Kinetis Motor Suite: Implementing Custom Hardware Lab Guide

Rev. 0.0

SOLUTIONS

SECURE CONNECTIONS FOR A SMARTER WORLD

1 Purpose

This document is provided as a hands-on lab guide for the NXP Technology day in Minneapolis Minnesota. The intent of the lab is to demonstrate how to set up custom hardware, modify the pin set up and add standalone user software to control the motor speed. For the case of this lab, we will use the FRDM-KV31F and FRDM-MC-LV3PH as the target hardware in its default configuration.

The lab will direct the user through a series of steps that show proper code development for you own custom hardware using the Kinetis Motor Suite reference design.

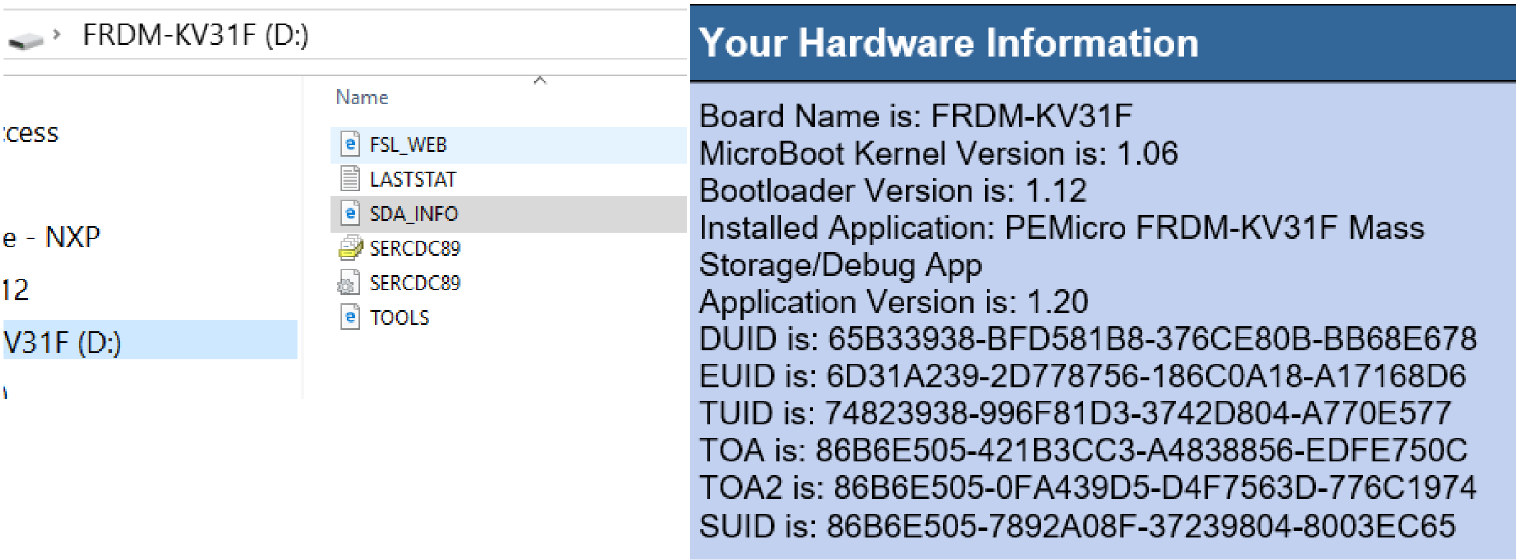
By the end of this lab the user will learn how to use:

- MCUXpresso IDE for code download debug and execution
- MCUXpresso Pins tool to add GPIO and ADC inputs and outputs
- KMS user ADC conversion option
- Convert ADC input 16 bit single ended input to LQ speed input
- How to command USER State and Motor Speed

2 Resources

The following resources are for reference. If you could review the below before completing the lab it is highly recommended.
Verify Debug Firmware – DO NOT mass erase MCU flash. Open the MSD FRDM-KVxx you have and verify it's the latest firmware apps and drivers. Update as needed. www.pemicro.com/opensda

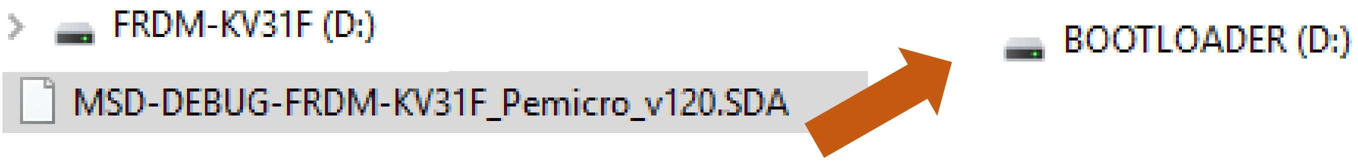
FRDM-KV31F - MSD-DEBUG-TWR-KV31F12_Pemicro_v120.SDA
FRDM-KV11Z - MSD-DEBUG-FRDM-KV11Z_Pemicro_v120.SDA
HVP-KV31F - MSD-DEBUG-HVP-KV31F120M_Pemicro_v121.SDA



Your Hardware Information

Board Name is: FRDM-KV31F
MicroBoot Kernel Version is: 1.06
Bootloader Version is: 1.12
Installed Application: PEMicro FRDM-KV31F Mass Storage/Debug App
Application Version is: 1.20
DUID is: 65B33938-BFD581B8-376CE80B-BB68E678
EUID is: 6D31A239-2D778756-186C0A18-A17168D6
TUID is: 74823938-996F81D3-3742D804-A770E577
TOA is: 86B6E505-421B3CC3-A4838856-EDFE750C
TOA2 is: 86B6E505-0FA439D5-D4F7563D-776C1974
SUID is: 86B6E505-7892A08F-37239804-8003EC65

Press MCU reset button and plug in USB cable



If you need MCUXpresso IDE, SDK or pin or config tools training you can go here.
https://www.nxp.com/docs/en/supporting-information/APF-DES-T2744-MCUXpressor.pdf

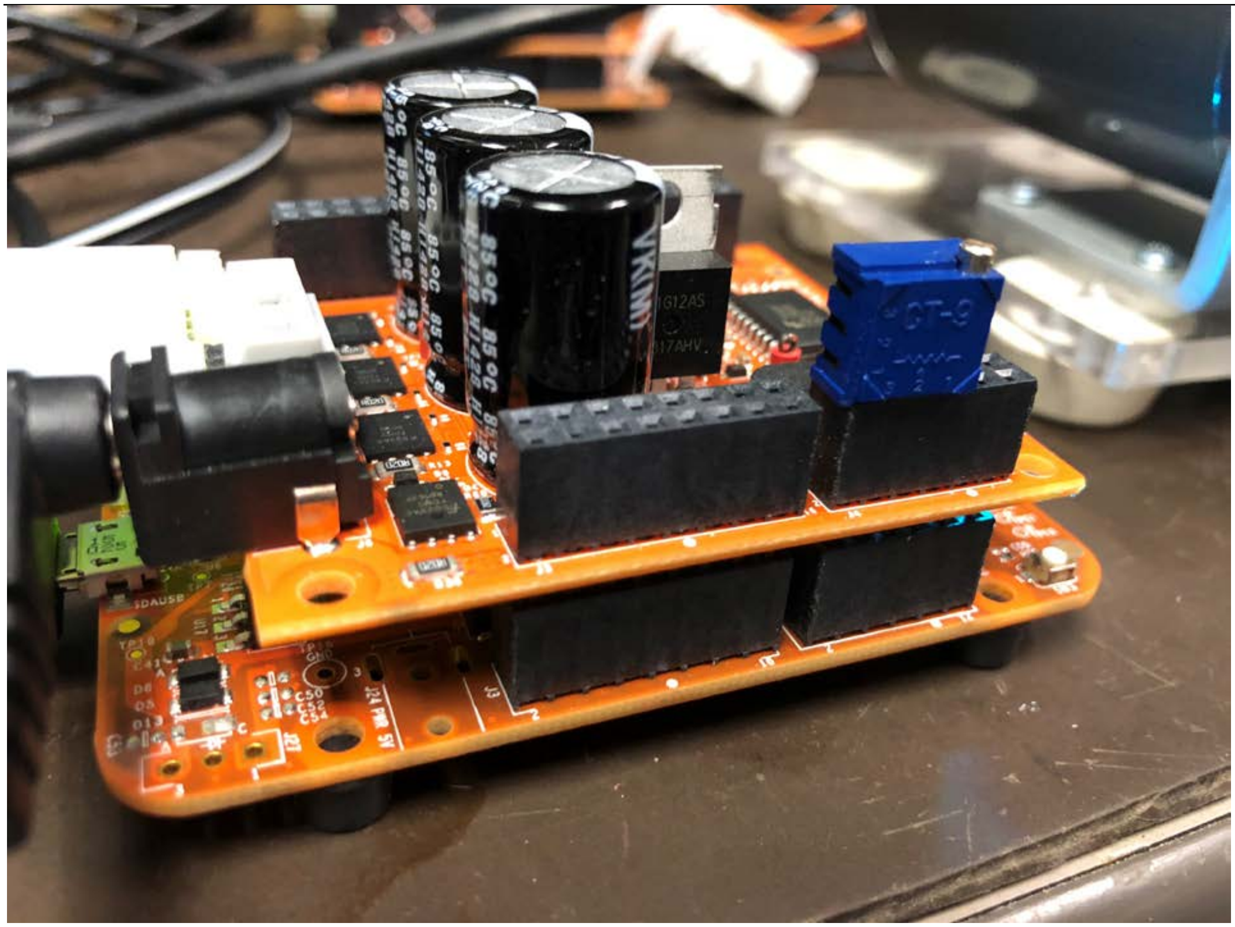
3 Overview and Essential Background

Bench Setup

This LAB uses the FRDM-KV31F120M, the FRDM-MC-LV3PH, a motor and KMS on a PC running Windows.

