Instruktioner - uppgift 1501f-ht17

Tommy Andersson (2017-12-13)

Översikt och syfte med uppgiften:

I den här uppgiften kommer du att göra en reaktiontidsmätare för två personer dvs ungefär samma tillämpning som du tidigare gjort i Matlab. Här skall du dock använda dig av Arduino Due, C-programmering och ett realtidsoperativsystem FreeRTOS. På vägen dit kommer du lära dig att bygga ett projekt från grunden i Atmel Studio, lära dig mäta med logikanalysator och förstå principerna för schemaläggning och semaforer i ett realtidsoperativsystem.

Utförande:

Uppgiften görs normalt två och två, men det går att jobba ensam också.

Det rekommenderas att ha ett eget USB-minne för katalogerna med din personliga kod även om det inte är helt nödvändigt.

Som vanligt så innehåller all mjukvara buggar. Se det som en del av labben att eliminera eventuella existerande buggar om de påträffas!

Krav för godkänt:

Ha genomfört följande labbmoment enligt denna handledning:

1. Skapat ett Arduino Due projekt i Atmel Studio
2. Adderat tillämpliga moduler inklusive FreeRTOS
3. Skrivit kod för två tasks som kontinuerligt togglar digitala utgångar.
4. Utfört och skriftligt redovisat mätningar enligt dokumentet "Labbrapport 1501f-ht17 och lämna in rapporten på It's learning.
5. Vidarutvecklat projektet för reaktionstest med två deltagare, filmat och lämnat in på video eller youtubelänk på Its learning
6. Committa filerna till ditt personliga repo och pusha till github classroom.

Förberedelser

Instudering

1. Du skall ha läst föreläsningsmaterialet om RTOS
2. Du skall ha läst chapter 3.1-3.9 översiktligt i FreeRTOS-tutorialen (https://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf)
3. Du skall också ha läst igenom hela denna handledning innan du går till labbsalen!

Mjuk och hårdvaruförutsättningar för att kunna börja på uppgiften

---

Som i tidigare labbuppgifter, men det finns inget repo klart att ladda ner, projektet skall byggas från grunden.

Själva uppgiften

Labbsetup

Utrustning:

- Utvecklingskort Arduino Due
- LCD-shield
- USB-kabel
- Digitalt oscilloskop plus logik-pod
- Samma elektronikplatta med lysdiod och tryckknappar som för reaktionstest labben med Matlab

Labbens programmeringsuppgift, del 1

I denna del skall du skapa ett projekt från grunden i Atmel Studio, skriva kod som togglar två I/O - pinnar, testa med mätningar, lägga till FreeRTOS i projektet och i stället skapa två tasks som togglar I/O - pinnarna.

Skapa ett projekt

I alla projekt för inbyggda system är hårdvaran väsentlig. Oftast inleds ett projekt med ett färdigt utvecklingskort, i ett senare skede tillverkas ett dedicerat kort. Vid uppstart behövs en assemblerfil (startup-fil) som gör diverse initieringar för den aktuella processorn. Vidare behövs C-filer med en del kortspecifika definitioner och initieringar.

Som utvecklingskort använder vi Arduino Due och som tur är dessa filer klara.
Skapa först ett projekt med Atmel Studio: File/New/Project. Välj alternativet GCC C ASF Board Project. Bestäm namn och plats.
I projektet finns nu en main.c fil som i stort sett är tom men innehåller ett anrop till en initieringsfunktion för kortet.
(Alternativt kan GCC C Executable Project väljas för att sen med Board Wizard för att initiera projektet för Arduino Due)

När detta är klart är det dags att addera lämpliga moduler till projektet.
Moduler som du behöver i detta skede

ASF module	**Beskrivning**
Generic board support (driver)	Innehåller kretskort-specifika definitioner och prototypfunktioner, såsom kretskort initiering funktionen.
System Clock Control (service)	Modul för klockkontroll (SYSCLK). Erbjuder funktioner för åtkomst, konfiguration, aktivering och inaktivering av klockor.
GPIO – General purpuse Input/Output (service)	Tillhandahåller funktioner för att initiera I/O pinnar som ingång eller utgång.
Standard serial I/O (stdio) (driver)	Standard I/O hantering modul, vilken implementerar stdio seriellt gränssnitt på SAM-enheter (kan behövas för debugging).
PIO – Parallel Input/Output Controller (driver)	Denna modul kan hantera upp till 32 fullt programmerbara ingång/utgång pinnar.
PMC - Power Management Controller	Denna modul optimerar strömförbrukningen genom att kontrollera samtliga system- och användarperiferi klockor.

Modulerna som inte redan finns med läggs till med ASF Wizard.
När du öppnar den dyker det upp en lista med **Available Modules** och en lista till höger **Selected Modules** med moduler som redan är tillagda.
Markera en modul i taget. Klicka sedan på knappen Add to selection. Nu kommer du kunna se modulen i Selected Modules. Klicka på Apply. Kryssa i rutan I accept the license agreement när det behövs och gå vidare.

