



## Enchipsdatorer – Laboration 3 Klocka och Knapp

# Enchipsdatorer – Laboration 3 Klocka och Knapp

## Målsättning

- Skriva en terminalmeny med sifferval.
- Sätta upp en timer med interrupt.
- Avstudsas en knapp med mjukvara.
- Programmera en klocka med hjälp av en timer och interrupts.
- Få dioder att blinka.

## Förberedelse

Läs på om interrupt, kontaktstuds och hårdvarutimers (TIM-enheterna). Besvara frågorna i kap. 3.1 och 3.2 om elektrisk karaktäristik.

## Examination

Resultat för laborationen presenteras muntligt för en laborationshandledare under laborationstillfället. Svar på frågor i **fet stil** ska besvaras under redovisningen. Krav på kodstil måste följas.

## Genomförande

Laborationen har 4 timmar handledd tid, men kan ta mer tid att genomföra, vilket då görs på egen hand.

Det är viktigt att kodningsarbetet sker individuellt. Det är okej att diskutera problemlösning med andra men det är absolut förbjudet att kopiera kod från andra. De lösningar som tas fram skall föras med bra kommentarer, korrekt indentering och bra variabelnamn. Koden ska m.a.o. vara lätt att läsa och förstå.

## Hårdvara

- 1x Kopplingsdäck
- Kopplingskablar
- 1x Fyrsiffrig sjusegmentsdisplay (drivkrets TM1637)
- 1x Tryckknapp
- 1x Motstånd
- 1x Prob (finns i labbsalar och/eller lånas)

## Enchipsdatorer – Laboration 3 Klocka och Knapp

### 1. Introduktion

Denna laboration har två delar: Kontaktstuds samt klocka driven av hårdvarutimer. Kontaktstuds är helt klart den svårare av de två. Ett tips kan vara att börja med kontaktstuds och därmed hinna få hjälp med det som lär ge mest problem. Men det är helt upp till dig hur du angriper problemen.

Nästa kapitel hjälper dig sätta upp hårdvaran. Se till att detta fungerar innan du ger dig in på laborationsuppgifterna.

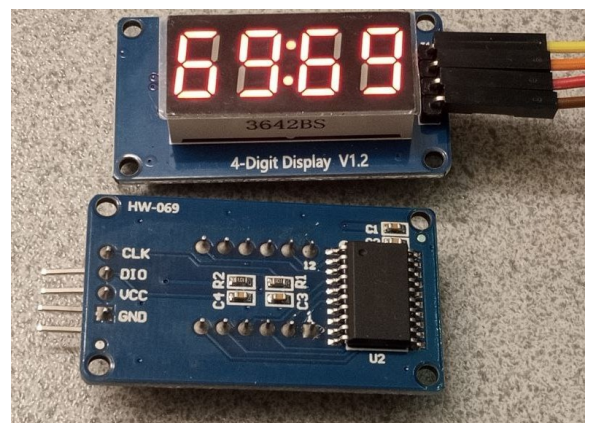
### 2. Display och meny

#### 2.1. Fyrsiffrig sju-segmentsdisplay

Tag fram din fyrsiffriga sju-segmentsdisplay av den modell som ses i bilden.

VCC för denna enhet ska vara kopplad till +5V.

Öppna CubeMX. Välj två ben att ansluta DIO (Data Input/Output) och CLK till. Sätt dessa som GPIO\_Output. Döp dem till SEG\_DIO och SEG\_CLK.



Styrkretsen (TM1637) på enheten ska ibland skriva på DIO. Läsning fungerar endast om porten på din STM32 är satt i rätt läge. Öppna GPIO-inställningarna för det ben du döpte till SEG\_DIO. Ändra dess GPIO mode till Output Open Drain. Se bilden.

Pin N...	Signal o...	GPIO ou...	GPIO m...	GPIO Pu...	Maximu...	User Label	Modified
PA5	n/a	Low	Output P...	No pull-u...	Low	SEG_DIO	✓
PC2	n/a	Low	Output ...	No pull-u...	Low	SEG_CLK	✓
PC3	n/a	Low	Output P...	No pull-u...	Low	SEG_CLK	✓
PC13-A...	n/a	n/a	External ...	No pull-u...	n/a	B1 [Blue...	✓

PC2 Configuration :	
GPIO output level	Low
GPIO mode	Output Open Drain

Spara och generera kod.

Ladda ner filen `quad_sseg.zip` från Canvas och lägg till dess innehåll till ditt projekt. Se instruktioner från förra laborationen för detaljer om hur.

