GUI Quick Start Guide: InstaSPIN UNIVERSAL

Version 1.0.3 *Motor Solutions*

Overview

A series of MotorWare projects are available to help you evaluate InstaSPIN™-FOC or InstaSPIN™-MOTION on your desired Piccolo MCU. The InstaSPIN UNIVERSAL GUI allows you to instrument and interact with these projects over a JTAG connection.

The InstaSPIN Universal GUI Supports:

- Piccolo InstaSPIN enabled controllers
  - LAUNCHXL-F28027F LaunchPad for InstaSPIN-FOC
    - Includes on-card XDS100v2 JTAG (isolated)
  - LAUNCHXL-F28069M LaunchPad for InstaSPIN-FOC and InstaSPIN-MOTION
    - Includes on-card XDS100v2 JTAG (isolated)
  - TMDSCNCD28027F controlCARD for InstaSPIN-FOC
    - Does NOT include on-card emulation or isolation
    - Select the XDS100v1 emulator when executing the GUI
  - TMDSCNCD28054MISO controlCARD for InstaSPIN-FOC and InstaSPIN-MOTION
    - Includes on-card XDS100v2 JTAG (isolated)
  - TMDSCNCD28069MISO controlCARD for InstaSPIN-FOC and InstaSPIN-MOTION
    - Includes on-card XDS100v2 JTAG (isolated)
- Any InstaSPIN enabled Piccolo devices on custom hardware with a JTAG connection
- GUI Composer Runtime (included in the download), or GUI Composer as installed in CCSv5.5 and higher
- 3-phase Inverters
  - The GUI itself has no dependence on the inverter
  - The MotorWare projects are board specific. Be sure to build the MotorWare binary for the appropriate board/inverter.
  - Build a MotorWare based binary for your own custom inverter

Version: 1.0.3

Revision History:

1.0.3	**January 22, 2015**	**Update for LAUNCHXL-F28069M**
1.0.2	**April 23, 2014**	**Update for F2805x and InstaSPIN-MOTION support**
1.0.1	**October 30, 2013**	**First release**

Table of Contents

TI Spins Motors

Installation

1. Run the UNIVERSAL GUI installation .exe using the latest version from www.ti.com/tool/instaspinuniversalgui

   a. Accept the license agreement (Figure 1)

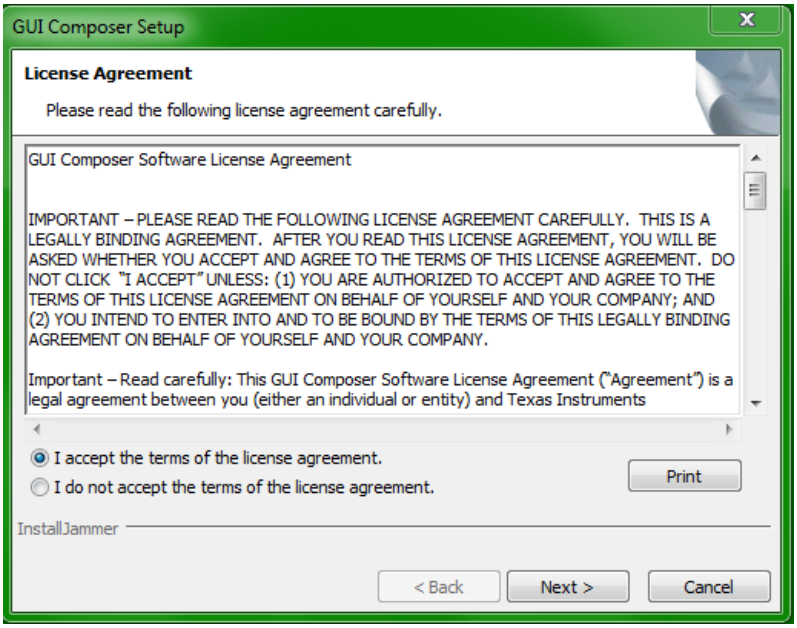


Figure 1 - License Agreement

   b. Recommended to keep default destination location (Figure 2)

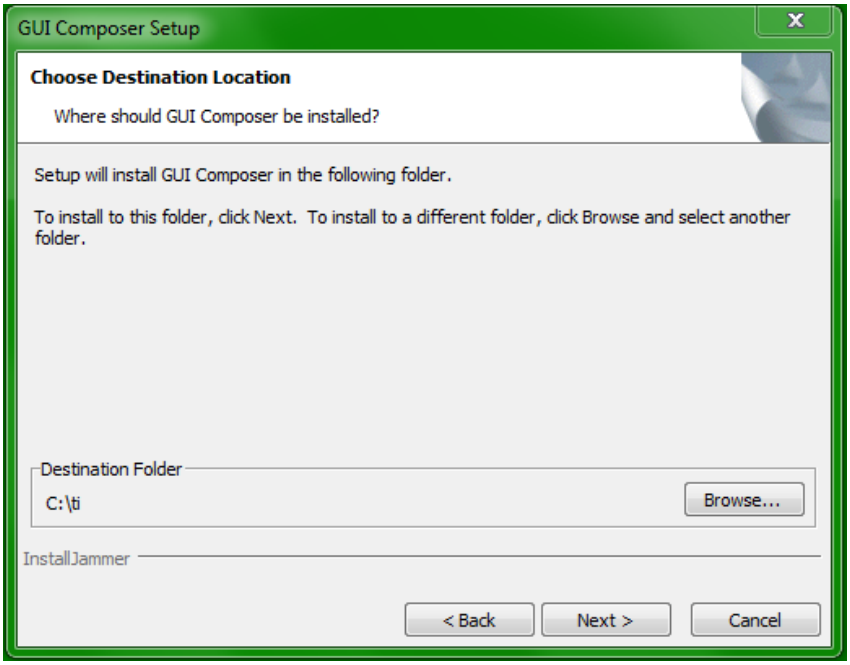


Figure 2 - Destination Folder

2. Install Code Composer Studio v5.5 or higher from
   http://processors.wiki.ti.com/index.php/Download_CCS

Hardware Set-up

Please review the quick start and/or hardware guides for your particular controlCARD / LaunchPad and motor drive kit – available through MotorWare - for details of hardware set-up, including jumper and switch settings

Overview of Process for using the GUI

1. Create a binary from any MotorWare project using CCStudio

   A series of MotorWare projects are available to help you evaluate InstaSPIN™-FOC and/or InstaSPIN™-MOTION on your desired Piccolo MCU. The MotorWare projects are board specific, so you must build the MotorWare binary for the appropriate board/inverter, or for your own custom inverter

   The InstaSPIN UNIVERSAL GUI has no dependence on the inverter. The InstaSPIN UNIVERSAL GUI allows you to instrument and interact with any of the InstaSPIN MotorWare projects over a JTAG connection.

   Example: You will typically start with proj_lab02a, 2b, or 2c, which will automatically identify the motor parameters. The MotorWare directory is organized by board. Select proj_lab02a, 2b, or 2c from the projects directory for the board that you are using (Figure 3).

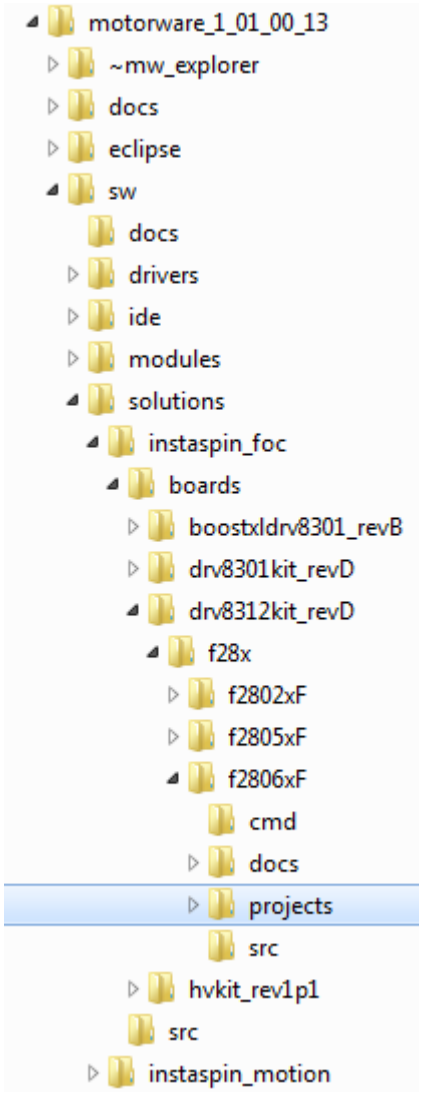


Figure 3 - MotorWare Directory Structure

2. Launch GUI Composer UNIVERSAL GUI using either method:

    a. Standalone: In standalone mode, the InstaSPIN UNIVERSAL GUI will run as an executable file on your PC. CCStudio is required to create the binaries. After the binaries are transferred to the MCU, CCStudio can be closed.

        Note: Some of the MotorWare projects may require you to change the motor settings and recompile the binaries. Therefore, we recommend that you use the Standalone method to test your board and control design.

    b. Inside of CCStudio: This method is recommended when using the MotorWare projects to evaluate InstaSPIN. This method allows you to modify variables and recompile the binaries, as instructed in the MotorWare projects.

Create a Binary

1. **The InstaSPIN Universal GUI allows you to instrument bound variables for each compiled MotorWare project (.out) on the Piccolo MCU**

   a. The variables are in the gMotorVars structure (as defined in the main.h or main_position.h header files)

   b. Each InstaSPIN MotorWare project will use a subset of these variables

      Example: the ability to change the Speed Kp is introduced in proj_lab05b:

      CTRL_setKp(ctrlHandle,CTRL_Type_PID_spd,gMotorVars.Kp_spd);

      If you compile and load proj_lab03.c, changing Speed Kp has no effect because the variable is not used in this project.

   c. If you are not sure if a particular variable is used in a project, do a text search for gMotorVars.xxxx in the project .c file

2. **Use CCStudio to compile a MotorWare project into a .out that can be loaded onto the Piccolo MCU**

   a. CCS version and compiler version notes

      i. CCS (download)

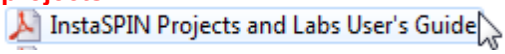
         1. CCSv5.5+ is required to run the GUI inside of CCS GUI Composer

      ii. Compiler (download or through Help → Check for Updates)

         1. Recommend 6.2.3+, or if necessary, version 6.1.5

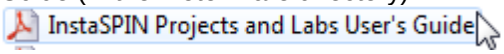
         2. **Do NOT use compiler versions 6.2.2, 6.2.1, or 6.2.0 as they include an IQMath compiler bug**

3. **Select the MotorWare project that meets your application needs. See the InstaSPIN Projects & Labs User's Guide (in the MotorWare directory) for a list of available projects.**

4. **The user.h file specifies the settings for your motor. Make sure that the appropriate settings are selected in the user.h, and that the file is saved before building the project.**

   a. Follow the detailed instructions for each project in the InstaSPIN Projects & Labs User's Guide (in the MotorWare directory)

   b. See the “Use Example” section of this document for a quick overview and example:

      i. InstaSPIN-FOC Use Example

      ii. InstaSPIN-MOTION Use Example

   c. Visit the **InstaSPIN e2e forum** for tips, tricks, and assistance.

Connect the GUI

1. With the DC bus powered, connect a USB cable from your PC to the controlCARD or LaunchPad
2. If using the high voltage kit, proceed with connecting to the target before energizing the high voltage AC input (110-220Vac) or high voltage DC Bus (50-350Vdc)
3. Verify the connection to the FTDI XDS100v2 emulator by checking the Windows Device Manager for Ports: USB Serial Port (COMxxx) and TI XDS100 Channel A and B (Figure 4)

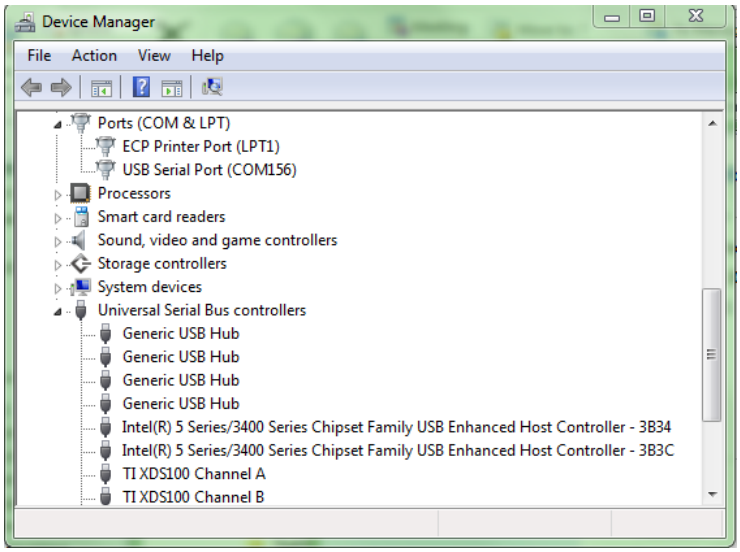


Figure 4 - Windows Device Manager

Run the GUI

Option 1: Standalone GUI .exe

The InstaSPIN Universal GUI is a web-based application that runs on your PC. CCStudio is required to create the binaries. After the binaries are transferred to the MCU, CCStudio can be closed. Some of the MotorWare projects may require you to change the variables in the user.h files and recompile the binaries. Therefore, we recommend that you use the Standalone method to test your board and control design. We do not recommend using the Standalone method when evaluating InstaSPIN.

