# DEVELOPMENT AND VALIDATION OF MODEL-BASED SOFTWARE FOR TA-ISG (TORQUE ASSIST-INTEGRATED STARTER GENERATOR) MOTOR CONTROLLER FOR HYBRID CAR APPLICATION

**BITS ZG629T: Dissertation**

by

Mr. Paul Lalthuamsanga

ID No: 2010HZ12772

**Dissertation work carried out at**

**Tata Technologies Ltd (Tata Motors Ltd Premises), Pune.**

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE**
**PILANI (RAJASTHAN)**

March, 2012

# DEVELOPMENT AND VALIDATION OF MODEL-BASED SOFTWARE FOR TA-ISG (TORQUE ASSIST-INTEGRATED STARTER GENERATOR) MOTOR CONTROLLER FOR HYBRID CAR APPLICATION

**BITS ZG629T: Dissertation**

by

Mr. Paul Lalthuamsanga

ID No: 2010HZ12772

**Dissertation work carried out at**

**Tata Technologies Ltd (Tata Motors Ltd Premises), Pune.**

Submitted in partial fulfillment of M.S. Software Systems degree programme

Under the Supervision of
Mr. Prasanta Sarkar, Project Manager,
Tata Technologies Ltd (Tata Motors Ltd Premises), Pune.

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE**
**PILANI (RAJASTHAN)**

March, 2012

## CERTIFICATE

*This is to certify that the Dissertation entitled "**DEVELOPMENT AND VALIDATION OF MODEL-BASED SOFTWARE FOR TA-ISG (TORQUE ASSIST-INTEGRATED STARTER GENERATOR) MOTOR CONTROLLER FOR HYBRID CAR APPLICATION"** and submitted by Paul Lalthuamsanga having ID-No. 2010HZ12772 for the partial fulfillment of the requirements of M.S. (Software Systems) degree of BITS embodies the bonafide work done by him/her under my supervision.*

*Signature of the Supervisor*

Prasanta Sarkar

*Place: PUNE*
*Date: 2 April 2012*

*Sd...*
*Mr. Prasanta Sarkar,*
*Project Manager,*
*Tata Technologies Ltd*
*(Tata Motors Ltd Premises), Pune.*

## Acknowledgements

*I would like thank to all my colleagues, my supervisor, my examiner and the department head, at Advance engineering Department, Tata Motors Ltd, Pune, for their support and encouragement in the fulfillment of this project. I would also like to thank God for giving me good health and good environment to work with good people around me.*

*Paul.*

## ABSTRACT

**Title:** Development and validation of Model based software for TA-ISG (Torque assist Integrated starter generator) motor controller for Hybrid Car Application.

**Supervisor:** Prasanta Sarkar.
**Name of student:** Paul Lalthuamsanga.
**Semester:** Second
**ID no:** 2010HZ12772.

This dissertation gives a way to apply a model-based design for designing and validation of motor controller for Hybrid Car application. It also highlights a specific methodology for modeling a microcontroller peripherals and PCB circuitry components in a MATLAB model.

All the system components including software model, processor execution model, PCB circuitry model, inverter model, motor model, engine model and Hybrid ECU functionality are all modeled in MATLAB simulink. Each component are modeled first, and then integrated one by one. Model in loop simulation (MIL) was performed for different functional checking of the control and also for PID parameter tuning. Fault detection logic and injection into the plan environment are also modeled and validated in the MIL simulation.

Code generation using Real-Time workshop of MATLAB from the model used in MIL was performed and porting was done to microchip microcontroller. All measurement and calibration attribute as taken into account during code generation and porting. Then, Hardware in loop simulation (HIL) with the controller hardware was performed for hardware and software integration validation. The above simulation done on MIL was again done on HIL system.

This report describes above work in detail.

**Signature of Student**

[illegible] 2/04/2012.

Paul Lalthuamsanga

Date: 2 April 2012

**Signature of Supervisor**

[illegible]

Prasanta Sarkar
Tata technologies Ltd (Tata Motors premises)
Pimpri, Pune.

# TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

## ABBREVIATION

| | |
|---|---|
| A | Motor Armature Red phase winding marking |
| A2L | ASAP2 standard file |
| AC | Alternating Current |
| ADC | Analog to digital converter |
| B | Motor Armature Yellow phase winding marking |
| C | Motor Armature Blue phase winding marking |
| CAN | Controller area Network |
| CPU | Central processing unit |
| DC | Direct Current |
| DUT | Device under Test |
| ECU | Electronic unit |
| EMF | electromagnetic force |
| FIFO | First in First out |
| GUI | Graphical user interface |
| HIL | Hardware in loop simulation |
| IGBT | insulated gate bipolar transistor |
| ISG | Integrated starter generator |
| Kd | Differential gain |
| Ki | Integral gain |
| Kp | Proportional gain |
| LTI | Linear time invariant |
| MBD | Model based design |
| MCU | Microcontroller unit |
| MIL | Model in loop simulation |
| MOSFET | metal–oxide–semiconductor field-effect transistor |
| PID | Proportional Integral Differential |
| PCB | Printed circuit board |
| PWM | Pulse width modulation |

| | |
|---|---|
| RTW | Real-time workshop |
| SD | Shutdown |
| SPI | Serial peripheral Interface |
| SCI | Serial Communication Interface |
| TA-ISG | Torque assist-integrated starter generator |
| Ts | Sampling time |
| RAM | Random access memory |
| IDE | Integrated development environment |
| GUIDE | Graphical user interface development environment |
| P | Number of Pole in machine |
| CCP | CAN calibration protocol |
| ISR | Interrupt service routine |
| PI | Proportional Integral |
| uC | Microcontroller |

# 1 CHAPTER 1: INTRODUCTION

## 1.1 INTRODUCTION AND PROBLEMS

Hybrid Car Technology is one of the technologies which could be the solution for global environment pollution problems. Such a technology would required many aggregates on the power-train of the vehicle which comprises of engine or fuel-cell, battery and motor. Hybrid Car can be classified according to the level of hybridization, called hybridization factor, the ratio of electric power provided to the wheel of the total power deliver to the wheel. Depending on values of the hybridization factor, there can be strong, medium, and mild hybridize Car. In a strong hybrid car application, electric motor could alone drive the wheel without engine assistance, but in a medium or mild hybrid car application, motor needs engine assist.

We will be focusing on the mild hybrid application where we are using a Claw Pole electric machine (TA-ISG) [Fig1] as a motor. In a mild hybrid application, the TA-ISG (Torque assist integrated starter generator) machine acts as the starter to crank the engine , provides torque assist to engine when the vehicle accelerates [Fig1], and also acts as the generator to charge the battery when it doesn't operate in motoring mode [Fig2]. This mild hybrid system using a TA-ISG can be adapted with conventional engine vehicles with only minimal system changes using belt system linking to engine. It can provide good improvement in fuel consumption.

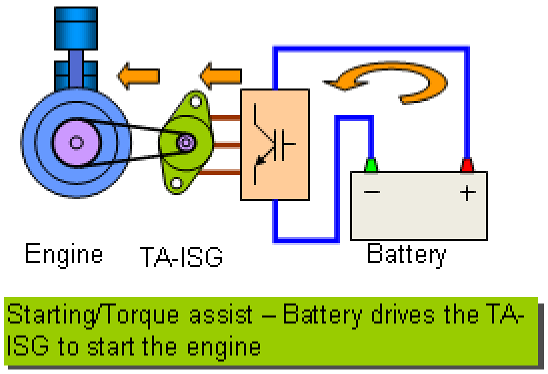

Engine
TA-ISG
Battery
−
+
Starting/Torque assist – Battery drives the TA-ISG to start the engine

Figure 1: Starting/Torque assist – Battery drives the TA-ISG to start the engine

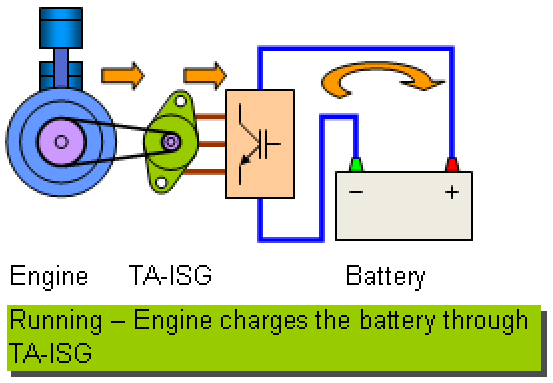


Figure 2: Running – Engine charges the battery through TA-ISG

These power-train systems require a complex computerized control system, communication, and so handling a hand-written C source codes algorithm would be a cumbersome. So, a modern approach of MATLAB model based design was used, where there can be a full traceability of model from requirement documents to the actual program codes dumped to the microcontroller. Early finding of software bugs as well as logical mistake can be done before actual testing on test bench. So this dissertation project addresses those issues using MIL (Model in loop simulation) and HIL (Hardware in loop simulation) validation methodology. Today, vehicle manufacturers are moving towards model based software development and simulation for easy maintenance and control of the complex control algorithm. In this project, we will be focusing on the development and validation of TA-ISG controller software in which model-based design approach was adopted and validation process using MIL and HIL was used, before deployment to the actual working environment.