För att testa om enheten fungerar kan denna kod köras:

## Enchipsdatorer – Laboration 3 Klocka och Knapp

```
for (int i = 0; i < 10; i++)  
{  
    uint32_t dly = 250;  
    qs_qs_put_big_num(i);           HAL_Delay(dly);  
    qs_put_digits(i, i, i, i, 0);   HAL_Delay(dly);  
    qs_put_digits(i, i, i, i, 1);   HAL_Delay(dly);  
}
```

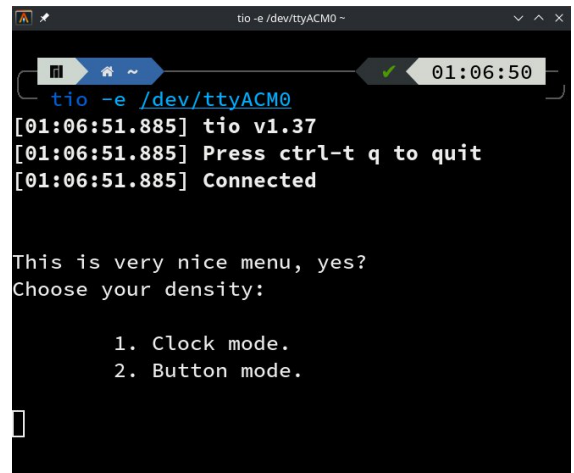
I denna laboration bör du att använda `qs_put_digits()` och `qs_put_big_num()`.  
Läs i `quad_seg.h` för detaljer kring dessa.

### 2.2.Serieanslutningsmeny och menyval

Skriv en funktion `uart_print_menu()` som skriver ut en meny, inte helt olik den i bilden till höger. Anropa den innan main-loopen.

Använd putty (eller valfritt program, i bilden används tio) för att ansluta till ditt kort och bekräfta att det blir snyggt.

Att läsa in text över UART är lite bökigt, eftersom man måste veta hur många bytes man vill läsa när man gör funktionsanropet. Vi gör det lätt för oss och begär endast ett.



The screenshot shows a terminal window titled 'tio -e /dev/ttyACM0 ~'. The prompt is 'tio v1.37'. The user has entered 'Press ctrl-t q to quit' and the terminal has responded with 'Connected'. Below this, the text 'This is very nice menu, yes?' is displayed, followed by 'Choose your density:'. A list of two options is shown: '1. Clock mode.' and '2. Button mode.'. The cursor is positioned at the end of the list.

På samma sätt som det finns `sprintf()` som ett alternativ till `printf()` så finns också `sscanf()` som alternativ till `scanf()`. Den här funktionen läser in en tangentbordstryckning och ger tillbaka den siffra som tryckts in, eller `-1` (som felkod) om ingen siffra kunde läsas.

```
int uart_get_menu_choice()  
{  
    char str[1] = { '\0' };  
    uint16_t str_len = 1;  
    HAL_UART_Receive(&huart2,  
                    (uint8_t *) str,  
                    str_len,  
                    HAL_MAX_DELAY);  
    int ret = -1;  
    sscanf(str, "%d", &ret);  
    return ret;  
}
```

## Enchipsdatorer – Laboration 3

### Klocka och Knapp

Skriv även en funktion `uart_print_bad_choice()` som säger åt användaren att deras inmatning är felaktig.

Här är main-loopen för ditt program. Dock utan USER CODE-kommentarer, så var försiktig:

```
while (1)
{
    uart_print__menu();
    int menu_choice = uart_get_menu_choice();
    {
        switch (menu_choice)
        {
            case 1:          clock_mode(); break;
            case 2:          button_mode(); break;
            default: uart_print_bad_choice(); break;
        }
    }
}
```

Här är skelett för de två funktionerna som du kan klistra in så att allt laddar.

```
void clock_mode()
{
    /*** init segment ***/
    /*** main loop ***/
    while (1)
    {
    }
}
```

