Univerza v Ljubljani

MACHINE VISION GROUP

# Sistemi Daljinskega Vodenja
# Vaja 3

Matej Kristan
<matej.kristan@fe.uni-lj.si>

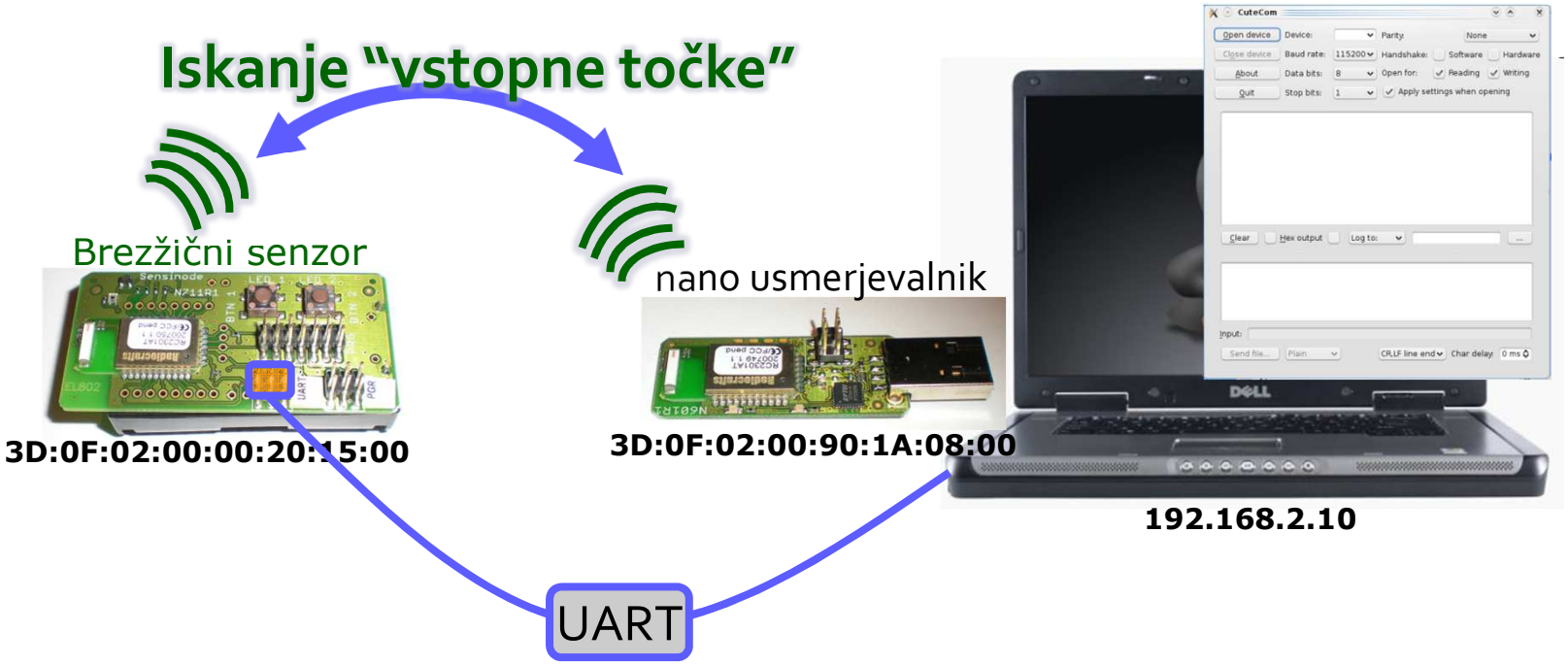
Laboratorij za Strojni Vid
Fakulteta za elektrotehniko, Univerza v Ljubljani
matej.kristan@fe.uni-lj.si

# Česa smo se naučili v drugi vaji: Delovanje

Komunikacija preko UART, Signalizacija z LED diodami.

