



## Enchipsdatorer – Laboration 1 Tärning

# Enchipsdatorer – Laboration 1 Tärning

## Målsättning

- Bli bekant med utvecklingsmiljön CubeIDE.
- Lära sig sätta upp IO-ben i CubeMX.
- Förstå användningen av register i mikroprocessor-programmering.
- Öva på att sätta upp en genomtänkt main-loop.
- Få dioder att blinka.

## Förberedelse

Läst på det som föreläsning 1-4 tar upp. Det rekommenderas att man läst kursboken enligt läsanvisningarna, vilket finns i slutet på föreläsningsanteckningarna.

Laddat ner datablad och referensmanual för MCUn från Canvas. Svarat på frågorna i kapitel 2.1 (Manualbladsläsning).

## Examination

Resultat av laborationen presenteras muntligt för en laborationshandledare under laborationstillfället. Svar på frågor i **fet stil** ska ha besvarats 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 förses med bra kommentarer, korrekt indentering och bra variabelnamn. Koden ska m.a.o. vara lätt att läsa och förstå.

## Hårdvara

- 1x kopplingsdäck
- Kopplingskablar (hona-hane)
- 7x lysdioder med passande motstånd
- 1x 7-segments-display med passande motstånd

## Enchipsdatorer – Laboration 1 Tärning

### Introduktion och överblick

Denna laboration går ut på bygga en elektronisk tärning som rullas av att du håller nere en knapp. Du kommer stegvis att bygga dig till en fungerande lösning.

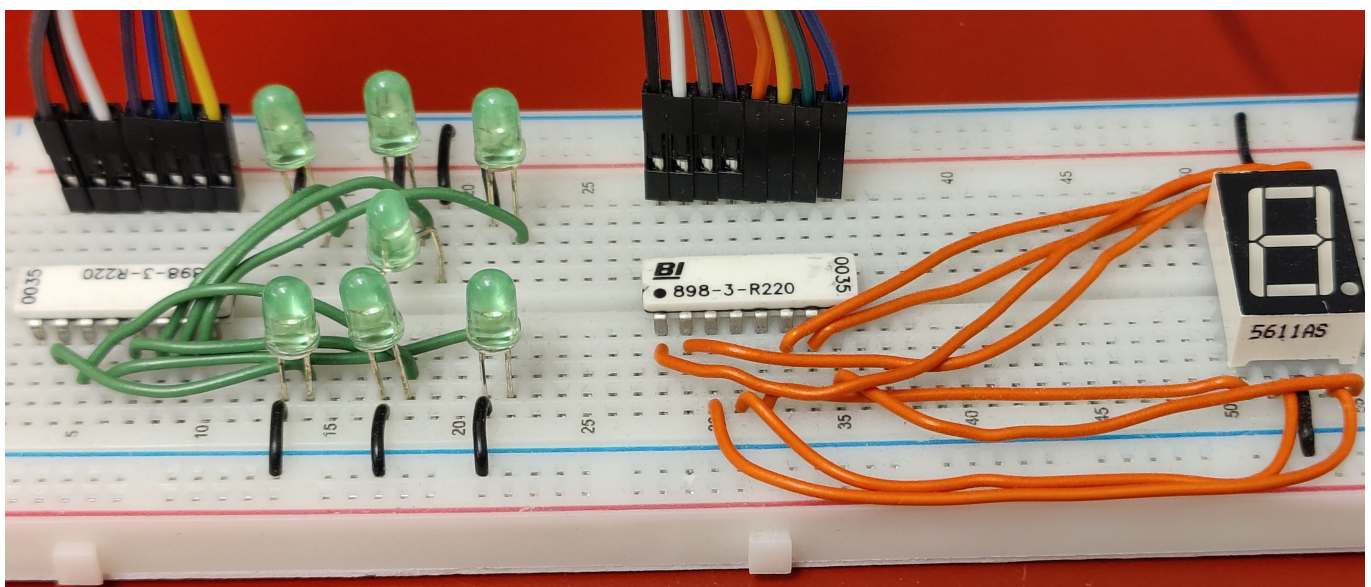
I första delen så kommer du att få planera kopplingar samt få titta i genom databladerna för att lära dig om register och minnesareor. Du får hjälp att hitta relevant information ur ett dokument på hundratals sidor. Sedan en kort kodningsuppgift som använder sig av saker du slagit upp.



Nästa del är att bygga tärningen som representeras av sju stycken dioder. Dessa sju ska vara utplacerad på ett kopplingsdäck så att det efterliknar prickarna på en tärning. Detta är vad du ser till vänster i bilden inunder.

I sista delen så ska en sju-segmentsdisplay användas för att visa upp vilken siffra som är på tärningen. Detta ses till höger längst ner på denna sida. Till detta behöver några bitmaskar förberedas.

Allra sist finns en övning om hur stora tal (större än 8 bitar) lagras i minnet på en dator (endianness). Denna övning är frivillig men rekommenderas.