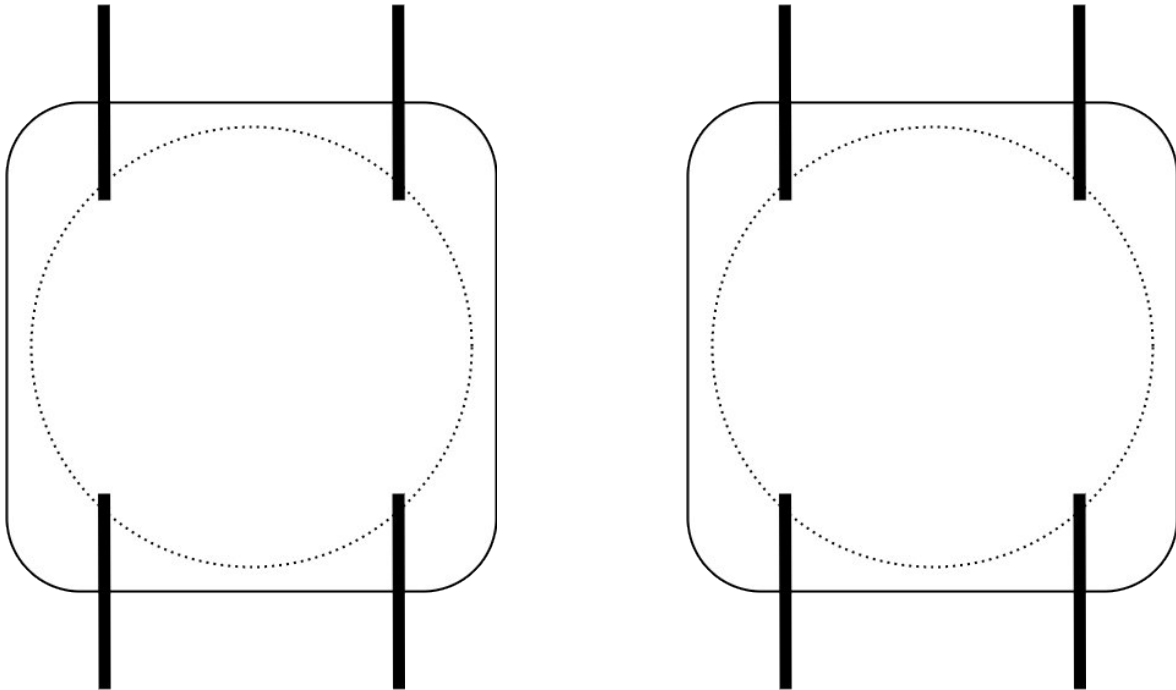
```
void button_mode()
{
    /*** init segment ***/
    /*** main loop ***/
    while (1)
    {
    }
}
```

## Enchipsdatorer – Laboration 3 Klocka och Knapp

### 3. Tryckknapp

#### 3.1. Fysiska egenskaper

Rita hur kopplingarna är gjorda inne i din tryckknapp. Ena bilden när knappen inte trycks och den andra när den hålls nere. Använd din multimeter (pipfunktion) för att undersöka.



Gör **EJ** denna anslutning! **Ditt kort kommer att gå sönder!** Detta är en teoretisk fråga:

Tänk dig att dessa kopplingar görs:

Nedre vänstra benet till VDD.

Nedre högra benet till GND.

Övre vänstra benet till en GPIO-port.

Sedan trycks knappen ned.

**Vad kommer då att hända, elektriskt?**

## Enchipsdatorer – Laboration 3 Klocka och Knapp

### 3.2.Pull-up

Öppna *referensmanualen*. Läs kapitel 8.3.8 (External interrupt/wakeup lines), kapitel 8.3.9 (Input configuration) samt titta på figur 18 som finns i kapitel 8.3.9.

**Vilken buss är det som styr över hur snabbt input-värden kan ändras?**

Öppna fliken Clock Configuration i CubeMX och titta bland siffrorna längst till höger.

**Med vilken klockfrekvens kan din MCU märka förändringar i input-värden?**

Titta i figur 18 (Input floating/pull up/pull down configurations) i *referensmanualen*. Där kan du se att pull-up och pull-down motstånd är valbara eftersom det står "on/off" intill dem. För denna laboration ska du ha båda dessa avstängda. Istället ska ett externt motstånd användas.

Öppna *databladets*. Slå upp kapitlet "I/O port characteristics", ett underkapitel till "Electrical characteristics". När en port är konfigurerad som input så kommer Schmitt-triggern att läcka en viss ström.

**Vad är högsta läckströmmen in i en input-port?**

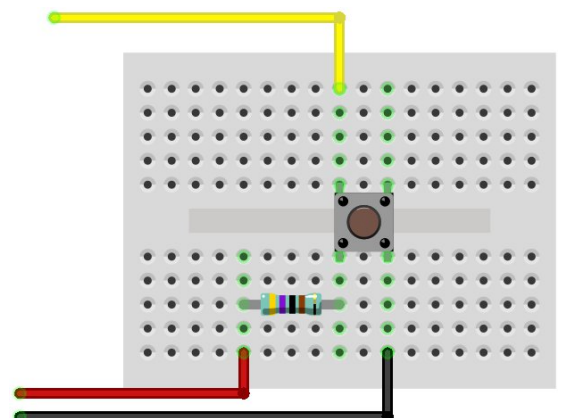
MCU:n kommer med möjligheten att slå på/av interna pull-up och pull-down-motstånd för I/O-portar.