The InstaSPIN Universal GUI application and associated files can be found in C:\ti\guicomposer\webapps

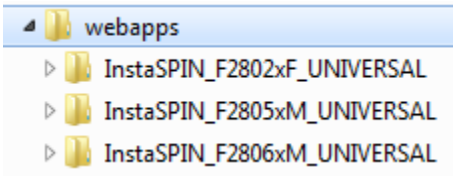


Figure 5 - Standalone GUI .exe Directories

a. The differences between the webapps is

   i. InstaSPIN_F2802xF_UNIVERSAL is for evaluating InstaSPIN-FOC on the F28027F MCU.

   ii. InstaSPIN_F2805xM_UNIVERSAL is for evaluating InstaSPIN-FOC or InstaSPIN-MOTION on the F28054M MCU.

   iii. InstaSPIN_F2806xM_UNIVERSAL is for evaluating InstaSPIN-FOC or InstaSPIN-MOTION on the F28069M MCU.

   iv. These changes are contained in the .appsettings file

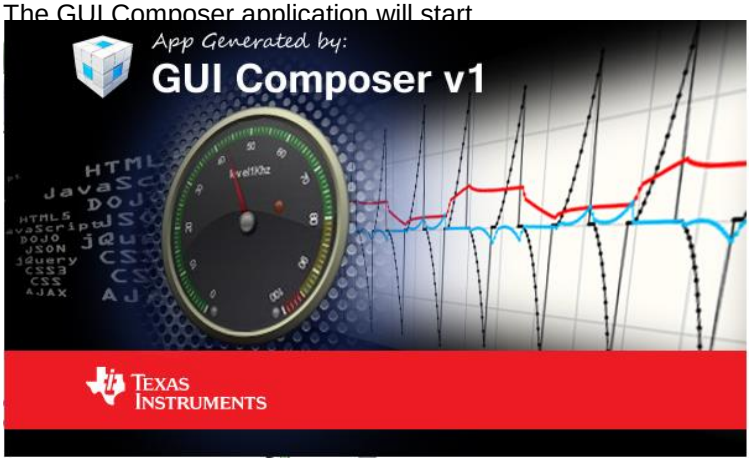
b. All other content in the folders is 100% identical

2. Rename the compiled MotorWare project (the binary): appProgram.out. Copy the file to the appropriate webapp folder.

a. The .appsettings file can be modified with a text editor to point at a specific compiled binary. Search for appProgram.out and replace it with the name of your binary.

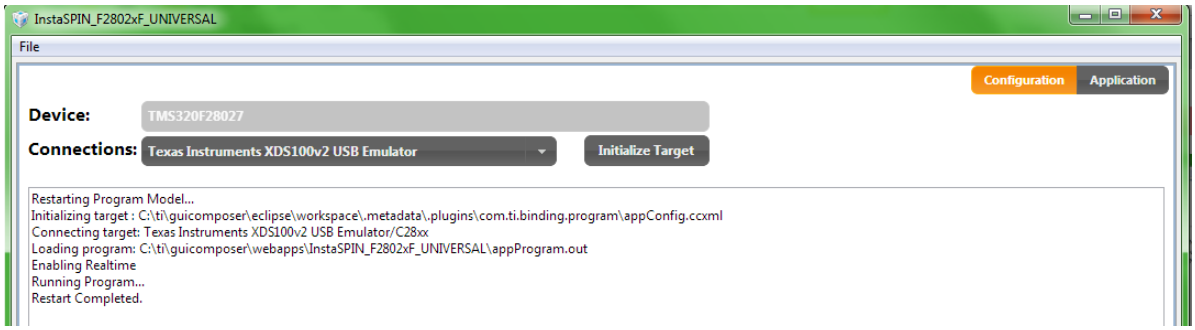
3. Run the executable from the webapp folder: **InstaSPIN_UNIVERSAL.exe**

4. The GUI Composer application will start



5. GUI Composer will initialize, then:
   a. Connect to the Piccolo device through the XDS100v2 emulator
   b. load \appProgram.out into the memory of the Piccolo device.
      i. The connection settings and binary name can be modified using a text editor by changing the .appsettings file.
   c. If using an emulator other than the XDS100v2, the connection will fail
      i. Permanently change the default emulator in the .appsettings file, or select the appropriate emulator from the drop-down menu and then click Initialize

6. The GUI should launch in less than 3 minutes
   a. If it takes longer to launch the GUI, disable any software on your PC that could be redirecting HTTP or browser sockets.



Configuration Tab of a successful GUI Launch

Option 2: Run the GUI in the CCStudio IDE

The InstaSPIN Universal GUI application and associated files can be found in C:\ti\guicomposer\webapps

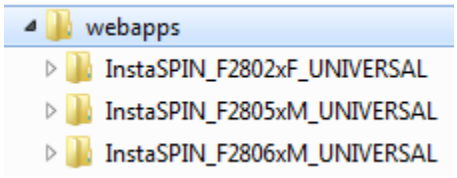
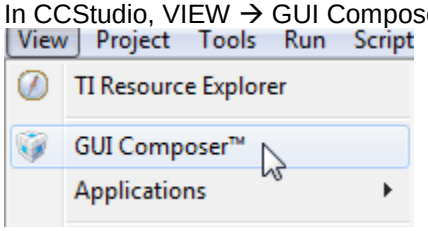


Figure 6 - Universal GUI .exe Directories

1. Zip one of the "webapps" folders, ex:

   Example: Create a .zip file for C:\ti\guicomposer\webapps\InstaSPIN_F2802xF_UNIVERSAL

2. In CCStudio, VIEW → GUI Composer



3. Select Import Project Icon (Figure 7), and point to the Zip file

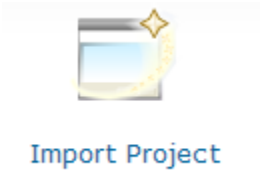


Figure 7 - CCS Project Icon

This will create a webapp folder in your CCS workspace

4. Select this workspace location each time you start CCStudio

   Example: C:\workspace\.GUIComposerWS\InstaSPIN_F2802xF_UNIVERSAL

5. Select the GUI by double clicking on the Project's app.html file (Figure 8):

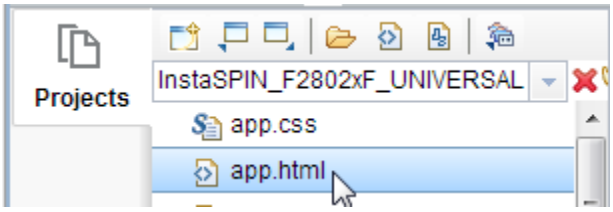
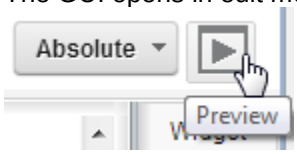
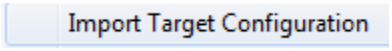
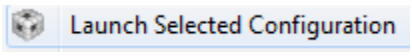


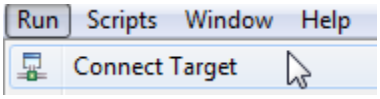
Figure 8 - Projects app.html file

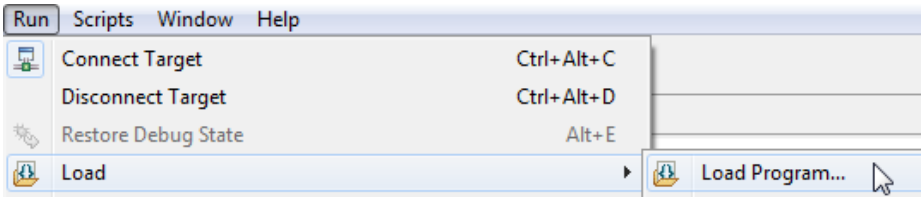
6. The GUI opens in edit mode. To interact with the GUI, select the Play arrow:

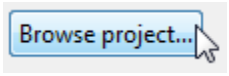


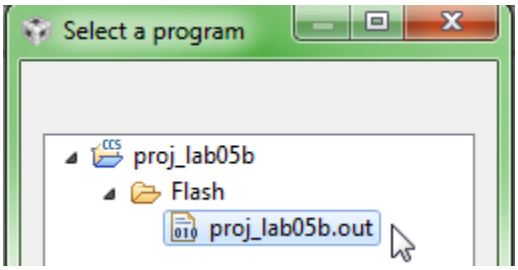
7. Follow these steps to use the GUI in CCS
   a. Import an emulation+cpu "Target Configuration"
      i. View → Target Configurations
      ii. Right Click → 

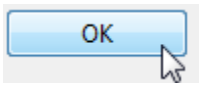
      iii. \sw\ide\ccs\ccs5\targetConfigs\TMS320F28027_xds100v2.ccxml
   b. Launch deubg session
      i. Right Click on the .ccxml file → 

   c. Connect target
      i. 

   d. Load compiled .out Program (or symbols if program is already in MCU flash)
      i. 

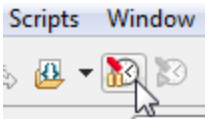
      ii. 

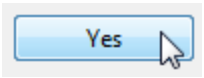
iii. 

iv. 

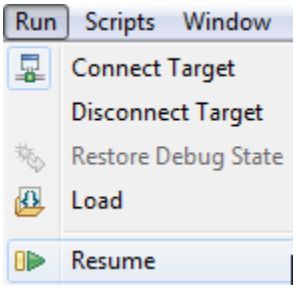
e. Note, steps a-d are accomplished automatically by using the “Debug” Icon as described in the InstaSPIN Projects & Labs User’s Guide

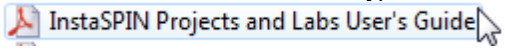
f. Enable silicon realtime mode

i. 

ii. If asked to enable realtime mode select 

g. Run → Resume

i. 

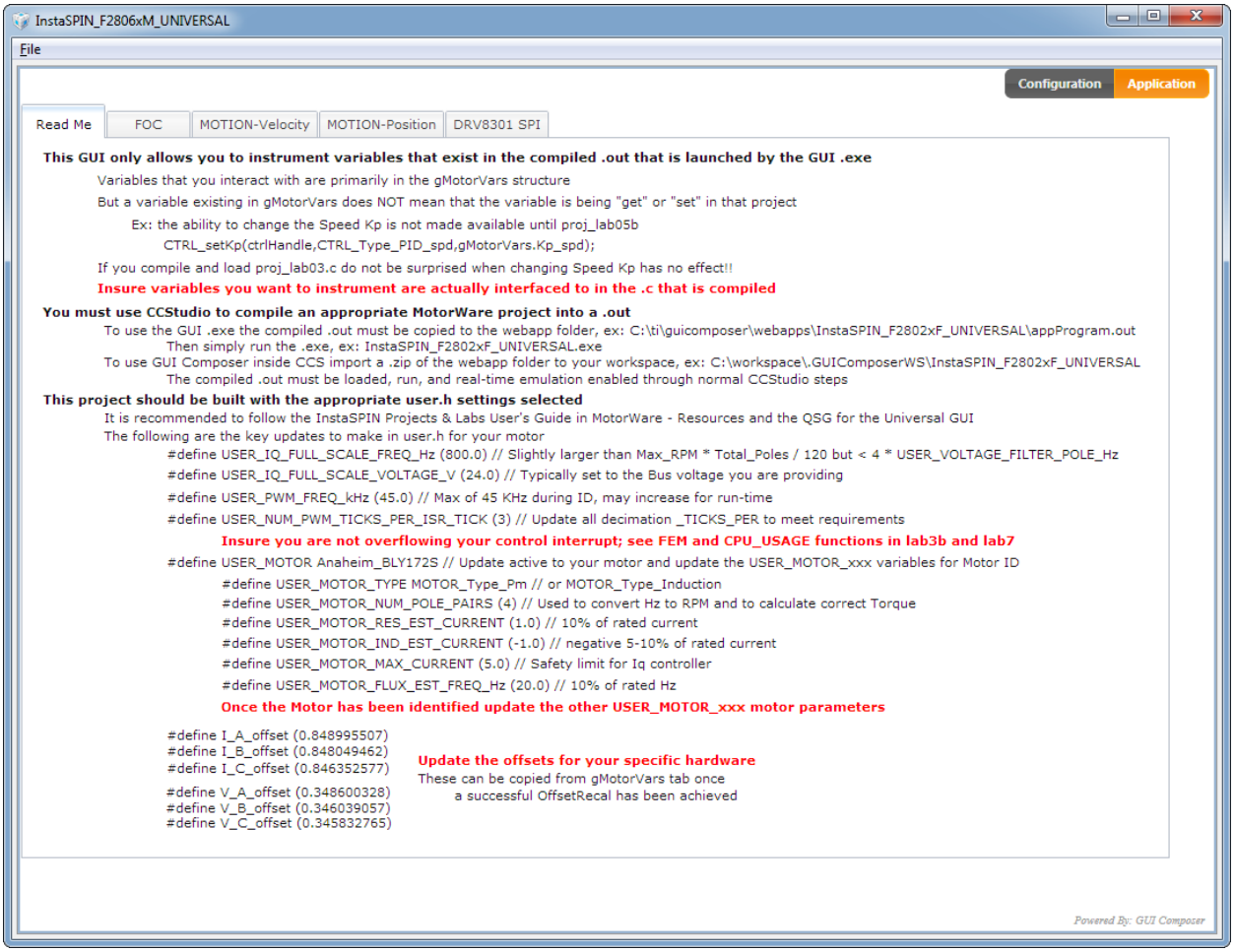
h. See the MotorWare InstaSPIN Projects and Labs User’s Guide (in the MotorWare\documents directory) for more details:


