En kort STM32 HAL-referens.

När vi i CubeMX aktiverar olika peripherals kommer CubeMX att generera kod åt oss och se till att ta med rätt HAL-drivrutiner i vårt projekt. Dessa hamnar under Drivers/STM32F4xx_HAL_Driver/Src och interfacet i …/Inc. Dessa filer innehåller mycket dokumentation om hur drivrutinerna används!

Kodgenereringen skapar en del #defines i main.h utifrån de namn vi gett pinnar mer mera. Även variabler för periferienheterna skapas och kod som initierar dem efter våra önskemål skapas.

T.ex. variabeln htim1 för timer 1. Detta är en struct skapad av CubeMX och dess innehåll är anpassat för HAL-funktionerna och används som handtag/handle för att utföra operationer på den peripheral som handtaget representerar. Tänk på att detta är en variabel som representerar en peripheral, det är inte en pekare till en peripheral. Ofta innehåller structen en pekare till den faktiska peripheralens minnesutrymme/registerutrymme och kallas då ofta Instance.

HAL använder denna struct-medlem (Instance) internt i sina funktioner för att komma åt rätt register för en specifik peripheral

HAL-funktionerna vill dessutom oftast ha en pekare till ett handtag. Med timer 1 som exempel: Vi vill ha adressen till variabeln/structen htim1, d.v.s. en pekare till htim1. Det får vi om vi skriver: &htim1.

Nedanstående funktioner är bara ett litet litet utdrag av allt som finns!

Hjälpfunktioner som använder SysTick:

```
void HAL_Delay( uint32_t Delay )
```

Väntar angiven tid i millisekunder. Kallas ”busy wait”.
Exemplet kommer att ge en delay på 500 millisekunder, alltså 0.5s, alltså en halv sekund.

```
HAL_Delay(500);
```

- `uint32_t HAL_GetTick( void )`
  Ger tick-värde i millisekunder. Börjar om på 0 efter 2^{32} millisekunder.
  Värdet now i exemplet kan lätt jämföras med andra värden för att få tidsskillnader.

```
uint32_t now = HAL_GetTick();
```

GPIO:

- ```
  void HAL_GPIO_WritePin( GPIO_TypeDef  *GPIOx,
                          uint16_t       GPIO_Pin,
                          GPIO_PinState  PinState )
  ```
  Sätter pinnen GPIO_Pin på port GPIOx hög eller låg med *GPIO_PIN_SET* resp. *GPIO_PIN_RESET* som PinState.
  Första exemplet kommer sätta pinnen PC3 låg. Har PC3 namngivits till 'LED_R' så gör andra raden samma sak:

  ```
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_RESET);
  ```

- ```
  GPIO_PinState HAL_GPIO_ReadPin( GPIO_TypeDef *GPIOx,
                                  uint16_t      GPIO_Pin )
  ```
  Kontrollerar om en specifik pinne är hög eller låg.
  Ex. om man vill köra kod om namngiven pinne "BUT_1" är låg:

  ```
  if (!HAL_GPIO_ReadPin(BUT_1_GPIO_Port, BUT_1_Pin) { //kod }
  ```

- ```
  void HAL_GPIO_TogglePin( GPIO_TypeDef *GPIOx,
                           uint16_t      GPIO_Pin )
  ```
  Togglar en specifik pinne. Om den är låg blir den hög och vice versa.
  Ex. om man vill toggla pinnen PC3:

  ```
  HAL_GPIO_TogglePin(GPIOC, PC3);
  ```

GPIO-Callback:

- ```
  void HAL_GPIO_EXTI_Callback( uint16_t GPIO_Pin )
  ```
  Skapar du denna funktion kommer HAL att anropa den åt dig när ett externt interrupt har skett. Anropet sker i flera steg från en EXTI-ISR. Parametern anger vilken GPIO-pinne interruptet skedde på. För att anropet ska ske måste EXTI-interrupten vara aktiverad i NVIC.

UART-utskrifter:

- ```
  HAL_StatusTypeDef
  HAL_UART_Transmit( UART_HandleTypeDef *huart,
                     const uint8_t      *pData,
                     uint16_t            Size,
                     uint32_t            Timeout )
  ```
  Skriver ut en textsträng över UART, vilket för F411RE är USB-anslutningen.
  För att få en ordentlig radbrytning så måste radslutet vara "\r\n" och inte bara "\n".
  Variabeln Timeout är antalet tick som funktionen tillåts köra innan den ger upp.
  Se slutet av detta dokument för icke-triviala exempel.

  ```
  HAL_UART_Transmit(&huart2, "Hello World!\r\n", 13, 100);
  ```
  ```

Timer:

- `HAL_StatusTypeDef`
  `HAL_TIM_Base_Start_IT( TIM_HandleTypeDef *htim )`
  Startar timer htim och gör att ev. valda interrupt kommer ske. Vill man inte aktivera interrupt utan bara starta timern använd funktionen utan _IT på slutet.
  Ex. Ser till att timer 1 börjar räkna. Du har i CubeMX slagit på interrupt för t.ex. update-eventet på timer 1 och vill att det ska köras:

```
HAL_TIM_Base_Start_IT(&htim1);
```

Timer - PWM

- `HAL_StatusTypeDef`
  `HAL_TIM_PWM_Start( TIM_HandleTypeDef *htim, uint32_t Channel )`
  Startar PWM på angiven timer och kanal (Capture/Compare).
  (Startar även timern om den inte redan är startad. (Dock inte med interrupt på.))
  Ex. Startar PWM-utgång för kanal 1 och kanal 2 på timer 1. Du behöver ha konfigurerat upp det rätt i CubeMX innan.

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
```