**Vad är det typiska värdet på en ports interna pull-up-motstånd?**

**Vad är det typiska värdet på en ports interna pull-down-motstånd?**

Man vill att så lite ström som möjligt ska gå mellan VDD och GND när knappen trycks in. Men är motståndet för stort så kommer signalen från knappen bli fördröjd. Detta eftersom det också finns kondensatorer i kretsen.

Koppla enligt bilden till höger. Röd till 3.3V, svart till GND och gul till GPIO. Ditt motstånd ska vara mellan 1 k $\Omega$  och det som är typiskt för en GPIO-ports interna styrka.



## Enchipsdatorer – Laboration 3 Klocka och Knapp

### 3.3.CubeMX

Öppna CubeMX och sätt GPIO-portens läge till GPIO\_EXTIx<sup>1</sup> och ge den ett passande namn.

Öppna GPIO-menyn och ställ in den på passande flanktriggning. Du vill bara ha en interrupt när knappen trycks ner, inte när den släpps. Sätt inställningen "GPIO Pull-up/Pull-down" till "No pull-up and no pull-down".

Pin Na...	Signal on ...	GPIO outp...	GPIO mode	GPIO Pull...	Maximum ...	User Label	Modified
PA5	n/a	Low	Output Pu...	No pull-up...	Low	LD2 [Gree...	✓
PC0	n/a	n/a	External I...	No pull-up...	n/a	MY_BTN	✓
PC2	n/a	Low	Output Op...	No pull-up...	Low	SEG_DIO	✓
PC3	n/a	Low	Output Pu...	No pull-up...	Low	SEG_CLK	✓
PC13-AN...	n/a	n/a	External I...	No pull-up...	n/a	B1 [Blue ...	✓

PC0 Configuration

GPIO mode: External Interrupt Mode with Falling edge trigger detection

GPIO Pull-up/Pull-down: No pull-up and no pull-down

User Label: MY\_BTN

Gå även in i fliken NVIC och bocka för så knapptryckning genererar en interrupt.

GPIO	Single Mapped Signals	RCC	SYS	USART	NVIC
NVIC Interrupt Table					
				Enabled	Preemption Priority
					Sub Priority
EXTI line0 interrupt				<input checked="" type="checkbox"/>	0
EXTI line[15:10] interrupts				<input type="checkbox"/>	0

Spara, generera kod, kompilera och ladda över koden på ditt kort. Du vill ha rätt inställningar för GPIO-benet i och med testerna i nästa del.

<sup>1</sup> Siffran efter EXTI är den port som interrupten kommer komma från. De 16 olika portarna får bara förekomma en gång. Det betyder att bara en av tex PA5, PB5, PC5 etc. kan generera interrupts.



## Enchipsdatorer – Laboration 3 Klocka och Knapp

### 3.4. Kontaktstuds

(dessa instruktioner är för Tektronix TBS 2000) Hämta en prob<sup>2</sup> och starta ett oscilloskop. Koppla in proben så att du ser vad som händer på GPIO-porten knappen sitter på. Glöm inte att du kanske måste justera probinställningen på oscilloskopet så att spänningsnivån stämmer.

Tryck på knappen ACQUIRE. En meny ska komma upp. Sätt RECORD LENGTH till 20k.

Justera skalorna (spänning och horisontell) så att du tydligt kan se när knappen är intryckt och ej.

Tryck på knappen MENU under TRIGGER (uppe till höger). Inställningarna bör vara:

- Type – Edge
- Channel – bestäm själv
- Coupling – DC
- Slope – fallande
- Level – se nedan
- Mode – Normal

Öppna *databladet* för din MCU. I kapitlet Electrical Characteristics finns en tabell & en figur ("I/O static characteristics", "FT/TC I/O input characteristics"). Oscilloskopets Level-värde ska vara den spänning som räknas som en nolla då VDD är 3.3V.

När RUN/STOP lyser grönt kan du börja trycka på din tryckknapp. Justera SCALE för din kanal så att du tydligt ser när signalen är hög och låg.

Sätt horisontell SCALE till 400 ns. **Vad är högsta antalet flanker du kan observera från första till ett stabilt värde är uppnått?**

Sätt horisontell SCALE till 4 ms. **Vilken är den längsta tiden<sup>3</sup> från första flanken till sista flanken, innan stabilt värde infinns?**

**Uppkommer kontaktstuds vid fler tillfällen än när du trycker ner knappen?**

---

<sup>2</sup> Om ingen handledare finns som kan låna ut en prob till dig just nu så brukar det hänga svarta prober (med lite gröna detaljer på) i labbsalarna. Leta på arbetsbänkarna och bland sladdarna.

<sup>3</sup> En snabb tryckning håller ner knappen i c:a 50 ms.