8. You can now use the GUI to instrument the code running on the MCU

Overview of the InstaSPIN Universal GUI

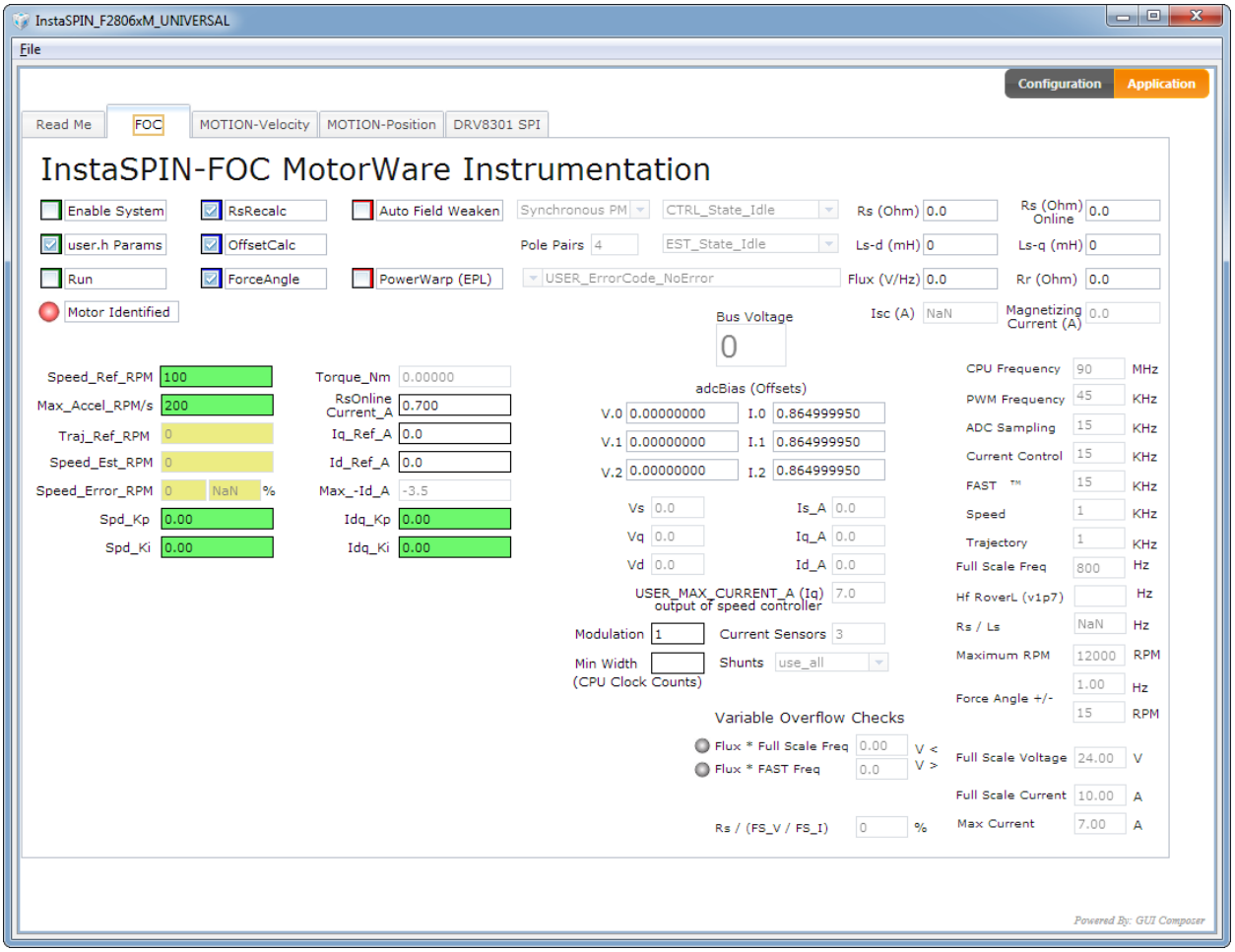
The InstaSPIN Universal GUI is made up of a series of tabs. Each Tab has a specific function.

Read Me



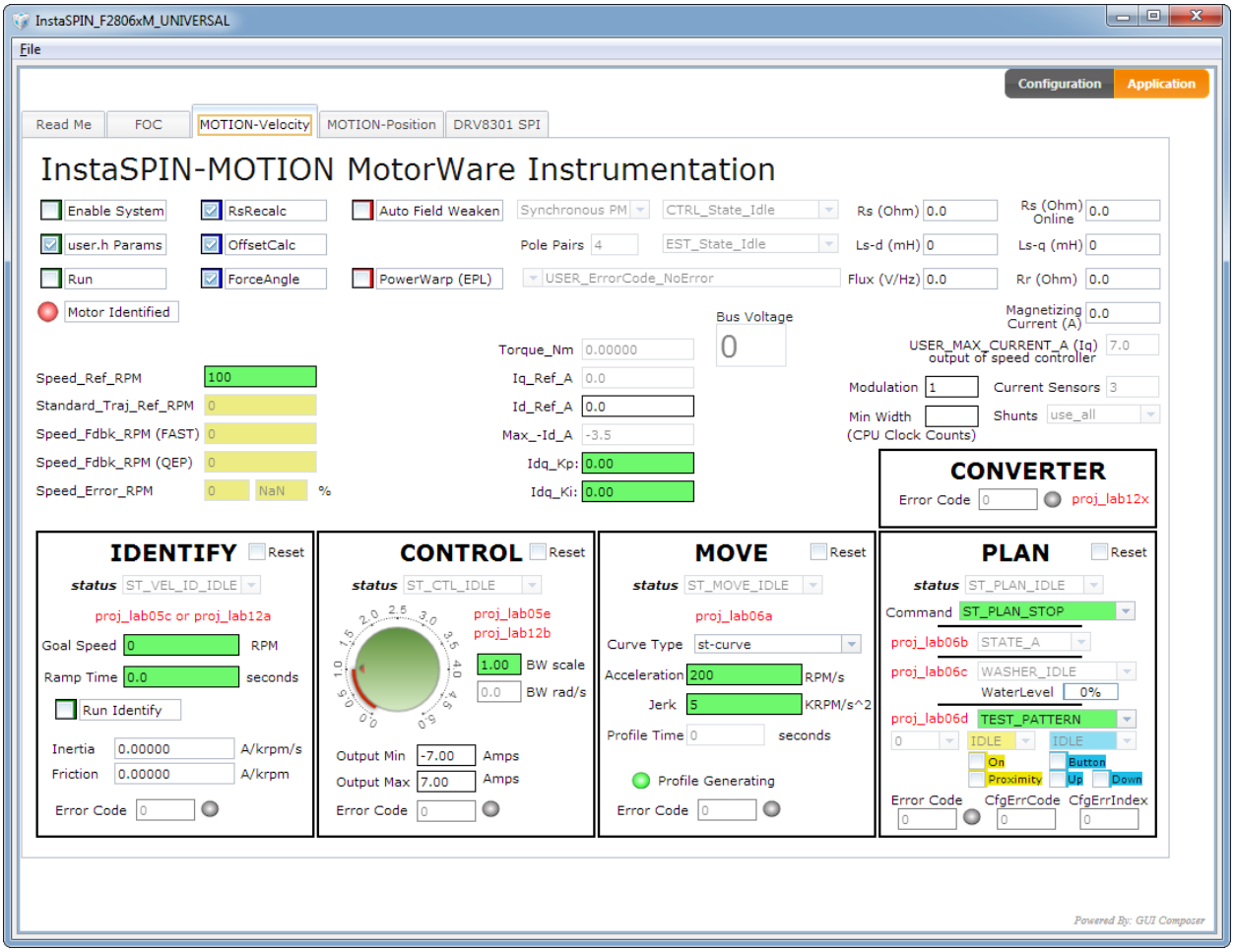
This tab is designed to provide instructions and general usage information about the GUI.

FOC



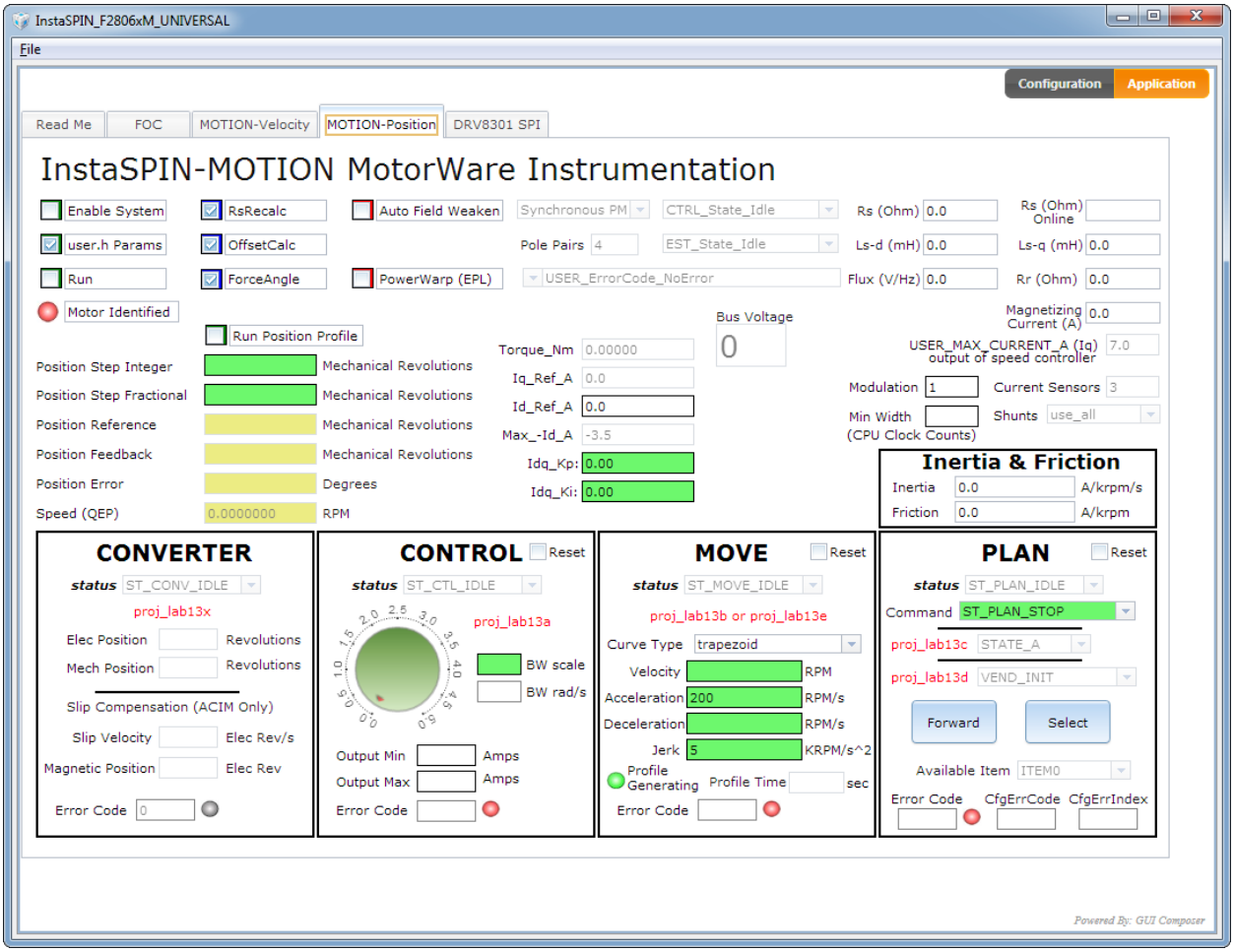
This tab instruments the InstaSPIN-FOC labs. See the InstaSPIN-FOC Use Example for information about the labs that are run from this tab.

