Application Note, V1.0, July 2007

AP16117

XC164CS/CM

Speed control of Brushless DC motor with Hall sensor using DAvE Drive for Infineon XC164CM/CS microcontrollers

Microcontrollers

Never stop thinking

Edition 2007-07-04

Published by
Infineon Technologies AG
81726 München, Germany

AP16117		
Revision History:	2007-07	V1.0
Previous Version:	none	

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com

Table of Contents **Page**

1 Introduction

1.1 Overview

This application notes describes the Sensored BLDC motor control using the XC164CS/CM microcontroller. The motor control software uses the XC164CS/CM peripherals while the mathematical computations like PI control algorithm uses the DSP Data processing (MAC Unit) functionality of the microcontroller. The software is written in 'C' language with the PI-controller subroutine in assembly to make use of the MAC unit advantages.

The basic fundamentals of the BLDC motors and how to control the speed of such motors with the XC164CS/CM controller are explained. The advantages of the microcontroller peripherals which are specifically designed for the motor control operations are discussed: Capture and Compare Unit for modulation and PWM generation (CAPCOM6) and fast 10-bit Analog to Digital converter (ADC). The software for the motor control is generated using the DAvE Drive tool and the usage of this tool for the development of BLDC motor control with XC164CS/CM Infineon microcontrollers is explained.

1.2 Motor Control with XC164CS/CM microcontrollers

The XC164 family of microcontrollers has dedicated peripherals which are specifically designed for the motor control applications. The key features of the microcontroller are:

- High performance 16-bit CPU with Five Stage Pipeline and MAC Unit.
- Control Oriented Instruction set with High efficiency.
- 14-channel A/D converter with programmable resolution (10-bit or 8-bit).
- Capture/Compare Unit for flexible PWM signal generation (CAPCOM6) – 3/6 Capture/Compare channels and 1 Compare channel.
- 8-Channel Peripheral Event controller.
- 16-Priority-Level Interrupt system.
- Two Asynchronous/Synchronous serial channels (USARTs).
- Bootstrap Loader for flexible system initialization.

With the intensive autonomous use of dedicated peripherals designed for motor control, more the CPU load can be saved. The CPU can be used for performing other key tasks of the application.

1.3 DAvE Drive for Motor Control

DAvE Drive is a tool used to configure the software for the control of Brushless DC Motor. The configuration for a selected application can be done easily using the user friendly GUI environment.

DAvE Drive uses the technology of Digital Application virtual Engineer (DAvE), a code generator for Infineon microcontrollers. Along with the DAvE generated code, the Motor Control Library (MCL) will be used for the generation of a complete application source code. The motor control library contains the control algorithm specific code.

This software can be used along with the hardware and motor supplied with the DAvE Drive kit.

The software can be modified in an Integrated Development Environment called MiniIDE, if needed.

For hex file generation, the tool chains of Keil and Tasking are supported. The generated hex file can be downloaded into Flash memory of the microcontroller using the DAvE Drive Run time Control Panel. In addition, you can communicate with the running microcontroller to control the operation of the motor and to monitor the variables, optionally with graphical display.

1.4 Hardware and Software Components

For a workable system, the following hardware components are required:

- PC with Microsoft Windows XP or Windows 2000 operating system, with serial port or serial port emulation.
- Infineon Easy Kit with Infineon XC164CS\XC164CM controller.
- Infineon 3-Phase Motor Driver Board, connected with Easy kit.
- Brushless synchronous 3-Phase Motor, e.g., BPMC low power BLDC motor.
- Power Supply for Easy kit, Driver board and 3-Phase Motor.
- Cables and connectors for Power supply, serial port and 3-Phase motor.

Additionally the following software components are needed:

- Keil (uV3) or Tasking (8.5r2) Tool chain for Infineon XC16x.
- Infineon DAvE Drive Software Package V1.0.

2 Operation of Brushless DC Motors

2.1 BLDC Basics - Hall Sensor Mode

The Brushless DC motors are a variant of Permanent magnet DC motors. PM DC Motors are synchronous motors in which the rotor field is driven with a constant current. By driving the rotor winding with a constant current, a fixed magnetic flux is established within the motor. The same also can be achieved by replacing the rotor winding with permanent magnets. By changing the stator magnets with three phase windings, the commutation can be achieved electronically compared to the mechanical commutation in common DC motors. Such motors are called Brushless DC motors. As this type of construction eliminates the need of brushes, the maintenance is reduced and the reliability is increased. Figure 1 shows the stator and rotor arrangement in Brushed and Brushless DC motor.

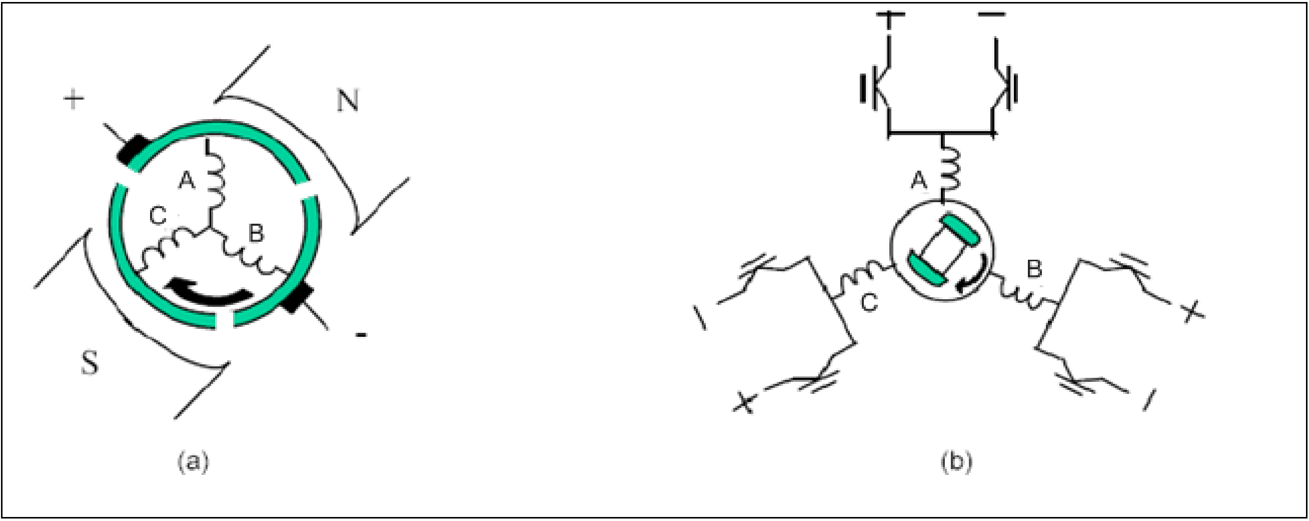


Figure 1 a) Brushed DC Motor, b) Brushless DC Motor

In case of Common DC motors, the brushes automatically will come into contact with the commutator of a different coil causing the motor to continue its rotation. But in the case of BLDC motors, the commutation has to be done through electronic switches which need the position of the rotor. The appropriate stator windings have to be energized when the rotor pole lines up with the winding. It is possible to drive a BLDC motor with the predefined commutation interval. But in order to achieve precise control of speed and maximum generated torque, electronic commutation should be done with known rotor position.

Most BLDC motors have internal sensors to provide rotor position information. Hall sensors are the most common type among them. Three phase motors typically have three Hall Sensors. Whenever the rotor magnetic poles pass near the Hall sensors, they give a high or low signal, indicating the N or S pole is passing near the sensor. Every Sensor outputs high level for 180 electrical degrees of electrical rotation and low level for 180 electrical degrees of electrical rotation. For a single pole pair machine, both electrical and mechanical degrees are same. For two pole machine, there are two electrical revolutions per mechanical revolution. In general, the relationship between mechanical and electrical degrees is as stated below.

Electrical revolution = Mechanical revolution / Pole pairs

Figure 2 shows a simplified representation of single polepair BLDC motor with Hall Sensors (H0, H1, H2).

The arrow represents the magnetic field of the rotor while the coils windings (A, B, C) represent the stator.

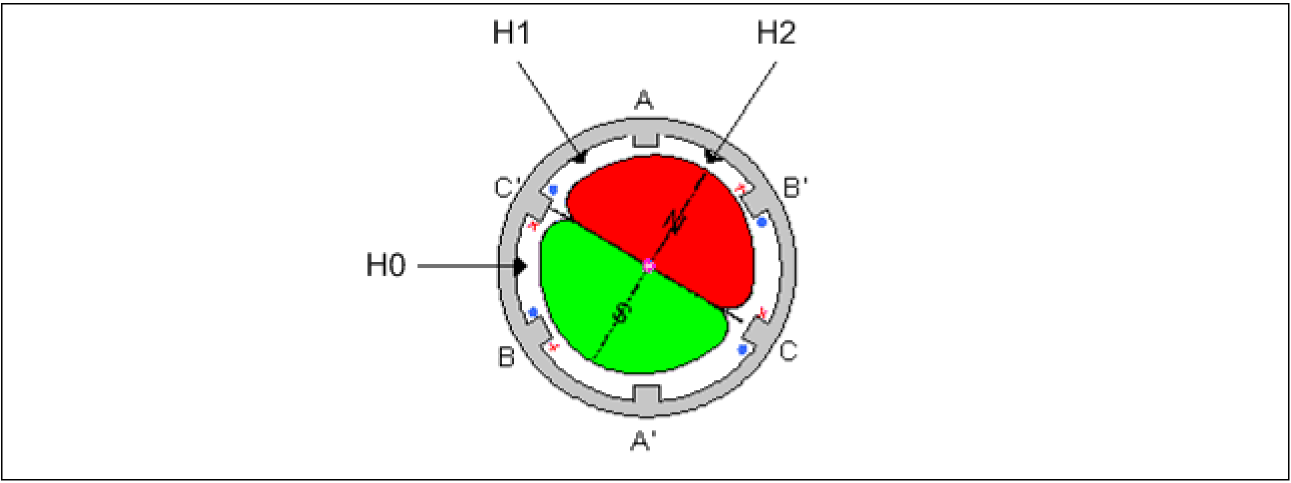


Figure 2 Single Pole pair BLDC motor with Hall Sensor

For every 60 electrical degrees of rotation, one of the Hall sensors changes its state and each combination of Hall sensors state represents a specific rotor position. For such each rotor position, the exact sequence of commutation can be determined.

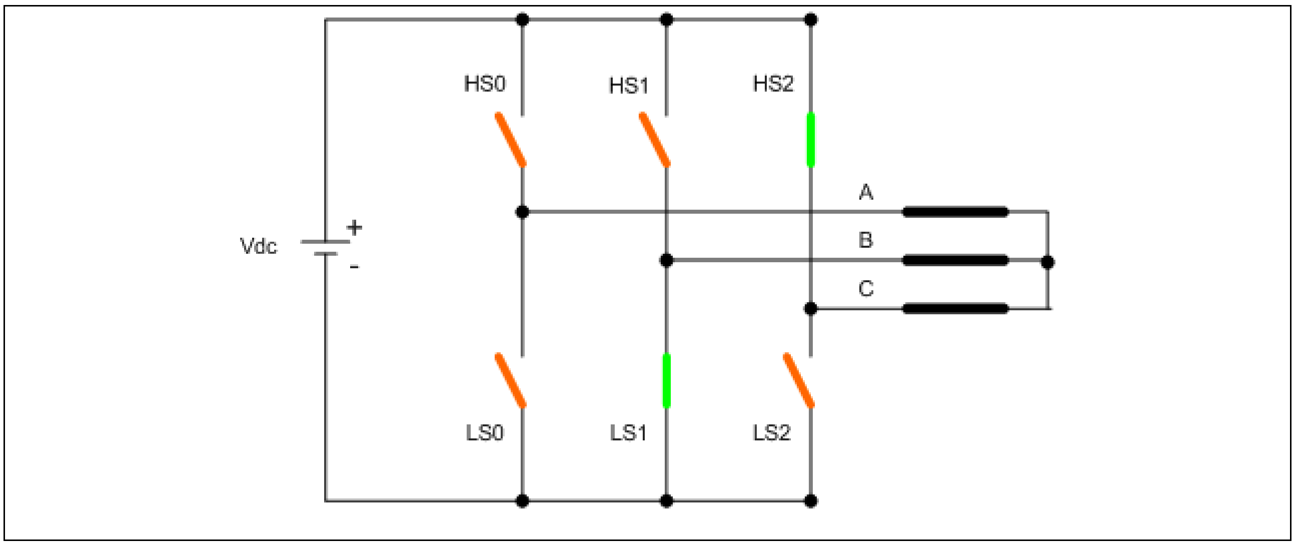


Figure 3 Motor phase winding connection with inverter switches

Each commutation sequence has one of the windings energized to positive power (current enters into the winding), the second winding is negative (current exits the winding) and the third is in a non-energized condition. Torque is produced because of the interaction between the magnetic field generated by the stator coils and the permanent magnets. Ideally, the peak torque occurs when these two fields are at 90 degrees to each other and falls off as the fields move together. In order to keep the motor running, the magnetic field produced by the windings should shift position as the rotor moves to catch up with the stator field. What is known as “Six-Step commutation or Block Commutation” defines the sequence of energizing windings.

Figure 3 shows the connection of motor phase winding with the inverter switches. At any point of time only two phases will be conducting the current. The closed circuit is formed by switching on a high side switch of one leg and a low side switch of another leg thereby establishing a path for the flow of current.

The Hall pattern is formed by arranging the Hall inputs in the order [H2 H1 H0].

Hall Pattern [H2 H1 H0]	1 0 0	1 0 1	0 0 1	0 1 1	0 1 0	1 1 0
C	+	0	-	-	0	+
B	-	-	0	+	+	0
A	0	+	+	0	-	-
HS0	Off	On	On	Off	Off	Off
HS1	Off	Off	Off	On	On	Off
HS2	On	Off	Off	Off	Off	On
LS0	Off	Off	Off	Off	On	On
LS1	On	On	Off	Off	Off	Off
LS2	Off	Off	On	On	Off	Off

Figure 4 Commutation Sequence for a BLDC Motor

Figure 4 shows a table formation which relates the rotor position and the commutation sequence. The " + " means High-side switch (HS) on and Low-side switch (LS) off, " - " means LS on and HS off. The " 0 " means both LS and HS are off.

2.2 Speed Control of Brushless DC Motor

In automotive applications most commonly used voltage levels are 48V, 24V and 12V. The motor manufacturer specifies the operating voltage level and speed range. The speed of the motor is directly proportional to the applied voltage. Using Pulse Width Modulation (PWM) by switching the transistors on and off, a varying average voltage can be applied to the motor. This average DC voltage determines the motor speed. When both the high side and low side transistors were permanently on for the 100% commutation period, the motor will run at rated speed provided the rated dc voltage is supplied. For operating the motor at a desired speed below the rated speed, the commutation pattern applied at either high side or low side should be pulse width modulated.

There are two control schemes are possible:

1. Open-loop speed Control (Voltage Control)
2. Closed-loop speed Control

In Open loop speed control, the duty cycle is calculated based on the set reference speed. In case of closed loop speed control the actual speed is measured and compared with the reference speed to find the error difference. This error difference will be supplied to the PI controller. The output from the PI controller gives the desired duty cycle.

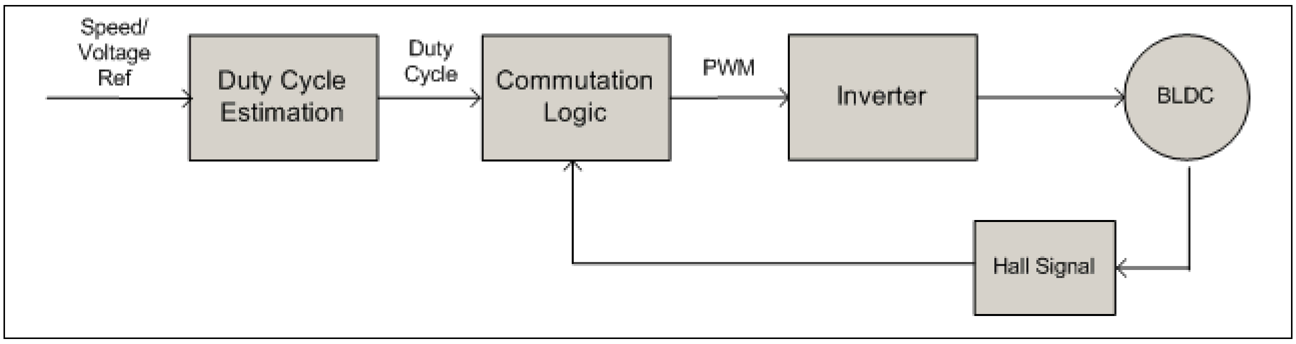


Figure 5 Open loop Speed (Voltage) Control

Figure 5 shows the open loop speed control of BLDC motor. The duty cycle for a set reference speed is estimated based on the nominal base speed of the motor.

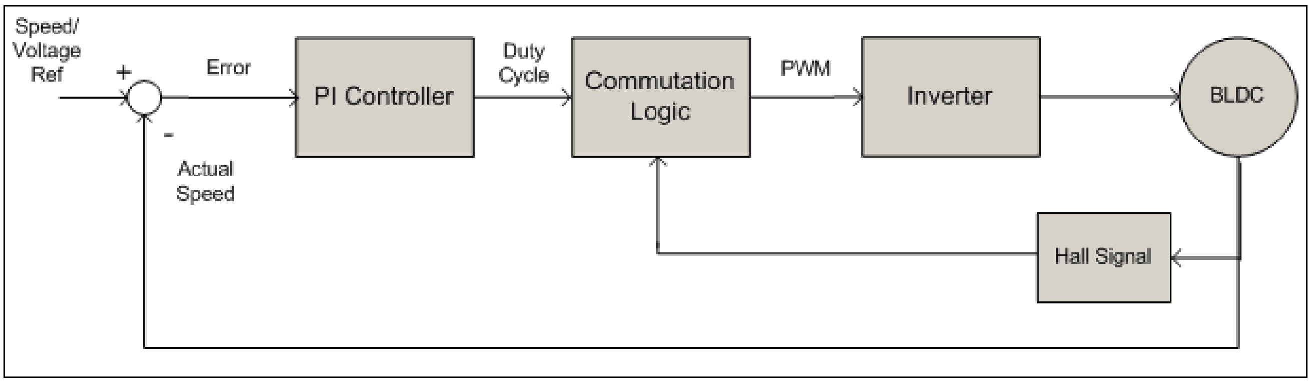


Figure 6 Closed loop Speed Control

Figure 6 depicts the closed loop speed control of the BLDC motor.

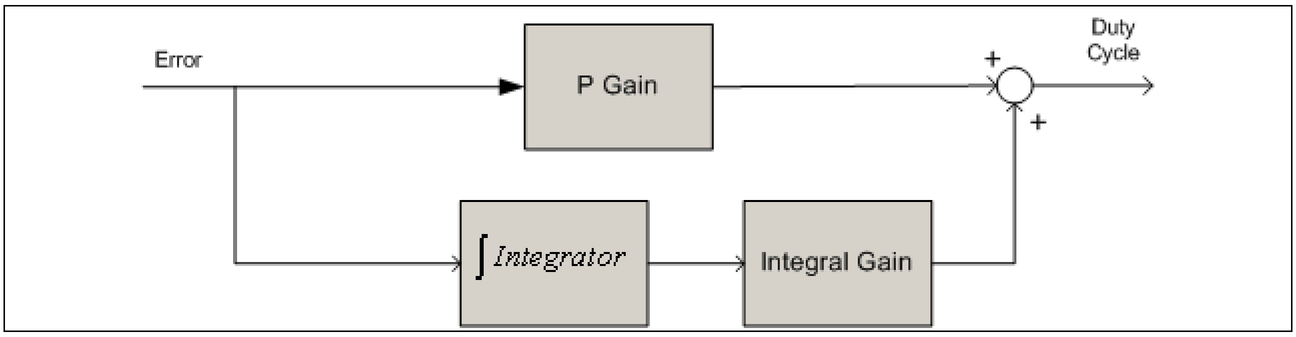


Figure 7 PI controller functional representation

A PI controller is used for regulating the speed. The error difference between the reference speed and the actual speed is fed to the controller.

The PI controller functionality is shown in Figure 7.

In continuous time domain, the duty cycle output is given by,

$$\text{Duty Cycle} = K_p * error + K_i * \int error * dt$$

In discrete time domain, the PI controller is implemented as described by the following equations.

$$\mathrm{yn}(k+1) = \mathrm{yn}(k) + \mathrm{Ki} * \mathrm{e}(k)$$
$$\mathrm{y}(k+1) = \mathrm{yn}(k+1) + K_p * \mathrm{e}(k)$$

Where,

K_i	=	Integral Gain
K_p	=	Proportional Gain
e(k)	=	Error value
y(k+1)	=	Next computed duty cycle
yn(k)	=	Integrated error value till last computation
yn(k+1)	=	Current Integrated error value

The actual K_p and K_i values are scaled and will be used in target as follows:

k_p	=	$K_p * 2^{15} / 64$
k_i	=	$K_i * 2^{15}$

Where,

k_p and k_i are the Scaled Proportional and Integral Gain values used in software.

To get as high accuracy, small round-off errors, as possible both input and output values are scaled to fill in available 16 bits. The actual values of K_p and K_i which are scaled for accuracy reasons were finally scaled down to actual values for the duty cycle calculation.

During the execution of the PI controller function, the actual values are recalculated for the duty cycle update. The implementation details are explained in section 7.2.4

3 Hardware Implementation with XC164CM/CS Microcontroller

Figure 8 depicts the hardware implementation of Hall sensored BLDC motor with the XC164CM/CS controller and BTS 7960 power driver board. The Hall signals of the motor are fed to the CCPOS0, CCPOS1 and CCPOS2 pins of the microcontroller. This microcontroller has a Capture/Compare Unit 6 (CAPCOM6) which generates the switching commutation pattern based on the input Hall pattern. The CCU6 also generates the PWM which is used for the speed or torque control. The controller generates the PWM through the CC6x andCOUT6x (x = 0,1,2) pins.

The 3-phase driver IC takes the switching patterns as input and output the signals for the 3-phase inverter. This driver IC is also capable of implementing short-circuit current protection, over/under voltage protection and over temperature protection by making use of the CTRAP functionality of the microcontroller. The CTRAP will force the CCU6 outputs into a passive state and no active modulation is possible, immediately stopping the motor operation.