## 1.2 MODEL_BASED DESIGN CONCEPT

Model-Based Design (MBD) is a mathematical and visual method of addressing problems associated with designing complex control, signal processing and communication systems. It is used in many motion controls, industrial equipment, aerospace, and automotive applications. Model-based design is a methodology applied in designing embedded software.

MBD provides an efficient approach for establishing a common framework for communication throughout the design process while supporting the development cycle ("V" diagram). In Model-based design of control systems, development is manifested in these four steps: 1) modeling a plant, 2) analyzing and synthesizing a controller for the plant, 3) simulating the plant and controller, and 4) integrating all these phases by deploying the controller. The model-based design paradigm is significantly different from traditional design methodology. Rather than using complex structures and extensive software code, designers can use MBD to define models with advanced functional characteristics using continuous-time and discrete-time building blocks. These built models used with simulation tools can lead to rapid prototyping, software testing, and verification. Not only is the testing and verification process enhanced, but also, in some cases, hardware-in-the-loop simulation can be used with the new design paradigm to perform testing of dynamic effects on the system more quickly and much more efficiently than with traditional design methodology.

### 1.2.1 STEPS IN MODEL –BASED DESIGN

The main steps in MBD approach are:

1. **Plant modeling**. Plant modeling can be data-driven or first principles based. Data-driven plant modeling uses techniques such as System identification. With system identification, the plant model is identified by acquiring and processing raw data from a real-world system and choosing a mathematical algorithm with which to identify a mathematical model. Various kinds of analysis and simulations can be performed using the identified model before it is used to design a model-based controller. First principles based modeling is based on creating a block diagram model that implements known differential-algebraic equations governing plant dynamics. A type of first principles based modeling is physical modeling, where a model is created by connecting blocks that represent physical elements that the actual plant consists of.
2. **Controller analysis and synthesis.** The mathematical model conceived in step 1 is used to identify dynamic characteristics of the plant model. A controller can be then be synthesized based on these characteristics.
3. **Offline simulation and real-time simulation**. The time response of the dynamic system to complex, time-varying inputs is investigated. This is done by simulating a simple LTI or a non-linear model of the plant with the controller. Simulation allows specification, requirements, and modeling errors to be found immediately, rather than later in the design effort. Real-time simulation can be done by automatically

generating code for the controller developed in step 3. This code can be deployed to a special real-time prototyping computer that can run the code and control the operation of the plant. If plant prototype is not available, or testing on the prototype is dangerous or expensive, code can be automatically generated from the plant model. This code can be deployed to the special real-time computer that can be connected to the target processor with running controller code. This way, controller can be tested in real-time against a real-time plant model.

4. **Deployment**. Ideally this is done via automatic code generation from the controller developed in step 3. It is unlikely that the controller will work on the actual system as well as it did in simulation, so an iterative debugging process is done by analyzing results on the actual target and updating the controller model. Model based design tools allow all these iterative steps to be performed in a unified visual environment.

### 1.2.2 ADVANTAGES OF MBD

Some of the notable advantages MBD offers in comparison to the traditional approach are:

- MBD provides a common design environment, which facilitates general communication, data analysis, and system verification between development groups.
- Engineers can locate and correct errors early in system design, when the time and financial impact of system modification are minimized.
- Design reuse, for upgrades and for derivative systems with expanded capabilities, is facilitated.

## 1.3 TA-ISG MACHINE AND CONTROL CONCEPT

Torque assist Integrated Starter Generator (TAISG) machine [Fig 3] is a claw-pole electric machine which has a rotor winding, and brush contact. The rotor flux can be individually controlled through H-Bridge in the Power inverter block [Fig 3]. Three phase winding armature are controlled through a 3-phase Bridge in the Power inverter block [Fig 3]. The rotor position is senses using hall sensors which are coupled to the TAISG machine [Fig

3]. Intelligent controller PCB was designed with microcontroller for computerize software control which interface with power inverter and machine.

