

# Lab Pingpong - Introduktion till utvecklingsmiljö och programmering av mikrokontroller STM32L4

Detta dokument beskriver steg-för-steg hur utvecklingsmiljön STMCubeIDE används, programmering av STM32L476 och lär samtidigt ut grundläggande kunskaper om att strukturera program för inbyggda system. Du ska i denna inledande del av kursen IS1300 Inbyggda system gå igenom hela det här exemplet på egen hand. Du ska gå igenom hela häftet ordagrant enligt anvisningarna. Räkna med att det tar flera dagars heltidsjobb att gå igenom häftet. Redovisning av uppgiften sker enligt:

## Assignment 1 (Deadline: November 5., 2024)

Start av Pingpong, redovisa till och med sidan 10 uppgift 3, 4 och 5 på svarsblad.

## Demo (Lab: November 6./8., 2024)

Pingpong, tillståndsdiagram och program redovisas individuellt på lab. Godkänd

Pingpong-uppgift är ett villkor för att få fortsätta laborationskurserna.

### Innan du börjar denna guide skall du:

- Ha ett Nucleo-L476RG utvecklingskort och ett Pingpongkort tillgängligt
- Hämta STMCubeIDE från [www.st.com](http://www.st.com) och installera det på din dator.  
Anvisningarna i detta häfte baseras på STMCubeIDE. Workspace är det ställe på hårddisken där du sparar dina programmeringsprojekt för STM32.
- Installera drivrutin för ST-LINK när du installerar STMCubeIDE. ST-LINK är det programmerings- och debuginterface som finns på kortet. Det används för att ladda ned program och för att kunna styra programexekveringen på kortet. Du behöver en USB-kabel mellan dator och kort (USB-A till USB-B-Mini). Troligen har du en sådan liggande hemma, annars får du köpa en.

Från halvledartillverkaren STMicroelectronics [www.st.com](http://www.st.com) hämtar du hem följande

- UM1724 User manual för STM32 Nucleo-64 boards
- Datablad för mikrokontrollerkretsen STM32L476RG
- Referensmanual RM0351 för STM32L4x5 and STM32L4x6 advanced Arm®-based 32-bit MCUsSTM32L476RG
- STM32CubeL4 innehåller HAL<sup>1</sup>-bibliotek och ett antal programexempel som går att köra på utvecklingskortet Nucleo-L476RG.
- UM1860: Getting started with STM32CubeL4 MCU Package for STM32L4 Series and STM32L4+ Series
- UM1884: Description of STM32L4/L4+ HAL and low-layer drivers
- UM2553: STM32CubeIDE quick start guide
- Hittar du något mer intressant så kan du passa på att hämta det också ...

Jobba igenom hela detta häfte på egen hand steg för steg. Du skall kunna redovisa de program du skriver för läraren. Det finns också en del uppgifter som skall lösas. Du skall kunna redogöra för och förklara svaren till dessa. Ett fungerande pingpong-program skall kunna redovisas när du är klar. Du kan ta hjälp av eller hjälpa dina kamrater för att

<sup>1</sup> HAL = Hardware Abstraction Layer, sköter kommunikation mellan applikationer och hårdvara

lösa uppgifterna. Ge i så fall tips och inte färdiga svar. En pedagogisk tanke med detta häfte är att du själv ska lära dig hitta lösningar. Kör du fast kan du även fråga din lärare!

I detta häfte kommer du att göra följande

- Studera ett färdigt exempel
- Förstå initieringskod
- Förstå hur programmeringsprojekt är organiserade
- Lära dig använda debugger
- Lära dig söka information i datablad och referensmanual
- Generera kod som initierar kretsens periferiheter
- Skapa testprogram för att testa enskilda moduler som du utvecklar
- Skapa ett Pingpong-program strukturerat som en tillståndsmaskin

#### **Assignment 1 (Deadline November 5, 2024)**

- Gör alla uppgifter och svara på alla frågor till och med sidan 10.

#### **Redovisning laboration November 6./8., 2024**

- Färdigställt tillståndsdiagram redovisas.
- Testprogram för utvecklade moduler redovisas.
- Programmet Pingpong med korrekt funktion enligt specifikation ska redovisas för läraren.

## Studera ett färdigt exempel GPIO-IOToggle

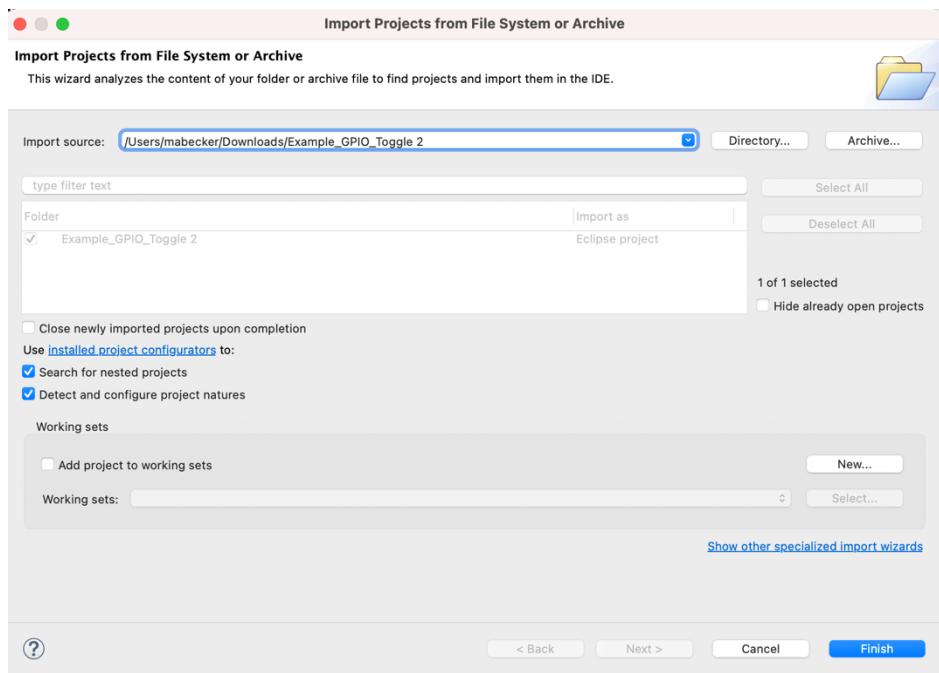
Programexemplet som vi ska studera heter GPIO-IOToggle. GPIO står för General Purpose Input Output. När en pinne på kretsen används för GPIO betyder det att den pinnen används för digitala in- och utsignaler. Pinnarna på mikrokontrollern kan användas för digitala in- och utsignaler eller för alternativa funktioner som timers, analog till digital omvandlare, seriekommunikation med mera. Vi kommer i det här exemplet att koncentrera oss på GPIO.

Programexemplet finns på Canvas sidan at lada new.

Starta STMCubeIDE

Open / Open Projects from File System...

Under Import source, bläddra fram till GPIO\_IOToggle\_22 och välj mappen.



Anslut utvecklingskortet NUCLEO-L476RG med USB-kabeln.

Project / Build all kompilar alla filer

Markera projektet STM32L476RG\_NUCLEO<sup>2</sup> i Project Explorer och gör  
Run / Debug länkar och laddar ned koden i microkontrollern  
Välj Debug as " STM32 MCU C/C++ Application"

Det bör fungera med alla defaultinställningar. Debug-interface ska vara ST-LINK och programmeringsgränssnittet SWD.

---

<sup>2</sup> Projektet borde egentligen heta GPIO\_IOToggle

Om allt nu fungerar så har koden kompilerats, länkats och laddats ned i microcontrollern. Du är nu i debugläge vilket innebär att du kan köra programmet. Programpekaren pekar på första instruktionen i main, dvs HAL\_Init().

Run / Resume      startar programmet. Nu ska grön lysdiod LD2 blinka på kortet.

Det finns funktionsknappar för de vanligaste kommandon som du behöver utföra.

För muspekaren över knapparna så ser du vad de betyder

Resume – startar programexekvering utan att stega

Suspend – stoppar programmet

Reset the chip and restart debug session – om du vill köra om programmet från start

Step over – stegar programexekveringen, funktioner exekveras utan att du stegar in

Step into – stegar programexekveringen och stegar även in i funktioner

Step return – stegar ut ur aktuell funktion

Restart Debugger – startar om debuggern utan att ladda ned programmet

Terminate – stannar programexekvering och lämnar debugläget

Som du ser så togglar utgången som driver LED2 med 100 ms fördröjning mellan varje växling. Programmet fortsätter i evighet eftersom det är en evighetsloop while (1) i programmet. Program i microcontrollern måste alltid gå i en evighetsloop eftersom det inte finns något operativsystem att återvända till när ett program avslutas.

Under debugging har du möjlighet att studera värden på register, variabler och minne.

Under fliken fönstret Show View har du flera alternativ att visa vad som händer under programkörning. Under fliken Expressions kan du analysera vilka värden variabler har när du stannat exekveringen i en brytpunkt. Om du använder Live expressions kan du se variabelvärdet under programkörning. Medan du jobbar med det här häftet bör du lära dig att använda debuggern för att testa att programexekvering går som den ska och att variabelvärdet är de rätta.

Du kan sätta brytpunkter i programmet. Programmet exekveras då fram till brytpunkten där det stannar så att du kan studera tillståndet och eventuella variabelvärdet i den punkten. Du kan sedan fortsätta programkörningen med Resume.

Brytpunkter sätter du eller tar bort genom att dubbelklicka på listen i marginalen till vänster om programtexten.

Kör programmet, stanna det, sätt brytpunkter och stega. Testa vad som händer!

Vi skall titta närmare på några av de filer som ingår i projektet och hur de hänger ihop

**startup\_stm32l476xx.s** hittar du i Project explorer under Example/Truestudio.

Det är en assemblerfil som körs innan vårt huvudprogram startas. Denna fil behöver du normalt inte ändra. I den här modulen sätts

- stackpekaren SP (stack pointer)
- programräknaren PC (program counter) initieras så att programmet startar med main-funktionen
- initierar vektortabellen med adresser till avbrottsrutinerna (ISR Interrupt Service Routine)

**main.c** hittar du i Project explorer under Example/User

Fil för huvudprogrammet, i denna fil finns funktionen main() där programexekveringen startar

### **main.h**

Headerfil för main.c

**stm32f30xx\_it.c** hittar du i Project explorer under Example/User Interrupt handlers, i denna fil finns avbrottsrutinerna.

### **stm32f30x\_it.h**

Headerfil för stm32f30x\_it.c

## **Header-filer, vad är det?**

När vi skriver program så är det lämpligt att modularisera programmet. Vi kan ha flera c-filer (programkod) och h-filer (headerfiler) som länkas samman till ett program.

Headerfilerna skall innehålla deklarationer av funktioner, konstanter etc. men ingen programkod. I headerfilen för en modul deklareras funktioner, datastrukturer, makron och variabler som är externa, det vill säga de variabler som deklareras på annan plats än i samma aktuella källkodsfil. Om du studerar programmet GPIO\_IOToggle i main.c så ser du att det består av initiering av HAL-biblioteket, initiering av klockan och initiering av GPIO-pinnarna innan det loopar i en evighetsloop som blinkar lysdioden.

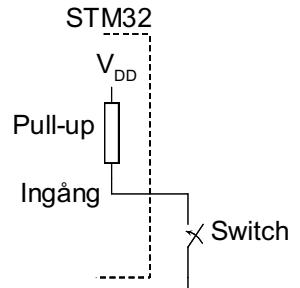
## Digitala in- och utgångar GPIO<sup>3</sup>

En digital ingång kan vara

- floating, flytande potential. Potentialen måste definieras av en extern krets.
- pull up, en intern resistor lyfter potentialen till att hålla ingången hög.
- pull down, en intern resistor sänker potentialen till att hålla ingången låg.

### Exempel på inkoppling av extern switch

När switchen är öppen hålls ingången hög av den interna pullup-resistorn. När switchen är sluten hålls ingången låg av att switchen sluts mot jord. Har man en extern pullup-resistor istället kan ingången vara floating.

