Univerzitet u Beogradu, Elektrotehnički fakultet

Sistemi u Realnom Vremenu
Projekat

Jovana Savić 2013/024

Opis problema

Potrebno je obezbediti startovanje akvizicije sa kanala A14, A15, A8 i A9 AD konvertora na svakih 100ms pomoću softverskog tajmera. Implementirati odloženu obradu prekida AD konvertora tako što se rezultat konverzije u prekidnoj rutini upisuje u red sa porukama (*queue*). Poruka treba da sadrži informaciju o kanalu koji je očitan i gornjih 8 bita rezultata AD konverzije.

Task ***xTask1*** čita podatke iz reda sa porukama i računa srednju vrednost zadnja 4 podatka za svaki kanal pojedinačno. Task ***xTask2*** ispituje tastere S3 i S4 i na pritisak odgovarajućeg tastera obaveštava task ***xTask1*** putem posebnog reda sa porukama o tasku kojem treba da posalje srednje vrednosti rezultata konverzije (***xTask3*** ili ***xTask4***). Task ***xTask1*** pakuje sva 4 podatka u jedan 32-bitni podatak i salje ga odgovarajucem tasku putem *mailbox*-a. Vrednosti se upisuju sa atributom *overwrite*. Taskovi ***xTask3*** i ***xTask4*** ne treba da se blokiraju kada treba da procitaju vrednosti iz mailboxa.

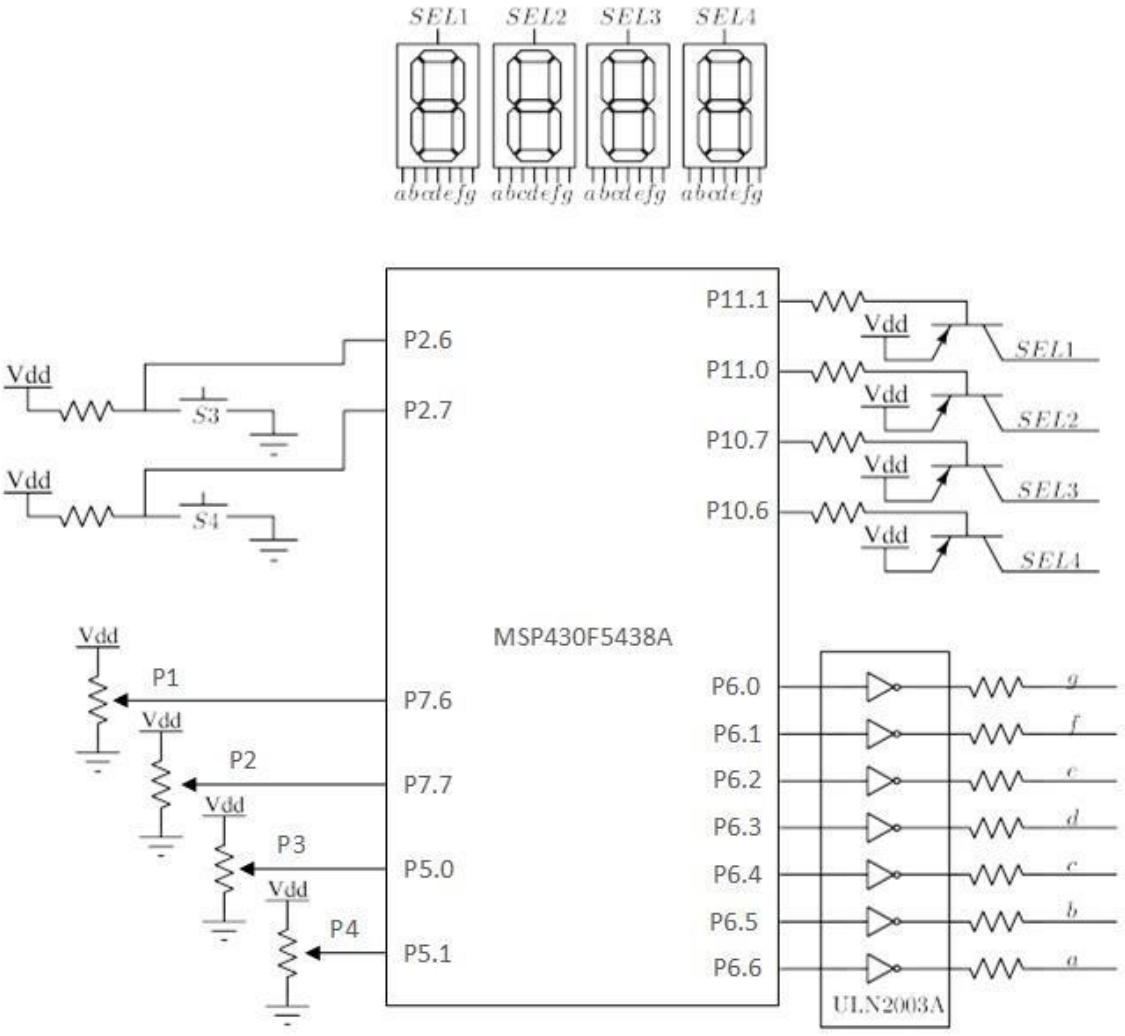
Taskovi ***xTask3*** i ***xTask4*** po prijemu podatka prikazuju sva četiri podatka na multipleksiranom LED displeju (najvisa 4 bita srednje vrednosti po kanalu, jedna cifra po kanalu). Obezbediti kontrolu pristupa izlaznoj periferiji.