A potentiometer is used as a speed input. Plugged into pins 2, 4 and 6 of J4

The LAB assumes that the PC is preloaded with MCUXpresso IDE version 10.2 and the related drivers. In this lab, only a USB connection to the debug port is needed.



Background on KMS ADC inputs

KMS employs a dual interrupt service routine (ISR) approach, A fast interrupt for dynamic control of the electrical of the motor and a slow interrupt as an update to the mechanical response of the motor.

During the Fast ISR, the three motor phase currents and sampled (FEEDBACK), the sensorless angle estimation is performed (EST) or the electrical angle will be provided by the sensor, the current controllers are run to update the reference voltage (CURRENT), and the space vector modulator is called to update the three PWM duty cycles (SVPWM). There are additional time-sensitive software components dealing with fault detection (DSM) and braking (BRAKE) that are called during this ISR.

During the Slow ISR, the feedback speed is estimated (EST) or calculated from the encoder (ENC), the reference speeds are updated (TRAJ), the speed (SPEED) or position (POSITION) is controlled, and the field weakening controller is run (FW). There is also some additional software executed to handle the drive (DSM) and user (USER) state machines.

ADC conversions measure the bus voltage and the phase currents at the Fast ISR rate. Since both ADC modules are being used for motor control, any additional ADC channel sampling must be scheduled in-between the motor control samples. This mechanism is built into the reference project. The code to set up these additional samples runs at the same frequency as the motor control. After processing the motor control sample, the current index of the user ADC channels is configured to be sampled. Then the motor control software is executed, after which the results from the user ADC channel is stored in the results buffer (adcxResults, where x is the ADC instance [0 or 1]). The ADC is then reconfigured for motor control sampling.

To configure this code, three things need to be set in the main.c file.

1. The macro definition for NUMBER_USER_ADC_CHANNELS should be set to the total number of ADC channels you want to sample on one of the ADC converters. For example, if you wish to sample two channels on ADC 0, this would be set to 2. If you want to sample one channel on ADC 0 and one channel on ADC 1, this would be set to 1.
2. The arrays adcUserChannelsChannelx (where x is the ADC instance (0 or 1)) identify the channels that you wish to sample. This array represents the round-robin list of channels where one is sampled per ADC converter during each ISR.
3. This additional ADC sampling is enabled or disabled using a static variable called bEnableRoundRobinAdc. This variable needs to be set to true in order for these additional ADC channels to be sampled.

Code Review - USER_States for speed control

- Idle
- Fault
- Self-commissioning (SCM)
- Inertia Estimation (Inertia)
- PWM Duty Control
- Voltage Control
- Current Control (Current)
- Speed Control (Speed)
- Position Control (Position) [Sensored Position]
- Motion Sequence (Plan)
- Braking (Brake)
- Encoder Alignment (Align) [Sensored Velocity or Sensored Position]

Custom Hardware Resources Needed for KMS

The KMS User’s Guide has a list of the resources used by the project and the tools. Open the Kinetis Motor Suite Users Guide and go to chapter 11 page 191.

1. Execution cycles and clock speed
2. Flash □ KMS pre-programmed code in the top 8K of the Flash.
   + (50472 to 58804 using MCUXpresso) or + (43,224 to 48456 using IAR)
3. RAM less than 8K
4. Peripherals□ FAC, ADC0, ADC1, PDB0, FlexTimer0 & 1, UART0, GPIO for Hall Sensor. If the FET pre-driver is used (TWR-MC-LV3PH) then add SPI0, more GPIO
5. KMS GUI interacts with UART0 and a RDA client in the reference project. If you don’t use UART0 you will need to reduce the com baud rate.
6. IDE uses Debug interface for programming and debug.

Lab sections

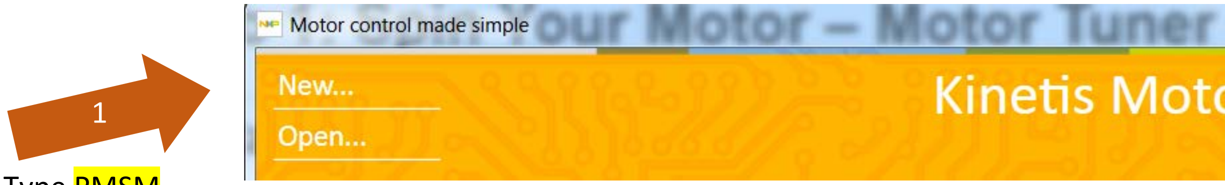
The major sections of this lab are:

- *Using MCUXpresso Pins Tool - Adding ADC input pins and GPIO output to bias the potentiometer*
- *Using MCUXpresso IDE to edit the Senorless Velocity KMS project adding*
    - *Defines*
    - *GPIO initialization code*
    - *Enable and set up user ADC sampling configuration*
    - *Create switch control code to change User States*
    - *Add conversion and control code to control motor speed*
- *Using KMS GUI to monitor the commanded speed and actual speed*

4 Running KMS – creating the reference project

Open KMS and create a FRDM-KV31 Sensorless Velocity project for MCUXpresso

Click on the Kinetis Motor Suite Icon(on the desktop or the windows menu bar.



1) Select New
2) Select Motor Type PMSM
3) Select KV3x
4) Select Freedom
5) Select Sensorless Velocity
6) Select MCUXpresso
7) Change the project Name to FRMDKV31F_SNLESSVEL_MXP_LastName (inserting your last name)
8) Click OK
9) Choose the Communication Port and Save and connect
10) Select Yes if the message pops up saying “The image on the MCU does not match the Application Image for the current project.”
11) Or load the application image if communications cannot be made by selection Project → Load Application Image
12) Connect the KMS GUI with FRDM board.
13) Enter the motor parameters and name of the motor
14) Select play button to save
15) Select Play button to measure the electrical properties
16) Select Play button to measure the mechanical properties, inertia
17) Select play to run to speed. A graph appears and the motor spins up to the rated speed.

5 Open MCUXpresso IDE and Import the project FRMDKV31F_SNLESSVEL_MXP_LastName

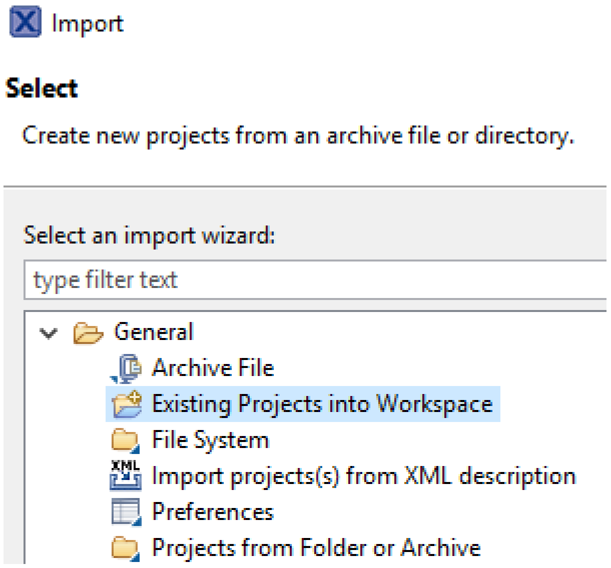
Click on the MCUXpresso IDE Icon (On the desktop or the windows menu bar) and create a new workspace area for your work.

Examine MCUXpresso IDE

A) You should see that the FRDM-KV31 SDK is already installed.

Select Import and select Existing project into workspace

B) From the MCUXpresso IDE quick start panel select Import SDK example



Import

Select

Create new projects from an archive file or directory.

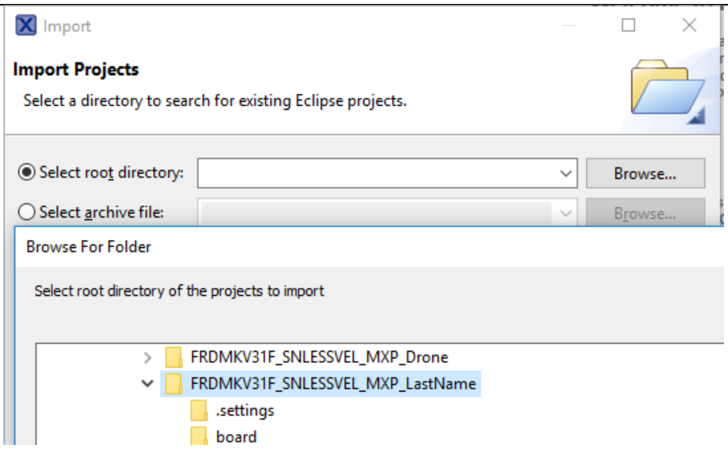
Select an import wizard:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Import projects(s) from XML description
  - Preferences
  - Projects from Folder or Archive

C) Select the “Existing Project into Workspace” and press “Next” on the bottom right

D) Select Browse and select the folder of the project you just created



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: [ ] Browse...

○ Select archive file: [ ] Browse...

