Univerza v Mariboru
Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Porocilo projekta pri predmetu :

NAPREDNI PROGRAMIRLJIVI ELEKTRONSKI SISTEMI

Mario Gavran

Maribor, februar 2018

Sadržaj

1. Naloga

Pogoji za ustreznost in uspešno opravljanje projekta so:

- program je zasnovan za mikrokrmilnik, ki omogoča uporaba RTOS sistema (tipično ARM Cortex-M).
- uporabljen je RTOS sistem s CMSIS API vmesnikom (lahko je Keil RTX, lahko je FreeRTOS ...)
- V sistemu je smiselno uporabljen RTOS, uporabljenih je več niti med katere je razdeljena funkcionalnost sistema in uporabljene so razne komunikacije med nitmi
- Uporabljena je lahko razvojna plošča (Atmel, STM, Texas ...), po potrebi pa še dodatna strojna oprema, ki pa naj bo izvedena z minimalnim naporom, vendar še vedno tako, da bo možno demonstrirati delovanje sisteme (drugače: povdarek projekta je na programski opremi)
- Na koncu semestra oddate projekt: izvorno kodo, morebitni načrt za strojni del, poročilo (ca. 5 do 10 strani) in predstavitev. V poročilu in predstavitvi naj bo povdarek na tem kako je bil uporabljen RTOS in njegove komponente, da ste dosegli funkcionalnost projekta.

Naloga:

Pametni semafor

- osnovno delovanje semaforja na križišču (dve cesti + pešci) + zajemanje tipke za pešce. Dodatno še implementirajte zaznavanje št. avtomobilov, hitrosti avtomobilov, deža (pogosteje zelena za pešce), prikazovanje kako dolgo do zelene ipd. Pripravite tudi osnovno strojno opremo, da boste lahko demostrirali delovanje programa.

2. Izbira rešitve

Program je zasnovan za 16MHz mikrokrmilniku, STM32F410, ki omogoča uporabo RTOS sistema. Izbran je CMSIS RTX RTOS kot operacijski sistem v realnem času. Narejena je osnovna strojna oprema da bo lahko demonstrirano delovanje programske opreme.

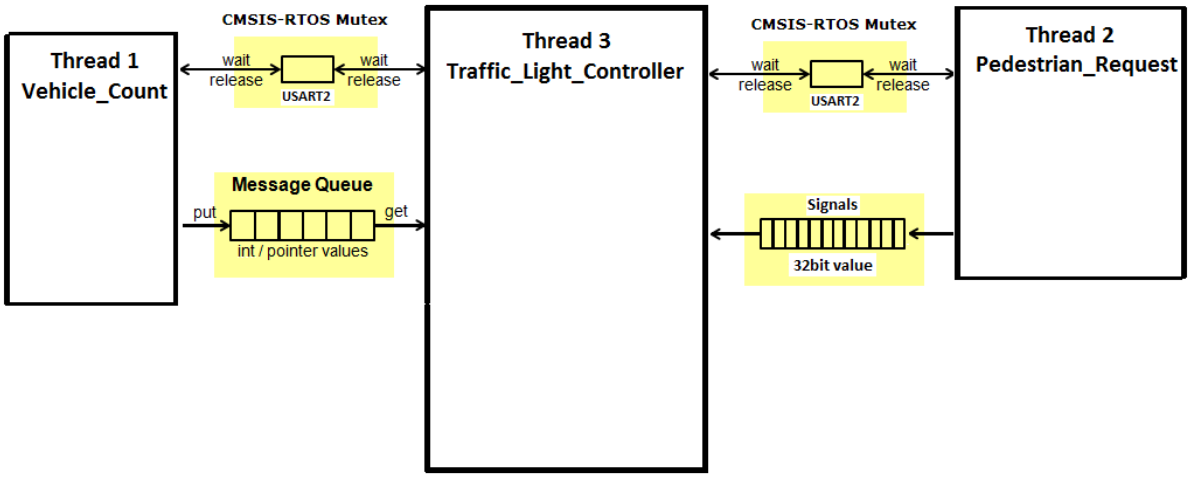
Za ta projekt je izdelan pametni krmilnik semaforja ki krmili stanja semaforja, določa koliko časa bo semafor zelen ali pa rdeč. Krmilnik semaforja še omogoča sprejem dva vhoda za tipke za pešce in dva vhoda za senzor števila avtomobilov pred semaforjem.

Program ves čas zajema število avtomobilov pred semaforjem. Na podlagi zaznanega števila avtomobiljev, program izračuna razliko med njima in ko se spremeni stanje posameznega semafora iz rdeče v zeleno ali obratno, program izračuna čas potreben za držanje zelene luči posameznega semaforja.

Tudi ves čas, kot za število avtomobiljev, program zajema stanje tipke za pešce. Če je pešec pritisnul tipko med tem ko je vključenja rdeča luč na semaforu, preko katerega pešec želi prečkat, čas zelene luči na tom semaforu se bo zmanjšalo na minimum.

