimer.fired() {
    counter++;
    if (locked) return;
    // atrodam payload
    radio_count_msg_t* rcm = (radio_count_msg_t*) call Packet.getPayload(
       &packet, sizeof(radio_count_msg_t));
    if (rcm == NULL) return;
    // ierakstām datus
    rcm->counter = counter;
    // nosūtām
    if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
                         sizeof(radio_count_msg_t)) == SUCCESS) {
      locked = TRUE;
    }
}

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

Radio komunikācijas piemērs

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

    if (rcm->counter & 0x1) {
      call Leds.led0On();
    } else {
      call Leds.led0Off();
    }
    // analoģiski pārējiem lediem

    return bufPtr;
}

} // Komponenta beigas
```

Atkāpe: “C pointer casting”

```
void *payload;
radio_count_msg_t *rcm;
rcm = (radio_count_msg_t *) payload;
```

- Kāpēc tā vajag?

Radio komunikācijas piemērs

```
// Makefile
COMPONENT=RadioCountToLedsAppC
BUILD_EXTRA_DEPS = RadioCountMsg.class

// uzbūvē Java klasi payload datu apstrādei
RadioCountMsg.class: RadioCountMsg.java
   javac RadioCountMsg.java

RadioCountMsg.java: RadioCountToLeds.h
   mig java -target=$(PLATFORM) $(CFLAGS)
        -java-classname=RadioCountMsg RadioCountToLeds.h
        radio_count_msg -o $@