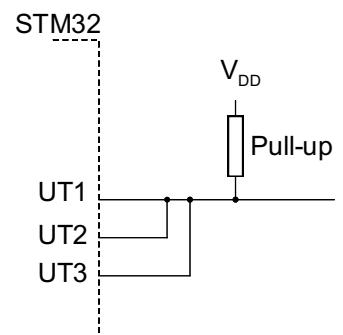


En digital utgång kan vara

- disabled, frikopplad. Ingen aktiv styrning av utgången hög eller låg.
- push-pull, aktiv styrning av utgången hög eller låg beroende av utdata.
- open drain, aktiv styrning låg. För att utgången skall kunna bli hög krävs yttre pullup-resistor.

Två utgångar som är konfigurerade som push-pull kan inte kopplas ihop. Då kan det inträffa att en utgång driver hög och en annan driver låg, då blir det stora strömmar mellan utgångarna som kan förstöra kretsen.

Flera utgångar kan dock kopplas samman om de drivs som open drain. Hopkopplingspunkten får då en logisk AND-funktion, utgången är hög om alla utgångar är höga, dvs det som håller ledningen hög är den yttre eller inre pullup-resistorn. Utgången kan konfigureras som open drain med intern pullup-resistor. Det räcker att en utgång går låg så blir ledningen låg. Det är en aktiv transistor som sänker ledningen. Flera utgångar som driver samma ledning med open drain är normalt olika kretsar som överför data via en gemensam ledning, en databuss. I2C<sup>4</sup> är exempel på en kommunikationsbuss som använder denna teknik.



Funktionen för de digitala portarna konfigureras med ett antal register

- control register, styr hur respektive bit konfigureras.
- data register, data ut till pinnarna eller data in från pinnarna.

Gå igenom kapitel 8 i referensmanualen RM0351 som beskriver GPIO. Granska speciellt avsnitt 8.3 som visar den logiska strukturen på I/O-pinnarna och avsnitt 8.4 som beskriver registren för GPIO.

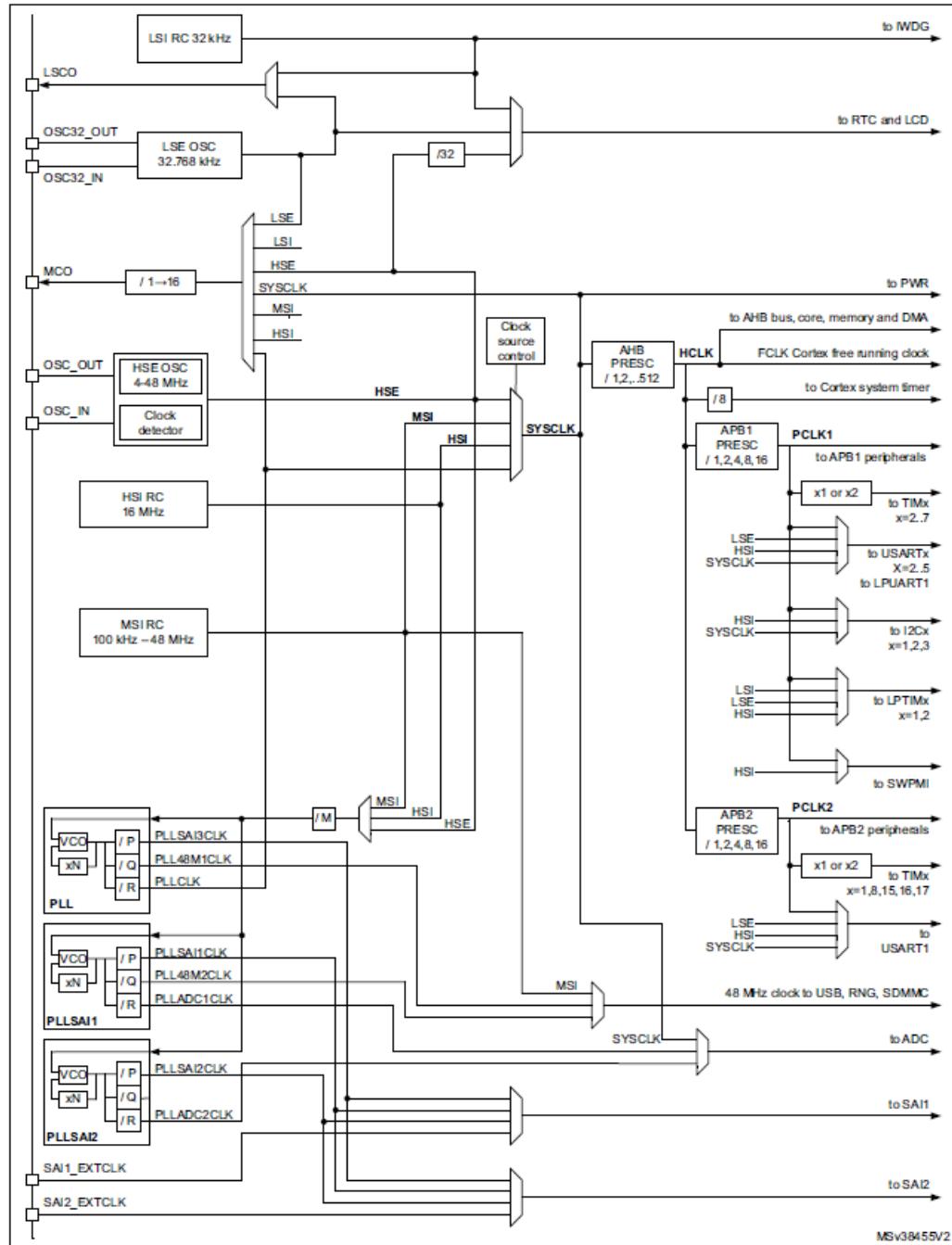
<sup>3</sup> GPIO = General Purpose Input Output

<sup>4</sup> I2C = I<sup>2</sup>C Inter-Integrated Circuit, lokal databuss för kommunikation mellan kretsar

## Klockning av mikrokontrollern

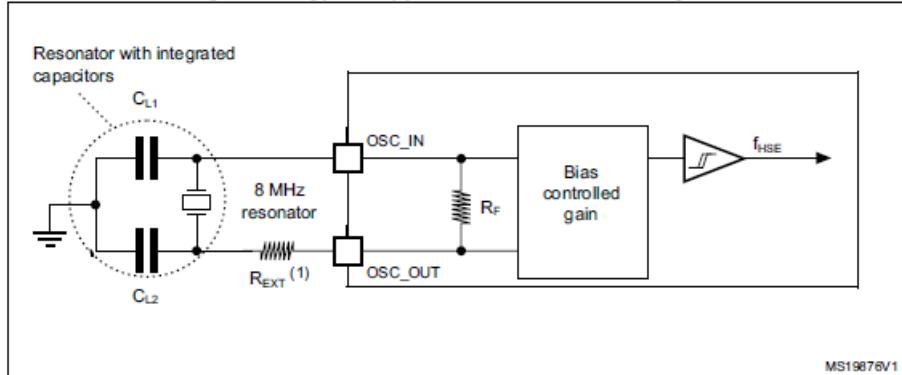
STM32 har en relativt avancerad struktur för att skapa klockning av olika enheter i microcontrollern. Gör man effektsnåla lösningar är det viktigt att inte klocka mer kretsar än vad som används och klocka med så låg frekvens som möjligt. Förlusteffekten är proportionell mot frekvensen, så varje klockning ger förlusteffekt. Först kan vi studera klockträdet i microcontrollern. Se figur 15 avsnitt 6.2 i Referensmanualen RM0351.

**Figure 15. Clock tree (for STM32L475xx/476xx/486xx devices)**



STM32 kan använda olika primära klockkällor. Vanligen används en yttre kristall för att ge oscillationsfrekvensen i en HSE<sup>5</sup>-oscillator. Man kan till exempel koppla en 8 MHz kristall tillsammans med några kondensatorer mellan OSC\_OUT och OSC\_IN, se figur 25 i databladet. Kristalloscillatörer kan göras väldigt exakta.

**Figure 25. Typical application with an 8 MHz crystal**



1.  $R_{EXT}$  value depends on the crystal characteristics.

Som alternativ kan en inbyggd 16 MHz HSI<sup>6</sup> RC<sup>7</sup>-oscillator användas som primär oscillator. Då får vi en enklare koppling men inte lika exakt klockfrekvens.

8 MHz är en väldigt låg klockfrekvens så den multipliceras vanligen upp till en högre frekvens. För den mikrokontroller vi använder kan frekvensen på systemklockan maximalt vara 80 MHz.

Om du studerar STM32L476RG-NUCLEO-kortet ser du att det saknas kristall monterad i position X3 på kortet. 8 MHz klocksignal kan komma från den mikrokontrolleren STM32F103C8T6 som finns i ST-LINK debuginterfacet på kortet. Den klockan baseras på den 8 MHz kristall som finns i ST-LINK debugkretsen. Det krävs dock en del omkopplingar på kortet för att använda denna. Vi kommer att använda den interna RC-oscillatören.

Vid programstart har det mesta av klockkretsarna initierats av SystemInit i filen `system_stm32l4xx.c`. SystemInit anropas från startup innan main startas.

Programmodulen `stm32l4xx_hal_rcc`<sup>8</sup> används för att från main-programmet göra alla övriga inställningar för klockdistribution. Det du som programmerar måste tänka på är att de flesta I/O-enheter har en klocka som måste aktiveras för den enheten. Klockorna ställs in i det grafiska gränssnittet som skapar initieringskod.

<sup>5</sup> HSE OSC= High Speed External oscillator

<sup>6</sup> HSI = High Speed Internal

<sup>7</sup> RC betyder att en resistor och kondensator i en oscillatorkrets bestämmer klockfrekvensen

<sup>8</sup> RCC = Reset and Clock Control

## Frågor om microcontroller

Nu kan det vara dags att stanna upp och läsa lite i manualerna, samt lösa några uppgifter, innan vi går vidare. Svara på frågorna på svarsbladet längst bak i häftet!

- 1) Läs i databladet för STM32L476xx kapitel 2 (den kontroller vi använder i NUCLEO-kortet är STM32L476RG<sup>9</sup>). Studera speciellt Table 2 och Figure 1 så får du en översikt över alla periferienheter som finns förutom digitala in- och utgångar.
- 2) Läs översiktligt igenom UM1860: Getting started with STM32CubeL4... Vi kommer att använda HAL-biblioteket när vi programmerar NUCLEO-kortet. När man som du är ny med en mikrokontroller kan det bli mycket information att ta in, men läs igenom översiktligt så att du får en inblick i arkitekturen för mjukvaran.
- 3) Läs sedan UM1884 Description of STM32L4/L4+ HAL and low-layer drivers, kapitel 2. Denna manual är vår uppslagsbok när det gäller API<sup>10</sup> för HAL<sup>11</sup>. I kapitel 2 finns en översikt över HAL-biblioteket. Läs kapitlet översiktligt och besvara följande frågor:
  - a) Det finns tre API programmeringsmodeller: polling, interrupt och DMA. Förklara vad som menas!
  - b) Vad menas med att koden är reentrant?
  - c) Vad menas med att implementationer av HAL APIs kan anropa user-callback functions, dvs. vad innebär user-callback?
- 4) Använd datablad för microcontroller och manual för NUCLEO-kortet för att ta reda på hur microcontrollern klockas. Vilka alternativ finns det för att klocka microcontrollern?
- 5) Nu skall du studera GPIO i microcontrollern, dvs. när en pinne används som ingång eller utgång. Du ska söka information i databladet för STM32L476xx när det gäller elektriska specifikationer och i referensmanualen RM0351 när det gäller logisk uppbyggnad av hårdvaran i periferienheten.
  - a) Hur bestäms det hur en pinne konfigureras som ingång eller utgång? Vilka register skall påverkas och på vilken eller vilka adresser ligger de om GPIO-port D skall påverkas.
  - b) Markera i figurerna 23, 24 och 26 nedan (från referensmanualen) vilka transistorer som är påverkade on eller off för följande fall
    - i) Ingång med pullup-motstånd
    - ii) Flytande ingång (floating)
    - iii) Utgång push-pull
    - iv) Utgång open drain
  - c) Om en pinne konfigureras som utgång, hur mycket ström kan den leverera?
  - d) Om en pinne konfigureras som ingång, hur hög spänning får man maximalt lägga på pinnen utifrån? Det kan vara olika för olika pinnar, specificera!
  - e) Inom vilket område ska matningsspänningen till microcontrollern ligga vid normal drift?