## Enchipsdatorer – Laboration 3 Klocka och Knapp

### 3.5.Problembeskrivning och krav för lösning

Lägg till två variabler:

```
uint16_t button_exti_count;  
uint16_t button_debounced_count;
```

Den första ska hålla antalet gånger interruptrutinen/callback-funktionen körts och den andra håller hur många gånger användaren upplever att de tryckt på knappen. EXTI är kort för EXTernal Interrupt (eller "EXTended Interrupt").

Detta problem kan lösas genom att ha en kondensator intill knappen. Men om du heter Texas Instruments och ska tillverka 2 miljoner miniräknare i månaden, alla med 50 knappar styck, då är 100 miljoner kondensatorer i månaden (i flera år) dyrare än lite programmerarlön.

Lägg till interrupt/callback-funktionen och kör (bakom en kontroll att du är på rätt GPIO-port) `button_exti_count++`; i den.

Här är lite kod till `button_mode()`:

```
int b1_pressed;  
  
while (1)  
{  
    /* deal with debouncing the button... */  
    if (1) // For initial testing  
    {  
        button_debounced_count = button_exti_count;  
    }  
  
    // check b1 button (on board, active low)  
    b1_pressed = GPIO_PIN_RESET  
                == HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);  
    qs_put_big_num(b1_pressed ? button_exti_count  
                      : button_debounced_count);  
}
```

Bekräfta att detta visar antalet knapptryckningar (med studsar) på displayen.

Som du förhoppningsvis märkte när du använde proben så kan du få flanker både när du trycker och när du släpper. En icke godkänd (men fungerande) lösning, som även tar hänsyn till flankerna vid släpp är denna:

### Enchipsdatorer – Laboration 3 Klocka och Knapp

```
if (unhandled_exti) // was set in interrupt
{
    HAL_Delay(BOUNCE_DELAY_MS); // value measured by probing
    pressed = GPIO_PIN_RESET
            == HAL_GPIO_ReadPin(MY_BTN_GPIO_Port, MY_BTN_Pin);
    if (pressed)
    {
        button_debounced_count++;
    }
    unhandled_exti = 0;
}
```

`HAL_GPIO_ReadPin()` används efter fördröjningen eftersom du behöver säkerställa hurvida flanken kom vid nedtryckning eller släpp av knappen.

Kör lösningen given i detta kapitel. Den gör rätt på allt utom de två första punkterna i kravlistan. En giltig lösning som följer alla kodkrav skiljer sig på millisekund-nivå från vad du blivit given här.

Kodkrav:

- Hänsyn ska tas till längsta tiden som mättes upp med oscilloskopet.
- Loopen får inte ha någon fördröjning.
- Det får **ALDRIG NÅGONSIN** förekomma fördröjningar i en interrupt. Denna regel gäller för **ALLA** interrupts, oavsett sammanhang.
- Koden ska bara registrera när knappen trycks ner, aldrig när den släpps.
- När B1 hålls in ska antalet studsar visas. När B1 är släppt ska avstudsade tryckningar visas.

Bonus:

- Avläsning ska göras `BOUNCE_DELAY_MS` efter *första* flanken av ett tryck/släpp och inte efter *sista* flanken.

Nästkommade sidor ger dig hjälp att lösa problemet. Försök komma så långt som möjligt på egen hand!

Godkänd knapp:

## Enchipsdatorer – Laboration 3

### Klocka och Knapp

### 3.6.Lösningshjälp

Lösningen måste ta hänsyn till att rätt tid har passerat innan det är dags att göra en `HAL_GPIO_ReadPin()`. Eftersom fördröjningsfunktioner är förbjudna så ställer detta till problem. En riktlinje<sup>4</sup> för vad som måste göras kan skrivas så här:

```
while (1)
{
    if ( is_there_unhandled_exti() )
    {
        if ( has_enough_time_passed(BOUNCE_DELAY_MS) )
        {
            b = read_button();
            handle_button(b);
        }
    }

    /* drive output etc. */
}
```

Så på något sätt måste tid mätas. Börja med att se över interrupten. I stället för att bara flagga *att* en interrupt har hänt, som med `HAL_Delay()`-lösningen på föregående sida...

```
if (GPIO_Pin == MY_BTN_Pin)
{
    button_exti_count++; // for raw data
    unhandled_exti = 1;
}
```

...så kan man vara lite smartare och spara undan *när* den hänt:

```
if (GPIO_Pin == MY_BTN_Pin)
{
    button_exti_count++; // for raw data
    last_flank_causing_exti = HAL_GetTick();
}
```

Vad ska `last_flank_causing_exti` jämföras med? Om `now` är systemets nuvarande tick-värde så borde denna rad finnas med:

```
if ( (now - last_flank_causing_exti) >= BOUNCE_DELAY_MS )
```

---

<sup>4</sup> Funktionsnamnen här är semantisk hjälp. Det *går* skriva en lösning så här, men det kommer att kräva rätt många globala variabler, vilket inte alltid är helt bäst. En snygg lösning håller sina variabler så lokala som möjligt.

