ST MCSDK

Unofficial presentation of ST Motor Control implementation

Software Architecture overview

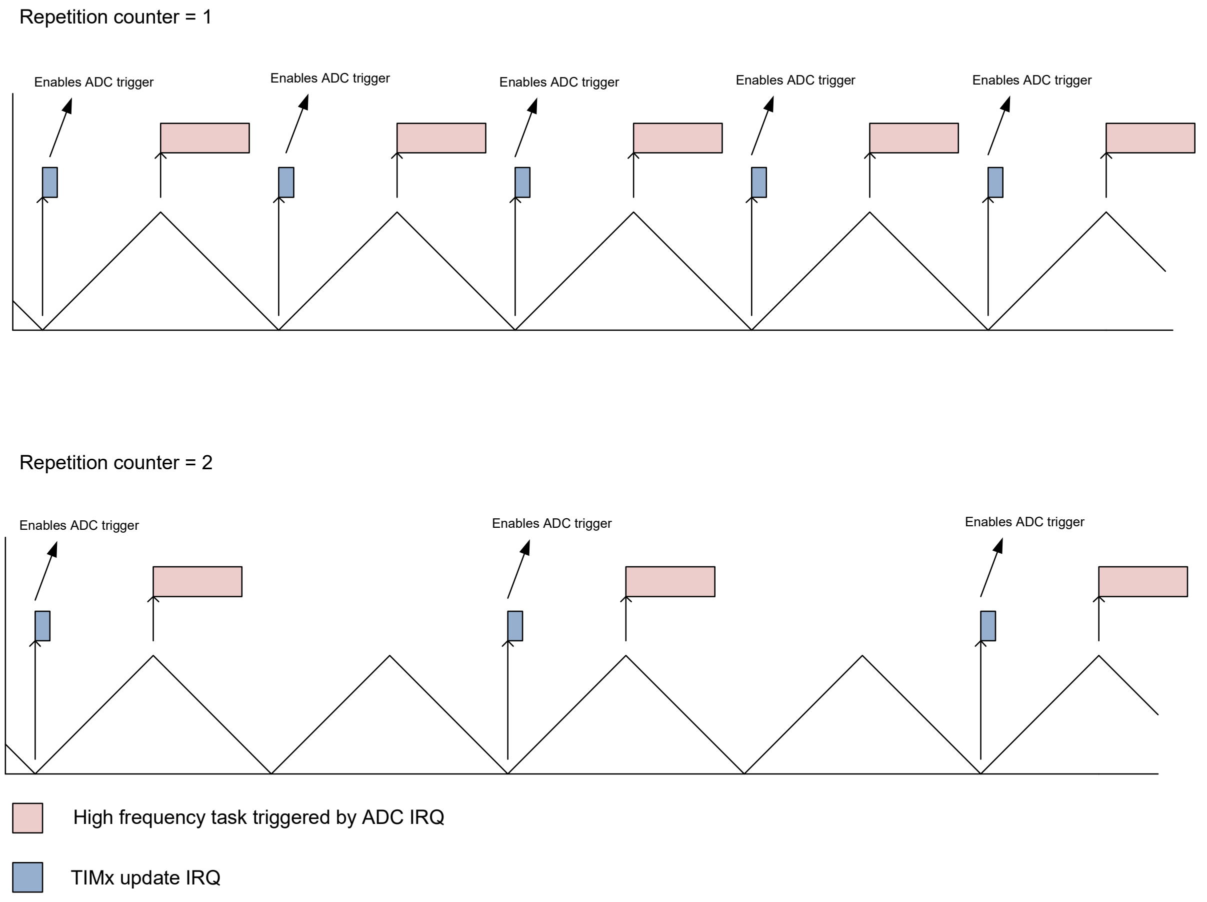
- The ST-MCSDK works only under interrupt. By default without OS.
- The control motor is based on three different interrupts :
  - PWM Timer Update interrupt – Highest priority : 0
    - Fires at regulation rate speed (PWM frequency if FOC is computed at each PWM cycle)
    - Reconfigure the ADC sequence, and enable the ADC if required (series dependent)
  - High frequency interrupt triggered by ADC end of conversion – running at regulation rate . – Medium priority : 2
    - The FOC algorithm is executed under this IT. The target is to compute the new PWM cycles values for each phases before the PWM timer reach 0. Once it is 0, the new computed values are taken into account for the next cycle.
  - Medium frequency interrupt, based on Systick – not synchronous versus the High frequency interrupt. Low priority : 4
    - This medium frequency task is in charge of executing the speed regulation loop.
    - The main motor control state machine is updated on this interrupt too.