## Enchipsdatorer – Laboration 1 Tärning

### 1. Komma igång

#### 1.1. Skapa projekt & Git

Starta ett nytt projekt på samma sätt som du gjorde i laboration 0: Lägg det intill den mapp som Laboration 0 är i, och utan knepiga tecken. Dvs. du bör skriva "tarning" och inte "tärning". Missa inte att du ska välja ditt kort med "Board Selector" och *inte* med det som öppnas som standard. Sparade du ditt kort som en favorit under laboration 0 så kan du hitta den om du klickar på stjärn-ikonen.

Om du vill så kan du nu göra en commit så fort CubeMX genererat koden. Commit brukar göras mellan 2 och 20 gånger per dag under vanligt arbete. En commit är alltså inte något som bara görs när allt fungerar perfekt, utan något som görs när man känner att man gjort ett halv-stort framsteg. T.ex. när man känner att man fått ett delmoments inställningar i CubeMX rätt.

Att göra en push (synkronisera din kod med molnet - "origin" på git-språk) görs däremot bara 1-2 gånger per dag, så det kan vänta till slutet av laborationstillfället.

#### 1.2. Innan du kan rulla

För att kunna börja allokeras processorben är det bra om du först har en tydligare bild av din uppgift. Börja med att här rita upp en tärnings sju möjliga platser för prickar. Dvs. de sex prickarna för en sexa och en extra prick i mitten:

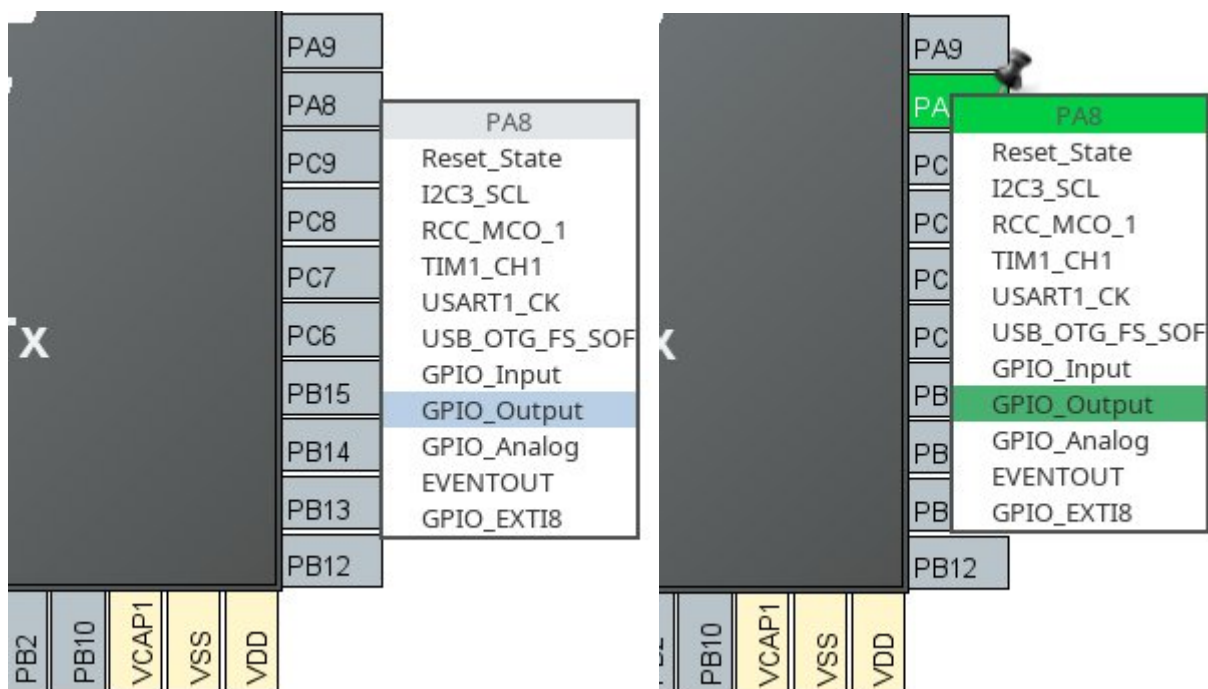
Namnge i din ritning varje prick med A..G. Ge namnen prefixet "DI\_" så att du får DI\_A, DI\_B, ... DI\_G.

## Enchipsdatorer – Laboration 1 Tärning

### 1.3.CubeMX

Nu ska du välja vilka ben som vilken diod ska vara ansluten till. Välj inga som heter "PCx" eftersom dessa ska användas i ett senare skede av labben. Vilka som helst som heter "PAx", "PBx" eller "PDx" kan du använda.

Om du t.ex. vill koppla PA8 till dioden DI\_A så klicka först på benet och välj sedan GPIO\_Output<sup>1</sup>. Bilden till vänster är hur det ser ut vid första klicket och till höger hur det ser ut om gjort rätt:



När du satt sju ben som GPIO\_Output så öppna konfigurationen för GPIO. Denna finner du under menyn "System Core" till vänster. Där kan du se en lista med de GPIO-ben som ska användas.