V programu je uporabljenih tri niti med katere je razdeljena funkcionalnost sistema. Funkcionalnosti sistema so: zaznavanje števila avtomobilov, zaznavanje pešca kateri želi prečkati cesto in nit za krmiljenje semaforja. Uporabljene so razne komunikacije med nitmi: Mutex za pomoč pri razhroščevanju, MessageQ za pošiljanje podatkov o število avtomobilov pred semaforjem in Signals za signalizacijo pešca ki želi prečkat cesto.

Programska oprema je napisana v C kodi in še potrebuje doradu. Program je narejen uz pomoč priročnikov: *''The Designer's Guide to the Cortex-M Processor Family''* in *''CMSIS-RTOS API and RTX Reference Implementation''.* V priročniku obstaja poglavlje *Developing with CMSIS RTOS* ki predstavlja uvod v porabljanje RTOS-a na Cortex-M mikrokrmilnik.



3. Uporabljanje CMSIS-RTOS-a in njegovih komponent

Programska oprema ki uporablja CMSIS-RTOS mora imeti *RTX_Config_CM.c* konfiguracijsko datoteko in RTOS knjižnico *RTX_CM_lib*. Da bo lahko C koda dostupala do tih datoteka, rabimo vključiti *cmsis_os.h*:

```
#include "cmsis_os.h"                   // ARM::CMSIS:RTOS:Keil RTX
```

Potem lahko naredimo več niti. Svaka nit bo imala *Thread ID* in *osThreadDef* atribute niti.

```
// Create thread handles:
osThreadId main_id;
osThreadId thread1_id;
osThreadId thread2_id;
osThreadId thread3_id;


// Function prototypes for threads:
void thread1 (void const *argument);
void thread2 (void const *argument);
void thread3 (void const *argument);


// Thread structure definitions:
osThreadDef(thread1, osPriorityNormal, 1, 0);
osThreadDef(thread2, osPriorityNormal, 1, 0);
osThreadDef(thread3, osPriorityNormal, 1, 0);
```

RTOS se pokrene z naslednjim kodom v main niti:

```
int main (void)
{

  osKernelInitialize ();         // Initialize RTOS kernal.
  main_id = osThreadGetId ();    // main ID.


  // Create the threads, starts them running, read theri thread IDs:
  thread1_id = osThreadCreate(osThread(thread1), NULL);
  thread2_id = osThreadCreate(osThread(thread2), NULL);
  thread3_id = osThreadCreate(osThread(thread3), NULL);

  // Start RTOS kernal:
  osKernelStart();
```

Mutex

Mutex je uporabljen kot pomoč pri razhroščevanju programa.

Postopek vpeljave Mutex-a:

1. Deklaracija mutexa:

```
// Declare Mutex container and Mutex ID:
osMutexDef (uart2_mutex);                // Declare mutex.
osMutexId  uart2_mutex_id;               // Mutex ID.
```

2. Prireditev Mutex-a v niti:

```
// Create Mutex:
uart2_mutex_id = osMutexCreate(osMutex(uart2_mutex));
```

3. Zavzetev Mutex-a ko je zahtevan periferalni dostop:

```
osMutexWait(uart2_mutex_id, osWaitForever);
```

4. Ko končamo dostop do periferije, otpustimo mutex:

```
osMutexRelease(uart2_mutex_id);
```

MessageQ

MessageQ je uporabljen za pošiljanje podatka o številu avtomobilov s niti katera zajema senzor števila avtomobilov.

Postopek vpeljave MessageQ:

1. Nastavitev MessageQ-a

```
// Declare message queue and message id:
osMessageQDef(message_q, 1, uint32_t);  // Declare a message queue
osMessageQId  message_q_id;             // Declare an ID for the message queue
```

2. Prireditev MessageQ-a v niti:

```
// Create the message queue in a thread1:
message_q_id = osMessageCreate(osMessageQ(message_q), thread3_id);
```

3. Pošljanje sporočila:

```
// Try to send message every 200ms:
status = osMessagePut(message_q_id, data, 200);
```

4. Sprejemanje sporočila in preverjanje statusa sporočila:

```
event_message = osMessageGet(message_q_id, osWaitForever);

if(event_message.status == osEventMessage)
{
```

Signal event

Signali so uporabljen za naznako da pešc čaka na prosto cesto. Signal je 32 bitna beseda in jo vsebuje vsaka nit.

Postopek vpeljave Signals:

1. Definiramo strukturo ki vsebuje status in signal:

```
osEvent event_signal;      // Define event structure for signals.
```

2. V niti ki čaka na signal, pokličite funkcijo:

```
// Reciveing signals:
event_signal = osSignalWait(0,(hor_delay/2));      // Wait for signal to apear,
```

3. V drugi niti postavimo signal:

```
osSignalSet (thread3_id, H_PEDESTRIAN);
```