Vhodno-izhodne naprave (VIN)

Predavanja

11. Načrtovanje in programiranje vgrajenih sistemov – teorija in praksa

Robert Rozman rozman@fri.uni-lj.si

Tekoča obvestila

Predstavitev Red Pitaya

VIN projekt – zadnje predavanje

Vsebina

Vsebina I, II:

1. Načrtovanje vgrajenih sistemov (HW, SW, …)

2. Programiranje vgrajenih sistemov – primeri:
   - Cubesensors, Tevel, D13, Tinia, …

3. Nivoji programiranja

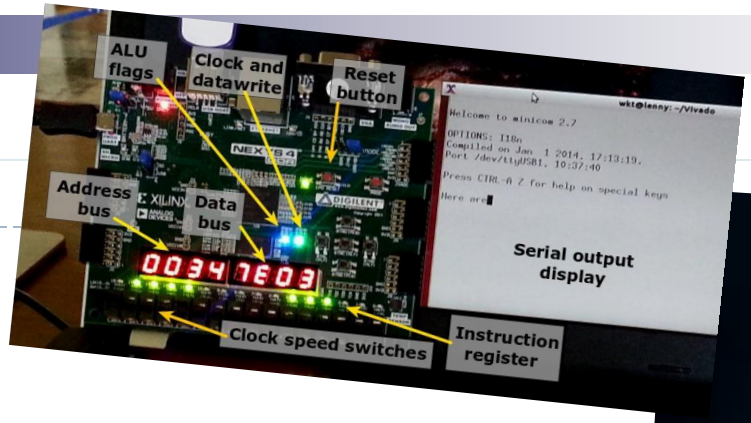
4. Podrobnejši primeri programiranja – RTOS
   - 4.1 Splošno o RTOS
   - 4.2 FreeRTOS
   - 4.3 MQX RTOS

Vsebina I:

1. Načrtovanje vgrajenih sistemov (HW, SW, …)

2. Programiranje vgrajenih sistemov :
   - splošen pogled
   - primeri :
     - A. Cubesensors („pametne kocke“)
       - realizacija v enotni zanki, končni avtomat
     - B. Tevel – univerzalni merilniki (ekspl. cona)
     - C. D13 („pametni hišni regulator“)
       - RTOS (primer MQX)
     - D. Tinia – Prijazen dom
     - E. Pametni zabojnik
     - F. Embedded Linux (UcLinux, Buildroot)
     - G. Simulacije
   - CubeIDE: razvoj in razhroščevanje
   - Kaj po razvoju ? Skrb za stabilnost sistemov v praksi
     - preventiva in kurativa

Vsebina II:



3. Nivoji programiranja

4. Podrobnejši primeri programiranja – RTOS
   - 4.1 Splošno o RTOS
   - 4.2 FreeRTOS
   - 4.3 MQX RTOS

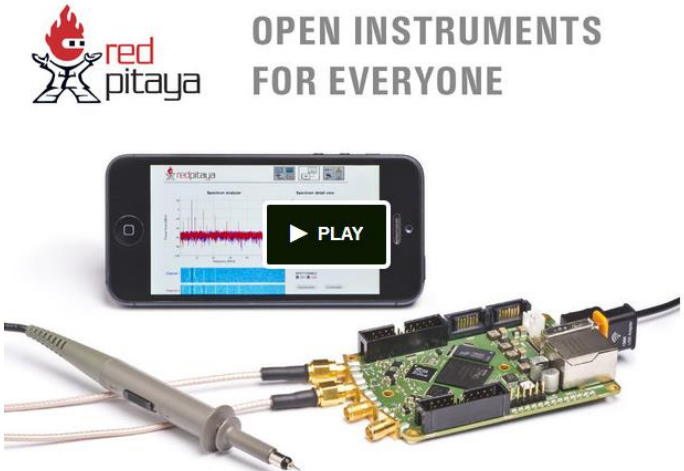

```
void mytask(uint_32 startup_parameter)  {
   /* Task initialization code */
   ....
   while (1) {
      /* Task body */
      ....
      ....
   }

}
```

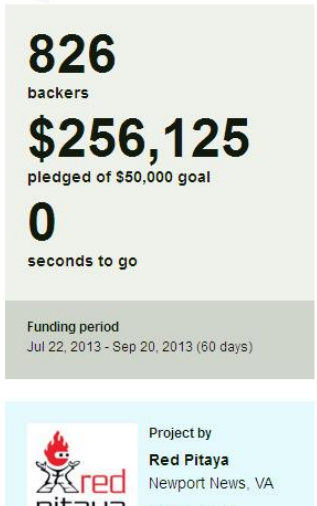
Zakaj HW (in SW) ?

Visionect is a decade-long market leader in developing ePaper solutions. Here's why.

We're constantly testing new technologies and creating the best solutions or applications of E Ink systems.

July 20, 2021



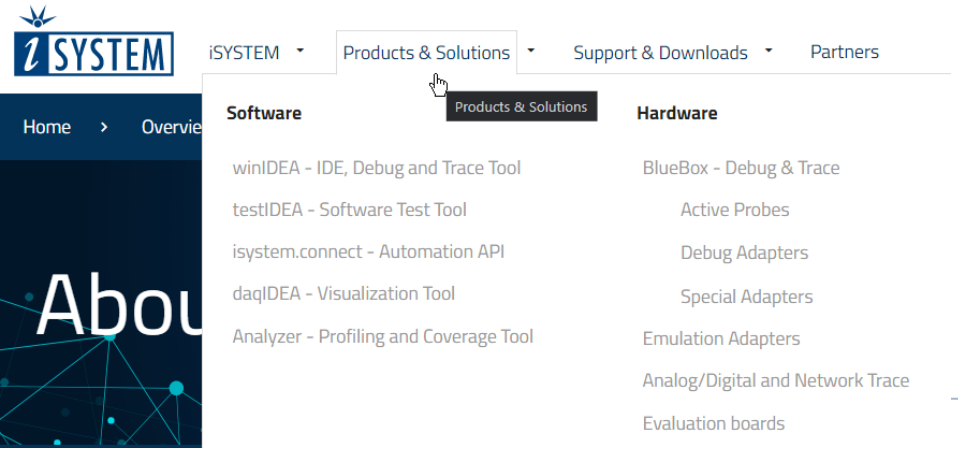


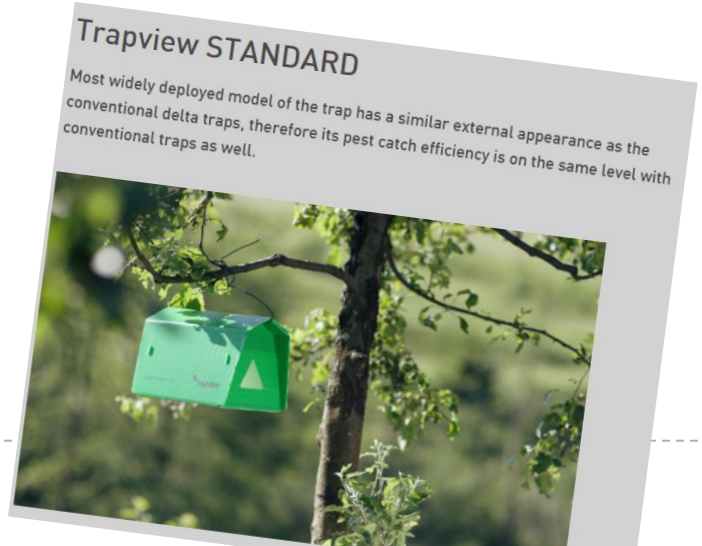
Zakaj HW (in SW) ?

Past Meetup

Code optimization on modern processors [Dejan Črnila, Dewesoft]

Space shuttle Atlantis launch monitoring with Dewesoft software





Prikaz primerov vgrajenih sistemov

D13 EQ

Tevel

Merilnik konc. plinov

STM Discovery

Tempmate S1

Cubesensor

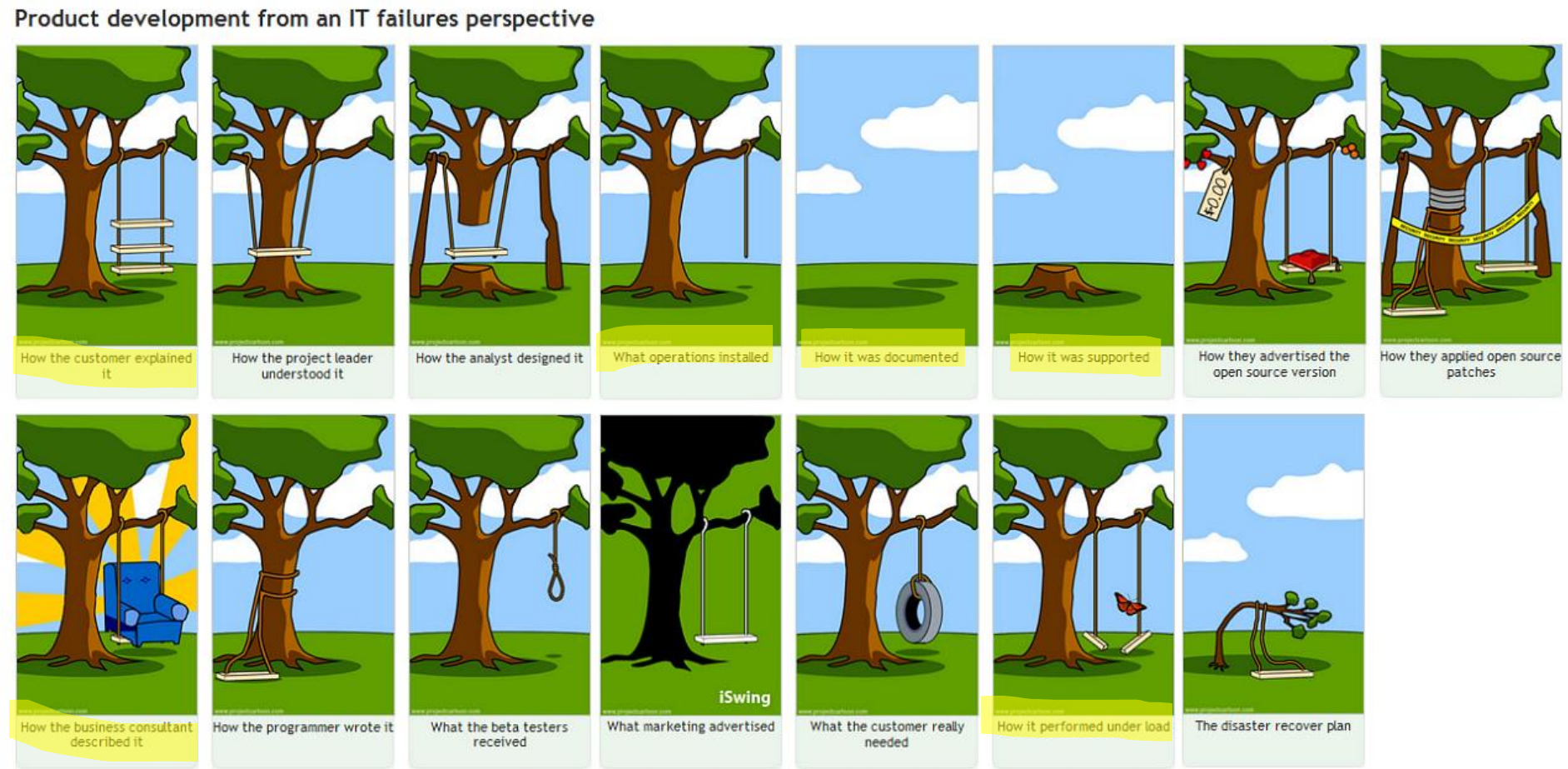
1. Načrtovanje vgrajenih sistemov

Običajen potek :

- Specifikacija – opredelitev zahtev ->
  - zelo pomembna !
- Izbira el. komponent, čipov, krmilnikov, itd… ->
  - pregled dokumentacije („Errata“, rok dobavljivosti, podpora,…)
- Načrtovanje PCB
- Prvi zagon – oživljanje sistema, razvoj SW
- Spremljanje delovanja

1. Načrtovanje vgrajenih sistemov

- Specifikacija – opredelitev zahtev
  - zelo pomembna !



1. Načrtovanje vgrajenih sistemov

Izbira el. komponent, čipov, krmilnikov, itd…

- Datasheet (DS):
  - „kako bi naj delovalo…“
- Errata:
  - „kaj vse ne deluje tako kot v DS…“
- Rok dobavljivosti
- Podpora ->

Prazna ?

Pentium FDIV bug:

The **Pentium FDIV bug** is a bug in the Intel P5 Pentium floating point unit (FPU). Because of the bug, the processor can return incorrect decimal results, an issue troublesome for the precise calculations needed in fields like math and science.

Errata : + dobro, vsaj znan problem

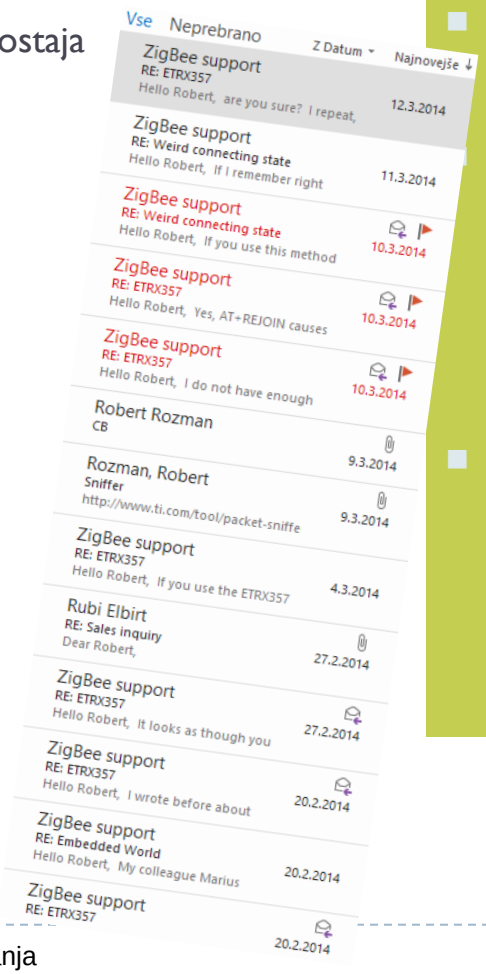
Kaj če naletimo na neznan problem?

- ***upamo na reprodukcijo in podporo***

Izbira el. komponent, čipov, krmilnikov, itd… - 2 zgodbi

Podpora - 1. zgodba :

- težave z brezžičnim modulom
  - cca. 70 emailov,
  - problem ostaja



Zaznani hrošči:

»stuck in error94«

Main.c : / **hack when zigbee module gets stuck** in error 94. nothing except AT&F resolves the issue, no ATZ, reset pin, power down */*

»no SEQ prompt«

```
/* Bits are set to 1, when message is in air (SEQ, but not yet ACK or NACK). */
/* In a perfect world this would not be needed, but it seems like module
 * sometimes (RARELY) does not send SEQ: after AT+UCAST*, but it does send ACK:
 * afterwards. If that happens, pending_messages buffer can become -1 long,
 * and old (or not yet used, invalid) messages are sent. This is a suspect
 * for ticket:185 */
```

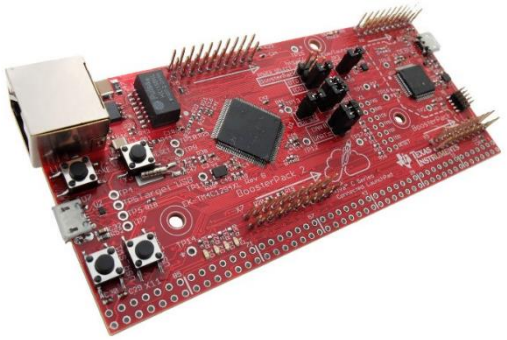
»not be able to reconnect «

```
static void check_initialized(void) {
        int r;
        /* zigbee module bug workaround.
         * Reset the module, or it will not be able to reconnect after seeing
         * a bunch of modules failing to connect (NEWNODE w/o FFD).
         */
```

Izbira el. komponent, čipov, krmilnikov, itd…

Podpora - 2. zgodba:

- Connected launchpad
  - nov izdelek – ne deluje stabilno ?
  - v enem letu ni rešitve



Connected LaunchPad Quick-start IoT Application dies after a day or two — Not Answered

We have experienced this issue on all Connected LaunchPads we have using the de-fault application shipped on the Connected LaunchPad (CLP) and also after compiling the 'qs_iot' project and programming it to the CLP.

Essentially the device stops working after a day. I do not have a specific amount of time although it should not be hard to let run a few times to see if it is always the same.

Jul 30, 2014 7:26 PM — Mike Aanenson — Community Member

Jan 7, 2015 7:19 PM

In reply to Dubnet:

Any progress on this?

Reply

Wed, Jan 28 2015 4:35 PM

In reply to Dubnet:

Hello All,

The updated code ("qs_iot" application and underlying layers) has been under test for close to two weeks now and working. The Connected Launchpad still looses connection to Exosite Server once in a while, but successfully connects back. The system state (like on time, led state and button press state) is not lost during the connect-disconnect-connect transition. The root cause (of why connection to the Exosite Server is lost in the first place) requires analysis of network traffic when the failure occurs which is intermittent..

Stellaris Sai — Intellectual 2355 points — TI Employee

Fri, Aug 14 2015 10:55 AM — Harold Broberg — Prodigy 110 points — Community Member

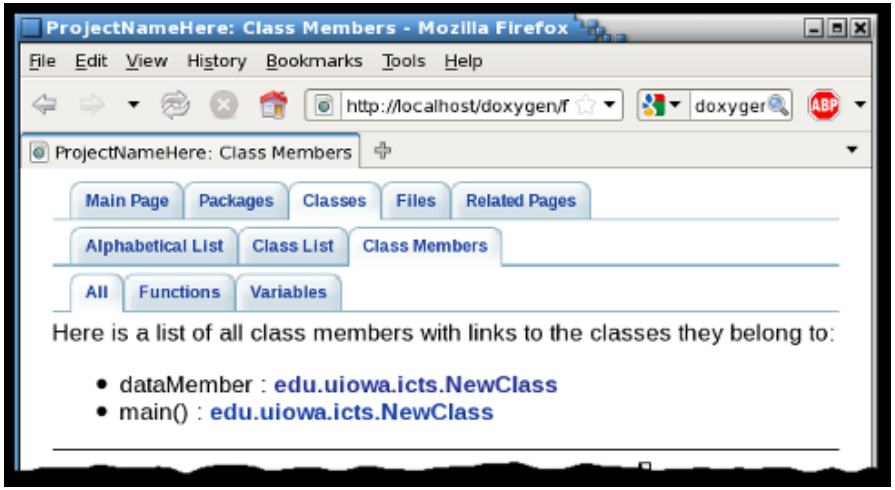
I tried that one already, but am trying it again now.
It stays on line (6 minutes now). It counts button presses and shows the temperature. But when I click an LED to on, on Exosite, after a bit (23 seconds & 34 seconds, measured) the onscreen button goes back to the off position. Neither LED ever turns on, with either button.