- `__HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__, __COMPARE__)`
  Macro för att sätta värdet för Capture/Compare på kanal __CHANNEL__ och timer __HANDLE__ till __COMPARE__.
  Ex. Du vill sätta CCR till 32768 på kanal 2, timer 1.
  Andra raden gör samma som första, men "rakt på" (skriver direkt till registret).

```
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 32768);
htim1.Instance->CCR2 = 32768;
```

Timer – Callback

- `void HAL_TIM_PeriodElapsedCallback( TIM_HandleTypeDef *htim )`
  Skapar du denna funktion kommer HAL att anropa den åt dig när ett interrupt sker p.g.a. ett en timer har "slagit runt" (nått upp till Counter Period (AutoReload Register). Anropet sker i flera steg från en timer-ISR. Parametern htim anger vilken timer interruptet gäller. För att anropet ska ske måste global- eller update-timer-interrupt vara aktiverade i NVIC.

- `void HAL_TIM_IC_CaptureCallback( TIM_HandleTypeDef *htim )`
  Skapar du denna funktion kommer HAL att anropa den åt dig när ett interrupt sker p.g.a. att en capture har skett. En extern händelse har triggat "fångandet" av timerns värde till ett Capture/Compare-register. Anropet sker i flera steg från en timer-ISR. Parametern htim anger vilken timer interruptet gäller. För att anropet ska ske måste global- eller capture compare-timer-interrupt vara aktiverade i NVIC.

ADC

- `__HAL_ADC_GET_FLAG(__HANDLE__, __FLAG__)`
  Macro för att kontrollera om den angivna ADC flaggan __FLAG__ är satt för angiven ADC __HANDLE__.
  Ex. Du vill kontrollera om en ADC-omvandling är klar på ADC 1:
  Andra raden gör ungefär samma sak som macrot, men ”rakt på”.

```
if (__HAL_ADC_GET_FLAG(&hadc1, ADC_FLAG_EOC)) { //kod }
if (hadc1.Instance->SR & ADC_FLAG_EOC) { //kod }
```

  ADC_FLAG_EOC står för ADC End of Regular Conversion flag.

- `uint32_t HAL_ADC_GetValue( ADC_HandleTypeDef *hadc )`
  Returnerar resultatregistrets innehåll från angiven ADC:
  Ex. Returnerar resultatregistrets innehåll från ADC1.
  Andra raden gör samma sak, men ”rakt på”.

```
uint16_t AdcValue = HAL_ADC_GetValue(&hadc1);
uint16_t AdcValue = hadc1.Instance->DR;
```

  Resultatet från AD-omvandlingen ligger nu i variabeln AdcValue.

- `HAL_StatusTypeDef HAL_ADC_Start_IT( ADC_HandelTypeDef *hadc )`
  Startar ADC-omvandling och ger interrupt när omvandlingen är klar. Är kontinuerlig adc-omvandling vald kommer omvandling och interrupt ske kontinuerligt.
  Ex. Starta omvandling på ADC1:

```
HAL_ADC_Start_IT(&hadc1);
```

ADC – Callbacks

- `void HAL_ADC_ConvCpltCallback( ADC_HandleTypeDef *hadc )`
  Skapar du denna funktion kommer HAL att anropa den åt dig när ett interrupt sker p.g.a. ett en ADC är färdig. Anropet sker i flera steg från en adc-ISR. Parametern hadc anger vilken adc interruptet gäller. För att anropet ska ske måste adc global interrupt vara aktiverat i NVIC.

UART-utskrifter med sprintf():

- ```
  HAL_StatusTypeDef
  HAL_UART_Transmit( UART_HandleTypeDef *huart,
                     const uint8_t      *pData,
                     uint16_t            Size,
                     uint32_t            Timeout)

  int sprintf(char       *str,
              const char *format_str, ... )
  ```

  För att komma åt sprintf() krävs denna import:

  ```
  #include <stdio.h>
  ```

  Det första argumentet till sprintf() är den buffert som resultatet ska sparas i. Den resulterande strängens längd är funktionens returvärde. En buffert för resultatet kan skapas så här:

  ```
  char     str[81] = { '\0' }; // init all as NUL
  uint16_t str_len;
  ```

  Funktionen printf() är väldigt avancerad. Sök efter "printf reference" på internet för att lära dig mer. Vi använder här sprintf(), en variant av printf(), eftersom vi inte har tillgång till något som liknar cout. Några exempel:

  ```
  int nbr = 3;
  str_len = sprintf(str, "The number three: %d\r\n", nbr);
  HAL_UART_Transmit(&huart2, (uint8_t*) str, str_len, 1000);
  // "The number three: 3\r\n"


  uint8_t   big = 255;
  char* fmt_str = "d: %d, x: %x, hex notation: 0x%X\r\n";
  str_len       = sprintf(str, fmt_str, big, big, big);
  HAL_UART_Transmit(&huart2, (uint8_t*) str, str_len, 1000);
  // "d: 255, x: ff, hex notation: 0xFF\r\n"


  uint8_t  small = 0x1f;
  uint32_t beef  = 0xDEADBEEF;
  str_len        = sprintf(str, "0x%08X - 0x%08x", small, beef);
  HAL_UART_Transmit(&huart2, (uint8_t*) str, str_len, HAL_MAX_DELAY);
  // "0x0000001F - 0xdeadbeef"


  int ok  = init(); // failed and returned 0
  str_len = sprintf(str, "init() was...%s\r\n", ok ? "good" : "bad");
  HAL_UART_Transmit(&huart2, (uint8_t*) str, str_len, 100);
  // "init() was...bad\r\n"
  ```
  ```