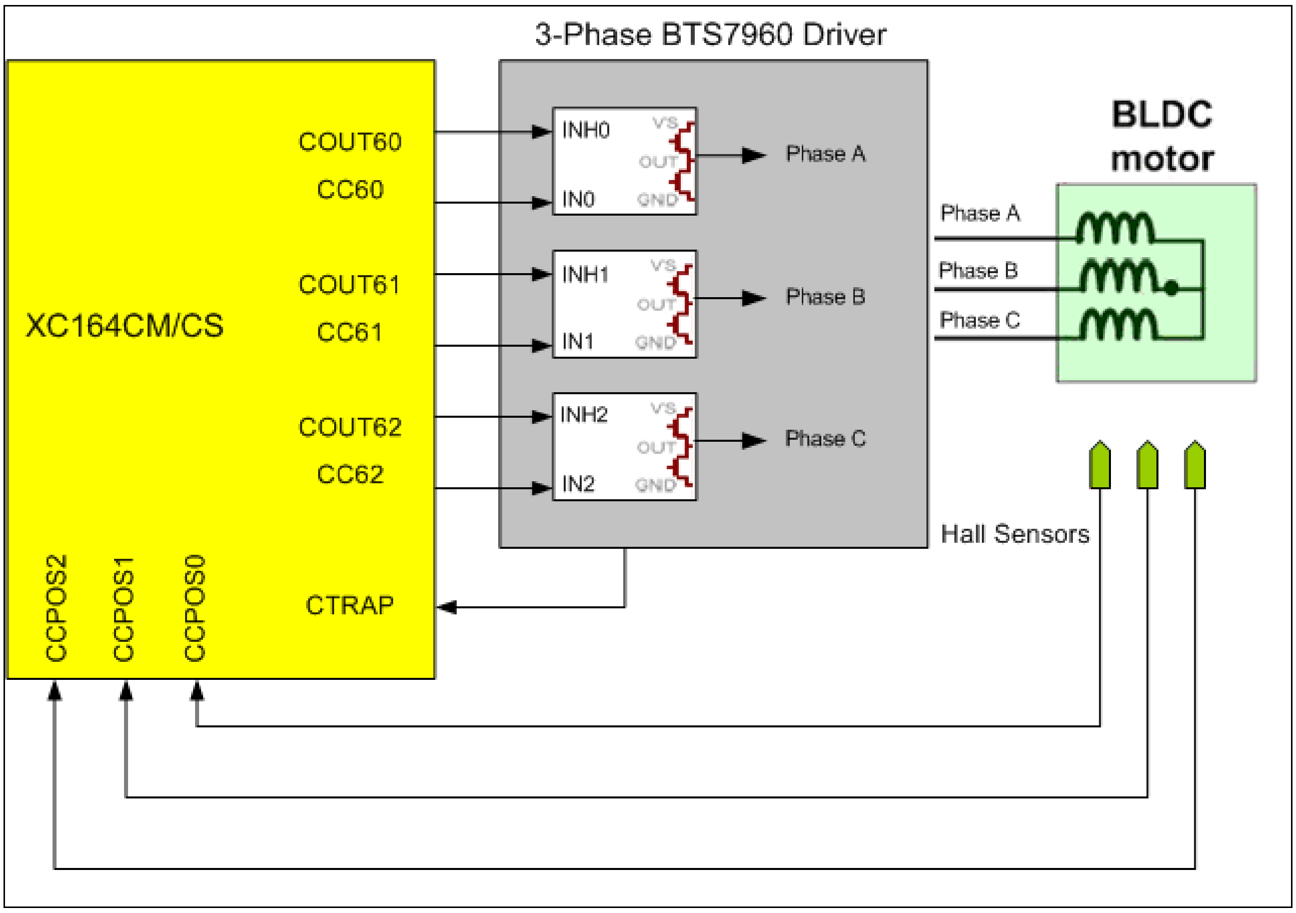


Figure 8 Hall sensored BLDC motor control with XC164CS/CM and Driver board

4 Motor Control with CAPCOM6

The functional representation of the CAPCOM6 unit is shown in Figure 9.

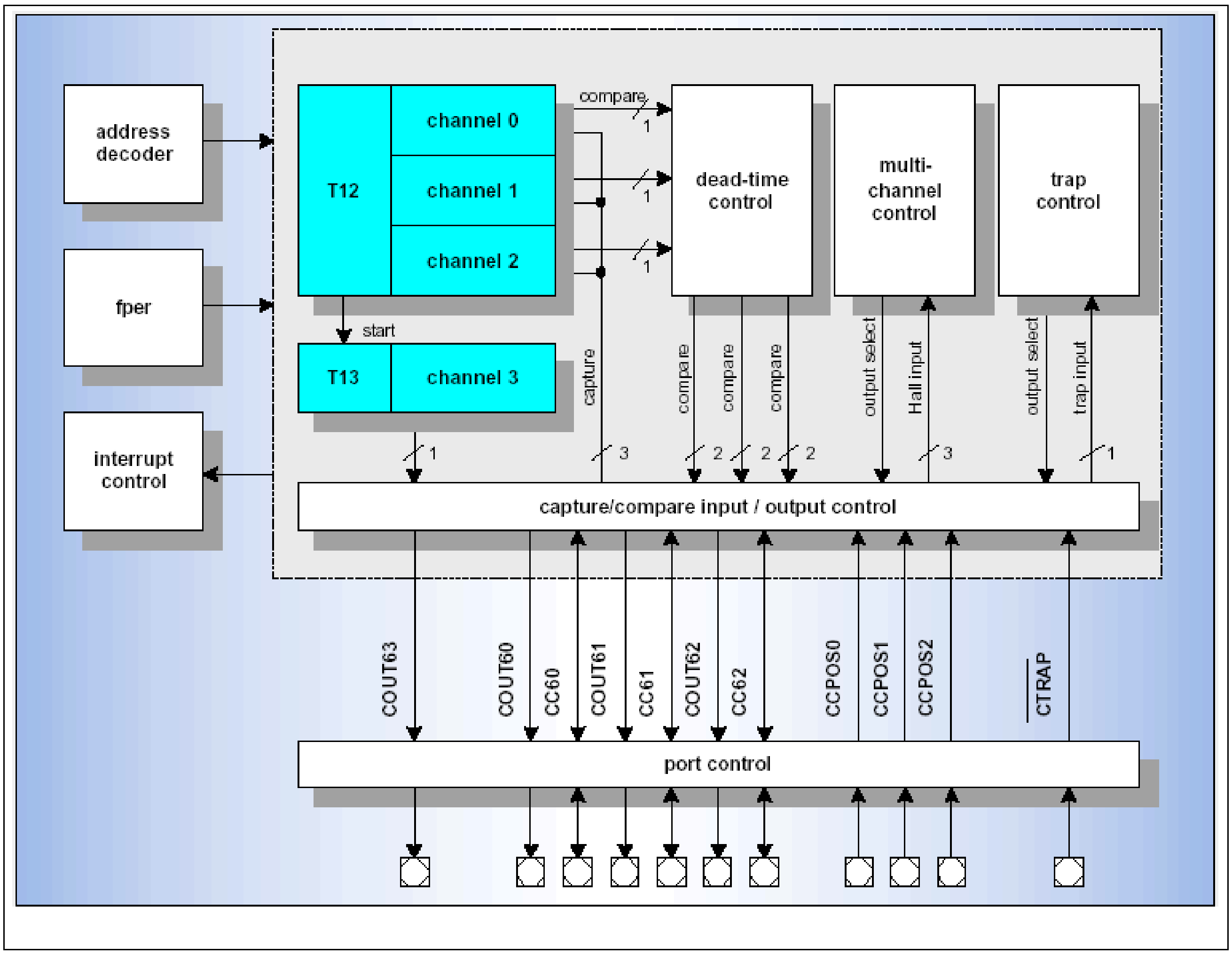


Figure 9 CAPCOM6 functional representation

The basic motor control can be implemented in three steps:

1. Sampling of Hall Pattern.
2. Hall Pattern and PWM output pattern software update.
3. Shadow transfer of updated patterns.

4.1 Sampling of Hall Pattern

The sampling of the Hall pattern (on CCPOSx) is done with the T12 clock. By using the dead-time counter DTC0 (mode MSEL6x= '1000') a hardware noise filter can be implemented to suppress spikes on the Hall inputs due to high di/dt in rugged inverter environment. In case of a Hall event the DTC0 is reloaded and starts counting. When the counter value of one is reached, the CCPOSx inputs are sampled (without noise and spikes) and are compared to the current Hall pattern (CURH) and to the expected Hall pattern (EXPH). If the sampled pattern equals to the current pattern the edge on CCPOSx was due to a noise spike and no action will be triggered (implicit noise filter). If the sampled pattern equals to the next expected pattern the edge on CCPOSx was a correct Hall event, the bit CHE is set which causes an interrupt.

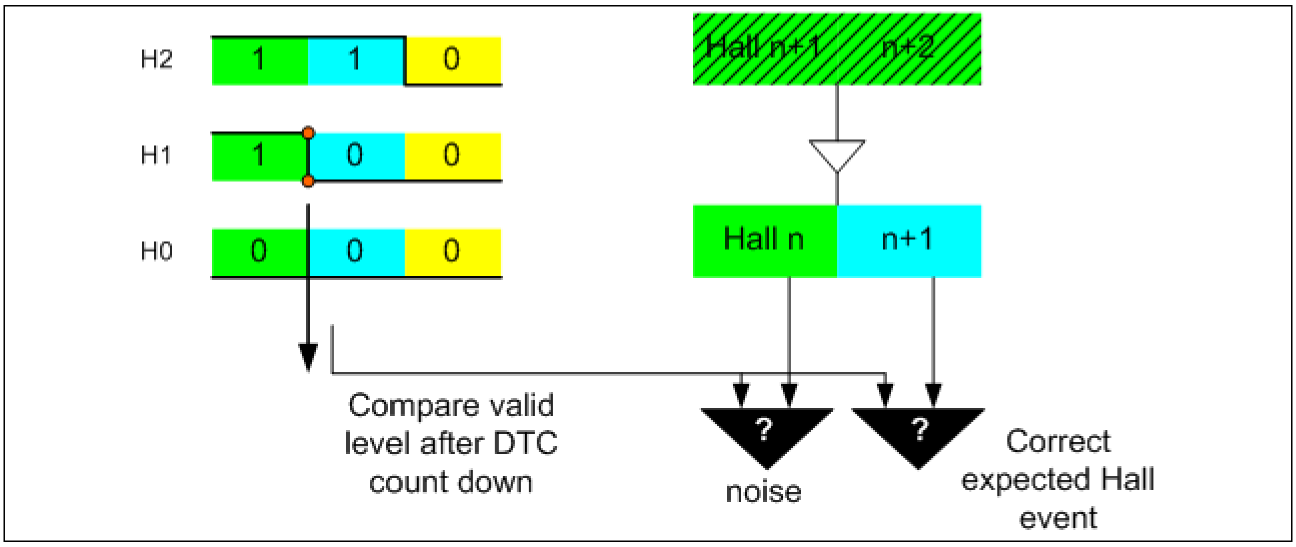

H2
1
1
0
H1
1
0
0
H0
0
0
0
Hall n+1
n+2
Hall n
n+1
Compare valid level after DTC count down
?
noise
?
Correct expected Hall event

Figure 10 Hall Pattern Sampling

With reference to Figure 10, the sampling of Hall pattern is illustrated as follows:

- The Sensor H1 changes it's level which is detected by the edge detection logic and this triggers a Downcounter.
- When the downcounter is finished it compares the hall sensor input levels (at CCPOSx-pins) with the value in this register MCMOUT.
- Now we have three possibilities:
  - The value equals the actual value n – it was just a noise spike – nothing happens.
  - The value equals to the following state n+1 – we have a correct hall event cause we would expect this state if the motor turns right.
  - If it was neither the actual nor the following state – it was a wrong hall event and we have to react by software cause the motor does not turn properly.

4.2 Hall Pattern and PWM Output Pattern Software Update

After the actual Hall pattern is sampled at input pins CCPOSx (x = 0,1,2), the corresponding output pattern of CCU6 should be generated in order to control BLDC motor. The CCU6 generates the switching pattern thru its output channels CC6x and COUT6x (x = 0,1,2). It provides two independent 16-bit timers (T12, T13); timer 13 generates the PWM which modulates DC rail voltage and hence motor speed. This PWM is delivered by the COUT6x output channels.

On detection of correct Hall event, CHE flag is set and hardware shadow transfer is triggered. This will copy the next state from the shadow register to the actual register. This is shown in figure 11.

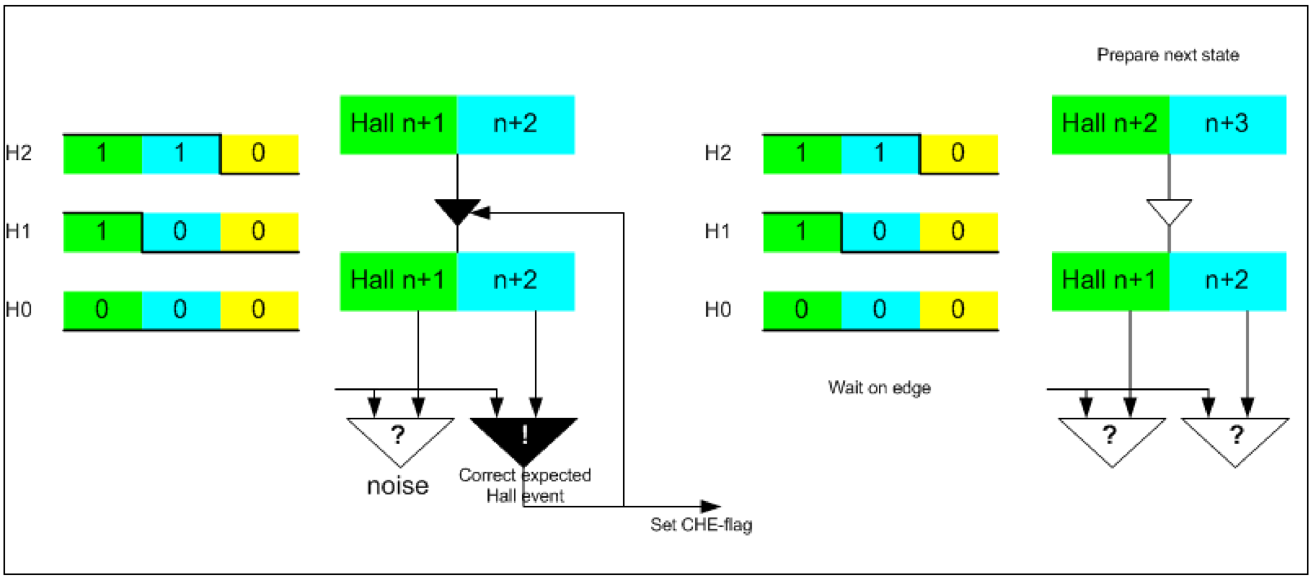


Figure 11 Correct Hall Event and Preparation for next edge detection

The hall pattern and PWM pattern are used by storing them in program as follows:

```
#define MCMOUTS_CTE_R(curhs,exphs,mcmps) (curhs << 11) | (exphs << 8) | (mcmps)
#define MCMOUTS_CTE_L(curhs,exphs,mcmps) (curhs << 11) | (exphs << 8) | (mcmps)

int const SP_St_Tab_r_l[16] =
{
0x0000,      MCMOUTS_CTE_R(1,5,0x32), MCMOUTS_CTE_R(2,3,0x2C),
             MCMOUTS_CTE_R(3,1,0x0E), MCMOUTS_CTE_R(4,6,0x0B),
             MCMOUTS_CTE_R(5,4,0x38), MCMOUTS_CTE_R(6,2,0x23), 0x0000,
0x0000,      MCMOUTS_CTE_L(1,3,0x23), MCMOUTS_CTE_L(2,6,0x38),
             MCMOUTS_CTE_L(3,2,0x0B), MCMOUTS_CTE_L(4,5,0x0E),
             MCMOUTS_CTE_L(5,1,0x2C), MCMOUTS_CTE_L(6,4,0x32), 0x0000
};
int const SP_Tab_r_l[16] =
{
0x0000,      MCMOUTS_CTE_R(5,4,0x38), MCMOUTS_CTE_R(3,1,0x0E),
             MCMOUTS_CTE_R(1,5,0x32), MCMOUTS_CTE_R(6,2,0x23),
             MCMOUTS_CTE_R(4,6,0x0B), MCMOUTS_CTE_R(2,3,0x2C), 0x0000,
0x0000,      MCMOUTS_CTE_L(3,2,0x0B), MCMOUTS_CTE_L(6,4,0x32),
             MCMOUTS_CTE_L(2,6,0x38), MCMOUTS_CTE_L(5,1,0x2C),
             MCMOUTS_CTE_L(1,3,0x23), MCMOUTS_CTE_L(4,5,0x0E), 0x0000
};
```

Here curhs refers to current hall pattern, exphs refers to expected hall pattern and mcmps refers to the modulation pattern. The modulation pattern in table SP_St_Tab_r_l is used while starting up of the motor and the pattern in table SP_Tab_r_l is used while the motor is running.

The usage of this table was explained in the next section.

4.3 Shadow Transfer of Updated Patterns

Although Hall pattern and output pattern are updated via software, they don't take effect immediately. A mechanism named Shadow Transfer will synchronize them with pre-defined event. This is because usually PWM outputs are used to drive high-voltage or high-current applications, and should be synchronized with certain hardware event due to safety reason. To implement this mechanism, some special function registers come in pairs: a shadow register and an actual one. Writing operation targets shadow registers and not directly to the actual registers, while the read access targets the registers actually used. The register MCMOUTS is the shadow register of MCMOUT. The values in the MCMOUTS register are transferred to the actual register, MCMOUT, when there is a correct hall event or other events such as:

- A T12 period-match while counting up (T12pm).
- A T12 one-match while counting down (T12om).
- A T13 period-match (T13pm).
- A T12 compare-match of channel 1 (T12c1cm).

The transfer can also be requested by software by setting the corresponding shadow transfer request bit: bit STRMCM and bit STRHP of register MCMOUTS. By using this, the update takes place completely under software control.

Once the updated values in shadow register are transferred to the actual register, the next set of rotor position pairs and CCU6 output pattern are loaded into the MCMOUT register. After that, the MCMOUT controls the CCU6 output channels and these output channels decide which of the inverter switches are activated.

The switching selection and synchronization of modulation pattern is shown in figure 12.

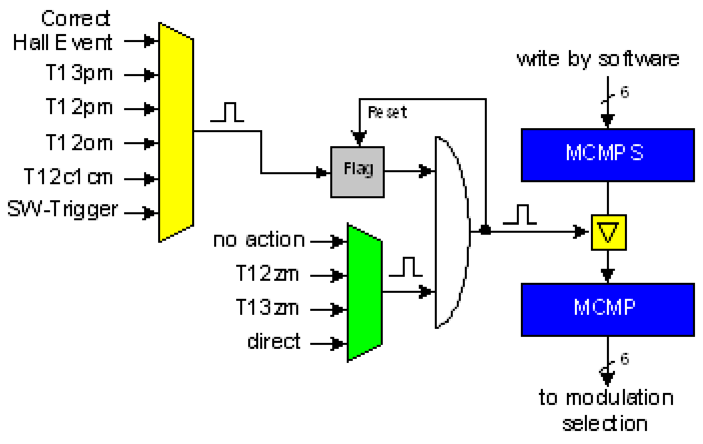


Figure 12 Modulation and Synchronization

During commutation initialization, the hall input pins of the microcontroller were read and the corresponding PWM pattern was updated to the actual register through software by setting the STRHP and STRMCM flags in the MCMOUTS register.

```
l_aux = (P1H & 0x0007) + g_direction;
CCU6_MCMOUTS = SP_St_Tab_r_l[l_aux] | 0x8080;
```

The current hall input was used as the index of the table SP_St_Tab_r_l. The microcontroller is now ready to deliver the PWM pattern as stored in the MCMOUT register. Now the Shadow register should be loaded with the next CURH, EXPH and PWM pattern. Though the same table can be used for loading the shadow register, the logic used here is always take the current hall pattern value to update the shadow register. Hence the same start up table (SP_St_Tab_r_l) was arranged such that the current hall pattern was read from the actual MCMOUT register and will be used as the index of the SP_Tab_r_l table.

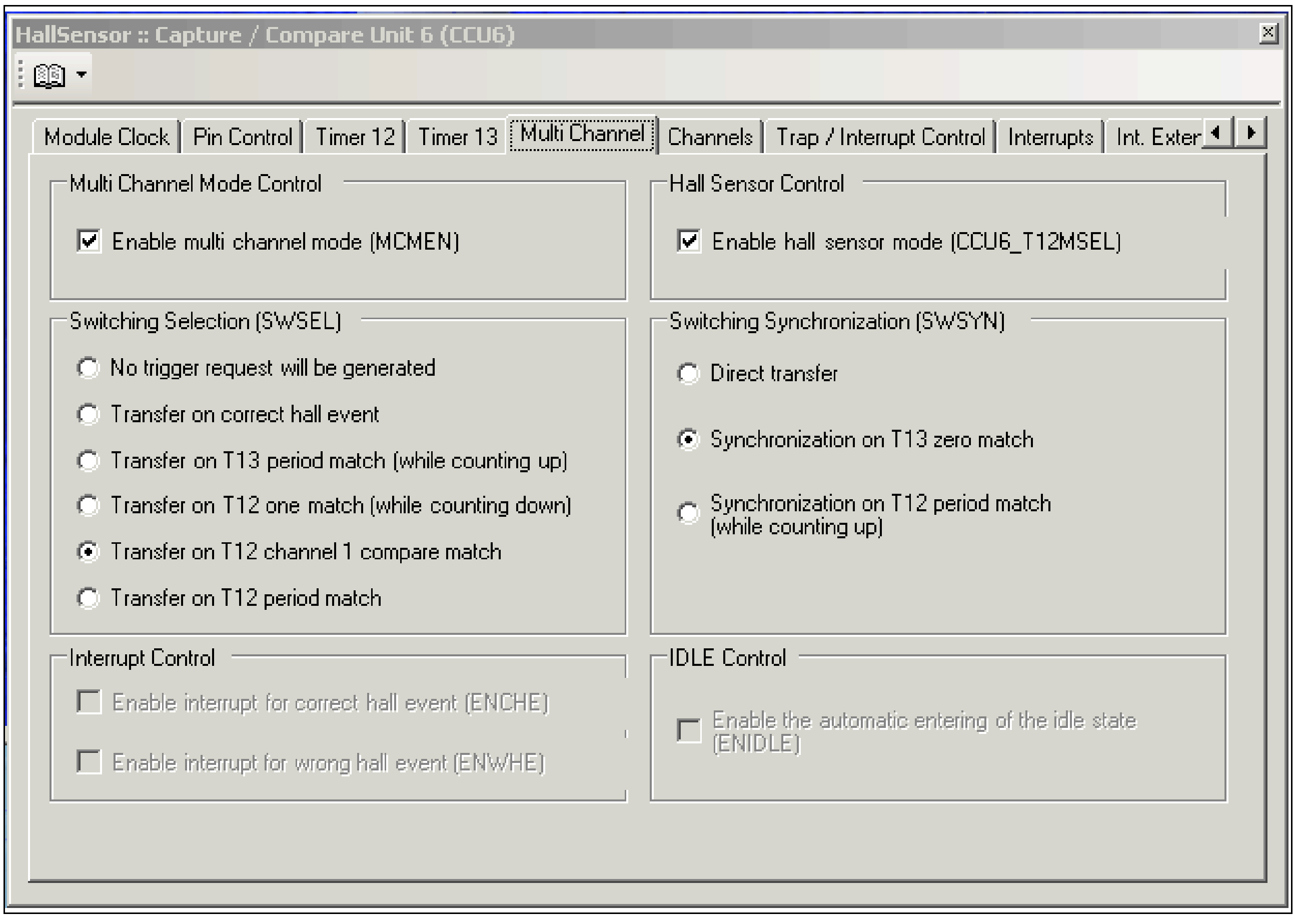
In commutation initialization, the usage is

```
CCU6_MCMOUTS = SP_Tab_r_l[l_aux];
```

During CCU6 channel 2 compare match, the usage is

```
CCU6_MCMOUTS = SP_Tab_r_l[((CCU6_MCMOUT & 0x3800) >> 11) + g_direction];
```

The DAvE configuration for the Hall sensor mode selection, MCMOUT switching selection and synchronization selection is shown in figure 13.


