MOTOR CONTROL

Firmware User Guide: Low Voltage BLDC Motor Control using SAM Devices

USER GUIDE

LV Kit

Introduction

This user guide covers the firmware configuration details of motor control algorithms. The algorithms are Field Oriented Control (FOC) and Block Commutation (BC). Currently, the scope of the document takes care of FOC with sensorless operation and BC with HALL sensor operation. The supported controller is SAM D21.

Features

- FOC (Field Oriented Control) – Sensorless
- BC (Block Commutation) – Hall sensor
- Atmel® Start usage

Table of Contents

1 FOC Sensorless

The following sections explain the firmware configuration for FOC sensorless solution.

1.1 HV and LV Kit

In the file motor_control_defs.h, ensure the following is done to use the LV (low voltage kit) kit. Ensure that SAMHVDRIVE is disabled and ATBLDC24V is enabled. The default kit motor is M42BL024042 and the set of parameters is defined in the same file. All the parameters are explained in Section 1.2.

Table 1-1. LV Kit

Code listing in motor_control_defs.h

```
#define M42BL02402            /* 42BL02402-0026B-002 motor */
#define ATBLDC24V             /* low voltage demo */
/*#define SAMHVDRIVE*/       /* high voltage demo */
```

If the HV (High Voltage) kit should be used then the following has to be done. Ensure that the macro ATBLDC24V is commented and SAMHVDRIVE is enabled.

Table 1-2. HV Kit

Code listing in motor_control_defs.h

```
/* #define ATBLDC24V */             /* low voltage demo */
#define SAMHVDRIVE      /* high voltage demo */
```

Similarly add a suitable name for the motor as a #define and define all the equivalent parameters given in Section 1.2.

1.2 Configuration Parameters

Table 1-3. PWM_HPER_TICKS

Properties	Description
Name	PWM_HPER_TICKS
Units	Number
Description	PWM frequency used to control the motor.
Remarks	MCU frequency is kept at 48MHz, i.e. 48 PWM ticks corresponds to 1µs. To achieve a frequency of 6kHz (period of 166.67µs), 8000 PWM ticks is required. Centre aligned PWM is used hence half of the required ticks is used here. Similarly to achieve PWM frequency of 10kHz (period of 100µs), 4800 PWM ticks is required. (Macro would hold 2400.) Formula: PWM_HPER_TICKS = (MCU_FREQ_HZ) / (REQUIRED_FREQ_HZ * 2)

Table 1-4. START_SPEED_DEFAULT

Properties	Description
Name	START_SPEED_DEFAULT
Units	RPM
Description	Default start up speed in rpm
Remarks	The motor's initial startup speed and the value will also be displayed in the GUI for usage

Table 1-5. POLAR_COUPLES

Properties	Description
Name	POLAR_COUPLES
Units	Number
Description	Number of pole pairs in a motor
Remarks	None

Table 1-6. MIN_FRE_HZ

Properties	Description
Name	MIN_FRE_HZ
Units	Hertz
Description	Minimum frequency supported by the motor
Remarks	This value is used in radians per second within the code, for instance if the minimum fre-quency is 50Hz, this is realized as 314 rad per sec (2 * PI * freq.). The same in rpm would be (frequency * 60 / POLAR_COUPLES) Formula: Angular frequency (ω) = 2 * PI * MIN_FRE_HZ Minimum RPM = (MIN_FRE_HZ * 60) / PO-LAR_COUPLES

Table 1-7. MAX_FRE_HZ

Properties	Description
Name	MAX_FRE_HZ
Units	Hertz
Description	Maximum frequency supported by the motor
Remarks	Scaling factor: 1 rpm is represented as 1 * 16384 / MAX_RPM in the code. For example if maximum rpm is 6000 then 1 rpm would be 2.73. By integer division it would be 2. Essentially the maximum frequency supported by the motor plays role in resolution of the speed. Enter the value appropriate to the motor specification.

Table 1-8. R_STA

Properties	Description
Name	R_STA
Units	Ohm
Description	Stator Phase Resistance
Remarks	If the data sheet has provided line-to-line resistance, the value can be divided by 2

Table 1-9. L_SYN

Properties	Description
Name	L_SYN
Units	Henry
Description	Synchronous inductance
Remarks	Value in general it is one or two times of the phase self-inductance

Table 1-10. MAX_CUR_AMP

Properties	Description
Name	MAX_CUR_AMP
Units	Amperes
Description	Maximum current that can be allowed to the motor
Remarks	Used in the speed PI loop for limit check

Table 1-11. START_CUR_AMP

Properties	Description
Name	START_CUR_AMP
Units	Amperes
Description	Peak Start up current
Remarks	During the speed ramp, in ALIGN state the “d” current is ramped up to this configuration value

Table 1-12. ACC_RPM_S

Properties	Description
Name	ACC_RPM_S
Units	Rpm per second
Description	Acceleration ramp

Properties	Description
Remarks	During the speed ramp, allowed rpm per second is given here. This value is sampled down to 10ms and added to absolute speed during ramp. As explained earlier the scaling factor given for rpm is taken care with a derivative macro.

Table 1-13. DEC_RPM_S

Properties	Description
Name	DEC_RPM_S
Units	Rpm per second
Description	Deceleration ramp
Remarks	During the speed ramp descent, allowed rpm per second is given here. Similar to accelera-tion, deceleration is done.