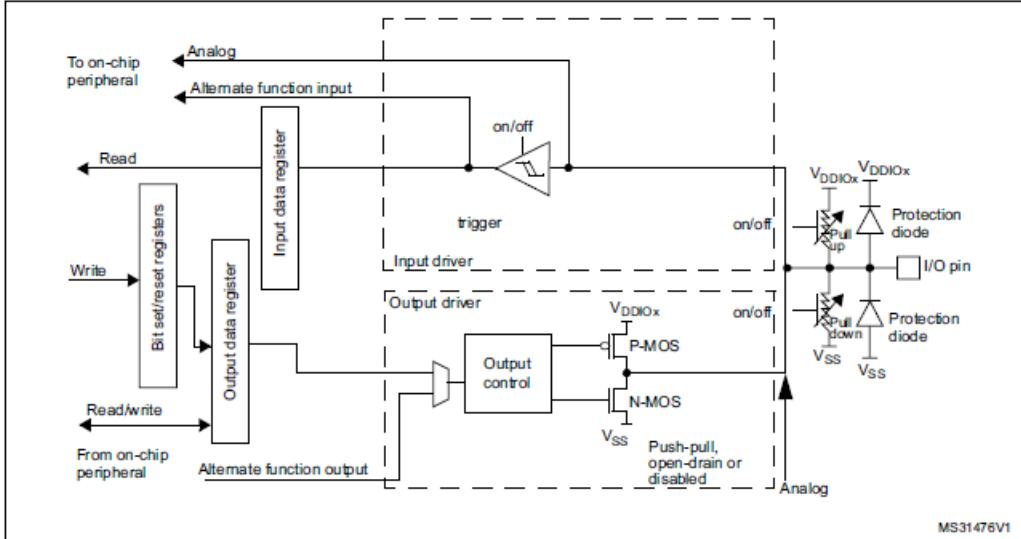
---

<sup>9</sup> L4 är produktserie, 76 är produktlinje i serien, R står för 64 pinnars kapsel, G betyder att kretsen har 1 MByte flashminne

<sup>10</sup> API Application Programming Interface

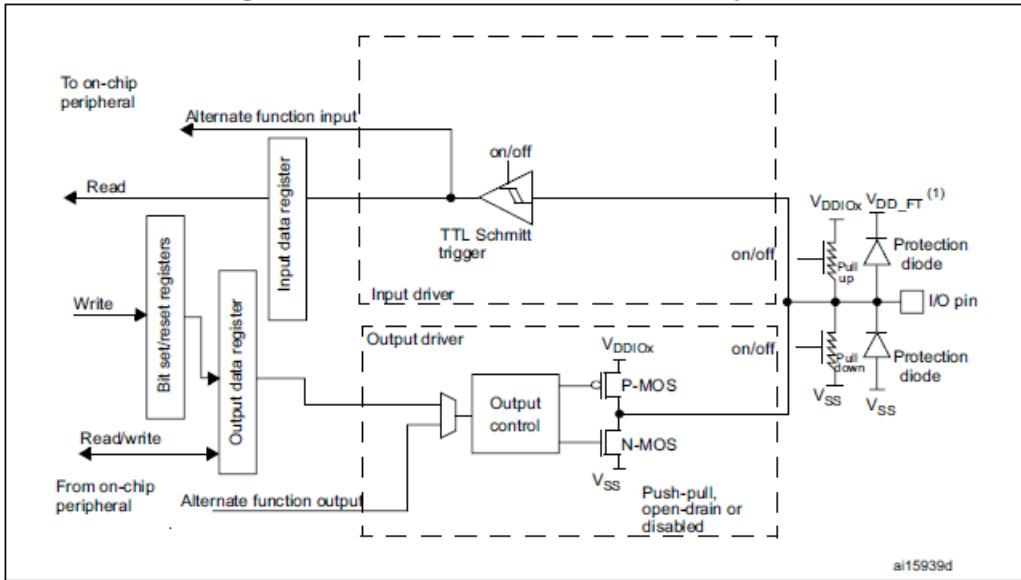
<sup>11</sup> HAL Hardware Application Layer

**Figure 23. Basic structure of an I/O port bit**



MS31478V1

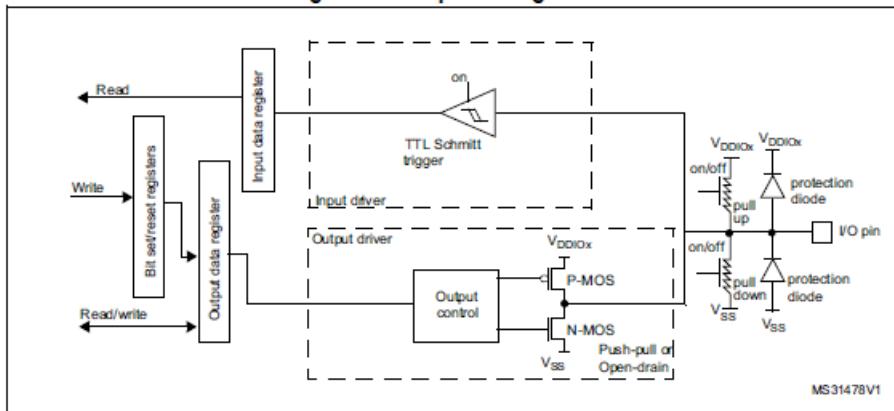
**Figure 24. Basic structure of a 5-Volt tolerant I/O port bit**



ai15939d

1.  $V_{DD\_FT}$  is a potential specific to five-volt tolerant I/Os and different from  $V_{DD}$ .

**Figure 26. Output configuration**



MS31478V1

## Programmeringsuppgift Pingpong

Vi skall nu gå igenom hur vi skapar ett program för NUCLEO-L476RG. Det program vi skall skapa är ett program för en enkel speltillämpning.

### Pingpong-spelet

Vi skall skapa ett program som du kan använda för att spela Pingpong!

Spelets regler är följande:

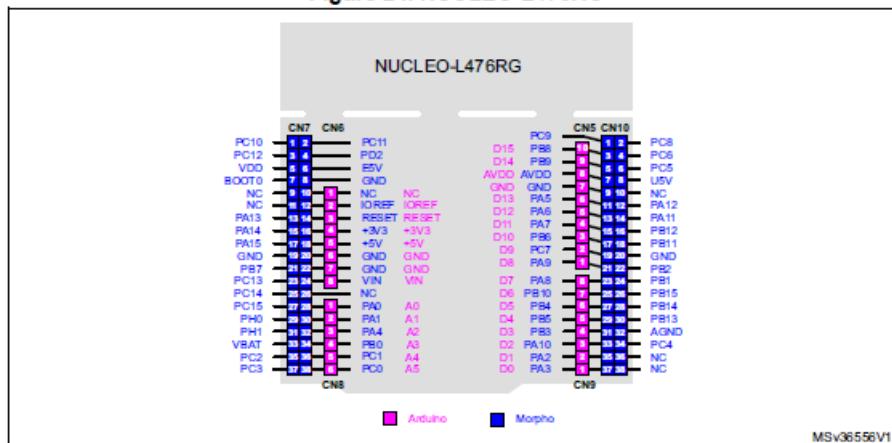
Det finns två spelare, L och R (Left och Right). Den som servar trycker på sin knapp. Då tänds lysdioder, en i taget, för att animera att en boll rör sig till motspelaren. En boll returneras om knappen trycks ned samtidigt som yttersta dioden är tänd. Om man trycker för tidigt eller för sent är det en tappad boll och motspelaren får poäng. En missad boll markeras genom att alla lysdioder blinkar till lite kort, varefter poängställningen visas. Poängställningen efter en vunnen och tappad poäng visas ett kort ögonblick genom att tända upp så många lysdioder på respektive sida som motsvarar antal poäng. Spelarna turas om att serva. Vid start kan vem som helst serva, men det är den andra spelaren som servar nästa gång. Om till exempel vänster spelare servar första gången är det höger spelare som servar nästa gång oberoende av vem som tappat poäng. Den som först kommer till fyra vunna poäng vinner. Resultatet ska visas under fem sekunder innan spelet startas om. Den tid varje lysdiod är tänd skall minska allteftersom beroende på hur länge en boll varit i spel. På så sätt ökar "bollhastigheten" och det blir allt svårare att hinna returnera bollen. Det kan vara lämpligt att sätta en maximal bollhastighet.

Vi skall nu gå igenom

- Initiering digitala in- och utgångar (GPIO, General Purpose Input Output)
- Initiering av klockkretsar
- Tända och släcka lysdioder
- Testning av program, testdriven utveckling
- Skriva en funktion som tänder önskad lysdiod
- Läsa av tryckknappar, kontaktstuds och avbrottshantering
- Arkitekturbeskrivning av hårdvara och mjukvara
- Skriva kod för en tillståndsmaskin (state machine)
- Modularisering av program

Du bör nu ha ett Pingpong-kort tillgängligt. Det ska anslutas på kontakten CN10 längst ut till kontaktpinnarna 2-22.

Figure 24. NUCLEO-L476RG



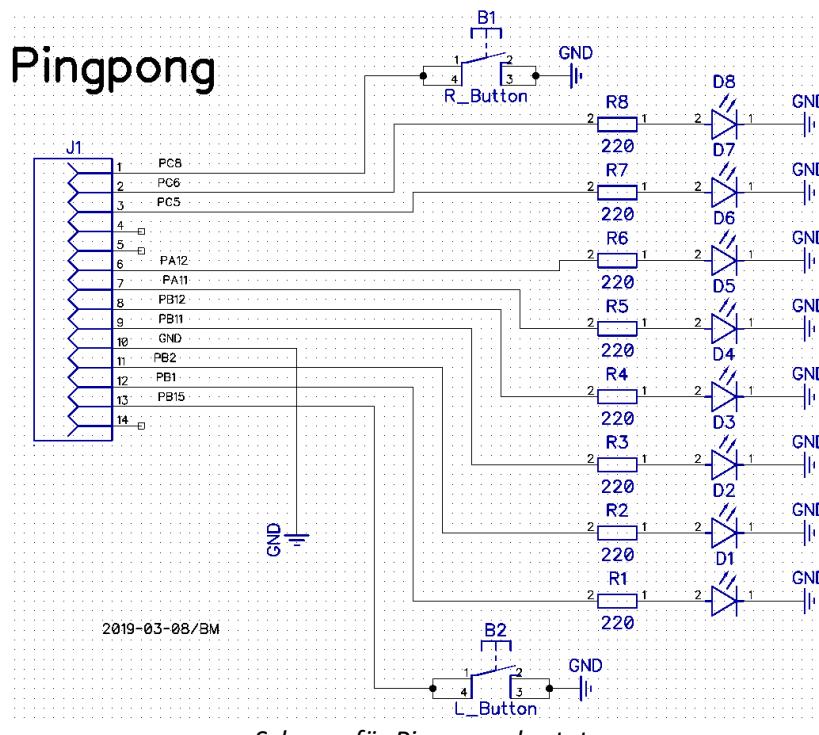
## Pingpong-kortet

Pingpong-kortet använder följande pinnar på STM32L476

Höger knapp R	PC8	R_button
Vänster knapp L	PB15	L_button
Lysdiod 1	PB1	LED1
Lysdiod 2	PB2	LED2
Lysdiod 3	PB11	LED3
Lysdiod 4	PB12	LED4
Lysdiod 5	PA11	LED5
Lysdiod 6	PA12	LED6
Lysdiod 7	PC5	LED7
Lysdiod 8	PC6	LED8



Pingpong-kortet ansluts till NUCLEO-L476RG i den yttre raden på motsvarande pinnar som finns angivna på kortet.



Innan du fortsätter rekommenderar jag att du läser igenom UM2553 STM32CubeIDE quick start guide. En del av det som står där har redan introducerats men det är en bra sammanställning och förklarar en del begrepp som kommer att användas här.

### Skapa initieringskod med STM32CubeMX

Vi ska först generera initieringskod med STM32CubeMX. Det ingår i STMCubeIDE och kommer att öppnas som ett perspektiv.

