MAH_SVE_CMYK

Inbyggda system och signaler – DA264A

*Mätningar –*

*uppgift 1501g-ht17*

Student: …

Student: …

*Tommy Andersson*

*december 2017*

**Mätningar – uppgift 1501g-ht17**

# 

Skriv inte ut detta dokument. Ha det öppet på datorn under laborationen och besvara frågorna direkt i dokumentet.

Denna del av uppgift 1501g-ht17 genomförs i anslutning till första programmeringsuppgiften i dokumentet ”Instruktioner - uppgift 1501g-ht17”.

Syftet är att få en förståelse för hur schemaläggningen i ett realtidsoperativsystem fungerar och hur man kan kontrollera funktionen med hjälp av oscilloskop/logikanalysator.

Det är ganska många foton som skall in, försök begränsa storleken på bilderna, blir jobbig att hantera på Its learning annars (de bilder jag lagt in är bara 12-13 kbyte vardera).

**Minnesåtgången**

Hur mycket minne behövdes för fallet utan FreeRTOS installerat?

Hur mycket minne behövdes för fallet med FreeRTOS installerat?

**Uppmätning av tid för Context Switch**

Vi vill först förstå hur FreeRTOS växlar mellan två tasks som löper parallellt.

För det vill vi göra mätningar med oscilloskopet. FreeRTOS kan jobba preemptive dvs avbryta en task som är ”running” och släppa fram nästa task som står på kö om den har högre eller samma prioritet som den första.

Detta sker genom interrupt av ”systicks” som genereras av en speciell timer som finns i alla Cortex-M-kärnor.

Vi låter varje task ligga i en loop och toggla en pinne hög, låg, hög… hela tiden. Detta bör vi kunna se med oscilloskopet. Använd probar (x10) vid mätningarna.

Det är ganska svårt att få stilla stående kurvor men prova med tidbasen TB: 50us och klicka SINGLE

Klistra in foto här där man ser detta:

Om du noga studerar själva övergången mellan de båda tasken ser du att under en liten stund ligger båda pinnarna stilla. Det är den tid som FreeRTOS behöver för ”context switch”.

Ändra tidbasen och försök mäta tiden.

Uppmätt tid för context switch:

Ungefär hur många klockcykler svarar detta mot?

**Schemaläggningskoll**

Som syntes ovan kan det vara svårt med triggningen.

FreeRTOS har stöd för en funktion som heter

void vApplicationTickHook(void). Den anropas av RTOS vid varje tick om den är aktiverad i FreeRTOS.h, meningen är att man kan lägga in en kort kodsnutt som behöver exekveras periodiskt för varje tick. Normalt skall man vara försiktig med denna funktion, den stjäl ju tid från övriga tasks.

Vi vill använda den för att sätta en pinne hög och därefter låg. Då kan vi se exakt när varje tick kommer på oscilloskopet.

Gör detta genom att skapa c/h-filer med void vApplicationTickHook(void)med kod enligt ovan (eller om du tycker det är enklare – lägg funktionen direkt i main.c filen före main() ).

Välj en pinne som du kommer åt även med LCD-shielden monterad.

Glöm inte initiera ”tick-pinnen” som output i main().

Det behövs också en ändring FreeRTOS.h:

Ändra #define configUSE\_TICK\_HOOK från 0 till 1.

Anslut CH1 med prob till ”tick-pinnen” och trigga på den så du kan se ticks på den. Eftersom tickspulserna är väldigt korta har oscilloskopet lite svårt att hitta dem och de kan se olika höga ut beroende på hur samplingen råkar ske (oscilloskopen är ju digitala).

För att underlätta triggningen kan vi lägga in en fördröjning så att triggpulsen blir lite bredare.

T.ex så här:

volatile int j=0; /\* makes sure j doesn't overflow \*/

for (int i=0; i<10; i++) /\* The delay counter \*/

{

j++; /\* some easy predictable operation \*/

}

Anslut CH2 till pinnen som **task\_player1** togglar. Nu skall kan du se hur **task\_player1** kör varannan tick (Klicka SINGLE för att få det tydligt).

Som du ser blir det en hel del störningar, det beror på kapacitiv koppling mellan signalledarna (kallas ”crosstalk”eller ”överhörning” på svenska)

Foto

Men det vore ju bra att ha fler kanaler på oscilloskopet så vi kan följa fler tasks! Men kanalerna behöver ju inte vara analoga, de behöver bara kunna visa låg/hög signal.

Detta kan man göra med en logikanalysator. Våra oscilloskop har en logikanalysatortillsats (logik-pod) med 8 digitala kanaler som kan användas i stället för CH2.

Dags att prova den!

Tillsatserna har inte varit använda tidigare, det är därför några anslutningar som behöver göras.

Monteringsinstruktion, följ bildsekvensen:

 

 

 

Inkopplingsinstruktion:

Anslut logik-poden till oscilloskopet



Klicka på knappen POD. CH2 kommer nu att ersättas med 8 signaler från logikpoden.



Längst till vänster står numren för varje digital kanal.

Om inte alla 8 kanalerna används är det ju onödigt att visa dem. När POD-knappen lyser och ingen annan meny är vald kan man välja vilka av de 8 kanalerna som skall visas med menyn längst till höger på skärmen.



Välj kanal med upp- och nerpilarna och toggla på/av med knappen mellan pilarna.