2. Programiranje vgrajenih sistemov

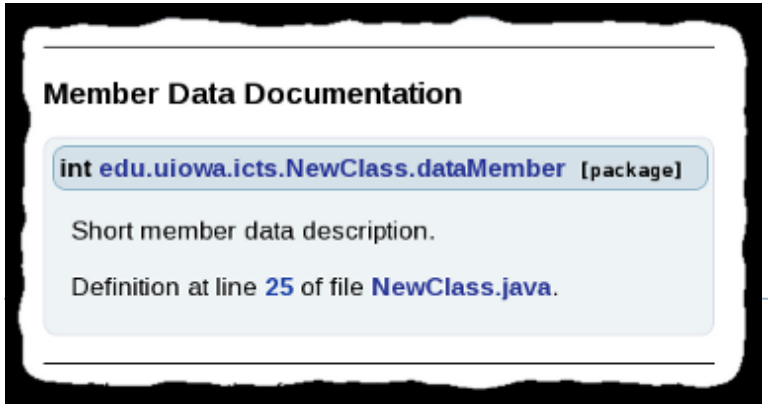
Splošno :

- Orodja :
  - IDE: CubeIDE, IAR, Keil, Eclipse
    - Pomembne funkcionalnosti : Debug, Profile, …
  - Doxygen.org :

```
/**
 * @brief   Short member data description.
 */
int dataMember;
```



Primer: D13.chm



2. Programiranje vgrajenih sistemov

Splošne metode :

razvoj

- Pravila robustnega programiranja
  - MISRA C ->
- Sledenje delovanja programa
  - Debugger
  - Serijska konzola
  - Log datoteka (lokalno)
  - Oddaljen nadzor/logiranje (splet)

spremljanje

```
2015-01-11 05:45:37 CRIT   232      0 FP      WDT has expired
2015-01-11 05:46:21 CRIT   232      0 MNG     WDT has expired
```

```
2015-01-09 15:00:02 INFO    60      0 CMDEXECUTE  CMD:Execute Cmd[72]
2015-01-09 15:00:02 INFO    60      0 CMDEXECUTE CMD:SendSett
2015-01-09 15:04:02 CRIT   232      0 CMDEXECUTE  WDT has expired
```

2. Programiranje vgrajenih sistemov

Pravila robustnega programiranja (preventiva)

- MISRA C (1998, 2004, 2012):
  - MISRA = **M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation
  - 143 pravil (preverljivih z analizo) in 16 smernic
  - skupine pravil:
    - razlike med prevajalniki (npr. velikost tipa Integer)
    - brez funkcij s pogostejšimi napakami (npr. malloc)
    - obvladljiva koda (pravila imenovanja, komentiranja…)
    - primeri dobre prakse
    - omejitve kompleksnosti
  - že integrirano v nekatera IDE orodja:
    - IAR, Green Hills, …

Rule 14.8 (required): **The statement forming the body of a *switch*, *while*, *do … while* or *for* statement shall be a compound statement.**

The statement that forms the body of a *switch* statement or a *while*, *do … while* or *for* loop, shall be a compound statement (enclosed within braces), even if that compound statement contains a single statement.

For example:

```
for (i = 0; i < N_ELEMENTS; ++i)
{
   buffer[i] = 0;          /* Even a single statement must be in braces */
}
```

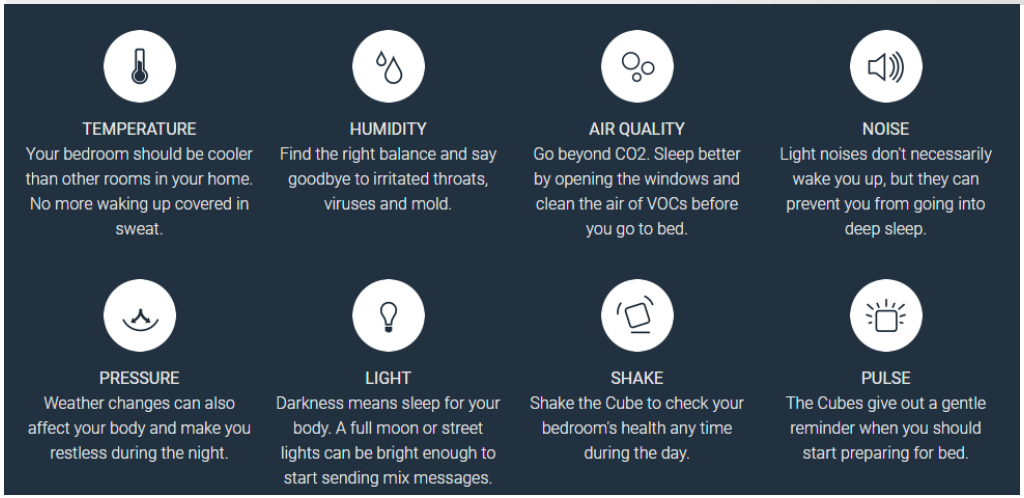
Rule 14.4 (required): **The *goto* statement shall not be used.**

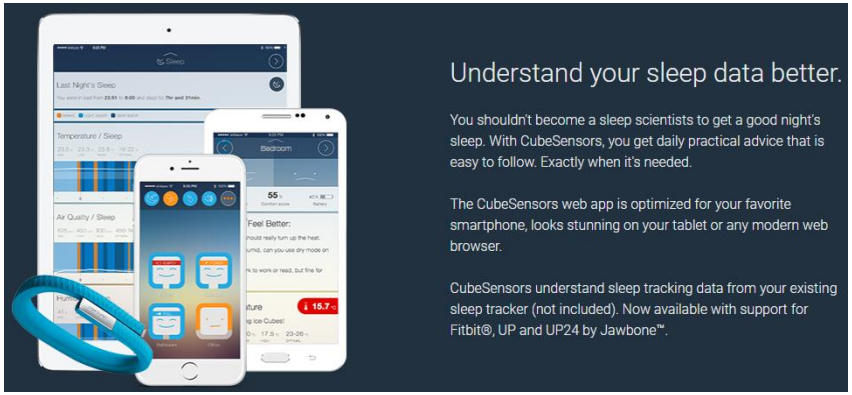
Rule 14.5 (required): **The *continue* statement shall not be used.**

Rule 14.6 (required): **For any iteration statement there shall be at most one *break* statement used for loop termination.**

2. Programiranje vgrajenih sistemov - Primeri

A. Cubesensors („pametne kocke“):





2. Programiranje vgrajenih sistemov - Prin

A. Cubesensors („pametne kocke“):

- Osnovni model CPU – ARM Cortex M0
- Brezžična komunikacija („Zigbee“)

- Zahteve :
  - nizka poraba, cena
- Odločitev:
  - prehod iz M3 prototipa na M0
- Posledice:
  - omejeno debugiranje
  - ni serijske konzole
  - zelo omejeni viri

2. Programiranje vgrajenih sistemov

A. Cubesensors („pametne kocke“):

Enotna glavna zanka (kompleksnejša izvedba) = **končni avtomat**

- brezžična komunikacija + branje senzorjev + spanje
- boljša organizacija kode, lažje vzdrževanje

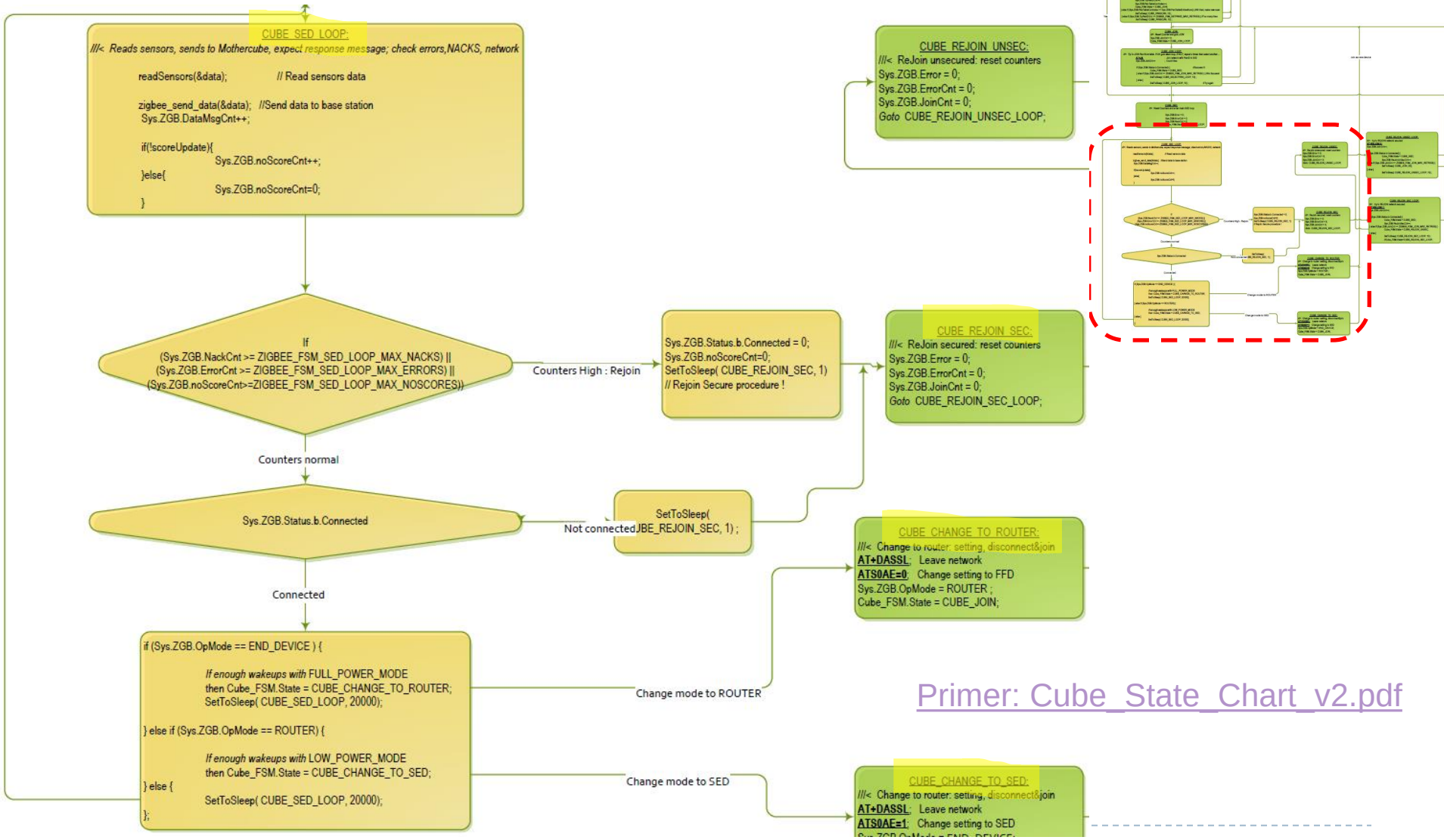
```
switch (FSM.State) {

    case CHECK_POWER_ON_REASON:
            ///<  FSM.State: after reset or power up. SW Reset and check if it can join right away...

          if  VSE_OK then FSM.State  = CHECK_BAUDRATE …

        break;

    case CHECK_BAUDRATE:
            ///<  FSM.State: after reset or power up. SW Reset and check if it can join right away...
          …

        break;
```

2. Programiranje vgrajenih sistemov

Cubesensors („pametne kocke“):

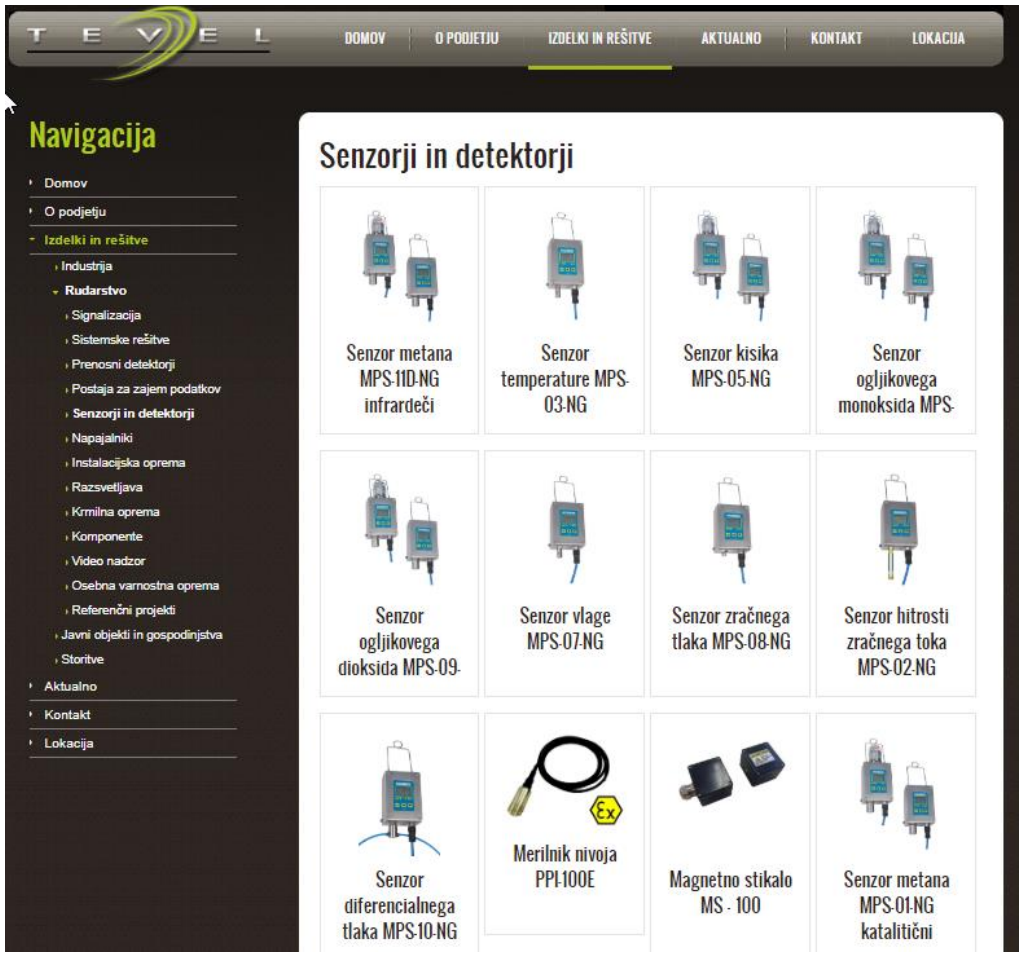
Diagram poteka (končni avtomat):

- Primer glavnega stanja



Primer: Cube_State_Chart_v2.pdf

B. Tevel Pametni merilniki (rudarstvo)



Tevel v Kazahstanu cilja na rudarsko panogo

(video, foto) Zasavci osvajajo azijske in balkanske rudnike

Gospodarstvo Prva stran

Tevel na Kosovu

10. 10. 2018

Gospodarstvo

Tevel načrtuje nakup nemškega Wölkeja

2. Programiranje vgrajenih sistemov - Primeri

в. Tevel Pametni merilniki (rudarstvo):

Enotna glavna zanka – enostavnejša izvedba

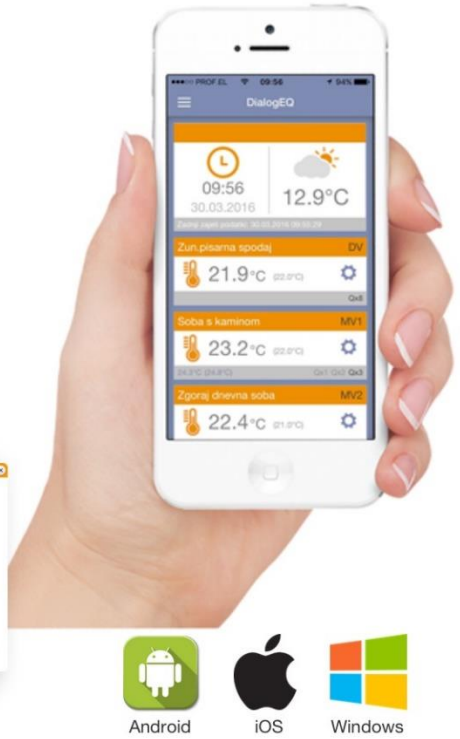
```
{ …


      if (Timer_1sec)   {
            readSensors(&data);  // Read sensors
            send_data(&data);     //  Send data to gateway
             Timer_1sec = 0;
      }

      if (Timer_50msec)   {
            readKeys(&keys);       // Read user keys
            readInputs(&inputs);   // Read digital inputs
            Timer_50msec = 0;
      }

}
```

2. Programiranje vgrajenih sistemov - Primeri

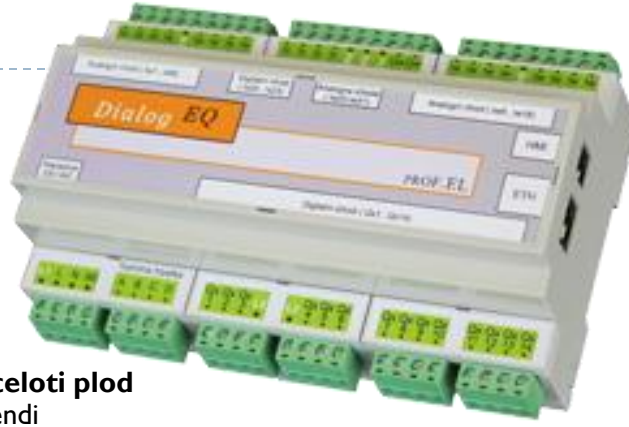
C. DIALOG EQ („pametni regulator“)

RTOS – ločeni procesi
(REG,TCP,WEB,MODBUS,CANBUS)

– zahtevnejša izvedba



2. Programiranje vgrajenih sistemov - Primeri



C. DIALOG EQ („pametni regulator“)

RTOS – ločeni procesi (REG,TCP,WEB,MODBUS,CANBUS) – zahtevnejša izvedba

Regulator DIALOG EQ (krajše: DEQ) predstavlja najnovejšo generacijo pametnih regulatorjev. **Je v celoti plod slovenskega znanja.** V podjetju PROF.EL smo ga razvili, saj želje strank in pa predvsem sodobni trendi narekujejo daljinski nadzor in avtomatizacijo ogrevalno/hladilnih sistemov oziroma celovite rešitve za pametni dom (smart house).

Regulator DEQ vam omogoča:

- 24h spremljanje delovanja in upravljanje s sistemom na daljavo (računalnik, tablica, pametni telefon),
- varno shranjevanje vseh uporabnikovih nastavitev v oblaku,
- pregled in analizo delovanja sistema,
- prijazen uporabniški vmesnik (interni WEB, WEB aplikacija, aplikacija za mobilne telefone Android, iOS, Windows),
- beleženje in shranjevanje podatkov (črna skrinjica v oblaku),
- daljinsko pomoč servisne ekipe,
- daljinsko posodabljanje (up-grade) programske opreme.

Algoritmi so pripravljeni za vodenje in nadzor kotlov na biomaso in olje, toplotnih črpalk, sončnih kolektorjev, sanitarne vode s cirkulacijsko črpalko idr.

2. Programiranje vgrajenih sistemov - Primeri

C. DIALOG EQ („pametni regulator“)

D13 („HVAC regulator“)

Izhodi in algoritmi za krmiljenje:

- Direktne veje
- Mešalne veje 2x
- Sanitarne vode
- Sončnih kolektorjev

Kompleksnejša izvedba:

- MQX RTOS
- Opravila :
  - FP_TASK glavni krmilni program
  - MODBUS_TASK Modbus strežnik
  - TCPCLIENT_TASK povezava s podatkovnim strežnikom v oblaku
  - httpd_server spletni strežnik – lokalni portal
  - CMDEXECUTE_TASK izvedba ukazov
  - FTPCLIENT_TASK FTP prenosi

2. Programiranje vgrajenih sistemov - Primeri

C. DIALOG EQ („pametni regulator“)

RTOS (primer MQX) :

Opravila („Tasks“)

```
const TASK_TEMPLATE_STRUCT  MQX_template_list[] =
{
   /* Task Index,      Function,          Stack, Priority,                 Name,                     Attributes,              Param,   Time Slice */
{ MNG_TASK,      MngTask,           1200,  TASK_PRIORITY_MNG_TASK,   MNG_TASK_DES,             MQX_AUTO_START_TASK,     0,       0 },

{ SHELL_TASK,   ShellTask,          2000,  TASK_PRIORITY_SHELL,  SHELL_TASK_DES,    0,                        0,       0 },

{ FP_TASK,         FunPgmTask,       2000,  TASK_PRIORITY_FP,         FP_TASK_DES,              0,                       0,       0 },

{ TNSH_TASK,      TelnetClientShell, 2000,  TASK_PRIORITY_TNETSH,TNSH_TASK_DES,     0,                        0,       0 },

{ TCPCLIENT_TASK,TCPClient_Task, 2000,  TASK_PRIORITY_TCPCLIENT,TCPCLIENT_TASK_DES,  0,                       0,       0 },

{ MODBUS_TASK,Modbus_Task,    2000,  TASK_PRIORITY_MODBUS,MODBUS_TASK_DES,  0,                        0,       0 },

{ EVTALM_TASK,EventAlmTask,       2000,  TASK_PRIORITY_EVTALM,EVTALM_TASK_DES, 0,                        0,       0 },

{ AIN_TASK,        AinTask,           500,   TASK_PRIORITY_AIN,        AIN_TASK_DES,             0,                       0,       0 },

{ NETMNG_TASK,NetMngTask,        1000,  TASK_PRIORITY_NETMNG,NETMNG_TASK_DES,        0,                       0,       0 },

{ 0 }

};
```

C. DIALOG EQ („pametni regulator“)

RTOS (primer MQX opravila) :

Glavna regulacijska zanka („FP_TASK“)

```
void FunPgmTask (uint_32 initial_data)
{
FunPgmInit();

// register task for system messages
rc = SysMsgRegister ();

// WDT control
WdtRegister (15000, WDT_ACTION_LOG);

// ------------------------------------  main execution loop ----------------------------
while (TRUE)  {

            _time_get_elapsed (&fp_start_time);   //Measure processing time fp_start_time

            WdtReset ();

            FunPrepareFPData();          // Prepare FP data
            FunRegulation();             // Iterate regulation loops
            FunCommitFPData();           // Commit any changes back to system

            _time_get_elapsed (&fp_end_time);   //Measure processing time
            _time_diff (&fp_start_time, &fp_end_time, &fp_loop_time);      // get elapsed time
            FPLoopTime=(fp_loop_time.SECONDS * 1000) + fp_loop_time.MILLISECONDS;

            _time_delay(1000-FPLoopTime);                  // wait for 1000 ms - loop time in ms
            }
_task_block();                                             // Shouldn‘t reach this point
}
```

```
/** @brief FP: Main Functional Program Task.
Calls FunPgmInit for initialization and then runs endless main FP loop.
 *
 *  This is main functional program task.
 *  It will first run Initializations: FunPgmInit();
 *  Then it will proceed in endless loop :
 *              FunPrepareFPData();     // Prepare FP data
 *              FunRegulation();        // Iterate regulation loops
 *              FunCommitFPData();      // Commit any changes back to system
 *              check if settings changed - if yes, then read all settings
 */
```

void FunPgmTask (uint_32 initial_data)

FP: Main Functional Program Task. Calls FunPgmInit for initialization and then runs endless main FP loop.