MOTION-Velocity



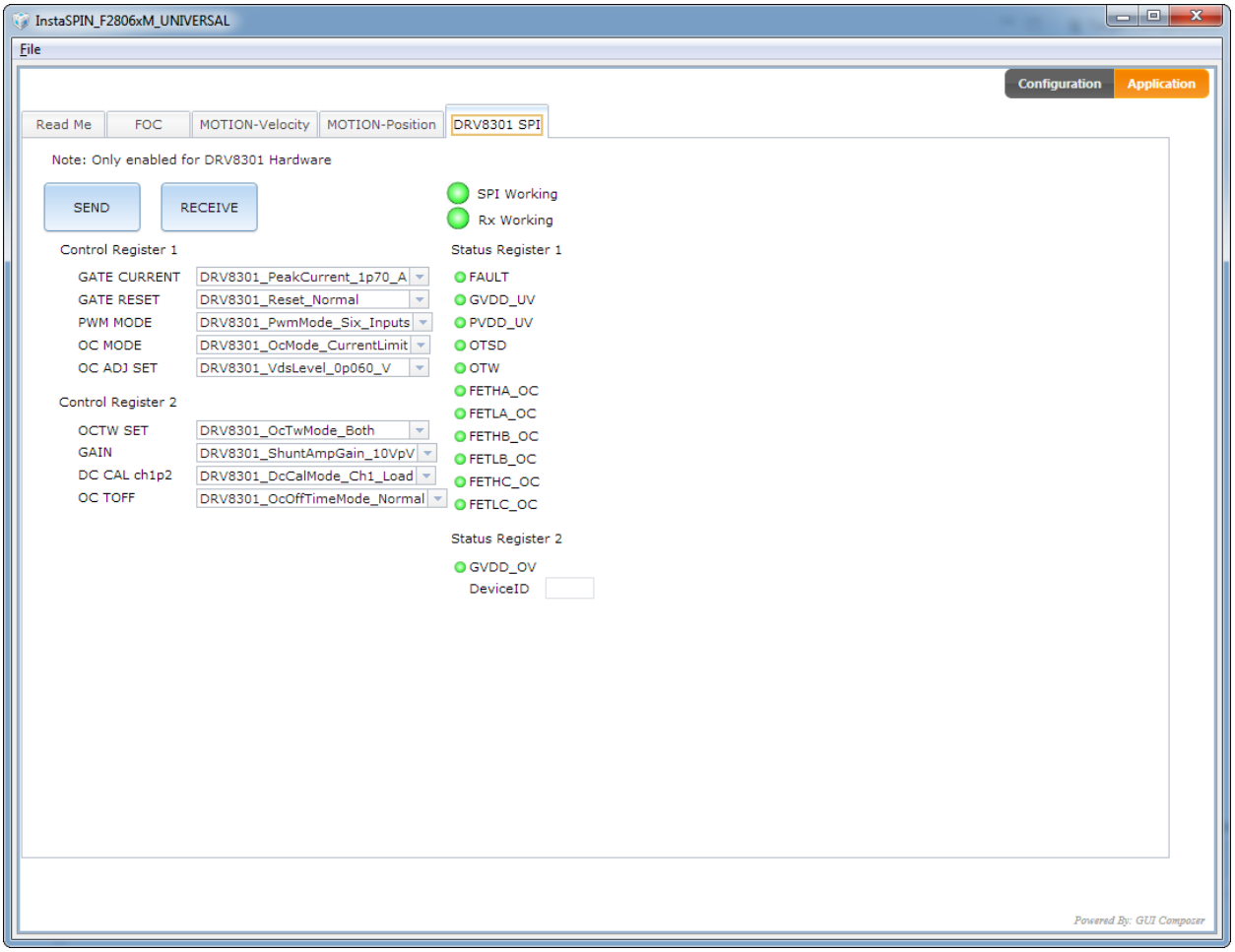
This tab instruments the InstaSPIN-MOTION velocity labs. This is used for both sensorless and sensored velocity labs. See the InstaSPIN-MOTION Use Example for information about the labs that are run from this tab.

MOTION-Position



This tab instruments the InstaSPIN-MOTION position labs. See the InstaSPIN-MOTION Use Example for information about the labs that are run from this tab.

DRV8301 SPI



This tab is used to modify the configuration of the DRV8301 gate driver, if necessary. It is used for the DRV8301-69M-kit and the BOOSTXL_DRV8301 kits only. The default values will work for most motors.

InstaSPIN-FOC Use Example

Controller: LAUNCHXL-F28027F

Inverter: BOOSTXL-DRV8301

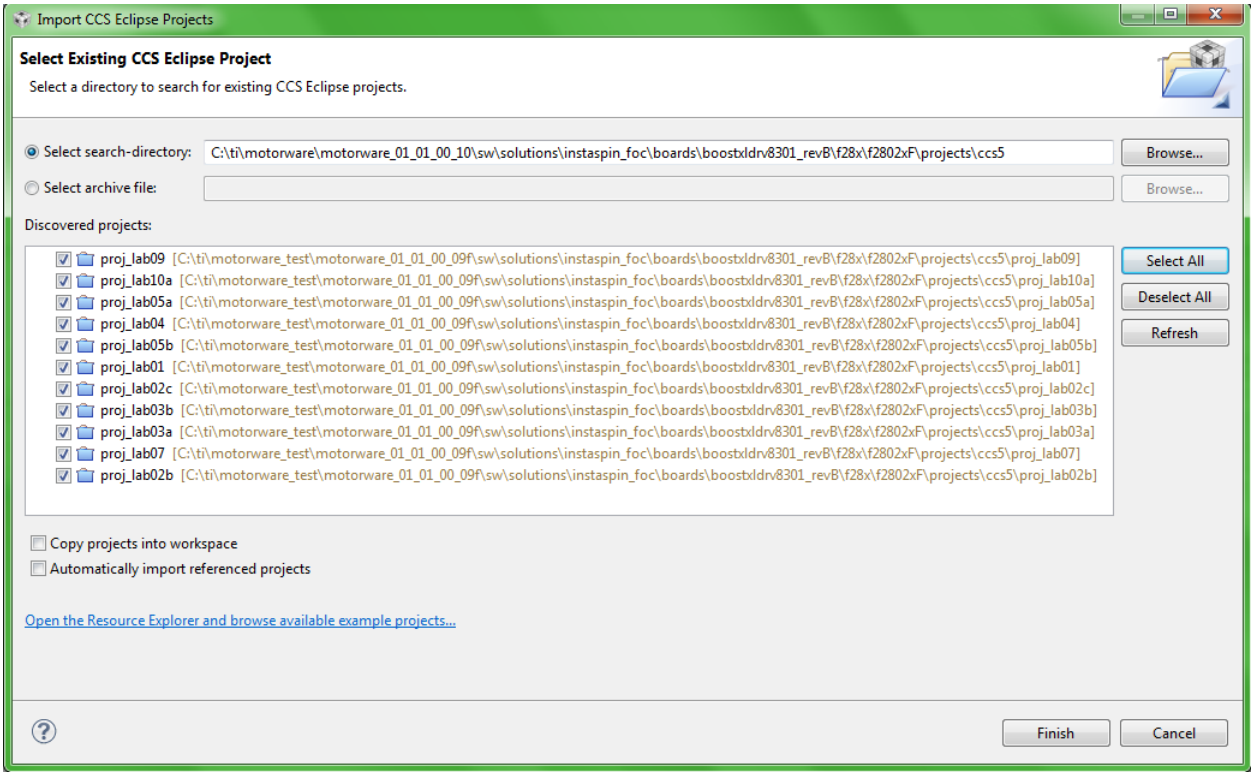
Motor: Anaheim Automation BLY172S-24V-4000: 24V, 8 poles, 4K RPM

Hardware Set-up

- Per the Kit Readme First and HW Guide Documents, set-up hardware
  - Example for LAUNCHXL-F28027F and BOOSTXL-DRV8301
  - LaunchPad
    - Removed JP1, 2, 3 so power can come from BoosterPack
    - S1 set to ON-ON-ON to allow JTAG
    - S4 set to OFF to allow LaunchPad to drive the BoosterPack Fault LEDs
  - BoosterPack
    - Motor phase wires connected (order only effects direction of motor)
    - DC power with appropriate 6-24V bus and up to 14A peak currents

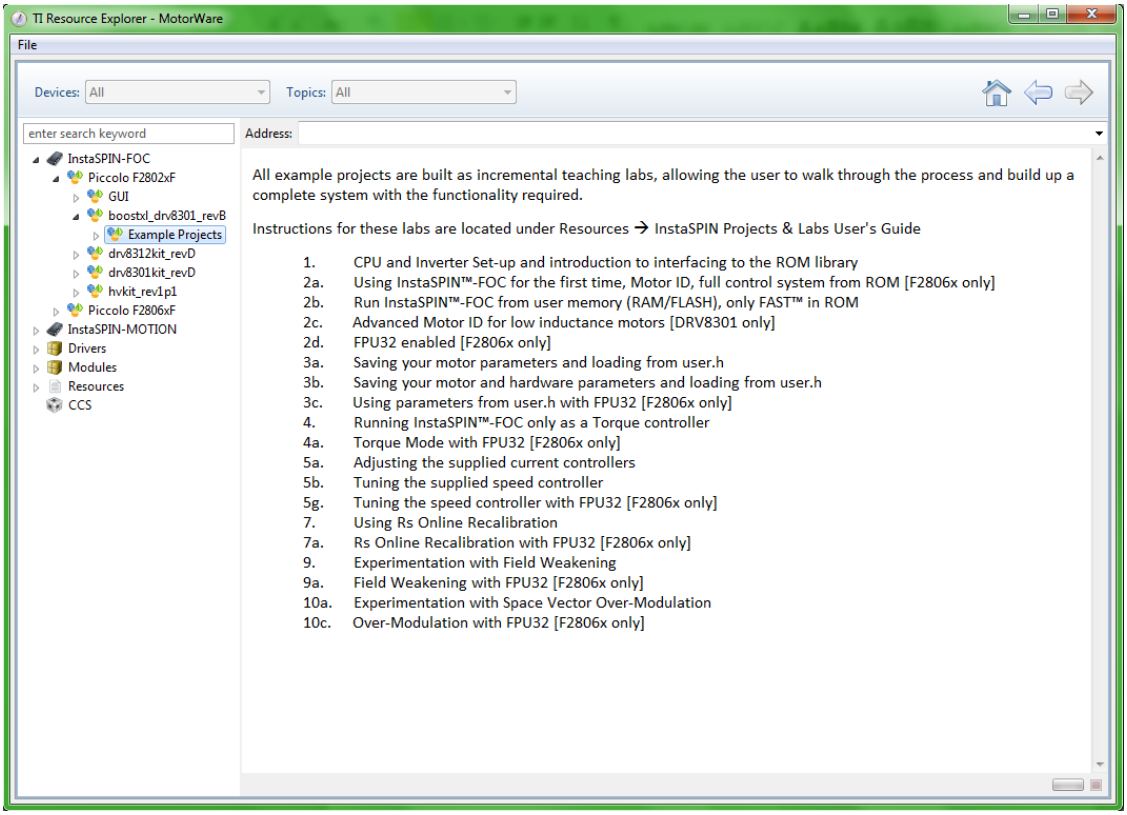
Software Projects

- In CCS, Projects → Import → and point to the latest version of the MotorWare "CCS5" directory for your combination of solution, board, and MCU
  Ex: InstaSPIN_FOC, the BoosterPack, F2802xF
  C:\ti\motorware\motorware_01_01_00_14\sw\solutions\instaspin_foc\boards\boostxldrv8301_revB\f28x\f2802xF\projects\ccs5



  - You may select any or all of the projects
    - Do NOT select to "Copy projects into workspace". The project will not find appropriate file references during build. We recommend working directly out of the C:\ti\motorware\ directory.
    - Recommended projects:
      - Lab2x – to identify your motor parameters
        - 2a or b – for standard motors
        - 2c – for high speed or low Ls or low flux motors
      - Lab5b – interface to tune the current and speed controllers
      - Lab9 – field weakening
      - Lab10a – over-modulation

- Lab5a – Torque control with PI tuning, no speed controller



Updating software for your motor (user.h)

- Open the associated src\user.h file in CCS Project Explorer or using a text editor C:\ti\motorware\motorware_01_01_00_14\sw\solutions\instaspin_foc \boards\boostxldrv8301_revB\f28x\f2802xF\src**user.h**
- **User.h can look overwhelming, but do not fear!**
  - **Most #define variables should not be modified and many are pre-compile calculations**
    - In future MotorWare versions the user.h file will be updated to display only the most relevant variables. The remaining variables will be accessible if required
  - Use the spreadsheet in MotorWare @ docs\labs\ motorware_selecting_user_variables.xlsx to help you update your user.h file.
- **Update these key user.h variables in each section**
  - **//! \brief CURRENTS AND VOLTAGES**
    - **#define USER_IQ_FULL_SCALE_FREQ_Hz (600.0)**
      - Set to the highest speed you want to run, where Hz = RPM * poles / 120
      - It is recommended to use a minimum of (500.0), even for low speed motors that may only run at 50 Hz
      - My example
        - 4 kRPM 8 pole motor = 267 Hz, but unloaded it will run faster and I expect to use field weakening to double my speed
        - **Use (600) Hz as my maximum**
    - **#define USER_IQ_FULL_SCALE_VOLTAGE_V (42.0)**
      - Typically the same as the Bus Voltage value
      - The GUI Variable Overflow Checks will set this value after the flux is identified
      - To maximize variable resolution for very low flux motors, set the value to as low as half of #define USER_ADC_FULL_SCALE_VOLTAGE_V (26.314)
      - This variable effects the SMALLEST flux value that can be identified
        - Ex: 0.002 V/Hz is a very small flux typically seen in high speed 12V hobby motors
        - Smallest flux = IQ_FULL_SCALE / Effective Estimation Frequency / 0.7
          - Effective Estimation Frequency is set in the DECIMATION section below and = PWM_FREQ /

PWM_TICKS_PER_ISR / ISR_TICKS_PER_CTRL / EST_TICKS_PER_CTRL

- Ex for 45V and 10 kHz effective estimation, smallest flux value = 0.0064 V/Hz
- To give some headroom we could solve for 0.0015 V/Hz * 0.7 / 15 kHz estimation = 15.75 V
- For Motor ID an IQ_FULL_SCALE value of (15) with estimation frequency of 15 KHz should work