Table 1-14. KP_V_A

Properties	Description
Name	KP_V_A
Units	Volt / Ampere
Description	Current loop proportional gain
Remarks	Same gain is used in both Iq and Id current control

Table 1-15. KI_V_AS

Properties	Description
Name	KI_V_AS
Units	Volt / (Ampere * Sec)
Description	Current loop integral gain
Remarks	Same gain is used in both Iq and Id current control

Table 1-16. KP_AS_R

Properties	Description
Name	KP_AS_R
Units	Amp / (rad/sec)
Description	Speed loop proportional gain
Remarks	Speed loop Kp value

Table 1-17. KI_A_R

Properties	Description
Name	KI_A_R
Units	Amp / ((rad/sec) * sec)
Description	Speed loop integral gain
Remarks	Speed loop Ki value

Table 1-18. STUP_ACCTIME_S

Properties	Description
Name	STUP_ACCTIME_S
Units	Sec
Description	Startup acceleration time
Remarks	In STARTING state, the time taken to reach the MINIMUM speed

Table 1-19. CUR_RISE_T

Properties	Description
Name	CUR_RISE_T
Units	Sec
Description	Current rising time during start up alignment
Remarks	The time required to reach the START_CUR_AMP in ALIGN state

Table 1-20. CUR_FALL_T

Properties	Description
Name	CUR_FALL_T
Units	Sec
Description	Direct current falling time after startup
Remarks	During RUNNING state the time required to ramp down the “d” current

Table 1-21. SAMPLING_FREQUENCY

Properties	Description
Name	SAMPLING_FREQ
Units	Hertz
Description	Sampling frequency of the motor control loop: 0.25 * MCU_FREQ_HZ / PWM_HPER_TICKS. (Half of PWM frequency, value is represented in Hertz. To maintain the sampling frequency same as PWM frequency, multiply it by 0.5 instead of 0.25.)

Properties	Description
Remarks	The general notation followed is to maintain the sampling frequency same as PWM frequency. If PWM frequency is 10kHz then sampling frequency should also be kept at the same value. However due to additional application requirements it's not possible to accommodate all the routines within the PWM control loop. Hence the sampling frequency is kept at half of the PWM control frequency.

1.3 Identification and Changing of PWM Pins

We need six PWM pins and four ADC pins to run the FOC sensorless solution. The six PWM pins are fixed in the board and the same can be changed in the workspace (if required). The pins can also be changed in the Atmel start.

WO_0 ,WO_1,WO_2 refers to high side PWM pins of phase A, B, and C respectively. WO_4, WO_5, WO_6 refers to the low side PWM pins of phase A, B, and C respectively.

Table 1-22. Code Listing PWM Pins

Code listing in atmel_start_pins.h

```
#define PWM_WO_0  GPIO(GPIO_PORTA, 8)
#define PWM_WO_1  GPIO(GPIO_PORTA, 9)
#define PWM_WO_2  GPIO(GPIO_PORTA, 10)
#define PWM_WO_4  GPIO(GPIO_PORTA, 14)
#define PWM_WO_5  GPIO(GPIO_PORTA, 15)
#define PWM_WO_6  GPIO(GPIO_PORTA, 16)
```

Similarly ADC pins can also be changed in the same file.

1.4 Steps to Make PWM and Sampling Frequency Same

The solution that is provided by default has PWM frequency at 6kHz and motor control loop sampling frequency at 3kHz. In general this is sufficient to turn the motor with good efficiency, but in case there is a need to change the motor control loop sampling frequency to the same as PWM frequency (6kHz) the following should be done.

Table 1-23. Change in Following Files

Steps	Description
Name	SAMPLING_FREQ
motor_control_defs.h	Change #SAMPLING_FREQ# to the following definition: 0.5 * MCU_FREQ_HZ / PWM_HPER_TICKS
Function: adc_result_ready	In the function adc_result_ready – triggered ADC channel should be pointed to the phase current channel and DC bus voltage should be measured using polling method. Refer Table 1-24.

Table 1-24. Code Listing adc_result_ready

Code listing in adc_result_ready
See code below

```
void adc_result_ready(const struct adc_module *const adc_inst)
{
    if (0U == adc_interrupt_counter)
    {
        adc_interrupt_counter = 1;
        /* store the first ADC result value */
        cur_mea[phaseindex[1]] =
                             ((int16_t)adc_result_data -
(int16_t)adc_calibration[phaseindex[1]]);
        /* select the next adc channel */
        adc_select_channel(adc_channel_pins[phaseindex[2]]);
        /* start the conversion */
        /*lint -e9078 -e923 */
        ADC->SWTRIG.reg |= (uint8_t)ADC_SWTRIG_START;
        /*lint +e9078 +e923 */
    }
else
    {
           /*lint -save -e9078 -e923 */
           /* MISRA 11.4, 11.6 VIOLATION */
           /* register access */
           adc_interrupt_counter = 0U;
           /* store the second ADC result value */
           cur_mea[phaseindex[2]] =
             ((int16_t)adc_result_data - (int16_t)adc_calibration[phaseindex[2]]);
           /* Both current ADC channel results are now ready: let's read
            the BUS voltage (via polling) */
           /* Disable the ADC interrupt */
           adc_disable_interrupt(adc_inst->hw,ADC_INTERRUPT_RESULT_READY);
           /* select the right channel */
           adc_select_channel((uint16_t)MOTOR_PHASE_DC_VOLTAGE_PIN);
```

Code listing in adc_result_ready

```
/* start the conversion */
        ADC->SWTRIG.reg |= (uint8_t)ADC_SWTRIG_START;
        /* something can be done while waiting for end of conversion */
        current_measurement_management();
        /* check if conversion is finished */
        while (0U == ADC->INTFLAG.bit.RESRDY)
        {
          /* wait for conversion to finish */
        }
        /* clear interrupt flag */
        ADC->INTFLAG.reg = ADC_INTFLAG_RESRDY;
        /* store ADC result in variable */
        while ((ADC->STATUS.reg & ADC_STATUS_SYNCBUSY) > 0U)
        {
          /* Wait for synchronization */
        }
        adc_dc_bus_voltage = ADC->RESULT.reg;
        /* motor control */
        motorcontrol();
        /* select the next channel */
        adc_select_channel((uint16_t)adc_channel_pins[phaseindex[1]]);
        /* Enable the interrupt */
        adc_enable_interrupt(ADC,ADC_INTERRUPT_RESULT_READY);
        /*lint -restore */
   }
   return;
}
```

2 BC HALL

The following section explains the HW and firmware configuration of Low voltage Kit.