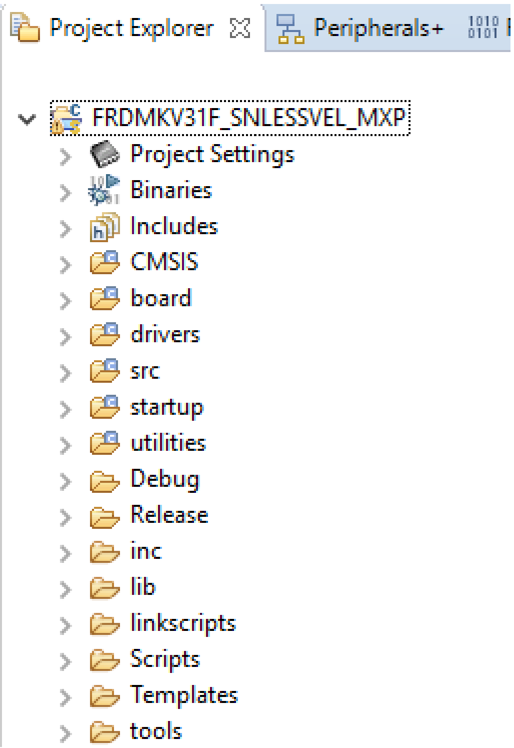
Browse For Folder

Select root directory of the projects to import

- FRDMKV31F_SNLESSVEL_MXP_Drone
- FRDMKV31F_SNLESSVEL_MXP_LastName
  - .settings
  - board

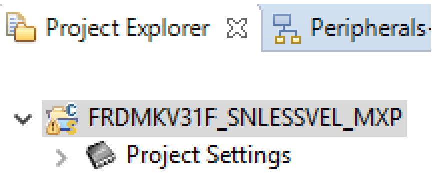
E) Select OK. Then select “Finish”

F) From the ‘Project Explorer’ window select the project and explore the files.



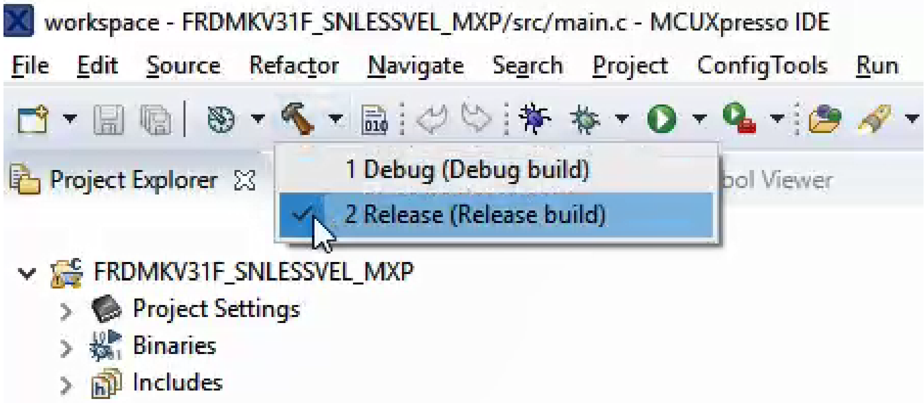
- FRDMKV31F_SNLESSVEL_MXP
  - Project Settings
  - Binaries
  - Includes
  - CMSIS
  - board
  - drivers
  - src
  - startup
  - utilities
  - Debug
  - Release
  - inc
  - lib
  - linkscripts
  - Scripts
  - Templates
  - tools

6 Running the KMS project on the FRDM-KV31 MCU

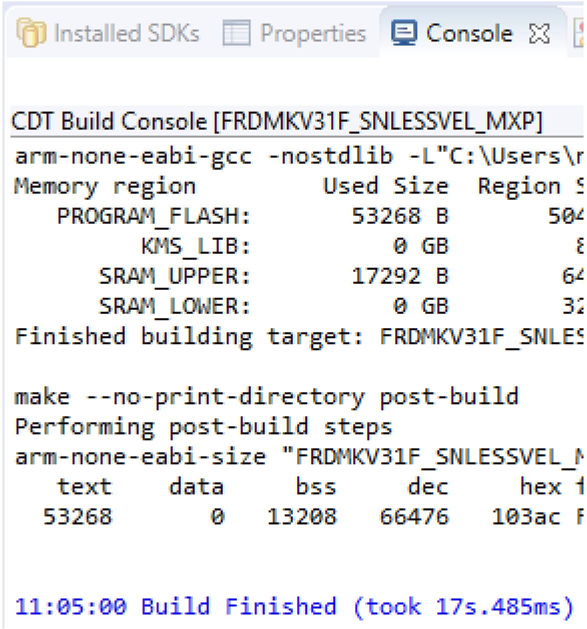


A) Select the Project.

B) From the ICON list select the down triangle next the hammer icon and select 'Release (Release build)'



C) This will build your project. You can view the results of the build in the Console window



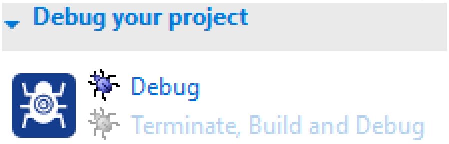
```
CDT Build Console [FRDMKV31F_SNLESSVEL_MXP]
arm-none-eabi-gcc -nostdlib -L"C:\Users\r
Memory region         Used Size  Region S
   PROGRAM_FLASH:        53268 B       504
         KMS_LIB:           0 GB         8
      SRAM_UPPER:        17292 B        64
      SRAM_LOWER:           0 GB        32
Finished building target: FRDMKV31F_SNLES

make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "FRDMKV31F_SNLESSVEL_M
   text    data     bss     dec     hex f
  53268       0   13208   66476   103ac F


11:05:00 Build Finished (took 17s.485ms)
```

D) Go back to the KMS GUI window. Disconnect the KMS GUI from the MCU

click on communication icon (- it will show this after disconnected .

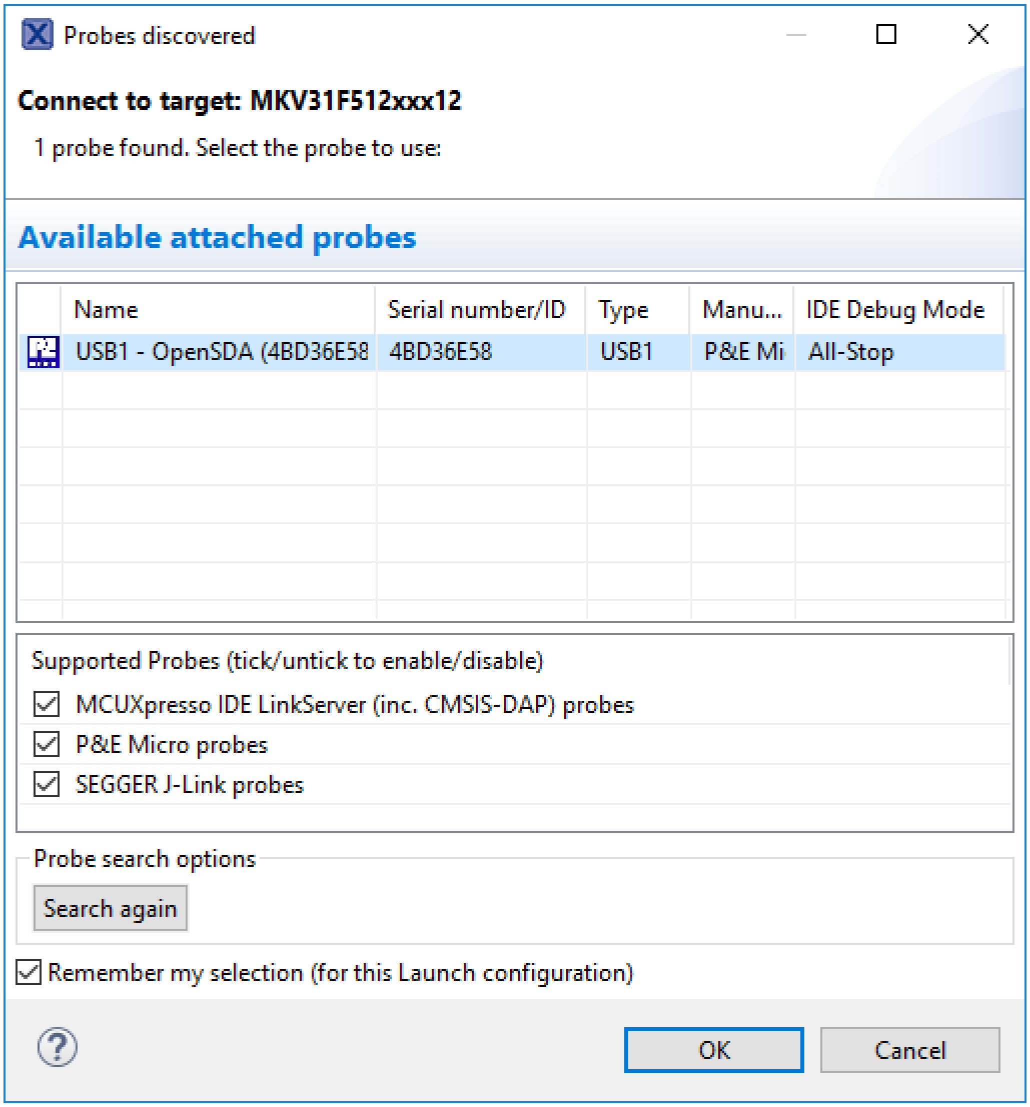
E) Go back to the MCUXpresso IDE

F) From the Quickstart Panel Debug our project select Debug.



NOTE: If you ever want to seach the debug probes again hold down the shift key and click on Debug.

G) Each time you search the debug probes you need to select the debug probe of your target and select OK.