Om du klickar på något av benen i listan (ej PA5/PC13) så dyker det upp en liten konfigurationsruta. Det är olika inställningar du kan ha för ett GPIO-ben. För de sju benen du valt fyll i fältet "User Label" så att det matchar din ritning på föregående sida. Lägg till portnamnen i din ritning.

*Exempel:* Skriv "DI\_A" i fältet "User Label" för PA8. Skriv även "PA8" intill pricken DI\_A i din ritning.

Spara och generera kod när du är klar.

<sup>1</sup> När ett ben konfigureras i CubeMX så kommer systemets initieringsrutiner att konfigurera ett register som heter MODER - "GPIO port MODE Register".

## Enchipsdatorer – Laboration 1 Tärning

## 2. Hårdvaruregister – Användning och innehåll

### 2.1. Manualbladsläsning

Här kommer en liten övning i att jobba med datablad och CubeIDE menad att bekanta dig med din MCUs olika hårdvaruregister. Som nämndes på första sidan så krävs endast svar på frågor i **fet stil** ett krav för att bli godkänd.

Börja med att öppna *referensmanualen* för F411 och slå upp kapitlet "Memory organization".

1. **Ligger data lagrat i minnet enligt big endian eller little endian på denna ARM-processor?**

Det finns en bonusövning allra sist i det här dokumentet om endianness. Att ha gjort den gör att du mycket bättre kan förstå dig på koden för sjusegmentsdisplayen.

Titta nu i kapitlet direkt efter det förra: "Memory map".

2. Du har använt dig av utskrifter över UART och då användes enheten USART2. **Vad heter den buss som USART2 ligger på?**
3. **Vilken adress börjar GPIOC på?**
4. **Vilken adress slutar GPIOC på?**

Dessa frågor har sina svar i kapitlet om GPIO i *referensmanualen*. De första två frågorna kan besvaras enbart genom att titta i dokumentets innehållsförteckning.

5. **Vad betyder förkortningen IDR?**
6. **Vad betyder förkortningen ODR?**
7. **Hur många olika portkonfigurationer kan ett GPIO-ben<sup>2</sup> försättas i genom att skriva till MODER?**
8. **Vilket läge är vanligtvis skönsalternativet<sup>3</sup> vid konfiguration av MODER?**

---

<sup>2</sup> Orden "ben", "port" och "pin" används (lite slarvigt) synonymt för att referera till samma sak. Det är dock så att alla portar inte har ett fysiskt ben du kan koppla saker till.

<sup>3</sup> Vad som väljs åt dig om du inte väljer själv. Som engelskans "default".

## Enchipsdatorer – Laboration 1 Tärning

Skumma igenom de två punktlistorna i kapitlen "GPIO main features" och "GPIO functional description".

9. **Hur många I/O-portar kontrolleras av en GPIO-enhet?**
10. Titta i CubeMX, hur många ben kan du hitta som hör till GPIOD (de som börjar PD)? Hur många hittar du som hör till GPIOE?
11. Har du någon gissning om varför så mycket verkar saknas?<sup>4</sup>

Nu ska vi ta reda på vilka minnesadresser som vår kompillerade kod kommer att ligga på. När den läggs över så sparas den i vad som kallas "flash-minne".

12. Öppna upp *databladet* för F411 och gå till kapitlet "Memory mapping". **Mellan vilka två adresser<sup>5</sup> ligger flash-minnet?**

### 2.2. Kontrollera med din mikrokontroller

Titta i CubeMX. Om ett ben heter "PB12" så betyder det att det hör till GPIO-enheten GPIOB och har pin-numret 12.

**Skriv ned enhet och pin-nummer för kortets blå knapp (B1) och kortets gröna diod (LD2).**

Vi kommer även att behöva lite kod för att göra något vettigt med debuggern. Skriv in följande kod men ersätt i else-satsen variablernas värden med svaren ovan.

```
95  /* USER CODE BEGIN WHILE */
96  int pressed = 0;
97  while (1)
98  {
99      pressed = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
100      if (pressed)
101      {
102          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
103      }
104      else
105      {
106          GPIO_TypeDef* ld2_gpio = GPIOE; // wrong! replace with answer
107          uint16_t ld2_pin_nbr = 13; // wrong! replace with answer
108          uint16_t ld2_pin = 0x01 << ld2_pin_nbr;
109          HAL_GPIO_WritePin(ld2_gpio, ld2_pin, GPIO_PIN_RESET);
110      }
111  }
/* USER CODE END WHILE */
```

---

<sup>4</sup> Ledtråd 1: Titta i *databladets* kapitel "Ordering information". Vad betyder R i F411RE?  
Ledtråd 2: Jämför sidorna 35 och 36 av *databladet*.

<sup>5</sup> När du anger **adresser** så ska de skrivas så hela bitbredden uppvisas. Om det är ett 32-bitars tal så måste 8 tecken finnas med: 0x0007FFFF är rätt. 0x7FFFF är fel.