Software Architecture overview

- The User interface is based on two different interrupts :
  - UART - RX
    - Used to received command from a host (PC or whatever)
    - Used for telemetry, data are send after a data_read command reception
  - EXTI
    - Used to read the start – stop button.

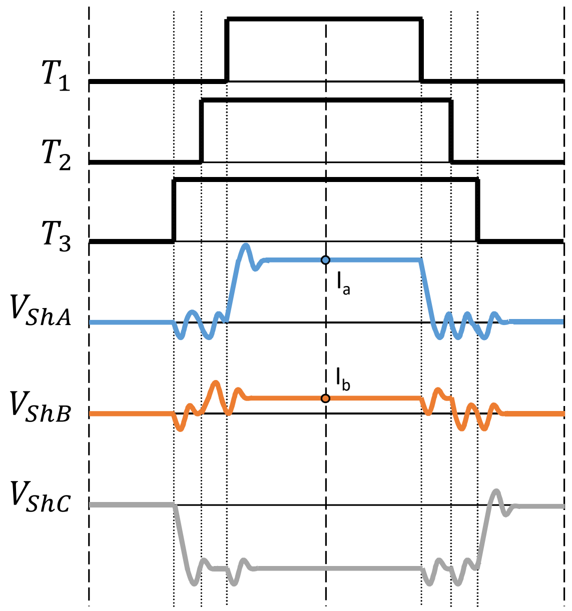
PWM configuration

- PWM is center aligned
- CCR4 is configured to fires the ADC
- (Basically CCR4 = ARR -1)
- The challenge is to reconfigure the PWM timer before it reaches 0.
- The new CCR[1,2,3] values are taken into account by the hardware thanks to shadow mechanism when the counter reaches 0.



Sampling theory

- According to the PWM duty cycles and the setled noise parameters the three shunt algorithm:
  - chooses which two of the three current shall be measured
  - chooses the point inside the PWM period to sample the currents
  - computes, if required, the third
- In case of low modulation index the preferred choose is to sample always Ia and Ib in the middle of PWM period.