Probes discovered

Connect to target: MKV31F512xxx12

1 probe found. Select the probe to use:

Available attached probes

	Name	Serial number/ID	Type	Manu...	IDE Debug Mode
	USB1 - OpenSDA (4BD36E58	4BD36E58	USB1	P&E Mi	All-Stop

Supported Probes (tick/untick to enable/disable)

☑ MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes

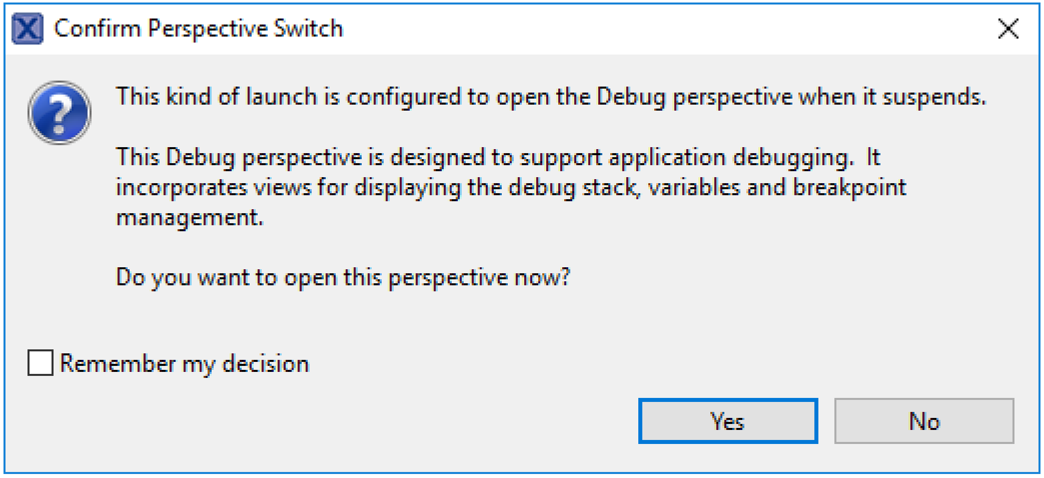
☑ P&E Micro probes

☑ SEGGER J-Link probes

Probe search options

☑ Remember my selection (for this Launch configuration)

H) Check Remember my decision and Click No on the confirm perspective switch



Confirm Perspective Switch

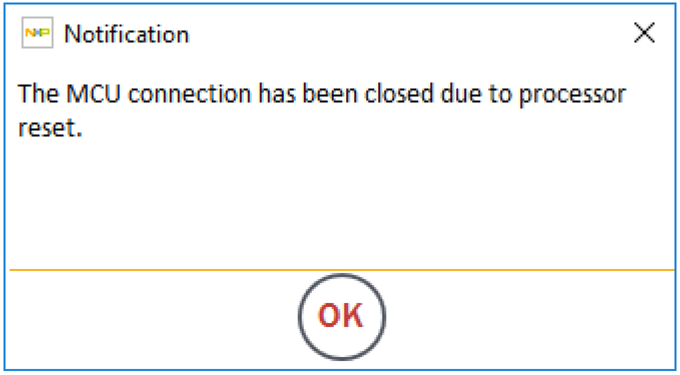
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

☐ Remember my decision

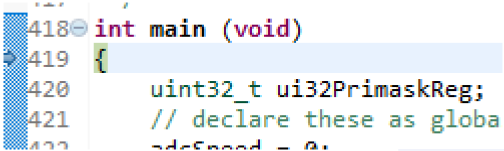
I) If you had KMS connected you will get a message – Press OK



Notification

The MCU connection has been closed due to processor reset.

J) Code execution will halt at the beginning of main.c



```
int main (void)
{
    uint32_t ui32PrimaskReg;
    // declare these as globa
    adcSpeed = 0;
```

K) Press Resume icon or Press F8

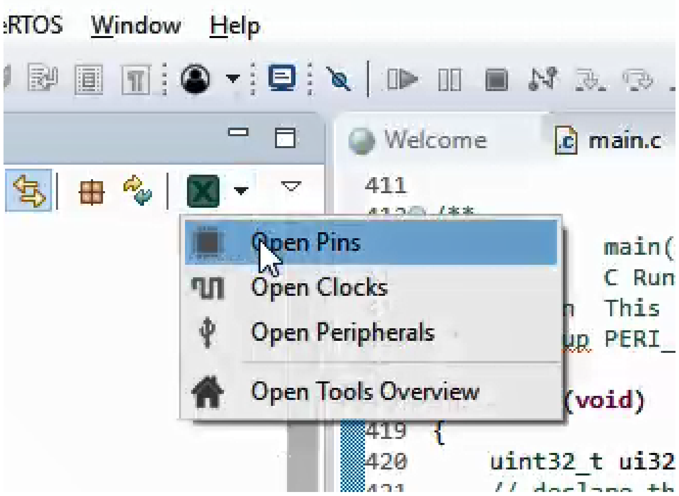
L) The Pause icon will light up indicating the MCU is running code. And the tri-color LED on the FRDM-KV31 board will start flashing blue and green if all is well. Red will show if there is a fault.

M) Close the debug session by selecting Terminate icon.

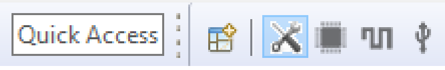
N) Go back to KMS GUI. Reconnect to the target by selecting - it should turn back to this

7 Open MCUXpresso Pins Tool and add ADC and GPIO

A) Select the Pins Tool

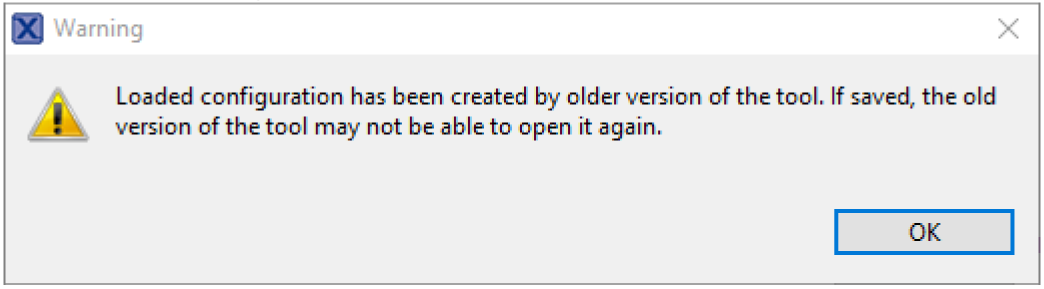


Note: after this entry future entry can be done in the top right had corner



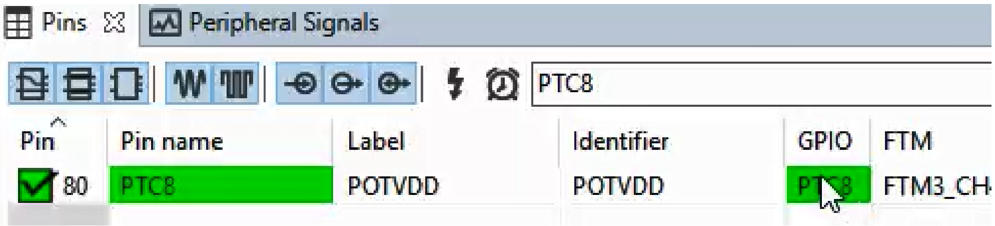
by selecting the chip icon.

B) Press OK to Warning



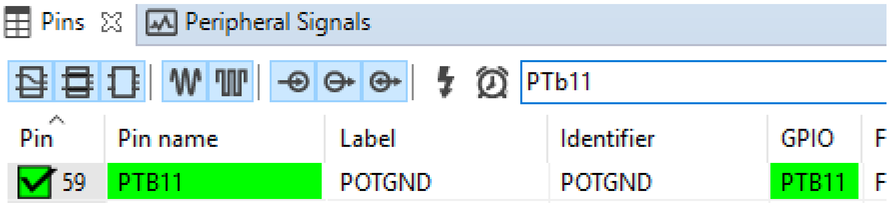
Loaded configuration has been created by older version of the tool. If saved, the old version of the tool may not be able to open it again.

C) Type in the filter box PTC8 and click on the PTC8 text in the GPIO Column. Change the Identifier and Label to POTVDD.



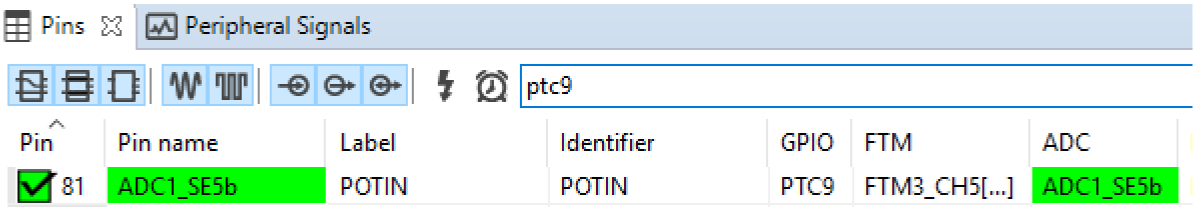
Pin	Pin name	Label	Identifier	GPIO	FTM
☑ 80	PTC8	POTVDD	POTVDD	PTC8	FTM3_CH

D) Type in the filter box PTB11 and click on the PTB11 text in the GPIO column. Change the Identifier and Label to POTGND.