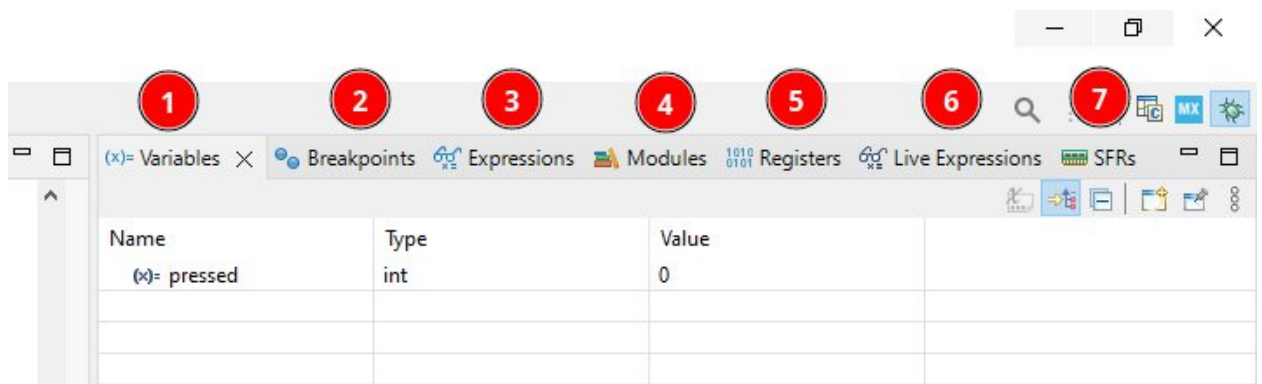


## Enchipsdatorer – Laboration 1

### Tärning

Starta en debug-session i CubeIDE. Koden ska vara pausad vid `HAL_Init()`. Tryck på Resume en gång och pausa sedan körningen igen. Till höger om koden byts filens "Outline" ut mot något annat. Expandera gärna ytan så att du lättare kan se bättre.

När koden är pausad så kommer vänster del av skärmen att ha en bild på anropshierarkin. Det kan vara så att i du i `main()` kallade på `WritePin()`. Då finns det en kontext som råder för `main()` och en annan för `WritePin()`. Prova att pausa koden och klicka mellan de olika funktionsnamnen när du går igenom punkterna nedan.



- 1) Detta är variabler som finns definierade där koden är just nu. Även funktionsargument kommer att dyka upp här. Om du kör och pausar så att du hamnar mitt i `ReadPin()` eller `WritePin()` så kommer det finnas saker här.
- 2) Punkter där debuggern kommer att pausa sin körning så fort den når. Högerklicka på *radnumret* till `int pressed = 0;` och välj "Toggle Breakpoint" för att se en punkt dyka upp här.
- 3) Lägg till egna uttryck som kan evalueras. Om du skriver in `pressed > 0` så kan du notera vad som händer när du klickar runt i anropshierarkin.
- 4) Används sällan/ej.
- 5) ARM-processorns interna register.
- 6) Går att använda för att inspektera globala variabler under körning. Du kan prova att flytta `int pressed = 0;` så att den blir en global variabel, lägg till "pressed", kör en debug-session och se vad som händer när du trycker på blå knappen.
- 7) Mikrokontrollerns alla hårdvaruenheter. Förkorningen betyder "Special Function Register".

## Enchipsdatorer – Laboration 1

### Tärning

Börja om debug-sessionen (kan göras med knappen "Terminate and Relaunch" till vänster om "Resume").

1. **När koden är pausad vid HAL\_Init(), vad står programräknaren (PC) på?** Du hittar den under (5). Det kan saknas nollor, så fyll på till vänster så att ditt svar är ett 32-bitars tal.
2. Titta på Memory Map i *databladet*. **Vad heter den region av adressrymden som PC pekar på?**

Tryck nu på Resume och sedan Resume en gång till (så att vi passerar förbi vår breakpoint). Tryck på kortets blå knapp när main-loopen kör. Dioden ska tändas och släckas av detta.

3. **Är den blå knappen aktivt hög eller aktivt låg?**

Pausa körningen och öppna fliken SFRs. Expandera de GPIO-enheter som knappen och dioden ligger på. Pausa och kör några gånger och se till att ha knappen intryckt vid ena pausen och släppt vid andra pausen.

4. **Vilka register ändrar sig (byter bakgrundsfärg) i och med att knappen/dioden ändrar tillstånd?** Om ett svar är "MODER på GPIOB" så skriv det som GPIOB->MODER.

SFRs-fliken uppdaterar inte sin lista varje paus om du inte expanderat i menyn. Expandera de sju registerklasserna under Cortex\_M4. Pausa och kör några gånger.