HallSensor :: Capture / Compare Unit 6 (CCU6)

Module Clock | Pin Control | Timer 12 | Timer 13 | Multi Channel | Channels | Trap / Interrupt Control | Interrupts | Int. Exter

Multi Channel Mode Control
- [✓] Enable multi channel mode (MCMEN)

Hall Sensor Control
- [✓] Enable hall sensor mode (CCU6_T12MSEL)

Switching Selection (SWSEL)
- ○ No trigger request will be generated
- ○ Transfer on correct hall event
- ○ Transfer on T13 period match (while counting up)
- ○ Transfer on T12 one match (while counting down)
- ● Transfer on T12 channel 1 compare match
- ○ Transfer on T12 period match

Switching Synchronization (SWSYN)
- ○ Direct transfer
- ● Synchronization on T13 zero match
- ○ Synchronization on T12 period match (while counting up)

Interrupt Control
- [ ] Enable interrupt for correct hall event (ENCHE)
- [ ] Enable interrupt for wrong hall event (ENWHE)

IDLE Control
- [ ] Enable the automatic entering of the idle state (ENIDLE)

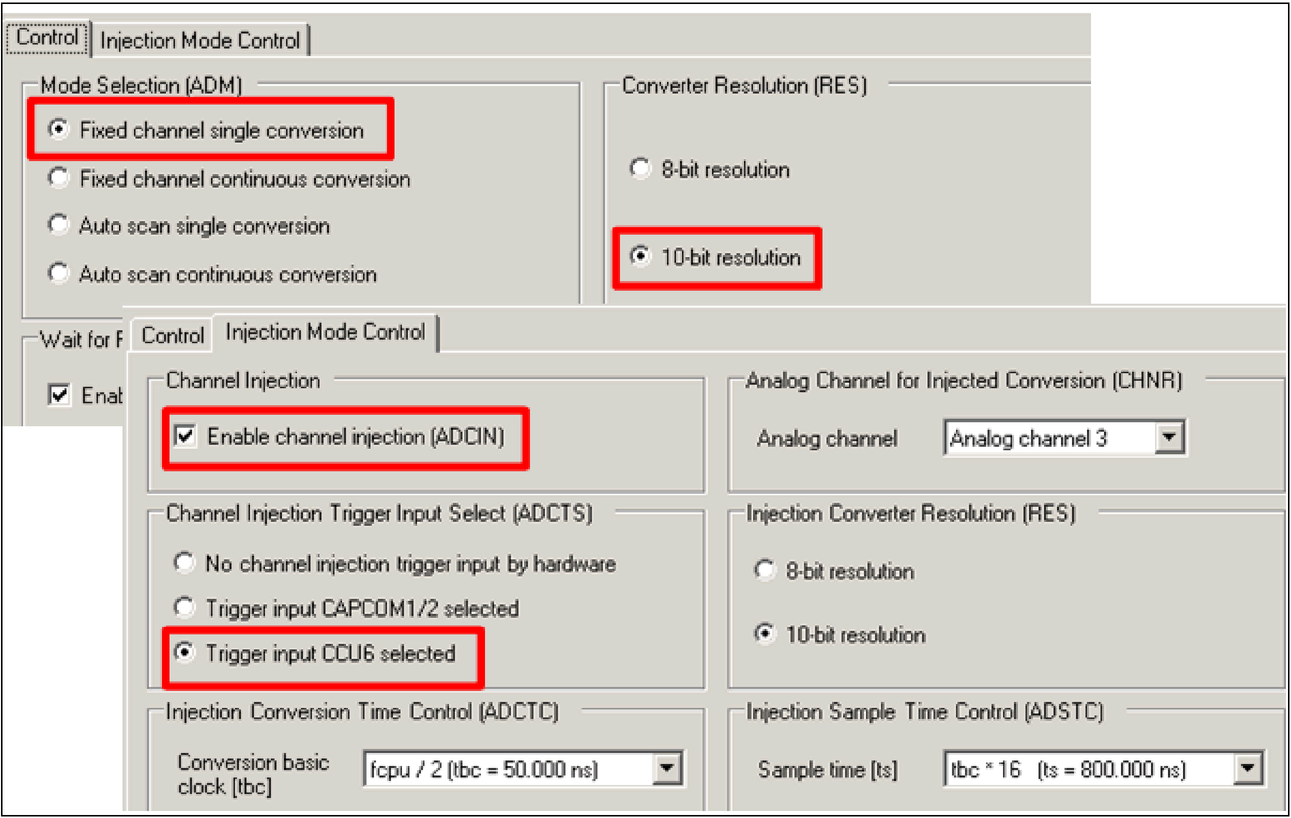
Figure 13 Multi Channel Mode Control configuration in DavE

5 ADC for Current Measurement

For analog signal measurement, the analog to digital converter of XC164CS/CM microcontroller has a programmable 8bit/10bit conversion resolution with 14-multiplexed input channels and a sample and hold circuit. It uses the method of successive approximation.

The A/D converter of XC164 supports four different conversion modes to meet the embedded application needs. The peripheral supports conversion modes such as fixed channel conversion, autoscan conversions or channel injection modes. For the current measurement in Hall Mode, only single channel is used for current measurement. Here the fixed channel single conversion with channel injection mode is selected. Also the channel injection is triggered by the period match of timer T13 of CAPCOM6E unit.

Figure 14 shows the ADC configuration settings in DAvE.



Control | Injection Mode Control

Mode Selection (ADM)
[✓ Fixed channel single conversion]
Fixed channel continuous conversion
Auto scan single conversion
Auto scan continuous conversion

Converter Resolution (RES)
8-bit resolution
[✓ 10-bit resolution]

Wait for F
[✓ Enat]

Control | Injection Mode Control

Channel Injection
[✓ Enable channel injection (ADCIN)]

Analog Channel for Injected Conversion (CHNR)
Analog channel: Analog channel 3

Channel Injection Trigger Input Select (ADCTS)
No channel injection trigger input by hardware
Trigger input CAPCOM1/2 selected
[✓ Trigger input CCU6 selected]

Injection Converter Resolution (RES)
8-bit resolution
[✓ 10-bit resolution]

Injection Conversion Time Control (ADCTC)
Conversion basic clock [tbc]: fcpu / 2 (tbc = 50.000 ns)

Injection Sample Time Control (ADSTC)
Sample time [ts]: tbc * 16 (ts = 800.000 ns)

Figure 14 ADC configuration in DAvE

6 Measurement and Control Implementation

6.1 Output Voltage Normalisation

The normalization of the output voltage is calculated as follows:

$$K_v = V_{dcmax} * f_{pwm} * 2^{15} / f_{cpu}$$

e.g.: $K_v = 12 * 20 * 10^3 * 2^{15} / (40 * 10^6) = 196.61V$

6.2 Current Normalisation

With Driver board BTS7960 the current is measured through shunt resistance R34. This is shown in figure 15.

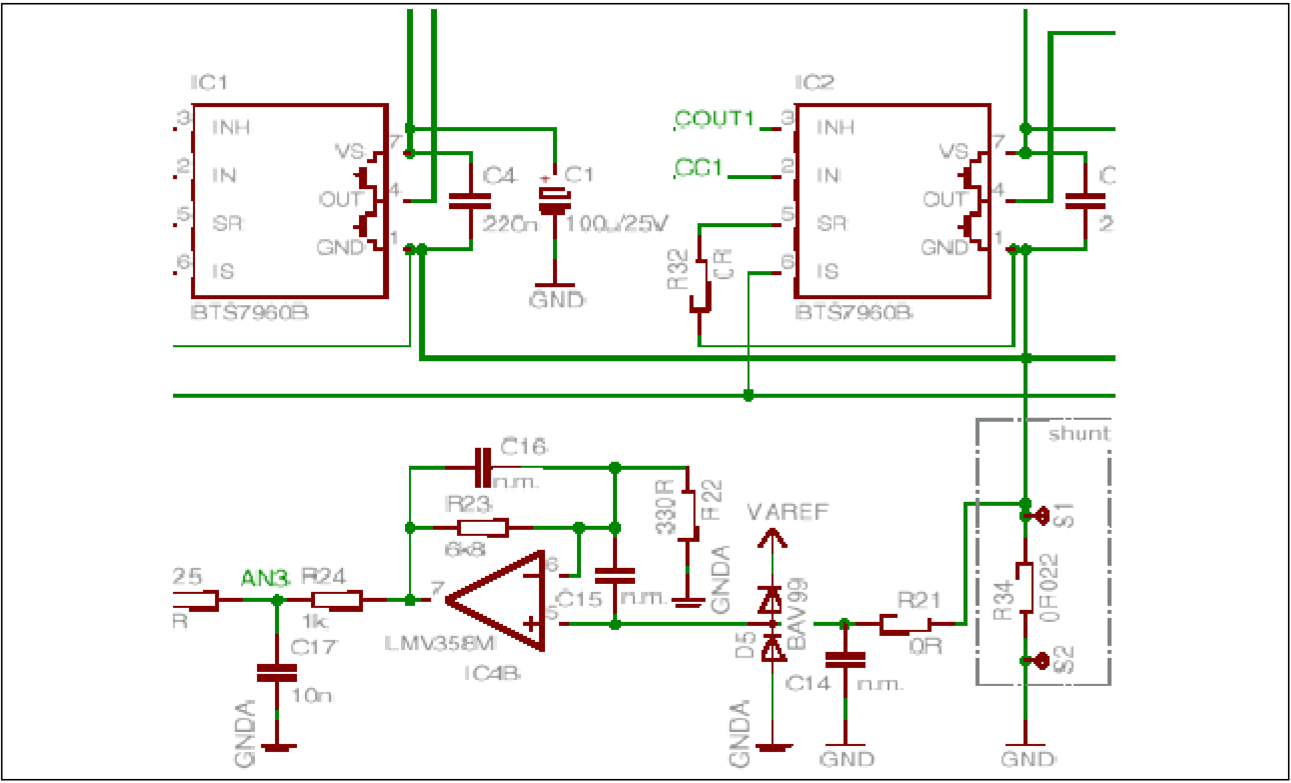


Figure 15 Current measurement with BTS7960 Driver board

The normalization for current is calculated as follows:

The Current sense ratio, I_{sr} $= \text{R34 * Current Amplifier Gain} = R34 * \left(1 + \frac{R23}{R24}\right)$

$= 0.022 * (1 + 6800/330) = 0.475333$

Now, the normalized current $K_c = \frac{V_{outAmp}}{I_{sr}} = \frac{5}{0.475333} = 10.518 \text{ A}$

6.3 Speed Normalisation

The scaling factor for speed is calculated as follows:

K_{Hall} = Speed resolution * 2^{15} ; Speed resolution = Nominal Speed * f_{pwm}/f_{cpu}

The chosen scaling factors influences the calculation of controller parameters. Changes in resolution will also change the controller parameters.

6.4 Reference Values Calculation

Reference values for voltage, current and speed are to be passed to the microcontroller in 1Q15 format, i.e. bit15 is the sign bit and bit14-bit0 refers to the value. The equation for scaling is:

$$X' = \frac{X}{K} * 2^{15}$$

X´ scaled 16bit integer
X physical value
K scaling factor (maximum physical value)
The scaling factor is the maximum physical value usable in the microcontroller without overflow.

Parameter Identification: **g_nv_start_ref**:

For Voltage Control (Open loop speed control):

'V_start_reference'[V] is the starting reference value of the voltage ramp in volt (set by user)

'V_start_reference' Range [0 - VDCmax] $$g\_nv\_start\_ref = \frac{V\_start\_reference}{K_v} * 2^{15}$$

For Speed Control (Closed loop speed control):

'Speed_start_reference'[V] is the starting reference value of the voltage ramp in volt (set by user)

'Speed_start_reference' Range [0 - K_{Hall}] $$g\_nv\_start\_ref = \frac{Speed\_start\_reference}{K_{Hall}} * 2^{15}$$

Parameter Identification: **g_nv_end_ref**:

For Voltage Control (Open loop speed control):

'V_end_reference'[V] is the final reference value of the voltage ramp in volt (set by user)

'V_end_reference' Range [0 - VDCmax] $$g\_nv\_end\_ref = \frac{V\_end\_reference}{K_v} * 2^{15}$$

For Speed Control (Closed loop speed control):

'Speed_end_reference'[V] is the final reference value of the voltage ramp in volt (set by user)

'Speed_end_reference' Range [0 - K_{Hall}] $$g\_nv\_end\_ref = \frac{Speed\_end\_reference}{K_{Hall}} * 2^{15}$$

Parameter Identification: **Ramp_up_time**:

'Ramp_time'[V] is the final reference value of the voltage ramp in volt (set by user)

'Ramp_time' Range [0 – 10^6] $$Ramp\_up\_time = Ramp\_time * f_{cpu} * 2^6$$

7 Software Implementation

7.1 Source Files

Table lists the source files that are generated by DAvE Drive for motor control. The table comprises also some their possible contents and some remarks.

Table 1 Generated Source Files

File	Possible Contents	Remarks
ADC.C, ADC.H	HW initialization for ADC ADC error interrupt after injected conversion for ADC readout and some time-critical actions ADC related definitions	
ASC0.C, ASC0.H	HW initialization for ASC0 ASC0 receive interrupt ASC0 related definitions	Files only generated if runtime communication is enabled.
CCU6.C, CCU6.H	HW initialization for CCU6 CTRAP interrupt for shutdown T12 period match interrupt for shutdown CC62 rising edge interrupt for BEMF speed calculation T13 period match interrupt for some PWM synchronous actions CCU6 related definitions	
COMM.C, COMM.H	Initialization for Runtime Control Panel ASC0 receive interrupt for Runtime Control Panel Runtime Control Panel related definitions	Files only generated if runtime communication is enabled.
COMPILER.H	Compiler related definitions	
CONFIG.H	Configuration related definitions	
CONTROL_LIB.H	Application-specific include files	
CTRL.C, CTRL.H	Subroutines for motor control, e.g. commutation Motor control related definitions	
DRIVE.C, DRIVE.H	Startup for drive program Drive related definitions	
IO.C, IO.H	Initialization for IO (input/output lines) IO related definitions	
MAIN.C, MAIN.H	HW and SW initialization Endless loop for non time-critical actions Definitions related to MAIN	
MCU.H	MCU related definitions	
PI_CTRL.C, PI_PARAMETERS.H	PI Control program PI Control related definitions	Files only generated if PI controller is used.
START.ASM	C startup routine	

7.2 Implementation of Software

The software is divided into several routines:

Main loop:

- Initialization (CPU, I/O ports, CAPCOM6, ADC, ASC0 (Run time control panel communication)
- Start, motor ramp generation, Stop, communication (Run time control panel communication)

Interrupt routines:

- CAPCOM6
    - Timer T13
    - Error Handling
    - Speed measurement
- ADC
- ASC0 (Run time control panel communication)

7.2.1 General Description

All peripherals are initialised during the execution of MAIN_viInit function. The start function initializes the motor start operation with commutation logic related timer, modulation, compare registers, software variables etc., The motor ramp function is called continuously till the ramp value reach the set reference value. The stop function is used to stop the motor in a normal way. This function is called whenever a manual stop operation performed or error detected during the motor is running. The communication related initialization is used for performing the monitor and control operation during the motor is running or in stopped condition.

The CAPCOM6 related interrupt routines are used for speed measurement, duty cycle calculation using PI control and for error detection.

7.2.2 Usage of CAPCOM6 Functionality

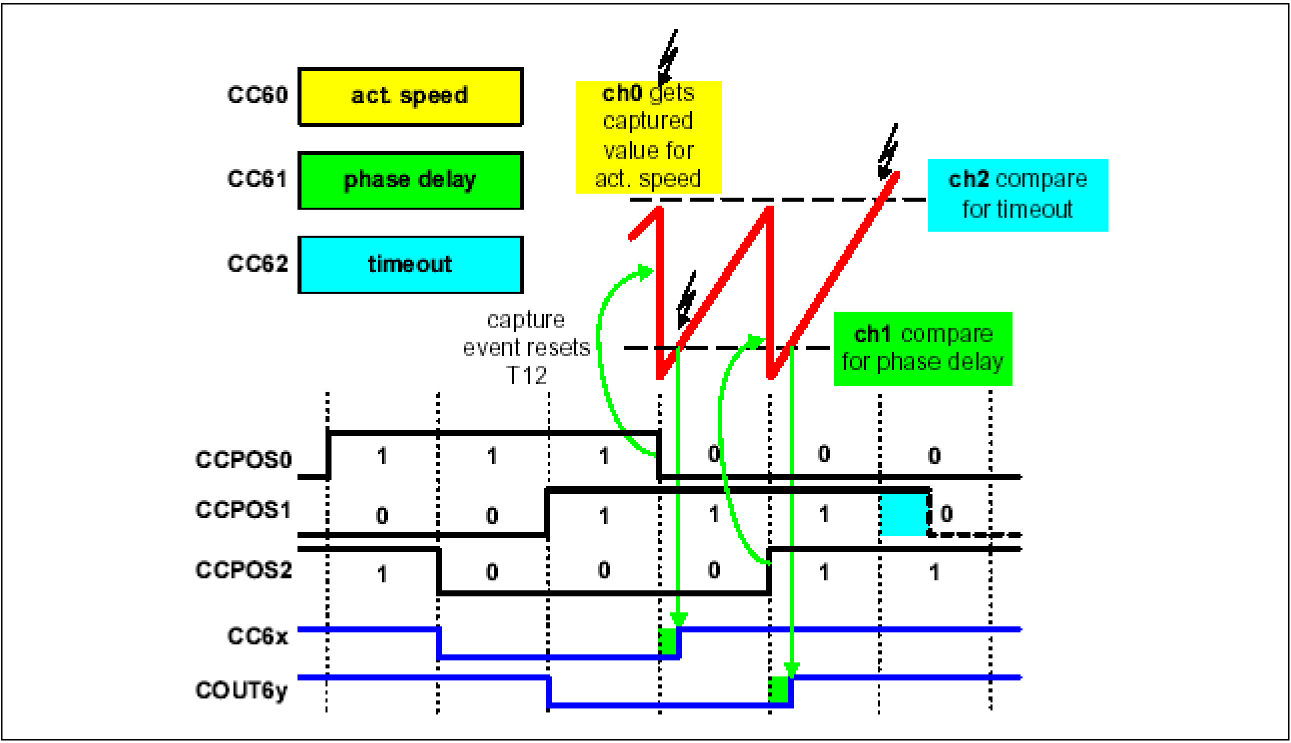


Figure 16 Hall Sensor Mode (MSEL6x = '1000b')

For Brushless-DC motors there is a special mode (MSEL6x = '1000b') which is triggered by a change of the Hall-inputs (CCPOSx). This is shown in figure 16. Here T12's channel 0 acts in capture function; channel 1 and 2 in compare function (without output modulation) and the multi-channel-block is used to trigger the output switching together with a possible modulation of T13. After the detection of a valid Hall edge the T12 count value is captured to channel 0 (representing the actual motor speed) and resets the T12. When the timer reaches the compare value in channel 1, the next multi-channel state is switched by triggering the shadow transfer of bit field MCMP (if enabled in bit field SWEN). This trigger event can be combined with several conditions which are necessary to implement a noise filtering (correct Hall event) and to synchronize the next multi-channel state to the modulation sources (avoiding spikes on the output lines). This compare function of channel 1 can be used as a phase delay for the position input to the output switching which is necessary if a sensorless back-EMF technique is used instead of Hall sensors. The compare value in channel 2 can be used as a time-out trigger (interrupt) indicating that the motors destination speed is far below the desired value which can be caused by an abnormal load change. In this mode the modulation of T12 has to be disabled (T12MODENx = '0').

7.2.3 Speed Calculation Function

The speed calculation needs the time between two Hall Sensor events. The time will be ascertained by Timer12 (CAPCOM6). On every correct Hall event, the Timer T12 will be reset and the value of Timer T12 will be saved in register CCU6_CC60R.

To reduce the measurement errors from Hall Sensors the time between two Hall sensor events is averaged over (6 * Polepairs) measured values. The sum of the measured values is divided by 2x (it is easier to implement) (2x > (6 * Pp)). The step size Δt_{T12} will determine the minimum speed and the time constant of the filter for the Hall Sensor signals.

t_n = CCU6_CC60R

$$t'_{Hall} = \frac{t_{(n+0)} + \ldots + t_{[n+(6*Pp)-1]}}{2^x}$$

The speed is calculated by the following equation:

$$n_{Hall} = \frac{60U / \min}{t'_{Hall} * \Delta t_{T12} * 2^x} \qquad \Delta t_{T12} = 1.6\mu s; 3.2\mu s; 6.4\mu s$$

Δt_{T12} determine the resolution for high speed and the value of minimum measurable speed. The delay of the speed calculation T_D depends upon the time T_H between two Hall Sensor events. The higher the speed, the speed will be calculated more often.

$$T_H = t_n * \Delta t_{T12} \qquad T_D = T_H * 6 * P_p$$

The speed calculation is getting executed on CCU6 channel 2 compare match.

The values of time measurement will be saved in a circular memory. Because of the continuous integration of the measured values, the oldest value must be subtracted to get always (6*Pp) values in the accumulator. There are used (6*Pp) values to minimize errors from the Hall Sensors. As next there will be executed the average calculation and speed calculation. Counter counts the Hall Sensor events. If counter has counted a certain number of events (e.g. 100), the average value from speed will be used for speed calculation. Before it the speed will be calculated directly from the measured values.