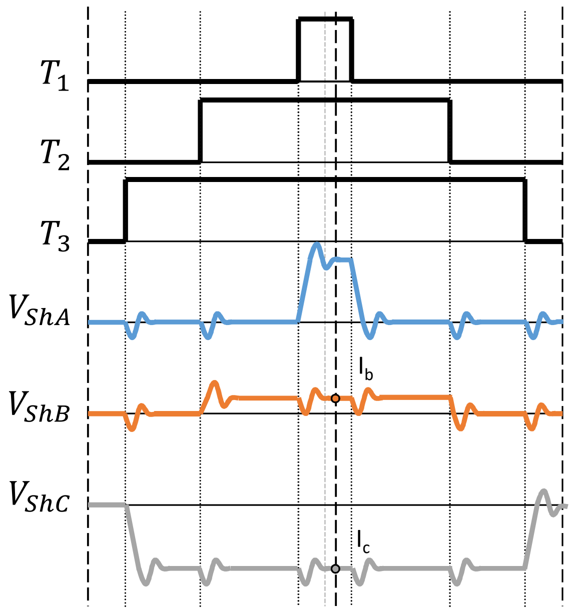


Sampling theory

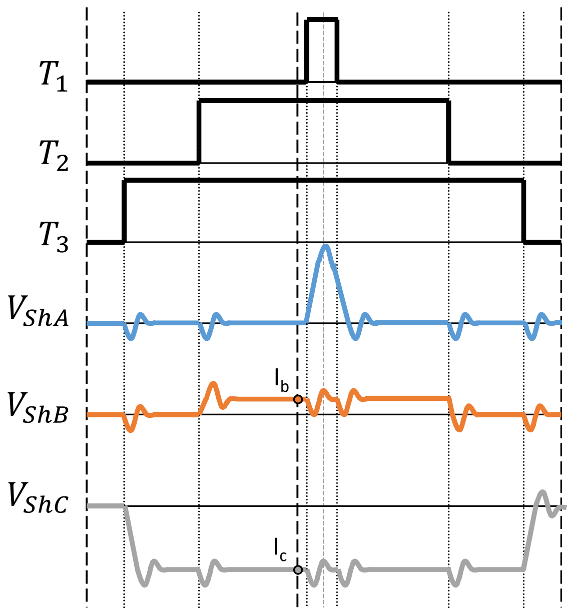
- In case of high modulation index the algorithm choose the two phases with the big space to sample the current and set a sampling point that is after the middle of the PWM period.
- If is not possible to sample after the middle of the PWM period the algorithm sets a sampling point that is before the last switching of the first half period.

.

Samplig after middle of the PWM period

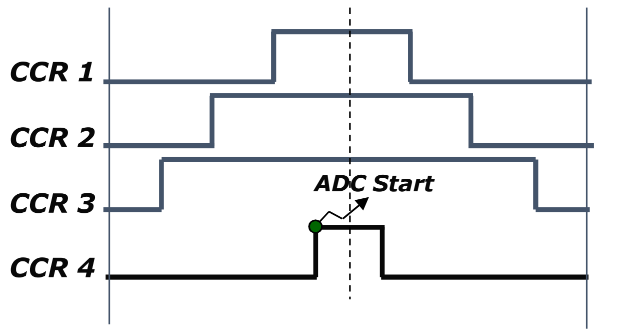


Samplig before the last switching of first half period

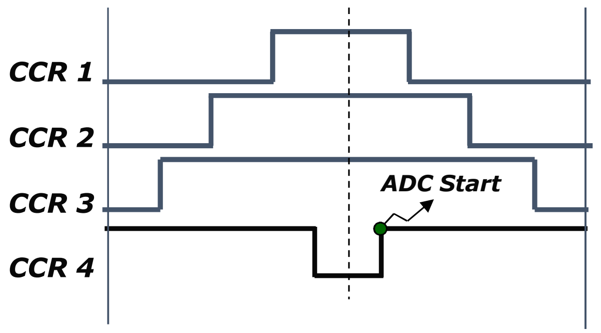


Sampling implementation

- ADC peripheral is triggered by CCR4 rising edge.
- To sample after the counter overflow it is required to invert the polarity of CCR4 signal.



Samplig before the last switching of first half period



Samplig after middle of the PWM period

- Note that the CCR1,2,3 represented here are the LOW sides. (Sampling is done when low sides are closed).
- We used center aligned mode 1 for CCR1,2,3 and center aligned mode 2 for CCR4 (excepting for sampling after middle of PWM period)

Sampling implementation - code

```
/* Check if sampling AB in the middle of PWM is possible */

  if ( ( uint16_t )( pHandle->Half_PWMPeriod - lowDuty ) > pHandle->pParams_str->Tafter ) {
 /* When it is possible to sample in the middle of the PWM period */
 pHandle->_Super.Sector = SECTOR_4;
/* set sampling point trigger in the middle of PWM period */
hCntSmp = ( uint32_t )( pHandle->Half_PWMPeriod ) - 1u;
 }  else  {
  /* Crossing Point Searching */
  hDeltaDuty = ( uint16_t )( lowDuty - midDuty );
  /* Definition of crossing point */
 if ( hDeltaDuty > ( uint16_t )( pHandle->Half_PWMPeriod - lowDuty ) * 2u )
 { /* hTbefore = 2*Ts + Tc, where Ts = Sampling time of ADC, Tc = Conversion Time of ADC */
  hCntSmp = lowDuty - pHandle->pParams_str->Tbefore;
 }  else {
  /* hTafter = DT + max(Trise, Tnoise) */
  hCntSmp = lowDuty + pHandle->pParams_str->Tafter;
  if ( hCntSmp >= pHandle->Half_PWMPeriod ) {
   /* It must be changed the trigger direction from positive to negative to sample after middle of PWM*/ /* Set CC4
as PWM mode 1 */
    pHandle->PWM_Mode = LL_TIM_OCMODE_PWM1; hCntSmp = ( 2u * pHandle->Half_PWMPeriod ) -
hCntSmp - 1u;
  }
 }
}
```