This is main functional program task. It will first run Initializations: **FunPgmInit()**; Then it will proceed in endless loop : **FunPrepareFPData()**; // Prepare FP data **FunRegulation()**; // Iterate regulation loops **FunCommitFPData()**; // Commit any changes back to system check if settings changed - if yes, then read all settings

Todo:
Temporary - should't be used in production code !!!

Definition at line **139** of file **fp.c**.

References **APPCFG_DEFAULT_FP_USER_ACCCODE**, **APPDBG_PRINTF**, **D13_GVARS::Day**, **FunCommitFPData()**, **FunLogCurrentState()**, **FunPgmInit()**, **FunPrepareFPData()**, **FunRegulation()**, **FunSimCommitFPData()**, **FunSimLogCurrentState()**, **FunSimPgmInit()**, **FunSimPrepareFPData()**, **FunSimRegulation()**, **FP_DATA::GVars**, **D13_GVARS::Hour**, **D13_GVARS::Minute**, **D13_GVARS::Month**, **Read_FPSettings()**, **D13_GVARS::Second**, and **D13_GVARS::Year**.

D. Tinia – prijazen dom

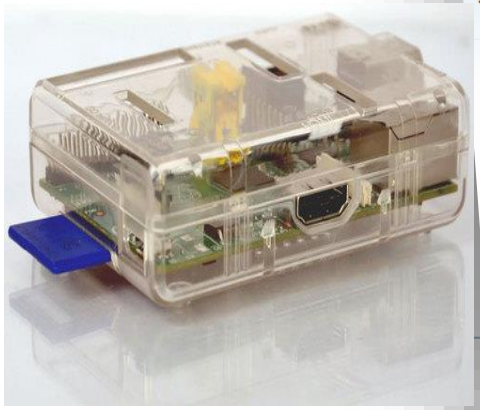
TBS – „Tinia Building Server”

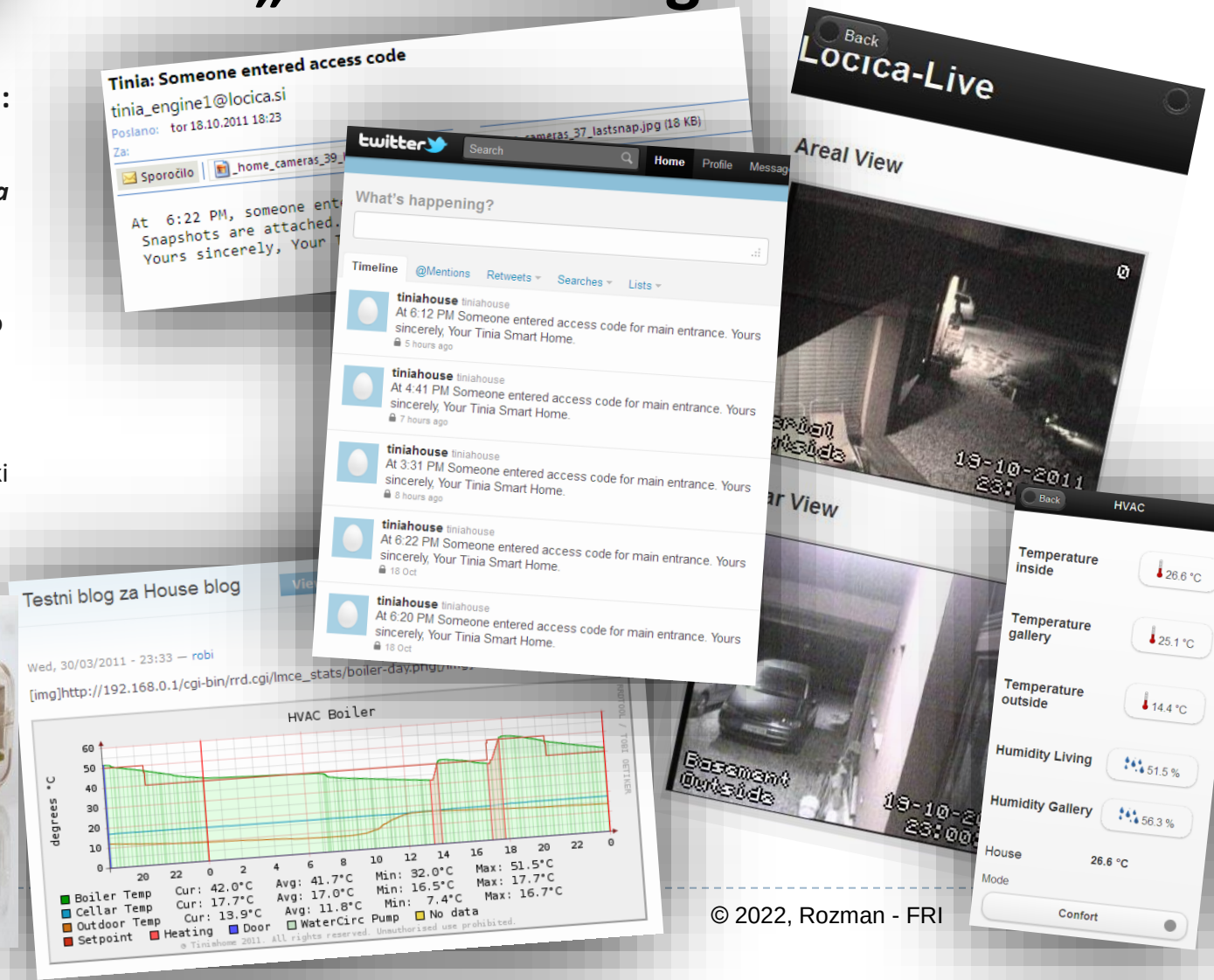
Kratek opis

TBS – „Tinia Building Server”:

Nadzor,upravljanje in vizualizacija delovanja prijaznega doma.

- majhen, varčen, tih (5W)
- povezuje zgradbo in pametno mesto
- informiranje, povratna inf.
  - •pametni telefoni, tablice
  - •splet, soc.omrežja
- programiranje s pravili,vtičniki
- povezava s soc.omrežji
  - •Twitter,FaceBook





INTEGRA BM SYSTEM

Industrial & Building Automation

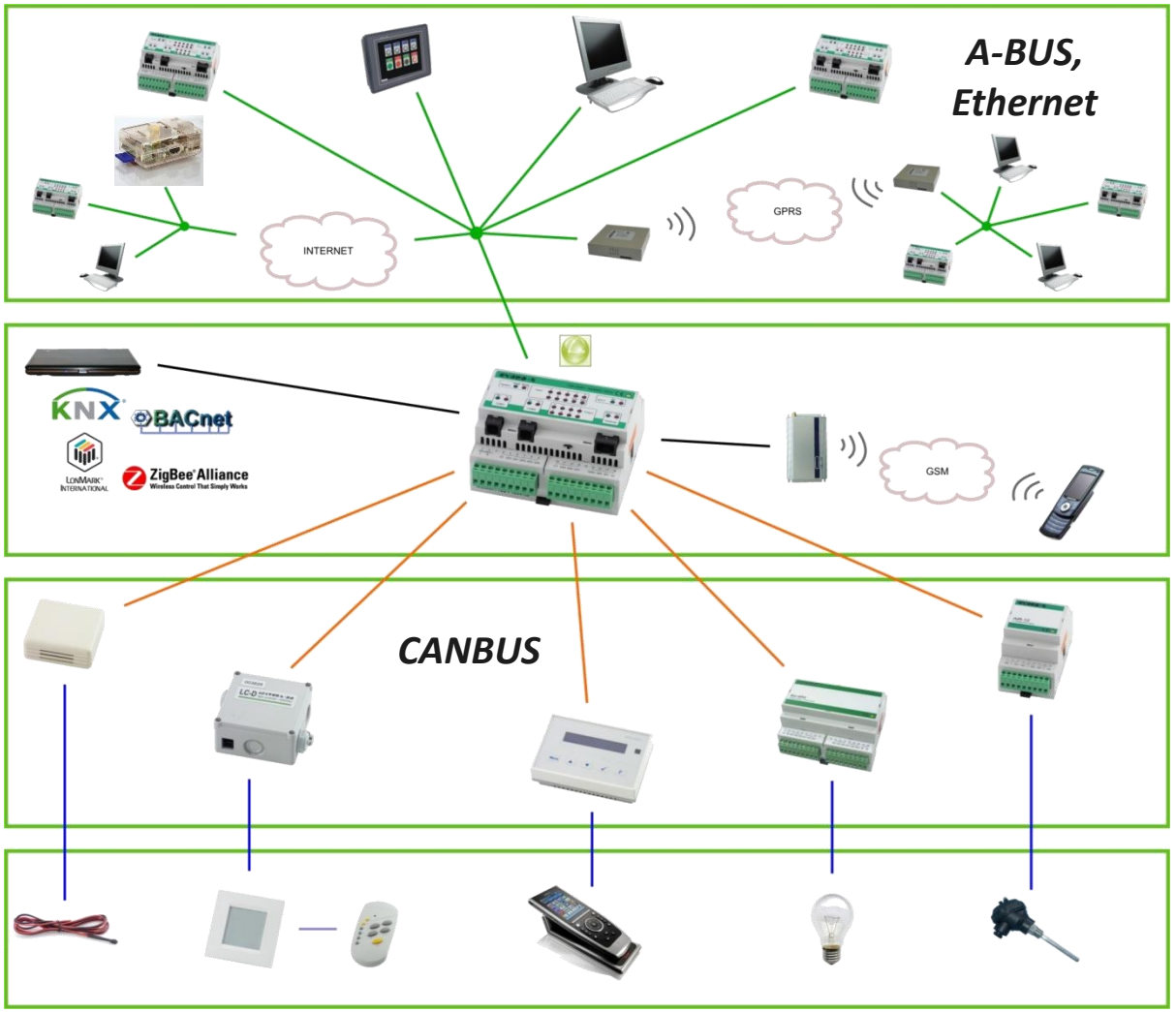
Generally

High level network

CyBro controller

Low level network

Accessories



Tinia – prijazen dom

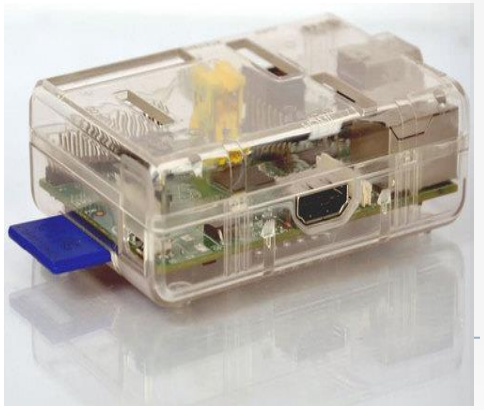
TBS – „Tinia Building Server”

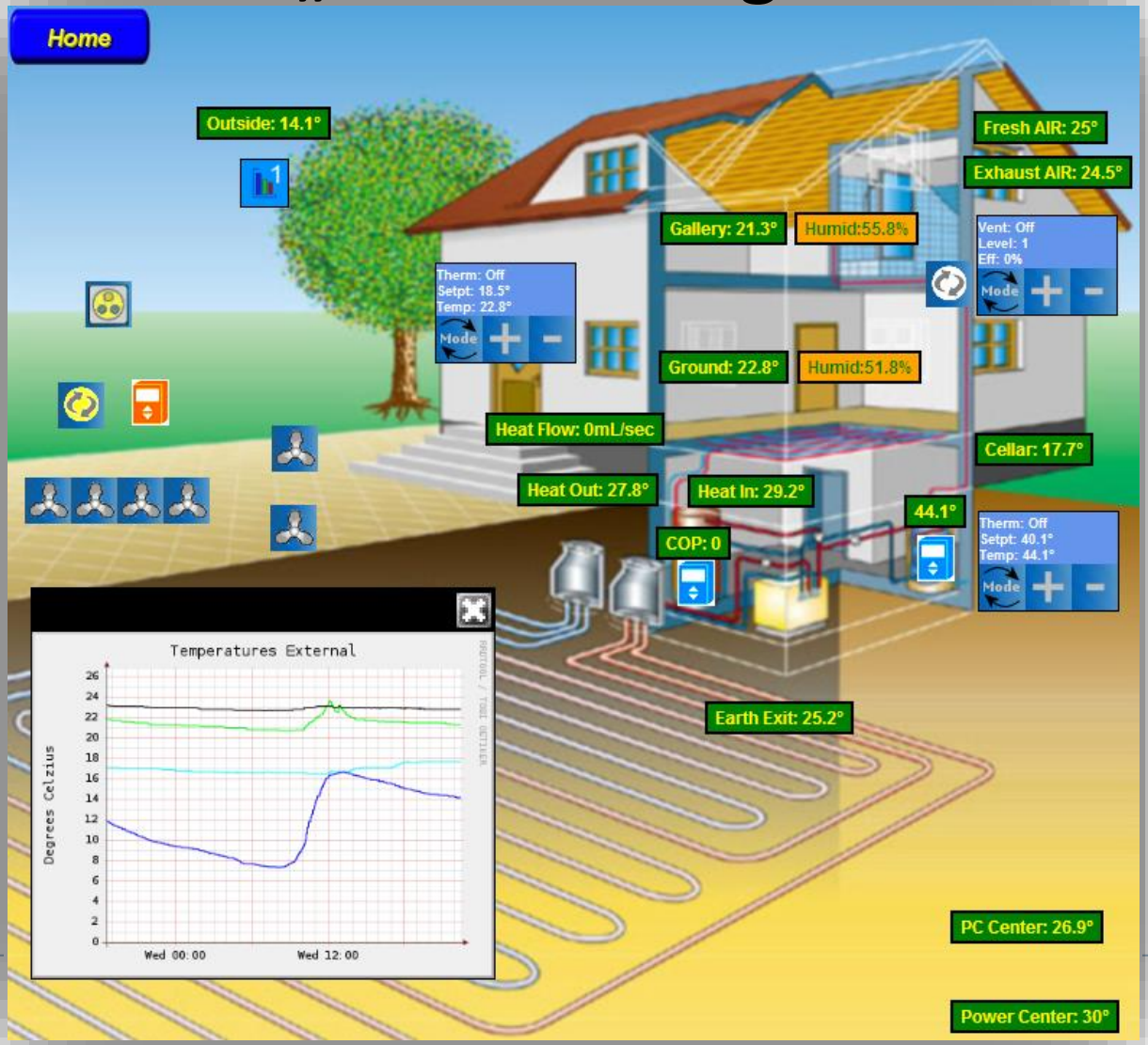
Kratek opis

TBS – „Tinia Building Server”:

***Nadzor,upravljanje in vizualizacija delovanja prijaznega doma*.**

- majhen, varčen, tih (5W)
- povezuje zgradbo in pametno mesto
- informiranje, povratna inf.
  - •pametni telefoni, tablice
  - •splet, soc.omrežja
- programiranje s pravili,vtičniki
- povezava s soc.omrežji
  - •Twitter,FaceBook





Ogrevanje (prostori, sanitarna voda)

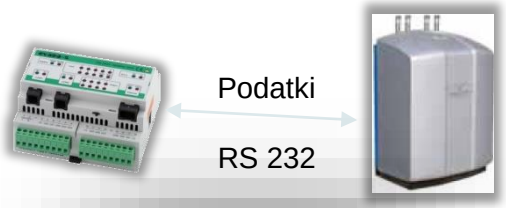
Toplotna črpalka zemlja-voda

- Zemeljski kolektor
- Talno in stensko ogrevanje
- Sanitarna voda
- Serijska komunikacija: Cybro COM2 <-> TČ

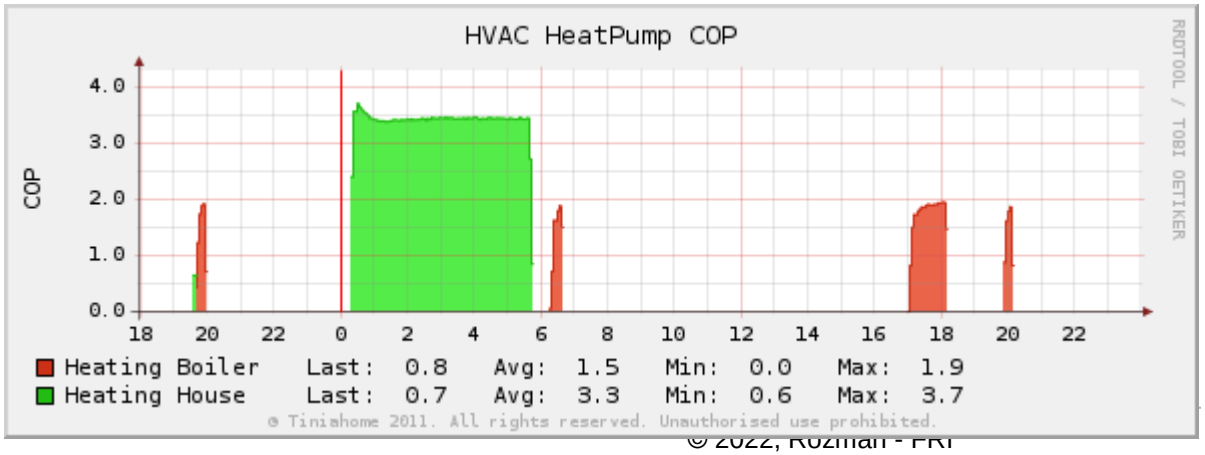
Grelno število (COP-Coefficient Of Performance):

$$COP = \frac{\text{Toplotna Moč}}{\text{Elektricna Moč}}$$

- COP ~ 3.5 Ogrevanje prostorov (Elekt.Moč =1.8 kW)
- COP ~ 2.0 Ogrevanje sanitarne vode (Elekt.Moč =3.0 kW)



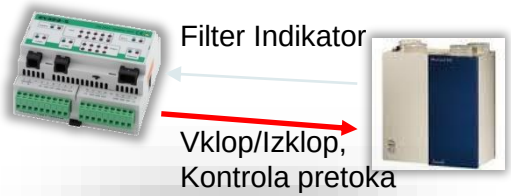
Primer zimskega dneva – COP toplotne črpalke :

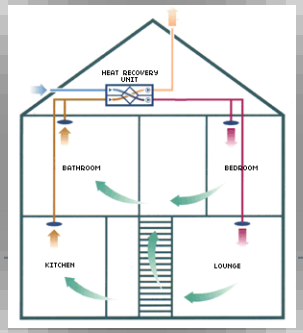


Prezračevanje

Prezračevanje s povratkom toplote (rekuperacija)

- Izčrpan zrak odda toploto svežemu zraku s >85% izkoristkom
- Vklop/izklop in kontrola pretoka
- Indikator zasičenosti filtra