7.2.4 Implementation of PI Controller

The PI controller functionality implementation was done in assembly and used the MAC unit functionality of the microcontroller. The MAC unit provides single-instruction-cycle, non-pipelined, 32-bit additions, 32-bit subtraction, left and right shifts, 16-bit by 16-bit multiplication and multiplication with cumulative subtraction/addition. The PI implementation in assembly with the MAC functionality has the high calculation power. For a 40MHz CPU clock the duty cycle calculation will be performed in ~1 µs.

The MAC unit includes the following major components.

- The 16-bit by 16-bit Signed/Unsigned Multiplier and Scaler
- Concatenation Unit
- One-bit Scaler
- The 40-bit Adder/Subtracter
- The Data Limiter
- The Accumulator Shifter
- The 40-bit Signed Accumulator Register
- The Repeat Counter

To implement the PI-controller functionality in software, the analog differential equation has to be modified into a time discrete differential equation. There is no prediction of the system parameters, that's why the discrete function has to be transformed into a recursive function. That means you calculate the actual controller output by means of the last output and the actual input of the controller.

Refer section 2.2 for the discrete function implementation.

The PI controller parameters are given to the function over structure, which has the following form:

```
struct pi_controller                        // PI-controller parameter
{
    long yn;                                // PI integral buffer
    int  kp;                                // Proportional constant
    int  ki;                                // Integral constant
    int  ymax;                              // Maximal ouput limit
    int  ymin;                              // Minimal ouput limit
};
struct pi_controller C_SDATA g_pi_array;
```

The reference value and the actual value are given directly to the PI controller function. The values are represented in 1Q15 format.

```
int pi_controller64(long *pi_parameter,int reference,int actual)
{

#pragma asm

        mov     R12,MCW         ;Save MCW register
        mov     MCW,#1536       ;Set saturation and shift left
        mov     R11,ZEROS       ;Load zero in R11
        CoLOAD  R11,R9          ;Load Accumulator (High) with R9 (reference)
        CoSUB   R11,R10         ;error = reference - actual
        CoSTORE R9 ,MAS         ;Load error in R9
        mov     R3,[R8+]        ;
        CoLOAD  R3,[R8+]        ;Load yn (integral buffer) in accumulator
        mov     R1,R8           ;Save parameters addres in R1
        mov     R5,[R1+]        ;Load Kp (proportional Constant) in R5
        mov     R6,[R1+]        ;Load Ki = T0/Ti (integral Constant) in R6
        CoMAC   R6,R9           ;yn = Ki * error + yn
        mov     R6,[R1+]        ;Load ymax (limit value max)
        mov     R7,[R1+]        ;Load ymin (limit value min)
        CoMIN   R11,R6          ;Limit max yn
        CoMAX   R11,R7          ;Limit min yn
        CoSTORE R4,MAH          ;Store yn-high in R4
        CoSTORE R3,MAL          ;Store yn-low  in R3
        mov     [-R8],R4        ;Store R4 in integral buffer(High)
        mov     [-R8],R3        ;Store R3 in integral buffer(Low)
        CoMUL   R5,R9           ;Kp * error
        CoSHL   #6              ;64 * Kp * error
        CoADD   R3,R4           ;y = yn + (64 * Kp * error)
        CoMIN   R11,R6          ;Limit max y
        CoMAX   R11,R7          ;Limit min y
        CoSTORE R4,MAS          ;Store y-high in R4 (return register)
        mov     MCW,R12         ;Restore MCW register

#pragma endasm
}
```

8 Conclusion

This application note aims to provide the details of the BLDC motor working principle and explains how to drive such motor. This software solution consumes only very limited CPU resources because of the high performance microcontroller and its dedicated peripherals for BLDC motor control. Also with DAvE Drive the user can quickly start to work with BLDC motor control with the hardware. The important features of the microcontroller peripherals and how to make use of these features for the driving logic of motor also discussed.

Appendix A FLOWCHARTS

Flow chart of Main function

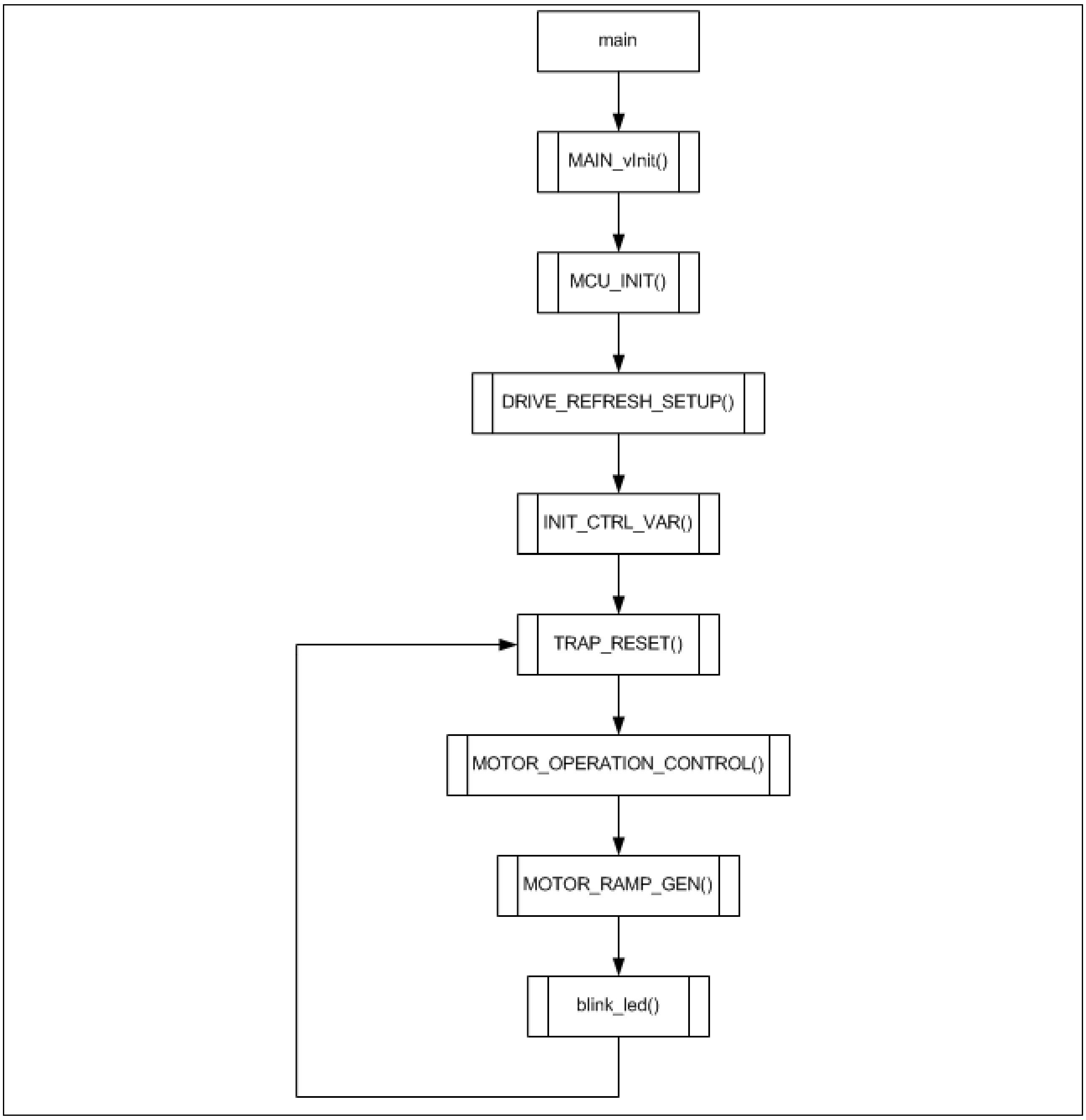


Figure 17 Flow chart of Main Function

Flow Chart of MAIN_vInit function

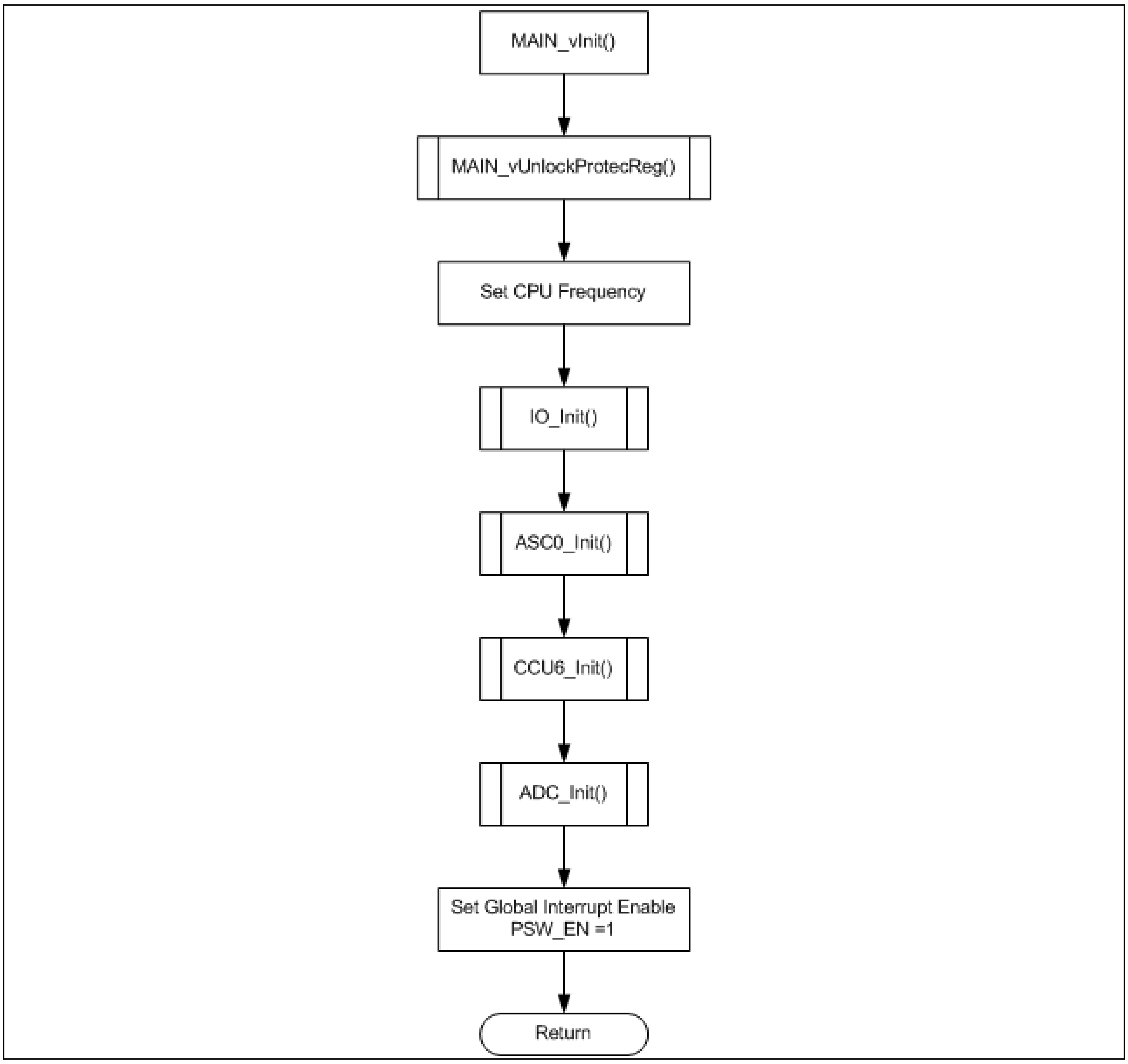


Figure 18 Flowchart of MAIN_vInit function

Flowchart of TRAP_RESET function

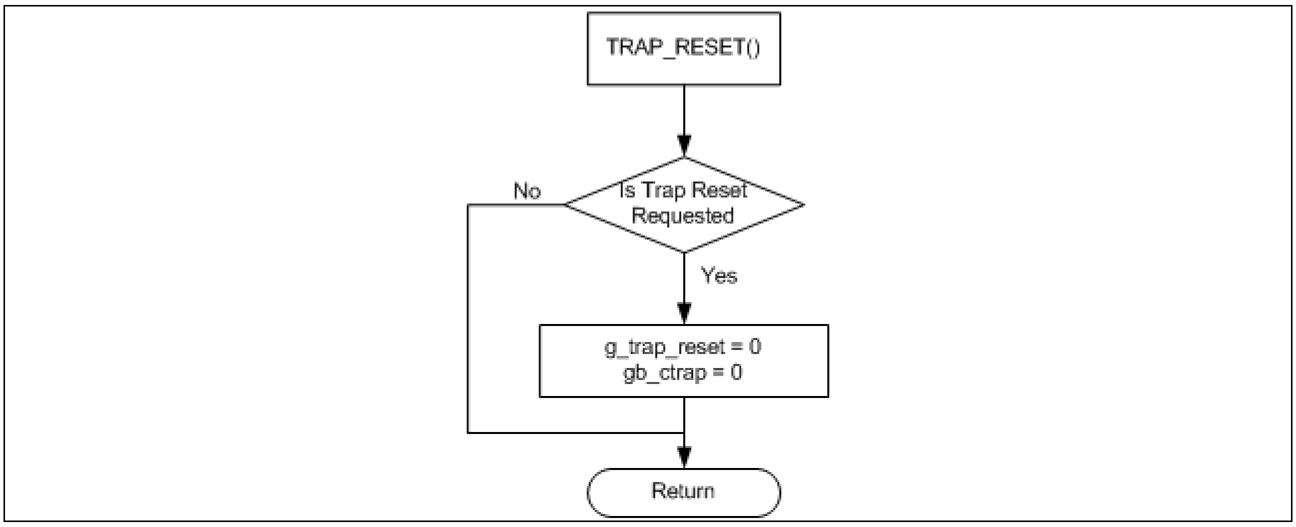


Figure 19 Flowchart of TRAP_RESET function

Flowchart of MOTOR_OPERATION_CONTROL function

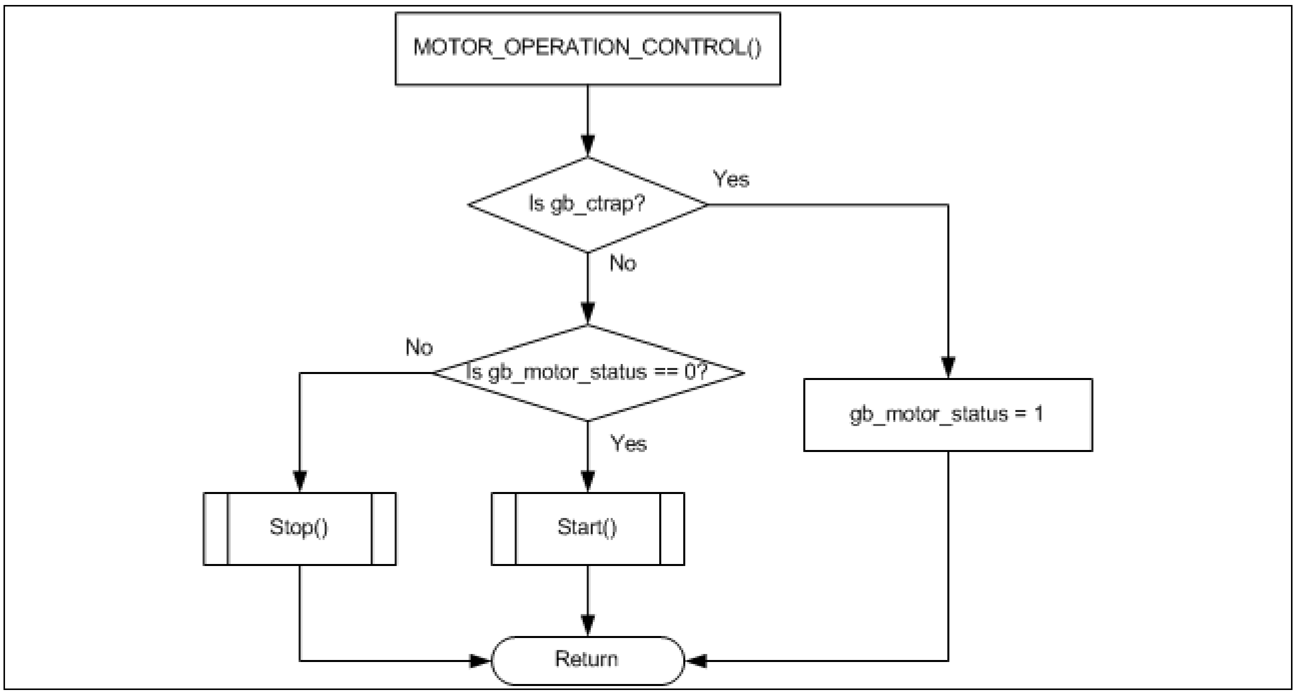


Figure 20 Flowchart of MOTOR_OPERATION_CONTROL function

Flowchart of Start function

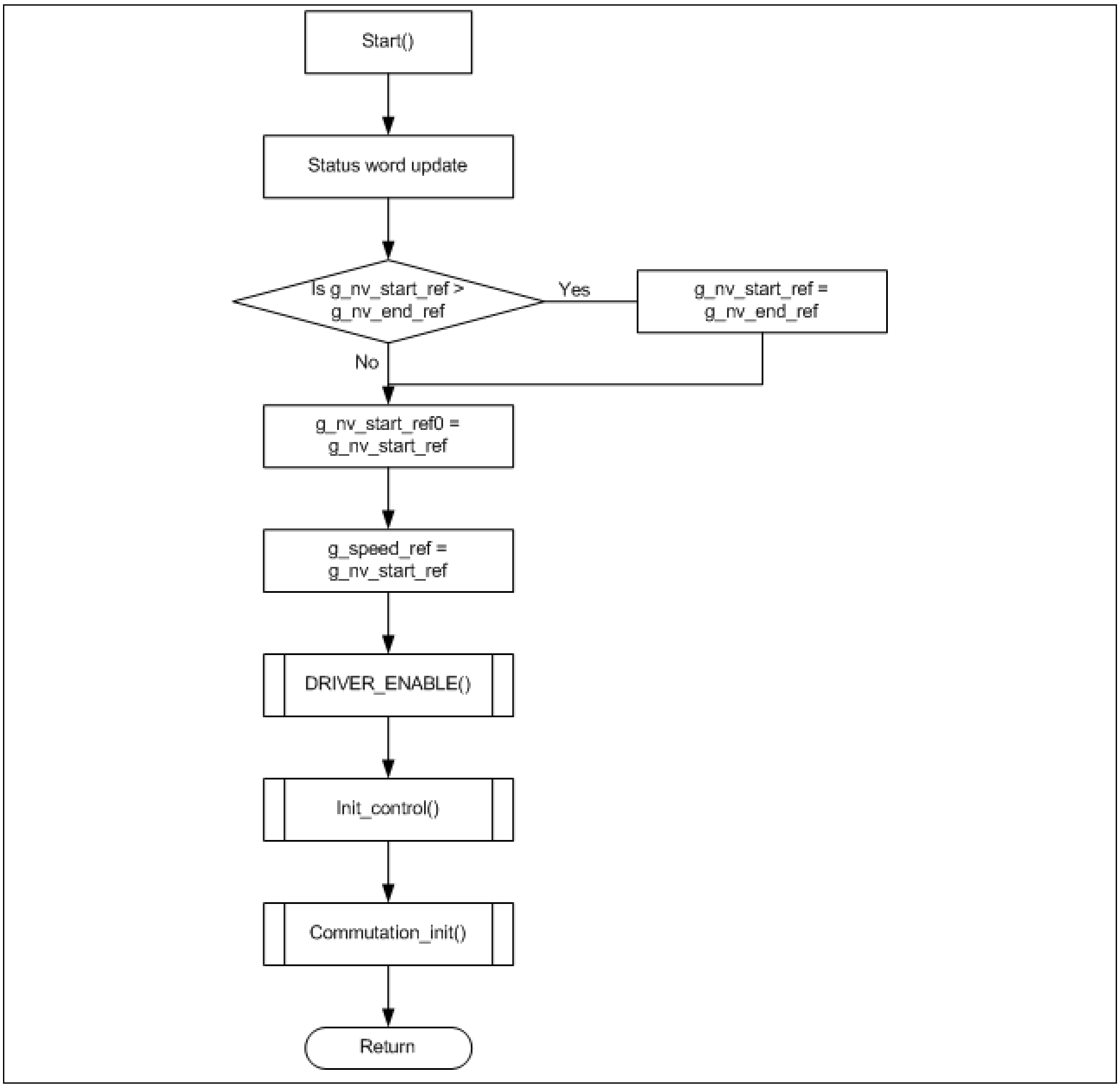

Start()
Status word update
Is g_nv_start_ref > g_nv_end_ref
Yes
g_nv_start_ref = g_nv_end_ref
No
g_nv_start_ref0 = g_nv_start_ref
g_speed_ref = g_nv_start_ref
DRIVER_ENABLE()
Init_control()
Commutation_init()
Return