2.1 LV Kit

The BC-HALL algorithm is supported only in Low Voltage Kit. For the complete connections, refer the kit manual.

The default motor used in LDO and the same define can be seen in block_commutation_cfg.h file.

Table 2-1. Motor Define

Code listing in atmel_start_pins.h
`#define M42BL02402            1`

2.2 Configuration Parameters

Table 2-2. MOTOR_POLE_PAIRS

Properties	Description
Name	MOTOR_POLE_PAIRS
Units	Number
Description	Number of pole pairs for the given motor
Remarks	Available in any given motor data sheet, for the kit motor the value is 4

Table 2-3. SPEED_KP_DEFAULT

Properties	Description
Name	SPEED_KP_DEFAULT
Units	Number (proportional gain). The unit can also be realized as (1 / rpm).
Description	Speed pi control proportional gain
Remarks	The value can also be changed via data visualizer in run time. The given PI control converts the given speed to an equivalent duty cycle. The gain is scaled by a factor of 64.

Table 2-4. SPEED_KI_DEFAULT

Properties	Description
Name	SPEED_KI_DEFAULT
Units	Number (proportional gain). The unit can also be realized as (1 / (rpm * minutes))
Description	Speed PI Integral gain
Remarks	The value can also be changed via data visualizer in run time. The given PI control converts the given speed to an equivalent duty cycle.

Table 2-5. START_SPEED_DEFAULT

Properties	Description
Name	START_SPEED_DEFAULT
Units	Rotations per Minute (RPM)
Description	When the motor starts the motor starts from the given reference speed
Remarks	NA

Table 2-6. SPEED_DEFAULT_DUTY

Properties	Description
Name	SPEED_DEFAULT_DUTY
Units	Number (ticks) with reference to period ticks
Description	The period is put at 2400 (50µs). A default value of duty cycle is required for the startup.
Remarks	As a thumb rule this value could be approximately 10% of the period and a relevant startup speed should be provided

Table 2-7. MOTOR_MINIMUM_SPEED

Properties	Description
Name	MOTOR_MINIMUM_SPEED
Units	RPM
Description	The minimum speed of the motor
Remarks	This value is used within DV for restriction purposes

Table 2-8. MOTOR_MAXIMUM_SPEED

Properties	Description
Name	MOTOR_MAXIMUM_SPEED
Units	RPM
Description	The maximum speed of the motor
Remarks	This value is used within DV for restriction purposes

Table 2-9. MOTOR_RAMPUP_SPEED_PER_MS

Properties	Description
Name	MOTOR_RAMPUP_SPEED_PER_MS
Units	RPM/ms (millisecond)
Description	During the change of speed, the ramp for the rpm to be provided in ms units

Properties	Description
Remarks	Assuming the motor wants to change from 2000 to 2500 rpm, a value of 1 in this column, will make the system to take 500ms to achieve 2500 rpm from 2000 rpm. A value of 0 would imply an instant jump from 2000 to 2500.

The macro MOTOR_MAXSPEED_TARGET is not used anywhere.

2.3 Identification and Changing of PINS

Changing of PWM pins is similar to section given in FOC (1.3).

Hall Pins:

Though the HALL pins can also be changed in a similar way as the PWM pins but there is a hard reference done within the code, so the user has to be extra cautious if a change of pin is desired. Appropriate pins and register should be changed in the functions motor_start and update_communication. Currently this is practiced for optimization reasons.

Table 2-10. Hall Pins Changes Within Motor Control

Code listing in motor_start and update_communication

`curhall1 = (uint8_t)((REG_PORT_IN0 & 0x00000008U)>>1U);`
`curhall2 = (uint8_t)((REG_PORT_IN0 & 0x00040000U)>>17U);`
`curhall3 = (uint8_t)((REG_PORT_IN0 & 0x10000000U)>>28U);`

2.4 To Run a Different Motor with BC-HALL

Block commutation principle demands two significant changes within TCC peripheral to turn a motor.

1. Hall sensor pattern.
2. Commutation pattern.

Sample motor patterns are provided within the given code base. However, if a new motor is provided, following should be done.

Hall sensor pattern

We know that the hall sensors is three wired and grey coded. To optimize the implementation the following is done within the code. Typical Hall sensor pattern would move in the following way:

1(001) → 3 (011) → 2 (010) → 6 (110) → 4 (100) → 5 (101) → 1(001)

Assuming we read 1 as the valid Hall signal, the software should have the ability to know that the next pattern would be 3. Hence an array of 16 bytes is created, in the array index of 1, value 3 is stored. Similarly, if the hall sensor signal is 6, in the array index of 6 the value 4 is stored, which would be the next hall pattern. Refer Table 2-11 for the pattern.