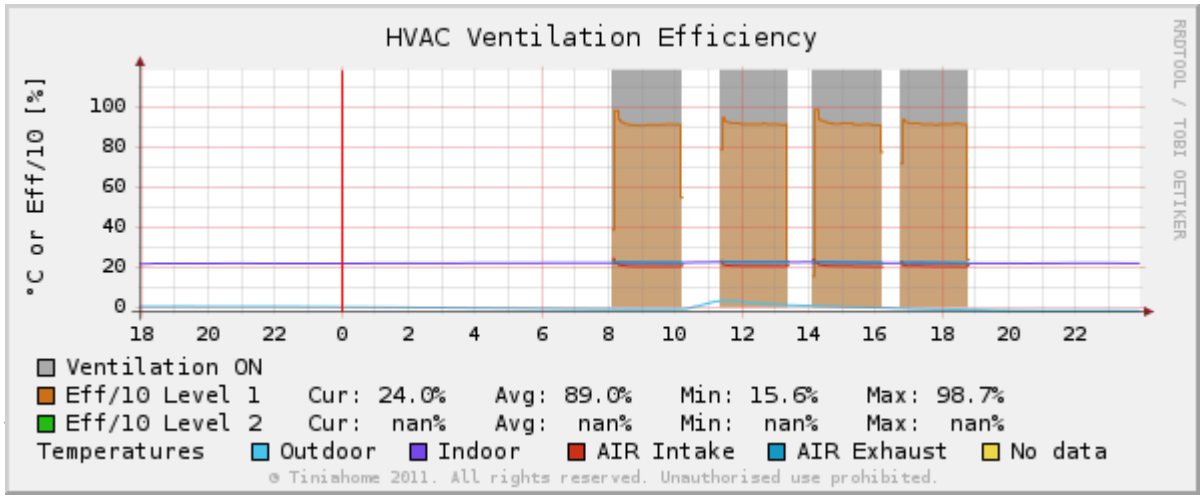


Učinkovitost rekuperacije:

$$EFF. \approx \frac{\text{Svež zrak temp.} - \text{Zunanji zrak temp.}}{\text{Izčzčrpazrak temp.} - \text{Zunanji zrak temp.}} \ [\%]$$

- Primer : Eff ~ 90% ko:
  - Zunanja Temp. ~0°C
  - Notranja Temp. ~21°C
  - Sveži zrak se segreje od ~0°C do ~19°C (rekuperacija)

Primer zimskega dneva – učinkovitost rekuperacije



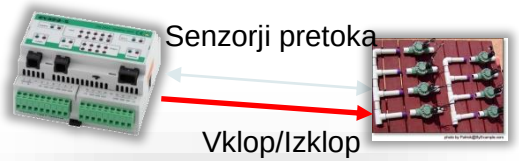
Zalivanje
(vrt, zelenica, ribnik)

Zalivanje (vrt, zelenica, ribnik)

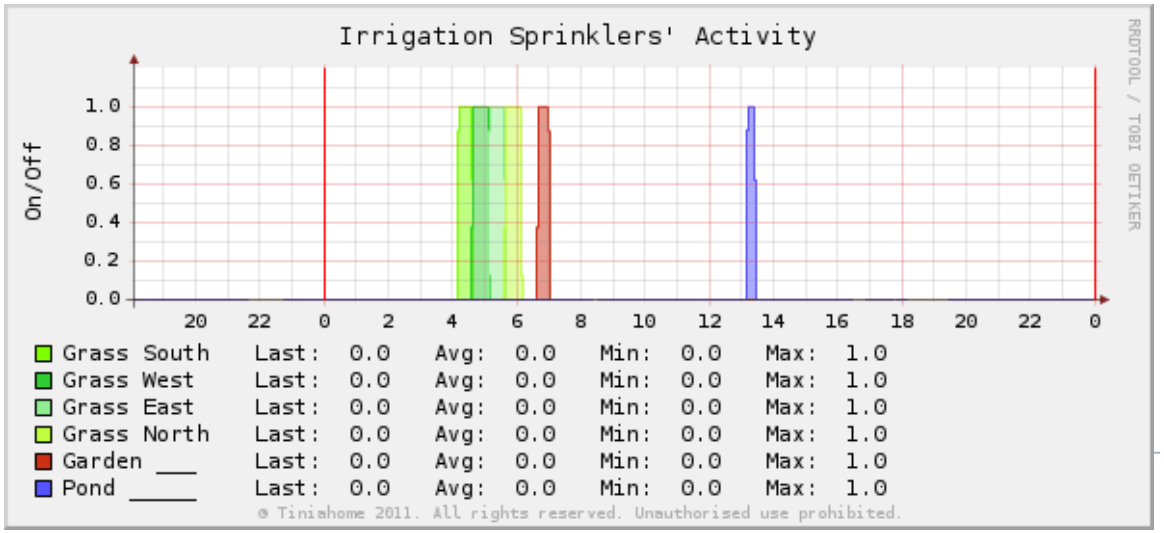
- Kontrola zalivanja v skladu z **urnikom** in **nivojem vlage** v tleh

- Vklop/Izklop & Zaznavanje pretoka

- Zaznavanje pretoka:
  - Preklop med dvema viroma :
    - podtalnica
    - vodovod



Primer poletnega dne - Zalivanje



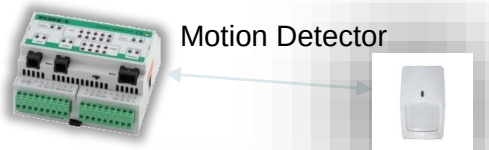
Vzorci gibanja - prisotnost

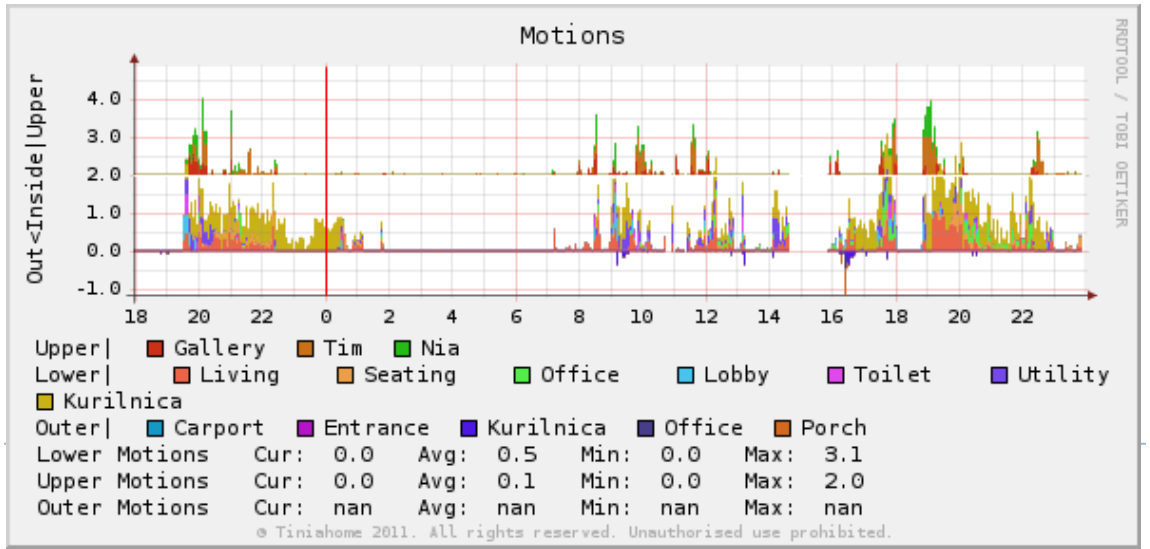
Detekcija gibanja/prisotnosti

- Detekcija gibanja v posameznih prostorih
- Informacija o prisotnosti

Uporaba vzorcev gibanja oziroma prisotnosti :

- Upravljanje :
  - Razsvetljave
  - Ogrevanja, hlajenja
  - A/V naprav
- Profiliranje, napovedi :
  - Dogodkov v prihodnosti
  - Energetskih potreb
  - Nastavitev

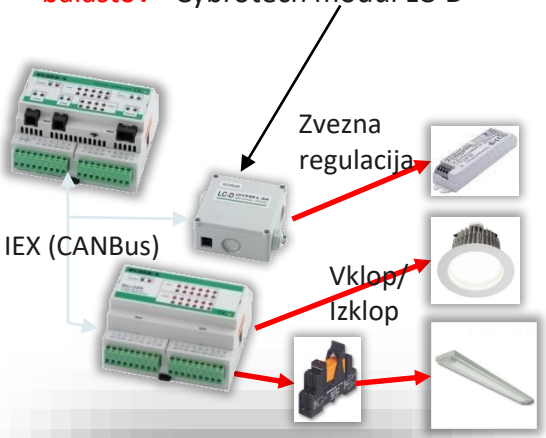




Razsvetljava

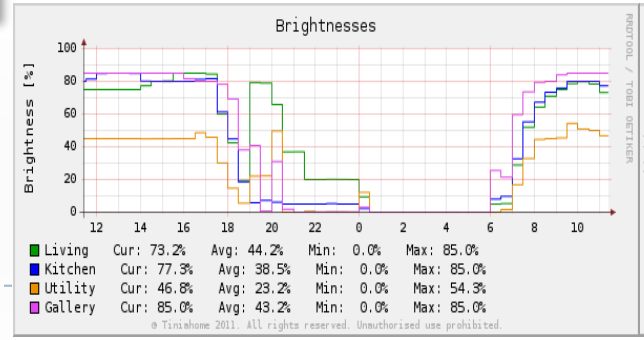
Vklop/Izklop in zvezna regulacija razsvetljave

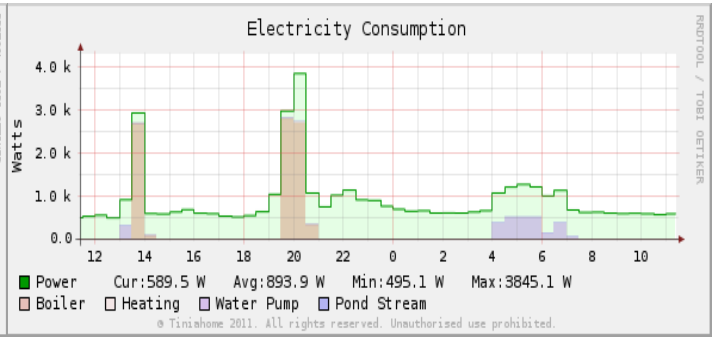
- **Vklop/Izklop** kontrola s Cybrotech moduli BIO-24R in BIO-24T, Zunanjimi releji
- **Zvezna regulacija** s pomočjo DALI balastov - Cybrotech modul LC-D



- Luči se upravljajo v skupinah
- Običajno krmiljena s pomočjo scen in zaznavanje osvetljenosti:
  - **Statične scene** – npr. : Prehrana,Obisk, Romantika,TV, Branje,Relaksacija, …
  - **Dogodkovne scene**: Ko se vklopi TV, nastavi bližnjo luč na 20%.
- Zmanjševanje porabe :
  - **Časovne luči** (izklopi po določenem času odsotnosti)
  - Vklopi luč samo, ko je to **res potrebno** (trenutna osvetljenost)
  - Nastavi zvezne luči samo na **potrebno stopnjo** (glede na osvetljenost)

Primer meritev osvetljenosti in nadzora porabe el. energije
(glavni porabnik el. Energije so posebej izpostavljeni)





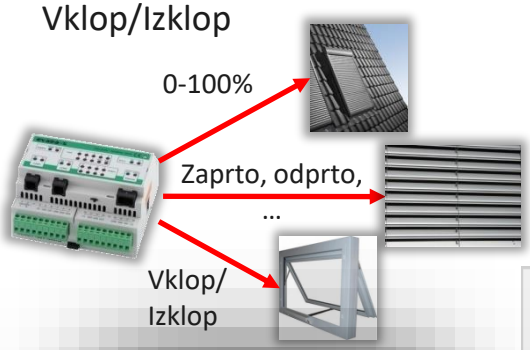
Rolete, Žaluzije, Okna

Pasivno ogrevanje/hlajenje…

- **Rolete**: med 0% - 100%
  (0% odprte, 100% zaprte)

- **Žaluzije** imajo stanja :
  Zaprto(100%), Senčeno(75%), Odprto(50%), Solarno pasivno (25%), Dvignjeno(0%).

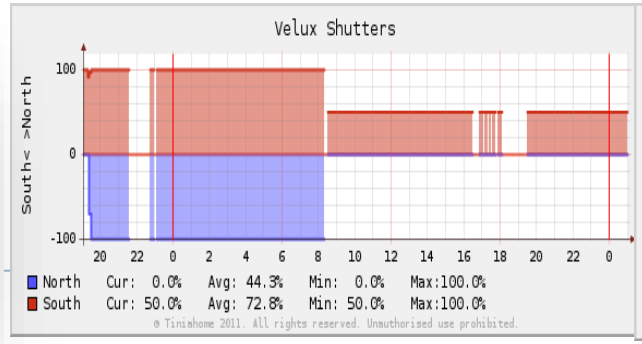
- **Motorizirana** okna:
  Vklop/Izklop

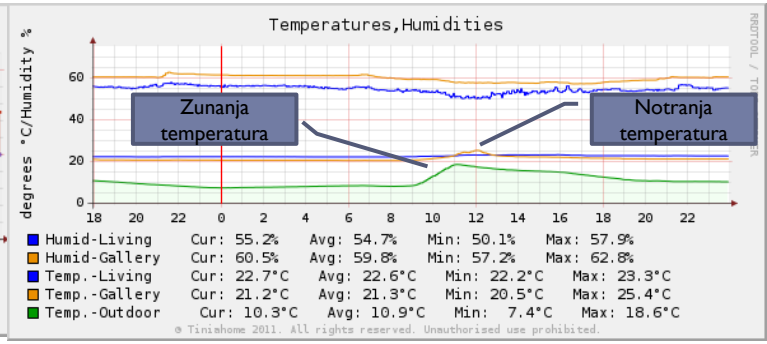


- Strešna okna z roletami :
  - Severna, običajno:
    - **Odprta v toplem vremenu** za boljšo osvetlitev
    - **Zaprta v hladnem vremenu** za ohranjanje toplote
  - Južna, običajno:
    - **Odprta v hladnem, sončnem vremenu** za pasivno ogrevanje
    - **Zaprta v vročem vremenu** proti pregrevanju
- Žaluzije:
  - **Senčene ali zaprte ob izrazitem sončnem vremenu poleti**
  - **Odprte v "solarni" poziciji ob sončnih dnevih pozimi**

- Motorizirana okna (s komarniki) :
  - **Odprta v poletnih nočeh za pasivno ohlajanje**

Primer stanj rolet in temperatur v sončnem zimskem dnevu:

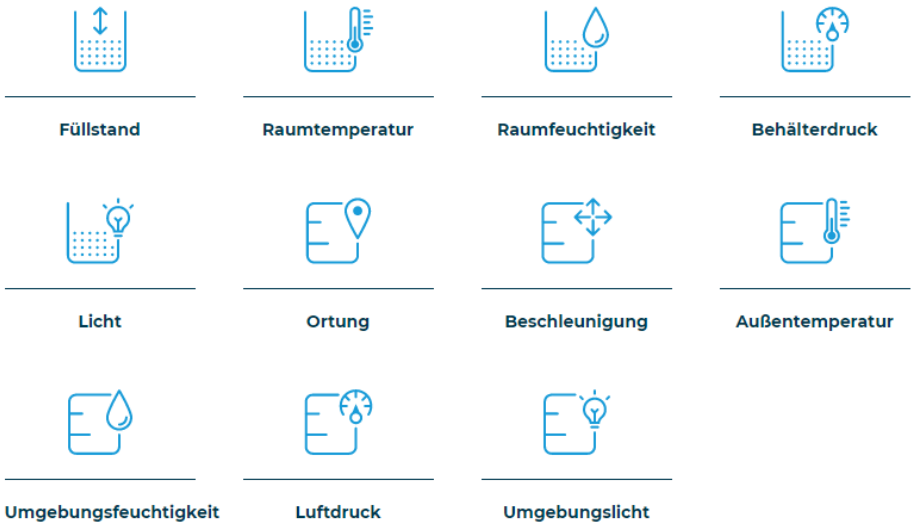




2. Programiranje vgrajenih sistemov - Primeri

E. Pametni zabojnik

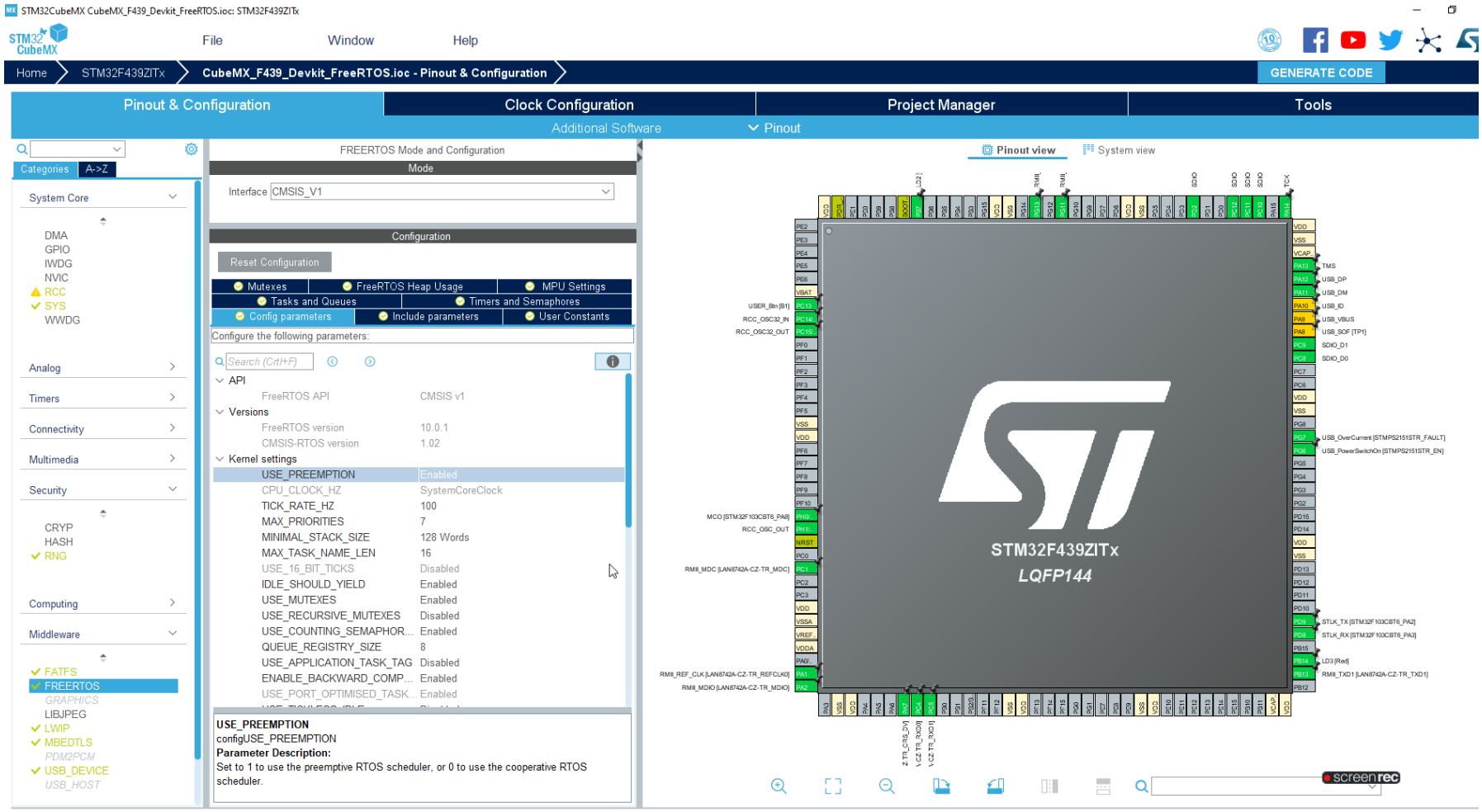
Merilnik tlaka, temperature, nivoja, pozicije, … za pametne zabojnike



2. Programiranje vgrajenih sistemov - Primeri

E. Pametni zabojnik

Merilnik tlaka, temperature, nivoja, pozicije, … za pametne zabojnike



2. Programiranje vgrajenih sistemov - Primeri

F. Embedded Linux (UcLinux, Buildroot)

Buildroot na STM32F769

bootlin

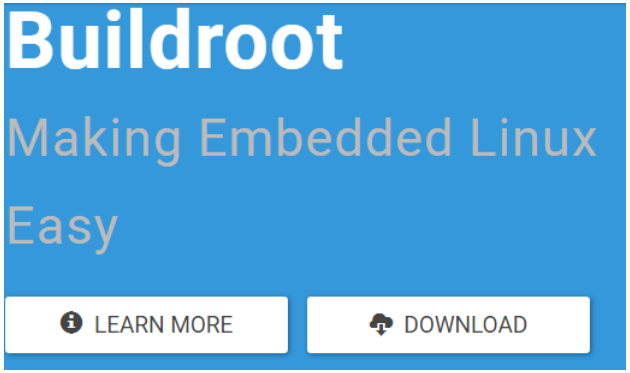
- Understanding the Linux graphics stack
- Real-time Linux with PREEMPT_RT

https://bootlin.com/

https://www.emcraft.com/

yocto PROJECT

THE YOCTO PROJECT. IT'S NOT AN EMBEDDED LINUX DISTRIBUTION, IT CREATES A CUSTOM ONE FOR YOU.