5. **Vilket enhet uppdateras mellan varje paus?**
6. **Varför tror du att den gör den det?**

Godkända svar:



## Enchipsdatorer – Laboration 1 Tärning

### 2.3.Din första kodningsuppgift

Låt oss skriva en funktion som stämmer lite bättre semantiskt med hur människor brukar tänka runt knappar. Leta upp en plats ovanför main() att skriva kod på och börja där med detta kodskelett:

```
51 static void MX_USART2_UART_Init(void);  
52 /* USER CODE BEGIN PFP */  
53 int is_blue_button_pressed();  
54 /* USER CODE END PFP */  
55  
56 /* Private user code -----*/  
57 /* USER CODE BEGIN 0 */  
58  
59 // returns 1 if blue button is pressed  
60 // returns 0 if blue button isn't pressed  
61 int is_blue_button_pressed()  
62 {  
63     return 0;  
64 }  
65 /* USER CODE END 0 */
```

I vissa laborationsuppgifter så kommer det finnas krav på exakt hur din kod ser ut. Här är det första kravet:

- Koden får ej anropa HAL\_GPIO\_ReadPin().

Du undrar kanske hur du då ska läsa av benet. Om du har en GPIO-enhet, GPIOx, ta då och skriv in

GPIOx->

I din kod. När du skrivit '>'-tecknet så tryck Ctrl+mellanslag för att få upp de register som finns för enheten GPIOx. Om du sparar det uttrycket i en variabel, tex.

```
uint32_t reg_reading = GPIOE->OSPEEDR;
```

Så kommer den att dyka upp i debuggern, vilket är mycket hjälpsamt. OSPEEDR är dock ej registret du vill ha information från. Rätt svar såg du när du tittade på register som ändrade sig med knapptryckningar i förra övningen.

Denna uppgift har ett till kodkrav:

- Koden ska testa endast en bit. Alla andra bitar ska ignoreras.<sup>6</sup>

Ändra anropet från main() som sätter variabeln pressed så att din nya funktion används. LD2 ska lysa när knappen är nedtryckt.

---

<sup>6</sup> Ledtråd: Ditt resultat lär inte vara helt olikt vad som görs i else-satsen i main-loopen.

## Enchipsdatorer – Laboration 1 Tärning

### 3. Koppla och koda mer

#### 3.1. Koppla

Koppla nu in dina dioder enligt den ritning du gjorde i kapitel 1.

#### **GLÖM INTE MOTSTÅND!**

Du riskerar att ha sönder ditt kort annars. Ett motstånd per diod. Någonstans mellan 220 och 1000 Ohm blir bra. Anslut även GND.

För att testa kan du använda en av två metoder:

- Använd en sladd kopplad (i serie med ett motstånd, såklart) till någon av kortets spänningsben (AVDD, +3V3, VBAT).
- Ställ in din multimeter på dess diod-funktion (för modellen UT131B: välj pip och tryck på HOLD/SEL).

#### 3.2. Koda mer

Dags att bygga din tärning! Bygg vidare från koden som finns i main-loopen. Lägg till en variabel för att ha koll på vad tärningen har för värde, t.ex. så här:

```
uint8_t die_value = 1; // 1 <= die_value <= 6
```

Vår "slump" kommer av att processorn är oerhört snabb. Så länge knappen hålls inne så ska tärningens värde ändra sig ( $5 \rightarrow 6 \rightarrow 1 \rightarrow 2 \dots$ ). Du kan lägga till `HAL_Delay(100)` i main-loopen medan du testar. När det fungerar så ska den inte vara där eller vara väldigt kort (`HAL_Delay(1)` blir bra).