- My example
  - **Using default of (42.0) to start, may update after ID**

- **//! \brief CLOCKS & TIMERS**
  - **#define USER_PWM_FREQ_kHz (30.0)**
    - Very low inductance, high short circuit currrent motors typically require 45-60 kHz
    - Most other motors will be in the 8-30 KHz range
    - Use lower PWM frequencies when possible to reduce switchign losses
    - My example
      - **Using default of (30.0) KHz**
- **//! \brief DECIMATION**
  - Determines rates of the control loops and effects interrupt loading
  - ISR = PWM_FREQ / **USER_NUM_PWM_TICKS_PER_ISR_TICK**
    - Uses ePWM 1st/2nd/3rd event hardware to trigger the ADC start of conversion; the ADC conversion done interrupt acts as the main ISR for the control system
    - Best if ISR is <= 15 KHz maximum, and <= 10 KHz typical
      - If not possible be sure to test using FEM and CPU_USAGE functions using lab3b
  - CTRL = ISR / **ISR_TICKS_PER_CTRL**
    - Insure CTRL is effective <=15 kHz
      - Be careful of 15 KHz rates on 60 MHz MCUs, you may over flow the interrupt as more system code is added, especially during the motor identification process
      - If unsure, test using FEM and CPU_USAGE functions using lab3b
    - CURRENT = CTRL / **CTRL_TICKS_PER_CURRENT**

- 5-15 KHz typical
- EST = CTRL / **CTRL_TICKS_PER_EST**
  - Whole divisor of CURRENT, 2.5-15 KHz typical
- SPEED = CTRL / **CTRL_TICKS_PER_SPEED**
  - 1 KHz typical
- TRAJ = CTRL / **CTRL_TICKS_PER_TRAJ**
  - Same as speed, 1 KHz typical

- My example
  - **ISR (3) to get effective 30 kHz / 3 = 10 kHz ISR**
  - **CTRL (1) for 10 kHz**
  - **EST (1) and CURRENT (1) for 10 kHz**
  - **TRAJ (10) and SPEED (10) for 1 kHz**

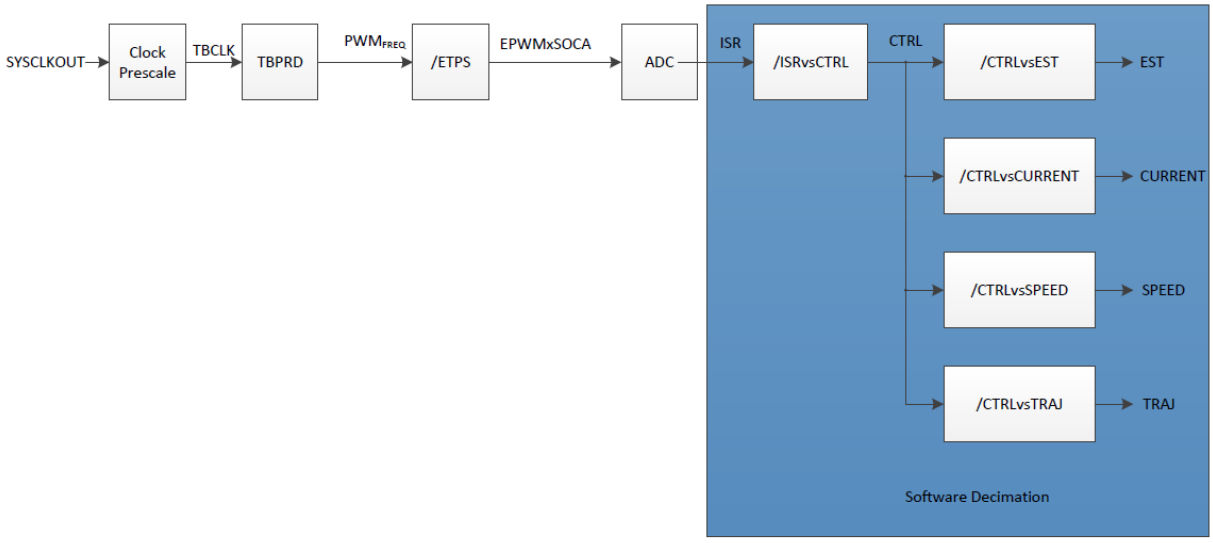


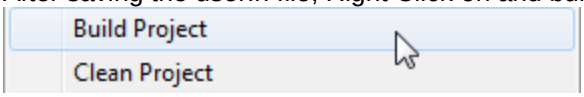
Figure 9-3. Real-Time Scheduling Tick Rates

- **//! \brief USER MOTOR & ID SETTINGS**
  - Each motor saved must have a unique enumeration
    - #define Anaheim_BLY172S 102
    - **#define My_Motor 103**
  - **Comment out (//) all but one USER_MOTOR selection**
    - //#define USER_MOTOR Anaheim_BLY172S
    - **#define USER_MOTOR My_Motor**

- Set-up your motor
  - #elif (USER_MOTOR == My_Motor)
  - **#define USER_MOTOR_TYPE** **MOTOR_Type_Pm**
  - **#define USER_MOTOR_NUM_POLE_PAIRS** **(4)**
    - if # of poles is incorrect it only effects the relationship between the RPM command (which is coverted to a Hz command based on the POLE_PAIRS) as well as the Torque calculation; there is no effect on the quality of control
    - If you don't know your pole pair number exactly you can take a guess to start, and once motor is identified and running you can command a multiple of 60 RPM to see if the motor is making the expected number of revolutions per second (60 RPM = 1 rev/sec; 180 RPM = 3 rev/sec, etc)
      - faster than 1 rev/sec then poles should be reduced
      - slower than 1 rev/sec then poles should be increased
  - #define USER_MOTOR_Rr (NULL)
    - ID'd for ACI motors only
  - #define USER_MOTOR_MAGNETIZING_CURRENT (NULL)
    - ID'd for ACI motors only
  - **#define USER_MOTOR_Rs** **(NULL)**
    - Update after Motor ID
  - **#define USER_MOTOR_Ls_d** **(NULL)**
    - Update after Motor ID
  - **#define USER_MOTOR_Ls_q** **(NULL)**
    - Update after Motor ID
    - Set same as Ls_d unless different Ls_d and Ls_q are known by design; FAST can compensate for this saliency
  - **#define USER_MOTOR_RATED_FLUX** **(NULL)**
    - Update after Motor ID
  - **#define USER_MOTOR_RES_EST_CURRENT** **(0.4)**
    - ~10% of rated current
    - Used to ineject current for Rs test AND to start-up motor for EST_STATE Ramp_Up. For high cogging torque motors increase RES_EST_CURRENT until the motor spins during Ramp_Up

- **#define USER_MOTOR_IND_EST_CURRENT (-0.4)**
  - NEGATIVE ~10% of rated current
  - Used to weaken the field during Ls testing
  - If having trouble identifying a correct Ls with a large RES_EST setting required for a high cogging motor, reduce this value for Ls estimation stability. Ex: (5.0) and (-3.0)
- **#define USER_MOTOR_MAX_CURRENT (4.0)**
  - Rated peak current of motor required to produce maximum torque. This is HIGHER than the rated current of the motor.
  - This is maximum Iq torque command produced by the Speed controller (PI or SpinTAC), IF using the speed controller
- **#define USER_MOTOR_FLUX_EST_FREQ_Hz (30.0)**
  - ~10% of rated max speed = 267 Hz * 10%
  - Important to that this is high enough as we measure the voltage produced by flux (V/Hz) which will be larger at higher Hz, especialy for low flux (high speed) motors
  - Also note resolution comments in the USER_IQ_FULL_SCALE_VOLTAGE_V section

- After saving the user.h file, Right Click on and build the appropriate project



Using the GUI

- For Standalone Mode, follow the instructions in the section above:

  Running the GUI Standalone GUI .exe

  - Copy the .out just built to the appropriate Webapp folder, and rename to "appProgram.out" ex:

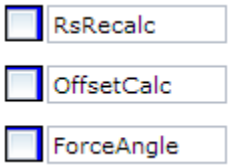
    …\sw\solutions\instaspin_foc\boards\boostxldrv8301_revB\f28x\f2802xF\projects\ccs5\proj_lab05b\Flash\proj_lab05b.out

    to the appropriate webapp folder:

    C:\ti\guicomposer\webapps\InstaSPIN_F2802xF_UNIVERSAL\appProgram.out

  - Run the executable C:\ti\guicomposer\webapps\InstaSPIN_F2802xF_UNIVERSAL\InstaSPIN_UNIVERSAL.exe

Start-up Options



RsRecalc

If ENABLED - when Run is enabled - Performs a recalculation of Rs.

If DISABLED – when Run is first enabled – uses the most recent value of Rs.

Rs accuracy is critical for low speed operation. Rs changes as the motor windings heat from high current usage = high torque demanded from the motor. Loaded washing agitation is a good example. See the Rs On-line (always running recalibration) feature in MotorWare Lab7 if required.

OffsetCalc

If ENABLED – when Run is enabled - performs a recalculation of the ADC offsets.

Length of this recalibration is adjustable, see the User's Guide.

Values can be saved and loaded from user.h, bypassing this calculation in the future.

Important Notes:

- **If the USER_IQ_FULL_SCALE_VOLTAGE or CURRENT values are changed the saved offsets must be changed as well.**
- **If DISABLED the offset values / adcBias will be loaded from user.h settings only. Only ever DISABLE if the real values are in user.h!!!**

ForceAngle

Force Angle can be thought of as trajectory generation for the angle feedback (replacing FAST over a user set area) to the FOC controller. It creates an estimated angle that rotates at a user set rate (in user.h).

It should be used whenever FAST is not producing an accurate angle estimate, i.e. at initial start-up and if trying to operate continuously at very low speeds.

It should typically be ENABLED when first starting

- though it doesn't have to be, FAST can still start up the motor, but usually not as gracefully

and then DISABLED for normal operation, unless

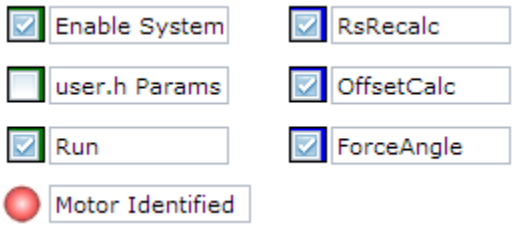
- you have times where you run at or through very low speeds for long enough that the FAST estimator drifts and provides poor estimates into the system
    - in which case the ForceAngle can help you move out of this area and into a speed where FAST is providing good feedback

While application/motor/sense/acceleration dependent, once the motor is running you will often continue to track through zero speed well enough (depending on Bemf and deceleration rate).

Motor ID

To perform a Motor Identification:

- SELECT Enable System
- user.h Params is NOT selected
- Start-up Options
    - Recommend keeping OffsetCalc, RsRecal and ForceAngle selected
- SELECT Run



user.h Params

If user.h Params is enabled when Run is selected, the control system will bypass Motor ID and

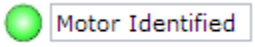
- Load all settings from user.h, including Offsets and all USER_MOTOR settings
- Do not select until you have updated the user.h fully

- Exception: In the proj_lab2x labs this selection has no effect, project will never load from user.h and motor ID will always be performed

Motor Identification States

This is fully covered in the User's Guide Chapter 6. Following is the example for the PM Synchronous Motor.

- OffsetRecalc is performed before the Motor ID process begins
- EST_State_
  - RoverL
    - Injects ½ of **USER_MOTOR_RES_EST_CURRENT** at **USER_R_OVER_L_EST_FREQ_Hz**
  - Rs
    - Injects **USER_MOTOR_RES_EST_CURRENT**
  - RampUp
    - Starts motor using current amplitude of **USER_MOTOR_RES_EST_CURRENT** at a rate of **USER_MAX_ACCEL_EST_Hzps** until speed of **USER_MOTOR_FLUX_EST_FREQ_Hz**
    - **Motor must continue spinning – at the RPM of USER_MOTOR_FLUX_EST_FREQ_Hz -from this point until Idle or ID results should be considered invalid**
  - RatedFlux
    - Current is minimized while keeping speed to detect Flux
  - Ls
    - Injects **USER_MOTOR_IND_EST_CURRENT** into Id (field weakening) to detect the inductance
  - RampDown
    - ID process ends and motor slows to 0 speed
  - Idle
- While you should insure that the Motor Identified light turns green, this does NOT mean that the identified parameters are correct, just that the identification state finished without a serious error
  - You can start ID without a motor even attached and still get a green light. This only alerts you to very specific and serious errors



Motor ID Tips

- **Scaling of hardware Vph and USER_IQ_FULL_SCALE_VOLTAGE is critical, especially for low inductance (high speed motors)**
- Note that during Motor ID there are wait time time-outs established for each step. These times may need to be increased for the RampUp especially if you increase the ID speed or decrease the Estimation Frequency
    - See “user.c”
- RoverL
    - If RoverL is >= 2000 there is a variable overflow, so you MUST use lab2c to attempt ID (this happens with low Ls / high speed motors)
        - RoverL is provided in the panel for F2805x & F2802x devices which use the variable ctrl.RoverL
        - RoverL is not provided for F2806x as it uses a different variable, found under controller_obj and then Rover.L
    - **USER_R_OVER_L_EST_FREQ_Hz** is (300) in the DRV83x projects for 1500 Hz high speed motors. For < 1 KHz motors reduce to (100) and for > 1 KHz change to (200).
- Rs
    - Be sure to use 10% of I-rated for **USER_MOTOR_RES_EST_CURRENT**
    - important not to overheat the motor with too high a current
    - Only increase in small increments if required to start the motor spinning for the Ramp_Up Estimation State (see below)
- Ramp_Up
    - High Cogging Torque motors may not start-up
        - Increase **USER_MOTOR_RES_EST_CURRENT** in small increments until it does (0.2 or 0.5 A depending on scale of max current)
    - When **USER_MOTOR_FLUX_EST_FREQ_Hz** is increased (see Ls) the Ramp_Up may time out
        - Increase **USER_MAX_ACCEL_EST_Hzps** to hit the target speed before time-out
        - Or increase the wait time in user.c pUserParams->estWaitTime[EST_State_RampUp] = (uint_least32_t)((**5.0** +
- Rated_Flux
    - If the motor stops spinning increase **USER_MOTOR_FLUX_EST_FREQ_Hz**
    - The smallest flux that can be ID’d = **USER_IQ_FULL_SCALE_VOLTAGE_V** / FAST_EST_Hz / 0.7