Figure 21 Flowchart of Start function

Flowchart of Stop function

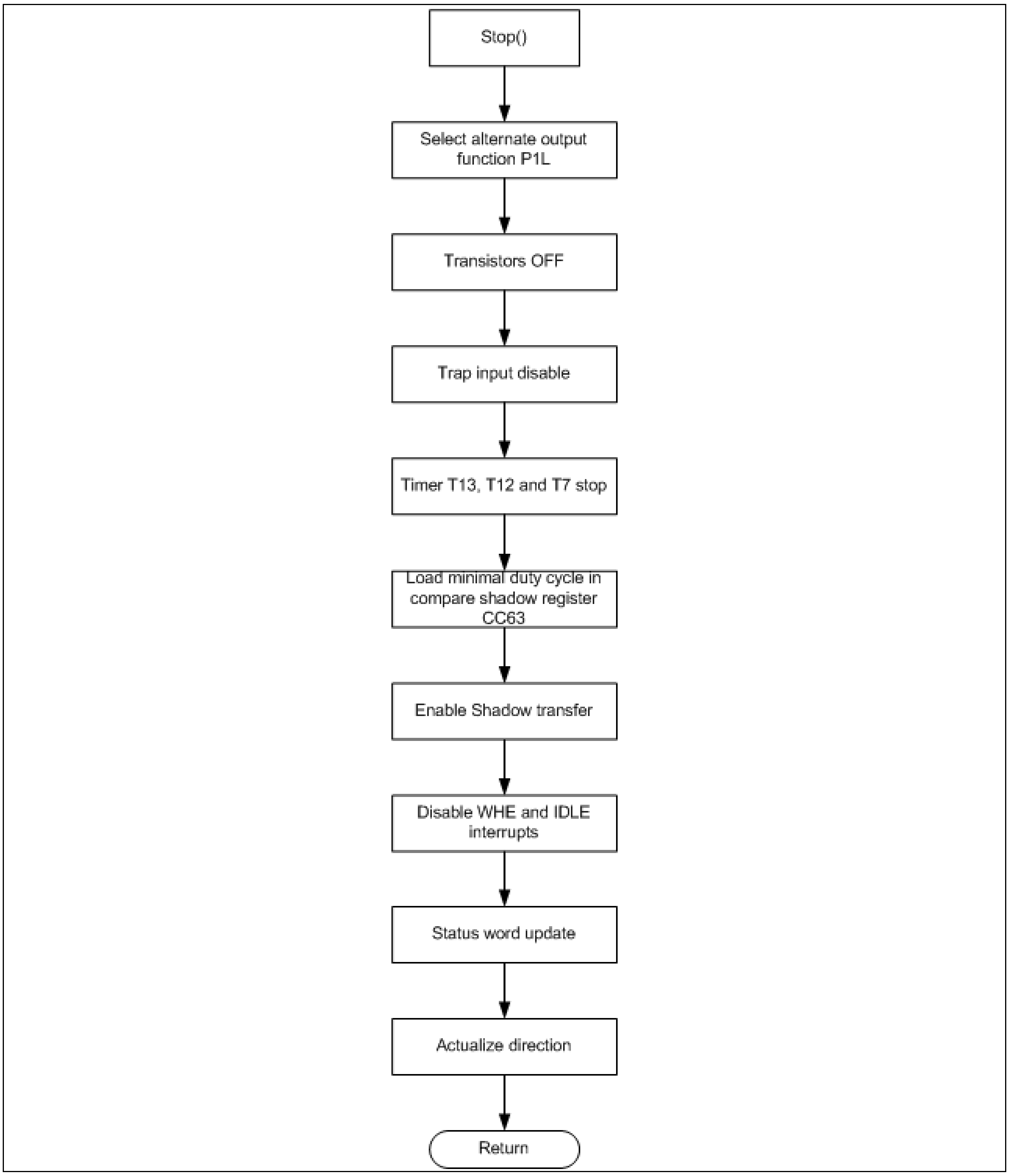

Stop()
Select alternate output function P1L
Transistors OFF
Trap input disable
Timer T13, T12 and T7 stop
Load minimal duty cycle in compare shadow register CC63
Enable Shadow transfer
Disable WHE and IDLE interrupts
Status word update
Actualize direction
Return

Figure 22 Flowchart of Stop function

Flowchart of init_control function

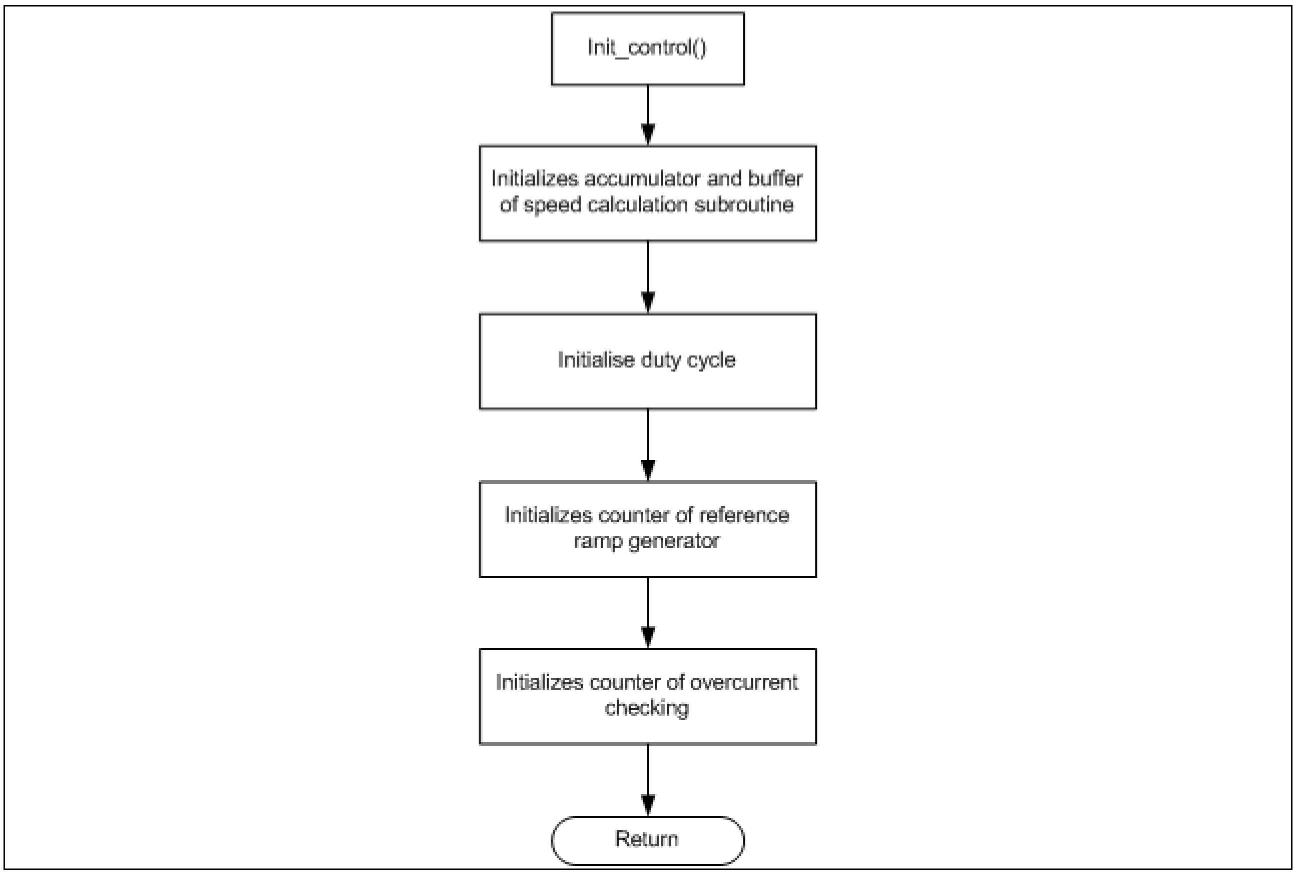

Init_control()
Initializes accumulator and buffer of speed calculation subroutine
Initialise duty cycle
Initializes counter of reference ramp generator
Initializes counter of overcurrent checking
Return

Figure 23 Flowchart of init_control function

Flowchart of Commutation_init function

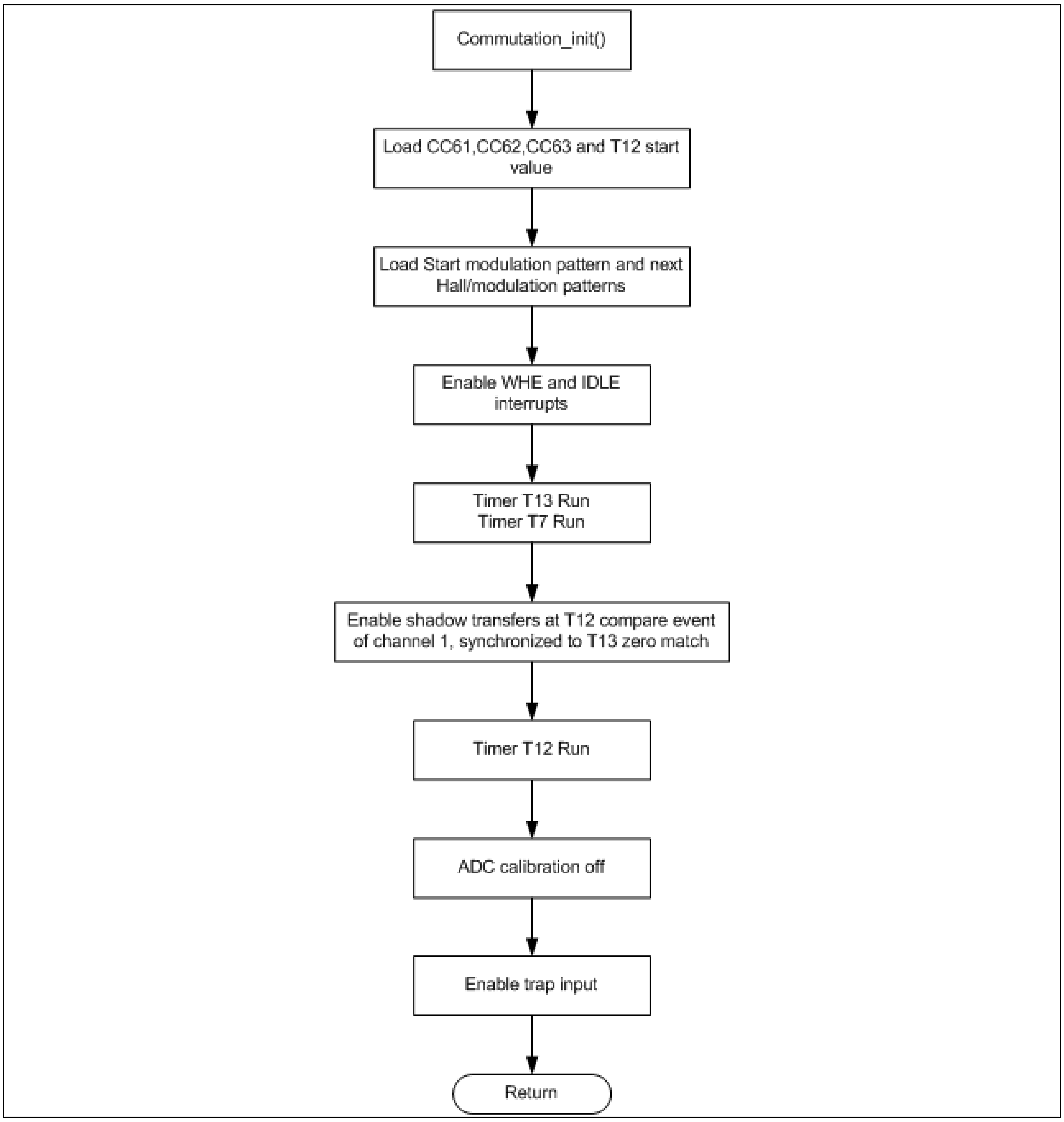

Commutation_init()
Load CC61,CC62,CC63 and T12 start value
Load Start modulation pattern and next Hall/modulation patterns
Enable WHE and IDLE interrupts
Timer T13 Run
Timer T7 Run
Enable shadow transfers at T12 compare event of channel 1, synchronized to T13 zero match
Timer T12 Run
ADC calibration off
Enable trap input
Return

Figure 24 Flowchart of Commutation_init function

Flowchart of speed_ramp_generation

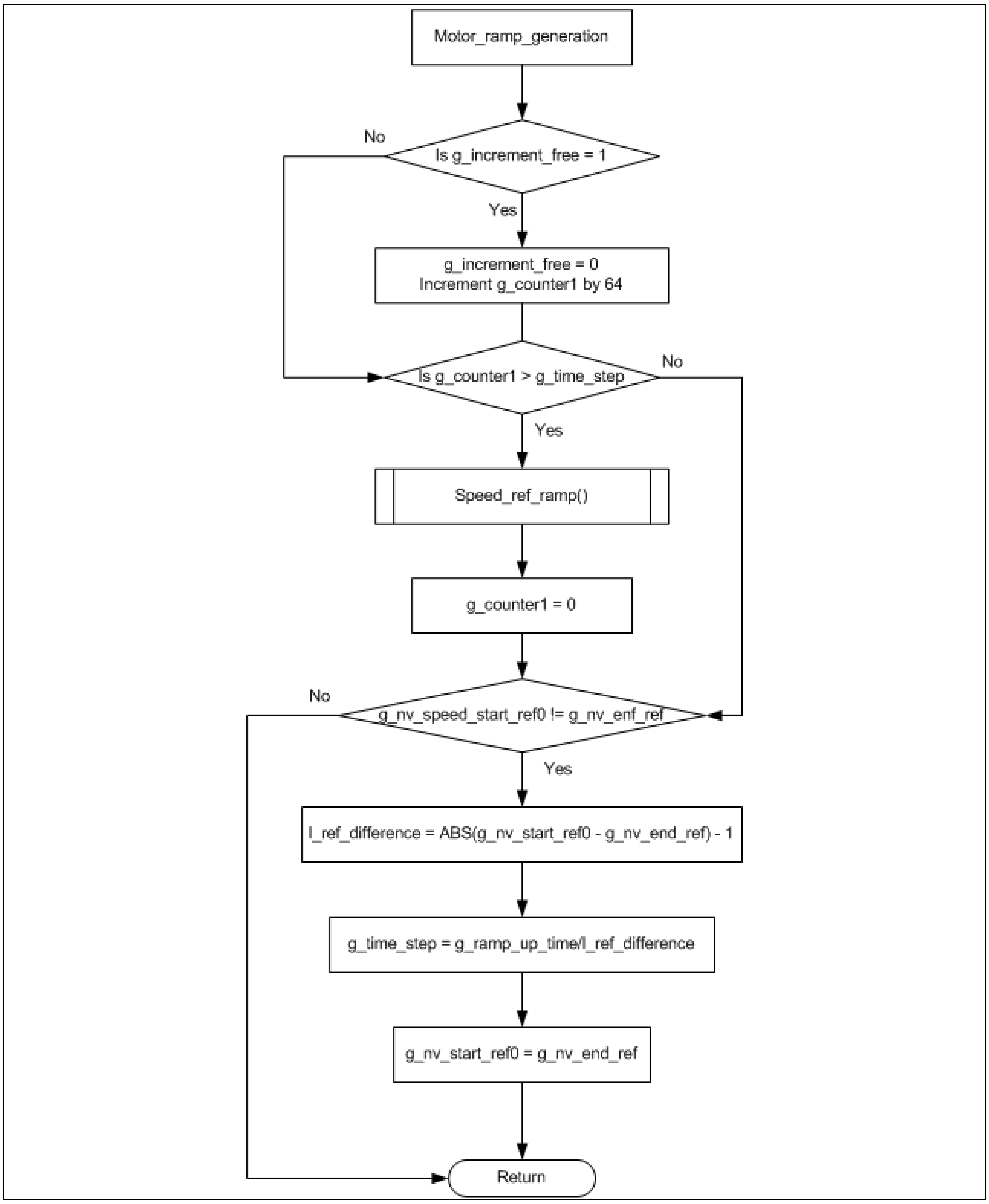

Motor_ramp_generation
Is g_increment_free = 1
No
Yes
g_increment_free = 0
Increment g_counter1 by 64
Is g_counter1 > g_time_step
No
Yes
Speed_ref_ramp()
g_counter1 = 0
g_nv_speed_start_ref0 != g_nv_enf_ref
No
Yes
l_ref_difference = ABS(g_nv_start_ref0 - g_nv_end_ref) - 1
g_time_step = g_ramp_up_time/l_ref_difference
g_nv_start_ref0 = g_nv_end_ref
Return

Figure 25 Flowchart of Speed_ramp_generation function

Flowchart of speed_ref_ramp function

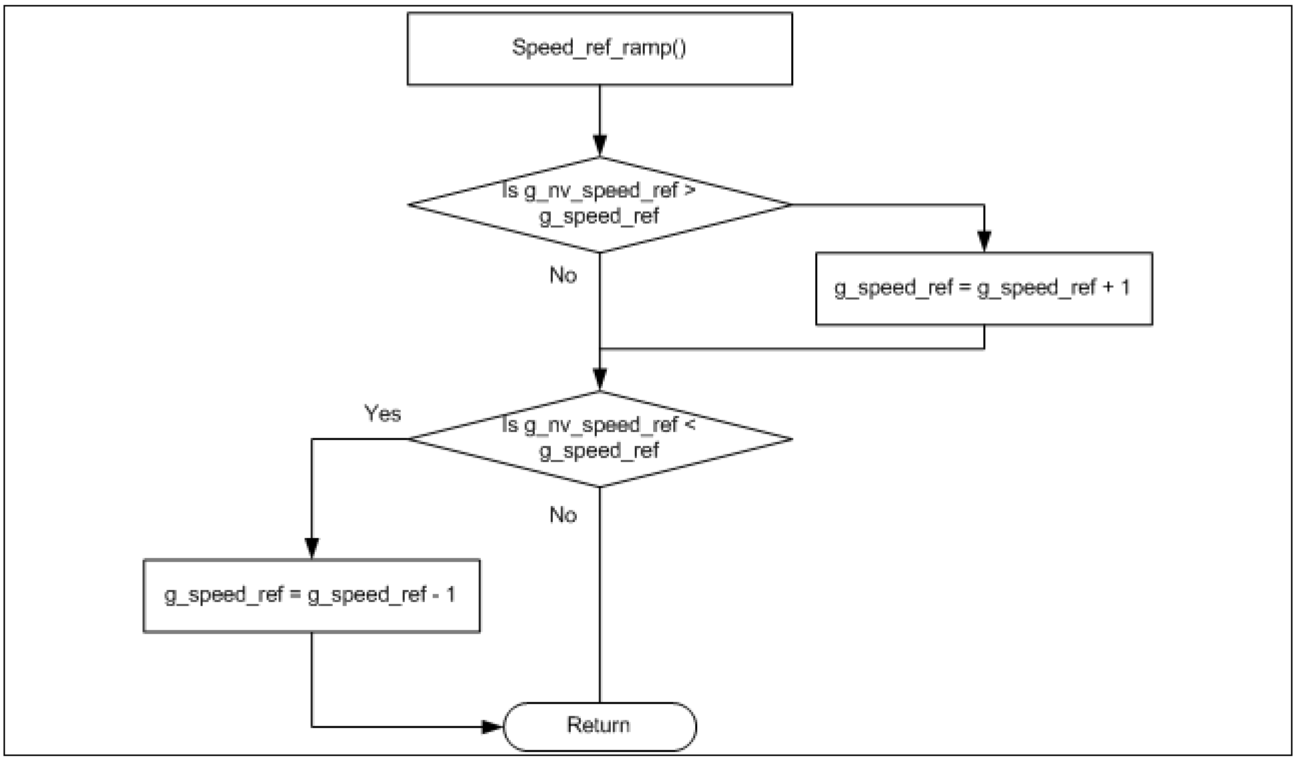

Speed_ref_ramp()
Is g_nv_speed_ref > g_speed_ref
No
g_speed_ref = g_speed_ref + 1
Yes
Is g_nv_speed_ref < g_speed_ref
No
g_speed_ref = g_speed_ref - 1
Return