Skriv nu funktionen

```
void put_die_dots(uint8_t die_nbr)
```

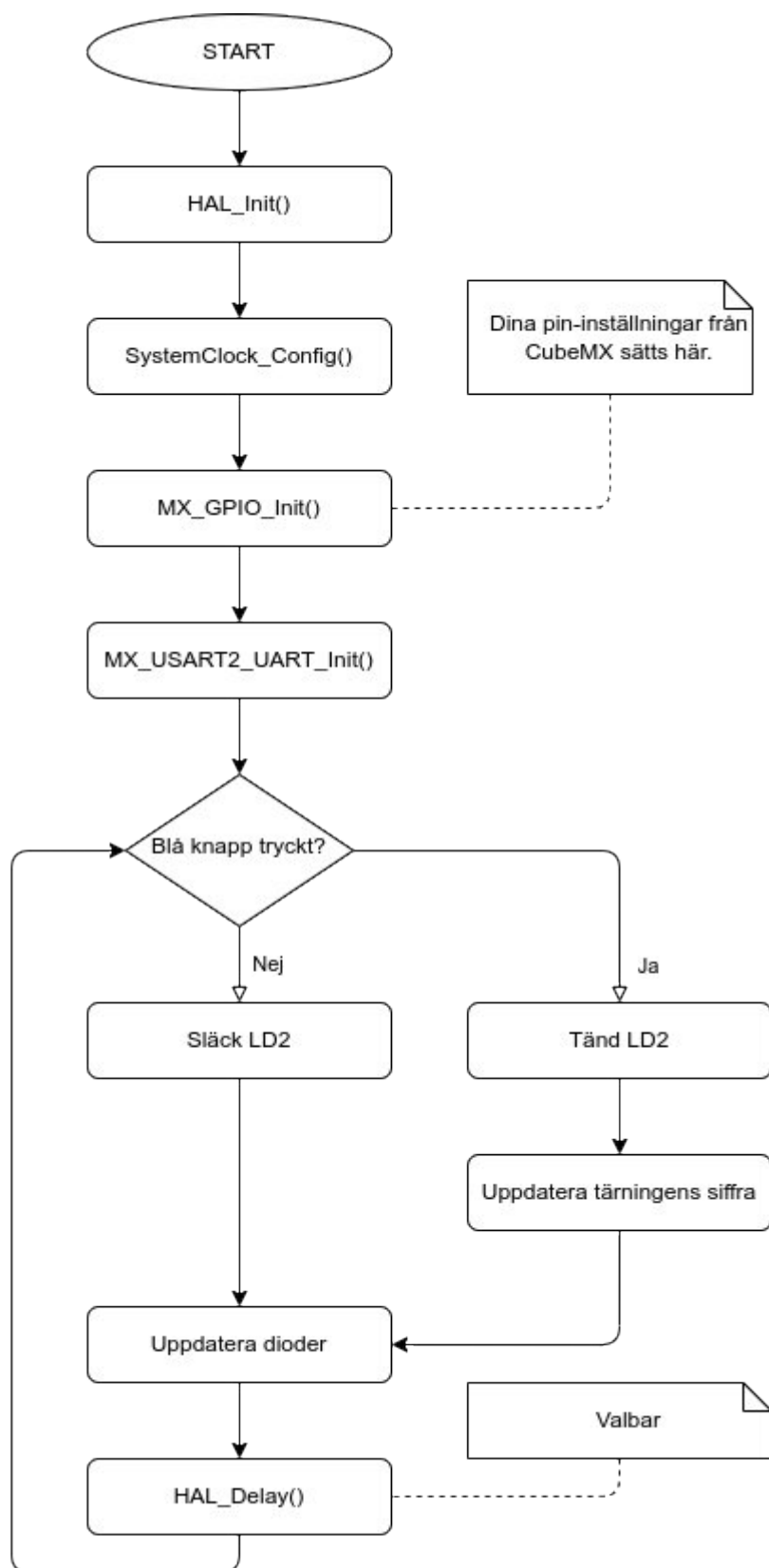
Den har dessa krav:

- Visar upp tärningsprickar enligt siffran som skickas som argument.
- Ska tända och släcka dioderna med `HAL_GPIO_WritePin()`.
- Dioderna ska refereras till med de namn de fick i din ritning/CubeMX.
- Ska ha en switch-sats och inga if-satser.
- Ett önskat argument (utanför 1-6) ska tända alla sju dioderna.

Flödesschemat på nästa sida visar hur allt ska klistras ihop. Du kan även lägga en for-loop (med `HAL_Delay()` i) innan main-loopen för att testa att `put_on_die_dots()` fungerar för siffrorna 1-6 samt önskat argument.

## Enchipsdatorer – Laboration 1

### Tärning

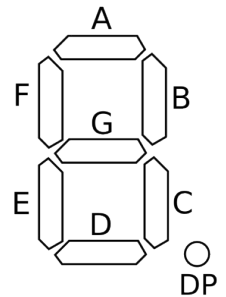


## Enchipsdatorer – Laboration 1 Tärning

### 4. Sjusegments-display

#### 4.1.Förberedelse

Fyll i tabellen nedan. Se hur det är gjort för siffran 5 och fyll i resten på samma sätt. Raden med "err" ska användas för att signalera att någonting gått snett med ditt program.



	DP	G	F	E	D	C	B	A	hex
0									
1									
2									
3									
4									
5	0	1	1	0	1	1	0	1	0x6D
6									
7									
8									
9									
err	1	1	0	1	1	1	0	0	0xDC
	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	