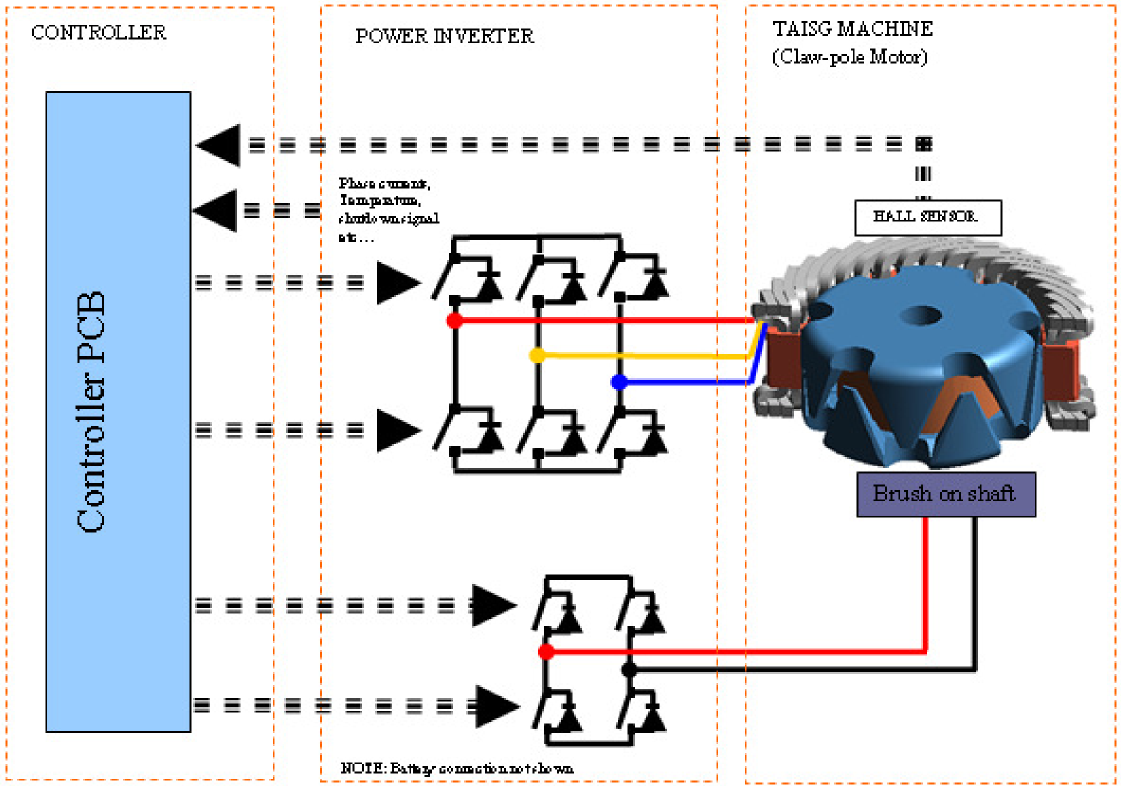


Figure 3: Block diagram of TA-ISG Control

Torque assist Integrated Starter Generator (TAISG) machine [Figure 3] is a claw-pole electric machine which has a rotor winding, and brush contact. The rotor flux can be controlled using an H-Bridge as shown in the Figure above. The Machine has 12 poles (P) armature star winding i.e. P/2 = 6 electrical cycles. Each electrical cycle will have 6 commutations sequence according to the table [Table 1] shown below for phase Red(A), phase Yellow (B) and phase Blue (C) conductions.

Table 1: Hall commutation sequences Table

| **sequence** | **Hall signal (A,B,C)** | **Phase Conduction** |
|---|---|---|
| 1 | 0 | A+C- |
| 2 | 1 | A+B- |
| 3 | 3 | C+B- |
| 4 | 7 | C+A- |
| 5 | 6 | B+A- |
| 6 | 4 | B+C- |

### 1.3.1 COMMUTATION SEQUENCE

Figure 4 shows an example of Hall sensor signals with respect to back EMF and the phase current. Figure 5 shows the switching sequence that should be followed with respect to the Hall sensors. The sequence numbers on Figure 4 correspond to the numbers given in Figure 5. For every 60 electrical degrees of rotation, one of the Hall sensors changes the state. It takes six steps to complete an electrical cycle. In synchronous, with every 60 electrical degrees, the phase current switching should be updated. However, one electrical cycle may not correspond to a complete mechanical revolution of the rotor.