Figure 26 Flowchart of speed_ref_ramp function

Flowchart of Interrupt CCU6_viNodeI1

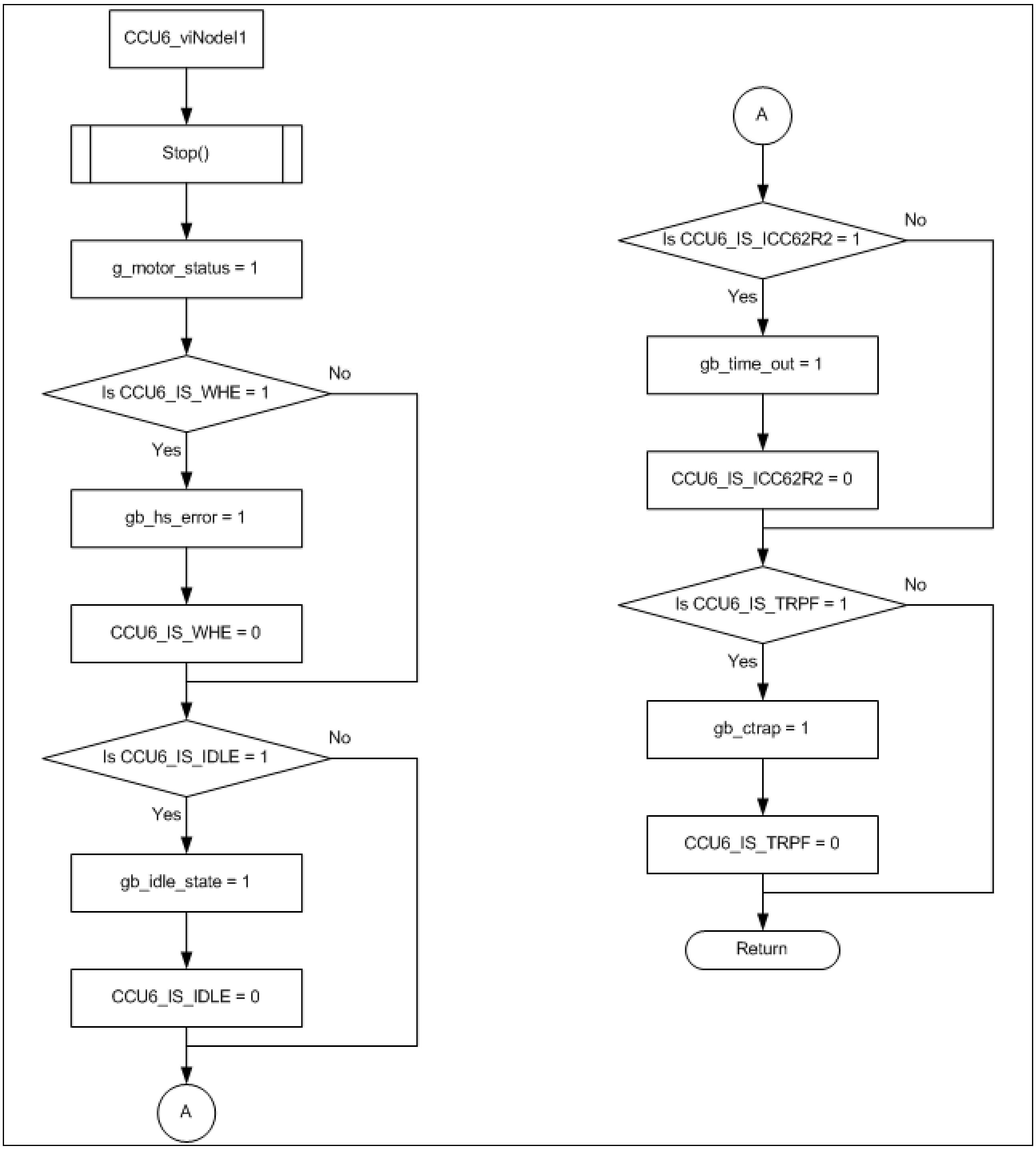


Figure 27 Flowchart of Interrupt CCU6_viNodeI1

Flowchart of Interrupt CCU6_vi_NodeI2

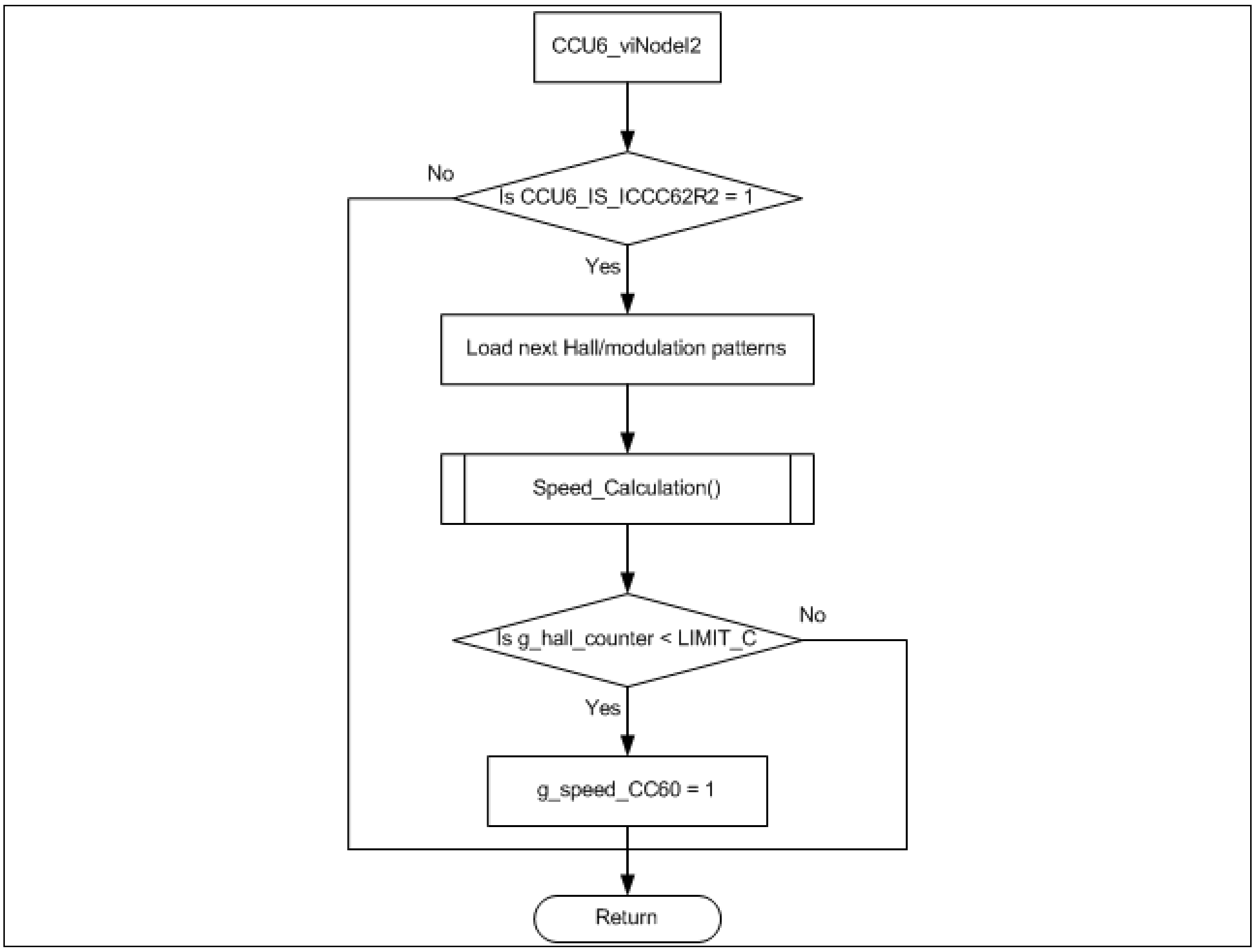


Figure 28 Flowchart of Interrupt CCU6_viNodeI2

Flowchart of Speed Calculation

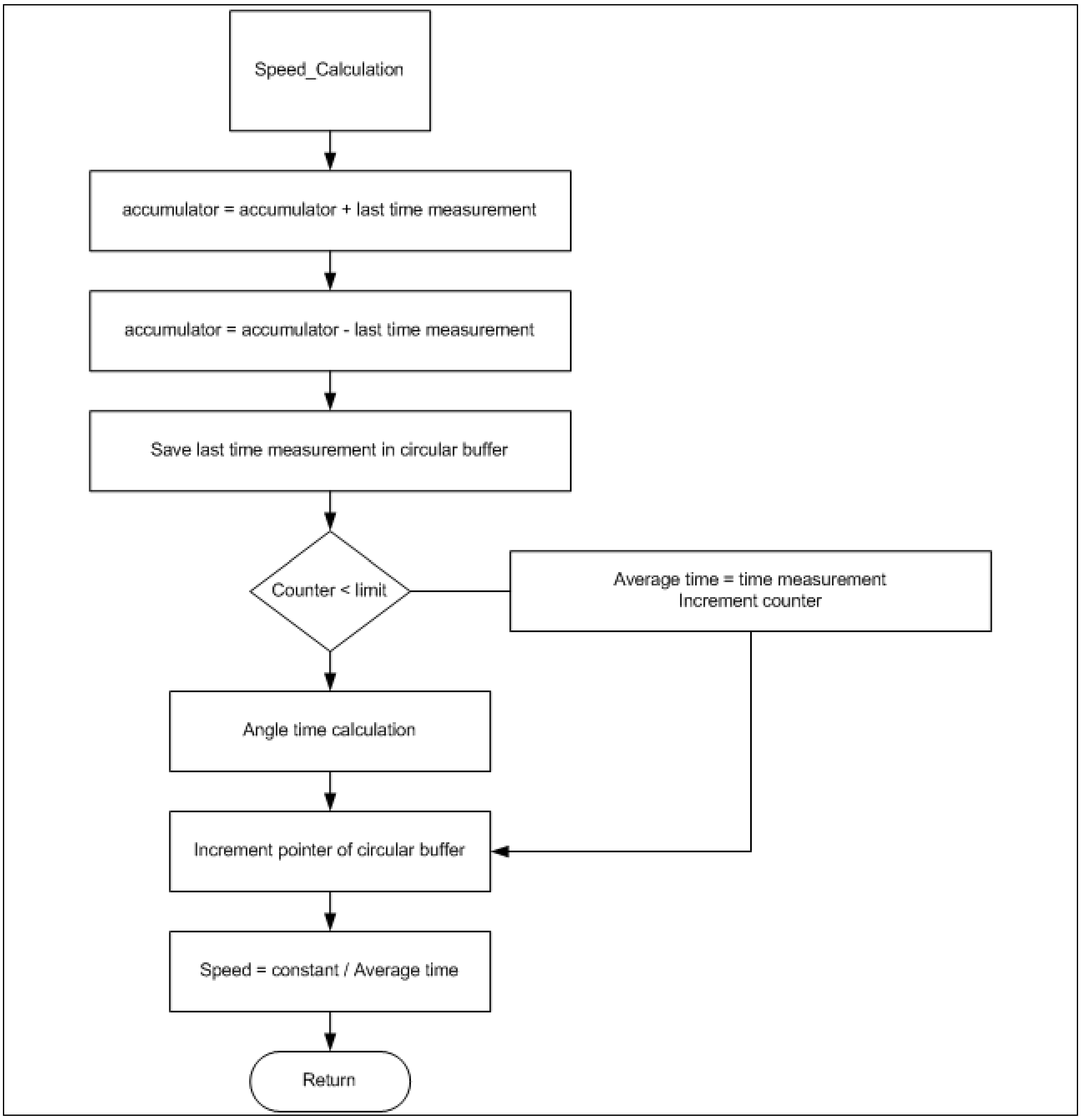

Speed_Calculation
accumulator = accumulator + last time measurement
accumulator = accumulator - last time measurement
Save last time measurement in circular buffer
Counter < limit
Average time = time measurement
Increment counter
Angle time calculation
Increment pointer of circular buffer
Speed = constant / Average time
Return

Figure 29 Flowchart of Speed calculation

Flowchart of Interrupt CCU6_viNodeI3

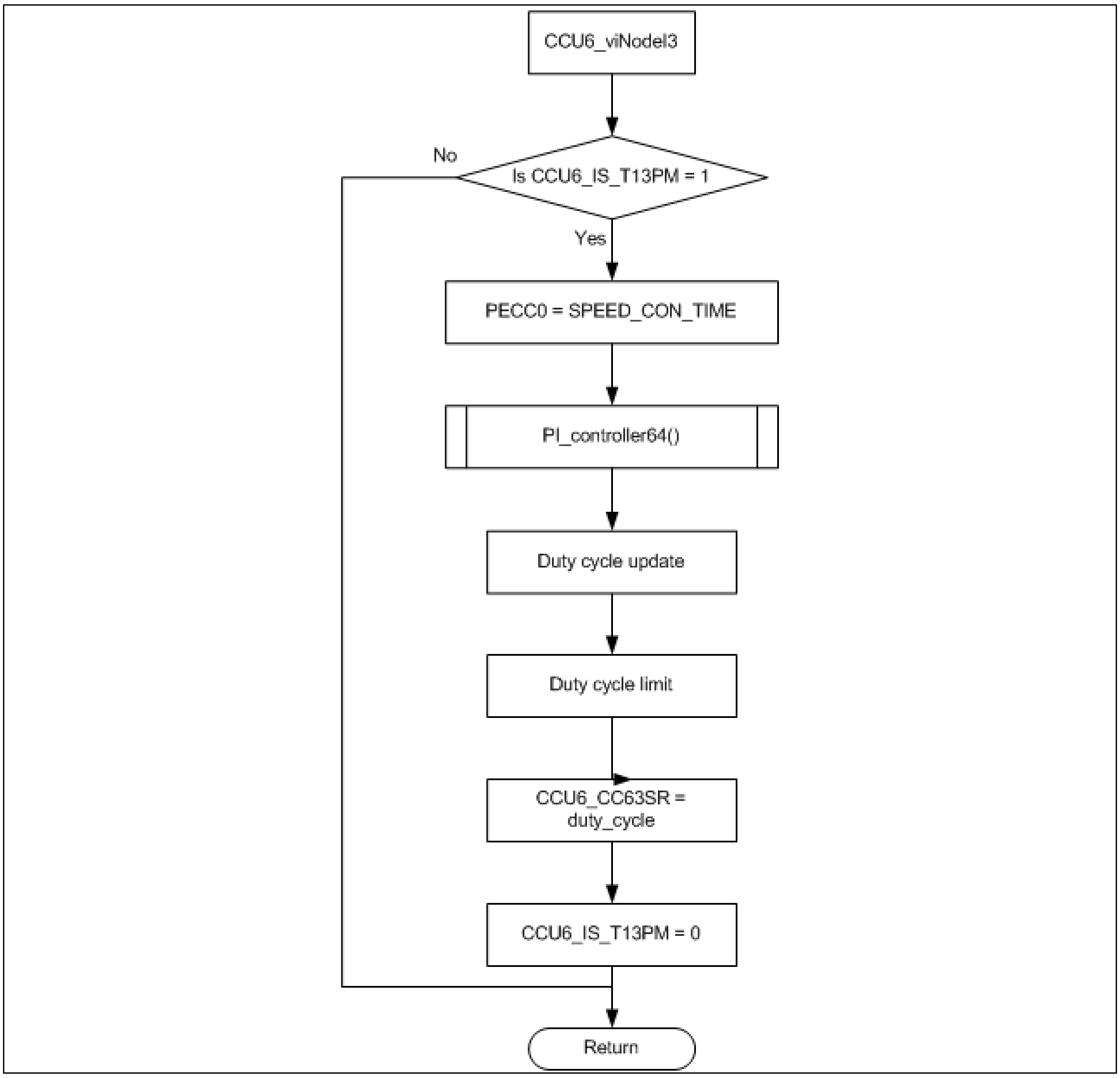


Figure 30 Flowchart of Interrupt CCU6_viNodeI3

Flowchart of PI_controller64

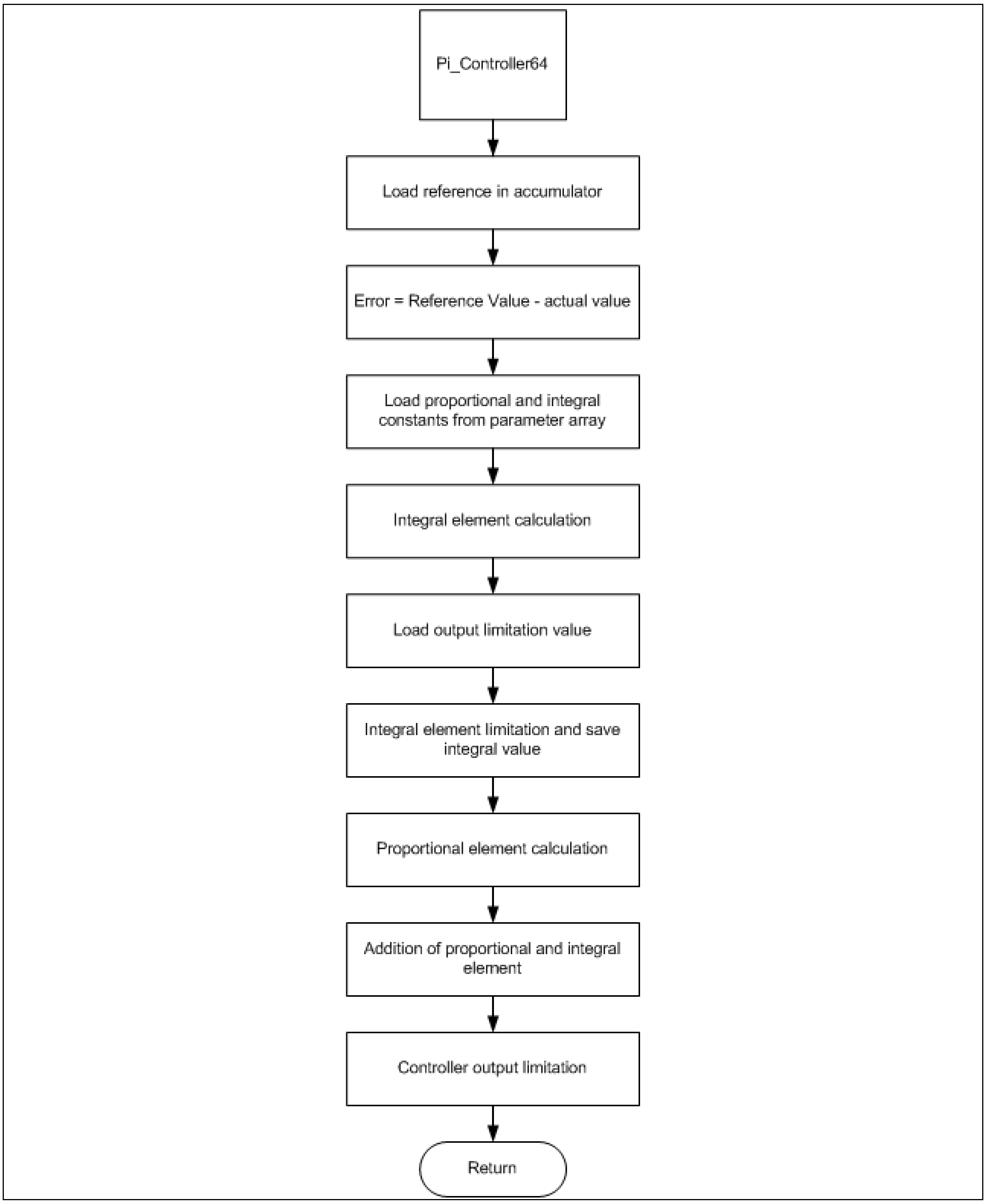

Pi_Controller64
Load reference in accumulator
Error = Reference Value - actual value
Load proportional and integral constants from parameter array
Integral element calculation
Load output limitation value
Integral element limitation and save integral value
Proportional element calculation
Addition of proportional and integral element
Controller output limitation
Return

Figure 31 Flowchart of PI_controller64

Appendix B Microcontroller Easy kit and Driver Board Connection

The picture of the XC164CM microcontroller easy kit is shown in figure 32. The BUX1 and BUX2 connectors of the kit are marked in Red.

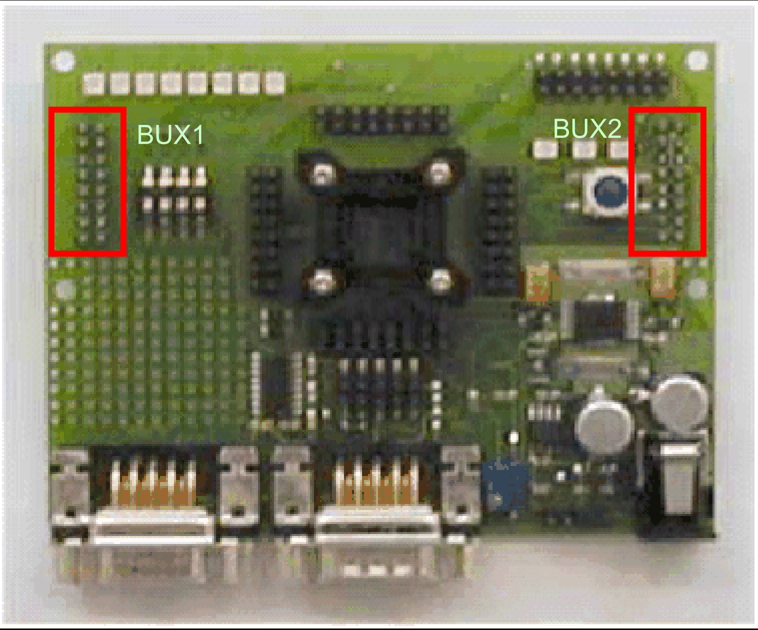


Figure 32 XC164CM Easy kit

The corresponding pin descriptions of BUX1 and BUX2 connectors are shown in figure 33.

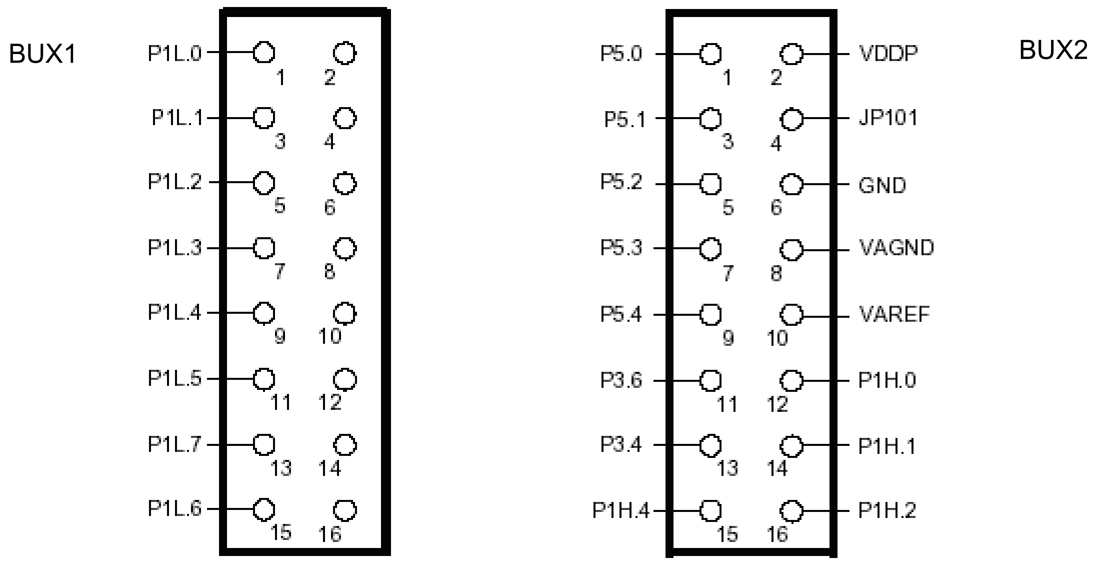


Figure 33 Pin description of Connectors BUX1 and BUX2

The picture of the BTS7960 driver board is shown in Figure 34. The X1 and X2 connectors of the driver board are marked in red.

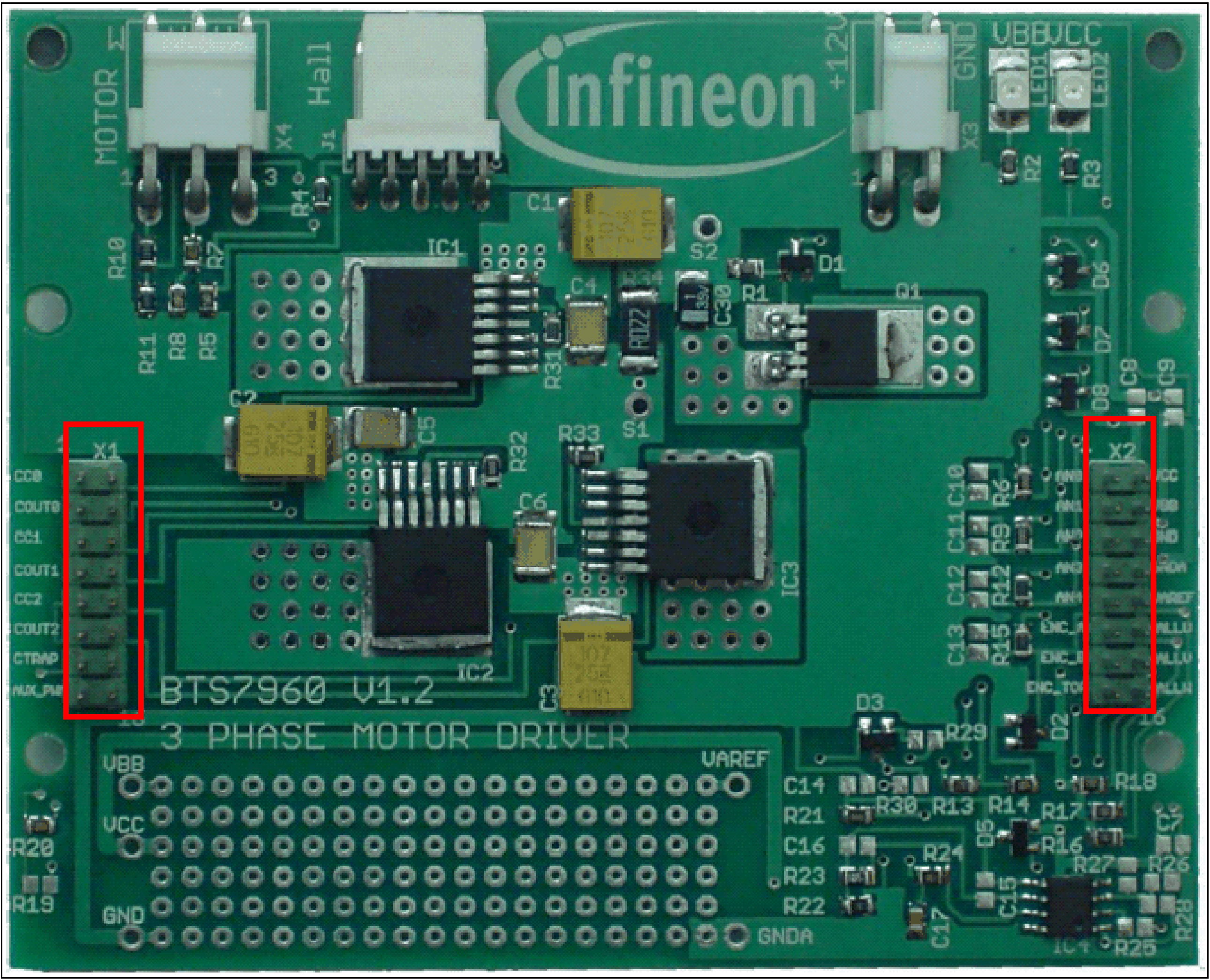


Figure 34 BTS7960 3-Phase Motor Driver

The detailed pin description of connectors X1 and X2 in BTS driver board is shown in Figure 35.