De genererade filerna hamnar under `src/ASF` och tillägg görs automatiskt headerfilen i asf.h

Skriva kod - del 1

Det finns redan en initeringsfil för kortet men även systemklockan behöver initieras om vi vill kunna utnyttja processorn för fullt (84 MHz). Det fixas genom att anropet `sysclk_init();` läggs in i början main.c.
Du skall nu skriva kod direkt i main.c som snabbt togglar (dvs hög, låg, hög osv...) två I/O-pinnar. Använd pinnar som du kommer åt även när LCD-shielden sitter på plats.
Programmet skall alltså göra de initieringar som behövs och därefter ligga i en loop och toggla pinnarna.
Eftersom vi vill att det skall gå snabbt använder vi ioport-modulen i ASF. Gå tillbaka till uppgift 1501a-ht17, word-dokumentet, så ser du hur man kan göra.
När du skrivit koden testar du att utgångarna skiftar som de skall med hjälpa av oscilloskopet.
Passa också på att **notera** hur mycket program- respektive dataminne som behövs (framgår av utskriften, fliken OUTPUT, när projektet byggs). **Värdena skall senare redovisas i word-rapporten**.

Dags att installera FreeRTOS!

FreeRTOS finns som modul `FreeRTOS mini Real-Time Kernel` och adderas till projektet med ASF Wizard (välj nyaste versionen om det finns flera att välja på).
I Solution Explorer ser man att freertos-katalogen hamnar under katalogen `src/ASF/thirdparty/freertos-x.y.z`. I katalogen `config` skapas konfigurationsfilen `FreeRTOSConfig.h`.

Tasks

Nu skall du skriva kod för två tasks som i denna del bara skall toggla de två pinnarna du valt ut innan.

Skapa en ny c-fil med lämpligt namn för den första av de två tasks som ska köras. (markera src-mappen, välj Add New Item under Projectmenyn eller använd kortkommando...)
Denna c-fil skall innehålla följande funktion:

```
void task_player1(void *pvParameters)

{
    while(1)
    {
```

```
  /* Insert your code toggling one I/O pin here (first high then low) */
     }
}
```

Inkludera ASF.h för att funktionsanropen ska fungera.
Skapa också en h-fil som motsvarar denna c-fil och som du inkluderar i main.c och i c-filen du just skapat.

h-filen behöver innehålla några definitioner också, utöver deklarationen av task_player1:

```
#define TASK_PLAYER1_STACK_SIZE (2048/ sizeof(portSTACK_TYPE))
#define TASK_PLAYER1_PRIORITY   (1)
```

Se föreläsningspowerpointen om vad detta betyder. Ett vanligt problem som dyker upp när man använder RTOS är det blir stack overflow och det kan vara klurigt att komma på detta. Det är därför bra att starta med stor stack (om minnet räcker) och minska efteråt när man vet att programmet fungerar (Det finns även en speciell funktion i FreeRTOS som kan användas för kontrollera hur mycket av stacken som faktiskt används, ingår dock inte i kursen)

Gör nu på samma sätt för att skapa den andra tasken som du kallar task_player2.

Konfigurera FreeRTOS

För att kunna köra FreeRTOS måste det konfigureras i `FreeRTOSConfig.h`

Den färdiga konfiguration fungerar nästan i vårt fall...
Ett par saker behöver ändras just nu:
define configUSE_MALLOC_FAILED_HOOK
ändras från 1 till 0,
define configTICK_RATE_HZ ((portTickType)
1000) ändras från 1000 till 10000.
(det sista innebär att ticks kommer 10000 ggr/s, dvs var 100:e mikrosekund).
Detta är en ovanligt hög tick rate men underlättar mätningarna framöver.

Sätta samman allt i `main.c`

Än så länge är c-filerna du skapat bara vanliga funktioner. När `main()` körs skall de skapas som tasks och FreeRTOS dras i gång.
Justera `main.c` så den ser ut ungefär som nedan.

```
/* defines, includes of the h-files for the tasks
 * and declarations go here
 */

int main(void)
{
    /* Initialise the Due board */
    sysclk_init();
    board_init();

    /* initialise io-port pins */


     xTaskCreate(task_player1, (const signed char * const) "player1", T
ASK_PLAYER1_STACK_SIZE, NULL, TASK_PLAYER1_PRIORITY, NULL);

    xTaskCreate(task_player2, (const signed char * const) "player2", TA
SK_PLAYER2_STACK_SIZE, NULL, TASK_PLAYER2_PRIORITY, NULL);

    /* Start the FreeRTOS scheduler running all tasks indefinitely*/
    vTaskStartScheduler();

}
```

Förklaringar

- Funktionen `xTaskCreate()` skapar en task. Funktionen returnerar pdPASS om det lyckades, pdFAIL annars. Bra för felhantering men vi skippar detta här.
- argumentet `task_player1` är en funktionspekare till funktionen som skall utgöra tasken.
- ´(const signed char * const) "player1"´ är ett beskrivande namn på tasken kan underlätta debugging (Tyvärr finns det en slags bug i FreeRTOS när det gäller namnet på varje task; kompilatorn i Atmel Studio antar att en textsträng är en unsigned char, medan FreeRTOS antar att den är signed. Därför behövs en explicit typecasting göras)
- `TASK_PLAYER1_STACK_SIZE` se taskens h-fil
- `NULL` En pekare som kan användas som en parameter att skicka med till tasken
- `TASK_PLAYER_PRIORITY`, se taskens h-fil.
- `NULL` En pekare som kan användas som "handle" för att t.ex deleta tasken, vi behöver inte det.