include $(MAKERULES)
```

Radio komunikācijas dizains

1. Identificēt vajadzīgos interfeisus un komponentus
2. Pievienot tos moduļa *uses* sarakstam
3. Definēt stāvokļa mainīgos un start/stop funkcijas
4. Izsaukt vajadzīgajās vietās interfeisu komandas
5. Apstrādāt visus interfeisu notikumus
6. Pievienot izmantotos komponentus konfigurācijā
7. Savienot konfigurācijā interfeisus ar komponentiem

Seriālā porta komunikācija

- Tiek izmantoti tie paši interfeisi, tikai tos implementē citi komponenti:

Radio	**Seriālais ports**
ActiveMessageC	SerialActiveMessageC
AMSenderC	SerialAMSenderC
AMReceiverC	SerialAMReceiverC

Tālākie slaidi citai lekcijai

- Par sensoriem 2. lekcijā vēl nerunājām, bet te būs slaidi par šo tēmu
  - gadījumā, ja praktiskajos darbos jau noder.

Sensoru datu lasīšana

- Pēc TinyOS ideoloģijas jāpiedāvā gan augsta līmeņa (HIL), vispārīga piekļuve resursiem, gan arī aparatūras specifiskas iespējas zemā līmenī (HAL)
- Šis princips attiecas arī uz sensoriem

Sensoru savienojuma veidi

Analogie sensori:

1. Pievienoti ADC (Analog-to-Digital-Converter)

Digitālie sensori:

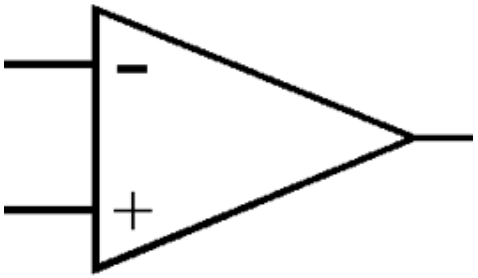
1. Pievienoti mikrokontroliera GPIO (General-Purpose I/O) kājām
2. Pievienoti pie maģistrāles ar specializētiem protokoliem, piemēram, I2C, SPI

Analog-to-Digital Converter (ADC)

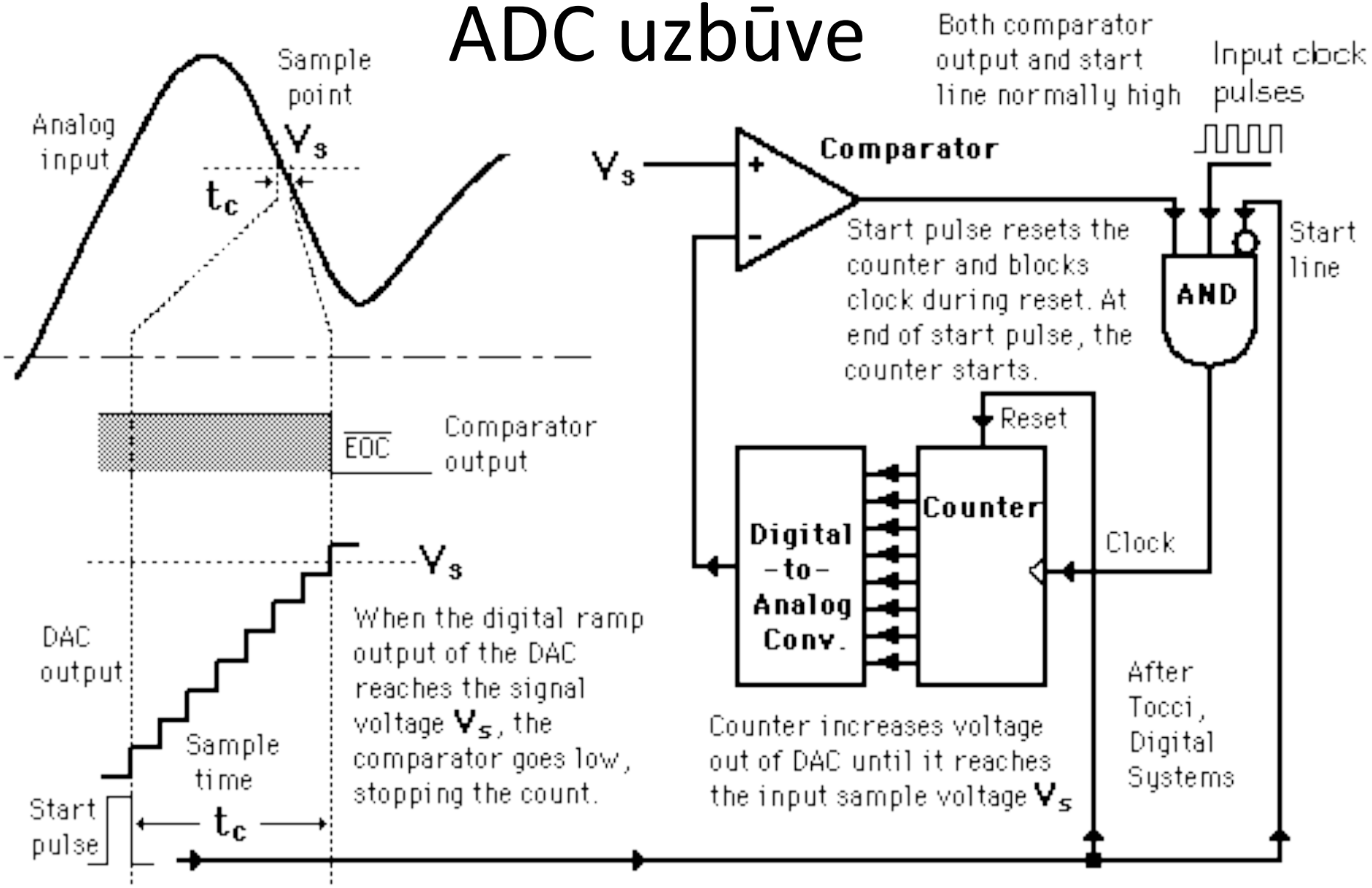
- Pārvērš analogu spriegumu skaitlī
- Izšķirtspēja tipiski 10-12 biti
- Kanālu skaits tipiski 6-8
- Daži kanāli pieslēgti iekšējiem sensoriem: temperatūra, spriegums
- ADC nolasīšana prasa zināmu laiku

ADC uzbūve

- Skaitītājs: 0, 1, ...
- DAC: ģenerē analogu spriegumu
- Komparators: salīdzina 2 signālus

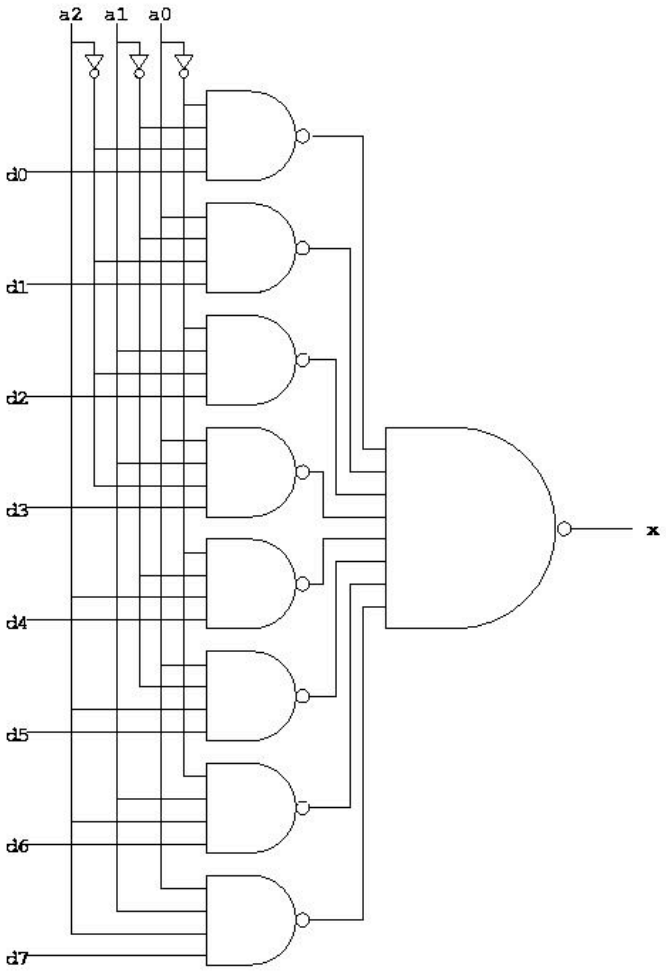


ADC uzbūve



http://www.8051projects.net/adc-interfacing/adc.gif

ADC kanāli



ADC lasīšana

- Komponentam ADC lasīšanai HIL līmenī jāpiedāvā Read un/vai ReadStream interfeisus un jāpieprasa AdcConfigure interfeisu konfigurācijai
- ADC konfigurācija platform-specifiska
- Ja nepieciešamas dažādas konfigurācijas, komponentam jāpiedāvā vairākas interfeisa instances