- For small flux machines (small motors, high speed) lower **USER_IQ_FULL_SCALE_VOLTAGE_V** to increase the resolution

- Ls
  - If the motor stops spinning or the Ls values are not stable and consistent increase **USER_MOTOR_FLUX_EST_FREQ_Hz**
  - Possibly change PWM frequency to 15-60 KHz, and use lab2c
  - Note that any Ls value < 1e-6 H should be considered invalid and incorrect

Values Returned for this example

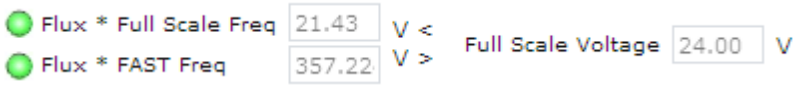
Rs (Ohm)	0.4110606	Rs (Ohm) Online	0.0
Ls-d (mH)	0.6587001	Ls-q (mH)	0.6587001
Flux (V/Hz)	0.0357224	Rr (Ohm)	0.0
Isc (A)	8.6356372	Magnetizing Current (A)	0.0

Motor ID Sanity Checks

- Rs / Ls — Rs / Ls 624.04 Hz
  - R/L gives a theoretical limitation of speed with a stable voltage source
    - **Note: the GUI displays Ls in mH, not H**
  - ctrl.RoverL (or controller_obj) uses the initial high frequency signals and will be different
  - Is this larger than your MAX_Hz? Does it seem reasonable for your motor?
    - The 2.5x my rated max frequency is reasonable
    - Note that high speed motors are often mis-designed with very low Ls, resulting in Rs / Ls much larger than MAX_Hz

- Flux / 2pi / Ls = Short Circuit Current = Isc — Isc (A) 8.6356372
  - Typically 2x+ the rated current and often much larger for small Ls or large Flux machines
  - Large Isc = low Ls = high speed, high current
  - Larger Isc may require faster PWM (30-60 KHz) and possibly faster current control (15 KHz)

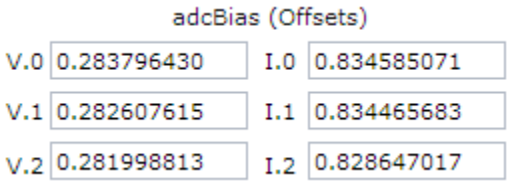
- Overflow / Resolution Checks, adjustment of IQ_VOLTAGE



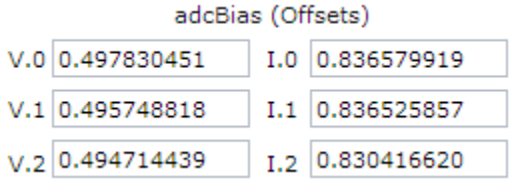
  - Minimize Full Scale Voltage vs. Full Scale Frequency to maximize resolution
    - This does a check using the IQ_FULL_SCALE_FREQUENCY which may be much higher than you actually plan to run your motor. Do a sanity check using the actual maximum frequency you plan to run (including any field weakening)
    - For High flux motors this often means increasing the IQ_FULL_SCALE_VOLTAGE to support the Bemf voltage at highest speeds
      - For example a 1.2 V/Hz motor will produce 1200V at 1KHz!
    - For Low flux motors this often means reducing the IQ_FULL_SCALE_VOLTAGE to improve your resolution (still bounded by the actual hardware scaling of USER_ADC_FULL_SCALE_VOLTAGE_V for actual resolution)
      - For my example, I will update user.h to **(24.0)**
  - FAST Frequency vs. Full Scale Voltage
    - Alerts that low flux motors should use a smaller IQ_FULL_SCALE_VOLTAGE to increase resolution or run the FAST EST_FREQ at a higher rate

Update user.h settings

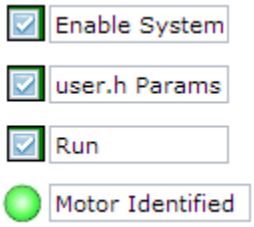
adcBias (Offsets) for Example Hardware



- Copy and paste the adcBias 0/1/2 values to the I_A/B/C_offset and V_A/B/C_offset #defines
  - Note, since I chose to change my IQ_VOLTAGE from (42.0) to **(24.0)** in previous section these offsets are no longer valid as they change with your IQ scaling.
  - If you change IQ_VOLTAGE or CURRENT, recompile, run, and update the adcBias in user.h



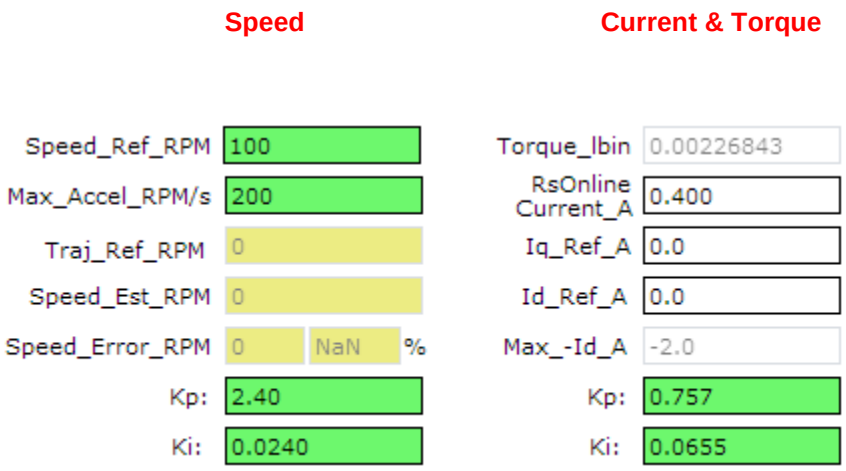
- Copy and paste the identified motor parameters to the USER_MOTOR #defines
  - **Note: The GUI displays inductance in mH while the user.h is in H.**
  - **Be sure to paste the GUI Ls_d / Ls_q value with a leading 0.000**
- Recompile and use the .out with your GUI if you want to be able to skip Motor ID in the future (enable the user.h Params in any project_lab03+ before you select Run)



Controller Tuning

InstaSPIN-FOC includes a PI controller for current and speed. To evaluate the InstaSPIN-MOTION speed and position control features, skip to the next section.

Default Controller Tuning



Note that both current and speed controller gains are effected by the IQ scaling variables and decimation timing, they will need to be changed as well if you update these user.h settings.

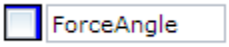
Current Controllers

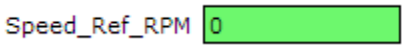
- The Iq and Id Current PI controller gains are numerically calculated & initialized
  - Kp = ¼ * Bandwidth
    - Bandwidth = [Ls / CTRL_FREQ_Hz * IQ_CURRENT / IQ_VOLTAGE]
    - This ¼ factor is to soften the controller a bit for better stability
  - Ki = CTRL_FREQ_Hz / Ls * Rs
- These can be changed simply through the GUI, which instruments the following user code
  - gMotorVars.Kp_Idq = CTRL_getKp(ctrlHandle,CTRL_Type_PID_Id);
  - CTRL_setKp(handle,CTRL_Type_PID_Id,gMotorVars.Kp_Idq);
- Some applications may want to do step response testing to meet desired response of over/undershoot and settling time
- **Note, current controllers can only be updated starting in proj_lab5a, previous to this any changes to those variables in the GUI will have no effect**

Speed Controller

- The PI Speed Control **cannot be auto tuned** based on the motor or system parameters

- Speed control relies on knowledge of inertia, mechanical linkages, and desired response
- Speed Gains are initialized using a "rule of thumb", which works decently for larger flux motors
  - Kp = 0.02 * MAX_HZ * MAX_CURRENT / IQ_CURRENT
  - Ki = 2.0 * CTRL_HZ * MAX_HZ * MAX_CURRENT / IQ_CURRENT
  - Experience shows that for low inertia motors a good starting point is to reduce the default Kp and Ki by /4 to /10. High inertia motors may require gains 4-10x larger.

- Tuning
  - Tune by testing various speeds and loads or tune by step response inputs (most popular)
  - May need to "gain stage", different KpKi sets for different speeds/loads/accelerations
  - May be able to empirically calculate if you know inertia (see Labs/UG)
  - Zero Speed tuning & experiment
    - Disable ForceAngle 

    - Set 0 speed 

    - Quickly rotate the motor shaft 90-180 deg and then let go
      - Now set Speed Kp to 0.2, Ki to 0.004
        - example for Anaheim motor under test
        - Notice how the motor shaft behaves like a spring-damper system, "compressing" as you turn and then "returning" once you remove the load
      - Increase Kp until the spring feeling is gone
      - Increase Ki to increase the stiffness of the motor
      - At this point the system might be slightly unstable, the following can help stabilize the system:
        - Increase Kp to increase the dampening
        - Reduce Ki to reduce oscillations

- These can be changed simply through the GUI, which instruments the following user code
  - gMotorVars.Kp_spd = CTRL_getKp(ctrlHandle,CTRL_Type_PID_spd);
  - CTRL_setKp(handle,CTRL_Type_PID_spd,gMotorVars.Kp_spd);
- **Note, speed controllers can only be updated starting in proj_lab5b, previous to this any changes to those variables in the GUI will have no effect**

InstaSPIN-MOTION Use Example

Controller: LAUNCHXL-F28069M

Inverter: BOOSTXL-DRV8301

Motor: Teknic M2310P-LN-04K: 24V, 8 poles, 4 krpm, 1000 line encoder

Hardware Set-up

- Set up the hardware according to the Kit Readme First file and HW Guide Documents
  - Example for LAUNCHXL-F28069M and BOOSTXL-DRV8301
    - BOOSTXL-DRV8301 Motor phase wires connected
      - For sensored projects the motor phase order is important and the incremental encoder must be connected
    - DC power with appropriate 24V bus and peak current according to motor

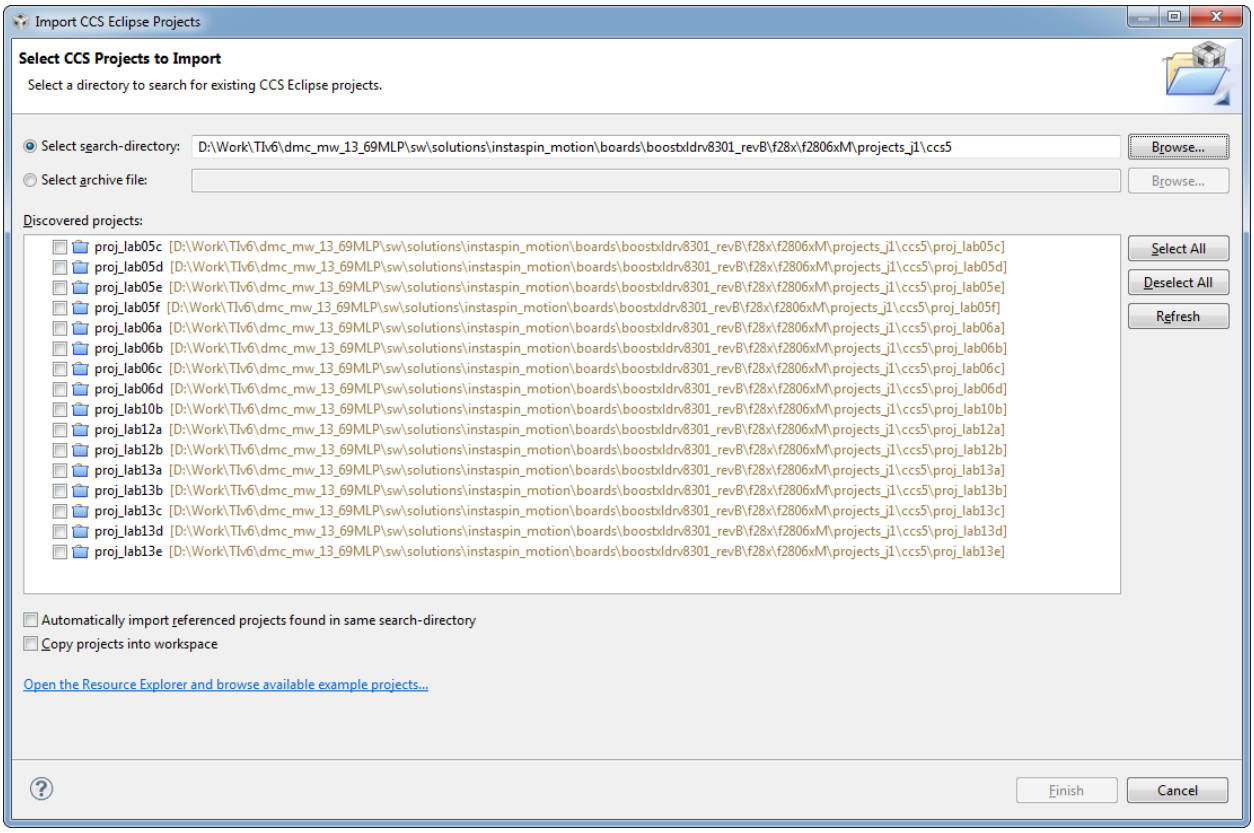
Software Projects

Import the lab projects

- In CCS, select:

  Projects → Import → and point to the latest version of the MotorWare "CCS5" directory for your combination of solution, board, and MCU

  Example: InstaSPIN_MOTION, the BOOSTXL-DRV8301, F2806xM
  C:\ti\motorware\motorware_01_01_00_14\sw\solutions\instaspin_motion\boards\boostxldrv8301_revB\f28x\f2806xM\projects_j1\ccs5

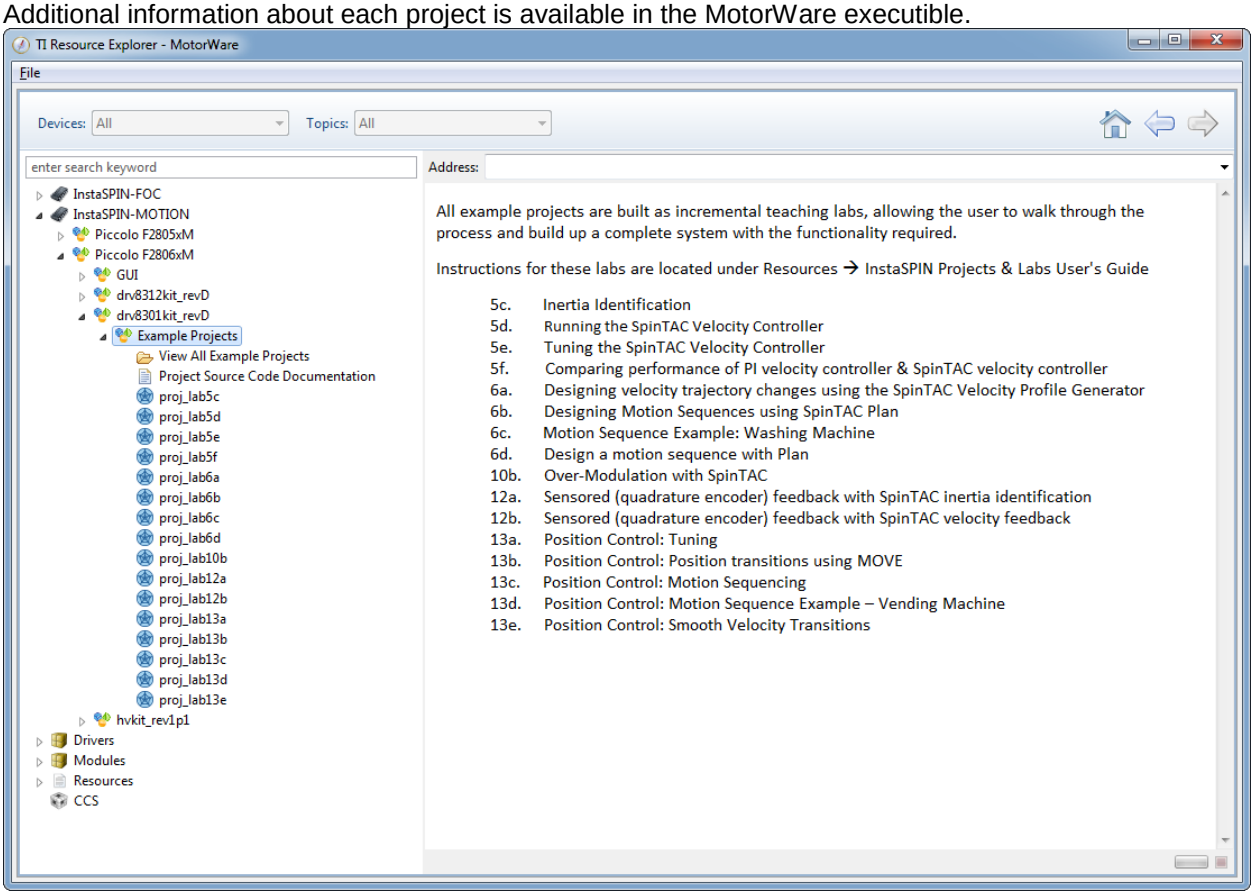


  - You may select any or all of the projects
    - Make sure you do NOT select to "Copy projects into workspace" or the project will not find appropriate file references during build. We recommend working directly out of the C:\ti\motorware\ directory.
    - For sensorless velocity control, import the following labs
      - Lab5c –identify system inertia and friction
      - Lab5e – interface to tune the current and speed controllers
      - Lab6a – advanced speed trajectory generation

- Lab10b – field weaking and over-modulation

- For sensored velocity control, import the following labs
  - Lab12a – identify system inertia and friction
  - Lab12b – interface to tune the current and speed controllers
- For position control, import the following labs
  - Lab12a – identify system inertia and friction
  - Lab13a – interface to tun the current and speed+position controllers
  - Lab13b – position transition

Additional information about each project is available in the MotorWare executible.



Updating software for your motor (user.h)

The motor identification process for InstaSPIN-MOTION is identical to the motor identification process for InstaSPIN-FOC. Identify the motor parameters using InstaSPIN-FOC Lab2a or Lab2c.

- In CCS, select:

  Projects → Import → and point to the latest version of the MotorWare "CCS5" directory for your combination of solution, board, and MCU

  Example: BOOSTXL-DRV8301, F2806xM, Teknic Motor
  C:\ti\motorware\motorware_01_01_00_14\sw\solutions\instaspin_foc\boards\boostxldrv8310_revB\f28x\f2806xM\projects_j1\ccs5

Follow the instructions in InstaSPIN-FOC Use Example Motor ID **to obtain the motor parameters**

After running lab 2a or 2c, copy the motor parameters from the InstaSPIN-FOC user.h file to the InstaSPIN-MOTION user.h file

- Open the user.h file that was modified as part of InstaSPIN-FOC lab 2a. It is located in "sw\solutions\instaspin_foc\boards\boostxldrv8301_revB\f28x\f2806xF\src"
- Locate the USER_MOTOR settings identified in lab 02a (or in lab 2c). It should be similar to the following:

```
396 #elif (USER_MOTOR == Teknic_M2310PLN04K)
397 #define USER_MOTOR_TYPE                 MOTOR_Type_Pm
398 #define USER_MOTOR_NUM_POLE_PAIRS       (4)
399 #define USER_MOTOR_Rr                   (NULL)
400 #define USER_MOTOR_Rs                   (0.3918252)
401 #define USER_MOTOR_Ls_d                 (0.00023495)
402 #define USER_MOTOR_Ls_q                 (0.00023495)
403 #define USER_MOTOR_RATED_FLUX           (0.03955824)
404 #define USER_MOTOR_MAGNETIZING_CURRENT  (NULL)
405 #define USER_MOTOR_RES_EST_CURRENT      (1.0)
406 #define USER_MOTOR_IND_EST_CURRENT      (-0.5)
407 #define USER_MOTOR_MAX_CURRENT          (7.0)
408 #define USER_MOTOR_FLUX_EST_FREQ_Hz     (20.0)
```

- Open the user.h file for InstaSPIN-MOTION. It is located in "sw\solutions\instaspin_motion\boards\boostxldrv8310_revB\f28x\f2806xM\src"

- Copy the USER_MOTOR settings from the InstaSPIN-FOC user.h into the InstaSPIN-MOTION user.h, The new entry should be similar to the following:

```
396 #elif (USER_MOTOR == Teknic_M2310PLN04K)
397 #define USER_MOTOR_TYPE                 MOTOR_Type_Pm
398 #define USER_MOTOR_NUM_POLE_PAIRS       (4)
399 #define USER_MOTOR_Rr                   (NULL)
400 #define USER_MOTOR_Rs                   (0.3918252)
401 #define USER_MOTOR_Ls_d                 (0.00023495)
402 #define USER_MOTOR_Ls_q                 (0.00023495)
403 #define USER_MOTOR_RATED_FLUX           (0.03955824)
404 #define USER_MOTOR_MAGNETIZING_CURRENT  (NULL)
405 #define USER_MOTOR_RES_EST_CURRENT      (1.0)
406 #define USER_MOTOR_IND_EST_CURRENT      (-0.5)
407 #define USER_MOTOR_MAX_CURRENT          (7.0)
408 #define USER_MOTOR_FLUX_EST_FREQ_Hz     (20.0)
409 #define USER_MOTOR_ENCODER_LINES        (1000.0)
410 #define USER_MOTOR_MAX_SPEED_KRPM       (4.0)
411 #define USER_SYSTEM_INERTIA             (0.02)
412 #define USER_SYSTEM_FRICTION            (0.01)
```

- Notice that there are now four new fields for MY_MOTOR:
  - **USER_MOTOR_ENCODER_LINES** – This should be set to the number of pulses on the motor's encoder. If the motor does not have an encoder, set this value to 1.0.
  - **USER_MOTOR_MAX_SPEED_KRPM** – This should be set to the maximum speed of the motor.
  - **USER_SYSTEM_INERTIA** –This value will be identified as part of lab 5c or lab 12a. Set it the initial default value to 0.02.
  - **USER_SYSTEM_FRICTION** - This value will be identified as part of lab 5c or lab 12a. Set the initial default value to 0.01.
- There is an additional new define for InstaSPIN-MOTION, **USER_SYSTEM_BANDWIDTH_SCALE** - This definition represents the default bandwidth for the SpinTAC controller. This value will be determined in lab 05e or lab 13a. For now, set this parameter to the default value of 1.0

```
332 //! \brief Defines the default bandwidth for SpinTAC Control
333 //! \brief This value should be determined by putting SpinTAC Control through a tuning process
334 #define USER_SYSTEM_BANDWIDTH_SCALE    (1.0)
```

- In addition to the USER_MOTOR settings, it is important that you copy ANY field that was modified in any of the previous labs or as part of your hardware design process into the InstaSPIN-MOTION user.h file.

Using the GUI

- For Standalone Mode, follow the instructions in the section above:

  Running the GUI Standalone GUI .exe

  - Copy the .out just built to the appropriate Webapp folder, and rename to "appProgram.out" ex:

    …\sw\solutions\instaspin_motion\boards\boostxldrv8301_revB\f28x\f2806xM\projects_j1\ccs5\proj_lab06a\Release\proj_lab06a.out

    to the appropriate webapp folder:

    C:\ti\guicomposer\webapps\InstaSPIN_F2806xM_UNIVERSAL\appProgram.out

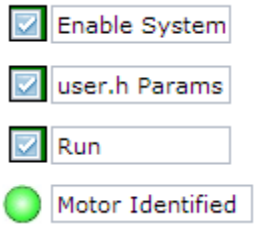
  - Run the executable C:\ti\guicomposer\webapps\InstaSPIN_F2806xM_UNIVERSAL\InstaSPIN_UNIVERSAL.exe

- In the GUI, there are separate tabs for Velocity control and Position control, make sure you have selected the tab that is appropriate for your application

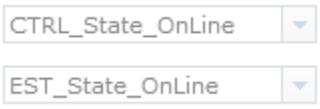
Inertia ID (Available in MOTION-Velocity, proj_lab05c & proj_lab12a)

To perform a system Inertia Identification:

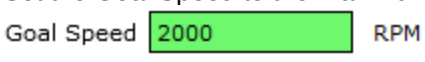
- SELECT Enable System
- SELECT Run



- Wait until CTRL_State_OnLine & EST_State_OnLine

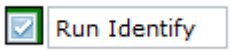


- Set the Goal Speed to the maximum application speed (without field weakening)

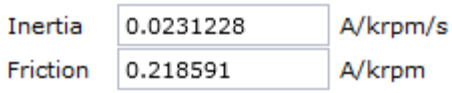


- If the application has large friction, reduce the Ramp Time. This will increase the rate of current change.

- SELECT Run Identify



- The motor should spin continuously until the test is completed
- When the test is complete, the Inertia and Friction values will be populated



Inertia ID Tips

Common Errors

This table lists the common errors that can occur in the Inertia ID process and how to solve them.