Ukaze smo pošiljali preko UART

Vozlišče lahko "ročno" postavimo na izbran kanal



# Vsebina vaj: Odkrivanje vstopne točke

- NanoStack.
- Aktivno odkrivanje VT.
- Avtonomno vozlišče

# Standardna postavitev

- Vozlišče komunicira preko **vstopne točke** (angl. gateway).
- Na voljo imamo **enajst** kanalov.
- Za uspešno komunikacijo morata biti **oba na istem kanalu**.
- Dogovarjanje za komunikacijo izvedeno preko **ICMP** paketkov (Internet Control Message Protocol).
- **Kdo oglašuje svojo prisotnost? Kdo posluša? Kdo poizveduje?**
- Dva tipa: (1) **Pasivno odkrivanje** in (2) **Aktivno odkrivanje**

# Pasivno odkrivanje vstopne točke

1. Navadno **Vstopna Točka odda RAM** vsakih **15s** na izbranem kanalu **in posluša**.
   - NanoStack funkcija: **portCHAR gw_advertisment(void)**
2. **Vozlišče periodično preklaplja** med kanali in posluša.
3. Na vozlišču se **sproži dogodek**: ROUTER_DISCOVER_RESPONSE
4. **Vozlišče si zapomni naslov MAC** vstopne točke (angl. gateway).
5. **Vozlišče lahko** pošilja/prejema podatke z vstopne točke.

ICMP Router Advertisement Message (RAM)