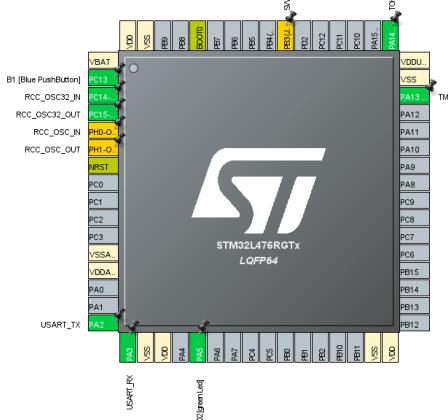
New / STM32 Project

Markera att du vill välja NUCLEO-64 och serien STM32L4 så bör du hitta kortet. Välj NUCLEO-L476RG

Ge projektet namnet Pingpong och öppna STM32CubeMX perspektiv. Eftersom det är första gången du startar ett nytt projekt kommer senaste version av HAL-biblioteket att laddas ned. Det kan ta lite tid...

Under fliken Pinout & Configuration visas alla pinnar för kretsen och hur de används.

GPIO Mode and Configuration								
Configuration								
Group By Peripherals								
<input checked="" type="checkbox"/> GPIO <input type="checkbox"/> Single Mapped Signals <input type="checkbox"/> RCC <input type="checkbox"/> SYS <input type="checkbox"/> USART2 <input type="checkbox"/> NVIC								
Search Signals <input type="text" value="Search (Ctrl+F)"/> <input type="checkbox"/> Show only Modified Pins								
Pin	Signal on Pin	GPIO o.	GPIO o.	GPIO m.	GPIO P.	Maximum	Fast Mode	User Label
PA5	n/a	Low	Output	No pull...	Low	n/a	LD2 [gr...	<input checked="" type="checkbox"/>
PC13	n/a	n/a	External...	No pull...	n/a	n/a	B1 [Blu...	<input checked="" type="checkbox"/>



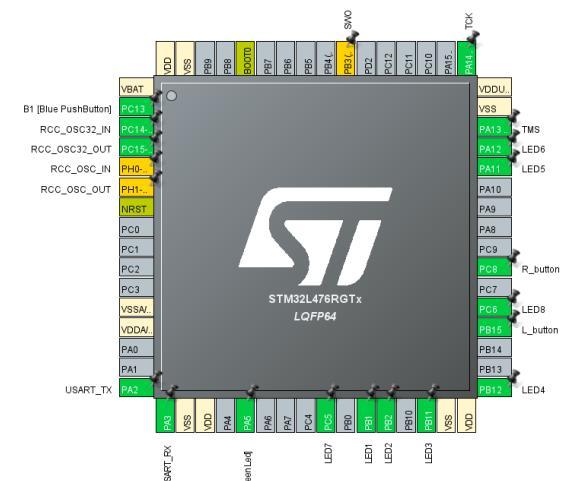
På Nucleo-kortet används bara två GPIO-pinnar för en lysdiod (LD2) och en tryckknapp (USER). De pinnar som redan är upptagna och används på kortet finns markerade i pinout view.

Nu ska vi konfigurera de pinnar som används för Pingpong-kortet. Konfigurera de pinnar som ansluts till knappar som GPIO\_input med pull-up och de pinnar som ansluts till lysdioderna som GPIO\_output push-pull.

Namnge pinnarna enligt figuren så att det matchar de pinnar som används i pingpongkortet.

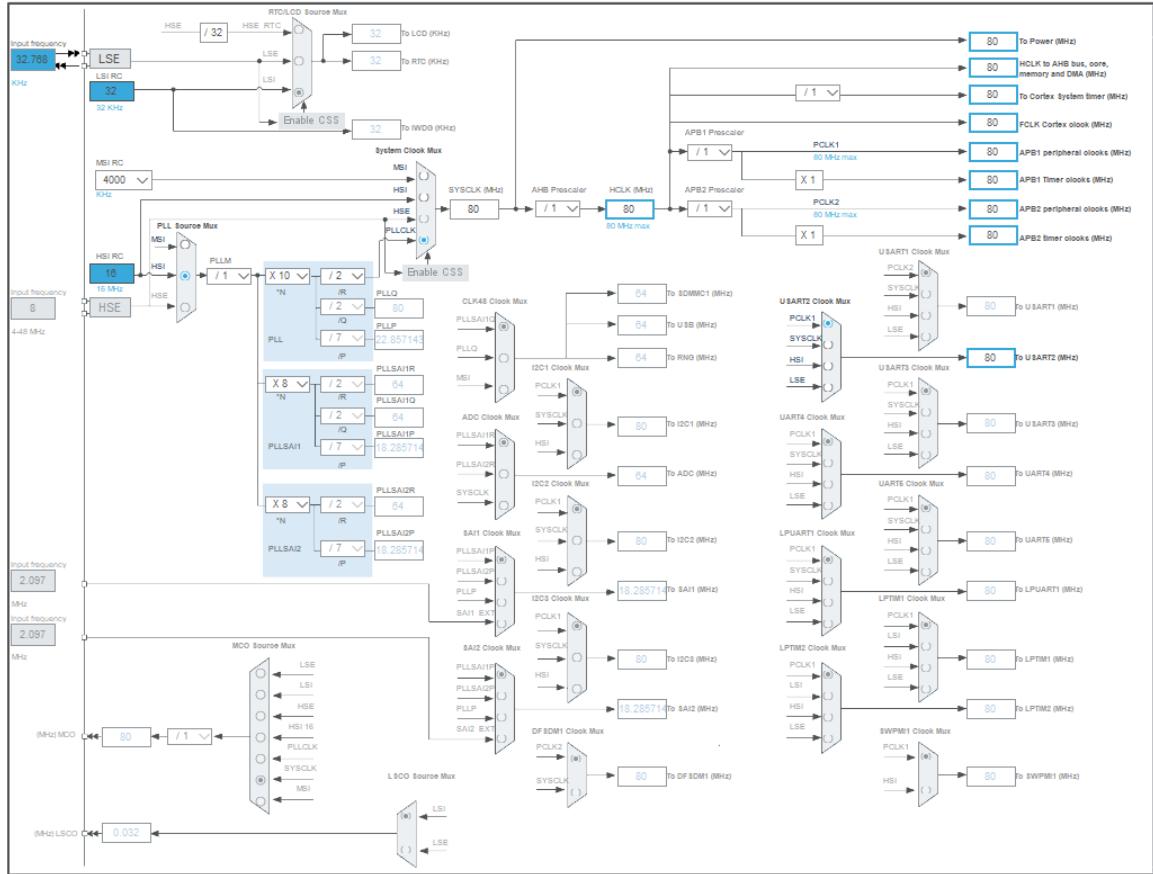
När du är klar med konfigureringen ska det se ut enligt figurerna nedan.

GPIO Mode and Configuration								
Configuration								
Group By Peripherals								
<input checked="" type="checkbox"/> GPIO <input type="checkbox"/> Single Mapped Signals <input type="checkbox"/> RCC <input type="checkbox"/> SYS <input type="checkbox"/> USART2 <input type="checkbox"/> NVIC								
Search Signals <input type="text" value="Search (Ctrl+F)"/> <input type="checkbox"/> Show only Modified Pins								
Pin Name	Signal on Pin	GPIO output I.	GPIO mode	GPIO Pull-up/Pull-down	Maximum	Fast Mode	User Label	Modified
PA5	n/a	Low	Output Push...	No pull-up and no pull-down	Low	n/a	LD2 [gre...	<input checked="" type="checkbox"/>
PA11	n/a	Low	Output Push...	No pull-up and no pull-down	Low	n/a	LED5	<input checked="" type="checkbox"/>
PA12	n/a	Low	Output Push...	No pull-up and no pull-down	Low	n/a	LED6	<input checked="" type="checkbox"/>
PB1	n/a	Low	Output Push...	No pull-up and no pull-down	Low	n/a	LED1	<input checked="" type="checkbox"/>
PB2	n/a	Low	Output Push...	No pull-up and no pull-down	Low	n/a	LED2	<input checked="" type="checkbox"/>
PB11	n/a	Low	Output Push...	No pull-up and no pull-down	Low	n/a	LED3	<input checked="" type="checkbox"/>
PB12	n/a	Low	Output Push...	No pull-up and no pull-down	Low	n/a	LED4	<input checked="" type="checkbox"/>
PC5	n/a	Low	Output Push...	No pull-up and no pull-down	Low	n/a	LED7	<input checked="" type="checkbox"/>
PC6	n/a	Low	Output Push...	No pull-up and no pull-down	Low	n/a	LED8	<input checked="" type="checkbox"/>
PC13	n/a	External Inter...	No pull-up and no pull-down	n/a	n/a	B1 [Blue ...	<input checked="" type="checkbox"/>	
PB15	n/a	n/a	Input mode	Pull-up	n/a	n/a	L_button	<input checked="" type="checkbox"/>
PC8	n/a	n/a	Input mode	Pull-up	n/a	n/a	R_button	<input checked="" type="checkbox"/>



Växla flik till Clock configuration. Då visas klockträdet, som visar hur klocksignaler distribueras i mikrokontrollern. Det finns en intern RC-oscillator som ger 16 MHz som multipliceras med 5 och ger 80 MHz systemklocka. Multiplikationen sker i PLL<sup>12</sup>. Kontrollera att klockor distribueras enligt vad som visas i klockträdet nedan.

<sup>12</sup> PLL Phase Locked Loop



Nu är konfigurationen av pinnar och klocka klar. Nu kan vi generera c-kod som initierar allt vi behöver så vi kan koncentrera oss på att skriva applikationskod.

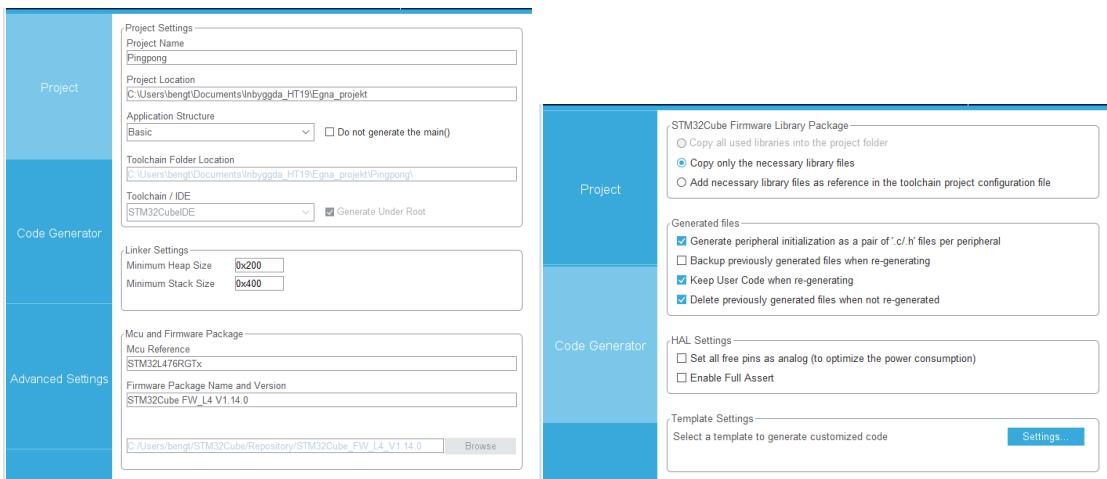
Välj fliken Project Manager

I fliken Project ska du se till så att det är rätt sökväg och att koden för ditt projekt skapas i den mapp du önskar.

Under fliken Code generator kan du välja att generera separata .c och .h filer för varje IP<sup>13</sup>. På så sätt blir inte main-filen så stor utan det genereras särskilda filer för initieringskoden. Det är heller inte nödvändigt att ladda ned biblioteket för varje nytt projekt som skapas.

Om "Keep User Code when re-generating" är markerad kommer all kod som du själv skriver inom parenteserna USER CODE BEGIN och USER CODE END inte att skrivas över om du genererar om initieringskoden. Detta förklaras senare när vi börjar skriva egen kod för Pingpong.

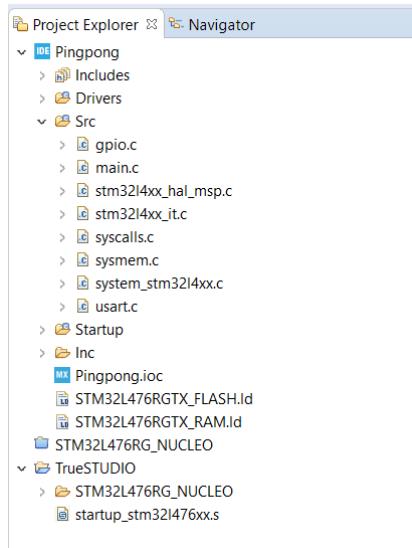
<sup>13</sup> IP Intellectual Property, i det här fallet betyder det nog ungefär varje periferienhet



Generera sedan kod med

Project / Generate Code

Ändra view till Project Explorer. Ditt projekt innehåller nu följande filer.