Array index 1 to 6 is used for CW (clock wise rotation) and 9 to 14 is used for CCW (counter clock wise). Array index 0, 7, 8, and 15 will be unused.

Table 2-11. Hall Sensor Pattern

Code listing in motor_control.c
`static const uint8_t HALL_ARRAY[16] = { 0, 3, 6, 2, 5, 1, 4, 0, 0, 5, 3, 1, 6, 4, 2, 0 };`

Commutation pattern

For every Hall sensor pattern read, equivalent commutation pattern should be applied to the TCC peripheral the same is stored in the variable COMMUTATION_ARRAY.

To understand the pattern, let's analyze one value. We are giving the PWM to high side and a simple ON/OFF switch to low sides. All the low side switches should be OFF so that there is no PWM supplied. All High side switches should be switched ON/OFF depending on the hall state. This is determined in the lower two nibbles.

The higher two nibbles determine the value of output line if there is no PWM is supplied to that pin.

Example: 0x4075. Here the commutation is to supply PWM to phase B (Phase V) and make the low side of Phase C (phase W) high. The rest is not supplied anything. A value of 1 in pattern output enable stops the PWM and the output line state is determined by equivalent pattern output value.

Pattern output value								Pattern output enable							
-	LS-W	LS-V	LS-U	-	HS-V	HS-V	HS-U	-	LS-W	LS-V	LS-U	-	HS -W	HS -V	HS -U

Pattern output value								Pattern output enable							
-	1	0	0	-	0	0	0	-	1	1	1	-	1	0	1

Table 2-12. Commutation Pattern

Code listing in motor_control.c
(code below)

```
        const uint16_t COMMUTATION_ARRAY[16] = {
               0,
               /* to achieve C+ B-, put the following in Pattern register H1H2H3: 001 */
               0x4075, //0x4075, HS //0x0237U, LS
               /* to achieve B+ A-, put the following in Pattern register H1H2H3: 010 */
               0x2076, //0x2076, HS //0x0157U, LS
               /* to achieve C+ A-, put the following in Pattern register H1H2H3: 011 */
               0x4076, //0x4076, HS //0x0137U, LS
               /* to achieve A+ C-, put the following in Pattern register H1H2H3: 100 */
               0x1073, //0x1073, HS //0x0467U, LS
               /* to achieve A+ B-, put the following in Pattern register H1H2H3: 101 */
               0x1075, //0x1075, HS //0x0267U, LS
               /* to achieve B+ C-, put the following in Pattern register H1H2H3: 110 */
               0x2073, //0x2073, HS //0x0457U, LS
               /* Not a valid pattern */
               0,
               0,
               0x2073,
               0x1075,
               0x1073,
               0x4076,
               0x2076,
               0x4075,
               0
};
```

2.5 PWM Frequency

The PWM frequency provided is 20kHz (50ms time period). If a change is needed the following line should be changed. The clock is set at 48MHz, and edge aligned PWM is used. 48 ticks would correspond to 1ms.

Table 2-13. PWM Frequency

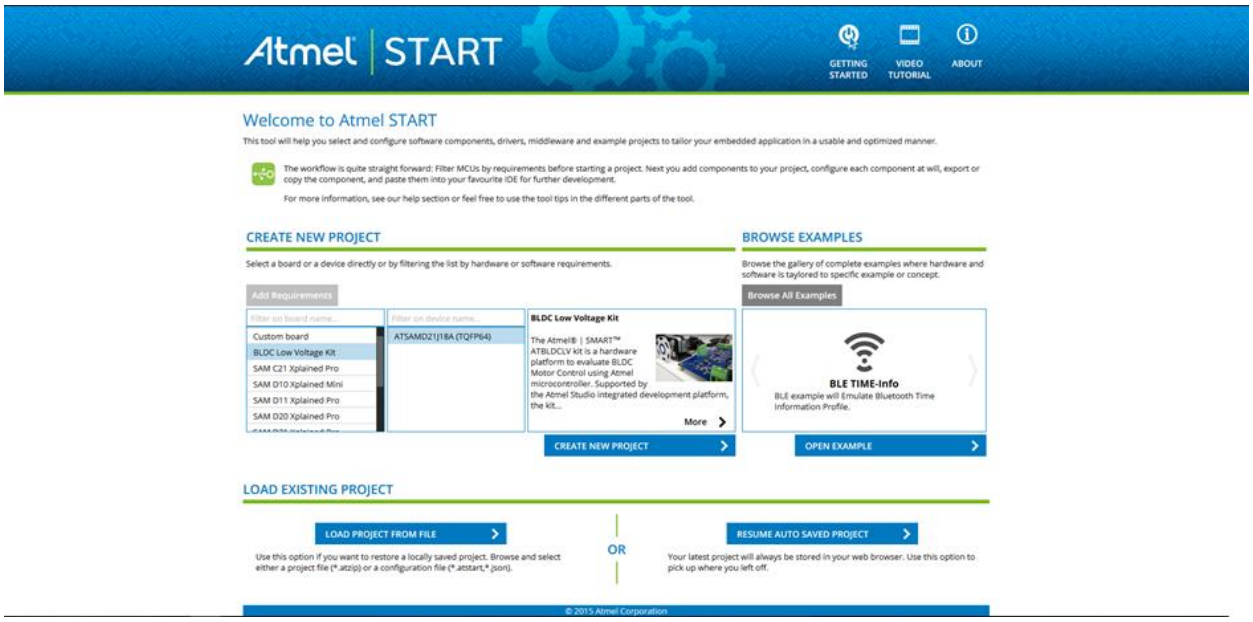
Code listing in motor_control.c – function (motor_pwm_init)
`pwm_set_parameters(&PWM_MOTOR_DRIVER, 2400, 240);`

3 BC HALL via Atmel Start

1. Go to http://start.atmel.com/.
2. The following web page would appear. Select BLDC Low voltage kit and click "Browse All Examples".