*Task ***xTask1*** treba da ceka na oba reda sa porukama istovremeno.

1.Opis korišćenog hardvera

Rešenje je implementirano na mikrokontroleru ***MSP430F5438A***. Šema ploče na kojoj se nalazi mikrokontroler zajedno sa pomoćnim hardverom je data na adresi[1]. Na slici *1.1.* je prikazana uprošćena šema na kojoj je prikazan hardver korišćen u sklopu projekta. Korišćeno okruženje za pisanje programa je ***Code Composer Studio 7***. Program je pisan u programskom jeziku C, a za sinhronizaciju aktivnosti, rešavanje pristupa resursima i međusobnu komunikaciju task rutina je korišćen ***FreeRTOS***.

Kao što se vidi sa slike korišćena su dva tastera – S3 i S4, multipleksirani LED displej i četiri potenciometra čiji su ulazi povezani sa AD konvertorom. Korišćene su i periferije koje postoje u samom mikrokontroleru – tajmeri i AD konvertor.



Slika 1.1. Blok šema korišćenog hardvera

[1] http://tnt.etf.rs/~oe4irs/old/RS_MSP430F5438A_sch.pdf

2.Arhitektura softvera

Projekat se može podeliti na tri nivoa apstrakcije u odnosu na komunikaciju sa hardverom. Na najnižem nivou je sloj zadužen za direktnu komunikaciju sa hardverom. Na ovom nivou je definisana inicijalizacija LED displeja i metoda za ispis cifre na određenoj poziciji. Poštovan je princip enkapsulacije tamo gde to ne narušava znatno brzinu i veličinu programa. Naravno, u ovu grupu spadaju i sve metode koje su korišćene za inicijalizaciju hardvera. Drugi nivo apstrakcije su metode koje direktno komuniciraju sa hardverom, ali i sa metodama i taskovima koji su na najvišem niovu. Uglavnom su to prekidne rutine. One predstavljaju srednji nivo apstrakcije zato što jesu direktna komunikacija sa hardverom, ali se radi odložena obrada prekidna. U slučaju tajmera to se automatski radi uz pomoć taska za servisiranje tajmera koji poziva prethodno definisanu *callback* funkciju, dok je u slučaju AD konvertora to urađeno time što se podaci šalju tasku ***xTask1*** uz pomoć *queue*-a, a obradu vrši sam task. U ovu grupu bi takođe mogao da se svrsta i ***xTask2*** obzirom da se u njemu proveravaju stanja tastera. Konačno, kao najviši nivo apstrakcije se mogu izdvojiti ***xTask1***, ***xTask3*** i ***xTask4*** koji uopšte "ne vide" hardver, odnosno ne bi morali da se menjaju ukoliko bi se prešlo na neki drugi mikrokontroler.