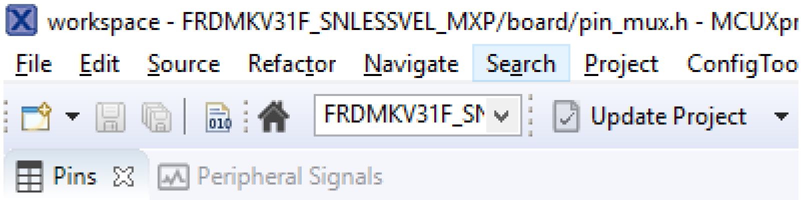
Pin	Pin name	Label	Identifier	GPIO	F
☑ 59	PTB11	POTGND	POTGND	PTB11	F

E) Type in the filter box PTC9 and select ACD1_SE5b text in the ADC column. Change the Identifier and Lable to POTIN



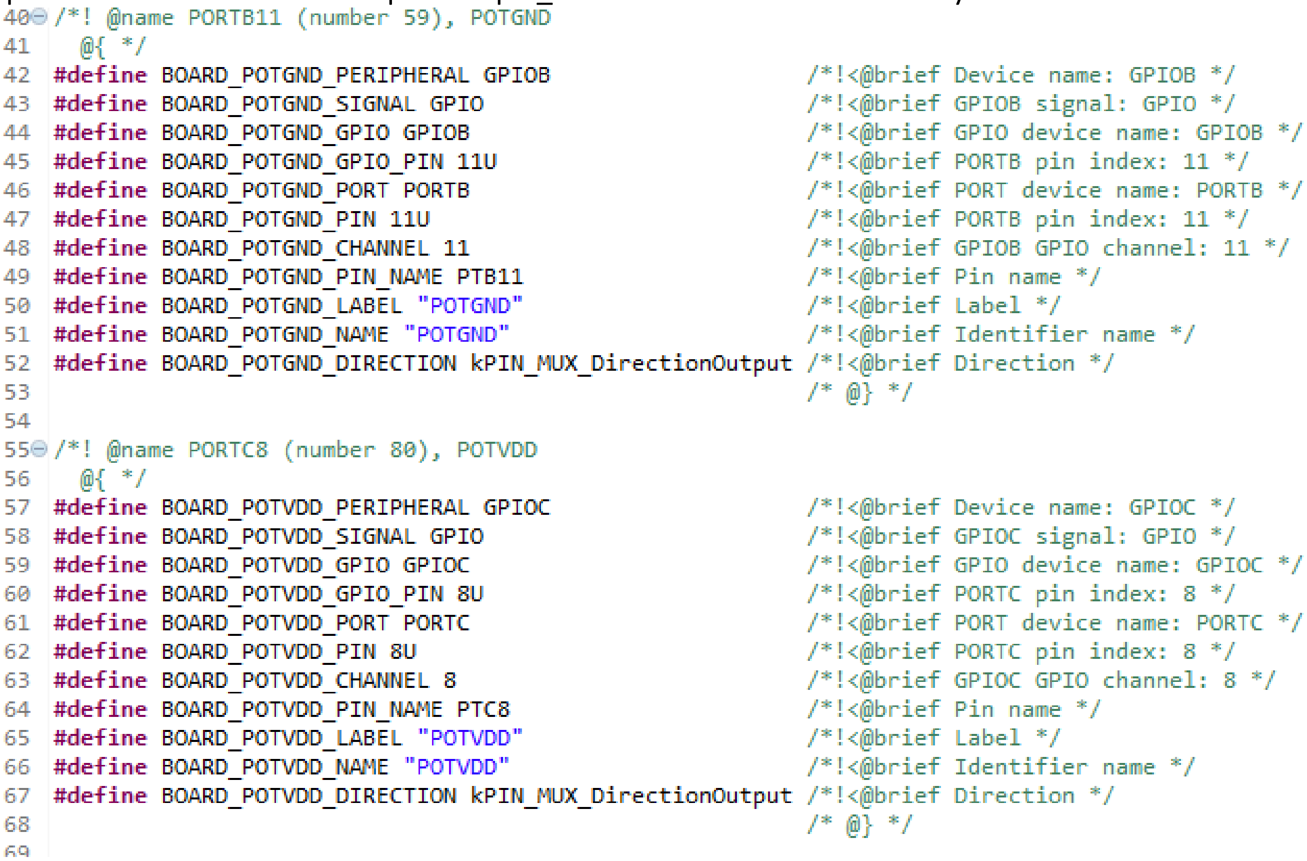
Pin	Pin name	Label	Identifier	GPIO	FTM	ADC
☑ 81	ADC1_SE5b	POTIN	POTIN	PTC9	FTM3_CH5[...]	ADC1_SE5b

F) Update project by selecting Update Project



G) Return to the 'Develop' perspective by selecting this Icon.

H) Open the folder 'boards' and open the pin_mux.h file in the editor. Check your handiwork.

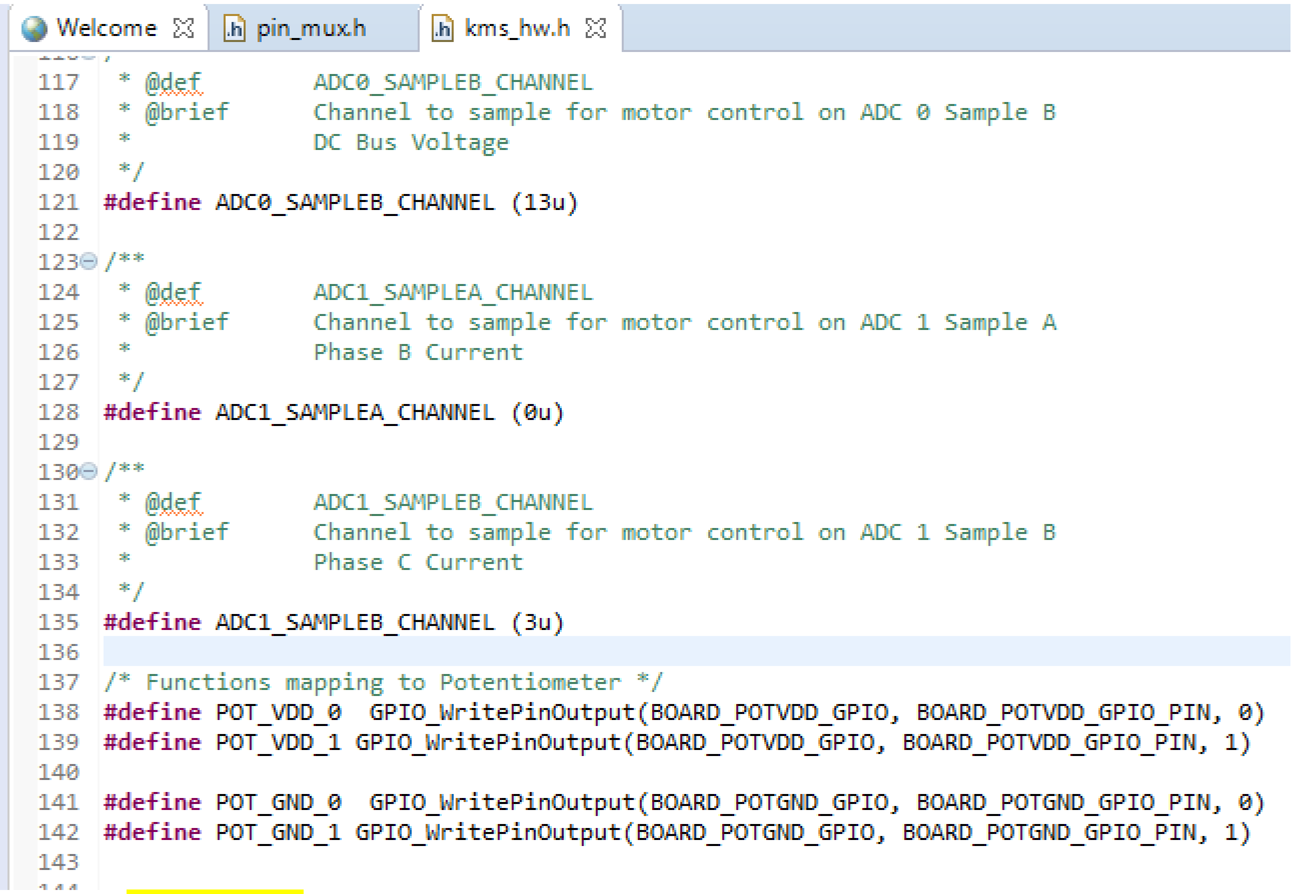


```
/*! @name PORTB11 (number 59), POTGND
  @{ */
#define BOARD_POTGND_PERIPHERAL GPIOB                   /*!<@brief Device name: GPIOB */
#define BOARD_POTGND_SIGNAL GPIO                        /*!<@brief GPIOB signal: GPIO */
#define BOARD_POTGND_GPIO GPIOB                         /*!<@brief GPIO device name: GPIOB */
#define BOARD_POTGND_GPIO_PIN 11U                       /*!<@brief PORTB pin index: 11 */
#define BOARD_POTGND_PORT PORTB                         /*!<@brief PORT device name: PORTB */
#define BOARD_POTGND_PIN 11U                            /*!<@brief PORTB pin index: 11 */
#define BOARD_POTGND_CHANNEL 11                         /*!<@brief GPIOB GPIO channel: 11 */
#define BOARD_POTGND_PIN_NAME PTB11                     /*!<@brief Pin name */
#define BOARD_POTGND_LABEL "POTGND"                     /*!<@brief Label */
#define BOARD_POTGND_NAME "POTGND"                      /*!<@brief Identifier name */
#define BOARD_POTGND_DIRECTION kPIN_MUX_DirectionOutput /*!<@brief Direction */
                                                        /* @} */

/*! @name PORTC8 (number 80), POTVDD
  @{ */
#define BOARD_POTVDD_PERIPHERAL GPIOC                   /*!<@brief Device name: GPIOC */
#define BOARD_POTVDD_SIGNAL GPIO                        /*!<@brief GPIOC signal: GPIO */
#define BOARD_POTVDD_GPIO GPIOC                         /*!<@brief GPIO device name: GPIOC */
#define BOARD_POTVDD_GPIO_PIN 8U                        /*!<@brief PORTC pin index: 8 */
#define BOARD_POTVDD_PORT PORTC                         /*!<@brief PORT device name: PORTC */
#define BOARD_POTVDD_PIN 8U                             /*!<@brief PORTC pin index: 8 */
#define BOARD_POTVDD_CHANNEL 8                          /*!<@brief GPIOC GPIO channel: 8 */
#define BOARD_POTVDD_PIN_NAME PTC8                      /*!<@brief Pin name */
#define BOARD_POTVDD_LABEL "POTVDD"                     /*!<@brief Label */
#define BOARD_POTVDD_NAME "POTVDD"                      /*!<@brief Identifier name */
#define BOARD_POTVDD_DIRECTION kPIN_MUX_DirectionOutput /*!<@brief Direction */
                                                        /* @} */
```