Note: Don't use Create New Project as the dependencies are high and user had to take care of it.

Figure 3-1. Browse Project



Welcome to Atmel START

This tool will help you select and configure software components, drivers, middleware and example projects to tailor your embedded application in a usable and optimized manner.

The workflow is quite straight forward: Filter MCUs by requirements before starting a project. Next you add components to your project, configure each component at will, export or copy the component, and paste them into your favourite IDE for further development.

For more information, see our help section or feel free to use the tool tips in the different parts of the tool.

CREATE NEW PROJECT

Select a board or a device directly or by filtering the list by hardware or software requirements.

BLDC Low Voltage Kit

The Atmel® | SMART™ ATBLDCLV kit is a hardware platform to evaluate BLDC Motor Control using Atmel microcontroller. Supported by the Atmel Studio integrated development platform, the kit...

BROWSE EXAMPLES

Browse the gallery of complete examples where hardware and software is taylored to specific example or concept.

BLE TIME-Info

BLE example will Emulate Bluetooth Time Information Profile.

LOAD EXISTING PROJECT

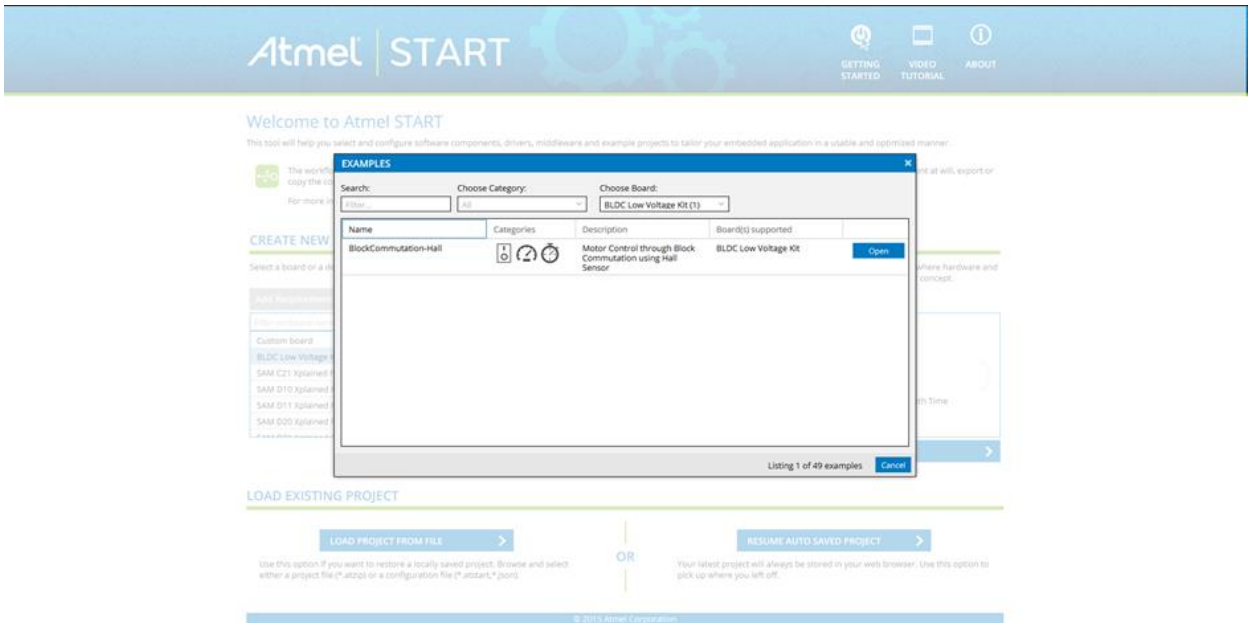
Use this option if you want to restore a locally saved project. Browse and select either a project file (*.atzip) or a configuration file (*.atstart,*.json).

Your latest project will always be stored in your web browser. Use this option to pick up where you left off.