2. Programiranje vgrajenih sistemov - Primeri

F. Embedded Linux (UcLinux, Buildroot)

Buildroot na STM32F769

```
# ------------------------------------------------------------------------
U-Boot SPL 2020.04 (Oct 18 2020 - 20:10:10 +0200)
# ------------------------------------------------------------------------
Trying to boot from XIP


U-Boot 2020.04 (Oct 18 2020 - 20:10:10 +0200)

Model: STMicroelectronics STM32F769-DISCO board
DRAM:  16 MiB
set_rate not implemented for clock index 4
set_rate not implemented for clock index 4
set_rate not implemented for clock index 4
Flash: 1 MiB
MMC:   sdio2@40011c00: 0
In:    serial
Out:   serial
Err:   serial
usr button is at LOW LEVEL
Net:
Warning: ethernet@40028000 (eth0) using random MAC
eth0: ethernet@40028000
Hit SPACE in 1 seconds to stop autoboot.
```

```
# ------------------------------------------------------------------------
Starting kernel ...
# ------------------------------------------------------------------------

Booting Linux on physical CPU 0x0
Linux version 5.6.15 (robi@Linux) (gcc version 8.4.0 (Buildroot 2020.05)) #1 PREEM
CPU: ARMv7-M [411fc270] revision 0 (ARMv7M), cr=00000000
CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
OF: fdt: Machine model: STMicroelectronics STM32F769-DISCO board
Reserved memory: created DMA memory pool at 0xc0ef0000, size 1 MiB
OF: reserved mem: initialized node linux,dma, compatible id shared-dma-pool
Using ARMv7 PMSA Compliant MPU. Region independence: No, Used 6 of 8 regions
Built 1 zonelists, mobility grouping off.  Total pages: 3794
Kernel command line: root=/dev/mmcblk0p1 rootwait rw
Dentry cache hash table entries: 2048 (order: 1, 8192 bytes, linear)
Inode-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
mem auto-init: stack:off, heap alloc:off, heap free:off
Memory: 12240K/15296K available (1924K kernel code, 166K rwdata, 384K rodata, 84K init, 115K bss, 3056K reserved,
0K cma-reserved)
```

Differences between UcLinux vs Linux

Maybe we can have session to explain what is the biggest difference...

Under MMU, every program runs in own address space, independently of others, it can also ask for more memory any time during execution...

Under UcLinux this is not the case, you have linear memory space for all apps, including Kernel... If one app goes wrong, it can affect others, since memory space is the same... No protection at all for this situation... This is the biggest difference.... And also for UcLinux, at least 32MB of RAM is recomended, at least to start with...

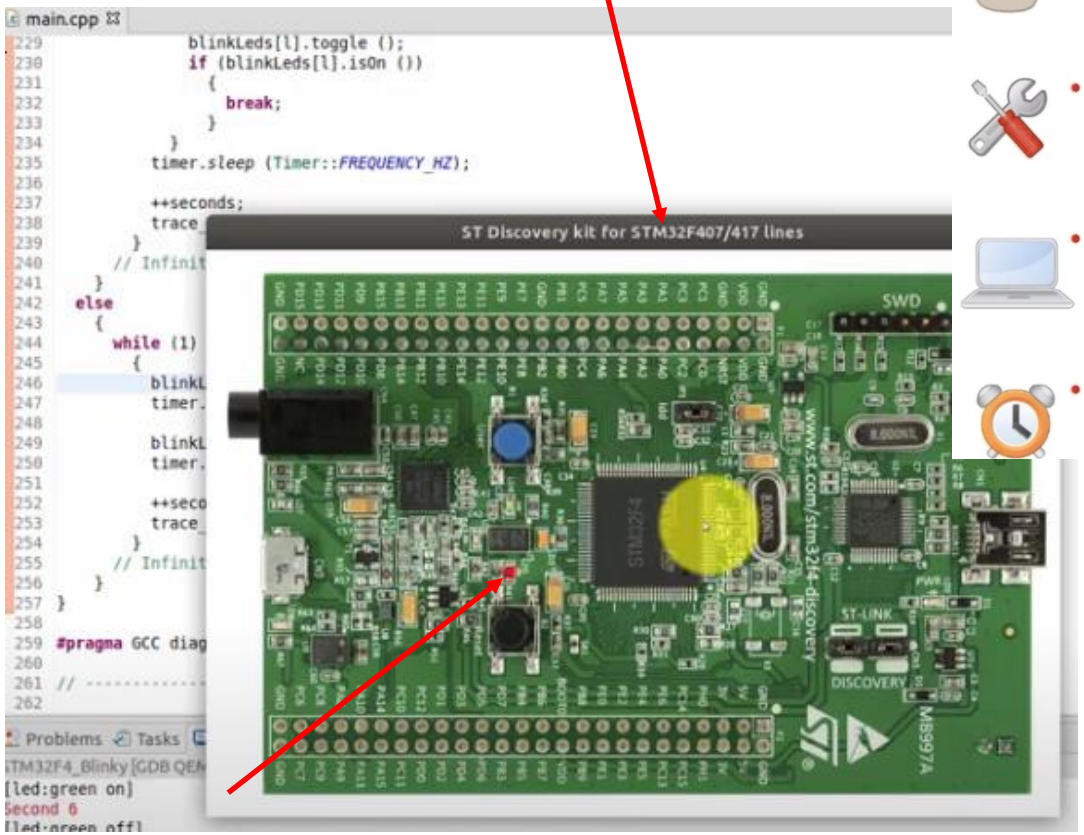
But I also see some real advantages of using UcLinux already...

Because SW looks quite the same, we can use tools that are already existing... We can develope on Linux and transfer.... So SW very much looks like real Linux SW. And that is the big advantage of this path... So I still think that it is a good way... But we need to also have practical experience, when system will be actually running to confirm all this...

G. Simulacije, Emulacije

QEMU – STM32 Discovery

https://www.qemu.org/



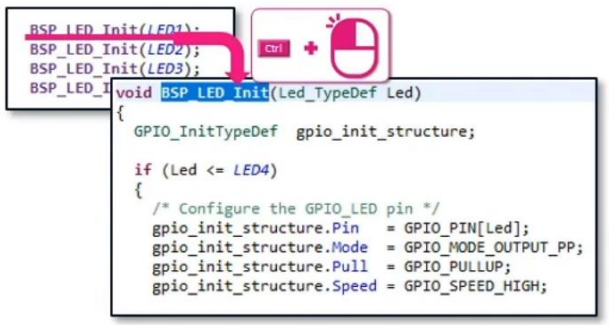
Why use QEMU?

QEMU
A generic and open source machine emulator and virtualizer

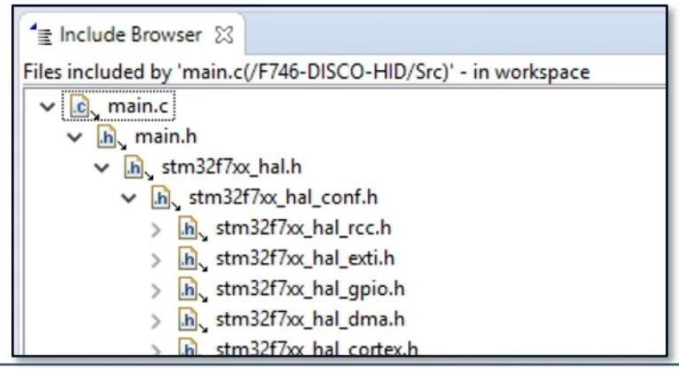
- Cost
  - free and open source software (GPLv2)
  - no development kit required
- Experiment without fear
  - Minimize the risk of corrupting valuable development boards
- Portability
  - Not tied to a lab bench --> only need QEMU and a laptop
- Reduce project timescales
  - work in advance of prototype board or silicon delivery

Razvoj in razhroščevanje (primer CubeIDE)

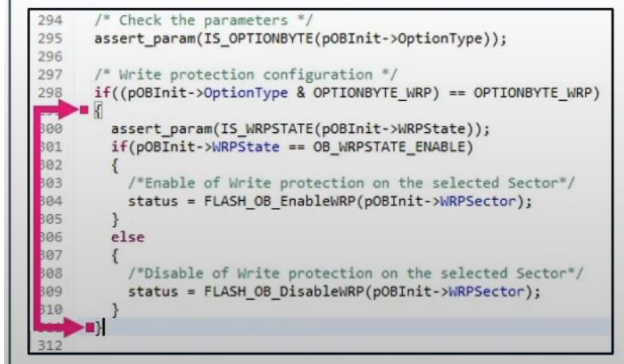
Symbol Hyperlink



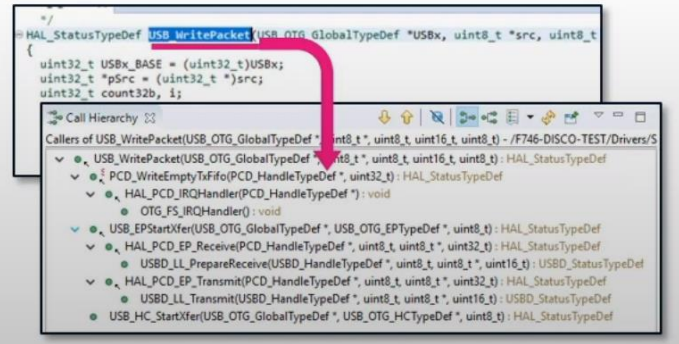
Include Browser



Brace Navigation



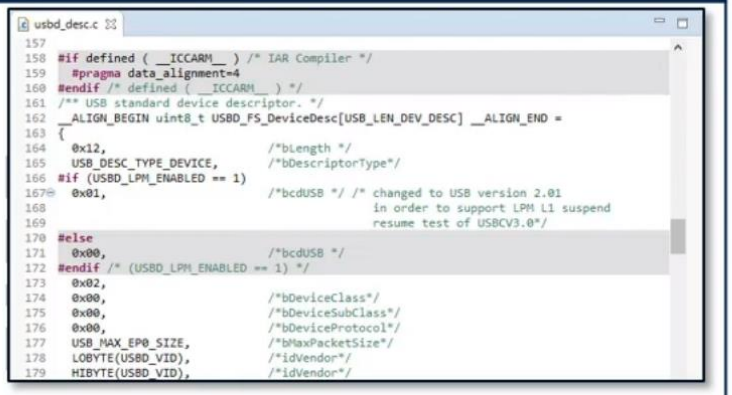
Call Hierarchy



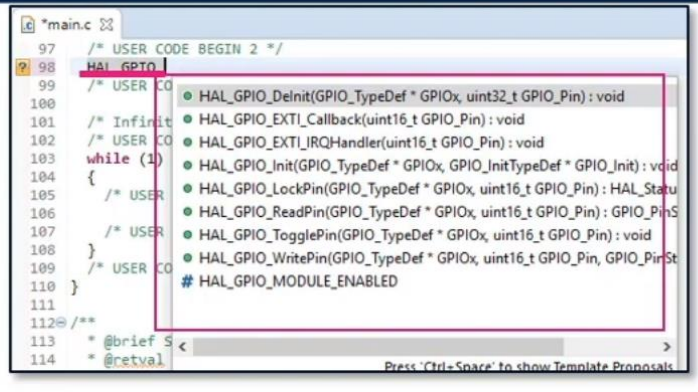
2. Programiranje vgrajenih sistemov

Razvoj in razhroščevanje (primer CubeIDE)

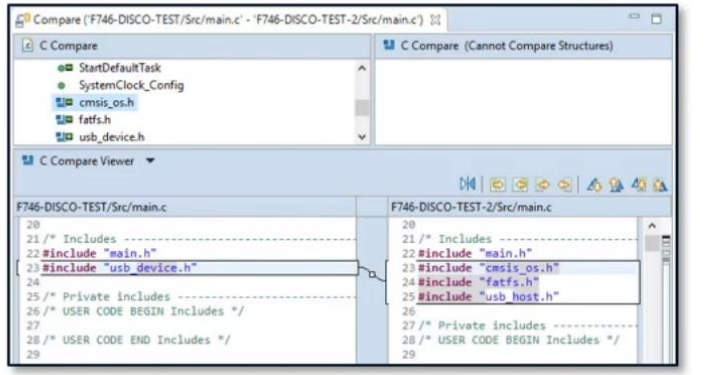
Highlight Inactive Code



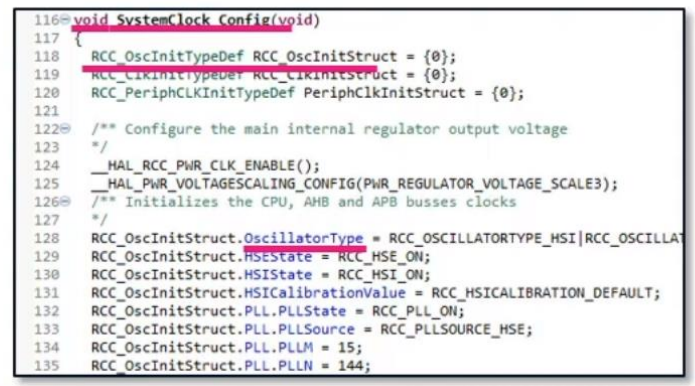
Auto-Complete



File Diff/Compare

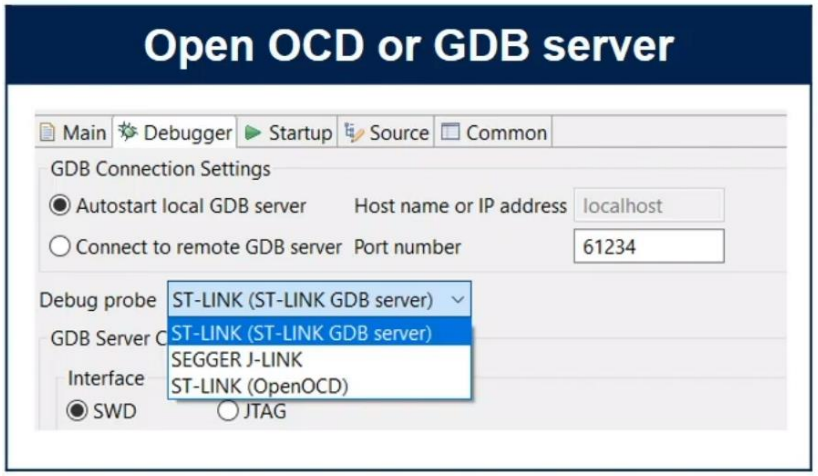


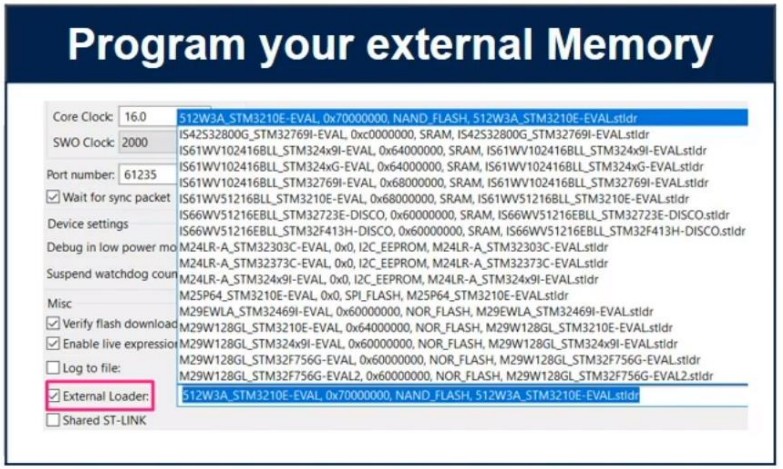
Syntax Highlight

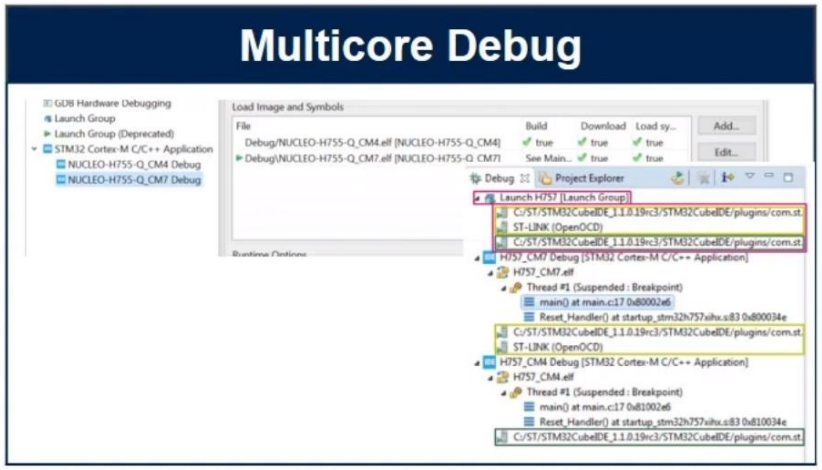


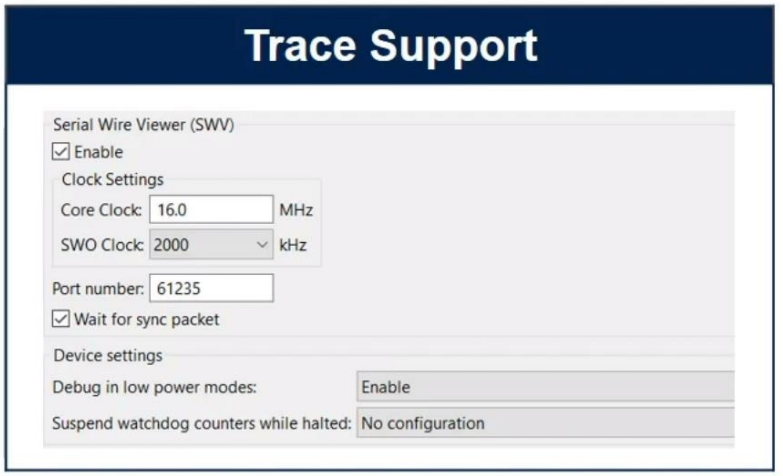
2. Programiranje vgrajenih sistemov

Razvoj in razhroščevanje (primer CubeIDE)