Table 1: List of common errors in the Inertia ID process

Error Code	**2003**	**2004**		**2006**
Meaning	Bad estimation value	Process timeout		Motor stops during test
Condition	High friction system	Motor spins	Motor starts slowly	High friction system
Solution	Decrease RampTime	Decrease GoalSpeed	Increase OutputTorque	Decrease RampTime

Update user.h Settings

- Copy the Inertia and Friction values to the Inertia and Friction #defines in the user.h file
    - These values will not change if you change the system current or speed scaling, but they might change if you adjust the PWM frequency or the current loop tuning
- Recompile and use this .out with the GUI in order to automatically load in the correct inertia and friction for future labs

Controller Tuning

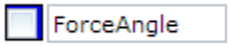
Current Controllers

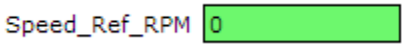
- The Iq and Id Current PI controller gains are numerically calculated & initialized during the motor identification process. The gains are calculated as follows:
    - Kp = ¼ * Bandwidth
        - Bandwidth = [Ls / CTRL_FREQ_Hz * IQ_CURRENT / IQ_VOLTAGE]
        - This ¼ factor is to soften the controller a bit for better stability
    - Ki = CTRL_FREQ_Hz / Ls * Rs
- These can be changed simply through the GUI, which instruments the following user code
    - gMotorVars.Kp_Idq = CTRL_getKp(ctrlHandle,CTRL_Type_PID_Id);
    - CTRL_setKp(handle,CTRL_Type_PID_Id,gMotorVars.Kp_Idq);
- You may want to conduct step response testing to validate that the current controller response meet the over/undershoot and settling time requirements of the application.
- **Note: current controller values will be updated starting in proj_lab5a. In earlier labs, changes to the current controller variables in the GUI will have no effect**

Speed Controller (Available in MOTION-Velocity, proj_lab05e - proj_lab06d)

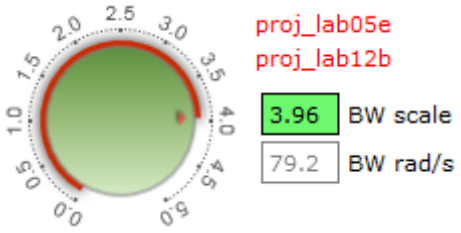
InstaSPIN-MOTION includes the SpinTAC speed controller. This controller works over a wider operating range than a PI controller and reduces the need for “gain staging”. This controller is tuned using a single parameter called Bandwidth. The stiffness of the system increases as this bandwidth is increased.

There are two suggested ways to tune this controller

- Method 1: Zero Speed tuning
    - Disable ForceAngle 

    - Set 0 speed 

    - Quickly rotate the motor shaft 90-180 degrees and then let go. The shaft should easily rotate.
    - Increase the bandwidth (BW scale) and feel how the controller is fighting to maintain zero speed. Repeat this process until it becomes difficult to move the motor shaft (this means that the controller is holding 0 speed).



- Once the controller is suitably holding 0 speed, set the Speed Reference (Speed_Ref_RPM) to the maximum speed in your application to ensure that the controller is stable across the entire operating range

  Speed_Ref_RPM 4000

  If the motor oscillates or vibrates reduce the bandwidth by 10-20%

- Replace the #define **USER_SYSTEM_BANDWIDTH_SCALE** value in the user.h file with the Bandwidth Scale value identified in the GUI

- Method 2: Step response tuning

  If the motor shaft is not accessible, you should conduct step response tuning

  - Set the acceleration and jerk to very large values
  - Set the curve type to trapezoidal

    Curve Type trapezoid

    Acceleration 120000 RPM/s

    Jerk 1000 KRPM/s^2

  - Conduct the Step Test
    - Set the speed reference to the minimum speed of your application

      Speed_Ref_RPM 100

    - Set the speed reference to approximately half of the maximum speed in your application

      Speed_Ref_RPM 2000

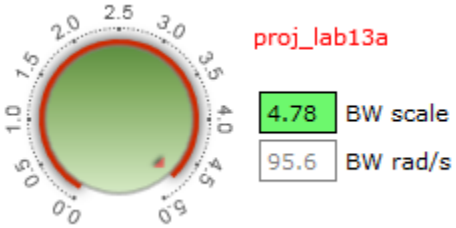
  - Observe how much overshoot the motor exhibits and how long it takes to settle to the correct speed
    - If you see too much overshoot, increase the bandwidth
    - If the motor oscillates for a long time after reaching the faster speed, decrease the bandwidth
  - Repeat the Step Tests and Bandwidth Scale adjustments until the motor exhibits little/no overshoot and quickly settles to the correct speed.
  - Replace the #define **USER_SYSTEM_BANDWIDTH_SCALE** value in the user.h file with the Bandwidth Scale value identified in the GUI

Position Controller (Available in MOTION-Position, proj_lab13a - proj_lab13e)

InstaSPIN-MOTION includes the SpinTAC position controller. This controller is a combined speed & position controller that is tuned using a single parameter, called Bandwidth. The stiffness of the system increases as this bandwidth is increased. The controller works over a wider operating range than a PI controller and reduces the need for "gain staging".

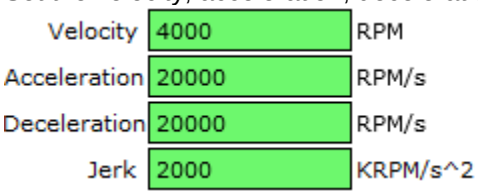
There are two suggested ways to tune this controller

- Method 1: Zero Speed tuning
  - Quickly rotate the motor shaft 90-180 degrees and then let go
  - Increase the bandwidth and feel how the controller is fighting to maintain zero speed. Adjust the bandwidth until the controller is suitably holding 0 position.

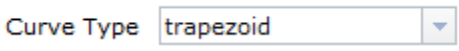


  - Do a couple position transitions (see the next section), to check the motor's response and stability.
    - If the motor oscillates or vibrates reduce the bandwidth by 10-20%
    - Replace the #define **USER_SYSTEM_BANDWIDTH_SCALE** value in the user.h file with the Bandwidth Scale value identified in the GUI

- Method 2: Step response tuning
  - If the motor shaft is not accessible, you should conduct step response tuning
  - Set the velocity, acceleration, deceleration, and jerk to very large values



  - Set the curve type to trapezoidal



  - Set the Position Step Integer to 1 mechanical revolution and Run the Position Profile

  - Observe how much overshoot the motor exhibits and how long it takes to settle to the correct position

- If you see too much overshoot, increase the bandwidth
- If the motor oscillates for a long time after reaching the position setpoint, decrease the bandwidth

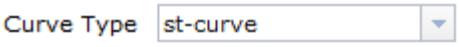
- Repeat the Step Tests and Bandwidth Scale adjustments until the motor exhibits little/no overshoot and quickly settles to the correct speed.
- Replace the #define **USER_SYSTEM_BANDWIDTH_SCALE** value in the user.h file with the Bandwidth Scale value identified in the GUI

Trajectory Generation

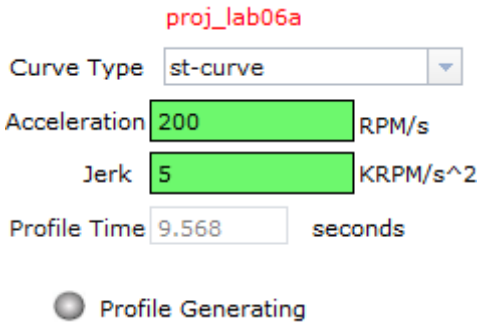
Speed Trajectories (Available in MOTION-Velocity, proj_lab06a - proj_lab06d)

SpinTAC Move provides the trajectory generation functions for InstaSPIN-MOTION. SpinTAC Move is a constraint based profile generator. Users provide the limits for the profile and SpinTAC will generate the fastest possible curve to meet those limits.

- Configure a motion profile
  - InstaSPIN-MOTION features several different curve types. Curve Type options include:

    - Trapezoid (Fixed Acceleration)
    - s-Curve (Fixed Jerk)
    - st-Curve (Continuous Jerk)
  - Select the st-curve

  - Set the Acceleration (rate of change of speed)

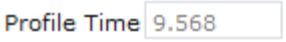
  - Set the Jerk (rate of change of acceleration)

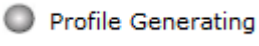
    - Set a low Jerk for smoother motion, and a high Jerk for fast transitions
- Run a motion profile
  - Change the speed setpoint

- The amount of time the profile will take is automatically calculated and displayed

- The LED indicator is on when SpinTAC Move is generating a profile

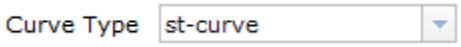
  

- Velocity profiles will be interrupted if the controller receives a new setpoint.

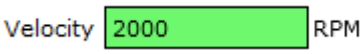
Position Trajectories (Available in MOTION-Position, proj_lab13b - proj_lab13e)

SpinTAC Move provides the trajectory generation functions for InstaSPIN-MOTION. SpinTAC Move is a constraint based profile generator. Users provide the limits for the profile and SpinTAC will generate the fastest possible curve to meet those limits.

- Configure a motion profile
  - InstaSPIN-MOTION features several different curve types. Curve Type options include:
    - Trapezoid (Fixed Acceleration)
    - s-Curve (Fixed Jerk)
    - st-Curve (Continuous Jerk)
  - Select the st-curve

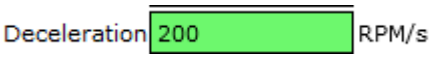


  - Set the Velocity (maximum speed in the profile)



  - Set the Acceleration (rate of change of speed)

  - Set the Deceleration (rate of change of speed)

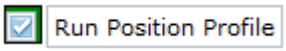


  - Set the Jerk (rate of change of acceleration)

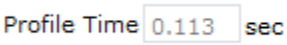
    - Set a low Jerk for smoother motion, and a high Jerk for fast transitions
- Run a motion profile
  - Set a position step
    - Set Position Step Integer for full revolutions

    - Set Position Step Fractional for partial revolutions

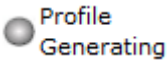
  - Select Run Position Profile



- The amount of time the profile will take is automatically calculated and displayed



- The LED indicator is on when SpinTAC Move is generating a profile



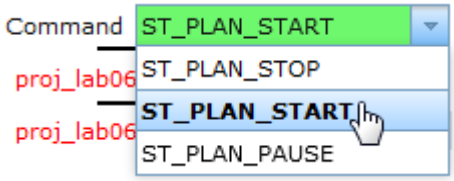
- Position profiles cannot be interrupted with a new setpoint. The controller will complete the current position profile, and then move to the next setpoint.

Motion Sequencing

Speed Motion Sequences (Available in MOTION-Velocity, proj_lab06b - proj_lab06d)

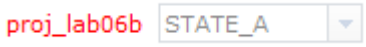
InstaSPIN-MOTION features SpinTAC Plan, a configurable motion sequence engine

- To start the motion sequence, set the command to ST_PLAN_START

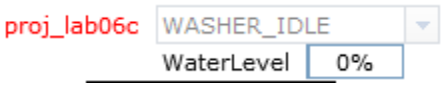


  - ST_PLAN_STOP will stop the current plan and return the controller to the idle state
  - ST_PLAN_PAUSE will pause the current plan until ST_PLAN_START is selected again

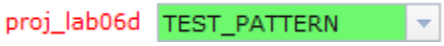
- Each lab has a different pre-configured motion sequence.
  - proj_lab06b is a simple A->B->C motion sequence. The GUI indicates the state.

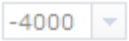


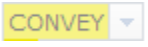
  - proj_lab06c is a Washing Machine motion sequence example. The GUI indicates the state and drum water level



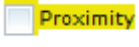
  - proj_lab06d includes three different selectable sequences: Test Pattern, Grocery Conveyor, or Garage Door
    - The motion sequence can be changed when status is ST_PLAN_IDLE



    - Test Pattern
      - Sequence of motions designed to test the motor
      - Current state indicated 

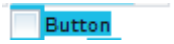
    - Grocery Conveyor motion profile simulates a conveyor belt at a grocery store
      - The GUI indicates current state 

and provides instrumentation
        - Turns on conveyor 

        - Simulates proximity switch 

    - Garage Door motion profile simulates a garage door

- The GUI indicates current state 

and provides instrumentation
    - Operation button to start motion 

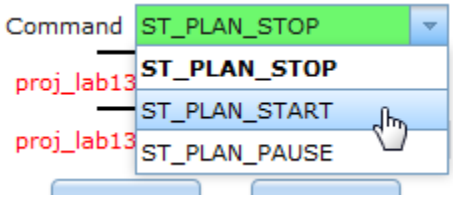
    - Up sensor (user to select) 

    - Down sensor (user to select) 

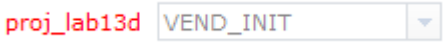
Position Motion Sequences (Available in MOTION-Position, proj_lab13c - proj_lab13e)

InstaSPIN-MOTION features SpinTAC Plan, a configurable motion sequence engine

- To start the motion sequence set the command to ST_PLAN_START

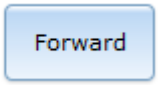


    - ST_PLAN_STOP will stop the current plan and return the controller to the idle state
    - ST_PLAN_PAUSE will pause the current plan until ST_PLAN_START is selected again
- Each lab has a different pre-configured motion sequence.
    - proj_lab13c is a simple A->B->C motion sequence. The GUI indicates the state:

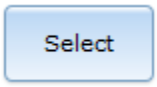

    - proj_lab13d is a Vending Machine motion sequence example
        - The GUI indicates the current state 

and the available item 

The GUI also provides instrumentation:



        - Button to advance to the next available item
        - Button to select the item and reduce the item inventory



Next Steps

- Continue to follow the InstaSPIN Projects & Labs User's Guide in MotorWare
  InstaSPIN Projects and Labs User's Guide
- Read through the User's Guide on relevant topics for your application
  InstaSPIN-FOC and InstaSPIN-MOTION User's Guide
- For details on how to customize this GUI, create your own, or export for standalone use please see the GUI Composer Wiki Site.
- Ask questions on the **InstaSPIN e2e forum**

WARNING

Do not close the GUI until the drive has been stopped. Failure to do so will leave the program running or put the processor into an unknown state, causing the system to continue to draw current, possibly damaging the controlCARD, board, host computer, motor and posing a fire hazard. After proper shut-down always disconnect the power supplies and remember that **capacitors are charged and will take time to dissipate!**