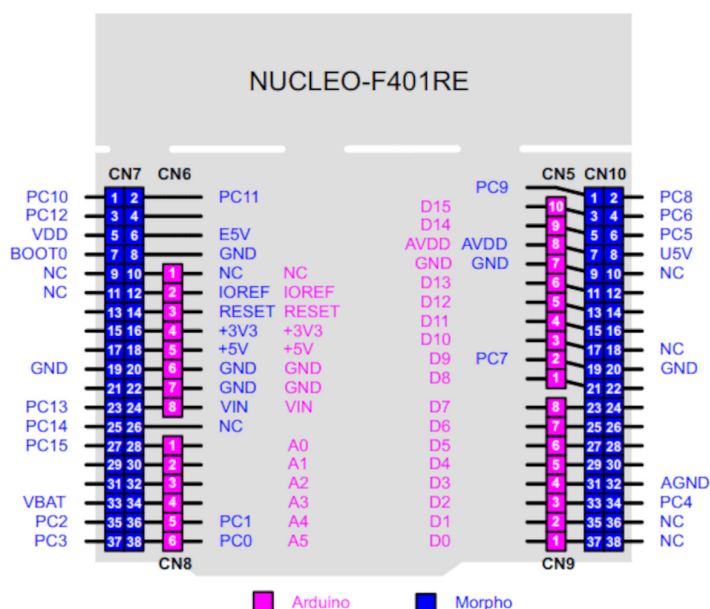
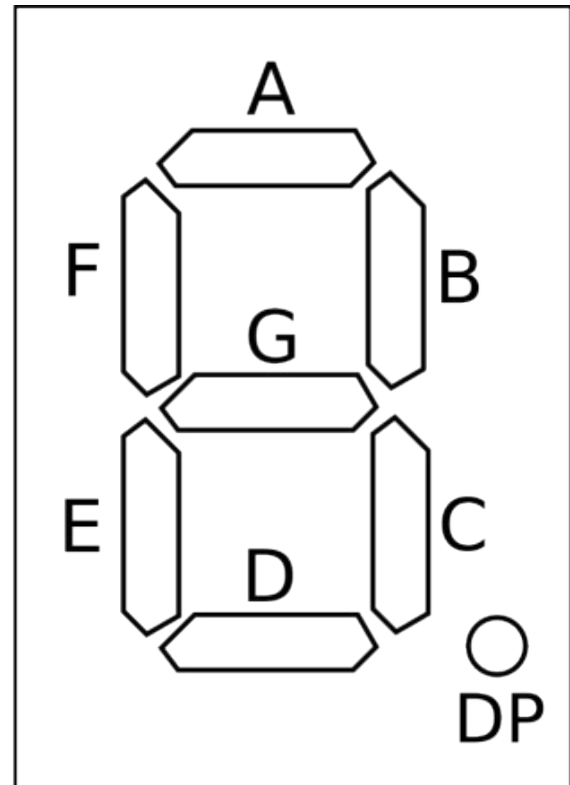
En sjusegments-display har 10 st ben. Lägg till benen i bilden på nästa sida (på långsidan för vissa varianter av komponenten). De två benen i mitten av en sida är jord.

Testa med hjälp av diodfunktionen på din multimeter (eller med hjälp av motstånd + spänningskälla på kortet) vilket ben som hör till vilket segment och skriv ut det i ritningen nedan. Lägg sedan till vilket PC-ben som hör till vilket segments-ben, i enlighet med nedersta raden i tabellen ovan. T.ex. ska du skriva "PC2" och "C" intill samma ben.

## Enchipsdatorer – Laboration 1 Tärning

När detta är klart, anslut displayen till ditt kort, tillsammans med motstånd - ett motstånd per segment. (Vad händer om du istället har enbart motstånd på jord-benen?)

Öppna CubeMX och ställ benen PC0 till PC7 till GPIO\_Output. Om du verkligen vill så får du ge dem namn, men det är onödigt för denna övning.



För att underlätta har alla ben som hör till någon annan enhet än GPIOC blivit utmarkerade ur bilden. Bilden säger 401 och inte 411 men de två MCU:erna har identisk pinout.

## Enchipsdatorer – Laboration 1

### Tärning

#### 4.2.Koda ihop

Skapa en array (globalt) i din kod som är 10 element lång och fyll den med de värden du räknade fram med hjälp av tabellen ovan.

```
const uint8_t sseg[10] = { ... , 0x6D, ... };
```

Att den har ordet "const" framför sig betyder att värdena i arrayen endast kan läsas från. Skapa även en global variabel

```
const uint8_t sseg_err = 0xDC;
```

att använda dig av ifall något blir fel.

Lägg till en funktion i din kod:

```
void put_on_sseg(uint8_t dec_nbr)
```

som gör så att siffran från main-loopen visas på din sjusegments-display. Som med tärningsprickarna så kan det vara en god idé att lägga till lite testkod innan main-loopen. Så här kan denna se ut

```
for(uint8_t i = 0; i <= 9; i++)  
{  
    put_on_sseg(i);  
    HAL_Delay(333);  
}  
put_on_sseg(88); // should display the error pattern
```

Krav för funktionen

- Får ej kalla på HAL\_GPIO\_WritePin().
- Ska ej innehålla någon switch-sats.
- Ska ha en if-sats som kontrollerar indata och använder sig av sseg\_err vid fel. Funktionen ska kunna hantera siffrorna 0 till 9.

När funktionen är klar och din lösning fungerar så visa upp ditt resultat för en handledare.

Kod följer kodkrav, lösning fungerar:



## Enchipsdatorer – Laboration 1 Tärning

### 5. Spara och Git

När allt fungerar så spara ditt arbete och gör en commit och push för din kod, så som du gjorde i slutet av laboration 0.

### 6. Har du Big Endian Energy?

Öppna Wikipediaartikeln för Endianness. Bilden med exempel där är väldigt hjälpsam.

<https://en.wikipedia.org/wiki/Endianness>

För programmerare som pillar i minne så har vi några hex-värden som vi gillar. Ett av dessa är 32-bitars-talet `0xDEADBEEF`<sup>7</sup> eftersom det är svårt att *inte* känna igen det<sup>8</sup> när man ser det.