© 2015 Atmel Corporation

3. The list of examples supported for the kit are loaded.

Figure 3-2. Select Example Project



EXAMPLES

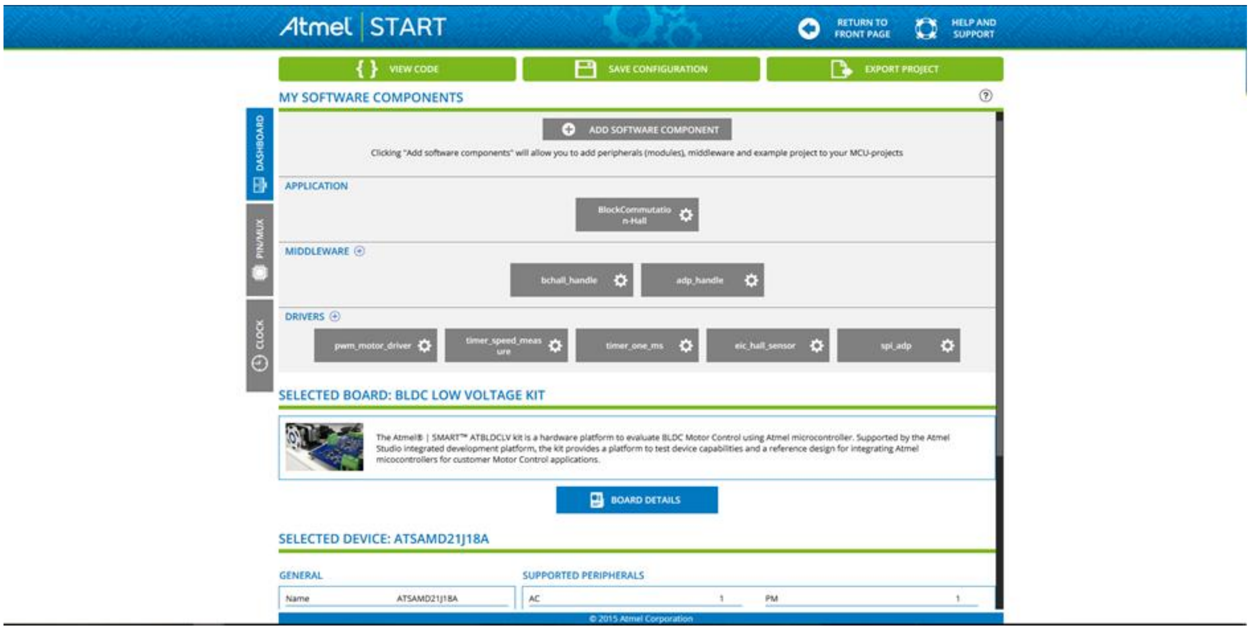
Search: Filter... Choose Category: All Choose Board: BLDC Low Voltage Kit (1)

Name	Categories	Description	Board(s) supported	
BlockCommutation-Hall		Motor Control through Block Commutation using Hall Sensor	BLDC Low Voltage Kit	Open

Listing 1 of 49 examples

4. On clicking "Open", the application is loaded in START.

Figure 3-3. List of Components in the Example Application



MY SOFTWARE COMPONENTS

Clicking "Add software components" will allow you to add peripherals (modules), middleware and example project to your MCU-projects

APPLICATION

MIDDLEWARE

DRIVERS

SELECTED BOARD: BLDC LOW VOLTAGE KIT

The Atmel® | SMART™ ATBLDCLV kit is a hardware platform to evaluate BLDC Motor Control using Atmel microcontroller. Supported by the Atmel Studio integrated development platform, the kit provides a platform to test device capabilities and a reference design for integrating Atmel micocontrollers for customer Motor Control applications.

SELECTED DEVICE: ATSAMD21J18A

GENERAL

Name	ATSAMD21J18A

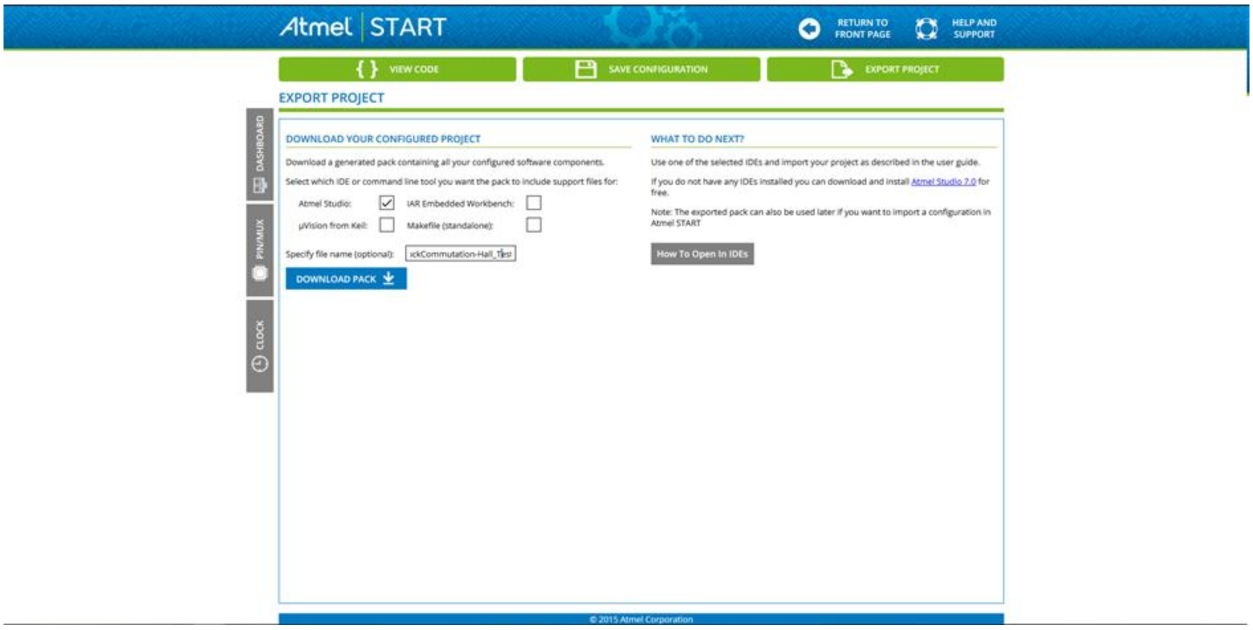
SUPPORTED PERIPHERALS

AC	1	PM	1

5. “EXPORT PROJECT” button opens the application pack download page, by clicking the “DOWNLOAD PACK” button, the application ATZIP file can be downloaded.

Note: Only the Atmel Studio project download is supported.

Figure 3-4. List of Components in the Example Application



EXPORT PROJECT

DOWNLOAD YOUR CONFIGURED PROJECT

Download a generated pack containing all your configured software components.

