ROSSI-AVR Versio 1.1, päivitetty 17.7.2009

Reaaliaikainen skaalattava moniajokäyttöjärjestelmä AVR-prosessoreille.
Käyttöjärjestelmä on kokeiluversio ja testaus on kesken.
Suunnittelu ja koodaus: Heikki Palomäki, email: heikki.palomaki<at>seamk.fi
Esitetyt tavumäärät voivat muuttua hieman versioiden mukaan.

Perusominaisuudet

ROSSI-AVR – käyttöjärjestelmässä jokaisella taskilla (tehtävällä, ohjelmasilmukalla, säikeellä, prosessilla) on oma pinonsa, jossa sen tila eli rekisterit ja ohjelmalaskuri on talletettuna. Vain yksi taski voi olla tietenkin kerrallaan ajovuorossa. Lisäksi taskit voivat olla ajovalmiina tai odottamassa jotain tapahtumaa: tahdistussignaalia, resurssin vapautumista tai viestiä. Ajovuorossa oleva taski luovuttaa ajovuoron joko siirtymällä odottamaan tapahtumaa tai järjestelmäkello luovuttaa ajovuoron. Järjestelmäkello (Scheduler) vaihtaa tarvittaessa taskia määritellyn periodin välein, esim. 1ms. Jos taski siirtyy odottamaan, se siirtää seuraavaa järjestelmäkellon taskinvaihtoa taskinvaihtoperiodin verran eteenpäin. Käyttöjärjestelmä voi antaa myös tahdistussignaalin annetun ajan, esim. sekunnin välein. Käyttöjärjestelmä käyttää järjestelmäkellossa laskurin TCNT0 vertailukeskeytystä ja tahdistuksessa ylivuotokeskeytystä.

Taskeilla ei ole prioriteettijärjestystä: seuraava ajovuoroinen taski etsitään kiertävällä periaatteella, jolloin jokainen taski saa vuorollaan yhtä paljon ajoaikaa. Näin mikä tahansa taski voi olla pelkkä ohjelmasilmukka ilman yhtäkään käyttöjärjestelmäkutsua ja silti muutkin taskin saavat prosessoriaikaa.

Käyttöjärjestelmä on skaalattava, eli sen laajuus voidaan käännösvaiheessa määritellä tarpeiden mukaan. Järjestelmäkello on vakio-ominaisuutena, signaalit, resurssit, postilaatikot ja lisätoiminnat ovat ehdollisia ja otetaan käännösvaiheessa mukaan vain tarvittaessa.

Käyttöjärjestelmä on tehty käyttäen WinAVR –kääntäjää. ROSSI-AVR on tarkoitettu toimimaan ATMEL:n ATmega, AT90 ja ATtiny –prosessoreilla. Testausta on tehty lähinnä ATmega32 ja AT90647 –prosessoreilla.

Sovellusohjelman alussa on määriteltävä käyttöjärjestelmän perusominaisuudet: taskien maksimi lukumäärä, pinon koko ja scheduler. Lisäksi määritellään ehdollisesti käännöksessä mukaan tulevat ominaisuudet, mikäli niitä tarvitaan: sekuntitahdistus, tahdistussignaalien määrä, resurssien määrä, postilaatikoiden määrä, niitten koko ja lisätoiminnot. Ehdolliset ominaisuudet on merkitty (++) tässä ohjeessa.
Rossin koodin koosta saa käsityksen, kun eräässä käännösvaiheessa, kun WINAVR –kääntäjän optimointi on arvossa ”Os”, ROSSI vie käännettäessä koodia 534 tavua. Maksimikokoonpano vie silloin 1596 tavua. Nämä arvot voivat muuttua eri kääntäjissä versioissa.

Header –tiedoston määrittelyt

Perusmäärittelyt

Sovellusohjelman alkuun tulee luonnollisesti kääntäjän omat header –tiedostot sekä rossi.h –tiedosto. Ne toimivat WinAVR –kääntäjällä ja ne on tarkistettava kääntäjän asennushakemiston mukaisiksi.

#include <avr/io.h>
#include <avr/interrupt.h>
#include <rossi.h>

Rossin header –tiedostossa on seuraavat määrittelyt, jotka asetellaan sovelluksen vaatimusten mukaisiksi.