8 Edit the source and header files to enable the ADC reading

A) Open the file **kms_hw.h** file and add the following text at line 137

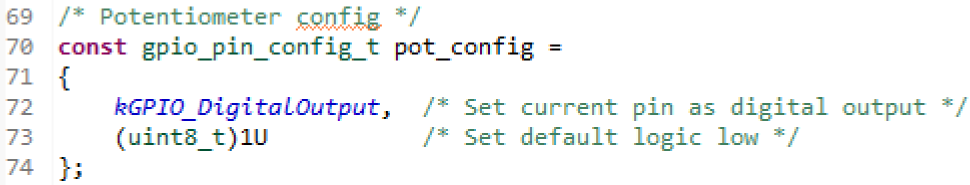
```
/* Functions mapping to Potentiometer */
#define POT_VDD_0  GPIO_WritePinOutput(BOARD_POTVDD_GPIO, BOARD_POTVDD_GPIO_PIN, 0)
#define POT_VDD_1 GPIO_WritePinOutput(BOARD_POTVDD_GPIO, BOARD_POTVDD_GPIO_PIN, 1)
#define POT_GND_0  GPIO_WritePinOutput(BOARD_POTGND_GPIO, BOARD_POTGND_GPIO_PIN, 0)
#define POT_GND_1 GPIO_WritePinOutput(BOARD_POTGND_GPIO, BOARD_POTGND_GPIO_PIN, 1)
```



```
117   * @def        ADC0_SAMPLEB_CHANNEL
118   * @brief      Channel to sample for motor control on ADC 0 Sample B
119   *             DC Bus Voltage
120   */
121  #define ADC0_SAMPLEB_CHANNEL (13u)
122
123  /**
124   * @def        ADC1_SAMPLEA_CHANNEL
125   * @brief      Channel to sample for motor control on ADC 1 Sample A
126   *             Phase B Current
127   */
128  #define ADC1_SAMPLEA_CHANNEL (0u)
129
130  /**
131   * @def        ADC1_SAMPLEB_CHANNEL
132   * @brief      Channel to sample for motor control on ADC 1 Sample B
133   *             Phase C Current
134   */
135  #define ADC1_SAMPLEB_CHANNEL (3u)
136
137  /* Functions mapping to Potentiometer */
138  #define POT_VDD_0  GPIO_WritePinOutput(BOARD_POTVDD_GPIO, BOARD_POTVDD_GPIO_PIN, 0)
139  #define POT_VDD_1 GPIO_WritePinOutput(BOARD_POTVDD_GPIO, BOARD_POTVDD_GPIO_PIN, 1)
140
141  #define POT_GND_0  GPIO_WritePinOutput(BOARD_POTGND_GPIO, BOARD_POTGND_GPIO_PIN, 0)
142  #define POT_GND_1 GPIO_WritePinOutput(BOARD_POTGND_GPIO, BOARD_POTGND_GPIO_PIN, 1)
143
```

B) Open file **kms_hw.c** and add the following lines of code after line 68.

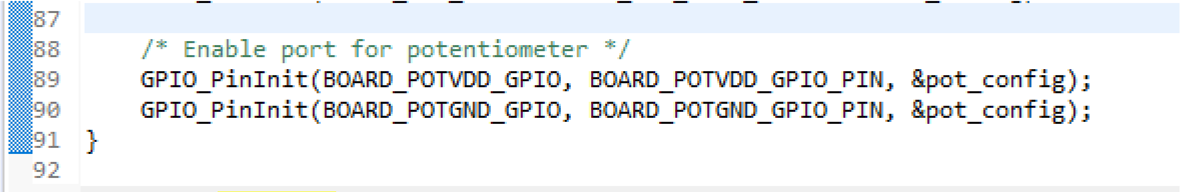
```
/* Potentiometer config */
const gpio_pin_config_t pot_config =
{
    kGPIO_DigitalOutput,  /* Set current pin as digital output */
    (uint8_t)1U           /* Set default logic low */
};
```



```
69  /* Potentiometer config */
70  const gpio_pin_config_t pot_config =
71  {
72      kGPIO_DigitalOutput,  /* Set current pin as digital output */
73      (uint8_t)1U           /* Set default logic low */
74  };
```

C) Next add this code after line 87.

```
    /* Enable port for potentiometer */
 GPIO_PinInit(BOARD_POTVDD_GPIO, BOARD_POTVDD_GPIO_PIN, &pot_config);
 GPIO_PinInit(BOARD_POTGND_GPIO, BOARD_POTGND_GPIO_PIN, &pot_config);
```



```
87
88      /* Enable port for potentiometer */
89      GPIO_PinInit(BOARD_POTVDD_GPIO, BOARD_POTVDD_GPIO_PIN, &pot_config);
90      GPIO_PinInit(BOARD_POTGND_GPIO, BOARD_POTGND_GPIO_PIN, &pot_config);
91  }
92
```

D) Open file **main.c** and edit the file to enable the user ADC inputs. We only need one input for this demonstration, but the code allows for an input on ADC0 and ADC1 to be sampled. We only need ADC!_SE5b – [ADC 1 Single Ended mux 5b].

E) Change line 194 – change it to true,
static const bool bEnableRoundRobinAdc = true; /* Enable use of ADC channels for application layer signals. When false optimized away */

F) Change line 197 to 1 channels
/* Storage for user ADC samples */
#define NUMBER_USER_ADC_CHANNELS (1)

G) Change line 198 to channel 5
static uint16_t adcUserChannelsChannel0[NUMBER_USER_ADC_CHANNELS] = {5};

H) Change line 199 to channel 5
static uint16_t adcUserChannelsChannel1[NUMBER_USER_ADC_CHANNELS] = {5};

I) Change line 200 to 1
static uint16_t adcUserChannelsHW_MUX0[NUMBER_USER_ADC_CHANNELS]= {1};

J) Change line 201 to 1
static uint16_t adcUserChannelsHW_MUX1[NUMBER_USER_ADC_CHANNELS]= {1};:

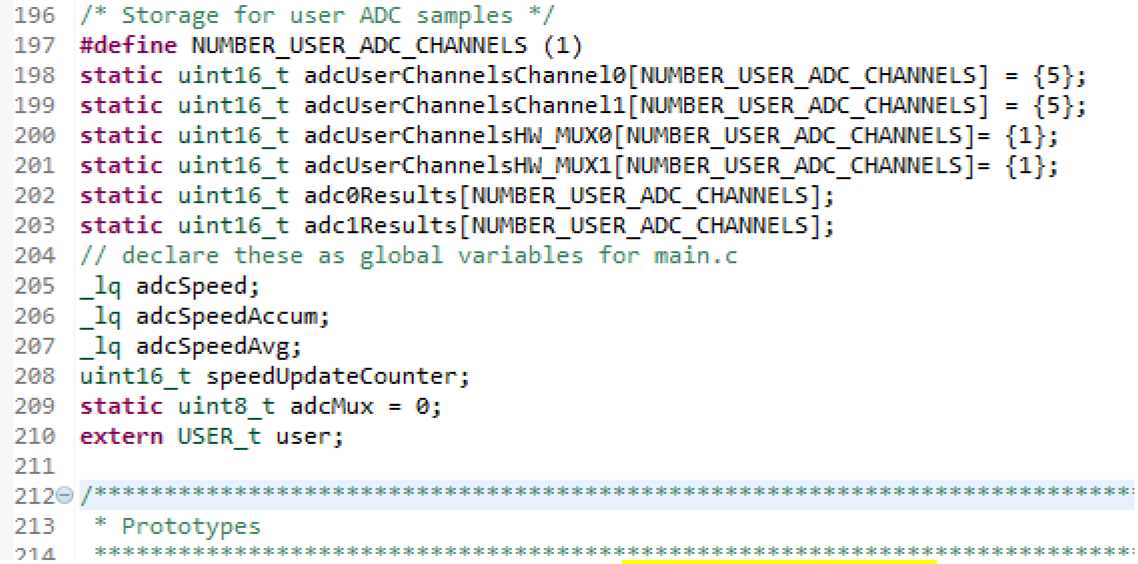
K) Add the following code lines to main.c at line 204

```
// declare these as global variables for main.c
_lq adcSpeed;
_lq adcSpeedAccum;
_lq adcSpeedAvg;
uint16_t speedUpdateCounter;
static uint8_t adcMux = 0;
```

L) Add the following global variable reference so that main can access user states.

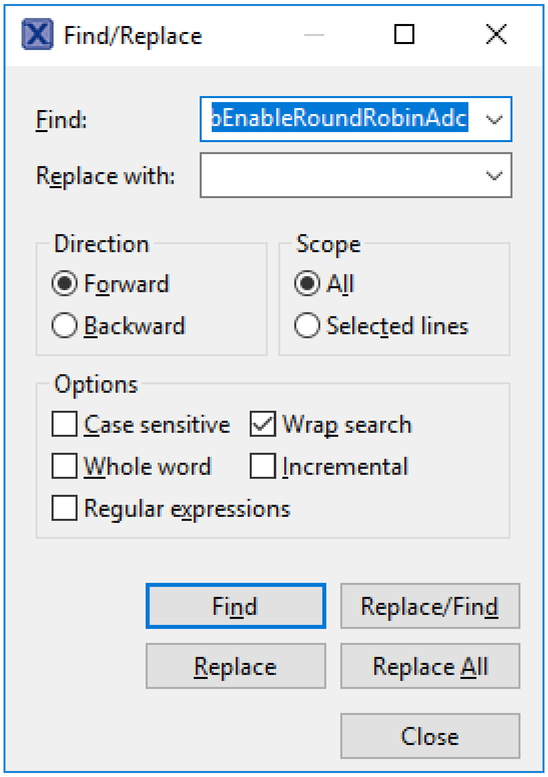
```
extern USER_t user;
```

It now show look like this:



```
/* Storage for user ADC samples */
#define NUMBER_USER_ADC_CHANNELS (1)
static uint16_t adcUserChannelsChannel0[NUMBER_USER_ADC_CHANNELS] = {5};
static uint16_t adcUserChannelsChannel1[NUMBER_USER_ADC_CHANNELS] = {5};
static uint16_t adcUserChannelsHW_MUX0[NUMBER_USER_ADC_CHANNELS]= {1};
static uint16_t adcUserChannelsHW_MUX1[NUMBER_USER_ADC_CHANNELS]= {1};
static uint16_t adc0Results[NUMBER_USER_ADC_CHANNELS];
static uint16_t adc1Results[NUMBER_USER_ADC_CHANNELS];
// declare these as global variables for main.c
_lq adcSpeed;
_lq adcSpeedAccum;
_lq adcSpeedAvg;
uint16_t speedUpdateCounter;
static uint8_t adcMux = 0;
extern USER_t user;

/*****************************************************************************
 * Prototypes
 *****************************************************************************
```

M) Find the next occurrence of the variable ‘bEnableRoundRobinAdc’. If you highlight the variable and then hit CTRL-F it will fill in the find window.