Select which IDE or command line tool you want the pack to include support files for:

Atmel Studio: ✓ IAR Embedded Workbench: ☐
µVision from Keil: ☐ Makefile (standalone): ☐

Specify file name (optional): ıckCommutation-Hall_Test

WHAT TO DO NEXT?

Use one of the selected IDEs and import your project as described in the user guide.

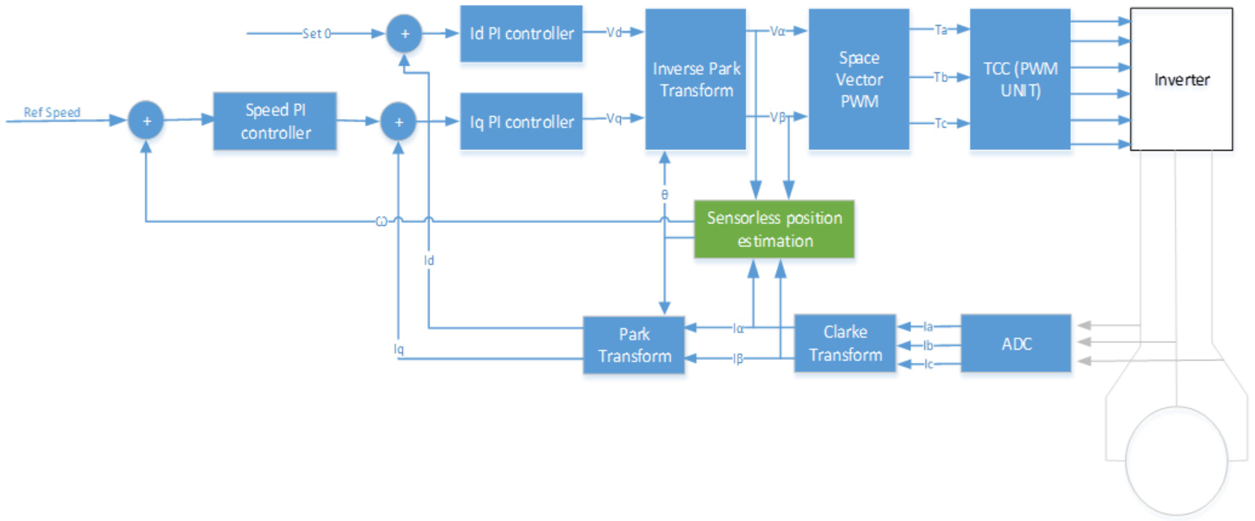
If you do not have any IDEs installed you can download and install Atmel Studio 7.0 for free.

Note: The exported pack can also be used later if you want to import a configuration in Atmel START

6. Open the download .atzip file in Atmel Studio 7, which will open the project creation dialog. The complete working application will be created through the dialog.

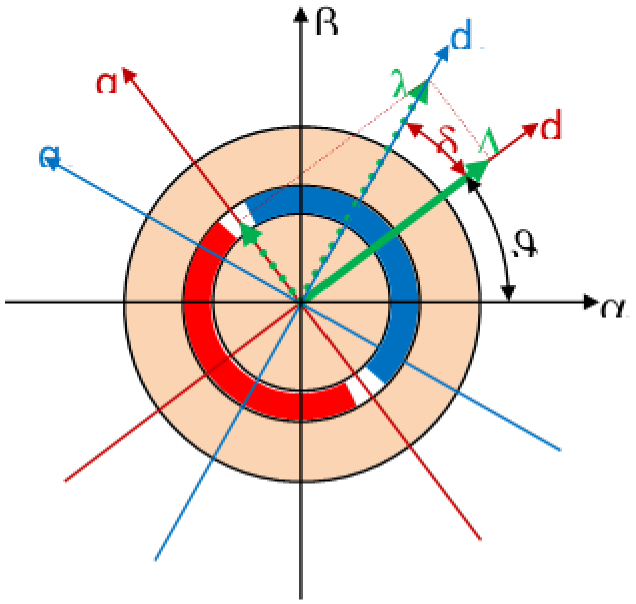
4 FOC Sensorless – Startup

The basic FOC block diagram is well known and given below.



4.1 Principles

- It's a reference system in which the variables which allow us to control the system are DC quantities (slowly varying in the time) rather than sinusoidal
- This allows the use of the classical PI controllers
- It is not important which variables are controlled through the PI controllers, but only the fact that we are working in a rotating reference system, which allows us to see these variables as DC quantities
- The classic FOC scheme provides the control of the stator currents, seen from a reference system which is linked to the permanent magnets flux Λ_m (rotor)
- Another possible choice is to orientate the rotating system as the total stator flux λ



$$\lambda_d = L_d i_d + \Lambda_m$$
$$\lambda_q = L_q i_q$$

4.2 Startup Procedure

This is a generic startup procedure that could be used to turn motors.