Du kan prova att kompilera och länka

Project / Build all

Förhoppningsvis går det igenom utan fel.

Den kod som genereras gör enbart initiering av HAL, klocka och hårdvaruinterface GPIO.

I princip ser main-programmet nu ut så här

```
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();
```

```

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}

```

Koden som skapas innehåller initiering av HAL (Hardware Abstraction Layer), konfigurering av klockningen och initiering av I/O-pinnarna. En UART som finns på kortet är också initierad.

Sedan går programmet in i en evighetsloop while(1) som inte gör annat än snurrar runt i evighet, eller till dess du stoppar körningen.

Som du ser så innehåller koden parenteser i form av kommentarer.

```

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

```

Om du placerar din egen kod mellan dessa parenteser kommer den inte att skrivas över när du genererar om koden med STM32CubeMX. Detta förutsatt att du markerat rätt under Project Manager / Code Generator. Det kan dock vara klokt att ha en backup av koden innan du testar att det fungerar.

För att komma tillbaka till perspektivet STM32CubeMx för att initiera om koden dubbelklickar du på

 [Pingpong.ioc](#)

i Project Explorer.

Låt oss analysera hur initieringen av GPIO går till. Hela NUCLEO-kortet initieras, men vi koncentrerar oss till den del som hör till Pingpong som finns i filen gpio.c:

```

/*Configure GPIO pins : PAPin PAPin PAPin */
GPIO_InitStruct.Pin = LD2_Pin|LED5_Pin|LED6_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : PCPin PCPin */
GPIO_InitStruct.Pin = LED7_Pin|LED8_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

```

```

HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pins : PBPin PBPin PBPin PBPin */
GPIO_InitStruct.Pin = LED1_Pin|LED2_Pin|LED3_Pin|LED4_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : PtPin */
GPIO_InitStruct.Pin = L_button_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLUP;
HAL_GPIO_Init(L_button_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : PtPin */
GPIO_InitStruct.Pin = R_button_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLUP;
HAL_GPIO_Init(R_button_GPIO_Port, &GPIO_InitStruct);

```

Låt oss undersöka vad som menas med raderna

```

/*Configure GPIO pins : PBPin PBPin PBPin PBPin */
GPIO_InitStruct.Pin = LED1_Pin|LED2_Pin|LED3_Pin|LED4_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

```

Det finns definierade strukturer (struct) och fördefinierade konstanter för att ge värden till alla register som skall ställas in. På så sätt blir det enkelt att konfigurera utan att behöva gräva ned sig i registerinställningar och bitvis manipuleringar av enstaka bitar i registren. Till HAL\_GPIO\_Init skickas adressen till GPIOB och adressen till GPIO\_InitStructure som parameter.

Låt oss kontrollera adressen till GPIOB:

Håll markören över GPIOB så kommer det att visa i ett popup-fönster vilken adress GPIOB har.

((GPIO\_TypeDef \*) (((0x40000000UL) + 0x08000000UL) + 0x0400UL))

Adressen till GPIOD är således 0x40000000 + 0x08000000 + 0x00000400 = 0x48000400

Kontrollera detta värde mot minneskartan i Table 1 i referensmanualen!

AHB2	0x4800 1C00 - 0x4800 1FFF	1 KB	GPIOH	<a href="#">Section 8.4.13: GPIO register map</a>
	0x4800 1800 - 0x4800 1BFF	1 KB	GPIOG	<a href="#">Section 8.4.13: GPIO register map</a>
	0x4800 1400 - 0x4800 17FF	1 KB	GPIOF	<a href="#">Section 8.4.13: GPIO register map</a>
	0x4800 1000 - 0x4800 13FF	1 KB	GPIOE	<a href="#">Section 8.4.13: GPIO register map</a>
	0x4800 0C00 - 0x4800 0FFF	1 KB	GPIOD	<a href="#">Section 8.4.13: GPIO register map</a>
	0x4800 0800 - 0x4800 0BFF	1 KB	GPIOC	<a href="#">Section 8.4.13: GPIO register map</a>
	0x4800 0400 - 0x4800 07FF	1 KB	GPIOB	<a href="#">Section 8.4.13: GPIO register map</a>
	0x4800 0000 - 0x4800 03FF	1 KB	GPIOA	<a href="#">Section 8.4.13: GPIO register map</a>

I filen `stm32f3xx_hal_gpio.h` är strukturvariablerna (struct) definierade och i filen `stm32f3xx_hal_gpio.c` finns funktionerna som skriver in värden i perifera registren som konfigurerar GPIO-portarna.

Använd debuggern för att undersöka värden på bitar som påverkas i register för GPIOB.

## Sätt en brytpunkt

```
80  /*Configure GPIO pins : PBPin PBPin PBPin PBPin */
81  GPIO_InitStruct.Pin = LED1_Pin|LED2_Pin|LED3_Pin|LED4_Pin;
82  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
83  GPIO_InitStruct.Pull = GPIO_NOPULL;
84  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
85  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

GPIOB	
> <b>MODER</b>	0x48000400 0xfa7ffe97
> <b>OTYPER</b>	0x48000404 0x0
> <b>OSPEEDR</b>	0x48000408 0x0
> <b>PUPDR</b>	0x4800040c 0x100
> <b>IDR</b>	0x48000410 0x10
> <b>ODR</b>	0x48000414 0x0
> <b>BSRR</b>	0x48000418 0x0
> <b>LCKR</b>	0x4800041c 0x0
> <b>AFRL</b>	0x48000420 0x0
> <b>AFRH</b>	0x48000424 0x0
> <b>BRR</b>	0x48000428 0x0
> <b>ASCR</b>	0x4800042c 0x0

Gå till SFRs i debugfönstret och scrolla ned till GPIOB

Kör programmet fram till brytpunkten och stega sedan över anropet till HAL\_GPIO\_Init. Du kommer att se att MODER ändras. GPIO mode register kommer att ställa in om en I/O-pinne ska vara ingång, utgång, alternativ funktion eller analog funktion. Det kan vara bra att ha lite koll på vad som händer i initieringen. Initieringsprogrammet STM32CubeMX fixar all initiering som vi ställer in i perspektivet STM32CubeMX. Behöver du initiera om något kan du gå tillbaka till initieringen genom att dubbelklicka på Pingpong.ioc i Project explorer.

Nu kan det vara dags att skriva egen kod. Funktioner som du kan anropa hittar du i HAL-manualen UM1786 under kapitel 21 HAL GPIO Generic Driver. Där finns en steg för steg-beskrivning i hur drivrutinen används. Eftersom all initiering är klar kan vi gå direkt på att testa att tända och släcka lysdioder. Ett tips! Det går bra att kopiera text ur en pdf-fil!

```
while (1)
{
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET); // Led 1
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);

    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_SET); // Led 2
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);

    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_SET); // Led 3
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);

    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_SET); // Led 4
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);

    HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_SET); // Led 5
    HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);

    HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_SET); // Led 6
    HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);

    HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_SET); // Led 7
    HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);

    HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_SET); // Led 8
    HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
}
```

Testa att kompilera och köra programmet. Nu ser det ut som att alla dioder lyser samtidigt. Eftersom de tänds och släcks så snabbt hinner vi inte uppfatta att de blinkar. Sätt en brytpunkt i början av while-loopen och testa att stega igenom loopen (step over) så ser du att varje diod tänds och släcks. while (1) är en evighetsloop som aldrig tar slut.

Du kan också testa att lägga en HAL\_Delay(50); efter att en lysdiod tänds och före att den släcks. Gör det för alla lysdioder.

Nu kan vi konstatera att det går att tända och släcka respektive diod. Fungerar det inte får du felsöka. Finns felet i programmet eller är det fel på hårdvaran? Om någon diod inte tänds går det att mäta med oscilloskop på utpinnarna om spänningen är den rätta. När jag gjorde det här exemplet första gången visade det sig att en lysdiod inte tändes.

Jag hade slarvat med lödningen av kortet och kunde åtgärda det. Alla kort som vi tillverkat har testats så de borde fungera.

Förhoppningsvis har du nu en hyfsad grundläggande förståelse för hur API<sup>14</sup> för GPIO används. Du kommer att ha anledning att studera datablad, referensmanualen och manualen för HAL-biblioteket när du utvecklar dina egna program.

## Testdriven programutveckling

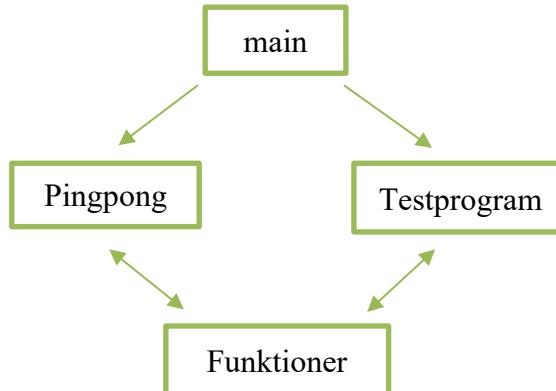
När vi utvecklar programmet skall vi försöka skriva felfri kod, eller i alla fall kod med så få buggar som möjligt. Det vanligaste sättet att utveckla program är att skriva koden först och sedan testa om den fungerar. Problemet är att kunna testa programmet och rätta fel utan att introducera nya fel. Här kommer nu att introduceras ett annat sätt att tänka; utveckla först ett program för testning och utveckla sedan kod som uppfyller de krav vi ställer på programmet, det vill säga klarar testet.

**Se till så att du sparar de testprogram du utvecklar så att du när som helst kan gå tillbaka och köra testprogrammen för att förvissa dig om att koden fungerar som avsett. Alla funktioner som du utvecklar i kursens programmeringsprojekt ska när som helst kunna testas med testprogram.**

Vi vill nu

- utveckla testprogram för att testa funktionerna som ska utvecklas
- utveckla funktioner i ett funktionsbibliotek som vi använder som beståndsdelar i Pingpong-programmet
- kunna gå tillbaka och köra testprogrammen om vi ändrar i funktionerna

Du ska nu för att fortsätta utvecklingen av pingpong-programmet utveckla två parallella program. Det första programmet är där du testar de funktioner som ska ingå i det slutliga Pingpong-programmet och det andra programmet är det färdiga Pingpong-programmet.



Vi ska utveckla två versioner av programmet. Det första vi ska göra är att bestämma vilka funktioner vi vill ska ingå i det program vi utveckla, dvs. pingpongprogrammet. Vi börjar med att utveckla testprogram så att vi kan testa funktionerna som ska utvecklas. När funktionerna testats och klarar testet kan funktionerna införlivas i det slutliga programmet. Skulle vi senare komma på att vi vill ändra i en funktion så kan vi gå tillbaka och köra om testet för att förvissa oss om att funktionen fortfarande fungerar

<sup>14</sup> API = Application Programming Interface

korrekt. På detta sätt kommer de funktioner som testas att vara **exakt samma** som de som används i färdiga programmet.

Testprogrammet ska inte ingå i det färdiga programmet utan endast användas under utveckling. Genom att använda lämpliga preprocessor-direktiv kan vi styra vilken C-kod som kommer att ingå när koden kompileras.

Definiera ett makro:

```
#define RUN_TEST_PROGRAM
```

Kan läggas under Private define, till exempel så här:

```
/* Private define -----*/
/* USER CODE BEGIN PD */
#define RUN_TEST_PROGRAM
/* USER CODE END PD */
```

Definitionen av makrot kan avaktiveras med

```
#undefine RUN_TEST_PROGRAM
```

Genom att testa om makrot RUN\_TEST\_PROGRAM är definierat kan vi styra vad som kompileras. Skriv följande kod under `/* USER CODE BEGIN 2 */`:

```
/* USER CODE BEGIN 2 */
#ifndef RUN_TEST_PROGRAM
    Test_program();
#else
    Pingpong();
#endif
/* USER CODE END 2 */
```

Eftersom utskrift med printf till konsolen bara ska ske i testprogrammen kan vi inkludera de extrakommandon som behövs i undantaget i kompileringsdirektivet.