#define Xtal kiteen_kiloherzit

Käyttöjärjestelmälle kerrotaan prosessorin kidetaajuus kilohertseinä. Oletusarvona on 6000 (6MHz):

#define SysClock aika_ms

Keskeytyksellä ohjatun taskien vaihdon (Scheduler) periodin asetus. Oletusarvona on 1 ms. Arvoksi suositellaan 1-10 ms, että reaaliaikaisuus säilyy.

#define Options 1

Lisätominnat otetaan käyttöön tällä määrittelelyllä. Kun lisäfunktiot ovat käytössä, ne vievät koodia 154 tavua lisää.

Taskien raja-arvot

ROSSIn ensimmäinen taski on pääohjelma, joka jää pyörimään ohjelmasilmukkaan taskinumerolla ’0’. Muut taskit ovat funktion muotoisia, jotka jäävät myös pyörimään ohjelmasilmukkaan. Jokaisen uuden taskin koodia suoritetaan sitä luotaessa heti seuraavaan taskinvaihtoon saakka, ennen kuin pääohjelma saa ajovuoron ja lähtee luomaan seuraavia taskeja.

#define TaskNbr taskien_määrä

Taskien maksimimäärä, joka sisältää pääohjelmana olevan taskin numero ’0’. Taskit numeroidaan nollasta alkaen eli maksimiarvo on TaskNbr-1. Jos taskin numero ei ole sallitulla alueella, sitä ei alusteta. Oletusarvona on 5 taskia eli taskit ’0’, ’1’, ’2’, ’3’ ja ’4’.

#define TaskSize pinonkoko

Taskien pinon koko annetaan tavuina. Pääohjelman eli taskin ’0’ pino määräytyy kääntäjän varaaman pinoalueen perusteella. TaskSize määrää muitten taskien pinon koon numerosta ’1’ lähtien. Pinon kooksi suositellaan vähintään 100. Oletusarvona on 120. Mitä enemmän sovellusohjelmassa on keskeytysrutiineja ja sisäkkäisiä funktioita, sitä suurempi on pinon oltava.

Tapahtumien lukumäärät

Tapahtumilla tarkoitetaan taskien toimintaa tahdistavia ja tietoa välittäviä tapahtumia: signaaleja, resursseja ja postilaatikoita. Tapahtumien yhteismäärä on maksimissaan 255. Maksimimäärä täyttyy, jos kaikkia kolmea tapahtumatyyppiä on 85 kappaletta.

#define SigNbr signaalien_määrä (++)

Signaalien maksimimäärä tarkoittaa käytettävissä olevia erillisiä numeroituja tahdistussignaaleja. Signaalit numeroidaan nollasta alkaen. Suurin numero on siis SigNbr-1. Sovellusohjelman vapaassa käytössä on kaksi signaalia vähemmän, koska signaaleilla ’0’ ja ’1’ on erikoismerkitys. Signaalin ’0’ odotus vastaa ajovalmiina olemista eli taski pääsee mahdollisten taskinvaihtojen jälkeen heti uudelleen ajoon. Signaalia ’1’ käytetään tahdistamaan tapahtumia annetun välein. Signaalit eivät varaa datamuistia. Signaalien käsittely vie 98 tavua lisää käyttöjärjestelmäkoodia.

#define TimerSign tahdistusperiodi (++)

Sekuntitahdistus lähettää singaalin ’1’ annetun periodin välein eli vapauttaa sitä odottavat taskit. Aika annetaan sadasosasekunteina, eli arvolla 100 tahdistussignaali tulee sekunnin välein. Kun tämä toiminta on käytössä, koodia kuluu lisää 114 tavua ja järjestelmä varaa käyttöönsä laskurin 0 ylivuotokeskeytyksen..

#define ResNbr resurssien_määrä (++)

Resurssien maksimimäärä tarkoittaa käytettävissä olevia erillisiä numeroituja resursseja. Ne numeroidaan nollasta alkaen eli suurin numero on ResNbr-1. Resurssit varaavat datamuistia yhden tavun resurssia kohden. Resurssien käsittely vie ohjelmakoodia 170 tavua lisää.

#define BoxNbr postilaatikoiden_määrä (++)

Postilaatikoiden maksimimäärä tarkoittaa käytettävissä olevia erillisiä numeroituja postilaatikoita. Ne numeroidaan nollasta alkaen eli suurin numero on BoxNbr-1. Postilaatikot varaavat datamuistia puskurin pituudesta riippuen. Postilaatikkojen käsittely vie koodia 526 tavua lisää.