LED displej – hardverska apstrakcija

Ovaj deo softvera se koristi za inicijalizaciju LED displeja i ispis cifre. Nalazi se u fajlovima *hal_LEDdisplay.h* i *hal_LEDdisplay.c*. U ovim fajlovima su definisani portovi selekcionih signala i signala kojim se selektuje cifra. Postoji metoda za inicijalizaciju `vHALInitLEDdisplay( void )` kojom se ovi portovi inicijalizuju kao izlazni i postavljaju na neaktivne vrednosti. Postoje dve metode za ispis cifre, metoda kojom se 16-bitni broj ispisuje u heksadecimalnom obliku `vDisplayNibble( uint16_t ucNumber, uint8_t ucWeight )`. Drugi argument je težina heksadecimalne cifre koja se ispisuje. Pre ispisa se ova težina maskira brojem 0x03 čime je obezbeđeno da bude u odgovarajućem opsegu. Takođe je uvedeno da se svi selekcioni signali postavljaju na neaktivnu vrednost (umesto da se na osnovu težine pretpostavlja koji je selekcioni signal bio aktivan i samo on gasi). Na ovaj način je metoda bezbedna u smislu da ne zavisi od redosleda kojim se cifre ispisuju. Radi kompletnosti uvedena je metoda za ispis cifre u decimalnom obliku `vDisplayDigit( uint16_t ucNumber, uint8_t ucWeight)`, iako se ne koristi u konkretnom projektu.

LED displej - multipleksiranje

Viši nivo apstrakcije upravljanja LED displejem se nalazi u fajlovima *LEDdisplay.h* i *LEDdisplay.c*. Na ovom nivou su definisane promeljive `uiDisplayNumber` i `ucDisplayWeight` , *mutex* `xLEDMutex` koji štiti ovaj resurs (broj koji se trenutno ispisuje) i tajmer `xTimer5` koji se koristi za multipleksiranje displeja. Postoji metoda kojom se inicijalizuje LED displej `vInitDisplay( void )` koja sadrži inicijalizaciju hardvera, promeljivih, stvara semafor i tajmer. Osim ove metode, ovde je definisana *callback* funkcija tajmera u kojoj se poziva metoda za ispis odgovarajuće heksadecimalne cifre i menja težina cifre koja se ispisuje u sledećem pozivu ove funkcije. Izabrano je da ispis ide preko 16-bitnog argumenta umesto, na primer, niza jer je mikrokontroler takav da poseduje 16-bitne registre pa je ovo brže nego slanje adrese niza i pristupa istom.

AD konvertor

Sve metode koje su u direktnoj vezi sa AD konvertorom se nalaze u fajlovima *adc.h* i *adc.c*. U sklopu ovih fajlova se nalazi funkcija za inicijalizaciju hardvera `vADCInitHardware(void)` kojom se obezbeđuje prikupljanje podataka sa sva četiri kanala. Konverzija se startuje na svakih 100ms, i ovo je rešeno uz pomoć tajmera. U *callback* funkciji ovog tajmera se startuje konverzija. Osim funkcije za inicijalizaciju hardvera napravljena je i uopštena funkcija za inicijalizaciju `vADCInitADC(void)` u kojoj se osim hardvera inicijalizuje tajmer i *queue* `xADCDataQueue` koji se koristi za čuvanje rezultata konverzije. Po završetku konverzije, AD konvertor šalje prekid. U okviru prekida se rezultat konverzije pakuje u poruku koja sadrži kanal sa koga je stigao rezultat i gornjih 8 bita konvertovane vrednosti. Ova poruka se smešta u *queue* koji odgovara AD konvertoru. Dalja obrada podataka se vrši u tasku ***xTask1***.

xTask2

Task ***xTask2*** se koristi za debaunsiranje tastera S3 i S4 i nakon detekcije pritiska tastera šalje poruku koja sadrži informaciju o tome koji je taster pritisnut tasku ***xTask1*** preko *mailbox*-a. U vezi sa ovim taskom je i tajmer koji se aktivira svakih 50ms. Ovo je task srednjeg prioriteta. Metode i promenljive u vezi sa ovim taskom su definisane u fajlovima *Task2.h* i *Task2.c*. Ovde je definisana metoda `vInitTask2( void )` kojom se inicijalizuje dati task, pri čemu inicijalizacija podrazumeva kreiranje taska, kreiranje *mailbox*-a `xTask2Queue` (*queue* dužine 1) i inicijalizaciju tajmera. Osim ove metode, ovde su definisane i *callback* funkcija tajmera u kojoj se šalje notifikacija tasku ***xTask2***, kao i sam task. Ovaj task je blokiran i čeka na notifikaciju tajmera. Nakon dobijanja notifikacije se odblokira i proverava stanja tastera. Ukoliko je detektovano otpuštanje tastera kreira se poruka koja sadrži informaciju o tome kom tasku treba poslati podatke – ***xTask3*** ukoliko je pritisnut taster S3 ili ***xTask4*** ukoliko je pritisnut taster S4. Ova poruka se smešta u *mailbox* ovog taska. Tajmer u ovom slučaju obezbeđuje da se ***xTask2*** odblokira na svakih 50ms.