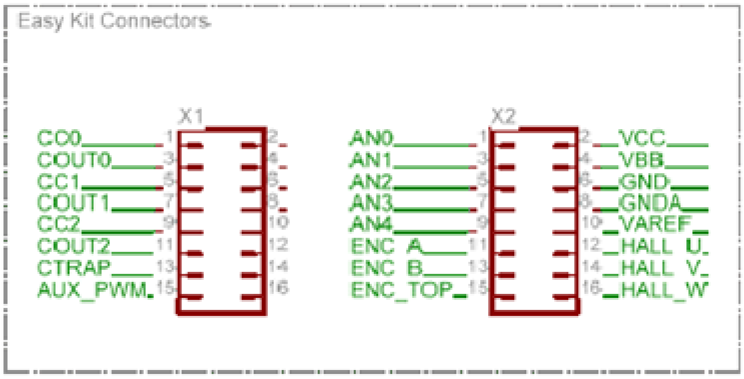


Figure 35 Connector description of BTS7960 driver IC

The connectors BUX1 and BUX2 of the XC164CM Easy kit microcontroller should be connected to the connectors X1 and X2 of the 3-phase BTS7960 driver board respectively.

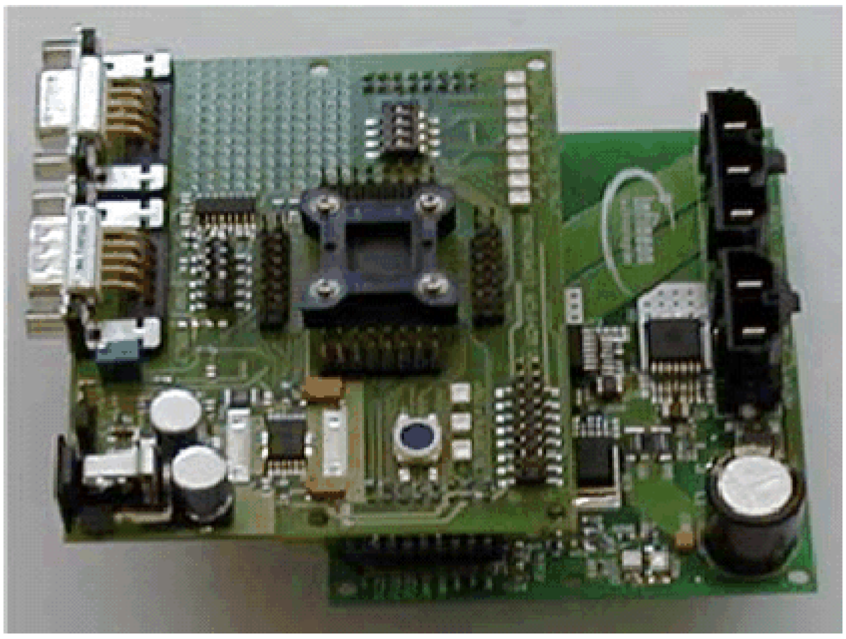


Figure 36 Microcontroller Easy Kit connected with the Driver Power board

The connection of microcontroller easy kit with the power driver board is shown in figure 36.

Appendix C Commutation Pattern with BTS7960 Driver IC

The commutation pattern is not necessarily to have a one to one mapping between the controller output and the signal fed at the Gate terminal of the transistor. It depends on so many factors like driver IC truth table, mapping of signals to driver IC input and at the Gate input to the transistor and the type of transistor (n-channel or p-channel) or we say passive state.

For the switching on and off the phase windings the modulation pattern from the CC6x/COUT6x (x = 0,1,2) of CAPCOM unit is fed to the driver IC as shown in figure 37.

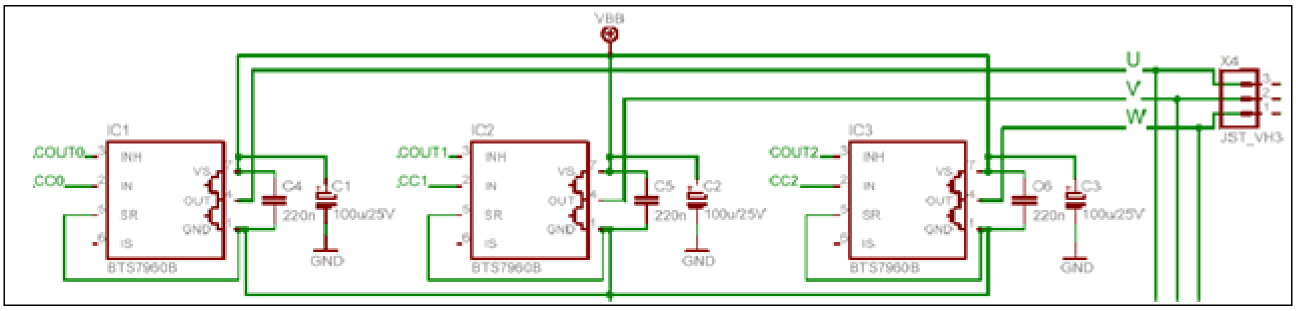
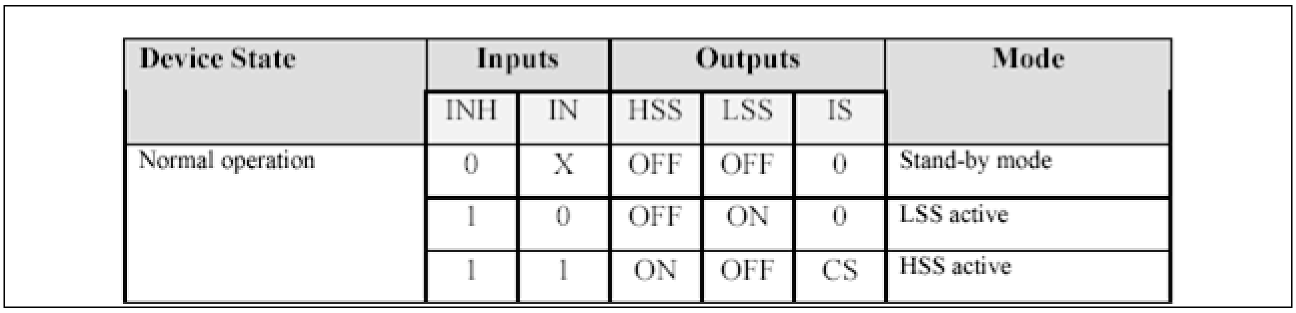


Figure 37 CAPCOM output and BTS7960 driver IC input connections



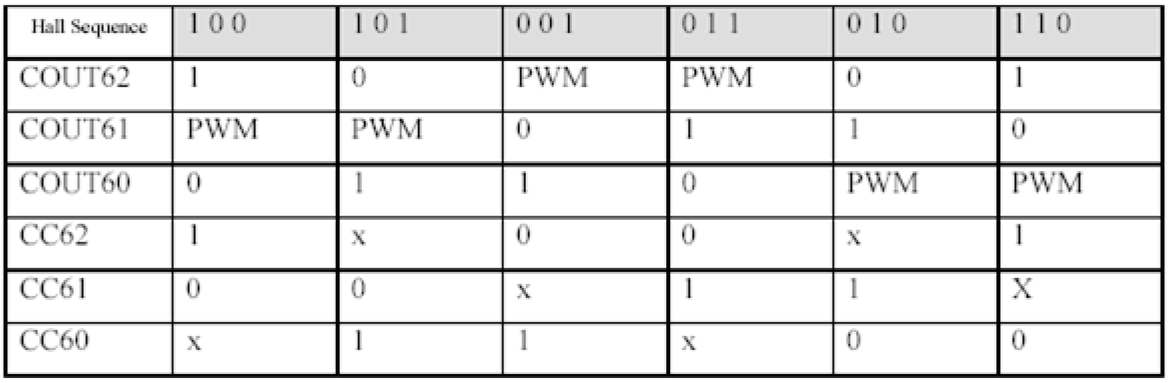
Device State	Inputs		Outputs			Mode
	INH	IN	HSS	LSS	IS	
Normal operation	0	X	OFF	OFF	0	Stand-by mode
	1	0	OFF	ON	0	LSS active
	1	1	ON	OFF	CS	HSS active

Figure 38 BTS7960 Driver IC truth table

The commutation table from Figure 4 should be mapped using the truth table of the BTS driver IC to arrive at the modulation pattern generated by the CAPCOM unit of the microcontroller.

Figure 38 shows the truth table of the BTS7960 driver IC. The COUT6x (x = 0,1,2) signals are fed to the INH pin of the driver IC and CC6x (x = 0,1,2) signals are fed to the IN pin of the driver IC. In the truth table ‘X’ represents either ‘0’ or ‘1’.

Figure 39 shows the mapped modulation pattern for the microcontroller.



Hall Sequence	1 0 0	1 0 1	0 0 1	0 1 1	0 1 0	1 1 0
COUT62	1	0	PWM	PWM	0	1
COUT61	PWM	PWM	0	1	1	0
COUT60	0	1	1	0	PWM	PWM
CC62	1	x	0	0	x	1
CC61	0	0	x	1	1	X
CC60	x	1	1	x	0	0

Figure 39 Modulation pattern table at the microcontroller side

The modulation pattern varies for different types of machine types. To provide flexibility for the user in configuring the modulation pattern, CAPCOM unit has a MCMOUTS register which stores the Current Hall Pattern (CHP), expected Hall pattern (EHP) and modulation pattern (MCMCP). At every correct Hall Event, a new Hall pattern with its corresponding modulation pattern will be loaded in this register. The register description is given in figure 40.

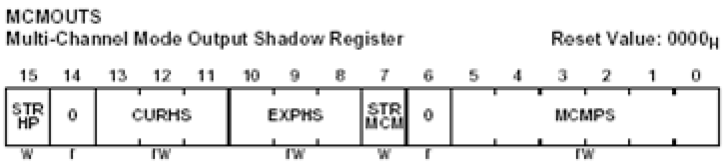
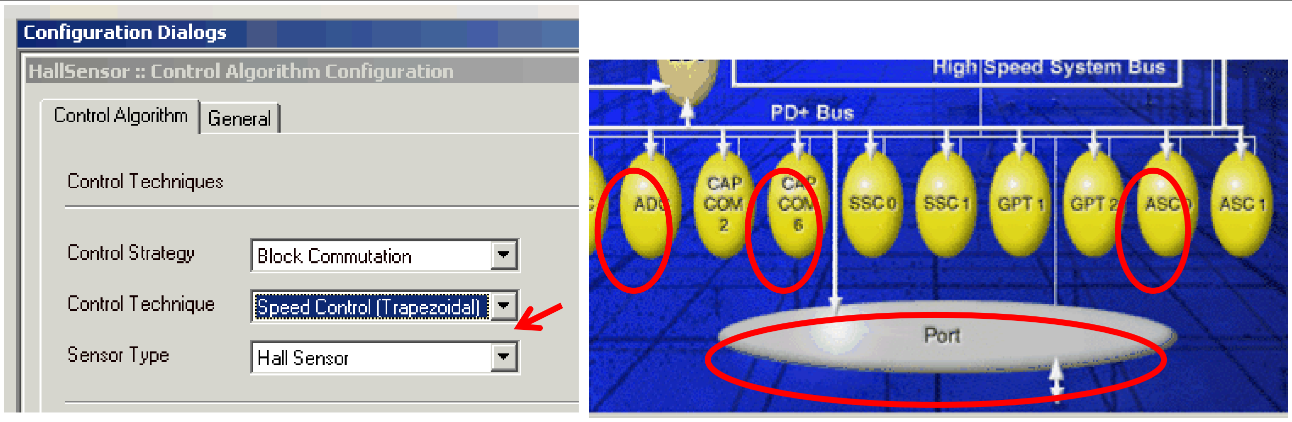


Figure 40 Multi-Channel Mode Output Shadow Register

Appendix D A Quick Start to Work with DAvE Drive

Control Technique Selection

Based on the Control technique selected in the Control Algorithm Dialogue, the necessary peripherals settings will be enforced on DAvE for that control technique. The list of peripherals used by the DAvE Drive for the speed control algorithm implementation are CC6, ADC, IO ports and ASC0 for download and runtime communication.

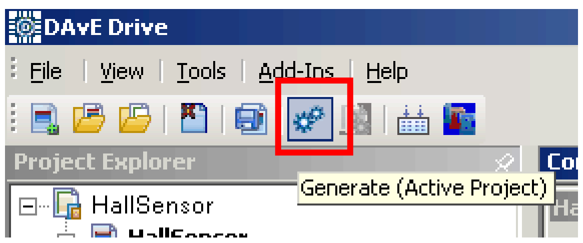


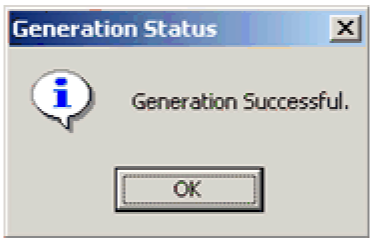
Control Strategy: Block Commutation
Control Technique: Speed Control (Trapezoidal)
Sensor Type: Hall Sensor

Figure 41 Selection of Control Technique

Code Generation

With the default configuration, the user can generate code, compile and download it to the target board using the Run time Control Panel.



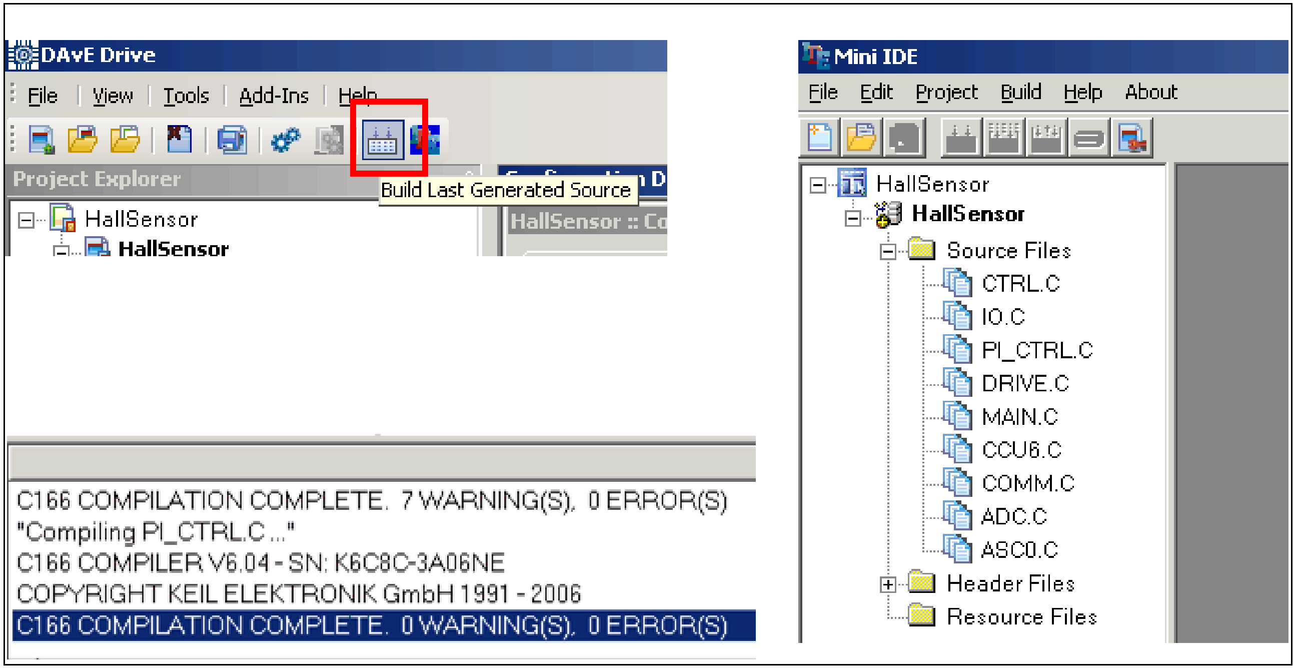


Generation Successful.

Figure 42 Code Generation

Compilation and Hex file generation

After Source code Generation, if the user wants to compile and generate hex file without modifying the software or viewing it then “Build Last Generated Source” option button should be selected. MiniIDE will open up and the compilation will start. The compilation status will be displayed in the log window.



```
C166 COMPILATION COMPLETE.  7 WARNING(S),  0 ERROR(S)
"Compiling PI_CTRL.C ..."
C166 COMPILER V6.04 - SN: K6C8C-3A06NE
COPYRIGHT KEIL ELEKTRONIK GmbH 1991 - 2006
C166 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

Figure 43 Compilation and Hex file generation

Modify Source, compilation and Hex-File Generation

By selecting the Display IDE, the user will be able to view and modify the source code generated. After modifying the code the user can compile the code and after successful compilation, hex file will be generated.

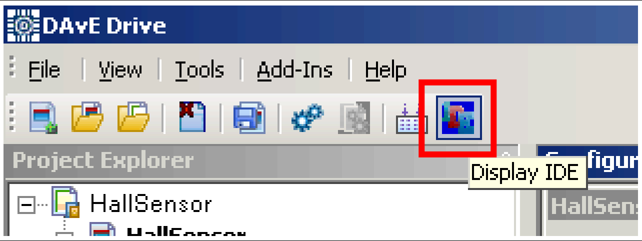
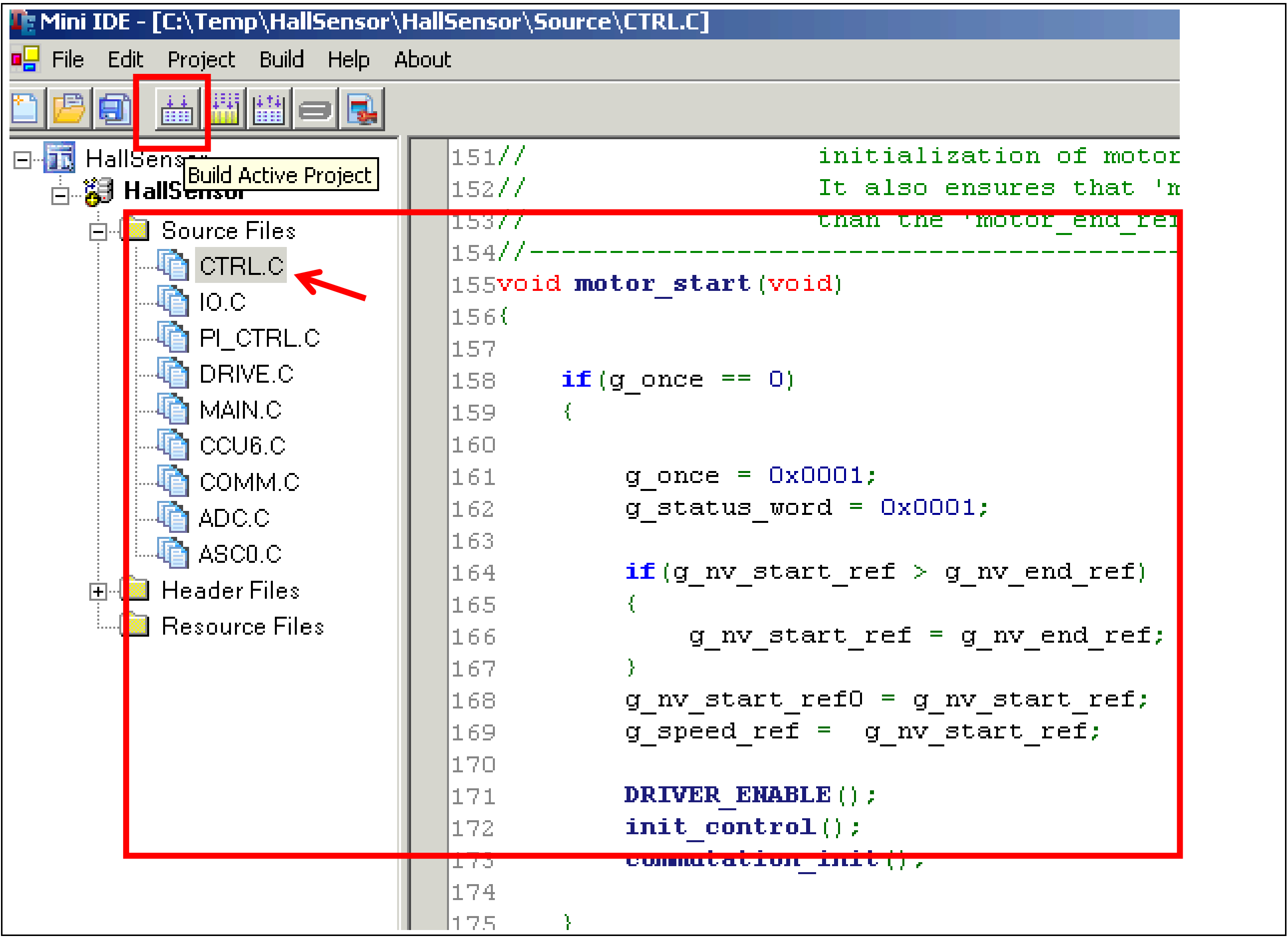


Figure 44 Selection of MiniIDE to modify code



```
//                        initialization of motor
//                        It also ensures that 'm
//                        than the 'motor_end_ref
//------------------------------------------------
void motor_start(void)
{

    if(g_once == 0)
    {

        g_once = 0x0001;
        g_status_word = 0x0001;

        if(g_nv_start_ref > g_nv_end_ref)
        {
            g_nv_start_ref = g_nv_end_ref;
        }
        g_nv_start_ref0 = g_nv_start_ref;
        g_speed_ref =  g_nv_start_ref;

        DRIVER_ENABLE();
        init_control();
        commutation_init();