```
/* Private define -----*/
/* USER CODE BEGIN PD */
#define RUN_TEST_PROGRAM
/* USER CODE END PD */

/* USER CODE BEGIN 0 */

#ifndef RUN_TEST_PROGRAM
extern void initialise_monitor_handles(void);
#endif

/* USER CODE END 0 */
```

Funktionerna Test\_program och Pingpong är ännu inte skrivna. Innan vi gör det kan vi planera hur filer och funktioner ska vara organiserade. Det kan vara lämpligt att dela upp all kod som ska skrivas i olika filer. Det är lämpligt att lägga funktioner som hör samman i en och samma fil. Skapa följande filer i ditt projekt

pingpong.c *här läggs pingpongprogrammet*

pingpong\_functions.c *här läggs alla funktioner som hör till pingpong*

test.c *här läggs testprogrammet*

Högerklicka på Src i Project Explorer och skapa c-filerna. Högerklicka på Inc och skapa tillhörande headerfiler Pingpong.h och Test.h.

Arbetsgången blir följande:

1. Vi behöver en funktion som tändar en lysdiod.
2. Skapa ett testprogram som kan testa funktionen.
3. Skriv funktionen.
4. Kör testet.
5. OK om testet klaras, annars gå tillbaka och ändra.

Lägg till följande tomta funktion i filen pingpong.c. Tänk på att motsvarande deklaration måste finnas i headerfilen.

```
void Pingpong(void)
{
} // End of function Pingpong
```

Lägg till följande tomta funktion i filen pingpong\_functions.c. Tänk på att motsvarande deklaration måste finnas i headerfilen.

```
void Led_on(uint8_t Lednr)
{
    return;
} // End of function Led_on
```

Lägg till följande funktioner i test.c, samt motsvarande deklarationer i headerfilen. Funktioner måste finnas deklarerade innan de används. Genom att lägga deklarationer i en headerfil (.h) kan de enkelt inkluderas i de filer där de används. Definitionen av funktionerna ligger i motsvarande källkodsfil (.c) som kompileras och länkas samman med övriga delar av programmet.

```
void Test_program(void)
{
    Test_Led();
}

void Test_Led(void)
{
    int8_t Lednr;
    /* Loop checking that all leds can be turned on*/
    for (Lednr=1; Lednr<= 8; Lednr++) {
        Led_on(Lednr);
        HAL_Delay(500);
    }
    Led_on(9); // Turn off led 8
    HAL_Delay(1000); // 1000 ms
    return;
}
```

Nu finns ett testprogram **Test\_program** som anropar ett test **Test\_Led**. Du kan nu köra testet. Sätt en brytpunkt i **Test\_program** på raden för **Test\_Led**. Kör programmet fram till brytpunkten och använd kommandot **Step Over** i debuggern för att köra en testfunktion i taget.

Det tänds dock inga lysdioder eftersom funktionen **Led\_on** inte är färdigskriven. Körningen av testet blir lite halvautomatiskt eftersom vi själva måste titta på pingpong-kortet för att se om lysdiодerna tänds och släcks.

Det är dags att skriva klart **Led\_on**:

```
void Led_on(uint8_t Lednr)
{
    uint8_t i;
    for (i=1; i<= 8; i++)
    {
```

```

switch(i){
    case 1 : // Led 1
        if (Lednr==i) HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
        else HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
        break;
    case 2 : // Led 2
        if (Lednr==i) HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_SET);
        else HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
        break;
    case 3 : // Led 3
        if (Lednr==i) HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_SET);
        else HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
        break;
    case 4 : // Led 4
        if (Lednr==i) HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_SET);
        else HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_RESET);
        break;
    case 5 : // Led 5
        if (Lednr==i) HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_SET);
        else HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, GPIO_PIN_RESET);
        break;
    case 6 : // Led 6
        if (Lednr==i) HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_SET);
        else HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, GPIO_PIN_RESET);
        break;
    case 7 : // Led 7
        if (Lednr==i) HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_SET);
        else HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, GPIO_PIN_RESET);
        break;
    case 8 : // Led 8
        if (Lednr==i) HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_SET);
        else HAL_GPIO_WritePin(LED8_GPIO_Port, LED8_Pin, GPIO_PIN_RESET);
        break;
    default :
        ;
}
} // end switch
} // end for-loop
return;
} // end function Led_on

```

Testkör igen. Om du nu har fått programmet att fungera skall det tända en lysdiod åt gången. Testet slutar med att alla lysdioder är släckta efter Led\_on(9).

Vi vet nu att vi har ett fungerande anrop av en funktion Led\_on. Led\_on är nu godkänd för att kunna användas i programmet Pingpong!

## Dokumentation med Doxygen

Nu ska vi kommentera upp koden och dokumentera den lite bättre. Vi ska använda ett system för att kommentera så att vi kan generera dokumentation med programmet Doxygen. Det är lite av standard att använda Doxygen för att generera dokumentation ur källkod<sup>15</sup>. Det finns lite olika sätt att markera för Doxygen vilka kommentarer som ska extraheras ur källkoden. Jag använder här att ett kommentarsblock för Doxygen börjar med `/**`. Det är lite skilda meningar om doxygenkommentarerna ska skrivas i källkoden (.c) eller i headerfilen (.h). Jag har valt att lägga kommentarer i källkodsfilerna.

För ytterligare information om vilka kommentarer som genererar information till Doxygen hänvisas du till dokumentation i Doxygen<sup>16</sup>.

Tag för vana att direkt när du skapar programmet skriva en kommentarstext i början av filen, kommentarstext för varje funktion och att kommentera programkoden på ett vettigt sätt, dvs. beskriv vilka uppgifter kodavsnitten utför. För att undvika blandning av

---

<sup>15</sup> Källkod är textfilen med programkod, i det här fallet programkod i språket C (.c)

<sup>16</sup> Doxygen <http://www.doxygen.nl/>

engelska och svenska väljer jag att skriva kommentarer på engelska. Tänk på att kod som utvecklas skall kunna förstås och underhållas av en annan programmerare.

Ett filhuvud<sup>17</sup> kan se ut så här:

```
/**  
*****  
@brief Pingpong, functions for Pingpong-program  
@file pingpong_functions.c  
@author Bengt Molin  
@version 1.0  
@date 12-August-2019  
@brief Functions and structures for program Pingpong  
*****  
*/
```

Ändra också kommentarshuvudet<sup>18</sup> för funktionen Led\_on så att det anpassar sig till Doxygen-standard:

```
/**  
@brief Led_on, turn one of the pingpong leds on  
  
@param uint8_t Lednr , number to the Led that is turned on  
Lednr can be 1-8, all other values turns all leds off  
  
@return void  
*/
```

Om du nu tycker att det var för många rader med case-satsen i Led\_on så kanske du hittar någon smartare lösning. Då har du ett testprogram som bara testar den funktionen. Programspråket C tillåter att man skriver väldigt kryptisk och svår läst kod. Det är dock bättre att skriva lättförståelig kod även om det blir fler rader.

## Skapa ett funktionsbibliotek

Eftersom vi har skapat en egen fil för alla funktioner som ska ingå i pingpong-programmet så utgör den filen också ett funktionsbibliotek. Källkodsfilen (.c) ska ligga i /Src och hederfilen (.h) i /Inc. Trolig arbetsgång:

- ✓ Skriv funktionerna till egen fil pingpong\_functions.c
- ✓ Kopiera funktionsdeklarationerna till en headerfil pingpong\_functions.h
- ✓ Skriv kommentarshuvud i filerna enligt Doxygen-standard
- ✓ Inkludera headerfilen i lämplig fil i ditt projekt

Funktioner måste finnas deklarerade innan de används. Genom att lägga deklarationer i en headerfil (.h) kan de enkelt inkluderas i de filer där de används. Definitionen av funktionerna ligger i motsvarande källkodsfil (.c) som kompileras och länkas samman med övriga delar av programmet.

Du kan få komplettera med att inkludera lämpliga headerfiler innan du får det att fungera, men hur du gör det överläter jag till dig att lista ut.

---

<sup>17</sup> Kommentarstext i början av en fil för att förklara vad filen innehåller, vem som skrivit och versionsnummer.

<sup>18</sup> Placeras för definitionen av funktionen.

Vill vi minska innehållet i main-filen kan vi flytta initieringen av klockan och lägga den i en egen fil clock.c och clock.h

Nu kan hela main.c bli så här kort om jag raderar onödiga kommentarer och User code parenteser. Observera dock att du bör behålla User code parenteserna, så att du kan generera om kodken i perspektivet STM32CubeMX.

```
#include "main.h"

#define RUN_TEST_PROGRAM

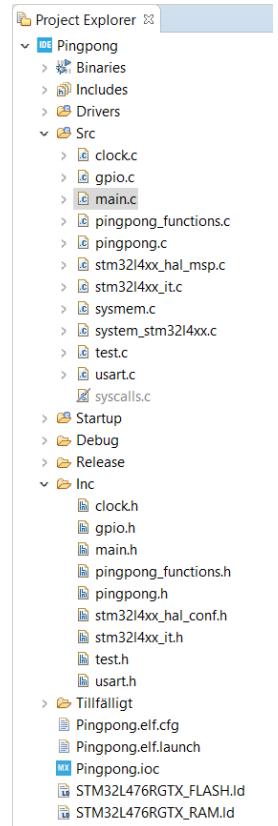
int main(void)
{
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* Configure the system clock */
    SystemClock_Config();

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();

#ifdef RUN_TEST_PROGRAM
    Test_program();
#else
    Pingpong();
#endif

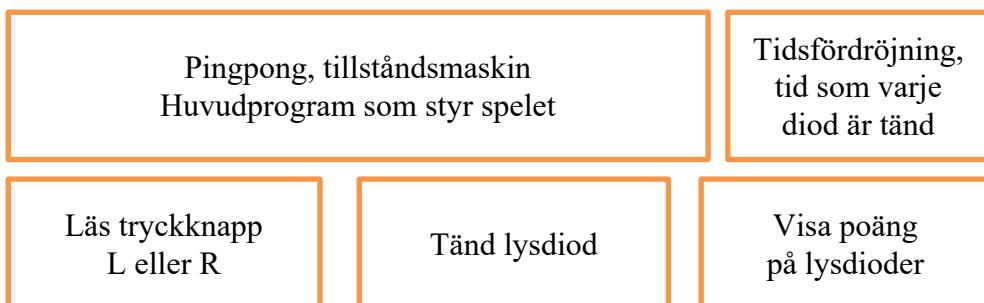
    while (1)
    {
    }
}
```



Nu har vi kommit en bit på väg och det kan vara dags att reflektera över arbetsgången vid den fortsatta utvecklingen av programmet Pingpong. Vi skall använda följande utvecklingsmodell vid utveckling av programvaran för ett inbyggt system.

- ✓ Skissa arkitektur för mjukvaran
- ✓ Utveckla tester och skriv testprogram för programmoduler
- ✓ Utveckla programmoduler till dess de klarar testen
- ✓ Integrera programmoduler till färdigt program
- ✓ Testa färdigt program

Arkitektur för pingpongprogrammet, blockschema för programmet



**Tänd lysdiod**, har vi redan gjort klart och testat

**Visa poäng**, när en boll är missad skall alla lysdioder blinka till lite kort (0,1 s) varefter poängställningen visas genom att kortvarigt (1 s) tända upp så många lysdioder som motsvarar poängställningen.

**Läs tryckknapp**, för serve eller retur av "boll"

Tidsfördröjning, vi behöver en tidfördröjning där fördröjningen kan ändras för att bestämma hastigheten på bollen. Vi kan göra denna på olika sätt; som en fördröjningsloop, använda en timer eller enklast att använda HAL\_delay.

**Huvudprogrammet Pingpong** hade jag tänkt ska skrivas som en tillståndsmaskin. Vi återkommer till den strukturen lite senare.

Vi fortsätter med programutvecklingen. Eftersom vi vet att det fungerar att tända lysdioder kan vi fortsätta med en funktion för att visa poäng. Fortsätt att jobba med projektet Test och skapa en funktion Show\_points som visar poängen.

### Funktionen Show\_points

1. Skriv först ett testprogram som kan visa att Show\_points fungerar. Det skall gå igenom alla poäng som kan inträffa för vänster spelare och höger spelare. Poäng som kan visas är 1, 2, 3 och 4. Maxpoäng är fyra. I det här läget ska det finna en funktion Show\_points som kan anropas, men innehållet är tomt.
2. Dags att ge innehåll åt Show\_points. Skriv en funktion Show\_points som uppfyller testen enligt ovan.