xTask3 i xTask4

Ovo su taskovi najnižeg prioriteta i suštinski rade iste stvari pa će biti opisani zajedno. Taskovi su definisani u fajlovima *Task3.h, Task3.c, Task4.h* i *Task4.c*. Oba taska imaju po jedan *mailbox* `xTask1QueueT3, xTask1QueueT4` u koji ***xTask1*** upisuje poruku. Potrebno je napomenuti da se redovi sa porukama inicijalizuju u okviru onih metoda koje upisuju vrednosti u te redove, pa su oba reda definisana u okviru taska ***xTask1***. Zbog toga se inicijalizacija ovih taskova svodi na kreiranje istih i njihovih pratećih tajmera. Metode kojima se ovi taskovi inicijalizuju su `vInitTask3( void )` i `vInitTask4( void )`. Ovi taskovi proveravaju svoj *mailbox*, ali se ne blokiraju ukoliko nije stiglo ništa. Ukoliko je stigla poruka, task će pokušati da zauzme *mutex* koji čuva LED displej. Poruka koju task dobija je u vidu 32-bitnog broja – tu su upakovane srednje vrednosti poslednje 4 konverzije sa svakog kanala (najnižih 8 bita sa kanala 0, sledećih 8 sa kanala 1 itd.). Ovi taskovi treba da uzmu 4 najviša bita rezultata svakog kanala i ispišu ih na LED displej. Iz poruke se uzmu potrebni podaci i smeste u jedan 16-bitni broj. Nakon ove obrade, potrebno je postaviti dati broj kao onaj koji se ispisuje. To je kritična operacija gde je potrebno zaštiti zajednički resurs, pa se pre toga traži mutex koji štiti LED displej. Task se blokira dok se ne oslobodi resurs. Onda kada dobije dozvolu, izračunat 16-bitni broj se upisuje u promenljivu

`uiDisplayNumber` koja predstavlja broj koji se trenutno ispisuje na LED displeju. Task odmah nakon toga oslobađa *mutex*. Pošto su ovo jedini taskovi koji mogu da se blokiraju na datom semaforu, blokiranje na semaforu neće izazvati problem u kome task većeg prioriteta čeka na task manjeg prioriteta. Na kraju, task se blokira na 20ms.Ovo je urađeno uz pomoć tajmera koji odblokira task na svakih 20ms slanjem notifikacije (na isti način kao što je to urađeno u slučaju taska ***xTask2***).