    }
```

Figure 45 Source code editing and build files for Hex file generation

Hex file download

After hex file generation, the user can download the hex file to the target and communicate with the target for the motor control operation. The user can use the Runtime Control panel for downloading, run time control and monitoring of application parameters.

In microcontroller board the config DIP switch should be as shown in figure 46 for downloading.

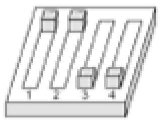
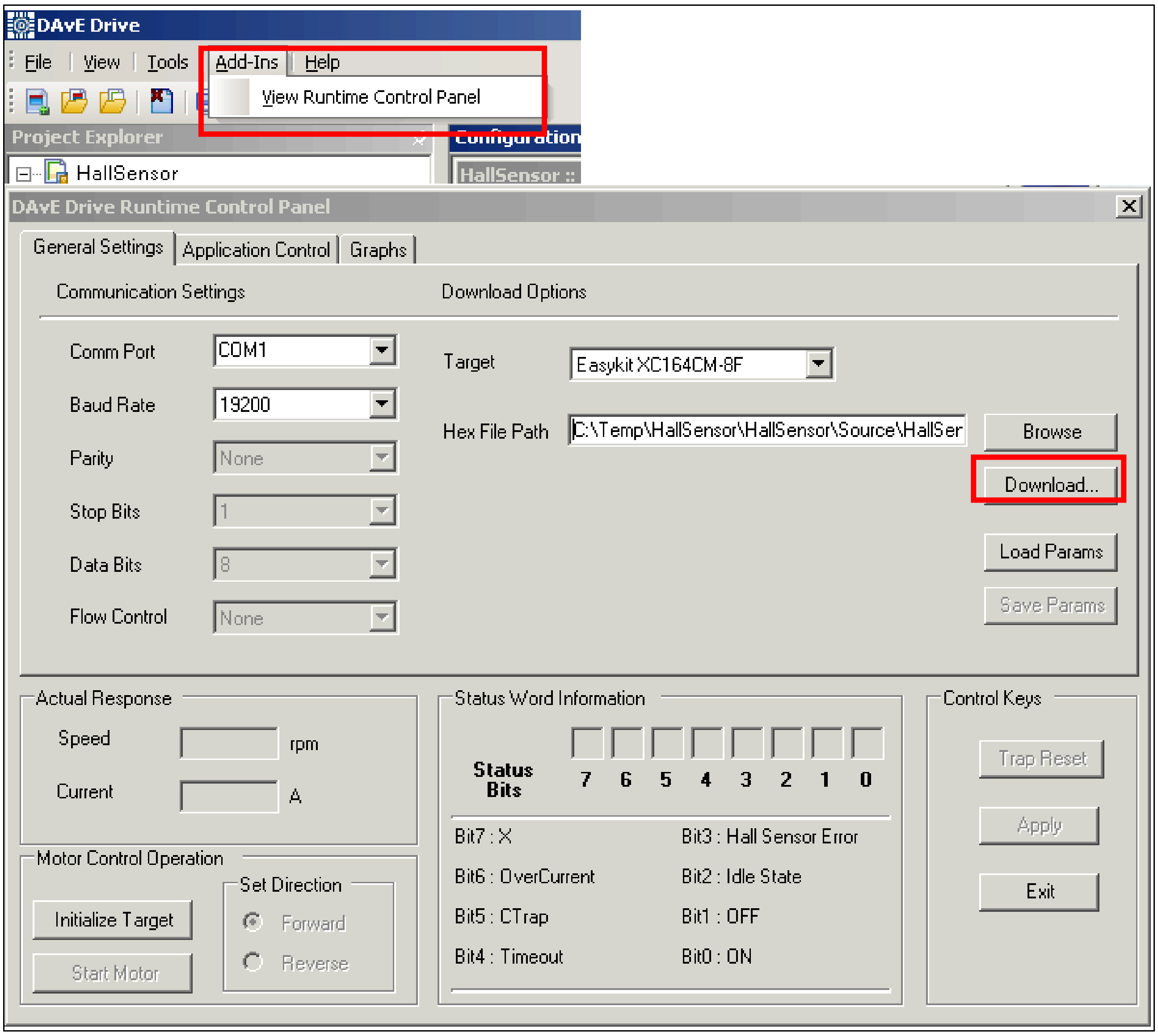


Figure 46 ASC Bootstrap loader mode for microcontroller Easy kit (Config DIP Switch)



Communication Settings

Comm Port	COM1
Baud Rate	19200
Parity	None
Stop Bits	1
Data Bits	8
Flow Control	None

Download Options

Target	Easykit XC164CM-8F
Hex File Path	C:\Temp\HallSensor\HallSensor\Source\HallSer

Actual Response

Speed rpm

Current A

Motor Control Operation

Set Direction

Status Word Information

Status Bits 7 6 5 4 3 2 1 0

Bit7 : X	Bit3 : Hall Sensor Error
Bit6 : OverCurrent	Bit2 : Idle State
Bit5 : CTrap	Bit1 : OFF
Bit4 : Timeout	Bit0 : ON

Control Keys

Figure 47 Hex file download with RS232 serial port

Initialize, Monitor and Control Target

On successful download of hex file to target, the user has to set the Config switch as shown in figure 48 to start the code.

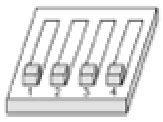
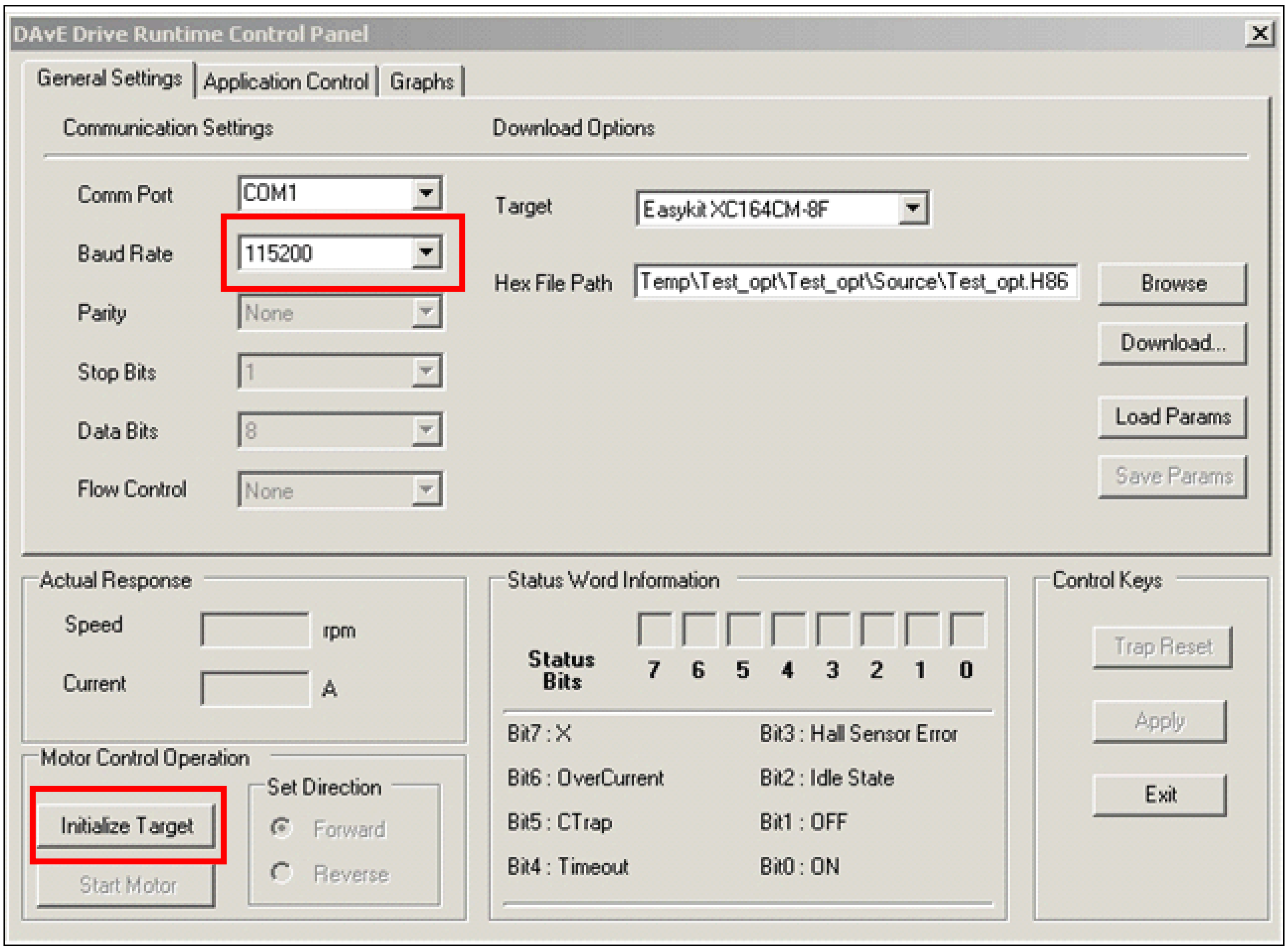


Figure 48 Standard start mode for Microcontroller Easy Kit

The user has to set the ASC baud rate as 115200 and press 'initialize' target.


DAvE Drive Runtime Control Panel

General Settings | Application Control | Graphs

Communication Settings

Comm Port: COM1
Baud Rate: 115200
Parity: None
Stop Bits: 1
Data Bits: 8
Flow Control: None

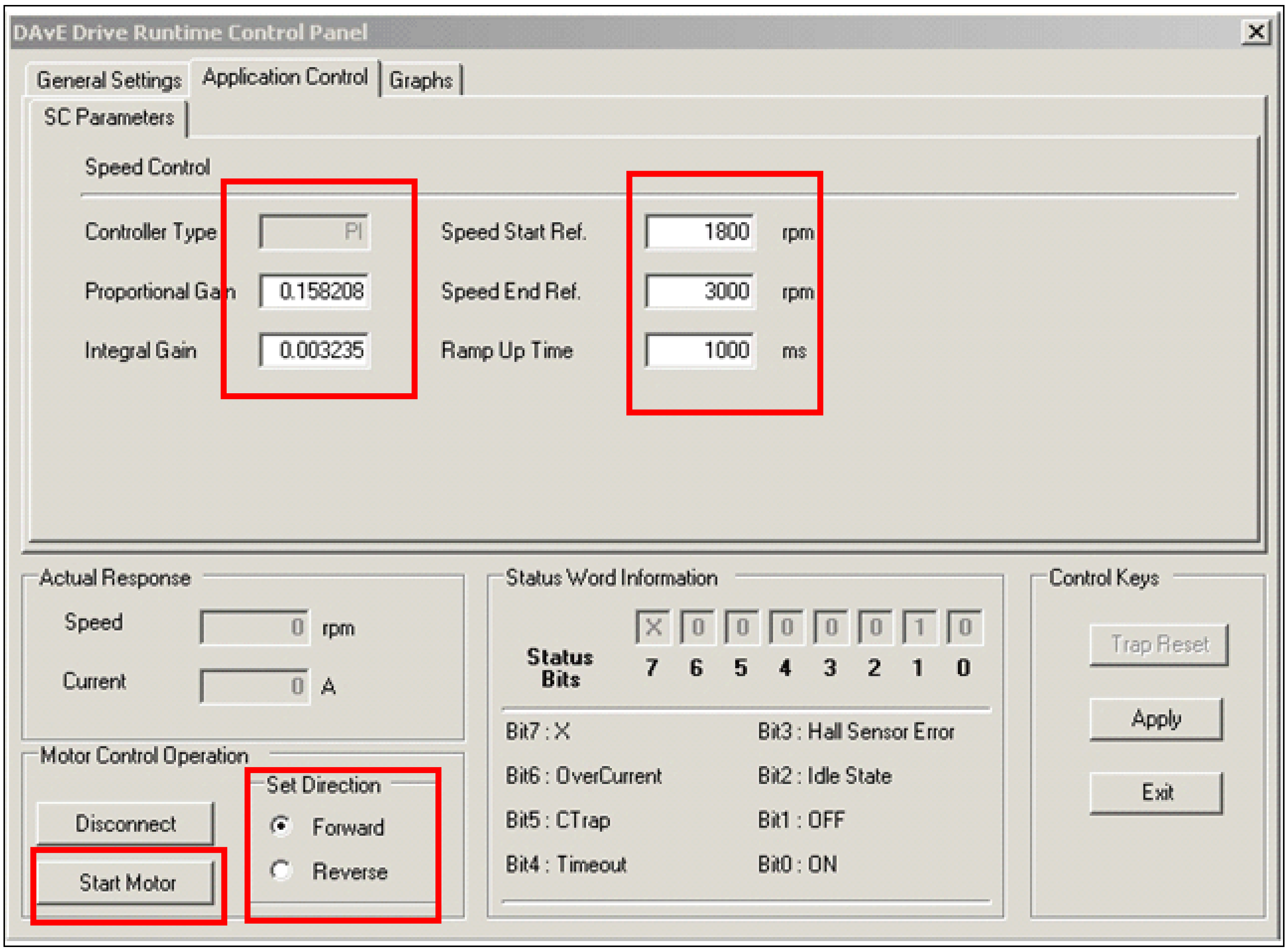
Download Options

Target: Easykit XC164CM-8F
Hex File Path: Temp\Test_opt\Test_opt\Source\Test_opt.H86

Bit7 : X
Bit6 : OverCurrent
Bit5 : CTrap
Bit4 : Timeout
Bit3 : Hall Sensor Error
Bit2 : Idle State
Bit1 : OFF
Bit0 : ON

Figure 49 Initialize target

After successful initialization, the user can start the motor with or without modifying the application parameters.


DAvE Drive Runtime Control Panel

General Settings | Application Control | Graphs

SC Parameters

Speed Control

Controller Type	PI	Speed Start Ref.	1800	rpm
Proportional Gain	0.158208	Speed End Ref.	3000	rpm
Integral Gain	0.003235	Ramp Up Time	1000	ms

Actual Response

Speed 0 rpm

Current 0 A

Status Word Information

Status Bits	7	6	5	4	3	2	1	0
	X	0	0	0	0	0	1	0

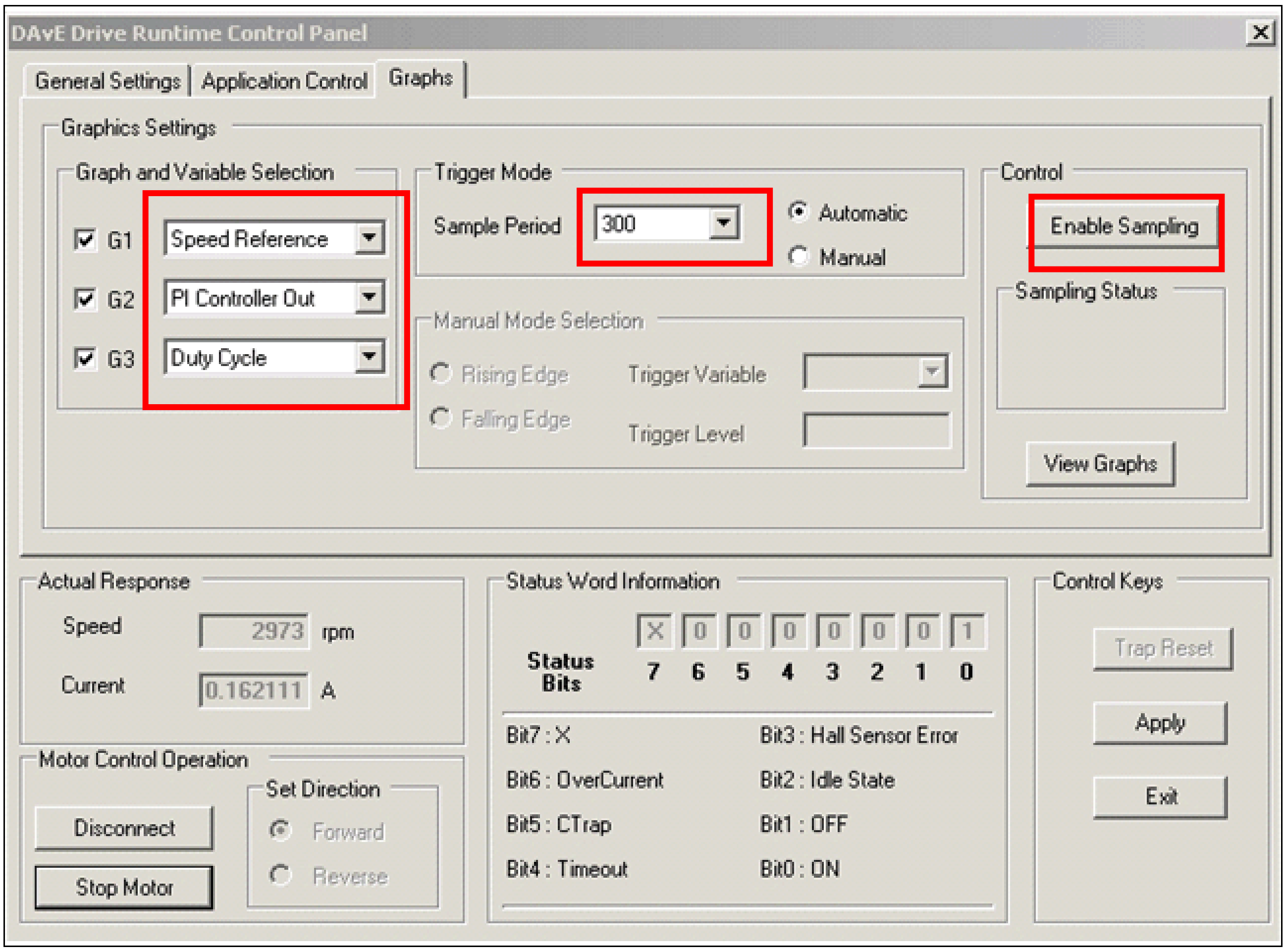
Bit7 : X	Bit3 : Hall Sensor Error
Bit6 : OverCurrent	Bit2 : Idle State
Bit5 : CTrap	Bit1 : OFF
Bit4 : Timeout	Bit0 : ON

Motor Control Operation

Set Direction: [✓ Forward] Reverse

Figure 50 Start/Stop, Forward/Reverse, Speed ref, P& I Gain settings

The User can select the variable and sample period for viewing in graphs. The user should select the enable sampling option to collect the new data for viewing in graph.



[✓] G1 Speed Reference
[✓] G2 PI Controller Out
[✓] G3 Duty Cycle

Sample Period 300

[✓ Automatic] Manual

Speed 2973 rpm
Current 0.162111 A

Status Bits	7	6	5	4	3	2	1	0
	X	0	0	0	0	0	0	1

Bit7 : X
Bit6 : OverCurrent
Bit5 : CTrap
Bit4 : Timeout
Bit3 : Hall Sensor Error
Bit2 : Idle State
Bit1 : OFF
Bit0 : ON

[✓ Forward] Reverse

Figure 51 Graph variable selection and enable sampling

After sampling the graph will be displayed automatically.

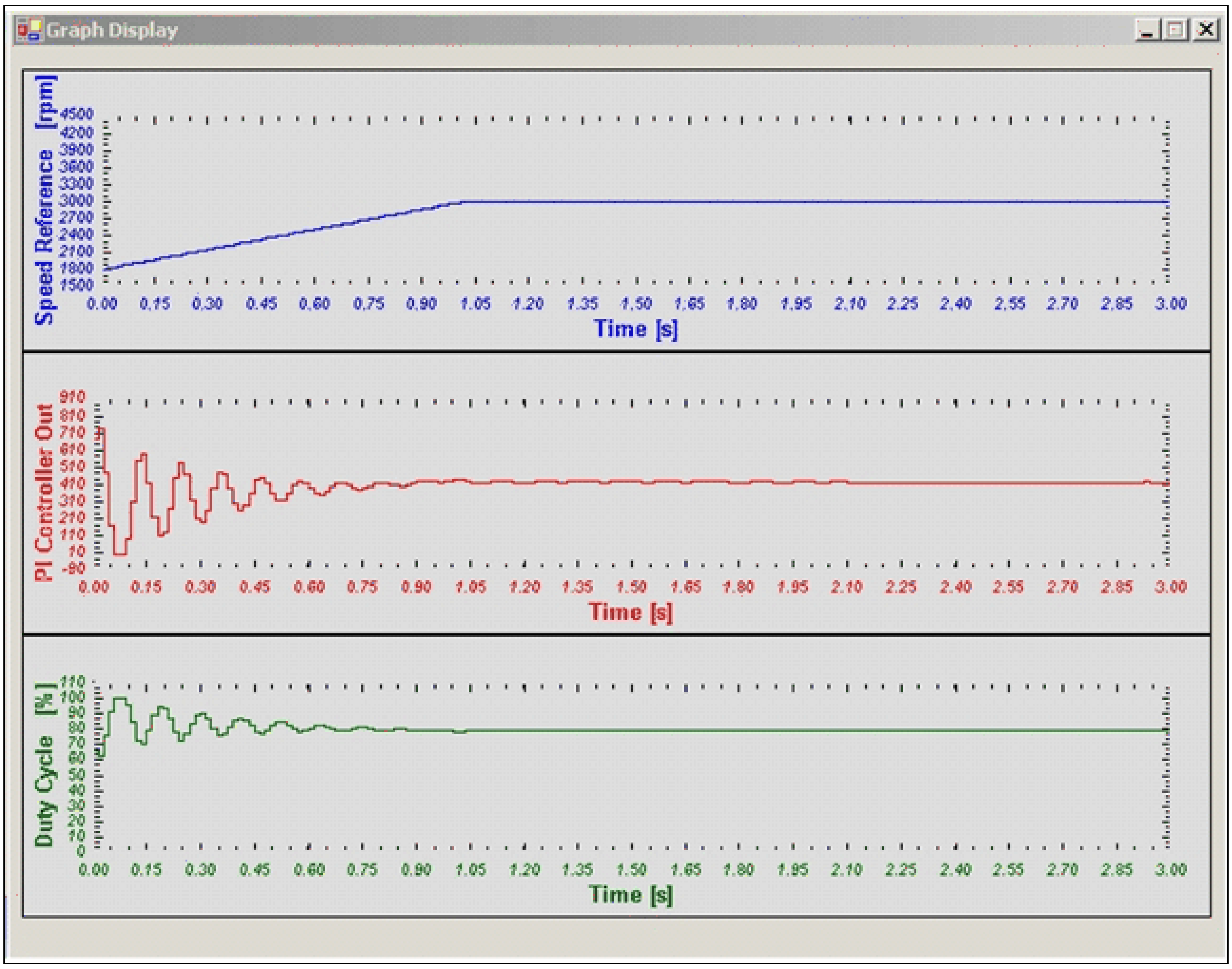


Figure 52 Graphical display of variables