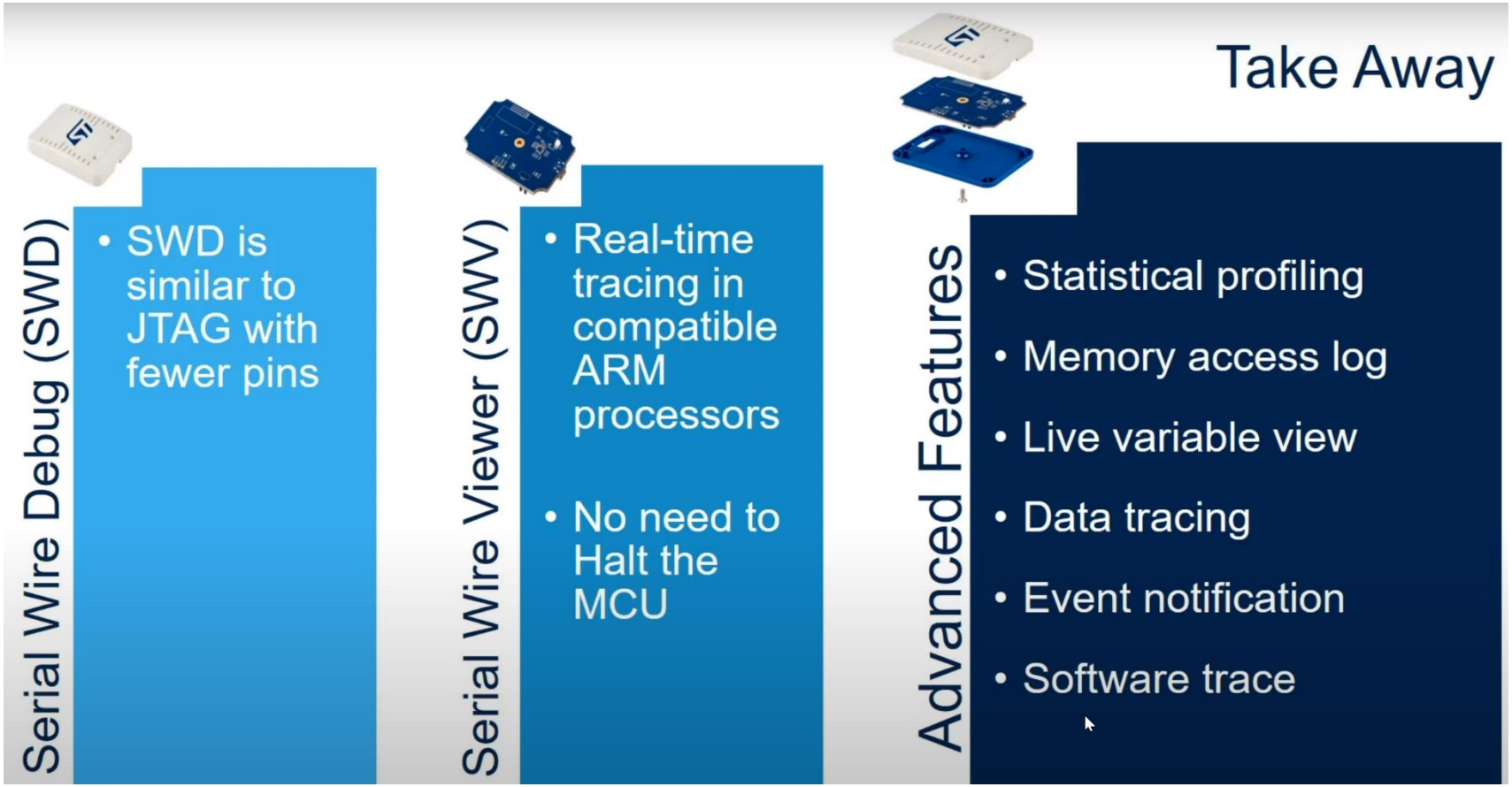


Razvoj in razhroščevanje (primer CubeIDE)




Core	Cortex M0 / M0+	Cortex M3 / M4	Cortex M33	Cortex M7
Debug	SWD*	JTAG / SWD	JTAG / SWD	JTAG / SWD
Trace	No	Trace port Serial Wire Viewer	Trace port Serial Wire Viewer	Trace port Serial Wire Viewer

Razvoj in razhroščevanje (primer CubeIDE)



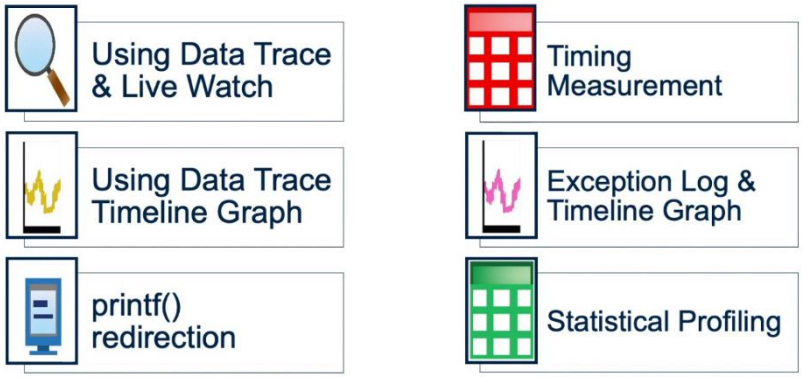
2. Programiranje vgrajenih sistemov

Razvoj in razhroščevanje (primer CubeIDE)

2. Programiranje vgrajenih sistemov

Razvoj in razhroščevanje (primer CubeIDE)

Razvoj in razhroščevanje (primer CubeIDE)



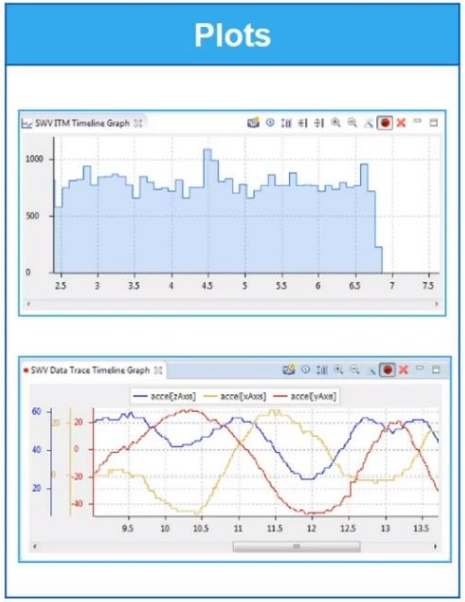
- SWV adds:

Real Time Trace

- That uses the SWD port and the SWO pin.

Advanced Debugging

- Without Halting the MCU



2. Programiranje vgrajenih sistemov

Razvoj in razhroščevanje (primer CubeIDE)

2. Programiranje vgrajenih sistemov

Razvoj in razhroščevanje (primer CubeIDE)

2. Programiranje vgrajenih sistemov

Kaj po koncu razvoja ?

- Dokumentacija (!*?)

- Spremljanje delovanja (kurativa) :
  - serijska konzola
  - log datoteke, sporočanje napak, daljinski nadzor

spremljanje

  - primer kritične napake:

```
2015-01-11 05:45:37 CRIT   232      0 FP      WDT has expired
2015-01-11 05:46:21 CRIT   232      0 MNG     WDT has expired
```

  - primer pomembne napake, ki zahteva popravke v kodi :

```
2015-01-09 15:00:02 INFO    60      0 CMDEXECUTE  CMD:Execute Cmd[72]
2015-01-09 15:00:02 INFO    60      0 CMDEXECUTE CMD:SendSett
2015-01-09 15:04:02 CRIT   232      0 CMDEXECUTE  WDT has expired
```

3. Nivoji programiranja – jeziki, knjižnice

Baremetal - zbirnik

```
INIT_IO:
  push {r5, r6, lr}
    // Enable GPIOD Peripheral Clock (bit 3 in AHB1ENR register)
    ldr r6, = RCC_AHB1ENR       // Load peripheral clock reg address to r6
    ldr r5, [r6]                // Read its content to r5
    orr r5, 0x00000008          // Set bit 3 to enable GPIOD clock
    str r5, [r6]                // Store result in peripheral clock register

    // Make GPIOD Pin12 as output pin (bits 25:24 in MODER register)
    ldr r6, =GPIOD_BASE       // Load GPIOD BASE address to r6
    ldr r5, [r6,#GPIOD_MODER]  // Read GPIOD_MODER content to r5
    and r5, 0x00FFFFFF          // Clear bits 31-24 for P12-15
    orr r5, 0x55000000          // Write 01 to bits 31-24 for P12-15
    str r5, [r6]                // Store result in GPIOD MODER register
  pop {r5, r6, pc}


LED_ON:
    push {r5, r6, lr}
    // Set GPIOD Pins to 1 (through BSSR register)
    ldr     r6, =GPIOD_BASE       // Load GPIOD BASE address to r6
    mov     r5, #LEDs_ON
    str     r5, [r6,#GPIOD_BSSR] // Write to BSRR register
    pop {r5, r6, pc}

LED_OFF:
    push {r5, r6, lr}
    // Set GPIOD Pins to 0 (through BSSR register)
    ldr     r6, =GPIOD_BASE       // Load GPIOD BASE address to r6
    mov     r5, #LEDs_OFF
    str     r5, [r6,#GPIOD_BSSR] // Write to BSRR register
    pop {r5, r6, pc}
```

https://github.com/LAPSyLAB/ORLab-STM32/tree/main/GPIO_LEDs

Baremetal - C

```
  /* USER CODE BEGIN 2 */


  RCC->AHB1ENR |= 0x08;
// Enable clock for GPIOD
  GPIOD->MODER |= 0x01000000;         //
MODE Register: bit 12 == out

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    GPIOD->ODR ^= 0x1000;             //
Toggle PD12

/* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    for (int i=0; i<0x1000000; i++) {};
// waste some time
  }
  /* USER CODE END 3 */
```

https://github.com/LAPSyLAB/STM32F4_Discovery_VIN_Projects/tree/main/LED_GPIO_C_Baremetal_C

HAL - C

```
/* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {

HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);

  /* USER CODE END WHILE */

   /* USER CODE BEGIN 3 */
 HAL_Delay(1000);
 }
 /* USER CODE END 3 */
```

```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx,
uint16_t GPIO_Pin)
{
  uint32_t odr;

  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  /* get current Ouput Data Register value
*/
  odr = GPIOx->ODR;

  /* Set selected pins that were at low
level, and reset ones that were high */
  GPIOx->BSRR = ((odr & GPIO_Pin) <<
GPIO_NUMBER) | (~odr & GPIO_Pin);
}
```

https://github.com/LAPSyLAB/STM32F4_Discovery_VIN_Projects/tree/main/LED_Blink_Demo

3. Nivoji programiranja – koncepti

Ena zanka

```
{ …
      if (Timer_1sec)   {
                readSensors(&data);
                send_data(&data);
                 Timer_1sec = 0;
      }

      if (Timer_50msec)   {
                readKeys(&keys);
                readInputs(&inputs);
                Timer_50msec = 0;
      }
}
```

Končni avtomat

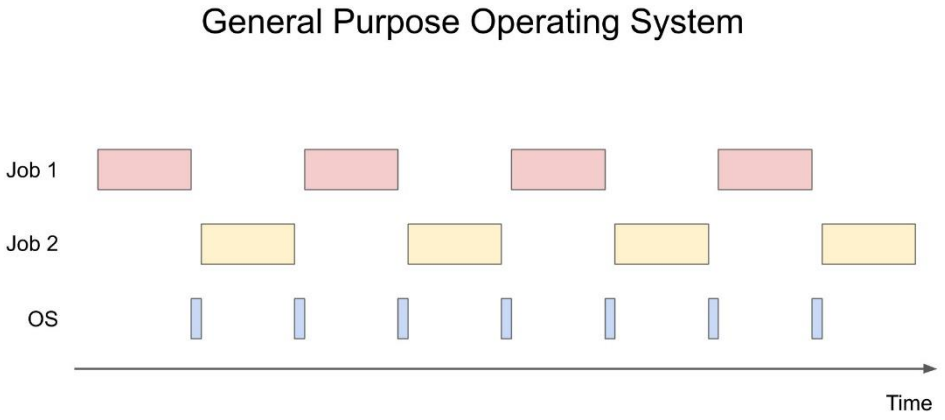
```
switch (FSM.State) {
      case CHECK_REASON:
         ///<  FSM.State: after reset.

          if  VSE_OK then
             FSM.State  = CHECK_BAUDRATE

             break;

      case CHECK_BAUDRATE:
             ///<  FSM.State: after reset.
               …

             break;
```

RTOS

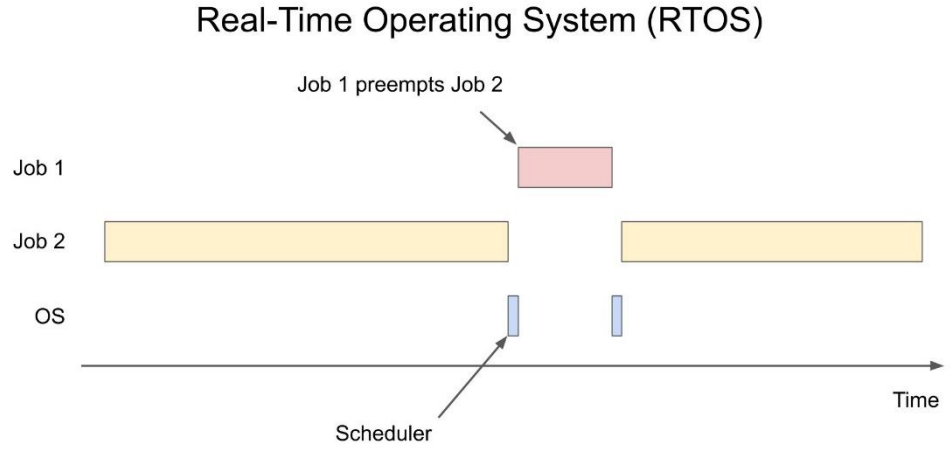
```
void StartTask02(void *argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  for(;;)
  {
   HAL_GPIO_TogglePin(GPIOD,
GPIO_PIN_13);
    osDelay(1000);
  }
  /* USER CODE END StartTask02 */
}

void StartTask01(void *argument)
{
  /* USER CODE BEGIN StartTask01 */
  /* Infinite loop */
  for(;;)
  {
HAL_GPIO_TogglePin(GPIOD,
GPIO_PIN_12);

    osDelay(1000);

  }
  /* USER CODE END StartTask01 */
}
```

4. Programiranje vgrajenih sistemov – OS, RTOS

OS



RTOS



4.1 Splošno o RTOS - Real Time Operating System

- RTOS upravlja čas in procese na mikroprocesorju ali mikrokrmilniku
  - v bistvu: „poenostavljen operacijski sistem“

- Funkcionalnosti RTOS:
  - Večopravilnost (multi-tasking)
  - Dodeljevanje opravil CPE s prioritetami
  - Sinhronizacija dostopov do virov:
    - V/I naprav
    - Pomnilnika (podatkovnih struktur)
  - Komunikacija med procesi (Inter-task communication)
  - Časovna predvidljivost (realno-časna odzivnost)
  - Servisiranje prekinitev

Zakaj uporabiti RTOS?

- Uporaba V/I naprav (že pripravljeni driverji (TCP, ETH, CANBUS,…)
- Se splača vse razviti iz nič (npr. svoj dodeljevalnik) ?
  - Diploma : Fabčič – 2021 – lasten RTOS „from scratch“
- Večopravilnost z možnostjo sinhronizacije
- Prenosljivost kode na druge CPE
- Upravljanje z viri
- Možnost dopolnitve z lastnimi funkcijami
- Obstoječa podpora za nekatere razširjene protokole:
  - TCP/IP, USB, Flash Systems, Web Servers,
  - CAN protocols, GUI, SSL, SNMP

Nekatere prednosti se hitro sprevržejo v težave in dodatno delo…

RTOS - Opravila

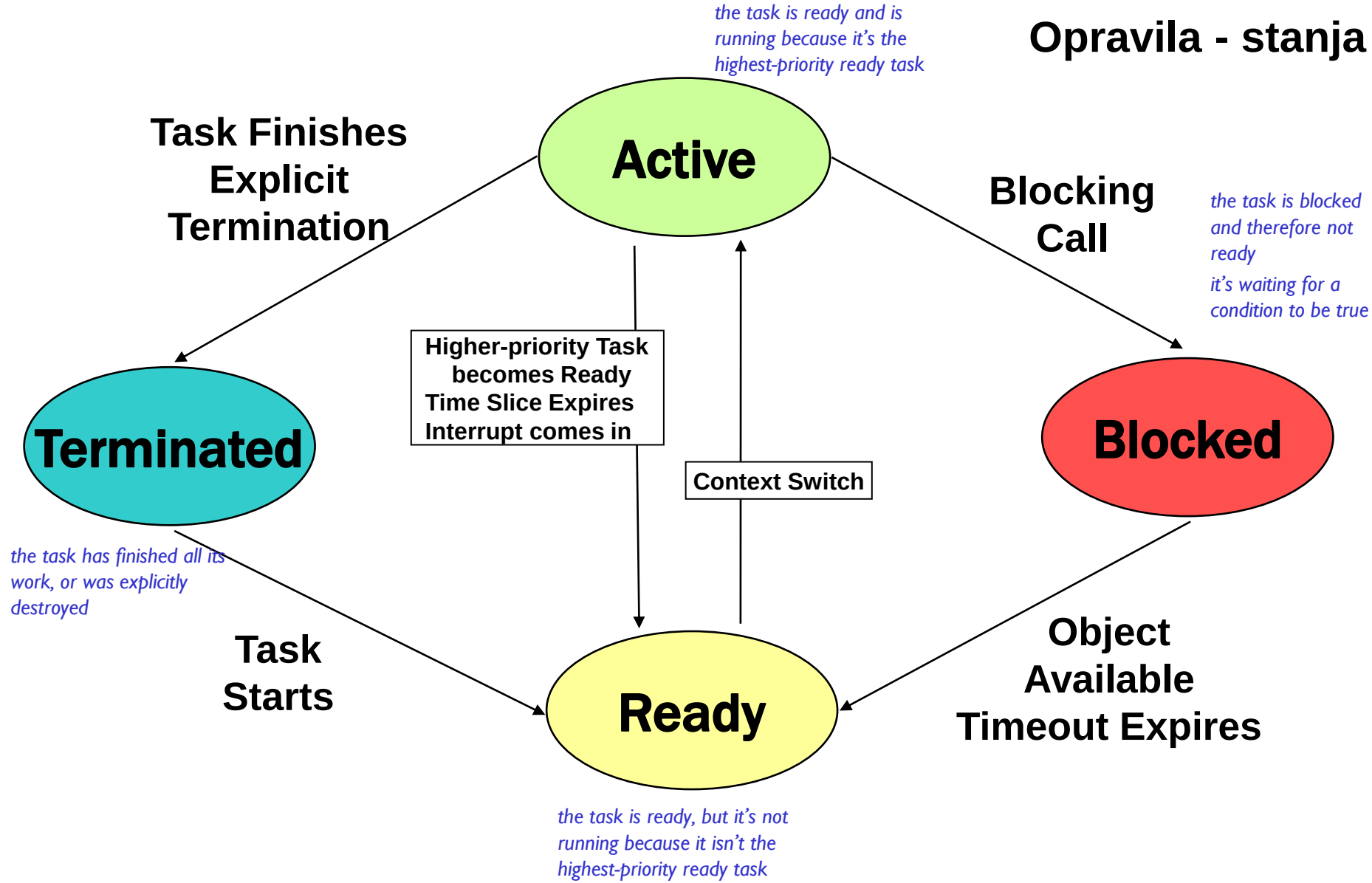
- Sistem oz. aplikacija je sestavljena iz več opravil
- Opravila se izmenjaje izvajajo
- V nekem trenutku je aktivno natanko eno opravilo (se izvaja na procesorju)
- RTOS odloča, kako si opravila delijo procesor („context switching“)
- Vsebina opravila („Task Context“)
  - Podatkovna struktura lastna vsakemu opravilu:
    - Vsebuje vse potrebne podatke za izvedbo opravila:
      - npr. spremenljivke, registre in sezname vseh uporabljenih virov

Tipična struktura kode opravila

```
void mytask(uint_32 startup_parameter)  {
   /* Task initialization code */
   ....
   while (1) {
      /* Task body */
      ....
      ....
   }

}
```

```
void StartTask02(void *argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  for(;;)
  {
   HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_13);
    osDelay(1000);
  }
  /* USER CODE END StartTask02 */
}

void StartTask01(void *argument)
{
  /* USER CODE BEGIN StartTask01 */
  /* Infinite loop */
  for(;;)
  {
HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);

    osDelay(1000);

  }
  /* USER CODE END StartTask01 */
}
```