MSB betyder här "Most Significant Byte" och LSB "Least Significant Byte". Sätt in de fyra bytes som utgör `0xDEADBEEF` i dessa rutor:

MSB=			LSB=
------	--	--	------

Givet en basadress B i minnet, spara ordet `0xDEADBEEF` följt av ordet `0xCAFED00D` där. Det första ordet ska alltså läggas på på B+0 och det andra på B+4 (vilket är B+1 med 32-bitars ordstorlek).

	B+0	B+1	B+2	B+3	B+4	B+5	B+6	B+7
big								
little								

---

<sup>7</sup> Fler kul koder: <https://en.wikipedia.org/wiki/Hexspeak>

<sup>8</sup> Som en övning: försök att gå genom stan, titta på skyltarna men att *INTE* läsa vad som står på dem. Att se det som en analfabet skulle gjort. GL;HF på den.

## Enchipsdatorer – Laboration 1

### Tärning

Skriv nu en funktion för att testa hur detta fungerar. Du kan lägga till anropet till denna precis innan main-loopen. Vi börjar med detta:

```
uint32_t arr[10];

void test_endianness()
{
    /*
       Initialize the array. The compiler knows that we're dealing
       with 32-bit numbers so if arr[i + 0] points to
       address 0x3000 then arr[i + 1] will point to 0x3004
       (and NOT to 0x3001).
    */
    for (int i = 0; i < 10; i += 2) // note: += 2
    {
        arr[i + 0] = 0xDEADBEEF;
        arr[i + 1] = 0xCAFED00D;
    }
    return;
}
```

Lägg en breakpoint på `return;`. Kör koden och inspektera minnet när breakpointen sker. Eftersom `arr[]` ligger utanför funktionen så dyker den inte upp i fliken "Variables". Du måste istället välja "Expressions" och lägga till "arr" som ett uttryck. Markera raderna `arr[0]` till `arr[9]` och få dem att ha "Hex" som sitt sifferformat.

Lägg nu till följande kod mellan loopen och retur-raden:

```
uint16_t a = arr[0];
uint16_t b = arr[1];
uint16_t c = arr[2];

uint8_t x = arr[0];
uint8_t y = arr[1];

// You may THINK you're 32 bits. I know better than you.
// Base pointers (arr, arr8) will point to same cell though.
uint8_t *arr8 = (uint8_t *) arr;

uint8_t p = arr8[0];
uint8_t q = arr8[1];
uint8_t r = arr8[2];
uint8_t s = arr8[3];
uint8_t t = arr8[4];
```

## Enchipsdatorer – Laboration 1 Tärning

Lägg till en breakpoint på raden `uint16_t a = arr[0];` och kör koden dit. Använd föregående sida och skriv där ner dina gissningar om vad som kommer finnas i variablerna efter körning. Kör koden för att kontrollera i Variables-fliken.

Markera det ord som är rätt (och stryk det som är fel):

För ett system med little endian – om jag har en u16 och läser in i den från en u32 så kommer jag få de bytes som är [ mest / minst ] värda med.

Lägg till dessa rader innan retur-raden:

```
uint16_t babe = 0xBABE;  
arr[0] = babe; // putting u16 in an u32 slot
```

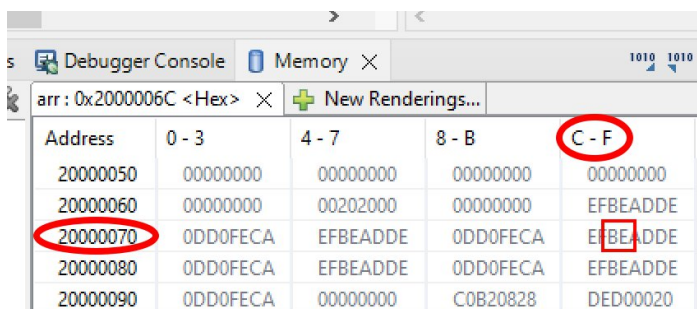
Innan du kör, fyll i byte-tabellen med din gissning (B+i är på u8-nivå, inte u32).

	B+0	B+1	B+2	B+3
före				
gissning				
verklighet				

Markera de ord som är rätt (och stryk det som är fel):

För ett system med little endian – om jag skriver en u16 till basadressen för en u32 så kommer jag att skriva över de bytes som är [ mest / minst ] värda.

När en u16 skrivs till en u32-plats (genom en u32-pekare) så kommer den mest värda halvan i minnet [ lämnas intakt / nollas ].



Address	0 - 3	4 - 7	8 - B	C - F
20000050	00000000	00000000	00000000	00000000
20000060	00000000	00202000	00000000	EFBEADDE
20000070	0DD0FECA	EFBEADDE	0DD0FECA	EFBEADDE
20000080	0DD0FECA	EFBEADDE	0DD0FECA	EFBEADDE
20000090	0DD0FECA	00000000	C0B20828	DED00020

Avslutningsvis går det att nämna att du kan titta i minnet på ett till sätt. Klicka på "Memory" (intill konsolen i en debug-session) och lägg till `arr` som uttryck. Vad som ligger i minnet visualiseras i segment av 4 byte. Detta är löpande byte-celler, från vänster till höger. Markerade BE ligger på adressen 2000007D.