## Enchipsdatorer – Laboration 3 Klocka och Knapp

Men om den jämförelsen får köra hela tiden så kommer koden registrera tryckning *varje* loop efter att `BOUNCE_DELAY_MS` gått. På något sätt måste koden ha koll på om `last_flank_causing_exti` blivit hanterad. Kanske genom att kontrollera så att den inte är samma som den senaste hanterade flanken?

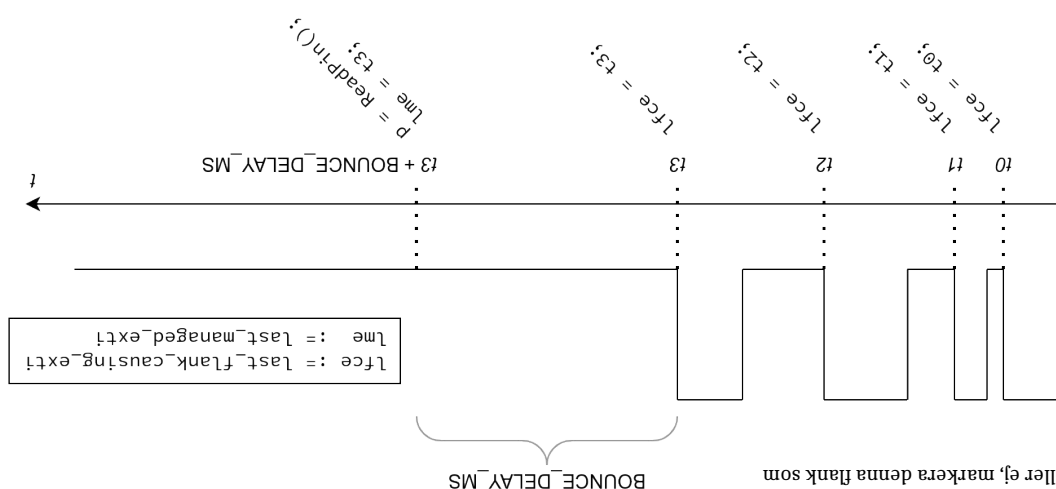
Skapa en variabel som ska hålla det sista hanterade ticket då en EXTI skedde, `last_managed_exti` är ett bra namn. Med denna borde du kunna göra en test likt den i `HAL_Delay()`-lösningen. I den användes raden

```
if (unhandled_exti)
```

Men du behöver här ha ett test som bygger på `last_flank_causing_exti`.

Nedan är mer eller mindre hela lösningen. Den använder sig av tipsen givna här. Om du fortfarande sitter fast, tveka inte att be en labbassistent om hjälp.

**LÄS ENDAST OM DU VERKLIGEN KÖRT FAST:**



Om `last_flank_causing_exti != last_managed_exti` →  
Om ohanterad, har studstiden gått? →  
Om tillräcklig tid gått, är knappen intryckt? Dvs. Uppdatera  
endast `button_debounce_count` när knappen tryckts in, ej  
när den släpps. →  
Oavsett om den är intryckt eller ej, markera denna flank som  
hanterad.

## Enchipsdatorer – Laboration 3

### Klocka och Knapp

## 4. Klocka

### 4.1. Välj en timer

Öppna *databladet* för din MCU och gå till kapitlet om timers och watchdogs. Där hittar du en tabell med de timers som just din MCU kommer med. Studera den lite kort. Du kanske märker att vissa siffror "saknas". Det är t.ex. så för STM32, om du har enheterna TIM6 eller TIM7 så faller de i kategorin "Basic timer". Din MCU kommer inte med någon timer av den typen. Du har endast timers av typerna "Advanced" och "General purpose".

För just denna uppgift gör det inte mycket skillnad vilken du väljer (alla klarar uppgiften lika väl) men du måste välja en.