xTask1

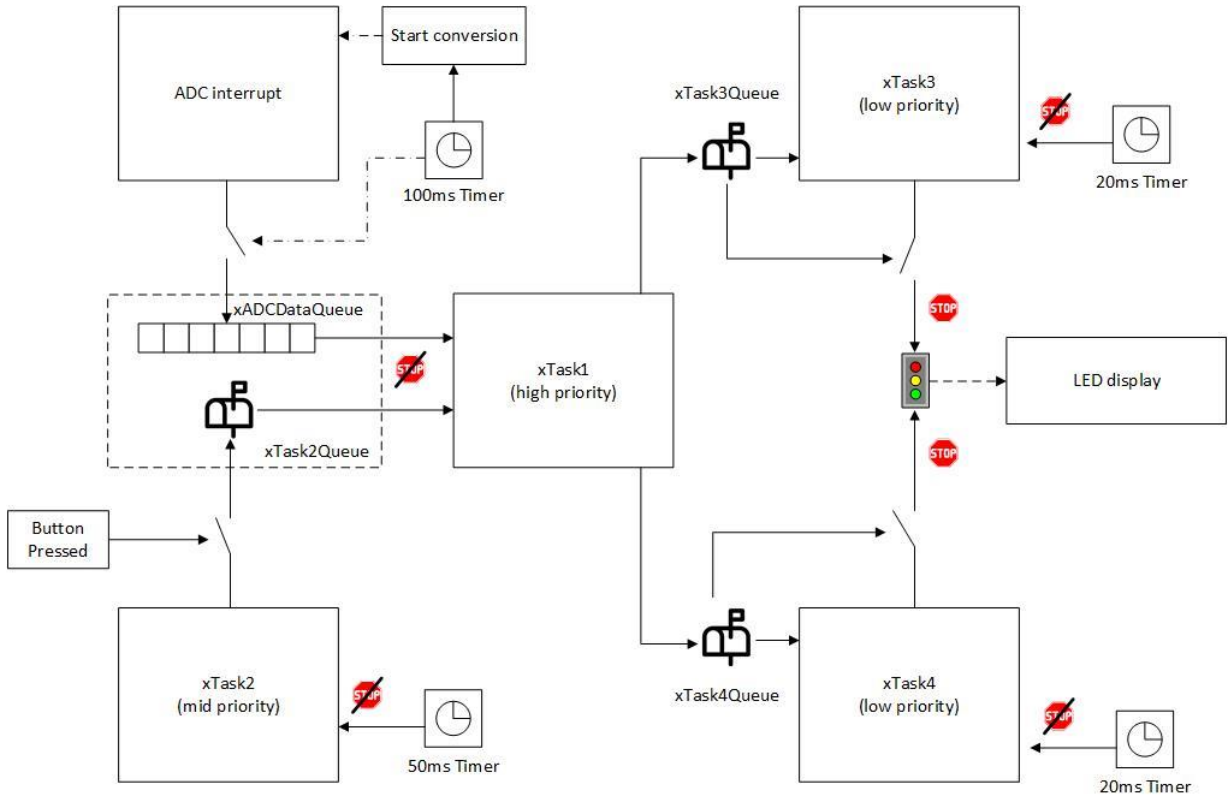
Ovo je task najvišeg prioriteta. Metode i promenljive u vezi sa ovim taskom su definisane u fajlovima *Task1.h* i *Task1.c*. U vezi sa ovim taskom su mailbox taska ***xTask3*** i mailbox taska ***xTask4***. Ovi redovi se inicijalizuju pri inicijalizaciji taska koja se izvršava pozivom funkcije `vInitTask1( void )`. Dati task čeka na poruku od taska ***xTask2*** ili rezultat konverzije. Task je blokiran dok ne dođe bar neka od željenih poruka. Da bi se obezbedilo čekanje na oba reda sa porukama istovremeno korišćen je RTOS objekat *QueueSet*. Ovo je objekat koji predstavlja apstrakciju više redova sa porukama koja obezbeđuje odblokiranje taska čim pristigne bar neka poruka. Ovaj objekat je principski isti kao jedan red sa porukama u kome bi poruka sadržala i još jednu informaciju koja govori o tome odakle je poslata. Na početku samog taska, pre petlje, se inicijalizuje skup redova sa porukama tako da se `xTask2Queue` i `xADCDataQueue` smeštaju u dati *QueueSet*. Bitno je napomenuti da je neophodno da su dati redovi već inicijalizovani pre smeštanja u *QueueSet*. Da ne bi došlo do problema usled inicijalizacije skupa redova poruka pre nekog od redova poruka, ova inicijalizacija je stavljena na početak taska, a ne unutar funkcije koja ga inicijalizuje (iako logički ima smisla da se nađe tu). Sam task je blokiran dok čeka na neku poruku. Nakon pristizanja poruke, u zavisnosti od toga odakle je ona stigla radi se odložena obrada prekida ili slanje poruka taskovima ***xTask3***, odnosno ***xTask4***. Ovaj task čuva poslednja 4 rezultata konverzije za svaki kanal u vidu 32-bitnog broja. Ovo je izabrano jer se pamćenje novog rezultata svodi na šiftovanje datog broja za 8 mesta ulevo i dodavanje pristiglog rezultata. Ovo je bilo elegantnije rešenje od korišćenja kružnih bafera ili prostog niza. Ukoliko je stigla poruka koju je poslao ***xTask2*** radi se računanje srednje vrednosti poslednja četiri rezultata konverzije za svaki kanal. Ove srednje vrednosti se pakuju u jedan 32-bitni broj gde najnižih 8 bita predstavlja srednju vrednost kanala 0, sledećih 8 kanala 1, itd. Nakon toga se proverava poruka koju je poslao ***xTask2***. Ukoliko je pristigla poruka *task3*, to znači da se taj 32-bitni broj stavlja u mailbox taska ***xTask3***. Analogno tome, ukoliko je stigla poruka *task4*, broj se stavlja u mailbox taska ***xTask4***.

Idle task

Idle task je task aktivira *low power mode* datog kontrolera.