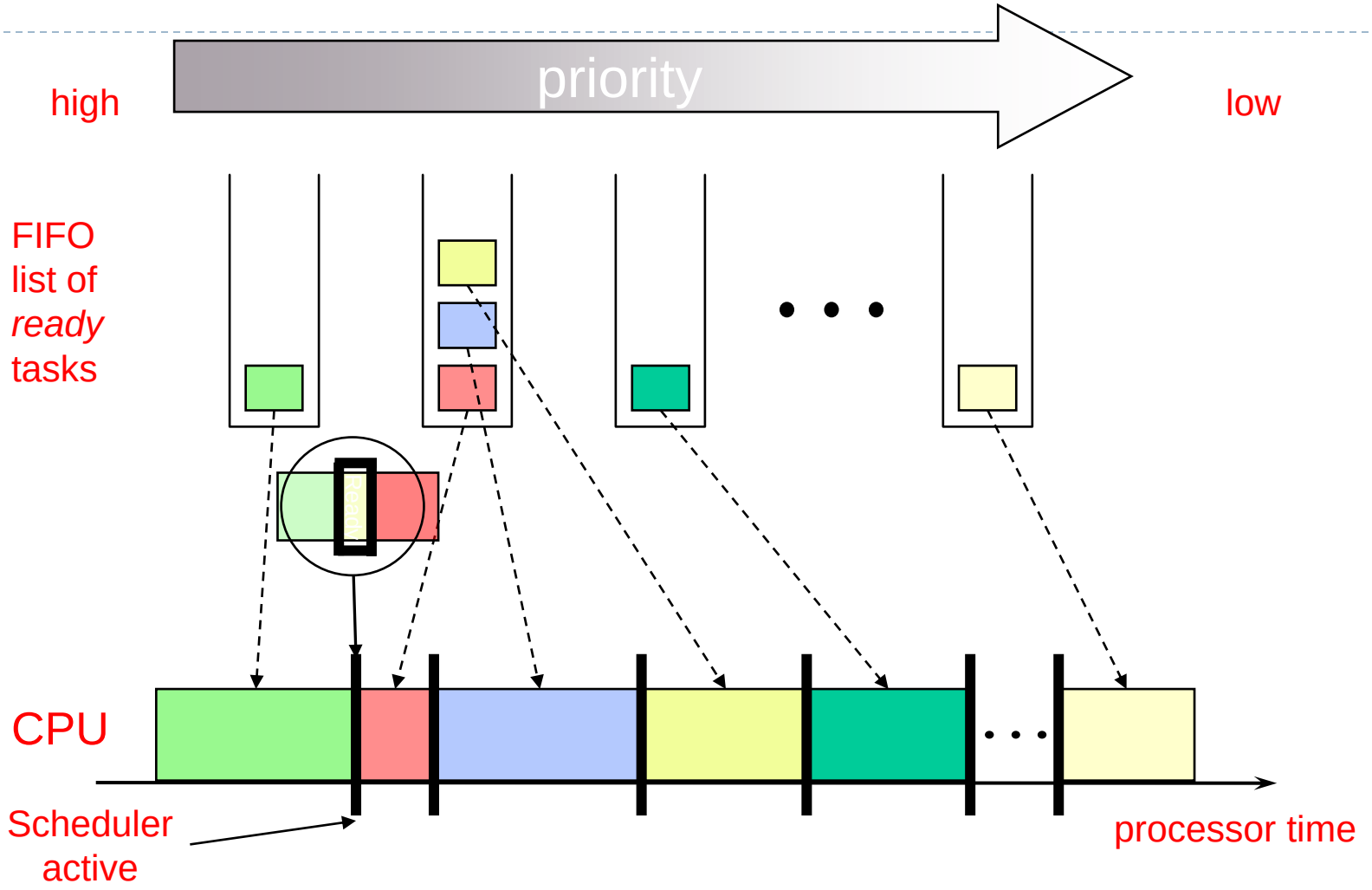
Opravila - stanja



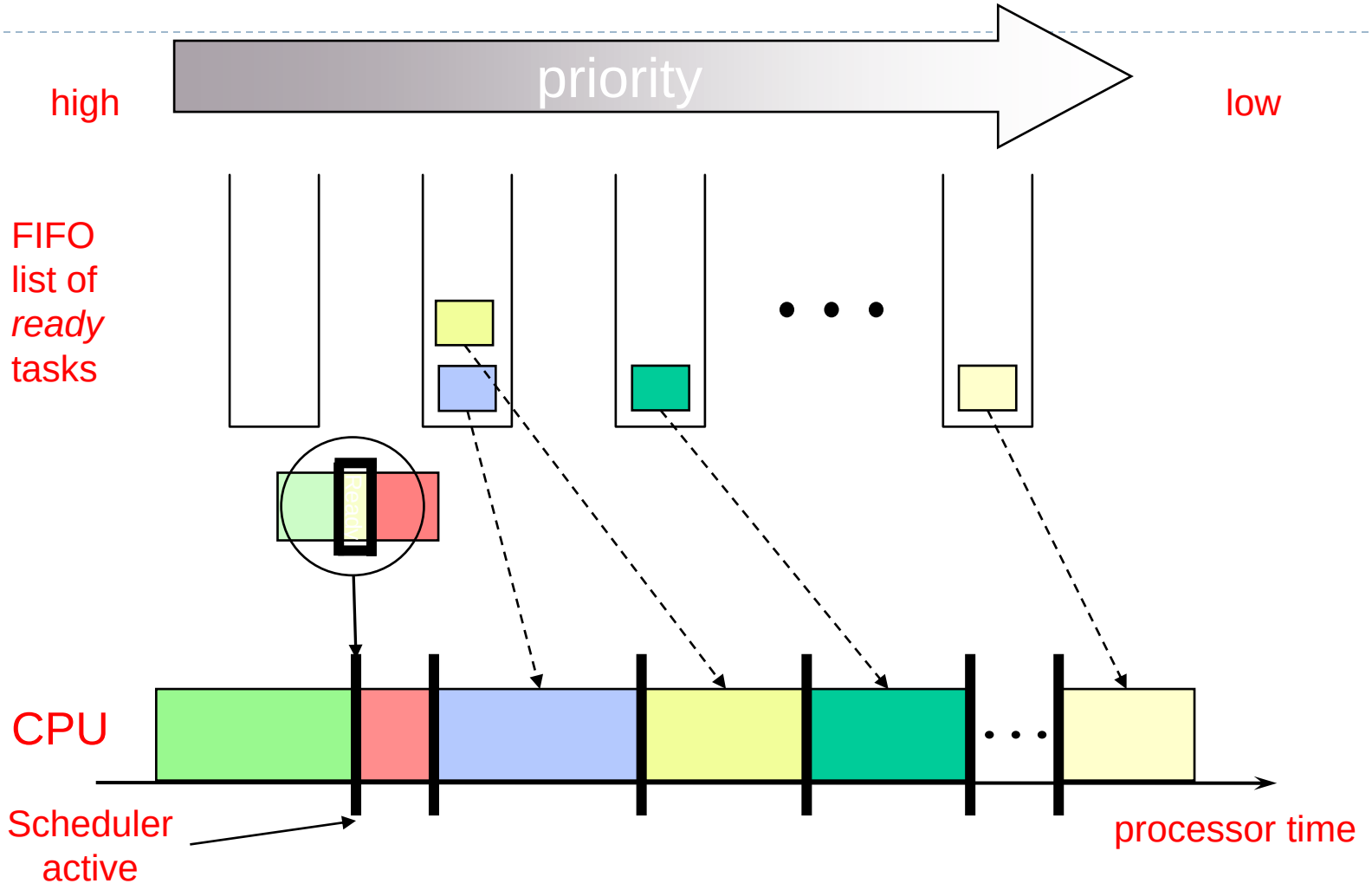
Dodeljevalnik („Scheduler“)

- **Običajni načini dodeljevanja:**
  - FIFO (tudi „priority-based preemptive“)
    - Aktivni je tisti z najvišjo prioriteto, ki je pripravljen najdlje časa
  - Round Robin
    - Aktivni je tisti z najvišjo prioriteto, ki je najdlje časa brez dodelitve procesorju

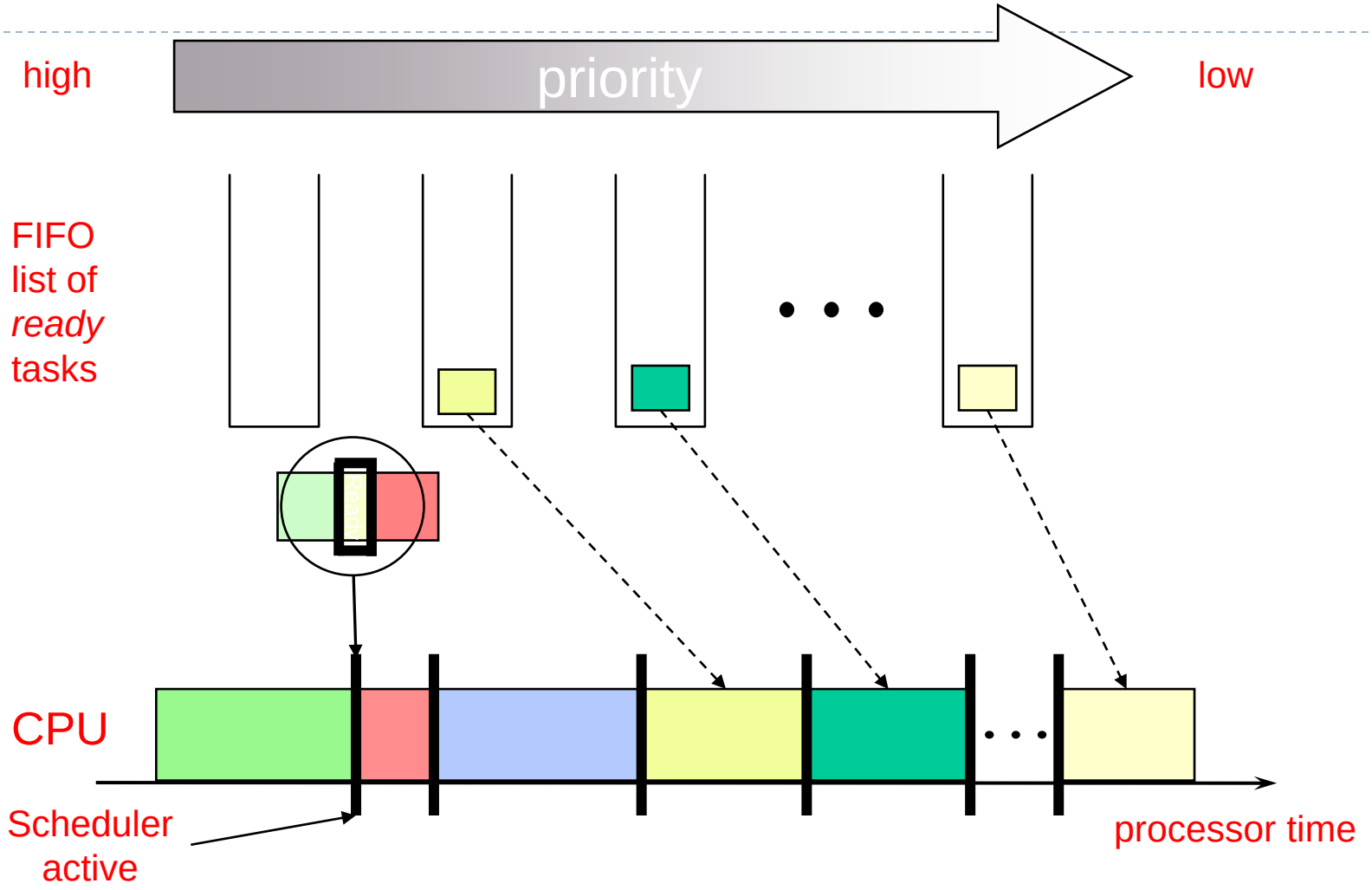
Priority Based FIFO Scheduling

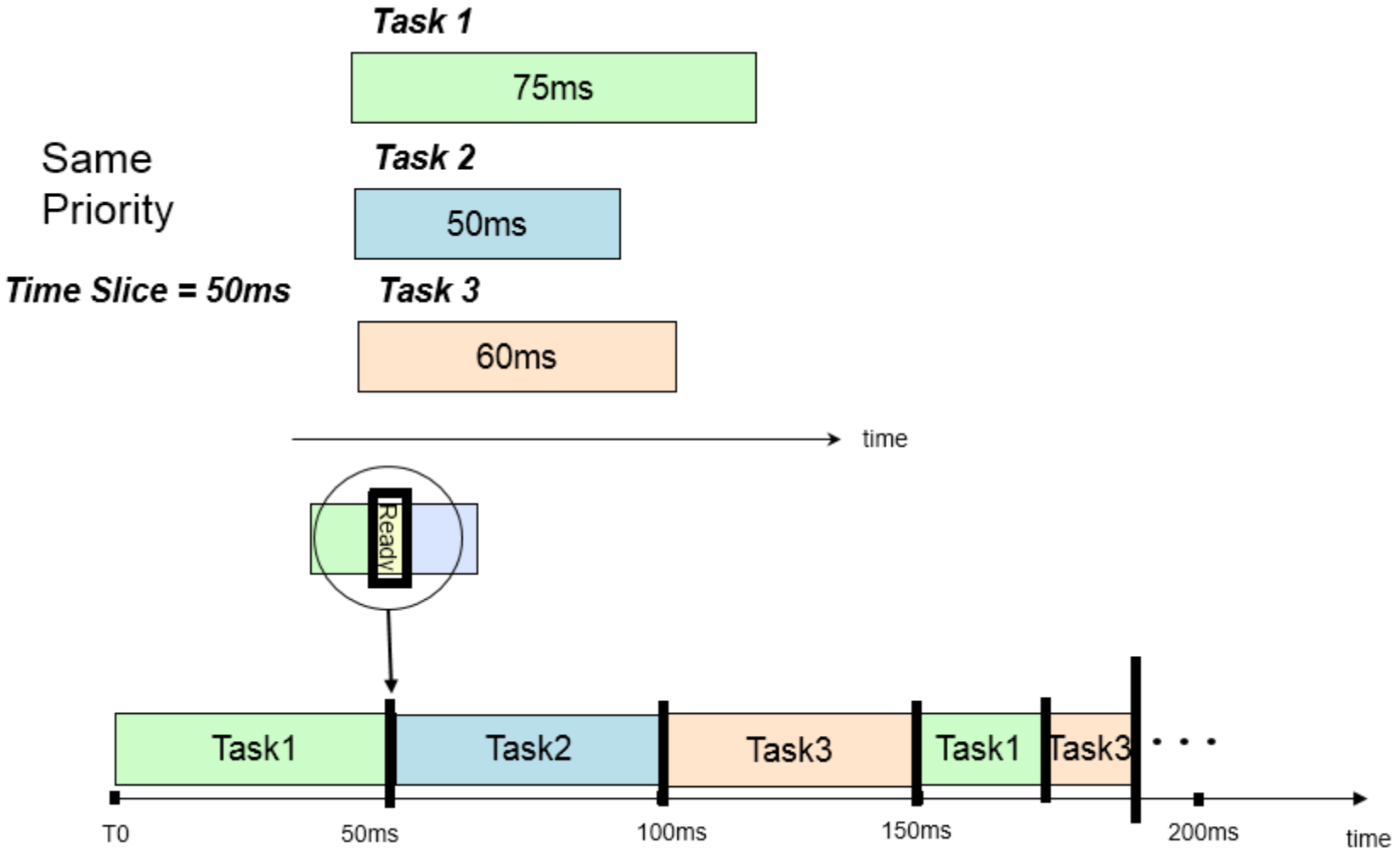


Priority Based FIFO Scheduling



Priority Based FIFO Scheduling



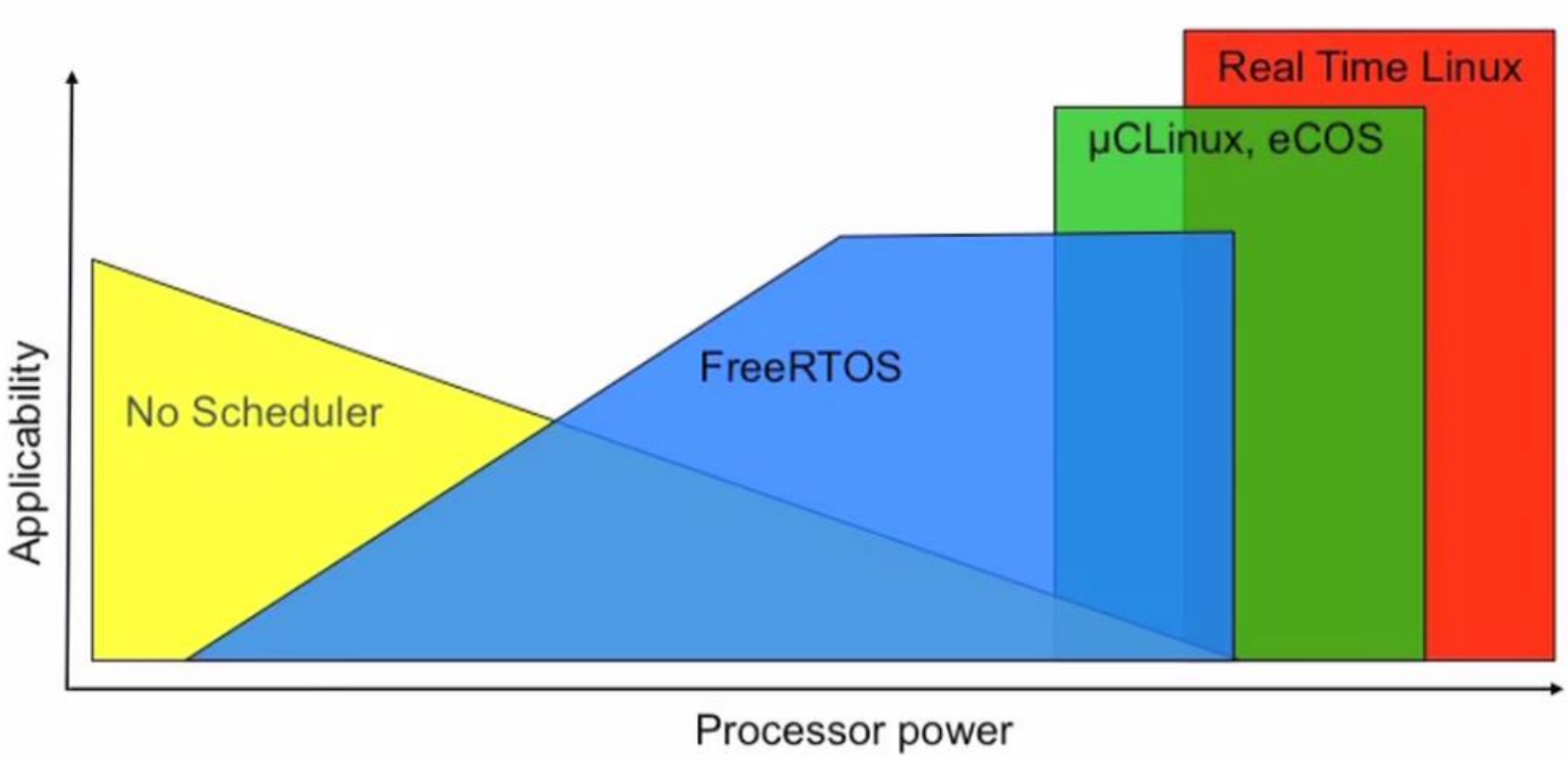

Task 1
75ms
Same
Priority
Task 2
50ms
Time Slice = 50ms
Task 3
60ms
time
Ready
Task1
Task2
Task3
Task1
Task3
T0
50ms
100ms
150ms
200ms
time

4.2. FreeRTOS (primer) :

- A Real Time Operating System
- Written by Richard Barry & FreeRTOS Team

- Huge number of users all over the world
  - 6000 Download per month
- Simple but very powerful

Kdaj uporabiti FreeRTOS ?