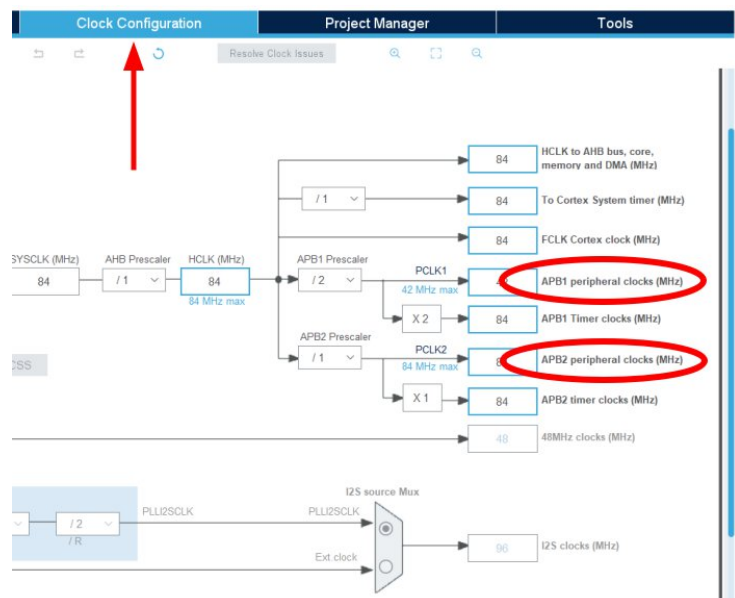
**Vilken timer valde du?**

**Vad är dess högsta värden för räknaren (Counter resolution) samt för Prescalern?**

Tabellen innehåller den maximala frekvensen som timern klarar. Men den kanske inte är konfigurerad att köra på högsta frekvensen. Detta måste undersökas.

Öppna Clock Configuration i CubeMX och titta bland klockfrekvenserna till höger, som i bilden. Det finns mer än en buss som driver timers!

Öppna kapitlet Memory Mapping i databladet.



**Vilken buss driver den TIM-enhet som du valde?**

**Vad för klockfrekvens (för timers) rapporterar CubeMX att denna buss har?**

## Enchipsdatorer – Laboration 3 Klocka och Knapp

### 4.2. Beräkningar

Här beskriver  $f_{TIM}$  frekvensen som timern ”slår runt” med. Detta är hur ofta dess associerade callback-funktion anropas.

Om vi har att  $p = PSC + 1$  och  $n = ARR + 1$

**Vad är det matematiska sambandet mellan dessa fyra?**

$n$        $p$        $f_{TIM}$        $f_{BUS}$

För denna uppgift ska du ha  $f_{TIM} = 2 \text{ Hz}$ .

Med just *denna* uppgift så gör det inte någon skillnad på om  $n$  är stor och  $p$  är liten, eller vice versa. Många gånger är det viktigt, men detta är ej ett sådant tillfälle.