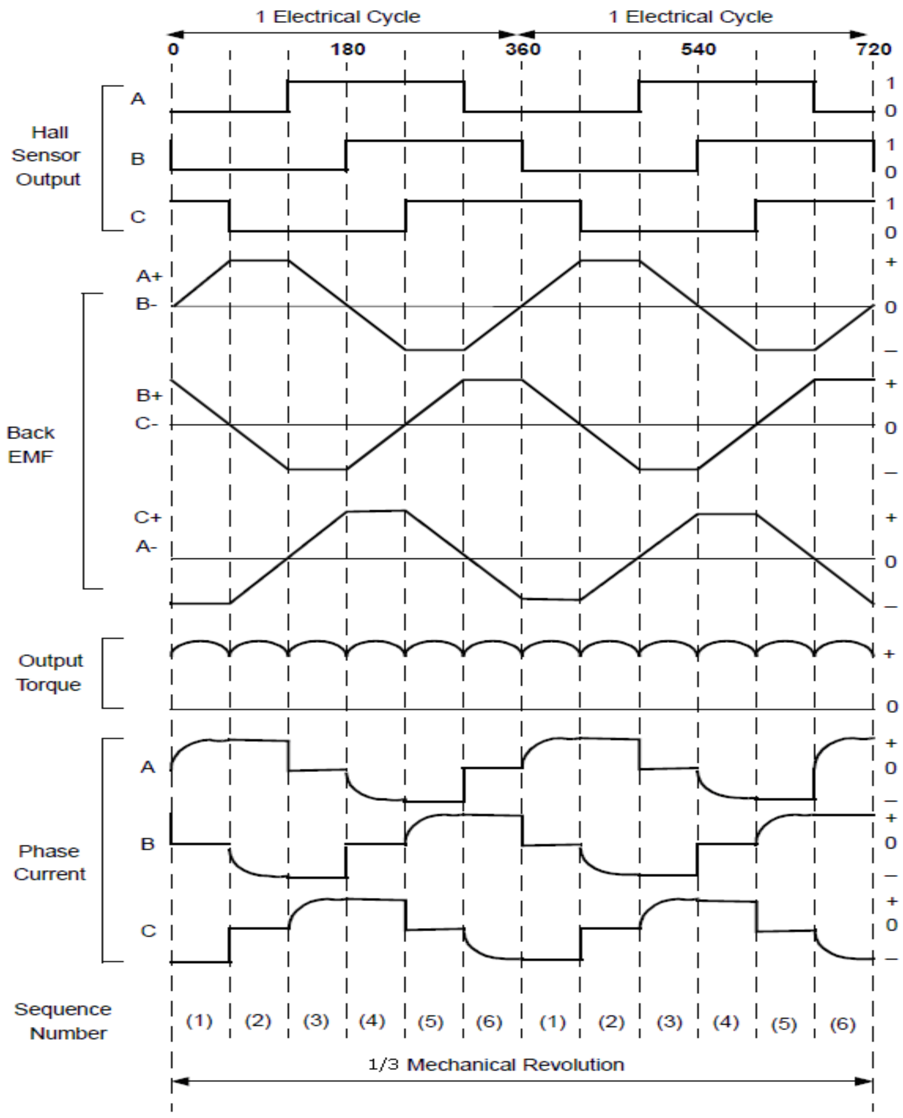


Figure 4: Hall sensor signal, back emf, output torque and phase current.