```
interface AdcConfigure<config_type> {
  async command config_type getConfiguration();
}
```

Augsta līmeņa sensoru lasīšana

```
interface Read<val_t> {
  command error_t read();
  event void readDone(error_t result, val_t val );
}
// ReadNow – Read asinhrons analogs

interface ReadStream<val_t> {
  command error_t postBuffer(val_t* buf, uint16_t count);
  command error_t read(uint32_t usPeriod);
  event void bufferDone(error_t result, val_t* buf, uint16_t count);
  // ja padoti vairāki buferi, readDone, ka visi pilni
  event void readDone(error_t result, uint32_t usActualPeriod);
}

interface Notify<val_t> {
  command error_t enable();
  command error_t disable();
  event void notify( val_t val );
}
```

ADC HIL līmeņa komponenti

- Katrai platformai jāimplementē šādi vispārīgi komponenti ADC lasīšanai:
  - AdcReadClientC
  - AdcReadNowClientC
  - AdcReadStreamClientC
- Lai lasītu pie ADC pieslēgtu sensoru, jāizveido komponents, kas sadarbojas ar kādu no šiem un piedāvā konfigurāciju

AdcReadClientC piemērs

```
generic configuration AdcReadClientC() {
  provides interface Read<uint16_t>;
  uses interface AdcConfigure<const
  msp430adc12_channel_config_t*>;
} implementation {
  // ...
}
```

DemoSensorC

- Komponents, kas implementē Read interfeisu
- Katra platforma implementē atšķirīgi, pārsvarā kā sprieguma sensoru
- tos/platforms/<platformas_nos>/ DemoSensorC.nc

DemoSensorC uz telosb

```
// tos/platforms/telosb/DemoSensorC.nc
generic configuration DemoSensorC() {
  provides interface Read<uint16_t>;
}

implementation {
  components new VoltageC() as DemoSensor;
  Read = DemoSensor;
}