Nu bör allt vara klart för att bygga projektet.
När du byggt utan fel så **notera** hur mycket minne som går åt. Jämför med den tidigare koden.

Detta är alltså priset för att använda RTOS...

(I versionen jag kört när jag utvecklat labben fås en varning om att configUSE_TICKLESS_IDLE not defined men fungerar bra ändå. Om man vill kan man komplettera FreeRTOSconfig.h med raden `#define configUSE_TICKLESS_IDLE 0` så försvinner varningen)

Om du inte gjort det innan är nu tid att initera git, addera filer och committa till ditt lokala repo (använda gärna ignorefilen som fanns med i förra labben).

Mätningar

I nästa steg skall du utföra mätningar med oscilloskop och logikanalysator enligt dokumentet "Mätningar - uppgift 1501g-ht17. När du är helt färdig med labben skall rapporten lämnas in på It's learning som vanligt.

Labbens programmeringsuppgift, del 2

I denna del skall du vidareutveckla koden för att bli en reaktionstidmätare för två deltagare med liknande funktion som i Matlab-labben 1501d-ht17.
Dock lite annorlunda krav:
K1: Ett spel utgöres av endast en omgång
K2: Spelet startas genom klick på knappen Select på LCD-shielden. Displayen clearas. Den stora lysdioden skall släckas om den är tänd.
K3: Displayen visar på valbart sätt att spelet startats.
K4: Efter en slumpmässig tid på 2-7 sekunder tänds den stora lysdioden.
K5: Programmet ska mäta hur lång tid det tar för varje spelare att reagera och trycka på sin knapp. Tiden skall mätas i millisekunder.
K6: När båda spelarna har tryckt på sina respektive knappar visas tiderna på displayen för player 1 och player 2. Det skall också framgå tydligt vem som vann.
K7: Om reaktionstiden för någon av spelarna överstiger 2 sekunder betraktas spelet som avslutat och resultatet visas i lämplig form på displayen (även användbart vid träning med en deltagare...).
K8: Det skall förstås inte vara möjligt för spelarna att fuska genom att hålla knapparna nertryckta från början.
K9: FreeRTOS och semaforanvändning skall ingå i lösningen.

Lösningstips

Uppgiften kan lösas på flera sätt, följande lösningstips måste inte följas.

Skapa en tredje task, namn t.ex. "control", som sköter start av spelet, displayen, den stora

lysdioden och tidmätningen.
Skapa två binära semaforer, en för varje player (görs i main()).
Låt control-tasken styra semaforerna så att de båda playertasken är blockerade fram till starten av tidmätningen. När sen respektive spelarknapp trycks ner indikeras detta med semaforerna.
Control-tasken pollar semaforerna och räknar antalet ticks från start fram till knapptryckningarna.

FreeRTOS-funktioner som kan vara användbara:
`xSemaphoreCreateBinary()`
`xSemaphoreTake()`
`xSemaphoreGive()`
`xTaskGetTickCount()`
`vTaskDelay()`
`vTaskDelayUntil()`

Info om dessa funktioner finns i:
https://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf
och
https://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V9.0.0.pdf

Slumptalsgenerator

Atmel-processorn har en inbyggd slumptalsgenerator, som kan generera ett nytt 32-bitars slumptal var 84:e klockcykel.
Du behöver i huvudsak två funktioner för att kunna använda slumptalsgeneratorn, se http://asf.atmel.com/docs/latest/sam3x/html/group__group__sam__drivers__trng.html

```
void trng_enable(Trng *p_trng); /* Görs vid initiering */
uint32_t trng_read_output_data(Trng *p_trng); /* Läs 32-bitars slumptal
 */
```

Argumentet till båda funktionerna `TRNG` anger instansen och finns redan som en `#define` i ASF.

Köra det färdiga programmet

Kör programmet och spela in en video när ni spelar och ladda upp antingen som länk till youtube-klipp eller som videofiler (obs max 30 sec per video), till It's learning.

Watchdog

I initieringsfilen för Due-kortet disablas watchdogen om man inte gör något åt det.
I en slutlig version av ett program vill man nästan alltid ha en aktiv watchdog. Den som vill får gärna aktivera watchdogen och fundera på var den bör reloadas (för att kolla att alla tasks verkligen körs kan man använda semaforer och ha en speciell task som sköter watchdogen…)

Inlämning på github och It's learning

Som i uppgift 1501f-ht17 plus videon.
Koden skall pushas upp till ditt repo på github classroom, länk finns på Its learning. Tänk på att koden skall vara väl dokumenterad så att det är lätt att förstå hur du har löst uppgiften, vilka I/O-pinnar som används till vad osv.