15s

Brezžično vozlišče

3D:0F:02:00:00:20:15:00

•event: ROUTER_DISCOVER_RESPONSE
•se konfigurira

Vstopna točka

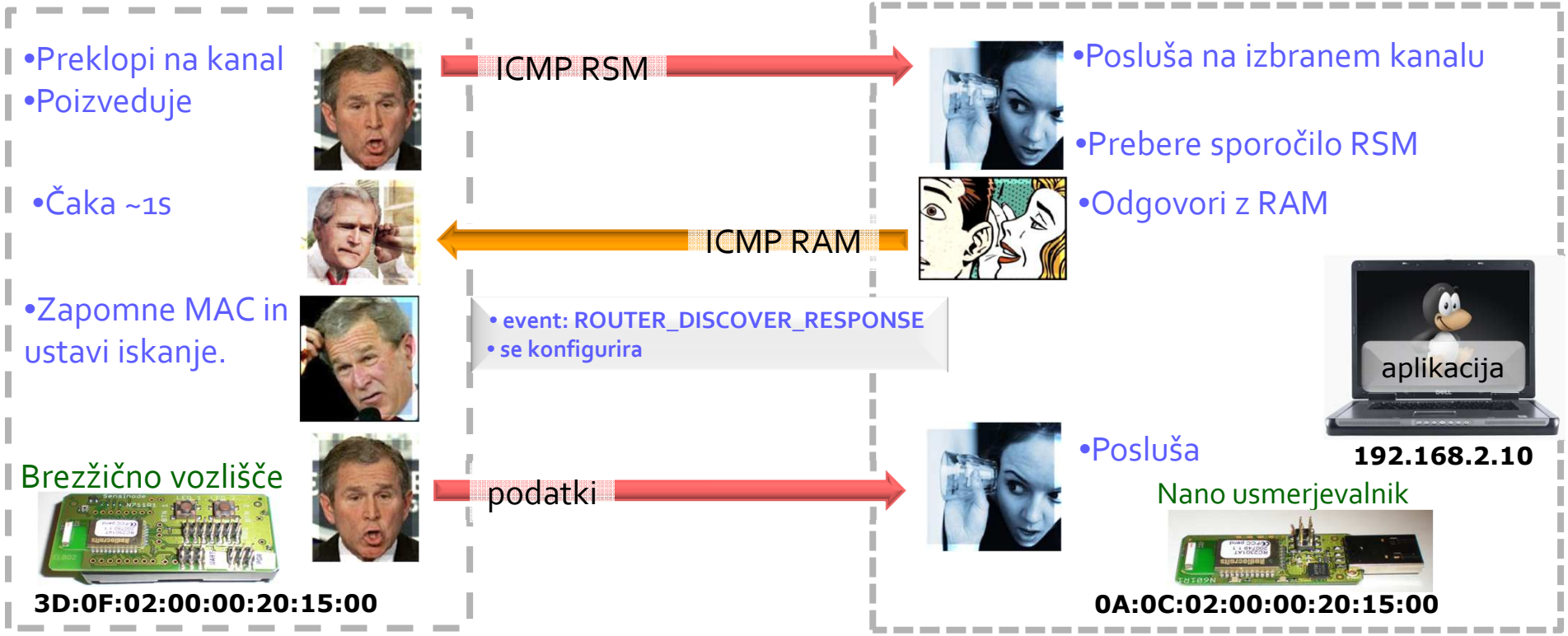
0A:0C:02:00:00:20:15:00


192.168.2.10

podatki, itd

# Aktivno odkrivanje vstopne točke

- **Vstopna točka samo posluša** na izbranem komunikacijskem kanalu.
- **Vozlišče periodično preklaplja med kanali**, npr. vsako sekundo
  - **Generira** ICMP **RSM** (Router Solicitation Message).
  - **Poplavi kanal** (angl. flooding) tako, da pošlje RSM vsem napravam na kanalu (MAC=FFF….FF).
- **Vstopna točka**, ki je na istem kanalu, **prejme RSM**.
- **VT** se **odzove na RMS z** Router Advertisement Message (**RAM**)



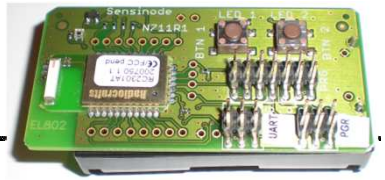
# Cilj današnje naloge: avtonomno vozlišče

- Vozlišče preklopi na kanal in poizveduje po vstopni točki.
- Če v eni sekundi ne dobi odziva, preklopi na nov kanal.
- Preklaplja ciklično med kanali.
- Ko prestreže odziv od vstopne točke in **preveri njeno identiteto(!)**.
- Če je identiteta ustrezna, si zapomni njene podatke.
- S tem je vozlišče pripravljeno na oddajanje na vstopno točko.
- Implementirali bomo dodatne funkcije za komunikacijo preko UART porta.

# Podrobneje o nalogi

- Naloga je sestavljena iz **dveh zaključenih podnalog**
- Pri vsaki se bomo ukvarjali s specifično funkcionalnostjo.
- **Prvi del:**
  - Spoznali se bomo z dogodki (angl. events)
  - Vozlišče neprestano poizveduje na ročno izbranem kanalu
  - Prestreže dogodek in izpiše na UART naslov MAC detektirane vstopne točke
- **Drugi del:**
  - Vozlišče avtomatsko preklaplja med kanali in poizveduje
  - Vozlišče v naprej pozna identiteto (MAC) iskane vstopne točke (VT).
  - Ko prestreže dogodek, preveri identiteto VTja in se primerno odzove.

# Tipična struktura programa



1. vključitve za predprevajalnik

```
#include <stdlib.h>
#include <string.h>
#include <sys/inttypes.h>
```

2. deklaracije

```
// global variables
portTickType    scan_start ;
// my functions
static void vAppTask( int8_t *pvParameters );
```

3. glavna funkcija

```
/* Main task, initialize hardware and sta
int main( void )
{
        /* Initialize the Nano hardware *
        LED_INIT(); // initialize leds
```

4. definicije naših funkcij

```
static void vAppTask( int8_t *pvParameters )
{
        pvParameters ; // just so that the c
        /* Start the debug UART at 115k */
```

1. Definicije in dodatne inicializacije
2. Glavna zanka, kamor vstavimo program.

4

```
static void vAppTask( int8_t *pvParameters )
{
    pvParameters ; // just so that the compiler doesn't complain.

1   // enter infinite loop
    for (;;) {
       /* here comes the main code */

2
      /* end of the main code */
     /* Sleep for 500 ms */
    vTaskDelay( 500 / portTICK_RATE_MS );
    }

}
```

# Konec.

- Hvala.

# Delo z dogodki

- Definicija: `stack_event_t stack_event ;`
- Inicializacija: `stack_service_init( callback_function )`
- Poslušanje: `waiting_stack_event( time_to_wait )`
- Če ni dogodka, vrne 0 (NULL), kar je *kazalec na* prazno polje.
- Če prestreže dogodek, kreira polje in *vrne kazalec* nanj

  !!V tem primeru moramo polje po uporabi sprostiti:

  `stack_buffer_free(buff_event_t buff_event)`

- Za lažjo analizo vsebine polja nam NanoStack ponuja:

  `parse_event_message(buff_event_t buff_event)`