4.2 FreeRTOS (primer) :

Opravila („Tasks“) - Primer

```
/**
* @brief Function implementing the ShellTask thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_Shell_Entry */
void Shell_Entry(void const * argument)
{
  /* USER CODE BEGIN Shell_Entry */
        printf_dma ("\r\nShell Task started.\r\n");
        if ( HAL_UART_Receive_IT(&huart3, &(UARTRxBuffer[Var.Uart.RxBufferInd]), 1) != HAL_OK) {
            Error_Handler();
         }

        shell_cmd_init(); ///< Init command shell

        /* Infinite loop */
        for(;;)
        {
            shell_cmd_check_rx();     ///< check if shell character received
            osDelay(100);
            ShelluxHighWaterMark = uxTaskGetStackHighWaterMark( NULL );
        }
  /* USER CODE END Shell_Entry */
}
```

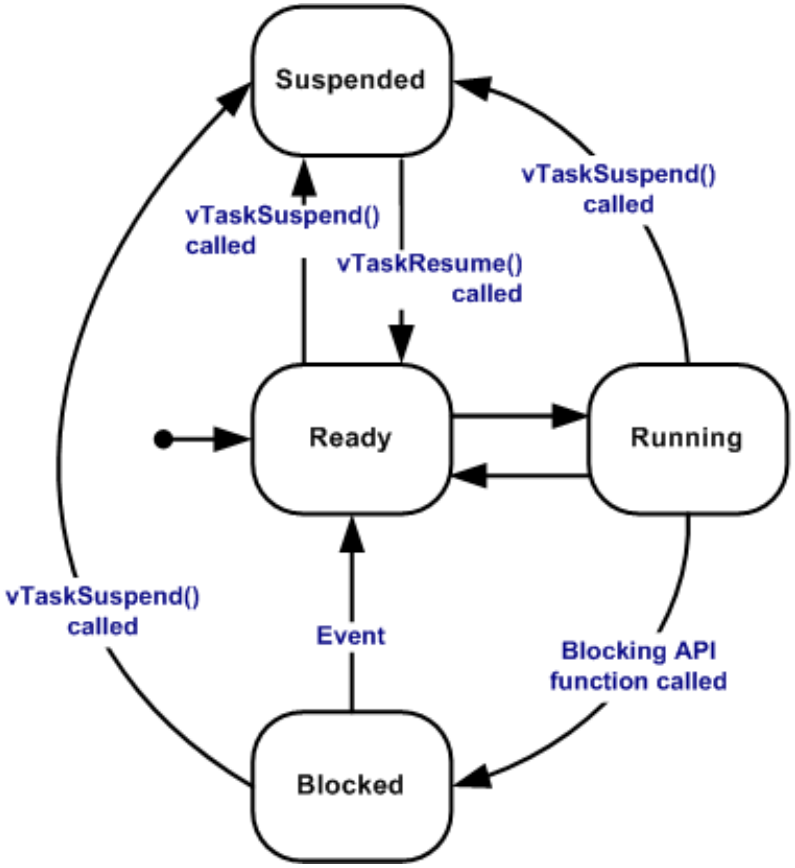
4.2. FreeRTOS (primer) :

Architecture Overview

- Tasks
  - task.c , task.h
    - creating, scheduling, and maintaining tasks.
- Communication
  - queue.c and queue.h handle communication. Tasks and interrupts use queues to
    - send data to each other and
    - to signal the use of critical resources using semaphores and mutexes.
- Hardware Interfacing

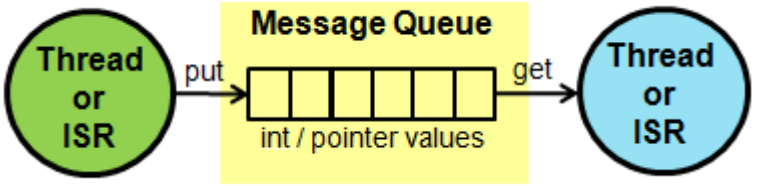
Stanje procesov

- Running
- Ready
- Blocked
- Suspended

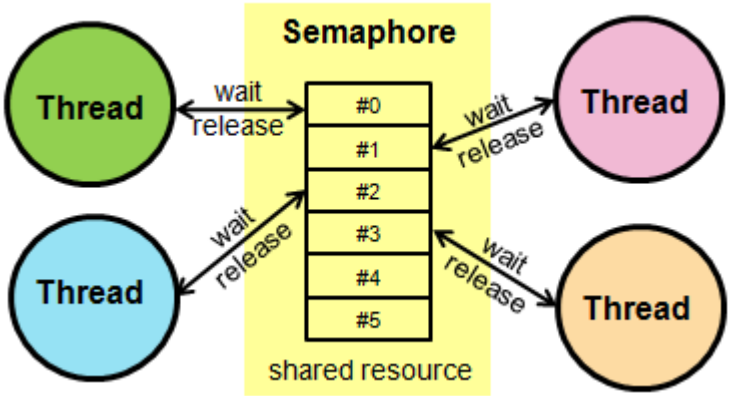


__RTOS : Komunikacija in sinhronizacija med procesi

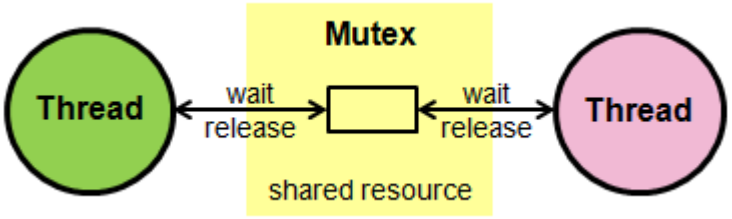
- Queues



- Binary Semaphores
- Counting Semaphores



- Mutexes
- Recursive Mutexes

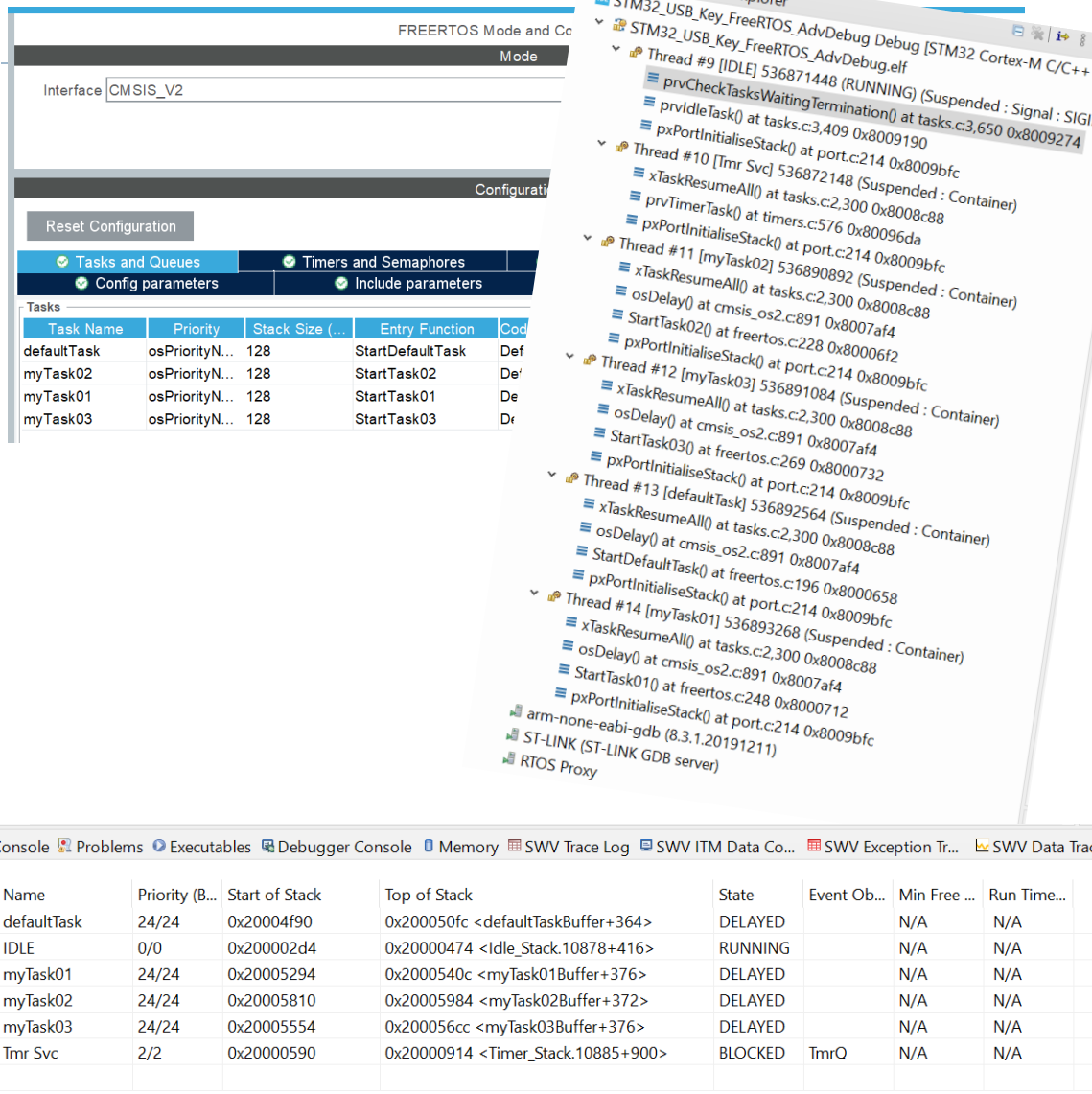


FreeRTOS uradna spletna stran:
http://www.freertos.org/

4.2. FreeRTOS (primer) : Utripanje LED diode (vsaka v svojem procesu)

Opravila („Tasks“)

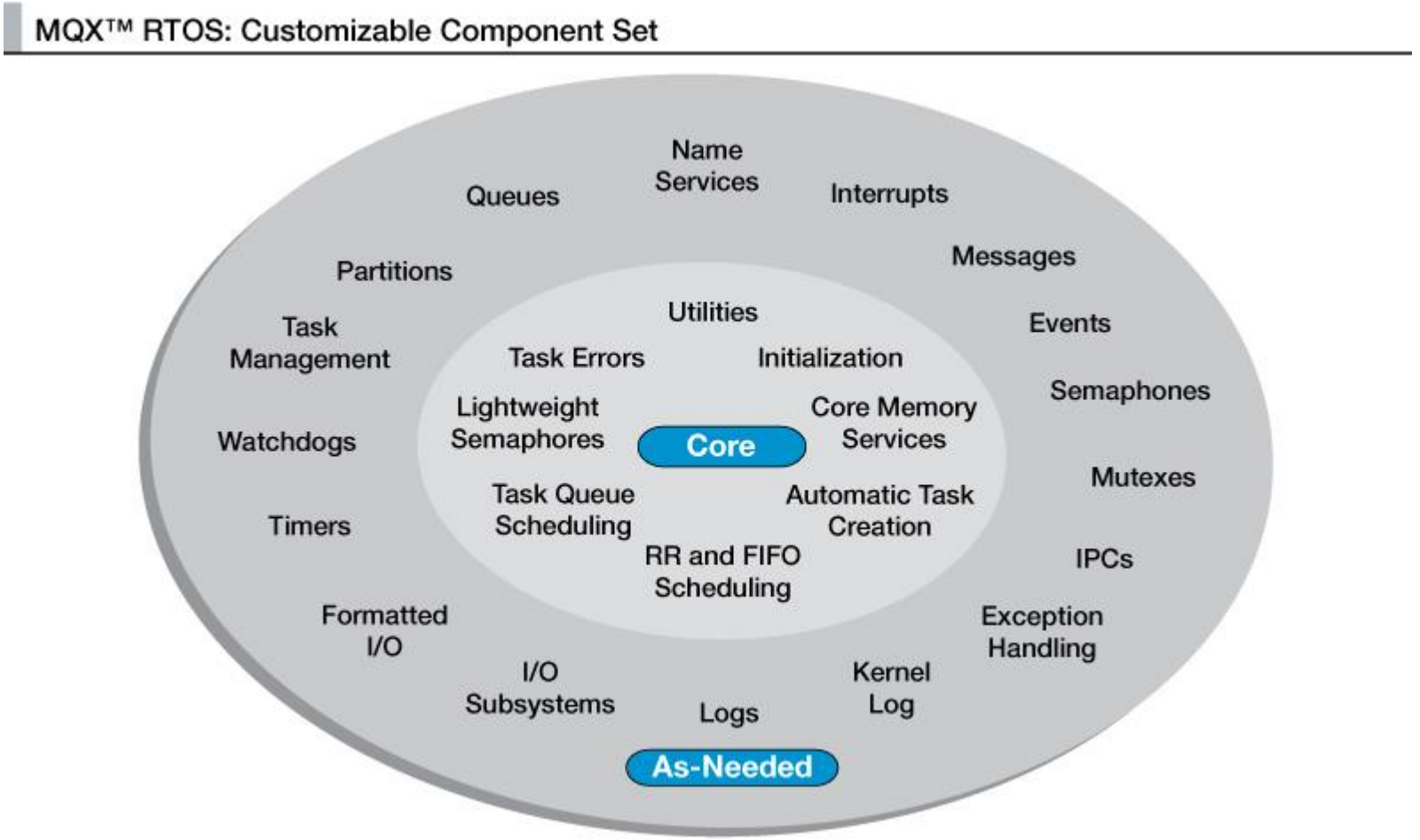
```
void StartTask02(void *argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  for(;;)
  {
   HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_13);
    osDelay(1000);
  }
  /* USER CODE END StartTask02 */
}

void StartTask01(void *argument)
{
  /* USER CODE BEGIN StartTask01 */
  /* Infinite loop */
  for(;;)
  {
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);

    osDelay(1000);

  }
  /* USER CODE END StartTask01 */
}
```



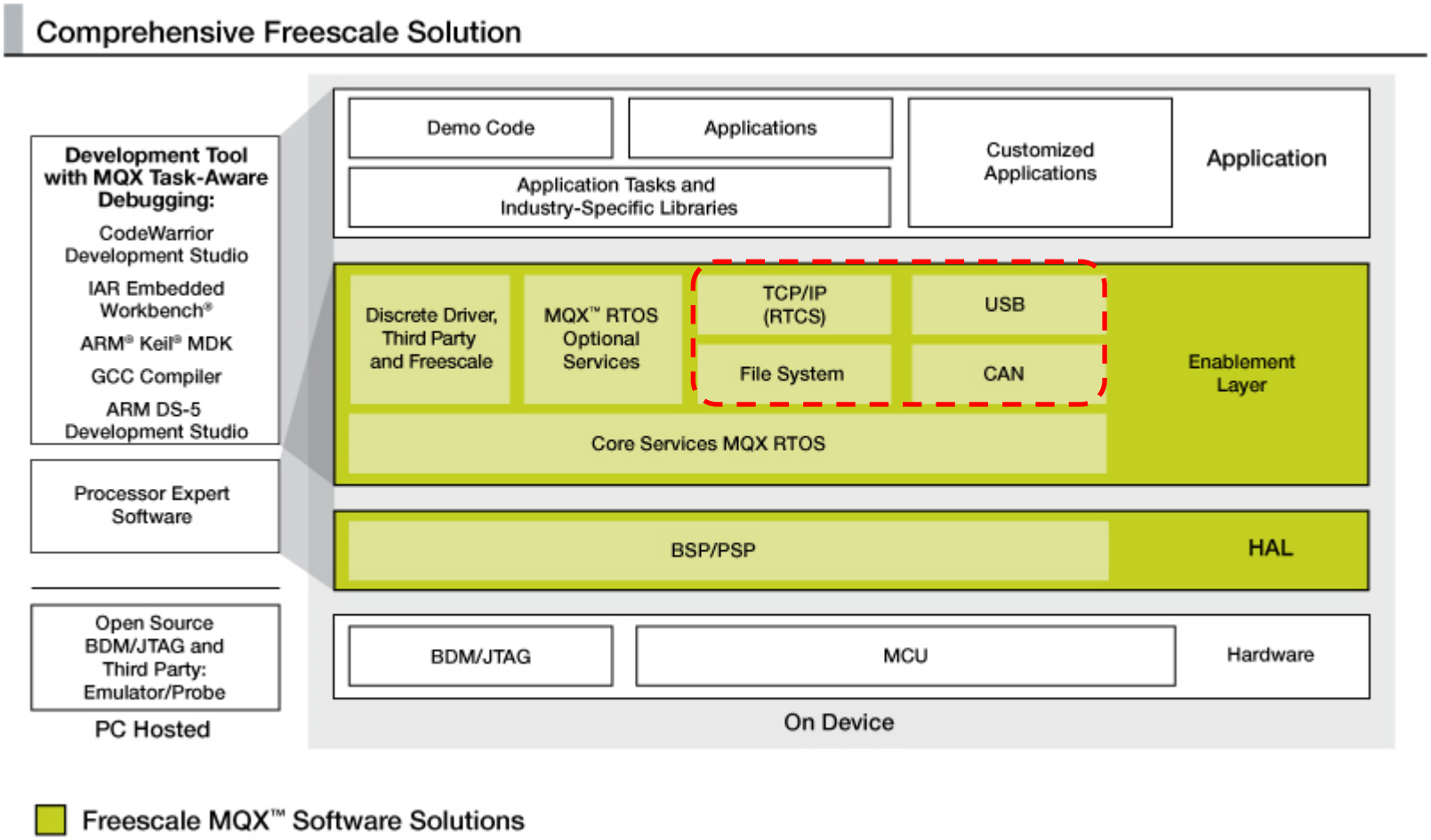
4.2. FreeRTOS (STM32F4 primer) :

Opravila („Tasks“)

4.3 MQX RTOS (primer) :



4.3 MQX RTOS (primer) :



4.3 MQX RTOS (primer) :

Opravila („Tasks“)

```
const TASK_TEMPLATE_STRUCT  MQX_template_list[] =
{
   /* Task Index,     Function,          Stack,  Priority,                    Name,                    Attributes,                    Param,   Time Slice */
{ MNG_TASK,      MngTask,           1200,   TASK_PRIORITY_MNG_TASK,   MNG_TASK_DES,          MQX_AUTO_START_TASK,        0,           0 },

{ SHELL_TASK,   ShellTask,          2000,  TASK_PRIORITY_SHELL,  SHELL_TASK_DES,      0,                          0,           0 },

{ FP_TASK,          FunPgmTask,        2000,   TASK_PRIORITY_FP,           FP_TASK_DES,           0,                          0,           0 },

{ TNSH_TASK,      TelnetClientShell,   2000,   TASK_PRIORITY_TNETSH,TNSH_TASK_DES,     0,                          0,           0 },

{ TCPCLIENT_TASK,TCPClient_Task, 2000,  TASK_PRIORITY_TCPCLIENT,TCPCLIENT_TASK_DES,  0,                          0,           0 },

{ MODBUS_TASK,Modbus_Task,    2000,   TASK_PRIORITY_MODBUS,MODBUS_TASK_DES,  0,                          0,           0 },

{ EVTALM_TASK,EventAlmTask,        2000,   TASK_PRIORITY_EVTALM,EVTALM_TASK_DES, 0,                          0,           0 },

{ AIN_TASK,         AinTask,           500,     TASK_PRIORITY_AIN,          AIN_TASK_DES,          0,                          0,           0 },

{ NETMNG_TASK,NetMngTask,        1000,   TASK_PRIORITY_NETMNG,NETMNG_TASK_DES,  0,                          0,           0 },

{ 0 }

};
```

4.3 MQX RTOS (primer MQX opravila) :

Glavna regulacijska zanka („FP_TASK“)

```
void FunPgmTask (uint_32 initial_data)
{
FunPgmInit();

// register task for system messages
rc = SysMsgRegister ();

// WDT control
WdtRegister (15000, WDT_ACTION_LOG);

// ------------------------------------  main execution loop ----------------------------
while (TRUE)  {

        _time_get_elapsed (&fp_start_time);   //Measure processing time fp_start_time

        WdtReset ();

        FunPrepareFPData();             // Prepare FP data
        FunRegulation();                // Iterate regulation loops
        FunCommitFPData();              // Commit any changes back to system

        _time_get_elapsed (&fp_end_time);   //Measure processing time
        _time_diff (&fp_start_time, &fp_end_time, &fp_loop_time);      // get elapsed time
        FPLoopTime=(fp_loop_time.SECONDS * 1000) + fp_loop_time.MILLISECONDS;

        _time_delay(1000-FPLoopTime);// wait for 1000 ms - loop time in ms
        }
_task_block();                                          // Shouldn‘t reach this point
}
```

```
/** @brief FP: Main Functional Program Task.
Calls FunPgmInit for initialization and then runs endless main FP loop.
 *
 *  This is main functional program task.
 *  It will first run Initializations: FunPgmInit();
 *  Then it will proceed in endless loop :
 *              FunPrepareFPData();     // Prepare FP data
 *              FunRegulation();        // Iterate regulation loops
 *              FunCommitFPData();      // Commit any changes back to system
 *              check if settings changed - if yes, then read all settings
*/
```

void FunPgmTask (uint_32 initial_data)

FP: Main Functional Program Task. Calls FunPgmInit for initialization and then runs endless main FP loop.

This is main functional program task. It will first run Initializations: **FunPgmInit()**; Then it will proceed in endless loop : **FunPrepareFPData()**; // Prepare FP data **FunRegulation()**; // Iterate regulation loops **FunCommitFPData()**; // Commit any changes back to system check if settings changed - if yes, then read all settings

Todo:
Temporary - should't be used in production code **!!!**

Definition at line **139** of file **fp.c**.

References **APPCFG_DEFAULT_FP_USER_ACCCODE**, **APPDBG_PRINTF**, **D13_GVARS::Day**, **FunCommitFPData()**, **FunLogCurrentState()**, **FunPgmInit()**, **FunPrepareFPData()**, **FunRegulation()**, **FunSimCommitFPData()**, **FunSimLogCurrentState()**, **FunSimPgmInit()**, **FunSimPrepareFPData()**, **FunSimRegulation()**, **FP_DATA::GVars**, **D13_GVARS::Hour**, **D13_GVARS::Minute**, **D13_GVARS::Month**, **Read_FPSettings()**, **D13_GVARS::Second**, and **D13_GVARS::Year**.