#define BoxLen postilaatikkojen_koko (++)

Postilaatikko toimii rengaspuskurina FIFO-puskurin tyyliin. Sillä voidaan esimerkiksi järjestää tiedonsiirtoputki kahden taskin välillä, jossa toinen lähettää jatkuvasti ja toinen vastaanottaa jatkuvasti. Yhden tiedon pituus on 16 bittiä eli yksi ’int’ –muuttuja. Yksi postilaatikko varaa datamuistia BoxLen+1 sanaa eli 2*(BoxLen+1) tavua.

Käyttöjärjestelmän funktiot

Käyttöjärjestelmän ja taskien alustukset

Alustustoimet tekee kääntäjän määrittelemä pääohjelma, eli main(). Samalla pääohjelmasta tulee käyttöjärjestelmän ensimmäinen taski, jonka numero on ’0’. Alustuksen jälkeen pääohjelma ei eroa muista taskeista muuten, kuin numeronsa perusteella. Lisäksi pääohjelmalla on kääntäjän varaama pinoalue omassa käytössään, muille varataan pinoalue erikseen.

void InitRossi(void);

Käyttöjärjestelmän alustus, jossa käytettävissä olevien taskien tiloiksi merkitään ’pois käytöstä’, paitsi kutsuvan ohjelman, joka jää ajovuoroon ja saa numeron ’0’. Lisäksi tässä kutsussa alustetaan kaikki resurssit vapaiksi ja postilaatikot tyhjiksi.

void CreateTask(ptr taski, int numero);

Uuden taskin luonti, jossa taski saa käyttöönsä oman pinoalueen ja pääsee välittömästi ajovuoroon seuraavaan taskinvaihtoon asti. Ensimmäinen parametri on taskin muodostavan funktion nimi, joka on esim. muotoa: void taski(void) {…}. Taskit numeroidaan siten, että pääohjelma, joka alustaa käyttöjärjestelmän, saa numeron ’0’ ja muut siitä järjestyksessä eteenpäin ’1’, ’2’ jne. enintään taskien maksimilukumäärä -1. Jos siis taskien maksimilukumäärä on 5, taskin numero saa olla siis enintään ’4’. Taski on rakenteeltaan pääohjelman tapainen, eli yleensä päättymätön silmukka. Jos taskin koodin suoritus päättyy koodin loppuun, se lopettaa toimintansa.

Taskien ajovuoroa ohjaavat ja tilaa tarkistavat käyttöjärjestelmäkutsut

Nämä ominaisuudet ovat käytössä, mikäli ne on sallittu ’#define Options 1’ –määrittelyllä Schedule – kutsua lukuunottamatta. Taskien ajovuoroa ei varsinaisesti tarvitse ohjata. Järjestelmäkello hoitaa sen automaattisesti ja muut käyttöjärjestelmä-kutsut tarvittaessa. Erikoistapauksissa voi olla tarvetta ohjata erikseen taskin vaihtoa, kun esim. silmukassa on odotettava jotain ulkoista tapahtumaa, mutta muitten taskien suoritusta ei haluta hidastaa tai jokin tehtävä halutaan suorittaa yhteen menoon ilman taskinvaihtoa.. Yleensä näiden käyttöjärjestelmäkutsujen käyttöä tulisi välttää.

void Lock(void); **(++)**

Käyttojärjestelmäkellon tekemän automaattisen taskinvaihdon esto. Taski voi vaihtua tämän kutsun jälkeen vain, jos taski siirtyy itse tapahtuman odotustilaan. Tämän kutsun käyttöä ei suositella, koska silloin reaaliaikaisuus menetetään. Tämä voi olla tarpeen hyvin aikakriittisten ohjelmanosien suorituksen varmistamiseksi.

void Unlock(void); **(++)**

Käyttöjärjestelmäkellon tekemän automaattisen taskinvaihdon uudelleensalliminen.

void Schedule(void);

Ajovuoron luovutus toiselle ajokelpoiselle taskille. Oma ajovuoro saadaan jälleen järjestelmäkellon kautta tai välittömästi, ellei ole muita ajokelpoisia taskeja.

char Scheduled(void); **(++)**

Taskinvaihdon tarkistus, jonka arvo on tosi, jos taski on menettänyt välillä ajovuoronsa.

void StopTask(int numero); **(++)**

Annetulla numerolla oleva taski menettää ajovalmiutensa ja lopettaa toimintansa. Pinoon jää kuitenkin taskin tila. Käyttöä ei suositella, koska seurauksena voi olla ristiriitatilanne.

void RunTask(int numero); **(++)**

Annetulla numerolla oleva taski siirtyy ajovalmiiksi, oli se sitä ennen, missä odotustilassa tahansa. Taski on kuitenkin oltava luotuna CreateTask –kutsulla, muuten tapahtuu järjestelmävirhe. Tämän käyttöjärjestelmäkutsun käyttämistä ei suositella, koska se voi aiheuttaa ristiriitaisia tilanteita.

char MyTask(); **(++)**

Oman taskin numeron kysyminen.