This code snippet comes from r3_2_f1xx_pwm_curr_fdbk.c , function R3_2_SetADCSampPointSectX available in the MCSDK : https://www.st.com/content/st_com/en/products/embedded-software/mcu-mpu-embedded-software/stm32-embedded-software/stm32cube-expansion-packages/x-cube-mcsdk.html

ADC configuration

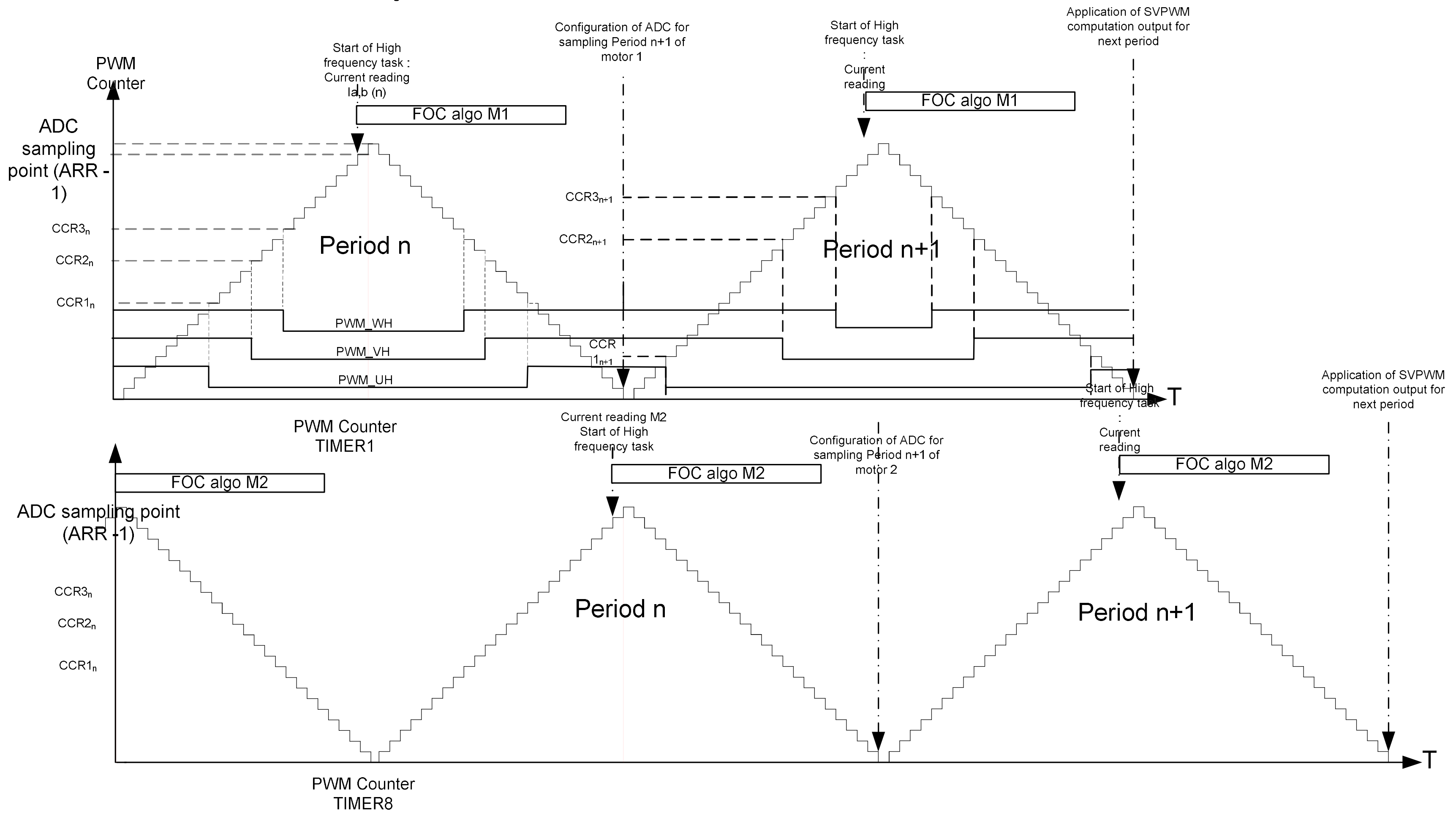
- ADC are always triggered by hardware for current sensing.
- In case of 2 ADCs, sampling the two phases simultaneously increases the accuracy, and reduce the overall sampling time.
- As the current input are the entry point of the FOC algo, it is key to guarantee that the scheduling of the ADC is not disturb by other ongoing sampling.
- This is the reason why ST prefers to use Injected channels when available for current sensing and let non-critical ADC conversion in regular channels.
- In case of 2 ADCs, the End of Conversion of one ADC is enough to start the FOC algorithm. We consider here that sampling time is the same for both acquisitions. So data are available in the same clock cycle for both ADCs.
- Injected data conversion are stored into JDRx registers (up to 4) and no DMA is required to copy the data into the SRAM.