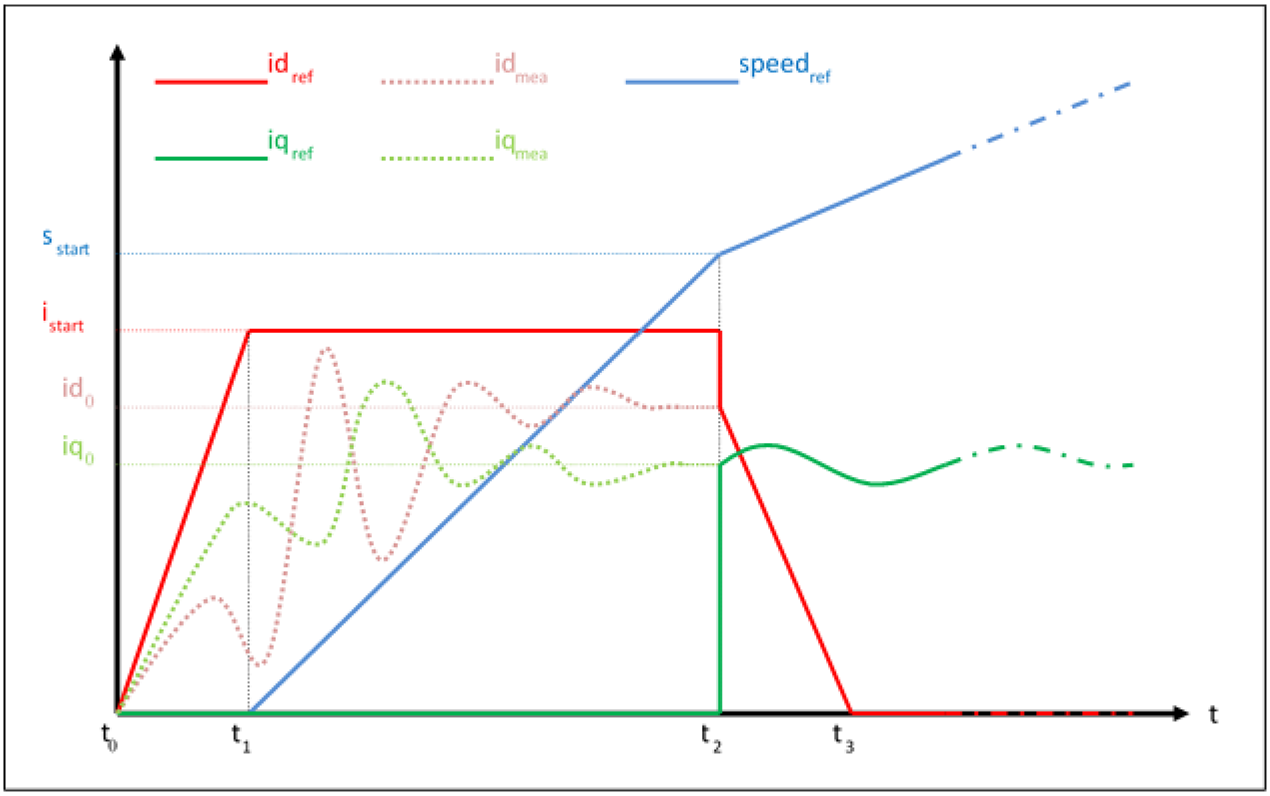
When the motor is stopped the position is usually unknown. The bemf observer cannot work when the speed is lower than a minimum value which is determined by the motor and the application. From this comes the necessity of an open loop acceleration procedure, which allows the motor to reach the minimum speed at which the observer can work correctly; at this point the speed loop can be closed and the control assumes the form already described in the block scheme.

Since the effective position is not known, we will use an arbitrary position for our (d, q) reference system; this means that the quantities which are referred to d-quantity or q-quantity in this process will not have any determined relationship with the rotor position, till when the speed loop will be closed.

Let's start with angle zero: our rotating reference system is aligned with (α, β) system and it is stopped. In sequence, we will perform the following operations:

- Increase the current reference at zero speed (that is keeping constant the position). This will produce a current vector with a constant argument (for simplicity, d reference is chosen to zero). When (and if) the current amplitude will be large enough, the rotor will tend to align to the current vector direction. The effective rotor position will depend on the starting position and on the load.
- When reached a current vector amplitude which is retained enough (depending on the application, this value can be equal to the maximum allowable current), we begin to change the current vector position. This is obtained keeping constant the current references, but changing the reference system position. The speed is increased linearly, and the position is obtained integrating the speed. In this acceleration process the real position of the rotor in respect to our reference system is still uncertain, so it is unknown how the current vector is divided into torque and flux components.
- During the acceleration process, the bemf observer works. When the speed is high enough, and after a minimum settling time, the observer will be more or less aligned with the real system, so the position of the system becomes known.
- When the speed is retained high enough and the observer settling time is elapsed, it comes the moment to close the speed loop. This means that we should align our reference system position with the rotor position, estimated with the bemf observer. Here the problem lies in the memories of the current PI controllers, which are referred to the actual arbitrary (d, q) system. The integral memories of the current PI controllers are voltages, referred to the actual rotating system, so the first operation to do is to refer them to the static (α, β) system with a Park inverse transformation, then to refer them to the new (d, q) reference system position with a direct Park transformation, performed after having update the angle. The same operations are needed for the current references, in order to avoid any discontinuity in the transition.
- From now, in advance, the normal closed loop speed control can be performed, so the q current reference will be determined by the speed control loop, while the d current reference, which is still present, can be gradually reduced to zero.

Figure 4-1. Startup Procedure



The important parameters, which have to be chosen for the startup procedure are:

- Startup speed: it is the speed reached during the startup phase. In the implementation, it coincides with the “minimum speed”, which is a parameter that can be modified by the user. It should be high enough to allow a good behavior of the phase estimation process.
- Startup time: it is the time required to reach the start up speed. It should be long enough to allow the estimation algorithm to stabilize (to recover from the initial condition errors). One second is usually a good value.
- Startup current: it is the current level imposed during the startup. It should be kept as low as possible, depending on the load.

5 ATMEL EVALUATION BOARD/KIT IMPORTANT NOTICE AND DISCLAIMER

ATMEL CORPORATION
1600 Technology Drive
San Jose, CA 95110
USA

6 Revision History

Doc Rev.	Date	Comments
42711A	04/2016	Initial document release.

Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**