Tahdistussignaalien käsittely

Tahdistussignaalit ovat käytössä, mikäli ne ovat sallittuja ’#define SigNbr’ –määrittelyllä. Taskit tai keskeytykset voivat tahdistaa toisten taskien toimintoja signaalien avulla. Kun taski siirtyy odottamaan numeroitua signaalia, se menettää ajovuoronsa eikä kuluta prosessorin aikaa odotuksen aikana. Kun joku ajossa oleva taski tai keskeytys lähettää signaalin, odottava taski pääsee ajovuoroon. Signaalit on numeroitu ’0’, ’1’ … ja numero on maksimissaan signaalien maksimimäärä -1. Kaksi ensimmäistä signaalia on erikoiskäytössä. Signaali ’0’ ei ole normaalissa käytössä, vaan tätä signaalia odottava taski pysyy ajovalmiina, mutta voi menettää ajovuoronsa hetkeksi. Jos sekuntitahdistus on sallittu, käyttöjärjestelmä lähettää signaalin ’1’ automaattisesti asetellun periodin välein.

void WaitSign(int signaalinumero); **(++)**

Tahdistussignaalin odotuksessa taski menettää ajovuoronsa ja jää odotustilaan. Se ei pääse ajoon, ennen kuin joku muu taski tai keskeytysrutiini lähettää samannumeroisten signaalin.

void SendSign(int signaalinumero); **(++)**

Tahdistussignaalin lähetys, jossa kaikki samannumeroista signaalia odottavat taskit siirtyvät ajovalmiiksi ja lähettävä taski säilyy myös ajovalmiina. Signaalin lähettänyt taski menettää ajovuoronsa, jos jokin muu taski on ajovalmiina.

void PutSign(int signaalinumero); **(++)**

Tahdistussignaalin lähetys keskeytysrutiinissa, jossa kaikki samannumeroista signaalia odottavat taskit siirtyvät ajovalmiiksi ja keskeytyksen aikana ajossa oleva taski säilyy ajossa. HUOM ! tämän kutsun jälkeen keskeytykset on kiellettynä. Keskeytysrutiinista paluu sallii automaattisesti keskeytykset, mutta jos kutsu suoritetaan normaalissa taskin koodissa, keskeytykset on sallittava erikseen ’enable;’ –komennolla.

Resurssien käsittely

Resurssit ovat käytössä, mikäli ne ovat sallittuja ’#define ResNbr’ –määrittelyllä. Taskeilla voi olla yhteisiä resursseja eli esim. funktioita, joita ne voivan käyttää vuoronperään. Usein funktio käsittelee yleisiä muuttujia tms. jolloin funktio ei toimi, jos suoritus keskeytyy ja joku toinen taski kutsuu samaa funktiota. Resurssina on esim. samat muuttujavaraukset, sama laiteresurssi tai ajallisesti etenevä kriittinen toiminta. Tällaisissa tapauksissa käytetään resurssin varausta ja vapauttamista. Ennen kuin taskin kutsuu yhteistä funktiota, se varaa numeroidun resurssin, joka kertoo muille, että funktio on käytössä. Kun funktio on suoritettu ja muut voivat käyttää yhteistä resurssia, sama numeroitu resurssi vapautetaan. Jos taskinvaihto tapahtuu kesken funktion suorituksen, muut funktiota tarvitsevat taskit jäävät odottamaan resurssin vapautumista eikä ongelmaa synny. Resurssinkäsittely on ikään kuin portti ulos ja portti sisään yhteiselle alueelle, jossa voi olla vain yksi kerrallaan. Resurssin odotus ei kuluta prosessorin aikaa. Resurssit numeroidaan ’0’, ’1’ … maksimissaan resurssien maksimilukumäärä -1.