ADC configuration - Code

```
/** * @brief It contains the TIMx Update event interrupt *
          @param pHandle: handler of the current instance of the PWM component *
          @retval none */
 __weak void * R3_2_TIMx_UP_IRQHandler( PWMC_R3_2_Handle_t * pHandle)
 {
 TIM_TypeDef* TIMx = pHandle->pParams_str->TIMx;
ADC_TypeDef * ADCx_1 = pHandle->pParams_str->ADCx_1;
ADC_TypeDef * ADCx_2 = pHandle->pParams_str->ADCx_2;
/* Disabling trigger to avoid unwanted conversion */
 LL_ADC_INJ_StopConversionExtTrig(ADCx_1);
 LL_ADC_INJ_StopConversionExtTrig(ADCx_2);
/* Set next current channel according to sector */
ADCx_1->JSQR = pHandle->pParams_str->ADCConfig1[pHandle->_Super.Sector];
ADCx_2->JSQR = pHandle->pParams_str->ADCConfig2[pHandle->_Super.Sector];
/* Enabling next Trigger */
 LL_TIM_CC_EnableChannel(TIMx, LL_TIM_CHANNEL_CH4);
LL_ADC_INJ_SetTriggerSource(ADCx_1, pHandle->ADC_ExternalTriggerInjected);
 LL_ADC_INJ_SetTriggerSource(ADCx_2, pHandle->ADC_ExternalTriggerInjected);
LL_ADC_INJ_StartConversionExtTrig(ADCx_1, LL_ADC_INJ_TRIG_EXT_RISING);
 LL_ADC_INJ_StartConversionExtTrig(ADCx_2, LL_ADC_INJ_TRIG_EXT_RISING);
 /* Set edge detection trigger according to PWM Mode 1 or 2 */
LL_TIM_OC_SetMode(TIMx, LL_TIM_CHANNEL_CH4, pHandle->PWM_Mode);
return &( pHandle->_Super.Motor ); }
```

This code comes from r3_2_f1xx_pwm_curr_fdbk.c

Dual Drive implementation


PWM Counter
ADC sampling point (ARR - 1)
CCR3n
CCR2n
CCR1n
Start of High frequency task : Current reading Ia,b (n)
FOC algo M1
Period n
PWM_WH
PWM_VH
PWM_UH
Configuration of ADC for sampling Period n+1 of motor 1
CCR3n+1
CCR2n+1
CCR 1n+1
Start of High frequency task : Current reading
FOC algo M1
Period n+1
Application of SVPWM computation output for next period
T
PWM Counter TIMER1
FOC algo M2
ADC sampling point (ARR -1)
CCR3n
CCR2n
CCR1n
Current reading M2 Start of High frequency task
FOC algo M2
Period n
Configuration of ADC for sampling Period n+1 of motor 2
Start of High frequency task : Current reading
FOC algo M2
Period n+1
Application of SVPWM computation output for next period
T
PWM Counter TIMER8