**Vilka värden ska du välja för att uppnå detta?**

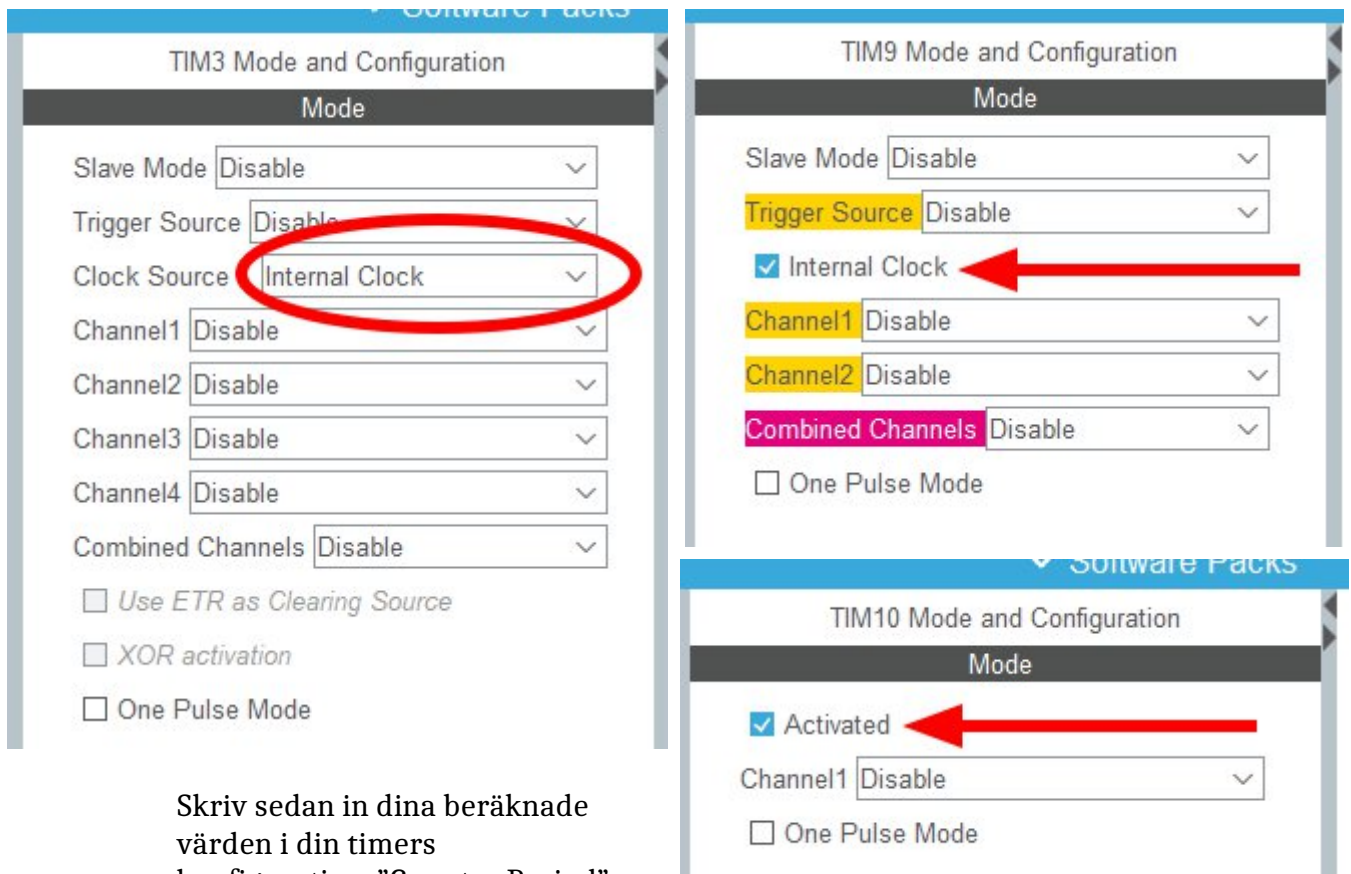
$$p = PSC + 1 =$$

$$n = ARR + 1 =$$

## Enchipsdatorer – Laboration 3 Klocka och Knapp

### 4.3.CubeMX

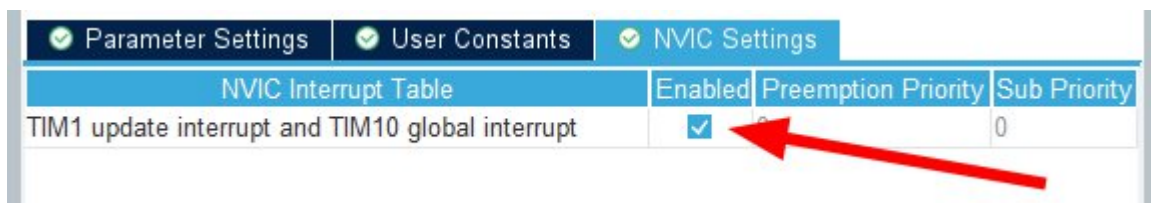
Beroende på vilken TIM-enhet du valde så ser aktiveringen lite olika ut. Här är tre olika sätt det kan se ut på:



Skriv sedan in dina beräknade värden i din timers konfiguration. "Counter Period" och "Auto Reload Register" är samma sak.

Du kan skriva in  $-1$  när du matar in dina siffror. Så om du har  $p = 200$  så kan du skriva  $200-1$  i fältet för Prescaler.

Öppna sedan fliken för NVIC och bocka för TIMx global interrupt. Bilden visar hur det ser ut för TIM10.







## Enchipsdatorer – Laboration 3 Klocka och Knapp

### 4.4. Programmera lösningen

Skriv din lösning. Du kommer att behöva använda funktioner som nämns i Timer-delen av `JU_STM32_HAL-beskrivning.pdf` för att kunna lyckas. En funktion som startar timern och en annan för callback-funktionen.

Krav på lösningen:

- Kolon-dioderna ska tändas och släckas varje halv sekund. Den ska alltså vara tänd ena halvan av en sekund och släckt den andra halvan.
- Klockan ska ej ha drift<sup>5</sup>.
- Vanligtvis visas `MM:SS` på displayen.
- Om kortets blå knapp hålls nedtryckt så ska `HH:MM` visas.
- Den ska klara överslag från halvsekundshändelser till sekund, sekund till minut, minut till timme och timme till dygn. Dygn behöver inte sparas. D.v.s. när klockan varit `23:59:59` så ska den nästa sekund vara `00:00:00`.
- Inga fördröjningsfunktioner är tillåtna.

Ett tips för att lätt testa funktionaliteten är att i initieringsdelen av `clock_mode()` sätta starttiden till `23:59:45`.

Godkänd klocka:

---

---

<sup>5</sup> Din lösnings sekunder ska vara lika snabba som andra sekunder. Varken mer eller mindre. Använd tidtagarur och verifiera över 20 sekunder om du är osäker.