// tos/platforms/telosb/VoltageC.nc
generic configuration VoltageC() {
  provides interface Read<uint16_t>;
}
implementation {
  components new Msp430InternalVoltageC();
  Read = Msp430InternalVoltageC.Read;
}
```

Msp430InternalVoltage**C**

```
generic configuration Msp430InternalVoltageC() {
  provides interface Read<uint16_t>;
  // te ir vēl ReadStream un ReadNow
}

implementation {
  components new AdcReadClientC();
  components Msp430InternalVoltageP;
  Read = AdcReadClientC;
  AdcReadClientC.AdcConfigure -> Msp430InternalVoltageP;
  // te ir vēl ReadStream un ReadNow lietas
}
```

Msp430InternalVoltage**P**

```
#include "Msp430Adc12.h"
module Msp430InternalVoltageP {
  provides interface AdcConfigure<const msp430adc12_channel_config_t*>;
} implementation {
  const msp430adc12_channel_config_t config = {
      inch: SUPPLY_VOLTAGE_HALF_CHANNEL, // ADC kanāls
      sref: REFERENCE_VREFplus_AVss,
      ref2_5v: REFVOLT_LEVEL_1_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;
  }
}
```

Pakešu sūtīšanas vingrinājumi

1. Uztaisīt aplikāciju, kas sūta pa radio divu dažādu tipu AM paketes: “Sveiki, esmu te” un “Mans skaitītājs pašlaik ir x”
2. Modificēt pirmo aplikāciju tā, lai pirmā veida paketes tiktu sūtītas pa seriālo portu, otrā veida: pa radio
3. Klausīties paketes “Cik ir Jūsu skaitītājs?” un sūtīt atbildē “Mans skaitītājs pašlaik ir x”
4. Pa radio sūtīt paketi, kurā iekšā studenta vārds (20B) un apliecības numura skaitliskā daļa (4B)

5. Extra: atrast, kur TinyOS kodā norādīts, ka maksimālais datu izmērs paketē ir 28

Sensoru lasīšanas vingrinājumi

1. Uztaisīt DemoSensorC analogu, kas lasa gaismas sensoru. ADC kanālu atrast *TMote Sky datasheet*
2. Reaģēt uz pogas "*User*" nospiešanu, ieslēdzot/ izslēdzot spīddiodi

Sensoru lasīšanas vingr. II

1. Lasīt temperatūras sensoru, sūtīt pa seriālo portu paketi, kura satur 2B garu nolasīto ADC vērtību un 2B garu konvertēto vērtību: Celsija grādi ar 2 zīmēm aiz komata. Piem, 25.12 grādi tiek attēloti kā skaitlis 2512. Konvertācijas algoritmu skatīt *TMote Sky datasheet*

-- sensoru sadaļas beigas --

Grābekļi, uz kuriem nekāpt

- Atšķirības indiāņos
- nesC valodā lokālos mainīgos jādeklarē funkcijas sākumā
- `make telosb reinstall` nepārkompilē programmu
- Moti nevar noprogrammēt, ja pašlaik tiek lasīts tās seriālais ports

Tutorials

- http://docs.tinyos.net/tinywiki/index.php/TinyOS_Tutorials

- Ir vērts izpētīt!

2. Eseja: TinyOS

- Tēma:
  - “Kas ir TinyOS vidē uzdevums, notikums un komanda?”
  - papildus: “lietas, par ko [TinyOS kontekstā] vēlos uzzināt vairāk vai kas palika neskaidras”

- Eseja var būt īsa, bet ir jāuzraksta par abām tēmām

- Termiņš: 19.09.2012 10:00

1. Esejas rezultāti

- Visas iesūtītās esejas tiek ieskaitītas
  - Ja atbilst dotajai tēmai, protams
- Visus darbus jāsūta uz pareizo e-pastu!
  - uldis.bojars at gmail.com

1. Praktiskais darbs

- Informācija wiki:
  - http://selavo.lv/wiki/index.php/LU-BST-b12:PD
- Morzes kods
  - izspīdināt savu vārdu uz gaismas diodes
- 2 bonus uzdevumi
  - iespēja iegūt papildus 100% punktu
- Termiņš
  - trešdien, 03.10.2011 10:00