Find/Replace

Find: bEnableRoundRobinAdc

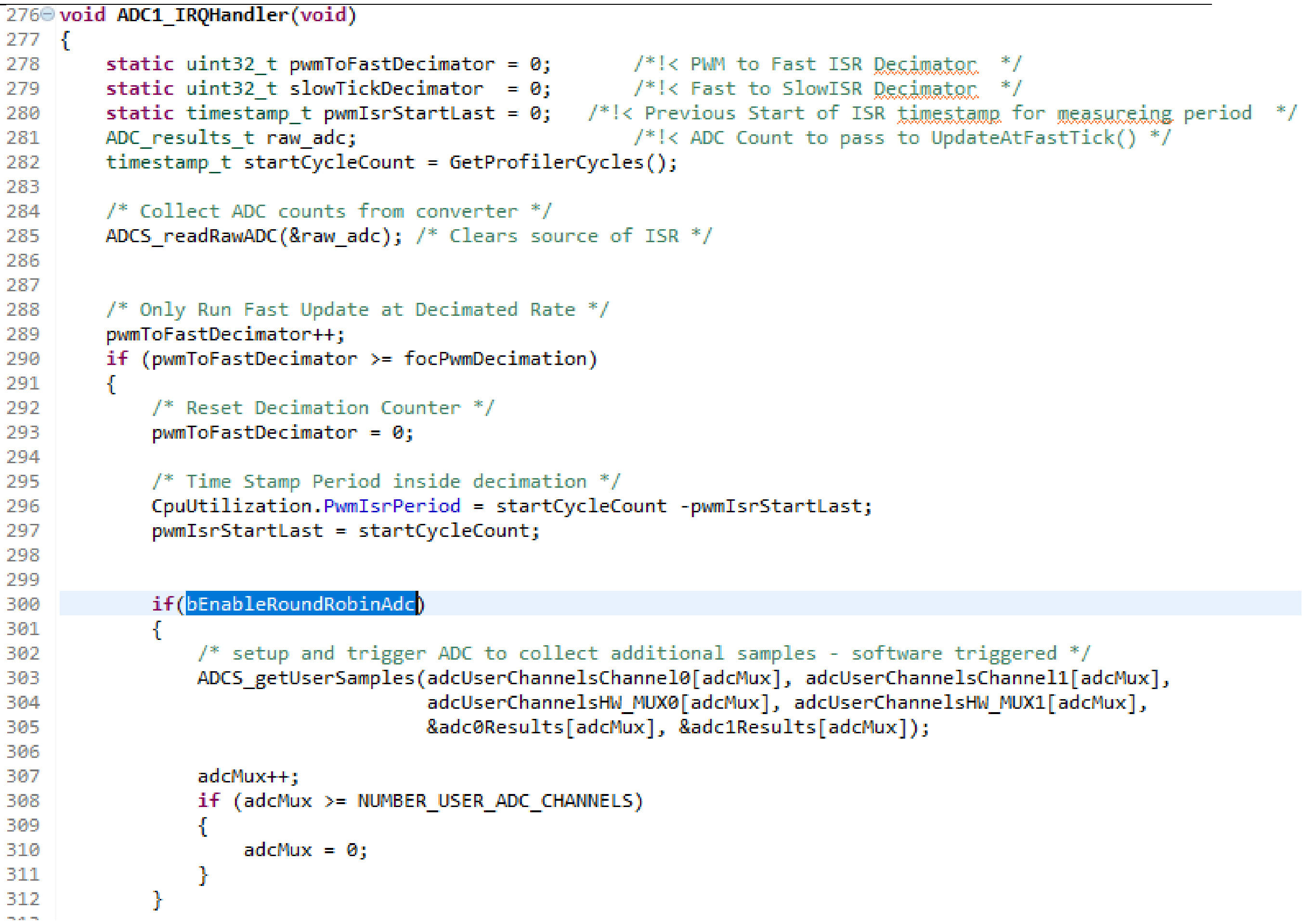
Replace with:

Direction
● Forward
○ Backward

Scope
● All
○ Selected lines

Options
□ Case sensitive ☑ Wrap search
□ Whole word □ Incremental
□ Regular expressions

N) Look at the code in the function ADC1_IRQHandler (). Notice the code that will now execute with the setting of ‘bEnableRoundRobinAdc’ true.



```
void ADC1_IRQHandler(void)
{
    static uint32_t pwmToFastDecimator = 0;        /*!< PWM to Fast ISR Decimator  */
    static uint32_t slowTickDecimator  = 0;        /*!< Fast to SlowISR Decimator  */
    static timestamp_t pwmIsrStartLast = 0;    /*!< Previous Start of ISR timestamp for measureing period  */
    ADC_results_t raw_adc;                         /*!< ADC Count to pass to UpdateAtFastTick() */
    timestamp_t startCycleCount = GetProfilerCycles();

    /* Collect ADC counts from converter */
    ADCS_readRawADC(&raw_adc); /* Clears source of ISR */


    /* Only Run Fast Update at Decimated Rate */
    pwmToFastDecimator++;
    if (pwmToFastDecimator >= focPwmDecimation)
    {
        /* Reset Decimation Counter */
        pwmToFastDecimator = 0;

        /* Time Stamp Period inside decimation */
        CpuUtilization.PwmIsrPeriod = startCycleCount -pwmIsrStartLast;
        pwmIsrStartLast = startCycleCount;


        if(bEnableRoundRobinAdc)
        {
            /* setup and trigger ADC to collect additional samples - software triggered */
            ADCS_getUserSamples(adcUserChannelsChannel0[adcMux], adcUserChannelsChannel1[adcMux],
                                adcUserChannelsHW_MUX0[adcMux], adcUserChannelsHW_MUX1[adcMux],
                                &adc0Results[adcMux], &adc1Results[adcMux]);

            adcMux++;
            if (adcMux >= NUMBER_USER_ADC_CHANNELS)
            {
                adcMux = 0;
            }
        }
```

9 Edit function main.c adding switch control code and ADC conversion code

A) Go to main.c line 419 using CTRL-L and typing 419.

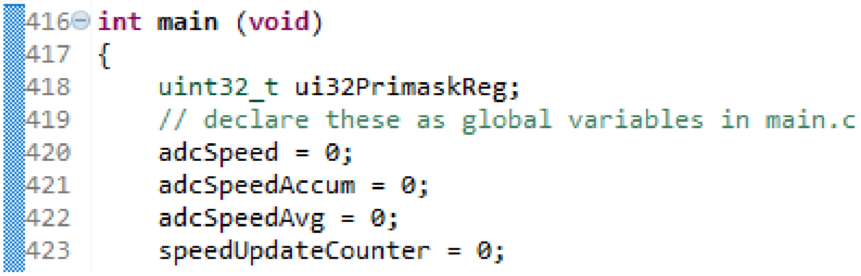
// declare these as global variables in main.c

adcSpeed = 0;

adcSpeedAccum = 0;

adcSpeedAvg = 0;

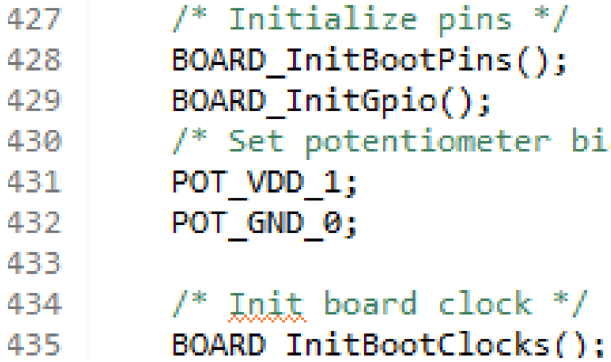
speedUpdateCounter = 0;



```
416 int main (void)
417 {
418     uint32_t ui32PrimaskReg;
419     // declare these as global variables in main.c
420     adcSpeed = 0;
421     adcSpeedAccum = 0;
422     adcSpeedAvg = 0;
423     speedUpdateCounter = 0;
```

B) Go to main.c line 431 using CTRL-L and typing 431. Add the following code to initialize the output pins biasing the potentiometer:

```
    /* Set potentiometer bias voltage so */
     POT_VDD_1;
     POT_GND_0;
```



```
427     /* Initialize pins */
428     BOARD_InitBootPins();
429     BOARD_InitGpio();
430     /* Set potentiometer bias voltage so */
431     POT_VDD_1;
432     POT_GND_0;
433
434     /* Init board clock */
435     BOARD_InitBootClocks();
```