Dual Drive implementation

- As you can see on previous slide, When ADCs are shared between both motors, we read values from one motor while we reconfigure the same ADC for sampling the second motor.
- We have to take care that the conversion is actually finished before reprogramming. This code is not present in Timer Update IRQ handler of stm32F1 series as we do not support dual drive for this series with MCSDK 5.X.

Dual Drive implementation

- This code snippet comes from F4 series : r3_2_f4xx_pwm_curr_fdbk.c

```
/* dual drive check */
ADCInjFlags = ADCx_1->SR & (LL_ADC_FLAG_JSTRT|LL_ADC_FLAG_JEOS);
if ( ADCInjFlags == LL_ADC_FLAG_JSTRT )
{  /* ADC conversion is on going on the second motor */
  do
  { /* wait for end of conversion */
    ADCInjFlags = ADCx_1->SR & (LL_ADC_FLAG_JSTRT|LL_ADC_FLAG_JEOS);
   }
   while ( ADCInjFlags != (LL_ADC_FLAG_JSTRT|LL_ADC_FLAG_JEOS) );

  }  else if ( ADCInjFlags == 0 )
  {  /* ADC conversion on the second motor is not yet started */
     while ( ( TIMx->CNT ) < ( pHandle->pParams_str->Tw ) )
     { /* wait for a maximum delay */ }
     ADCInjFlags = ADCx_1->SR &
(LL_ADC_FLAG_JSTRT|LL_ADC_FLAG_JEOS);
     if ( ADCInjFlags == LL_ADC_FLAG_JSTRT )
     { /* ADC conversion is on going on the second motor */
      do { /* wait for end of conversion */
           ADCInjFlags = ADCx_1->SR &
(LL_ADC_FLAG_JSTRT|LL_ADC_FLAG_JEOS);
          }
          while ( ADCInjFlags != (LL_ADC_FLAG_JSTRT|LL_ADC_FLAG_JEOS) );
      }
  } else
     { /* ADC conversion on the second motor is done */
     }
```

TIMER 1 and 8 shift

- The trick is to set one timer to Half_PWMPeriod -1

(Half because we are center aligned)

- And to start both timers synchronously.
- In order to do so, we configure TIMER 1 and TIMER 8 as slave of a third Timer (Timer 2), and we generate a software update on TIMER2 to start TIMER 1 and 8.
- Look at startTimers function in files pwm_common.c
- r3_2_f4xx_pwm_curr_fdbk.c initializes timer for dual motor.

Conclusion

- how to set the TIM1, TIM8 shifted in Phase (as you suggested)
  - I think it is covered, I suggest you to generate a project with MCSDK and read the IOC with cubeMX it really help to see how do we configure the peripherals.
- how to trigger the ADC1, ADC2 synchronized with TIM1, TIM8
  - With the reference manual and the R3_2_TIMx_UP_IRQHandler you should find.
  - Field JEXTSEL of ADC_CR2 register
- how to reconfigure the ADC1, ADC2 online to trigger in the alternating fashion for TIM1 (Motor 1) -> TIM8 (Motor 2) -> TIM1 (Motor 1) and so on
  - I think it is covered, in the R3_2_TIMx_UP_IRQHandler section
- how to trigger the DMA interrupt routine after the ADC conversion is finished. 1 DMA for both motors or 2 separate DMAs, depends which one you consider more appropriate
  - My advise, is to not use the DMA. Data are present in the ADC JDR register if you use injected channel, that I highly recommend.
- we will place our motor control control algorithm in the DMA routine(s).
  - FOC starting point is the phase Current values. This is the most critical task you have to do. Start the computation as soon as it is possible meaning under ADC EOS interrupt.

- Last word, these slides have been done on my free time and are not an official ST document. I hope they will be helpful for you even if the quality of the presentation is far from ST rules.