void ReserveRes(int resurssinumero); **(++)**

Kun resurssi varataan ja resurssi on vapaana, taski jatkaa omalla ajovuorollaan. Samalla resurssi merkitään varatuksi. Jos resurssi on jo varattuna, taski menettää ajovuoronsa ja siirtyy odottamaan resurssin vapautumista.

void FreeRes(int resurssinumero); **(++)**

Resurssia vapautettaessa resurssin tila merkitään vapaaksi. Jos resurssin vapautumista odotetaan, pienimmällä numerolla varustettu taski saa ajovuoron ja kutsun tehnyt taski jää ajovalmiiksi odottamaan ajovuoroaan. Jos resurssia ei odoteta, resurssin vapauttaneen taskin ajovuoro jatkuu.

char AskRes(int resurssinumero); **(++)**

Resurssin tilan kyselyssä paluukoodina on ’1’ (tosi), jos resurssi on vapaana. Jos se on varattuna, paluukoodina on ’0’ (epätosi).

Postilaatikoiden käsittely

Postilaatikot ovat käytössä, mikäli ne ovat sallittuja ’#define BoxNbr’ –määrittelyllä. Postilaatikko on rakenteeltaan 16-bittisiä (int) lukuja sisältävä FIFO –puskuri. Se on sisäiseltä rakenteeltaan rengaspuskuri. Taski voi lähettää postilaatikkoon postia eli 16-bittisiä lukuja. Toinen taski voi hakea postilaatikosta postia. Jos laatikko on tyhjä, se jää odottamaan eikä kuluta prosessorin aikaa. Lähettävä taski menettää ajovuoronsa, jos postia ollaan jo odottamassa ja pääsee ajoon jälleen omalla vuorollaan. Jos keskeytysrutiini lähettää postia, ajovuoroa ei menetetä. Postilaatikon täyttymistä valvotaan sinne lähetettäessä. Postilaatikot numeroidaan ’0’, ’1’ … maksimissaan postilaatikoiden maksimilukumäärä -1.

char SendBox(int postinumero, int data); **(++)**

Sanoma lähetetään numeroituun postilaatikkoon. Data on 16-bittinen luku, joka puskuroidaan postilaatikon FIFO –puskuriin. Palautuskoodina on ’1’ (tosi), jos postin lähetys onnistui ja ’0’ (epätosi), jos postilaatikko oli täynnä. Ajovuoro menetetään, jos postia odotetaan.

char PutBox(int postinumero, int data); **(++)**

Sanoman lähetys keskeytysrutiinissa numeroituun postilaatikkoon. Data on 16-bittinen luku, joka puskuroidaan postilaatikon FIFO –puskuriin. Palautuskoodina on ’1’ (tosi), jos postin lähetys onnistui ja ’0’ (epätosi), jos postilaatikko oli täynnä. Keskeytyksen aikana ajossa oleva taski ei menetetä ajovuoroaan. HUOM ! tämän kutsun jälkeen keskeytykset on kiellettynä. Keskeytysrutiinista paluu sallii automaattisesti keskeytykset, mutta jos kutsu suoritetaan normaalissa taskin koodissa, keskeytykset on sallittava erikseen ’enable;’ –komennolla.

int WaitBox(int postinumero); **(++)**

Postia vastaanottava taski hakee numeroidun postilaatikon FIFO-puskurista 16-bittisen luvun. Jos postilaatikko on tyhjä, taski menettää ajovuoronsa ja jää odottamaan postia.

char AskBox(int postinumero); **(++)**

Postilaatikon tilan kyselyssä paluukoodina on ‘0’ (epätosi), jos postilaatikko on tyhjä. Jos postia on laatikossa, paluukoodina on ’1’ (tosi).

void ClrBox(int postinumero); **(++)**

Numeroitu postilaatikko tyhjennetään.

Oikeudet ROSSIn koodiin

Kaikki oikeudenvaatijat pidätetään eli kukaan ei voi vaatia mitään oikeuksia ROSSIn koodiin. Se on nyt julkaistu vapaasti käytettäväksi ja se pysyy sellaisena. Otan mielelläni ehdotuksia vastaan. Tavoitteena on pitää tämä minimiversio näin pienenä korjauksin täydennettynä. Jos joku haluaa laajentaa ominaisuuksia, siitä tehdään oma versio. Jos käytät uusissa versioissa tätä ROSSIn alkuperäistä koodia hyväksesi, se on hyväksyttävää vain silloin, jos **lopputulos on myös täysin vapaasti käytettävissä.**

Rossi on siis freeware –ohjelma lähdekoodeineen !