Vi behöver just nu bara två kanaler t.ex. 1 och 2, koppla bort övriga.

Klicka nu på menyknappen längst ner (0/7) så markeringen lyser blått. Nu kan ändra positionen på skärmen med den vanliga POSITION-ratten och även ändra spänningsskalan med VOLTS/DIV-ratten.

De visade signalerna är helt digitala, dvs låga eller höga, men de verkliga signalerna är ju egentligen analoga. Således måste det finnas en spänningsnivå som avgöra om signalen skall visas som hög eller låg. Den bör vara olika beroende på kretsfamilj (TTL, CMOS …) och elektronikens matningsspänning och kan därför ställas in. Klicka på MENU (när POD lyser).

Nu får du upp en meny där du kan välja omslagsnivå. Välj lämplig omslagsnivå…

Dags att mäta igen.

Behåll prob 1 ansluten till ”tick-pinnen” och anslut kanal 1 och 2 på logikpoden till pinnarna som svarar på de båda tasken **task\_player1** respektive **task\_player2**. Glöm inte att även ansluta jord (svarta sladden) till GND på Due-kortet (kan t.ex. göras på stift på ovansidan av LCD-shielden).

Justera oscilloskopet så du tydligt kan se både ticks (på CH1) och hur de två tasken skiftar att köra.

(eventuellt kan du behöva ändra omslagsnivån )

Foto:

Framöver vill vi gärna kunna visualisera schemaläggningen även för tasks som inte bara håller på att toggla en pinne hela tiden utan utför något vettigt och dessutom kan ibland oscilloskopet (som ju samplar signalen) ge lite märkliga effekter när man har snabba signaler.

Vi kan göra detta genom låta bli att lägga signalerna låga i våra tasks (dvs lägga ut hög nivå hela tiden) och istället låta ticksfunktionen nollställa de båda pinnarna.

Alltså ändra koden och kör igen. Nu bör du kunna se hur signalerna går höga när tasken körs och nollställs vid tick.

Foto

Dags för lite experiment

Börja med att höja prioriteten för **task\_player1** från 1 till 2.

Testa!

Beskriv vad som händer:

Foto:

En mer realistisk situation är förstås att en task blir ”färdig”. Låt oss simulera programmet som skall köras med en delay.

Ändra koden för **task\_player1** så här:

/\* kod som sätter pinnen hög här \*/

volatile j;

j=0;

for (int i=0; i < 100; i++)

{

j++;

/\* kod som sätter pinnen hög här, behövs om tasken skulle avbrytas av tick \*/

}

/\* Här tänker vi oss att tasken är ”färdig” så nu behövs

kod som sätter pinnen låg här \*/

vTaskDelay(1);

Idén är att tasken skall bli färdig och sen vänta tills nästa tick innan den körs igen (behåll prioriteten 2 för **task\_player1**).

Beskriv vad som händer (ange speciellt taskens period)

Foto:

Prova att ändra till vTaskDelay(4) och se vad som händer.

Beskriv vad som händer (ange speciellt taskens period):

Foto:

Behåll vTaskDelay(4) men öka delayräknaren från 100 till 1000.

Hur stor blir nu taskens period?

Foto:

I samband med styr- och reglersystem och signalbehandling behövs det att en task säkert har kontant periodtid oberoende av taskens exekveringstid ( så länge den är mindre än periodtiden förstås).

Ta bort anropet till vTaskDelay(4). Lägg in följande deklarationer i början på **task\_player1** :  
 portTickType xLastWakeTime;

const portTickType xTimeIncrement = 4;

xLastWakeTime = xTaskGetTickCount();

och börja den yttre while-loopen med

vTaskDelayUntil(&xLastWakeTime, xTimeIncrement);

Läs om vTaskDelayUntil()i <https://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V9.0.0.pdf>.

TickCount är en variabel som startar på 0 när FreeRTOS startar och sen räknas upp för varje tick. Den är initierad (i FreeRTOS.h) som en 32-bitars variabel. Den kommer därför att ge overflow efter 232 ticks.

Räkna ut hur lång tid det dröjer tills det blir overflow i vårt fall:…

Kör programmet och beskriv vad som händer:

Lägg in foto:

Prova gärna (inte obligatoriskt) till slut även att ändra **task\_player2** så den blir periodisk på samma sätt som **task\_player1**. Låt den task som har kortast period ha högst prioritet.

Lägg gärna in foto:

Detta är alltså ett sätt åstadkomma schemaläggning med bestämda perioder för tasks. Det är inte säkert du behöver utnyttja det fullt ut i resten av labben men troligen i examinationsprojektet och i P2 i lp4 kommer du absolut att behöva det för att få till periodiska reglerloopar…

Nu bör du har förstått hur schemaläggningen fungerar är det dags för den ”riktiga” programmeringsuppgiften som finns i Instruktioner - uppgift 1501g-ht17.

Använd gärna logik-poden för att se vad som händer!

(För övrigt finns fler funktioner som kan vara användbara i andra sammanhang, t.ex. finns det möjlighet att trigga på kombinationer av de digitala signalerna… Se manualen.)

Lämna in denna del på Its learning i pdf-form in när du lämnar in programmeringsuppgiften på github (även en video skall lämnas in, se Instruktioner - uppgift 1501g-ht17).

Lämna gärna synpunkter på labben här: