

Lab Pingpong - Introduktion till utvecklingsmiljö och programmering av mikrokontroller STM32

Detta dokument beskriver steg-för-steg hur utvecklingsmiljön Embedded workbench används, programmering av STM32F3 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. Resultatet redovisas för lärare på laboration 2. Pingpongprogrammet skall vara klart när laboration 2 börjar. Du får andra uppgifter att gå igenom under laboration 2. Räkna med att det tar flera dagars heltidsjobb att gå igenom häftet. Att laboration 2, dvs. detta häfte, är godkänt är ett krav för att få fortsätta kursen.

Innan du börjar denna guide skall du

- Ha ett STM32F3 Discoverykort och ett Pingpongkort tillgängligt
- Hämta IAR Embedded workbench for ARM, kickstart edition (www.iar.com EWARM, 32k size limited) och installera det på din dator. Alternativt kan du använda andra utvecklingsmiljöer. Du hittar ett urval via ST Microelectronics webb eller genom att googla *stm32 c compiler*. Anvisningarna i detta häfte baseras på IAR Embedded workbench.
- Installera drivrutin för ST-LINK. ST-LINK är det debuginterface som finns på kortet. Det används för att ladda ned program och för att kunna styra program-exekveringen 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 hämtar du hem följande

- UM1570 User manual, Discovery kit with STM32F303VC MCU
- Datablad för mikrokontrollerkretsen STM32F303VCT6
- Referensmanual RM0316 för mikrokontrollern
RM0316: STM32F303xB/C/D/E, STM32F303x6/8, STM32F328x8, STM32F358xC, STM32F398xE advanced ARM®-based MCUs
- STM32CubeF3, zip-fil som du packar upp där du vill spara dina program för mikrokontroller STM32F3, innehåller HAL¹-bibliotek och programexempel
- STM32CubeMX, hämta och installera detta program som skapar initieringskod
- UM1766: Getting started with STM32CubeF3 for STM32F3 Series
- UM1786: Description of STM32F3xx HAL drivers and low-layer drivers
- 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 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!

¹ HAL = Hardware Abstraction Layer

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
- Köra ett program STM32CubeMX som genererar initieringskod
- Skapa ett Pingpong-program, strukturerad som en tillståndsmaskin

Redovisning laboration 2

- Alla program som du har gjort enligt detta häfte redovisas för läraren.
- Gör alla uppgifter och svara på alla frågor i svarshäftet.
- Du skall kunna svara på frågor om alla programdelar.

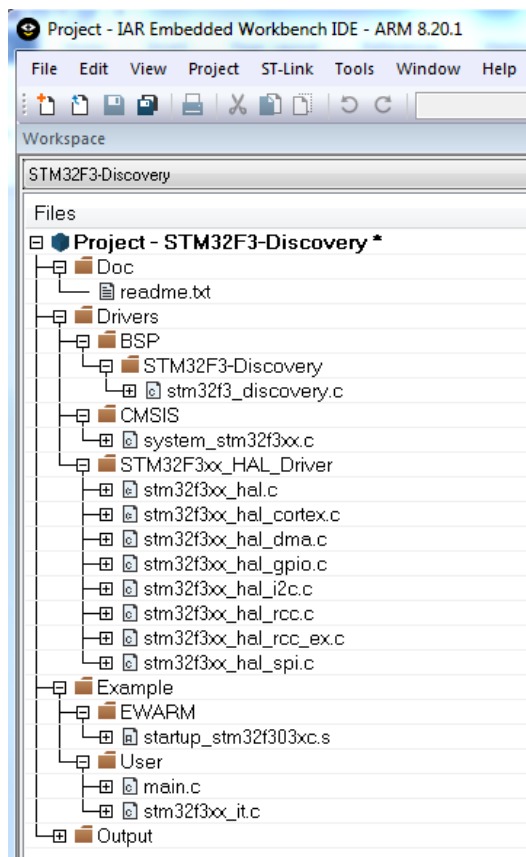
🚦 Studera ett färdigt exempel

Vi börjar med att hämta och studera ett färdigt exempel.

Starta IAR Embedded workbench.

File > Open Workspace... öppna *GPIO_IOToggle* enligt sökväg nedan

`\STM32Cube_FW_F3_V1.9.0\Projects\STM32F3_Discovery\Examples\GPIO\GPIO_IOToggle\EWARM\Project.eww`



Projektfönstret visar vilka filer som ingår i projektet. Det visar inte hur de är organiserade på hårddisken. Man kan skapa vilka grupper man vill och plocka in filer som man vill, men det är dock bra att hålla isär vilka typer av filer som ingår genom att organisera dem i grupper.

Under drivers ligger de filer som ingår i biblioteken. Dessa måste finnas tillgängliga och du skall inte ändra i dem.

BSP (Board Support Package) innehåller sådant som är specifikt för den mikrokontroller och det kort vi använder

CMSIS² (Cortex Microcontroller Software Interface Standard) är specifikt för ARM Cortex-kärnan.

Startupfilen är en fil som sätter stackpekare, programpekare och lägger in adresser för avbrottsvektorer. Denna fil är en assemblerfil som exekveras före allt annat.

User Här lägger du de filer som innehåller din egen programkod. Filen `main` ska innehålla funktionen `main` där programexekveringen börjar. För att lägga till filer högerklickar du på **User** och lägger till de c-filer du önskar. Filen `stm32f3xx_it.c` innehåller avbrottsrutiner (interrupt handlers).

² The ARM® Cortex™ Microcontroller Software Interface Standard (CMSIS) provides a single, scalable interface standard across all Cortex-M series processor vendors which enables easier code re-use and sharing across software projects to reduce time-to-market for new embedded applications

Om du skall starta ett helt nytt projekt skall du göra följande

- Skapa workspace
- Skapa ett nytt projekt
- Sätta options för projektet
- Lägga till källfiler
- Göra inställningar för det debugverktyg som används
- Kompilera
- Länka

Vi kommer att använda ett programverktyg STM32CubeMX som genererar initieringskod och skapar ett workspace med alla inställningar. Det underlättar att komma igång med egen programmering men det kan ändå vara av intresse att förstå de inställningar som används utifall du får problem.

Granska inställningar (Options) för projektet

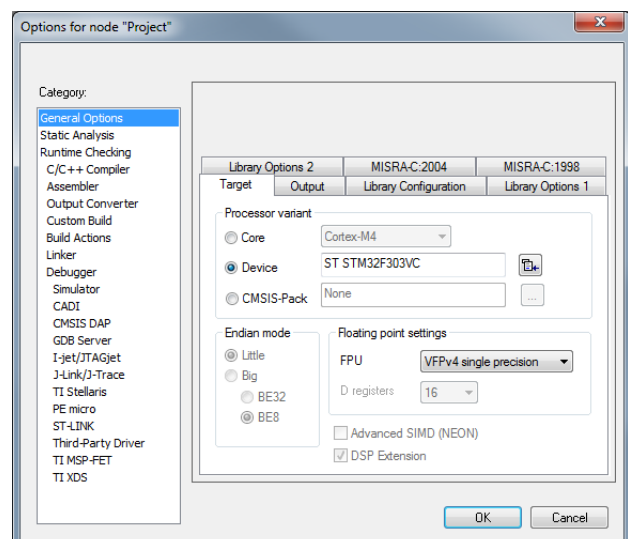
Markera projektmappen och högerklicka, välj Options



Vi kommer att gå igenom de olika alternativen och flikarna. Du behöver inte ändra något utan behåll standardinställningarna.

General options

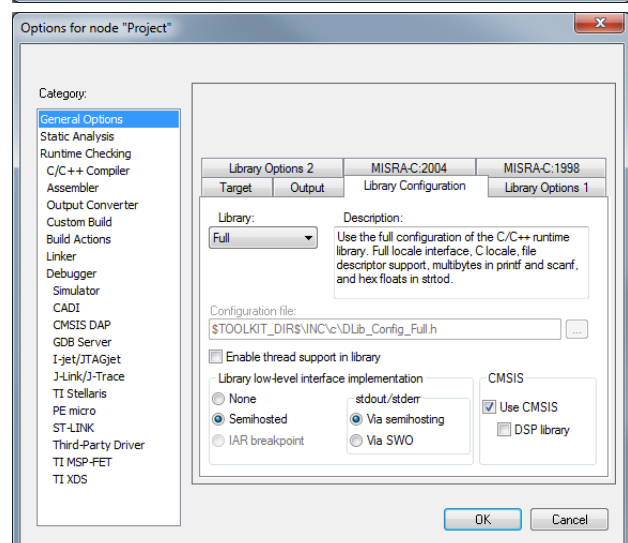
Krets STM32F303VC (den krets som sitter på discoverykortet är STM32F303VCT6)



CMSIS är vald

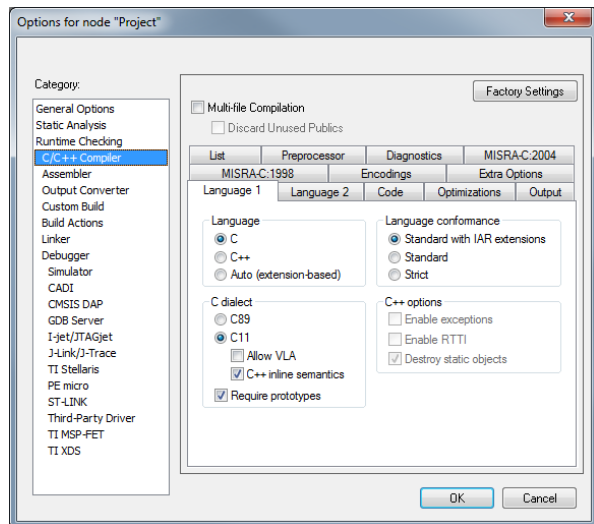
CMSIS är en standard som skall underlätta att flytta kod mellan olika ARM-baserade microcontrollers från olika tillverkare

The ARM® Cortex™ Microcontroller Software Interface Standard (CMSIS) provides a single, scalable interface standard across all Cortex-M series processor vendors which enables easier code re-use and sharing across software projects to reduce time-to-market for new embedded applications

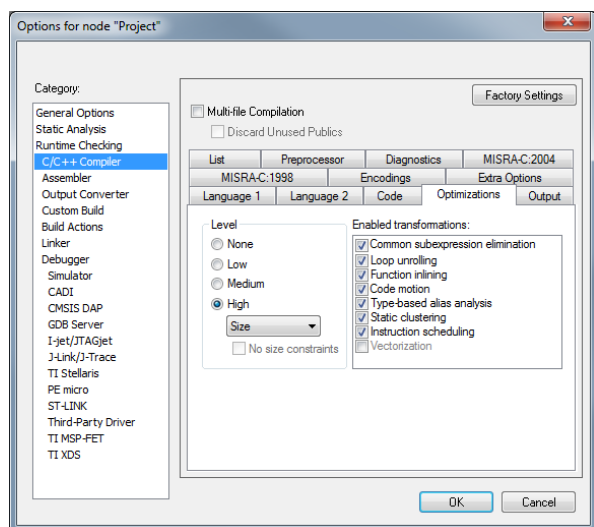


C/C++ Compiler


Programmet är skrivet i standard C, så det kan vara lämpligt att det är markerat att standard för språket C enligt C99 skall följas.



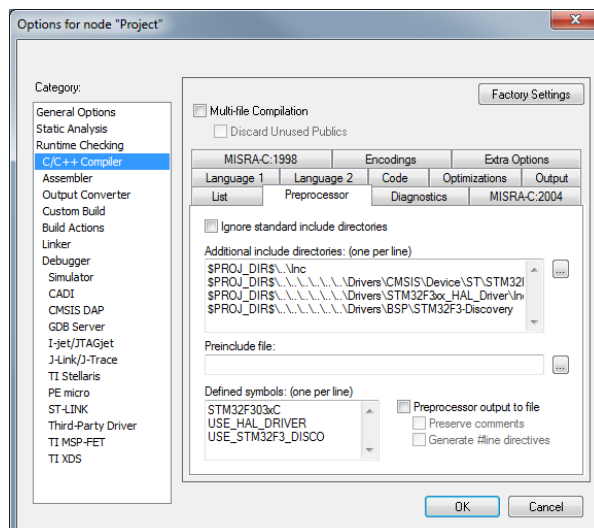
Du kan välja hur kompilatorn skall optimera koden. Ibland kan man få problem att debugga koden om det är för hög optimeringsgrad. Då kan du prova att gå ned till Medium eller Low.



Under fliken Preprocessor specificerar man sökvägar till de filer som skall inkluderas i projektet. c-filer lägger du till under User i projektfönstret. Sökväg till header-filer som skall inkluderas anger du här.

Du kan lägga till filer sökvägar genom att trycka på 

Sökvägarna bör vara relativa till den katalog där projektet ligger
\$PROJ_DIR\$

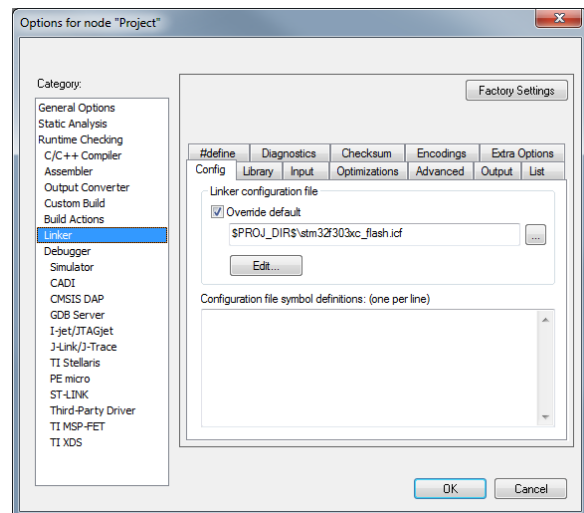


Sökvägarna beror på hur du har organiserat dina filer på hårddisken.

Man kan definiera symboler här som kan användas för att styra kompileringen.

Linker

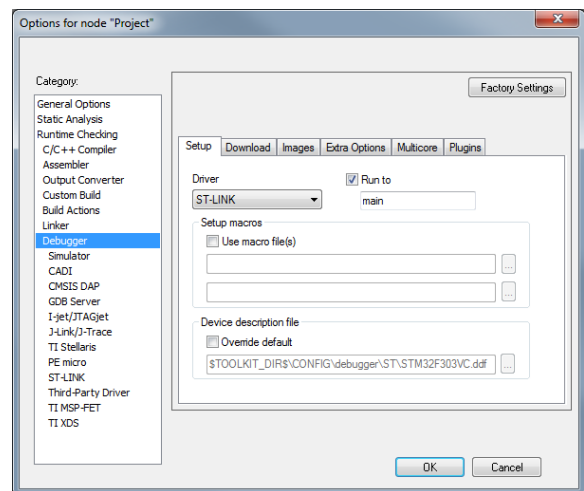
Linker är det program som länkar samman alla filer som ingår för att få en körbar kod som kan laddas ned i kretsen och exekveras. Filen stm32f303xc_flash.icf lagrar information om var interrupt-vektorer startar, var RAM³ och ROM⁴ börjar och slutar (måste stämma med kretsen förstås) och hur RAM används till stack och HEAP⁵. Du kan använda standardfilen som finns för den mikrokontroller vi använder.



Debugger

Den debugger som används för att programmera kretsen och styra exekveringen av programmet är ST-LINK. Hårdvaran för den finns på Discovery-kortet. Vi behöver således inte någon fristående programmerare.

Om du väljer Simulator kan du kompilera och testköra C-kod utan att programmera en mikrokontroller.



Under fliken Download ska **Use flash loader(s)** vara markerat. Får du problem med att programpekaren inte pekar på main efter att du laddat ned programmet har du antagligen inte markerat denna!

Välj ST-LINK lite längre ned under Debugger, kontrollera att programmeringsgränssnittet SWD⁶ är markerat.

På STM32F3 Discovery sitter det en mikrokontroller närmast kontakten USB ST-LINK som ingår i ST-LINK programmerare och debugger. Det går att använda ST-LINK på kortet för att programmera en yttre mikrokontroller om den kopplas till kontakten SWD. Byglarna på kontakten CN4 skall i så fall tas bort. Nu skall vi emellertid programmera den mikrokontrollerkrets som sitter på Discoverykortet så låt byglarna CN4 sitta kvar.

³ Random Access Memory, egentligen skriv- och läsminne där variabler, stack och andra tillfälliga data lagras.

⁴ Read Only Memory eller flashminne, här lagras program och konstanter som inte ändras vid programkörning.


⁵ HEAP, ett minnesutrymme i RAM som en mikroprocessor kan skriva till och läsa från

⁶ Serial Wire Debug (SWD) is a 2-pin electrical alternative JTAG interface that has the same JTAG protocol on top. It uses the existing GND connection. SWD uses an ARM CPU standard bi-directional wire protocol, defined in the ARM Debug Interface v5. (Wikipedia)

Nedladdning och körning av 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.

Nu skall du, om alla inställningar är korrekt, ha ett program som skall gå att kompilera, länka och ladda ned i microcontrollern på discoverykortet.

Project > Download and Debug eller tryck på playknappen 

Om programpekaren efter att du laddat ned programmet inte pekar på huvudfunktionen main kan du försöka med Project > Rebuild all

Om du nu får andra problem får du försöka lista ut vad som är fel och prova igen.

Om allt går vägen så kommer du in i debugläge, dvs. du kan köra programmet i microcontrollern och samtidigt ha kontroll över körningen.



För muspekaren över knapparna så ser du vad de betyder

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

Step into – stegar programexekveringen och stegar även in i funktioner

Step out – stegar ut ur aktuell funktion

Next statement – exekverar fram till nästa programsats

Run to cursor – exekverar programmet fram till markören

Go – startar programexekvering utan att stega

Break – stoppar programmet

Reset – om du vill köra om programmet från start



Make & Restart Debugger – kompilerar om, laddar ned och startar debuggern

Restart Debugger – startar om debuggern utan att ladda ned programmet

Stop Debugging – stannar programexekvering och lämnar debugläget

Starta programmet (Go). Som du ser så tänds alla lysdioder i ringen i tur och ordning varefter de släcks i tur och ordning. Programmet fortsätter i evighet eftersom det är en evighetsloop while (1) i programmet. Program i mikrokontrollern måste alltid gå i en evighetsloop eftersom det inte finns något operativsystem att återvända till när ett programmet avslutas.

Stoppa programmet med Break och gör Reset. Nu är programmet redo att startas om.

Under debugging har du möjlighet att studera värden på register, variabler och minne. Under fliken View har du flera alternativ att visa vad som händer under programkörning. Om du använder Live watch kan du se variabelvärden under programkörning. För att Live watch ska fungera måste variabeln vara deklarerad som *static*. 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 variabelvärden är de rätta.

Du kan sätta brytpunkter i programmet. Programmet exekveras då fram till brytpunkten där det stannar så att du kan studera tillståndet i den punkten.

Brytpunkter (röd prick) sätter du eller tar bort genom att klicka i marginalen till vänster om programtexten.

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

Om du får problem med att det inte verkar fungera att sätta brytpunkter och stega exekveringen som du förväntar dig kan du gå tillbaka till options och minska nivån på optimering vid kompileringen.

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

startup_stm32f303xc.s

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

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

main.h

Headerfil för main.c

stm32f30xx_it.c

Interrupt handlers, i denna fil skriver du avbrottsrutinerna. Vi kommer att återkomma till den senare, men vi kan redan nu inkludera den i projektet.

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 deklarerar funktioner, datastrukturer, makron och variabler som är externa, det vill säga de variabler som deklarerar 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 lysdioderna. Headerfilen main.h inkluderar några andra headerfiler, till exempel stm32f3_discovery.h som definierar namn för pinnar och portar till discoverykortet.

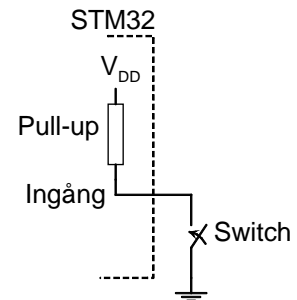
Digitala in- och utgångar GPIO⁷

En digital ingång kan vara

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

Exempel på inkoppling av yttre switch

När switchen är öppen hålls ingången hög av det interna pullup-resistorn. När switchen är sluten hålls ingången låg av att switchen sluts mot jord.

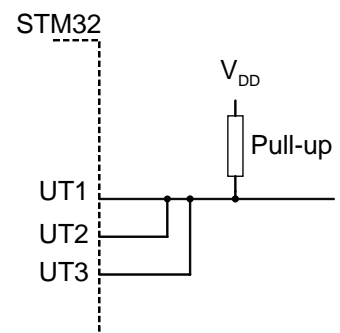


En digital utgång kan vara

- 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. I2C⁸ 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.

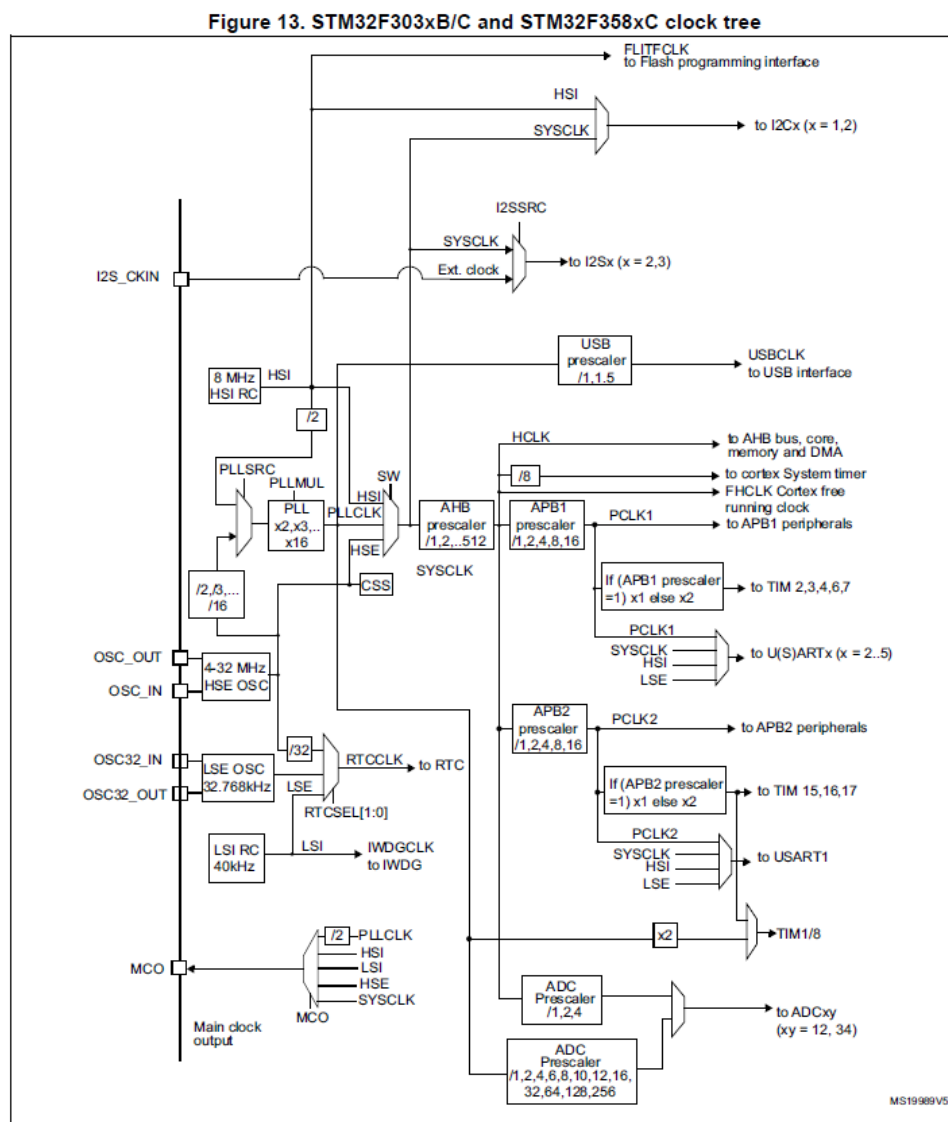
Gå igenom kapitel 11 i referensmanualen som beskriver GPIO. Granska speciellt 11.4 som beskriver registren för GPIO.

⁷ GPIO = General Purpose Input Output

⁸ I2C = I²C 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 mikrokontrollern. 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 mikrokontrollern. Se figur 13 avsnitt 9.2 i Referensmanualen RM0316.



STM32 kan använda olika primära klockkällor. Vanligen används en yttre kristall för att ge oscillationsfrekvensen i en HSE⁹-oscillator. Man kan till exempel koppla en 8 MHz kristall tillsammans med några kondensatorer mellan OSC_OUT och OSC_IN, se figur 15 i databladet. Kristalloscillatorer kan göras väldigt exakta. Som alternativ kan en inbyggd 8 MHz HSI¹⁰ RC¹¹-oscillator användas som primär oscillator. Då får vi en enklare koppling men inte lika exakt klockfrekvens.

⁹ HSE OSC= High Speed External oscillator

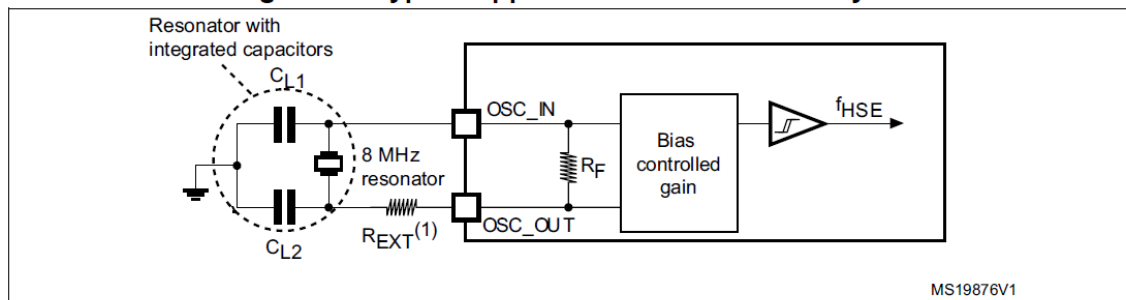
¹⁰ HSI = High Speed Internal

¹¹ RC betyder att en resistor och kondensator i en oscillatorkrets bestämmer klockfrekvensen

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 72 MHz.

Om du studerar STM32F3 Discovery-kortet ser du att det saknas kristall monterad i position X2 på kortet. 8 MHz klocksignal kommer från den mikrokontroller STM32F103C8T6 som finns i ST-LINK debuginterfacet på kortet. Den klockan baseras på den 8 Mhz kristall som finns i ST-LINK debugkretsen.

Figure 15. Typical application with an 8 MHz crystal



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

Vid programstart har det mesta av klockkretsarna initierats av SystemInit i filen system_stm32f30x.c. SystemInit anropas från startup innan main startas.

Programmodulen stm32f3xx_hal_rcc¹² 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.

Frågor om mikrokontroller

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

- 1) Läs i databladet för STM32F303xC kapitel 2 (den kontroller vi använder i Discoverykortet är STM32F303VCT6. Studera speciellt Figure 1 så får du en översikt över alla periferienheter som finns förutom digitala in- och utgångar.
- 2) Läs igenom UM1766 Getting started with STM32CubeF3... Vi kommer att använda HAL-biblioteket när vi programmerar Discoverykortet. 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 UM1786 Description of STM32F3xx HAL drivers, kapitel 2. Denna manual är vår uppslagsbok när det gäller API¹³ för HAL¹⁴. 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?

¹² RCC = Reset and Clock Control

¹³ API Application Programming Interface

¹⁴ HAL Hardware Application Layer

- 4) Använd datablad för mikrokontroller och manual för Discoverykortet för att ta reda på hur mikrokontrollern klockas. Vilka alternativ finns det att klocka mikrokontrollern?
- 5) Nu skall du studera GPIO i mikrokontrollern, dvs. när en pinne används som ingång eller utgång. Du ska söka information i datablad för STM32F303xC när det gäller elektriska specifikationer och i referensmanualen RM0316 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 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 får matningsspänningen till mikrokontrollern ligga vid normal drift?

Figure 42. Input floating/pull up/pull down configurations

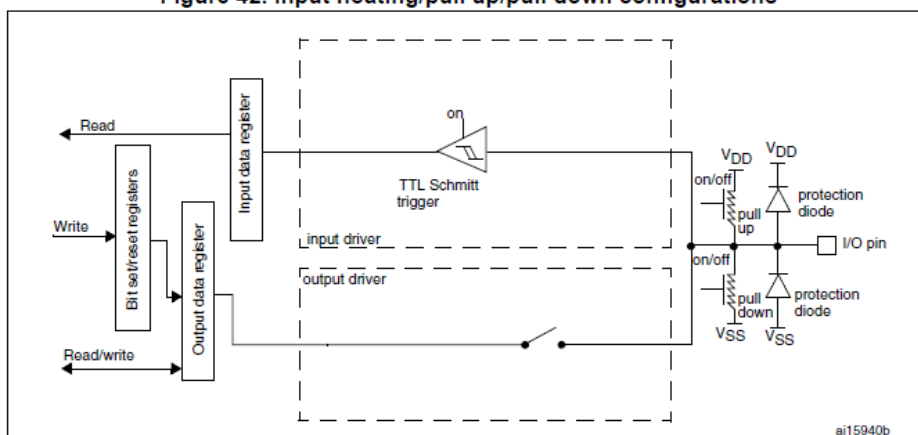
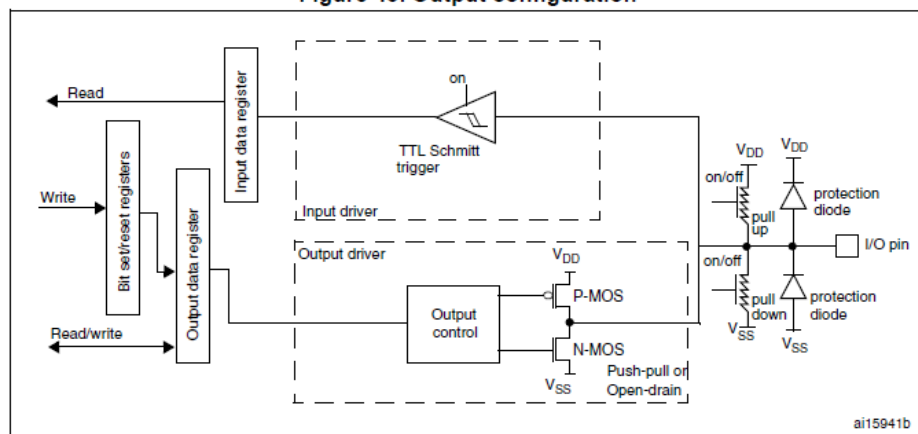


Figure 43. Output configuration



Programmeringsuppgift Pingpong

Vi skall nu gå igenom hur vi skapar ett program för STM32F3 Discovery-kortet. 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

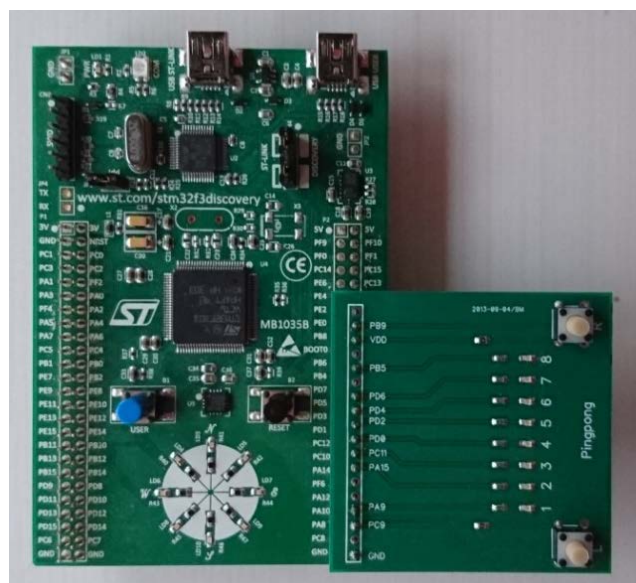
Pingpong-kortet

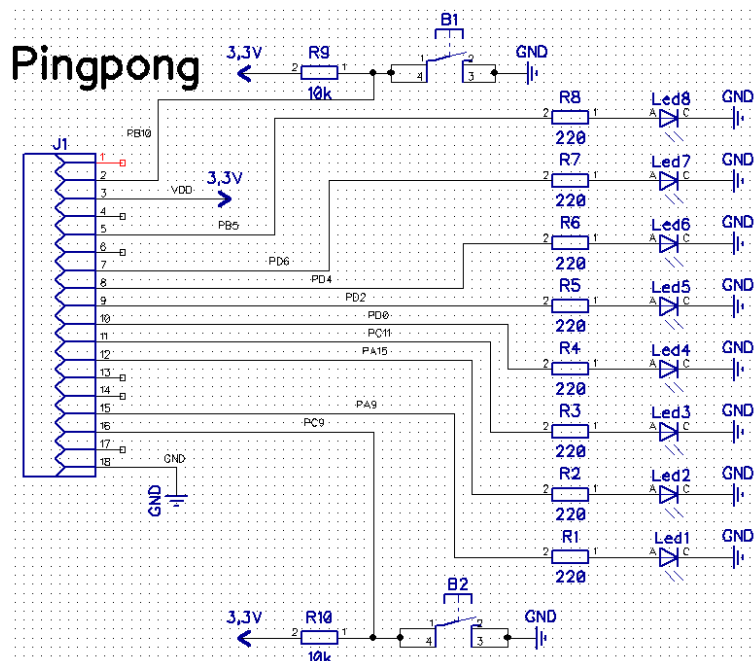
Pingpong-kortet använder följande pinnar på STM32F3

Höger knapp R	PB9
Vänster knapp L	PC9
Lysdiod 1	PA9
Lysdiod 2	PA15
Lysdiod 3	PC11
Lysdiod 4	PD0
Lysdiod 5	PD2
Lysdiod 6	PD4
Lysdiod 7	PD6
Lysdiod 8	PB5

Pingpong-kortet ansluts till STM32F3

Discovery i den yttre raden på motsvarande pinnar som finns angivna på kortet.

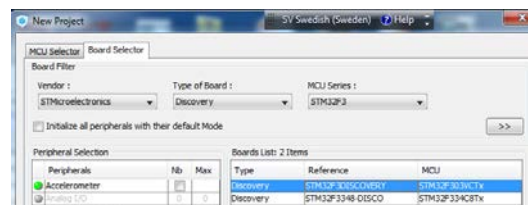




Schema för Pingpong-kortet

Skapa initieringskod med STM32CubeMX

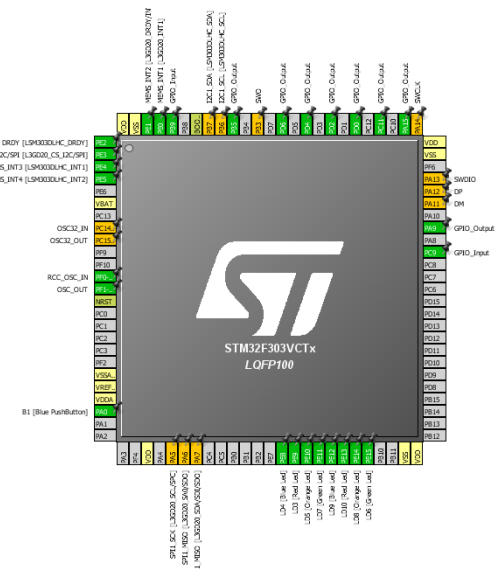
Vi ska först generera initieringskod med STM32CubeMX. Om du har följt anvisningarna i häftet har du redan installerat programmet så det är bara att starta det.



Starta ett nytt projekt för STM32F3 Discoverykortet. Under fliken Pinout visas alla pinnar för kretsen och hur de används.

Konfigurera de pinnar som ansluts till knappar som GPIO_input och de pinnar som ansluts till lysdioderna som GPIO_output. Konfigurera även PF0 som RCC_OSC_IN eftersom vi vill ha en extern klocka inkopplad till mikrokontrollern.

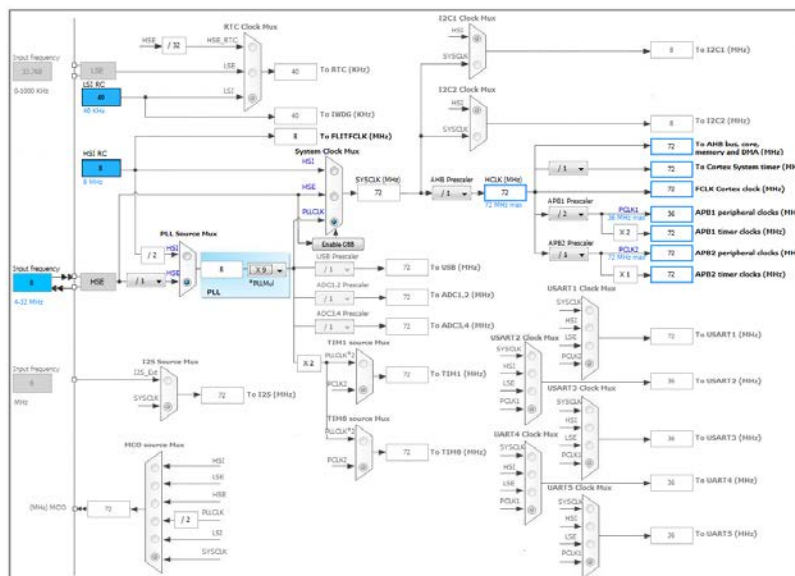
Skapa en mapp på hårddisken där du vill spara dina programmeringsprojekt. Spara projektet där. Namnge gärna filmapp och projekt till Pingpong.



Växla till fliken Configuration och klicka på GPIO. Sätt pinnarna för knapparna till ingångar utan pull up/pull down, dvs. floating input. Sätt pinnarna för lysdioderna till push/pull output. Namnge pinnarna enligt figuren nedan.

GPIO Single Mapped Signals RCC								
Search Signals								
Search (Ctrl+F)								
<input type="checkbox"/> Show only Modified Pins								
Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pull-down	Maximum output speed	Fast Mode	User Label	Modified
PA0	n/a	n/a	Input mode	No pull up pull down	n/a	n/a	B1 Blue ...	✓
PA9	n/a	Low	Output Push Pull	No pull up pull down	Medium	n/a	LED1	✓
PA15	n/a	Low	Output Push Pull	No pull up pull down	Medium	n/a	LED2	✓
PB5	n/a	Low	Output Push Pull	No pull up pull down	Medium	n/a	LED8	✓
PB9	n/a	n/a	Input mode	No pull up pull down	n/a	n/a	R_button	✓
PC9	n/a	n/a	Input mode	No pull up pull down	n/a	n/a	L_button	✓
PC11	n/a	Low	Output Push Pull	No pull up pull down	Medium	n/a	LED3	✓
PD0	n/a	Low	Output Push Pull	No pull up pull down	Medium	n/a	LED4	✓
PD2	n/a	Low	Output Push Pull	No pull up pull down	Medium	n/a	LED5	✓
PD4	n/a	Low	Output Push Pull	No pull up pull down	Medium	n/a	LED6	✓
PD6	n/a	Low	Output Push Pull	No pull up pull down	Medium	n/a	LED7	✓
PE0	n/a	n/a	External Event ...	No pull up pull down	n/a	n/a	MEMS_IN...	✓
PE1	n/a	n/a	External Event ...	No pull up pull down	n/a	n/a	MEMS_IN...	✓

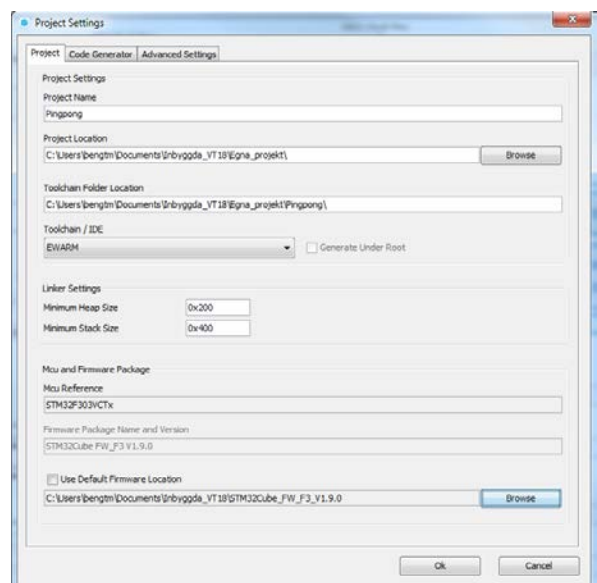
Växla flik till Clock configuration. Då visas klockträdet, som visar hur klocksignaler distribueras i mikrokontrollern. Det finns en intern RC-oscillator som ger 8 MHz, men du får en mer exakt klocksignal om den styrs av en extern kristaloscillator. Låt hela mikrokontrollern styras av HSE (high-speed external clock). Det finns 8 MHz klocksignal från ST_LINK-delen tillgänglig. Den är kopplad till PFO_OSC_IN (RCC_OSC_IN). Mikrokontrollern kan klockas med max 72 MHz. Det finns kretsar som kan multiplicera 8 MHz klocka till en högre frekvens. Aktivera en multiplikation med 9 gånger i PLL¹⁵. Sätt upp klockor enligt figuren nedan.



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.

Project > Settings

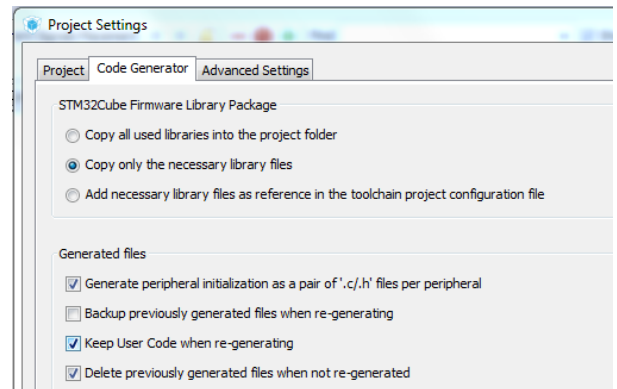
Se till att det är rätt sökväg och att koden för ditt projekt skapas i den mapp du önskar. Toolchain/IDE¹⁶ ska vara EWARM om du skapar kod för IAR Embedded Workbench for ARM. Under fliken Code generator kan du



¹⁵ PLL Phase Locked Loop

¹⁶ IDE Integrated Development Environment, integrerad utvecklingsmiljö

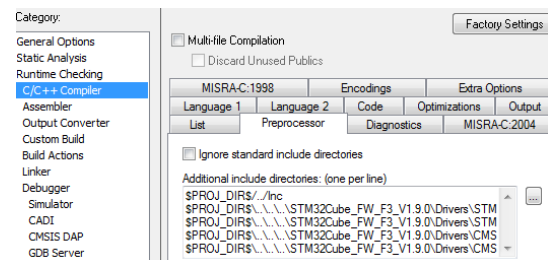
välja att generera separata .c och .h filer för varje IP¹⁷. På så sätt blir inte main-filen så stor utan det genereras särskilda filer för initieringskoden. Jag valde också att ändra sökväg till HAL-biblioteket (Firmware Location) eftersom jag redan har hämtat hem det biblioteket. Det är heller inte nödvändigt att ladda ned biblioteket för varje nytt projekt som skapas.



Generera sedan kod med Project > Generate Code

Öppna ditt programmeringsprojekt i IAR Embedded Workbench

Om du ofta flyttar din kod mellan olika datorer kan det vara lämpligt att ha en relativ sökväg till biblioteken istället för den absoluta som skapas när koden genereras. Ändra under Options. Detta förutsätter naturligtvis att den relativa sökvägen är densamma.



Ditt projekt innehåller nu filer enligt figuren till höger.

Du kan prova att kompilera och länka

Project > Make

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 */

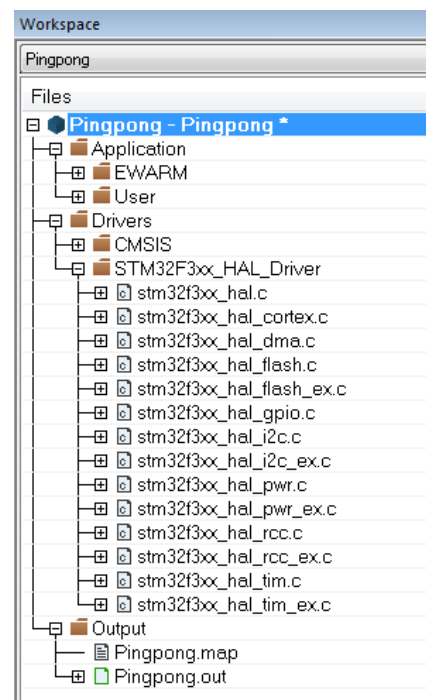
    /* MCU Configuration-----*/

    /* 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 */
```



¹⁷ IP Intellectual Property, i det här fallet betyder det nog ungefär varje periferienhet

```

SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_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), konfigurerings av klockningen och initiering av I/O-pinnarna.

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. Se Project settings för STM32CubeMX i figuren ovan. Det kan dock vara klokt att ha en backup av koden innan du testat att det fungerar.

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

```

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

/*Configure GPIO pins : PAPin PAPin */
GPIO_InitStruct.Pin = LED1_Pin|LED2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pin : PtPin */
GPIO_InitStruct.Pin = LED3_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(LED3_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : PDPin PDPin PDPin PDPin */
GPIO_InitStruct.Pin = LED4_Pin|LED5_Pin|LED6_Pin|LED7_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

/*Configure GPIO pin : PtPin */
GPIO_InitStruct.Pin = LED8_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;

```



```
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(LED8_GPIO_Port, &GPIO_InitStruct);

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

Låt oss undersöka vad som menas med raderna

```
/*Configure GPIO pins : PDPin PDPin PDPin PDPin */
GPIO_InitStruct.Pin = LED4_Pin|LED5_Pin|LED6_Pin|LED7_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(GPIOD, &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 GPIOD och adressen till GPIO_InitStructure som parameter.

Låt oss kontrollera adressen till GPIOD:

Sätt markören över GPIOD, högerklicka och välj "Go to Definition of"

Följ definitioner bakåt för att spåra vilken värde GPIOD har. Du hittar följande rader under din spårning:

```
#define GPIOD ((GPIO_TypeDef *) GPIOD_BASE)
#define GPIOD_BASE (AHB2PERIPH_BASE + 0x00000C00U)
#define AHB2PERIPH_BASE (PERIPH_BASE + 0x08000000U)
#define PERIPH_BASE ((uint32_t)0x40000000U)
```

Adressen till GPIOD är således $0x40000000 + 0x08000000 + 0x00000C00 = 0x48000C00$

Kontrollera detta värde mot minneskartan i Table 2 i reference manual!

AHB2	0x4800 1400 - 0x4800 17FF	1 K	GPIOF	Section 11.4.12 on page 243
	0x4800 1000 - 0x4800 13FF	1 K	GPIOE	
	0x4800 0C00 - 0x4800 0FFF	1 K	GPIOD	
	0x4800 0800 - 0x4800 0BFF	1 K	GPIOC	
	0x4800 0400 - 0x4800 07FF	1 K	GPIOB	
	0x4800 0000 - 0x4800 03FF	1 K	GPIOA	

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 GPIOD.

Sätt en brytpunkt

```
/*Configure GPIO pins : PDPin PDPin PDPin PDPin */
GPIO_InitStruct.Pin = LED4_Pin|LED5_Pin|LED6_Pin|LED7_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_MEDIUM;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

Visa innehåll i register under exekvering, View > Register, välj GPIOD

Kör programmet fram till brytpunkten och stega sedan över anropet till HAL_GPIO_Init

Register	
GPIOD	<fin>
MODER	= 0x00000000
OTYPER	= 0x00000000
OSPEEDR	= 0x00000000
PUPDR	= 0x00000000
IDR	= 0x00000000
ODR	= 0x00000000
BSRR	= 0x00000000
LCKR	= 0x00000000
AFRL	= 0x00000000
AFRH	= 0x00000000
BRR	= 0x00000000

- ✚ Vilka bitar i registren påverkas av initieringen? Observera att det inte är säkert att bitarna ändras. Det säkerställs i initieringen att bitarna har ett visst värde. Jämför med beskrivningen av registren i referensmanualen.

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.

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.

Förhoppningsvis har du nu en hyfsad grundläggande förståelse för hur API¹⁸ för GPIO används. Du kommer att ha anledning att studera datablad, referensmanualen och manualen för HAL-biblioteket när du utvecklar dina egna program.

¹⁸ API = Application Programming Interface

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. Vi skall försöka introducera 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

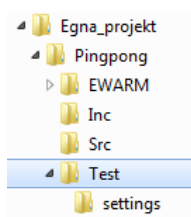
- utveckla testprogram för att testa funktionerna som ska utvecklas
- utveckla funktioner i ett funktionsbibliotek som vi använder som beståndsdelar i Pingpong-programmet
- kunna gå tillbaka och köra testprogrammen om vi ändrar i funktionerna

Skapa ett nytt projekt i Embedded workbench i samma workspace som Pingpong. Spara den i en ny mapp på hårddisken som du kallar Test.

Project > Create new project...

Genom att skapa flera projekt inom samma workspace kan du ha gemensamma filer med kod men olika main-filer. Du kan då lätt växla mellan testprogram, där du testar dina funktioner, och huvudprogrammet. Det är ett enkelt sätt att säkerställa att det är de funktioner du testat som används i ditt färdiga program eftersom samma filer används.

Katalog på hårddisken

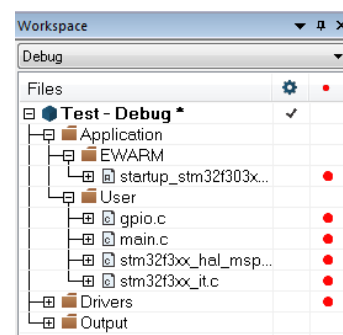


Struktur i workspace Embedded workbench



Du kan nu enkelt växla mellan de två projekten Pingpong och Test genom att trycka på rätt flik i nederkanten. Aktivera projektet Test och kopiera filstrukturen i det projektet från Pingpong.

Project > Import File List... från projektet Pingpong.ewp i mappen EWARM



Nu ska vi byta ut mainfilen i projektet. Ta bort filen main.c från projektet Test. Kopiera main.c från katalogen /Src till katalogen /Test på hårddisken. Lägg in kopian main.c från katalogen /Test till gruppen User.

Markera User, högerklicka och välj Add > Add Files...

File > Save workspace så att filer för projektet genereras på hårddisken

Du har nu samma struktur som ovan, men det är en annan main-fil som ingår. Om du vill kontrollera vilken fil som ingår i projektet kan du markera filen, högerklicka och välja Open Containing Folder...

Det som fattas nu är att sätta alla Options för projektet så att den kompilerar för rätt mikrokontroller och länkar samman alla filer korrekt. Det är lite pyssel att fixa detta genom att gå igenom Options för Pingpong och se till att Options för Test blir desamma. Jag har inte hittat något sätt att kunna kopiera Options.

Gå igenom hur man ställer in Options på sidan 3-5 i detta häfte. Ställ in Options så att de blir lika som för Pingpong.

Testa att kompilera och länka. Det kan vara något mer som måste fixas, men det kan du nog lösa själv.

Project > Make eller Project > Rebuild all

Om filstrukturen ligger lika relativ sökvägarna till biblioteken så kommer alla filer att hittas.

Nu kan vi skriva ett separat Testprogram i main.c för Test.

Skapa en funktion som tänder lysdioder

Vi använder nu det nya projektet Test till att skriva testprogram.

Vi börjar med att skriva och testa en loop som loopar en heltalsvariabel från 1 till 8.

```
int Lednr;
for (Lednr=1; Lednr<= 8; Lednr++) {
    printf("Lednr är %d \n",Lednr);
}
```

satsen printf har ingen mening på STM32-kortet eftersom vi inte har någon display att skriva ut på. Utskriften kommer att hamna i fönstret Terminal I/O när du kör programmet. %d betyder formatering för heltal, \n ger radframmatning

Aktivera terminalfönstret: View > Terminal I/O

Om du vill köra C-koden i programmet utan att köra den på Discovery-kortet kan du ändra i

Options > Debugger > Simulator istället för ST-Link.

Du kan nu köra programmet utan att STM32-kortet är anslutet!
Kom ihåg att ändra tillbaka när du skall köra på kortet.

Testkör programmet. I terminalfönstret skall du nu få utskriften enligt figuren till höger.

Nu är det dags att skapa en funktion.
Skriv in deklarationen av funktionen main i källkoden¹⁹.
Lämpligast efter kommentaren Private functions prototypes

```
Terminal I/O
Output:
Lednr är 1
Lednr är 2
Lednr är 3
Lednr är 4
Lednr är 5
Lednr är 6
Lednr är 7
Lednr är 8
```

```
/* Private function prototypes -----*/
void Led_on(uint8_t Lednr);
```

Funktionen måste finnas deklarerad innan den används i main. Själva funktionen kan du definiera efter mainfunktionen.

```
/* USER CODE BEGIN 4 */
void Led_on(uint8_t Lednr)
{
    printf("Lednr är %d \n",Lednr);
    return;
}
/* USER CODE END 4 */
```

och ändra koden i main till

```
int Led;
for (Led=1; Led<= 8; Led++) {
    Led_on(Led);
    HAL_Delay(500); // Delay 500 ms
}
```

Testkör igen. Du skall få samma utskrift i terminalfönstret.
Vi vet nu att vi har ett fungerande anrop av en funktion Led_on.

printf-raden kan vi nu ta bort eller kommentera bort raden så är det enkelt att koppla in den igen om vi behöver testutskrifter.

```
// printf("Lednr är %d \n",Lednr);
```

Den passerar däremot inte vårt test för en fungerande funktion eftersom inga lysdioder tänds. Vi har ju inte skrivit den koden ännu så nu det kan vara dags att göra det.

Vi måste

- tända rätt lysdiod och släcka övriga
loopa igenom alla åtta lysdioder och tänd om den ska tändas, annars släck

Ändra funktionen Led_on till följande kod i main.c

```
/* Function Led_on
Input: Turn on led Lednr
       Lednr can be 1-8, all other values turns all leds off
Output: No
*/
```

¹⁹ Källkod är textfilen med programkod, i det här fallet programkod i språket C (.c)

```

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

```

Nu kan du skriva testprogrammet i main så här kort:

```

/* USER CODE BEGIN 2 */

printf("Test av funktion Led_on");
int Led;
for (Led=1; Led<= 9; Led++) {
    Led_on(Led);
    HAL_Delay(500); // Delay 500 ms
}

/* USER CODE END 2 */

```

- Ett tips: För att få snygga indragningar kan du markera all text (CTRL-A) och göra Edit / Auto indent.

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. 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 `/**`

För ytterligare information om vilka kommentarer som genererar information till Doxygen hänvisas du till dokumentation i Doxygen.

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

```
/**
*****
@brief   Pingpong test program
@file    main.c
@author   Bengt Molin
@version 1.2
@date    4-Januari-2018
@brief    Testing functions and structures for program Pingpong
*****
*/
```

Ändra också kommentarshuvudet 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 har fått programmet att fungera skall det tända en lysdiod åt gången. Testet slutar med att alla lysdioder är släckta efter `Led_on(9)`.

Nu har du förhoppningsvis en fungerande funktion som kan tända en lysdiod och se till så att alla andra är släckta och vi har ett program för att kunna testa funktionen. Ska vi kunna använda funktionen i ett annat program så måste vi göra den tillgänglig. Då kan det vara bra att samla de funktioner vi skapar i en separat fil, dvs. vi kan börja skapa ett funktionsbibliotek.

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 testat den funk-

tionen. Programspråket C tillåter att man skriver väldigt kryptisk och svårläst kod. Det är dock bättre att skriva lättförståelig kod även om det blir fler rader.

Skapa ett funktionsbibliotek

När programmet enligt ovan fungerar skall du skapa en ny fil och lägga alla funktioner i den filen. Lägg filen i en annan filkatalog än katalogen för projektet Test, lämpligen c-filen i /Src och headerfilen i /Inc så att filen kan bli tillgänglig för andra projekt. Trolig arbetsgång:

- ✓ Flytta 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
- ✓ Lägga till filen `pingpong_functions.c` i projektet

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.

Testa att det fungerar!

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

Vi kan göra detsamma med initieringen av klockan och lägga den i en egen fil `clockinit.c` och `clockinit.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 koden med STM32CubeMX.

```
/**
*****
@brief   Pingpong test program
@file    main.c
@author   Bengt Molin
@version 1.2
@date    4-Januari-2018
@brief    Testing functions and structures for program Pingpong
*****
*/

/* Includes -----*/
#include "main.h"
#include "stm32f3xx_hal.h"
#include "gpio.h"
#include "clockinit.h"
#include "pingpong_functions.h"

int main(void)
{
    /* Reset of all peripherals, Initializes the Flash interface and the SysTick*/
```



```

HAL_Init();

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

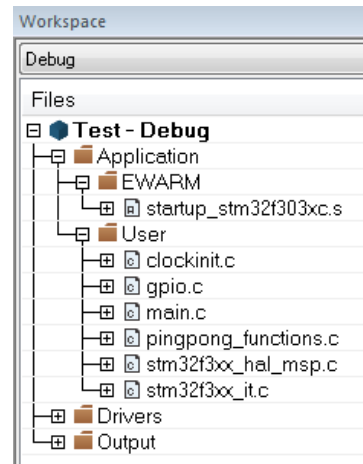
/* Initialize all configured peripherals */
MX_GPIO_Init();

printf("Test av funktion Led_on");

int Led;
for (Led=1; Led<= 9; Led++) {
    Led_on(Led);
    HAL_Delay(500); // Delay 500 ms
}

/* Infinite loop */
while (1)
{
}
}

```

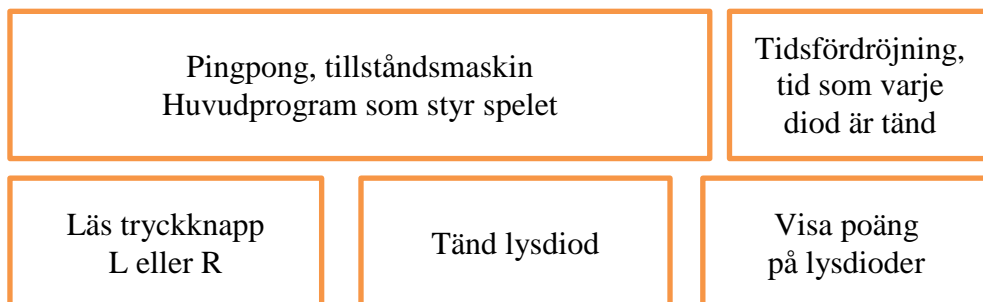


Filer enligt figur till höger ingår nu i testprogrammet.

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.

- ✓ Skissera 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 tidsfördröjning 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 finnas 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.

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 PC9 respektive PB9 till noll volt (jord). När knappen inte trycks ned hålls spänningen på ingångarna PC9 respektive PB9 hög av ett pullup-motstånd till matningsspänningen V_{DD} . Se figur på sidan 13. Ingångarna kan således initieras till floating input.

PC9 och PB9 är redan initierade till flytande ingång (floating input) i funktionen MX_GPIO_Init som skapades av programmet STM32CubeMX.

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

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

Läs User Manual UM1786, Description of STM32F3xx HAL drivers, avsnittet om GPIO-funktioner i kapitel 19 (HAL GPIO Generic Driver). Där hittar du den information du behöver om API²⁰ för GPIO²¹.

Nu börjar det bli mycket kod i testprogrammet. Vi kan antingen separera testningen i flera projekt eller göra någon case-sats så att vi endast kör det test som önskas. Vi kan också göra det enkelt för oss genom att kommentera bort kod som vi inte vill köra. Det

²⁰ API Application Programming Interface

²¹ GPIO General Purpose Input Output

gör du genom att sätta kommentarparenteser `/*` och `*/` kring koden du vill kommentera bort.

Med funktionen `HAL_GPIO_ReadInputPin` kan vi läsa värdet på en ingångspinne.

Testprogram

Mata in följande kod i testprogrammet. Testa att köra koden.

```
j=0;

while (1)
{
    while ( HAL_GPIO_ReadPin(L_button_GPIO_Port, L_button_Pin) != 0 );
                                   // Wait until L is pushed

    Led_on(j++); // Turn on led j

    while ( HAL_GPIO_ReadPin(L_button_GPIO_Port, L_button_Pin) == 0 );
                                   // Wait until L is released

    if (j==9) j=0; // Start from beginning again
}
```

Ibland när du trycker på L-knappen hoppar den framåt fler än en diod. Varför gör den det? Testa att lägga en liten fördröjning (100 ms) efter att lysdioden tänts och innan man väntar på att knappen har släppts! Testa även med en lite fördröjning efter att knappen släpps, men innan man testat om den tryckts ned.

L_hit och R_hit

- Skriv nu funktioner `L_hit` och `R_hit` som testat 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.

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)

/**
@brief R_hit, check if R button is pressed
```

```

@param void

@return bool, true if R button pushed, false otherwise
*/

bool R_hit(void)

```

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

Fortsätt att använda testprogrammet och lägg in följande testprogram:

```

j=0;

while (1)
{
    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 (j>8) j=9; // Start again from right
    }

    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
    }
}

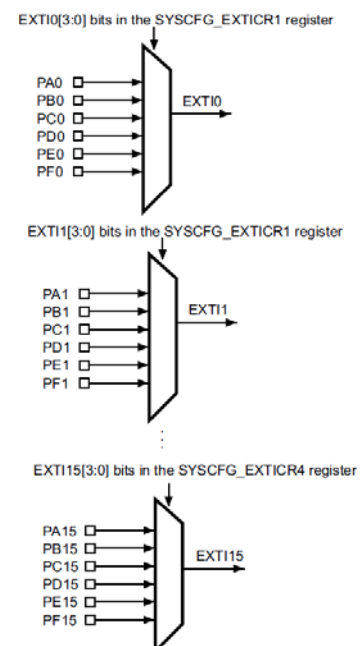
} // end while

```

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.

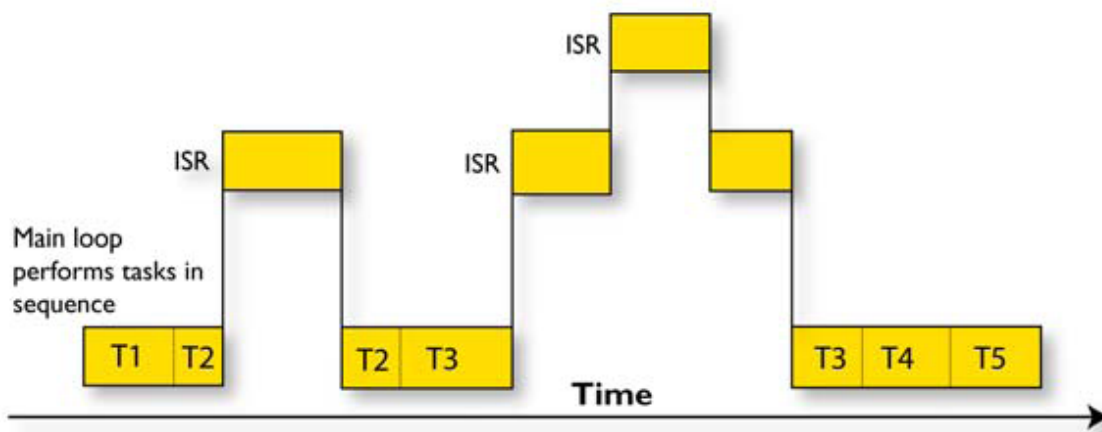
Vi kan göra på ett annat sätt; vi kan låta tryckknapparna generera ett avbrott och sätta en flagga att knappen har tryckts ned.

Externt avbrott via GPIO blir dock problem i detta fall. Båda tryckknapparna, PB9 och PC9, tillhör samma grupp. Det finns inte en avbrottsvektor 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 (styrs 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 avbrottsrutinen läsa status på pinnarna för att se vilken knapp som tryckts ned. Hänsyn måste tas till att tryckknappen är en mekanisk kontakt som kan studsas mellan hög och låg några gånger innan det blir stabilt. Vi skulle också kunna lösa det genom att



konfigurera om avbrottsvillkoret när "bollen" byter riktning så att det bara är den knapp som förväntas trycka som kan generera avbrott.

Att använda avbrott tillåter att programexekveringen kan, av en yttre händelse, styras till att exekvera en avbrottsrutin (ISR, Interrupt Service Routine). När avbrottsrutinen är klar återgår programmet till normal exekvering från den punkt där den blev avbruten.



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 RTOS²²)
- ✓ State machine models.

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

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

Det finns faktiskt en avbrottsrutin aktiv i programmet som vi jobbar med. Om du tittar i filen stm32f3xx_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, 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.

²² RTOS Real Time Operating System, realtidsoperativsystem

I vårt program används system timer SysTick för att generera avbrott en gång per millisekund och räkna 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_stm32f303xc.s finns följande rad som lägger in avbrottsvektorn (adress till avbrottsrutinen) i minnet.

```
DCD      SysTick_Handler          ; SysTick Handler
```

Filen är ett assemblerprogram som körs innan c-programmet startar om *Run to main* är markerat för Debugger under Options. DCD är ett assemblerdirektiv som allokerar plats i minnet och på platsen läggs adressen som anges.

I filen stm32f3xx_it.c finns alla avbrottsrutiner. De är deklarerade men från början tomma. 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 Go to definition of... Genom att spåra bakåt några gånger kommer du till uwTick och hur den deklaras.

```
static __IO uint32_t uwTick;
```

Några förklaringar:

static betyder att variabeln har ett reserverat minnesutrymme där värdet på variabeln ligger kvar till nästa gång uwTick används inuti en funktion.

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. Till exempel så här:

```
int8_t left,right,j;

/* Loop checking that all leds can be turned on*/
printf("Test 1: Testar att dioder tänds och släcks med Led_on\n");

for (j=0; j<= 9; j++)
{
    printf("Led nr %d \n",j);
    Led_on(j);
    HAL_Delay(500);
}
printf("Test 1 klart\n\n");
HAL_Delay(1000); // 1000 ms
```

```

/* Checking that points can be shown correct */
printf("Test 2 Testar att poäng kan visas med Show_points\n");
right=0;
for (left=1; left<= 4; left++)
{
    printf("L poäng %d",left); printf("   R poäng %d \n",right);
    Show_points (left,right);
    HAL_Delay(500); // 500 ms
}

left=0;
for (right=1; right<= 4; right++)
{
    printf("L poäng %d",left); printf("   R poäng %d \n",right);
    Show_points (left,right);
    HAL_Delay(500); // 500 ms
}

for (j=0; j<= 4; j++)
{
    printf("L poäng %d",j); printf("   R poäng %d \n",j);
    Show_points (j,j);
    HAL_Delay(500); // 500 ms
}
printf("Test 2 klart\n\n");
HAL_Delay(1000); // 1000 ms

/* Checking buttons */
printf("Test 3 Testar avläsning av tryckknappar med L_hit och R_hit\n");
printf("Tryck L för att flytta tänd lysdiod till höger\n");
printf("Tryck R för att flytta tänd lysdiod till vänster\n");
printf("Tryck L för att börja\n");
printf("Tryck bollen förbi 8 för att avsluta\n");

j=0;

while (j<9)
{
    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
    }
}
printf("Test 3 klart\n\n\n");
HAL_Delay(1000); // 1000 ms

```

Vi kan skicka utskrifter till terminalfönstret för att visa vilka tester som körs.

Vi har nu tester för att testa att följande funktioner fungerar:

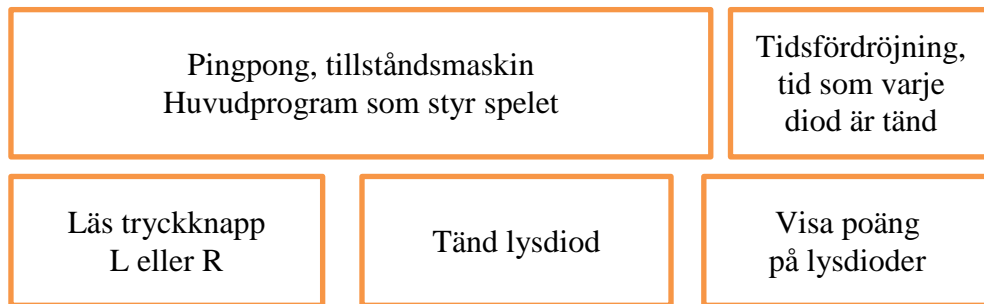
```

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);

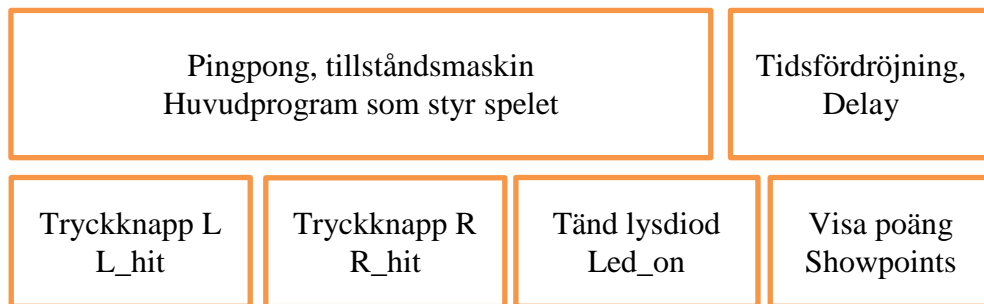
```

Alla funktioner finns i filen Pingpong_functions.c. Samma fil skall användas när vi gör klart Pingpong-programmet, men från ett annat projekt. Behåll testprogrammet. **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.

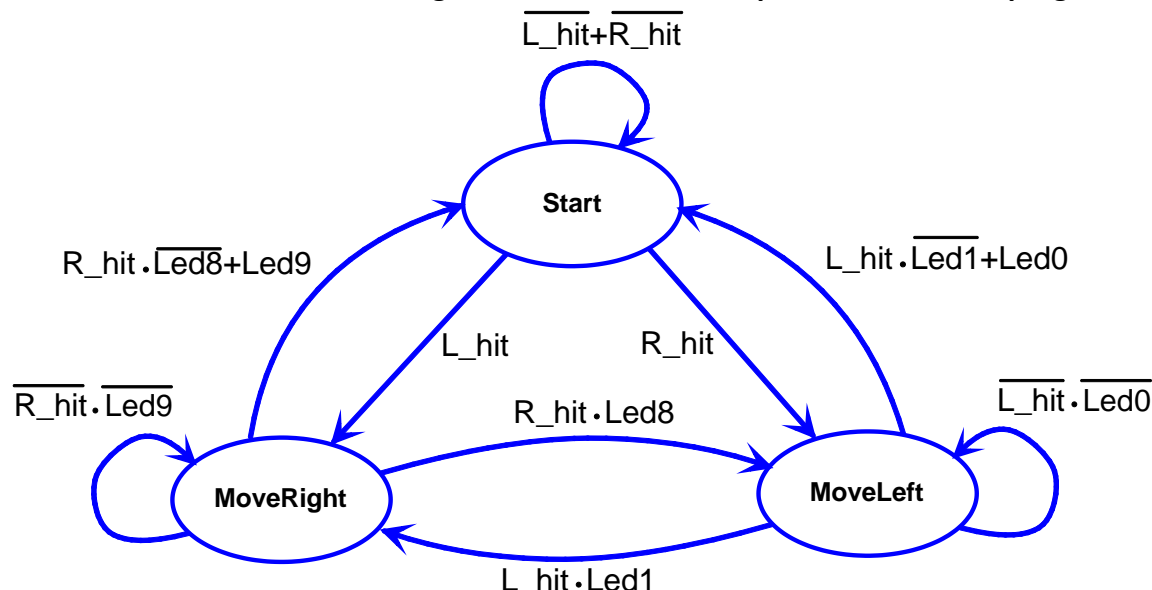


Nu är alla delar klara utom tillståndsmaskinen som skall styra hela pingpong-spelet.

- ✚ Nu kan det vara dags att återgå till vårt ursprungliga projekt Pingpong, där vi ska göra klart pingpong-spelet
- ✚ 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. Tillståndsdigram har du kommit i kontakt med i digitalteknikkursen. Utgå från ett starttillstånd där programmet står och väntar på att någon skall starta pingpong-spelet. Vi väntar på en serve! Fundera på vad som skall hända och i vilken ordning det skall göras innan du vänder till nästa sida.

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



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

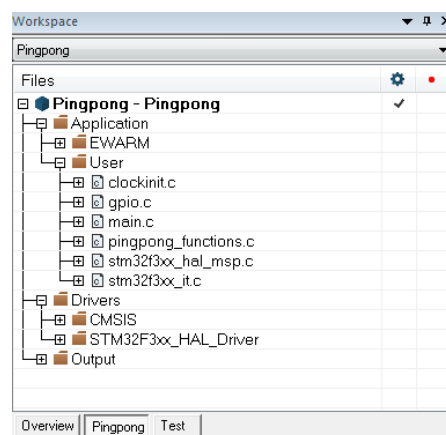
Start	Väntar på att någon skall starta spelet genom att serva
MoveRight	"Bollen", dvs lysande diod rör sig mot höger
MoveLeft	Lysande diod rör sig mot vänster

Växla till projektet Pingpong.

Du ska nu få ett Pingpong-program, som inte är helt färdigt, men visar hur programmet kan konstrueras.

Skriv in de delar av koden nedan som fattas i ditt program. Lägg in det på lämpliga ställen mellan olika `/* USER CODE BEGIN */` och `/* USER CODE END */` parenteser.

Testkör koden.



Koden är inte komplett, den visar att vi kan lämna start-tillståndet och hamna i MoveRight eller MoveLeft. Vad skall hända när vi ligger i MoveRight?

Vi måste vänta så lång tid 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 tillståndsdigrammet enligt föregående sida går programmet tillbaka till start vid missad boll.

```
/**
*****
@brief   Pingpong program
@file    main.c
@author   Bengt Molin
@version 1.2
@date    4-Januari-2018
@todo    Finish the program
*****
*/

#include "main.h"

#include "clockinit.h"
#include "pingpong_functions.h"

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

states State, NextState;

int main(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

    /* MCU Configuration-----*/

    /* 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();

    /* 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;
    }

}

}

```

Deklarationen av states kan lika gärna ligga i main.h som inkluderas.

```

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

states State, NextState;

```

- ✚ Gå igenom koden och undersök om den uppfyller specifikation 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 specifikation 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 du har ett fungerande program ska du lägga hela tillståndsmaskinen i en funktion Pingpong i en egen fil pingpong.c

Huvudprogrammet i main.c behöver då endast anropa Pingpong, vilket ger ett kort huvudprogram. 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.

Några tips så här i slutet av denna skrift.

Om du vill testköra små delar av C-kod så kan det vara bra att ha en workspace där du kan testa koden. Kör i simuleringsläge så behöver du inte ladda ned koden i en mikrokontroller efter varje kompilering. Det går lite fortare.

Lär dig använda debuggern. Sätt brytpunkter och undersök variablers värde. Lär dig stega, steg in i och stega över funktioner. Om du använder Live watch kan du se variabelvärden under programkörning. För att Live watch ska fungera måste variabeln vara deklarerad som *static*.