C) Got to line 507 and add the following code to control the motor user state:

```
        /* Switch is active Low but we want active high logic */
    if (GPIO_ReadPinInput(BOARD_SW3_GPIO, BOARD_SW3_GPIO_PIN) == 0)
    {
      user.state = USER_RUN_SPEED;
    }
    if (GPIO_ReadPinInput(BOARD_SW2_GPIO, BOARD_SW3_GPIO_PIN) == 0)
    {
      user.state= USER_RUN_PLAN;
    }
```

D) Next, add the conversion code that will take the ADC signed value and convert and scale it to LQ format.

```
        /* this code takes the ADC reading, downshifts it by 4 to remove noise
         and uses that as a percentage of the maximum applicaton speed (in this case
6krpm)*/

        adcSpeed = _LQmpyLQX((adc1Results[0] >> 4), 8, _LQ(6000.0/FULL_SCALE_SPEED_RPM),
24);
        adcSpeed = _LQsat(adcSpeed, _LQ(1.0), _LQ(0.0));
```

E) Add this code next to average the ADC reading and drive the

```
         /* this code averages the adc reading over 10 samples before setting
         *  it to user.command.targetSpeed. It will also handle setting the
         *  control mode if the commanded speed is larger than 0*/
        if(adcSpeed > _LQ(0.0))
        {
                        //user.state= USER_RUN_SPEED;
                        if(speedUpdateCounter >= 10)
                        {
                             speedUpdateCounter = 0;
                             adcSpeedAvg=(adcSpeedAccum/10)&0x00FF0000;
                             user.command.targetSpeed = adcSpeedAvg;
                             adcSpeedAccum = 0;
                        }
                        else
```

```
                                {
                                     adcSpeedAccum = adcSpeed + adcSpeedAccum;
                                     speedUpdateCounter++;
                                }
                }
                else
                {
                        user.state= USER_IDLE;
                        adcSpeedAccum = 0;
                        adcSpeedAvg = 0;
                        speedUpdateCounter = 0;
                }
```

F) Build, Download and debug the resulting code

10 Open KMS GUI, Add variables to watch window and Speed scope plot

A) Open the KMS GUI.

B) Reconnect to the MCU Go back to KMS GUI. Reconnect to the target by selecting - it should turn back to this

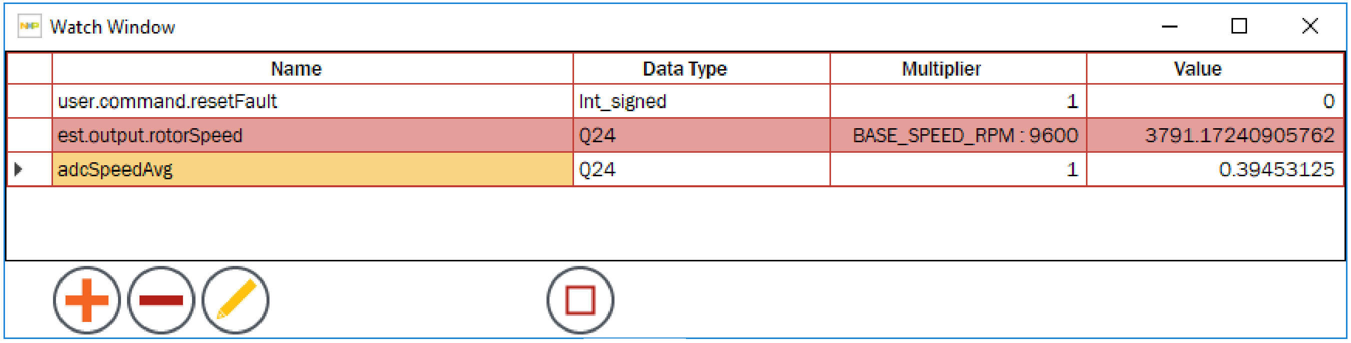
C) Try running speed and running the plan in the GUI.

D) Open the Watch Window by selecting

Add two variables. Hit the Plus Sign + − ✎
And est.output.rotorSpeed with a data type of Q24
Add adcSpeedAvg with a data type of Q24

E) Press the Run Button ▷ it will change to this □

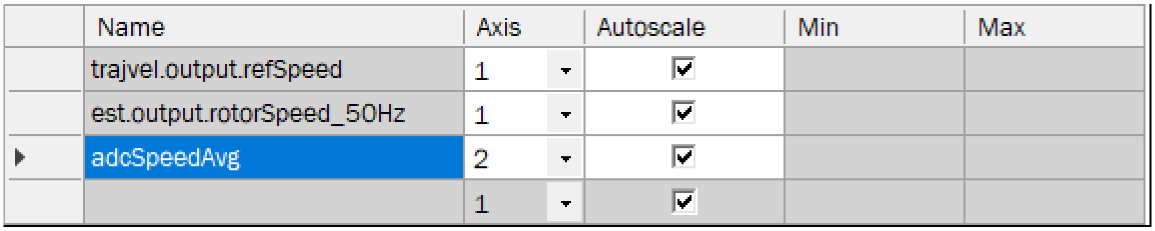
F) Press SW3 on the FRDM-KV31 Board and adjust potentiometer and watch the speed change



Name	Data Type	Multiplier	Value
user.command.resetFault	Int_signed	1	0
est.output.rotorSpeed	Q24	BASE_SPEED_RPM : 9600	3791.17240905762
adcSpeedAvg	Q24	1	0.39453125

G) Open the Software Oscilloscope by selecting

H) Select the SpeedFeedback plot and hit the plus sign to add the variable adcSpeedAvg on Axis 2



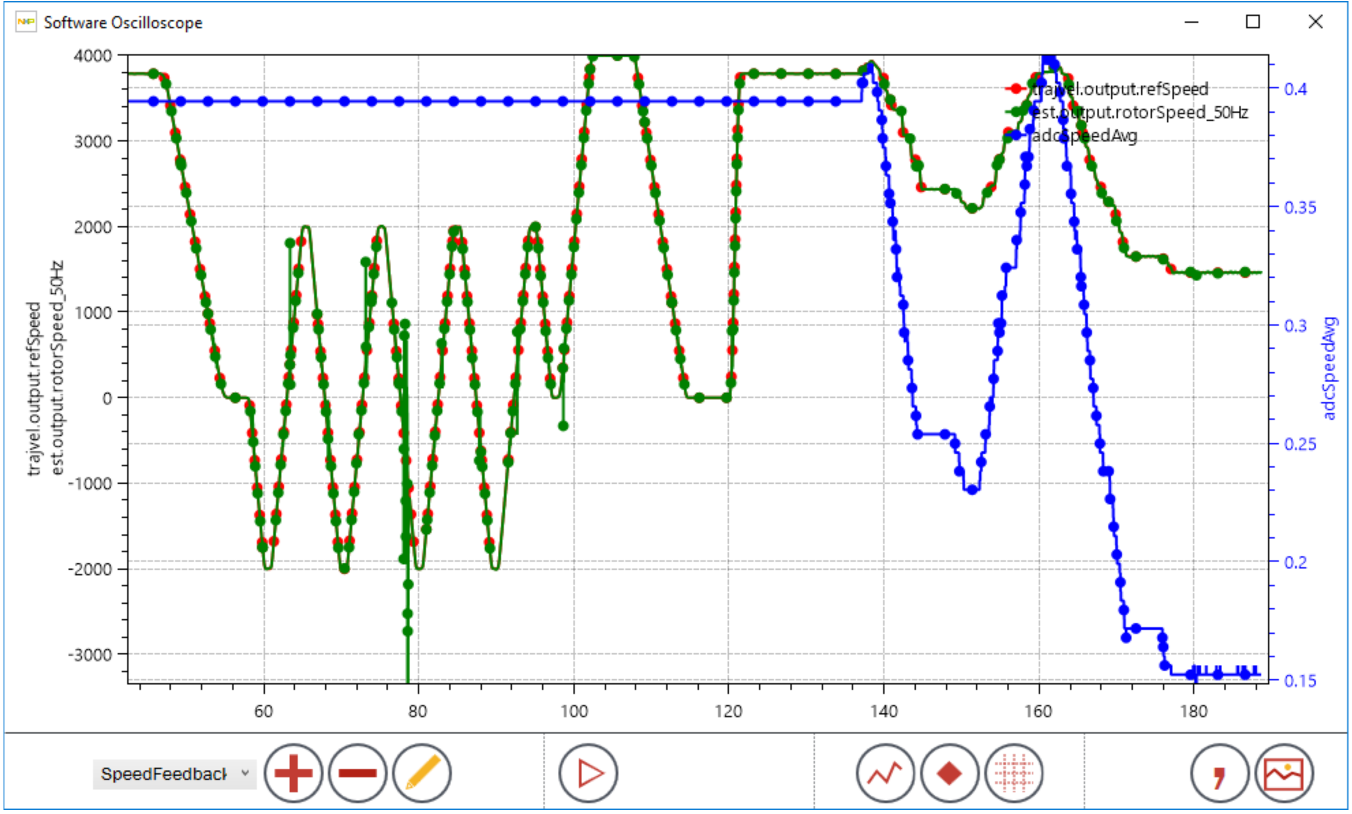
	Name	Axis	Autoscale	Min	Max
	trajvel.output.refSpeed	1	☑		
	est.output.rotorSpeed_50Hz	1	☑		
▶	adcSpeedAvg	2	☑		
		1	☑		

I) Press the Update button

J) And press play

K) Press SW2 and watch the graph show the motion control plan.

L) Press SW3 and while Adjusting the potentiometer and watch the resulting graph.



Software Oscilloscope
4000
3000
2000
1000
0
-1000
-2000
-3000
trajvel.output.refSpeed
est.output.rotorSpeed_50Hz
60
80
100
120
140
160
180
trajvel.output.refSpeed
est.output.rotorSpeed_50Hz
adcSpeedAvg
0.4
0.35
0.3
0.25
0.2
0.15
adcSpeedAvg
SpeedFeedback