3.Komunikacija između taskova

Na slici 3.1. se može videti uprošćena šema koja predstavlja komunikaciju između taskova.



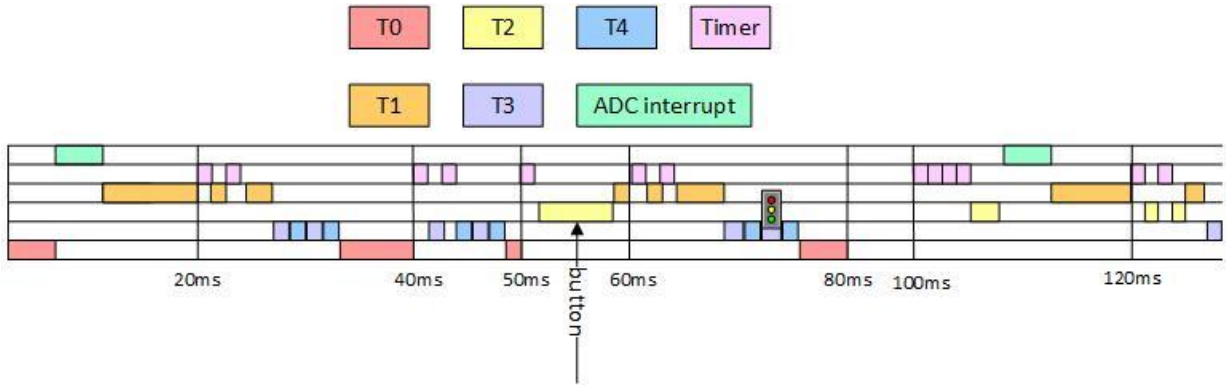
Slika 3.1. Komunikacija između taskova

Na šemi je precrtanim stop znakon predstavljena situacija u kojoj se task odblokira, ukoliko zadati uslov nije ispunjen task je blokiran. Tako, za ***xTask2*** imamo situaciju da je task po *default*-u blokiran. Onog trenutka kada prođe 50ms aktivira se tajmer koji šalje notifikaciju koja će odblokirati ovaj task. Pored toga, prikazan je *mailbox* u koji dati task upisuje poruku. Kao što se vidi sa slike poruka se šalje ako je pritisnut taster. U gornjem levom uglu je prikazan prekid AD konvertora. Kao što se vidi sa slike, konverzija se startuje na svakih 100ms uz pomoć tajmera, a time se indirektno i prekidi koji se dešavaju u *burst*-ovima od četiri prekida (pošto sva četiri kanala počinju konverziju istovremeno, praktično je i istovremeno završavaju) dešavaju na svakih 100ms. Zbog toga je prikazano da se upis u red sa porukama izvršava indirektno na 100ms. Kako task ***xTask1*** čeka na ma koju poruku one su objedinjene na šemi. Isto kao i u slučaju sa taskom ***xTask2***, lakše je prikazati ***xTask1*** kao po default-u blokiran, a odblokira se kada se desi prijem ma koje poruke. Isto tako taskovi ***xTask3*** i ***xTask4*** se blokiraju na 20ms uz pomoć tajmera, pa je ovde isto prikazano da se odblokiraju kada istekne predviđeno vreme, ali je u njihovom slučaju i prikazana situacija u kojoj se dešava blokiranje taska. Kao što se vidi sa šeme, prijem poruke od strane taska ***xTask1*** će inicirati zahtev za zajedničkim resursom koji je zaštićen *mutex*-om. Znak stop označava da postoji mogućnost blokiranja taska na datom semaforu. Kao što se i vidi sa šeme, ovi taskovi se ne blokiraju čekajući na poruke od taska ***xTask1***. Konačno, ovde valja pomenuti da se *callback* funkcije

tajmera ne izvršavaju u sklopu prekidne rutine tajmera, već posebnog sistemskog taska. U ovom projektu ovaj task ima najviši prioritet. Prema tome, prekidi i taskovi su prema prioritetu, počev od najvišeg dati sledećim redosledom: ***ADC interrupt***, ***timer callback***, ***xTask1***, ***xTask2***, ***xTask3*** i ***xTask4, idle task***. Za taskove istog prioriteta se koristi round robin algoritam raspoređivanja.

4. Vremenski dijagrami

Na slici 4.1. je prikazan vremenski dijagram. Na dijagramu nije prikazan tajmer koji kontroliše multipleksirani LED displej radi jednostavnosti. Ovaj dijagram je **samo ilustracija** vremenske logike programa.