```
/**  
@brief Show_points, shows points after missed ball  
@param1 uint8_t L_points points for left player  
@param2 uint8_t R_points points for right player  
@return void, no return value  
*/  
void Show_points(uint8_t L_points, uint8_t R_points)
```

Poängen skall visas genom att tända det antal lysdioder som svarar mot poängen. Dioderna skall tändas i ordning utifrån och in mot mitten. Exempel: Om ställningen är L=1 och R=1 skall lysdiod 1 och 8 vara tänd. Om ställningen är L=2 och R=3 skall lysdioderna 1,2 och 6,7,8 vara tända.

Kör testet och verifiera att funktionen Show\_points fungerar korrekt. Nu har vi funktioner för att hantera tändning av lysdioderna för att simulera en vandrande boll och att visa poängställning.

### Läsa av tryckknapparna

Nu kan det vara dags att läsa av tryckknapparna. Tryckknappen är en switch som sluter inspänningen på PC8 respektive PB15 till noll volt (jord). När knappen inte trycks ned hålls spänningen på ingångarna PC8 respektive PB15 hög av ett internt pullup-motstånd till matningsspänningen V<sub>DD</sub>. Se figur på sidan 11. Ingångarna ska således initieras till pull up.

PC8 och PB15 är redan initierade till pull up i funktionen MX\_GPIO\_Init som skapades under initieringen i perspektivet STM32CubeMX.

Vad ska häcka när man trycker på en knapp? Hur ska vi registrera det?

Vi kan använda pollning. Det innebär att programmet hela tiden ligger och testar ingången. Så långt den är hög har knappen inte tryckts ned, men när den blir låg så är knappen nedtryckt. Vi skulle också kunna använda avbrott.

Läs User Manual UM1884, Description of STM32F3xx HAL drivers, avsnittet om GPIO-funktioner i kapitel 28 (HAL GPIO Generic Driver). Där hittar du den information du behöver om API<sup>19</sup> för GPIO<sup>20</sup>.

Med funktionen HAL\_GPIO\_ReadInputPin kan vi läsa värdet på en ingångspinne.

### Testprogram för att testa knapptryckningar

#### L\_hit och R\_hit

- Skriv nu funktioner L\_hit och R\_hit som testar om vänster respektive höger knapp är nedtryckt och returnerar en boolsk variabel som kan vara sann eller falsk. Funktionerna skall ligga i filen Pingpong\_functions.c så att de är tillgängliga när vi gör pingpong-programmet klart. Boolsk variabeltyp bool finns definierad i stdbool.h
- När vänster knapp (L) trycks ned skall den tända lysdioden ”röra sig” ett steg mot höger.
- När höger knapp (R) trycks ned skall den tända lysdioden ”röra sig” ett steg mot vänster.

Mata in följande testprogram i test.c för att testa knapptryckningar. Testa att köra koden.

```
void Test_buttons(void)
{
    int8_t j;

    /* Checking buttons */

    j=4;
    Led_on(j); // Light on

    while (j<9 && j>0)
    {
        if ( L_hit() == true ) // Wait for left button hit
        {
            j++;      // next led to the right
            Led_on(j); // Light on
            HAL_Delay(100); // 100 ms
            while ( L_hit() == true ); // Wait for button release
            HAL_Delay(100); // 100 ms
        }

        if ( R_hit() == true ) // Wait for right button hit
        {
            j--;      // next led to the left
            Led_on(j); // Light on
            HAL_Delay(100); // 100 ms
            while ( R_hit() == true ); // Wait for button release
            HAL_Delay(100); // 100 ms
            if (j<1) j=0; // Start again from left
        }
    }
}
```

---

<sup>19</sup> API Application Programming Interface

<sup>20</sup> GPIO General Purpose Input Output

```
        return;
}
```

Använd följande funktionsdeklarationer i Pingpong\_functions.c och Pingpong\_functions.h:

```
/**  
@brief L_hit, check if L button is pressed  
@param void  
@return bool, true if L button pushed, false otherwise  
*/  
bool L_hit(void)  
  
bool L_hit(void)  
{  
    if (HAL_GPIO_ReadPin(L_button_GPIO_Port, L_button_Pin) == 0) return true;  
    else return false;  
}
```

Gör på samma sätt med R\_hit.

För att bool skall fungera måste du lägga till #include "stdbool.h" på lämpligt ställe.

Testprogrammet bör nu se ut så här

```
void Test_program(void)  
{  
    Test_Led();  
    Test_Show_points();  
    Test_buttons();  
}
```

Vi kan enklast välja vilket test vi vill köra genom att kommentera bort de test vi vill avaktivera.

```
void Test_program(void)  
{  
//    Test_Led();  
//    Test_Show_points();  
    Test_buttons();  
}
```

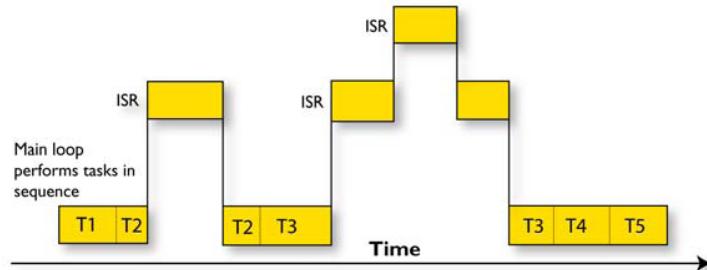
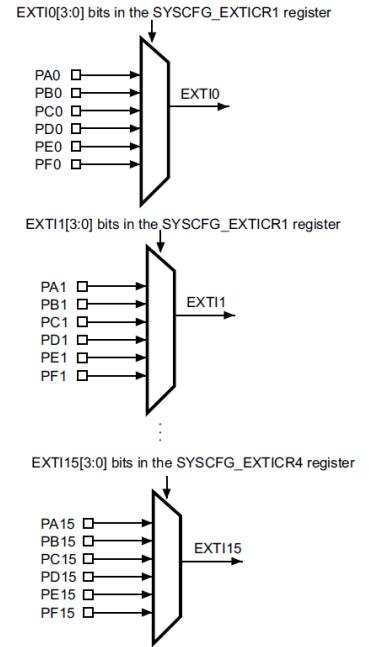
Kör igenom alla tre testerna och kontrollerar att du har en korrekt funktion innan du går vidare i häftet.

Nackdelen med pollning som vi använder här är att programmet blir upptaget av att hela tiden läsa av ingången. I just det här programmet kan det fungera bra eftersom mikrokontrollern inte har att utföra några andra uppgifter i väntan på att nästa lysdiod tänds.

Vi kan göra på ett annat sätt; vi kan låta tryckknapparna generera ett avbrott och sätta en flagga att knappen har tryckts ned.

Det finns inte en avbrotsvektor för varje I/O-pinne utan de är grupperade i 16 olika grupper, EXTI0 till EXTI15 för GPIO-pinnar Px0 till Px15. Med en multiplexer (styrts av bitar i kontrollregister) väljer man vilken grupp av pinnar som skall kunna generera avbrott, men vi vet inte vid ett avbrott vilken pinne som genererar det. Vi kan lösa det genom att i början av avbrotsrutinen läsa status på pinnarna för att se vilken knapp som tryckts ned. Hänsyn måste tas till att tryckknappen är en mekanisk kontakt som kan studsa mellan hög och låg några gånger innan det blir stabilt.

Att använda avbrott tillåter att programexekveringen kan, av en yttre händelse, styras till att exekvera en avbrotsrutin (ISR, Interrupt Service Routine). När avbrotsrutinen är klar återgår programmet till normal exekvering från den punkt där den blev avbruten. Avbrott används för tidskritiska händelser som kräver att programmet agerar så snabbt som möjligt.



Figur från IAR, Getting started, sidan 22. Superloop systems

Modeller för programexekvering kan delas in i

- ✓ Superloop systems (tasks are performed in sequence)
- ✓ Multitask systems (tasks are scheduled by an RTOS<sup>21</sup>)
- ✓ State machine models.

För att få avbrottshantering på STM32 att fungera måste du göra följande

- ✓ Adress till avbrotsrutinen (interrupt handler) måste vara definierad
- ✓ Konfigurera NVIC (Nested Vector Interrupt Controller)
- ✓ Konfigurera periferienhet som skall begära avbrott
- ✓ Skriva avbrotsrutinen, vad skall hända
- ✓ Tillåta avbrott från händelse på ingångspinnen
- ✓ Tillåta avbrott i huvudprogrammet

Det mesta av detta kan initieras genom att göra en ny initiering i perspektivet STM32CubeMx.

Det finns faktiskt en avbrotsrutin aktiv i programmet som vi jobbar med. Om du tittar i filen `stm32l4xx_it.c` så hittar du `SysTick_Handler`. I STM32 finns det ett flertal timers som kan användas för att räkna klockpulser och bestämma tid. Här används en system timer,

<sup>21</sup> RTOS Real Time Operating System, realtidsoperativsystem

SysTick, för att generera avbrott varje millisekund. Det är en 24 bitars nedräknare, som genererar avbrott när den räknat klart och som automatiskt laddar om startvärde (auto reload and end of count interrupt). Hur snabbt den räknar går att bestämma genom att programmera vilken källa som används som klocka och vilket värde som laddas i räknaren vid omstart. SysTick kan till exempel användas för att generera ett periodiskt avbrott för att växla process i ett realtidsoperativsystem, men vi kan använda den som bas för att bestämma när vi växlar tillstånd i vårt Pingpong-program.

I vårt program används system timer SysTick för att generera avbrott en gång per millisekund. Vid avbrott anropas `HAL_IncTick` som räknar upp en räknare uwTick. `HAL_Delay` använder uwTick för att räkna millisekunder. Den kan i Pingpongprogrammet användas för att räkna den tid varje lysdiod är tänd.

Adresserna till avbrottsrutinerna (avbrottsvektorn) ska ligga på förutbestämda adresser i minnet. När ett avbrott genereras av en händelse, i detta fall en timer, börjar avbrottrutinen exekveras på den adress som finns angiven på den förbestämda minnespositionen.

I filen `startup_stm32l476rgtx.s` finns följande rad som lägger in avbrottsvektorn (adress till avbrottsrutinen) i minnet.

```
.word SysTick_Handler
```

Filen är ett assemblerprogram som körs innan c-programmet startar. `.word` är ett assemblerdirektiv som allokerar plats i minnet och på platsen läggs adressen som anges. I filen `stm32l4xx_it.c` finns alla avbrottsrutiner. De är deklarerade men från början tomta. Utifall att ett avbrott av misstag aktiveras kan det vara bra att det finns en tom avbrottsrutin med ordnad retur istället för att programmet spårar ur.

Du kan undersöka hur `HAL_Delay` fungerar genom att markera `HAL_Delay`, högerklicka och välja Open Declaration. Genom att spåra bakåt några gånger kommer du till uwTick och hur den deklareras.

```
__IO uint32_t uwTick;
```

För att få veta vad `__IO` betyder kan du markera det, högerklicka och söka rätt på definitionen. Vi hittar deklarationen i filen `core_cm4.h`:

```
#define __IO volatile /*!< Defines 'read / write' permissions */
```

Det betyder alltså volatile, vilket i sin tur betyder att ett minne är flyktigt och kan ändras av andra programdelar. I detta fall eftersom det ändras till följd av avbrottsrutinen.

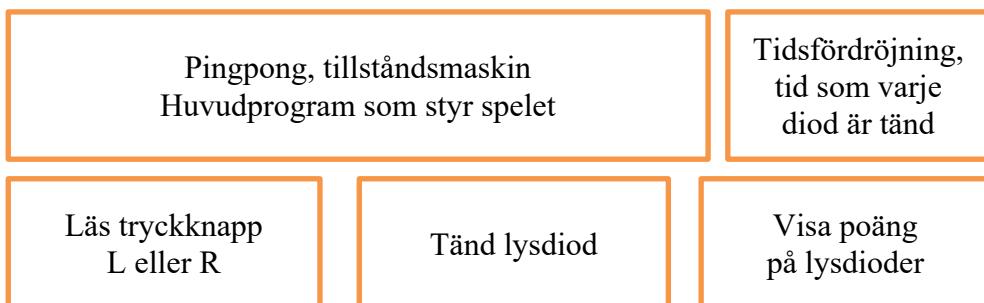
Innan vi går vidare tycker jag att du skall utvidga och strukturera upp testprogrammet lite.

Nu är det dags att fortsätta utvecklingen av Pingpong-programmet. Vi har nu tester för att testa att följande funktioner fungerar:

```
void Led_on(uint8_t Lednr);
void Show_points(uint8_t L_points, uint8_t R_points);
bool L_hit(void);
bool R_hit(void);
```

Alla funktioner finns i filen Pingpong\_functions.c. Samma fil skall användas när vi gör klart Pingpong-programmet. **Om du gör några ändringar i någon av funktionerna skall du gå tillbaka till testprogrammet och köra det för att testa att funktionerna fortfarande fungerar.**

Det kan vara dags att titta på vår planerade arkitektur för programmet och se hur långt vi har kommit. Arkitekturen för pingpongprogrammet, blockschemat för programmet har vi tidigare ritat så här:



Arkitektur för pingpongprogrammet, förfinat blockschema för programmet.



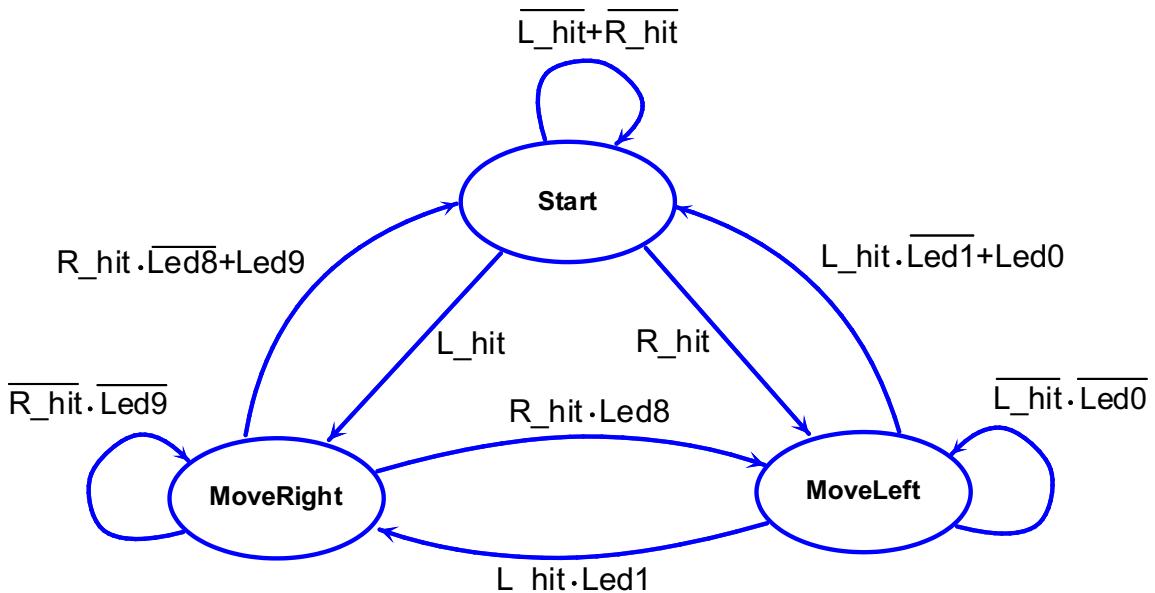
Nu är alla delar klara utom tillståndsmaskinen<sup>22</sup> som skall styra hela pingpong-spelet.

- ➡ Nu kan det vara dags att återgå till vår ursprungliga uppgift Pingpong, där vi ska göra klart pingpongspelet
- ➡ Låt testprogrammet vara kvar! Du behöver ha kvar testprogrammet för att kunna testa funktionerna utifall du gör några ändringar i dem.

Nu är det dags att ta fram papper och penna och fundera. Tillståndsdiagram har du kommit i kontakt med i digitalteknikkursen. Utgå från ett starttillstånd där programmet står och väntar på att någon skall starta pingpongspelet. Vi väntar på en serve! Fundera på vad som skall hända och i vilken ordning det skall göras.

**Ett första försök till tillståndsdiagram och hur det kan implementeras i vårt program**

<sup>22</sup> Eng. State machine, nästa tillstånd beror på nuvarande tillstånd och insignaler



Det här tillståndsdiagrammet är inte komplett och det är inte meningen att det skall vara det ännu. Jag kommer att gå igenom hur man kan skriva tillståndsdiagrammet i C-kod. Du kommer sedan att själv få göra klart Pingpong-spelet. Granska tillståndsdiagrammet kritiskt och fundera på om det är korrekt.

### Händelser

R_hit	trycker på höger knapp
L_hit	trycker på vänster knapp
Led1	Lysdiod 1 lyser
Led8	Lysdiod 8 lyser
Led0	Missad retur av vänster spelare
Led9	Missad retur av höger spelare

### Tillstånd

Start	Väntar på att någon skall starta spelet genom att serva
MoveRight	"Bollen", dvs lysande diod rör sig mot höger
MoveLeft	Lysande diod rör sig mot vänster

Du ska nu få ett Pingpong-program, som inte är helt färdigt, men visar hur programmet kan konstrueras. Lägg in följande kod i filen pingpong.c:

```

/*
*****
@brief  Pingpong statemachine for Pingpong-program
@author Bengt Molin
@version 1.0
@date   12-August-2019
@brief   Functions and structures for program Pingpong
*****
*/
/* Includes ----- */
#include "pingpong.h"

/* Define states for state machine*/
typedef enum
{
    Start,
    MoveRight,
    MoveLeft
}

```

```

} states;

static states State, NextState;

void Pingpong(void)
{
    bool ButtonPressed; // To remember that button is pressed

    uint32_t Varv, Speed; // Ball speed
    uint8_t Led; // LED nr

    State= Start; // Initiate State to Start
    NextState= Start;

    Speed= 500000; // Number of loops

    /* Infinite loop */
    while (1)
    {

        State = NextState;

        switch (State) // State machine
        {
            case Start:
                Led_on(0); // Turn off all LEDs

                if ( L_hit() == true ) // L serve
                {
                    Led = 1;
                    NextState= MoveRight;
                    while ( L_hit() == true ); // wait until button is released
                }
                else if ( R_hit() == true ) // R serve
                {
                    Led = 8;
                    NextState= MoveLeft;
                    while ( R_hit() == true ); // wait until button is released
                }
                else
                    NextState = Start; // Stay in Start state
                break;

            case MoveRight:
                {
                    Led_on(Led);
                    Varv = Speed;

                    while( Varv != 0 )
                    {
                        if ( R_hit() ) ButtonPressed = true; // R hit
                        Varv--;
                    }

                    if ( ButtonPressed ) // R pressed
                    {
                        if ( Led == 8 ) // and LED8 activa
                        {
                            NextState=MoveLeft; // return ball
                            Led=7;
                        }
                        else
                            NextState = Start; // hit to early
                    }
                    else
                    {
                        if ( Led == 9 ) // no hit or to late
                            NextState = Start;
                        else
                            NextState = MoveRight; // ball continues to move right
                    }
                    if ( !ButtonPressed ) Led++; // prepare to turn next LED on
                    ButtonPressed=false;
                }
                break;
        }

        case MoveLeft:
    }
}

```

```

{
    Led_on(Led);
    Varv = Speed;

    while(Varv != 0)
    {
        if ( L_hit() ) ButtonPressed = true; // L hit
        Varv--;
    }

    if ( ButtonPressed ) // L pressed
    {
        if ( Led == 1 ) // and LED1 active
        {
            NextState=MoveRight; // return ball
            Led=2;
        }
        else
            NextState = Start; // hit to early
    }
    else
    {
        if ( Led == 0 ) // no hit or to late
            NextState = Start;
        else
            NextState = MoveLeft; // ball continues to move left
    }
    if ( !ButtonPressed ) Led--; // prepare to turn next LED on
    ButtonPressed=false;
}

break;

default:
break;
}
}
} // End of function Pingpong

```

Ändra i main.c så att du kör Pingpong-programmet

```
#undef RUN_TEST_PROGRAM
```

Testkör koden. Använd Window / Show view / Live expressions. Studera värden på State och NextState under det att du kör programmet. Förbättra gärna programmet.

När vi lämnar start-tillståndet och hamna i MoveRight eller MoveLeft. Vad skall hända när vi ligger i MoveRight?

Vi ligger i en vänteloop så länge som lysdioden är tänd, men vi måste hela tiden polla om högra knappen trycks ned. När den tryckts ned skall vi testa om det är lysdiod 8 som lyser, i så fall returneras "bollen". Om det inte är lysdiod 8 när R tryckts ned har R tryckt för tidigt eller så är det en missad boll, i båda fallen är det poäng till motståndaren. Vi måste också se till att vi får en vandrande diod med lagom lång lystid.

I tillståndsdiagrammet enligt föregående sida går programmet tillbaka till start vid missad boll.

- ⊕ Gå igenom koden och undersök om den uppfyller specifikationen enligt tillståndsdiagrammet ovan så att det går att returnera "bollen" endast på yttersta positionerna.
- ⊕ Om programmet inte fungerar enligt tillståndsdiagrammet, rätta det till dess det fungerar.

Nu kan det vara dags att göra klart programmet så att det uppfyller den specifikationen som sattes upp tidigare. Vi kan repetera specifikationen:

Spelets regler är följande:

Det finns två spelare, L och R (Left och Right). Den som servar trycker på sin knapp. Då tänds lysdioder, en i taget, för att animera att en boll rör sig till motspelaren. En boll returneras om knappen trycks ned samtidigt som yttersta dioden är tänd. Om man trycker för tidigt eller för sent är det en tappad boll och motspelaren får poäng. En missad boll markeras genom att alla lysdioder blinkar till lite kort, varefter poängställningen visas. Poängställningen efter en vunnen och tappad poäng visas ett kort ögonblick genom att tända upp så många lysdioder på respektive sida som motsvarar antal poäng. Spelarna turas om att serva. Vid start kan vem som helst serva, men det är den andra spelaren som servar nästa gång. Om till exempel vänster spelare servar första gången är det höger spelare som servar nästa gång oberoende av vem som tappat poäng. Den som först kommer till fyra vunna poäng vinner. Resultatet ska visas under fem sekunder innan spelet startas om. Den tid varje lysdiod är tänd skall minska allteftersom beroende på hur länge en boll varit i spel. På så sätt ökar ”bollhastigheten” och det blir allt svårare att hinna returnera bollen. Det kan vara lämpligt att sätta en maximal bollhastighet.

- ⊕ Gör klart tillståndsdiagrammet så att det uppfyller spelets regler. Lägg till fler tillstånd och villkor för övergångar. **Obs! Rita klart tillståndsdiagrammet innan du kodar.** Det är här du tänker ut hur programmet skall fungera. Använd diagrammet på svarsblanketten som utgångspunkt.
- ⊕ Gör klart programmet så att det stämmer överens med det nya tillståndsdiagrammet. Testkör programmet!

När programmet är klart kan du utmana någon i din omgivning på en match!

När funktionen Pingpong avslutas kommer man tillbaka till huvudprogrammet och är redo för en ny match!

Det program du redovisar ska uppfylla specifikationen, men du kan fundera på om det går att förbättra programmet på någon punkt.

Du kan till exempel dela upp Show\_Points i en funktion Blink\_LEDs och Show\_Points så att delen som snabbt blinkar alla dioder blir en egen funktion. Glöm inte lägga till test på Blink\_LEDs och att köra om testen när du har ändrat i en funktion.

Du skulle också kunna låta en lysdiod lysa lite svagt genom att blinka snabbt med liten duty cycle för att visa vems tur det är att serva.

- ⊕ **Ett tips så här i slutet av denna skrift.**

Lär dig använda debuggern. Sätt brytpunkter och undersök variablers värde. Lär dig stega, steg in i och stega över funktioner. Om du använder Live Expressions kan du se variabelvärden under programföring.