Slika 4.1. Vremenski dijagram - primer

Na početku su svi taskovi blokirani. Izvršava se idle task. Potom se pojavljuje prekidna rutina AD konvertora. Po prijemu poruke sa prvog kanala se odblokira task xTask1, ali se vrlo brzo pojavljuju prekidi sa drugih kanala. Radi pojednostavljenja slike, prikazano je kao da će podaci sa sva četiri kanala stići praktično jedan za drugim, bez promene konteksta u međuvremenu. Nakon 20ms se aktivira *callback* funkcija tajmera koji odgovara trećem tasku. Kao što je već pomenuto, ovi taskovi imaju najviši prioritet, pa zbog toga dolazi do promene konteksta. Pretpostavljeno je da tajmer koji odgovara četvrtom tasku blisko prati tajmer *xTimerT3*. Nakon njegove aktivacije i četvrti task više nije blokiran. Pretpostavka je da prvi task još uvek nije blokiran, to jest i dalje obrađuje poruke koje je poslao AD konvertor. Kada se isprazni red sa porukama, ovaj task se blokira. Sada se smenjuju taskovi ***xTask3*** i ***xTask4*** obzirom da su istog prioriteta. Ovi taskovi jako kratko traju čak i kada im pristigne poruka (što se ovde smatra da nije slučaj), pa su sada svi taskovi osim *idle* taska ponovo blokirani.

Nakon 40ms se ponovo aktiviraju tajmeri koji će deblokirati treći i četvrti task. Nakon još 10ms se aktivira tajmer task koji odblokira drugi task. Sada je ***xTask2*** aktivan i pošto je najvišeg prioriteta preuzima procesor. Strelicom je predstavljeno da task detektuje pritisak tastera (npr. tastera S3). Nakon detekcije poslao je poruku tasku ***xTask1*** koji odmah nakon toga više nije blokiran i nastavlja izvršavanje. U toku izvršavanja, on je prekinut tajmer taskovima koji ponovo deblokiraju ***xTask3*** i ***xTask4***. Kako je xTask1 višeg prioriteta, on nastavlja sa izvršavanjem. Kada završi obradu poruke poslao je 32-bitni broj trećem tasku. Sada ovaj task zauzima semafor i menja ispis na LED displeju.

Sledeća interesantna situacija je nakon 100ms. Tada su se svi tajmeri aktivirali, odnosno svi taskovi osim prvog više nisu blokirani i započeta je AD konverzija. Počinje izvršavanje drugog taska. Njega prekida

prekidna rutina AD konvertora. Nakon te prekidne rutine se izvršava xTask1 koji više nije blokiran. Po njegovom završetku će na red ponovo doći drugi task. Na slici je prikazano kao da on ponovo šalje poruku i opet aktivira ***xTask1***.

5. Zaključak

Specifikacije projekta su veoma detaljne i precizne i ne ostavljaju mnogo prostora za interpretaciju, pa samim tim postoje velika ograničenja u vidu toga šta bi moglo da se menja, a da se ne izađe iz njihovih okvira. Ono što definitivno može da se poboljša jeste deo programa koji se bavi ispisom na LED displej. Iako pruža osnovne mogućnosti, ima prostora da se ova hardverska apstrakcija bolje uobliči i da se dobije generalnije rešenje. Ono što može takođe da se zameri jeste način na koji su definisani .h fajlovi. Naime, promenljive su deklarisane u ovim fajlovima kao eksterne, a definisane u .c fajlovima. Nekad se čini da takva ogranizacija programa poboljšava čitljivost, ali se i negde stiče utisak da su fajlovi zatrpani nepotrebnim informacijama. Kada je u pitanju formiranje poruka koje sadrže rezultate konverzije, izabrano je rešenje koje nije baš optimalno u smislu brzine, ali je davalo elegantan kôd. Svakako da bi se uz malo razmišljanja verovatno došlo do bržeg rešenja koje ne bi takođe bilo elegantno. Takođe, negde se može učiniti da se radi neka operacija, a kasnije u nekoj drugoj, sledećoj funkciji, gotovo njoj inverzna (pakovanje i raspakovanje poruke i priprema za ispis). Konačno, uložen je trud da se dobije rešenje koje je pre svega konzistentno, ali verovatno da postoje propusti koje bi trebalo ispraviti i bolje ogranizovati program.