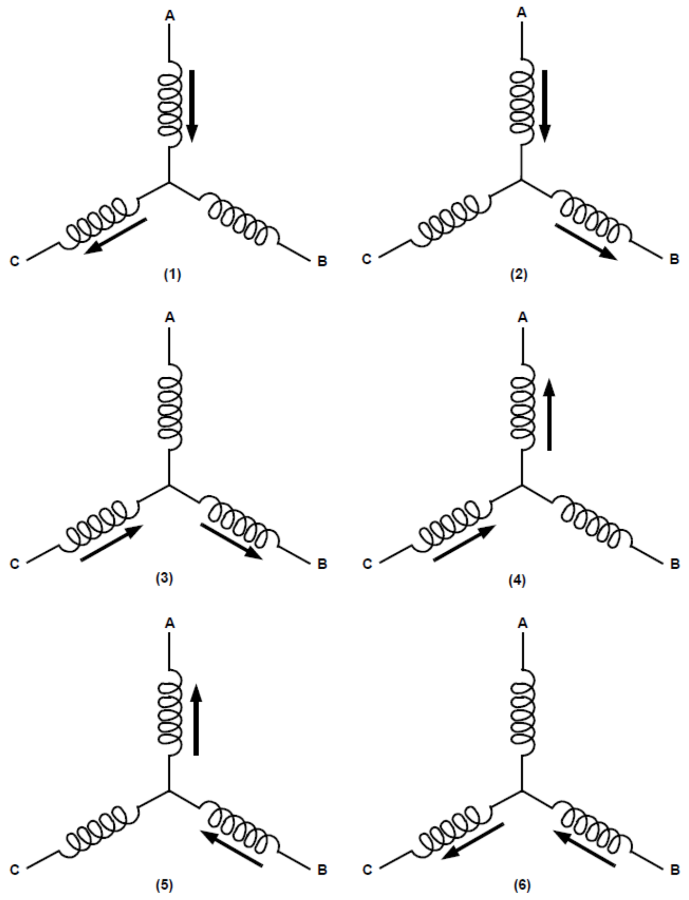


Figure 5: winding energizing sequence with respect to the hall sensor

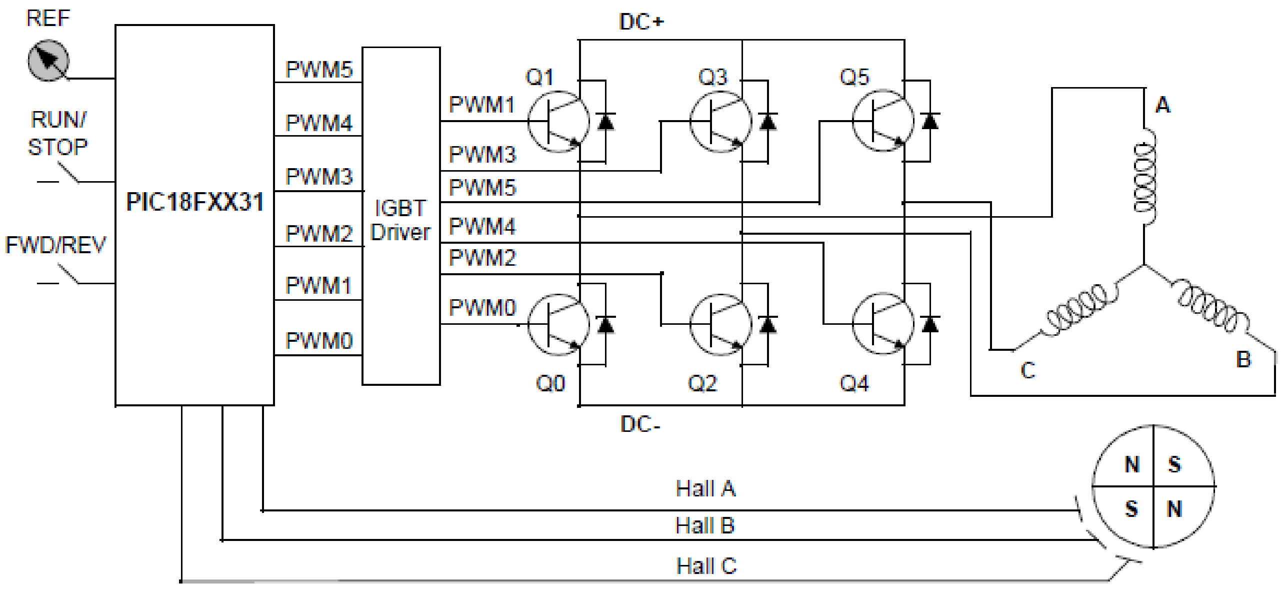


Figure 6: Control block diagram of Armature winding

The number of electrical cycles to be repeated to complete a mechanical rotation is determined by the rotor pole pairs. For each rotor pole pairs, one electrical cycle is completed. So, the number of electrical cycles/rotations equals the rotor pole pairs. Figure 6 shows a block diagram of the controller used to control a BLDC motor. Q0 to Q5 are the power switches controlled by the PIC18FXX31 microcontroller. Based on the motor voltage and current ratings, these switches can be MOSFETs, or IGBTs, or simple bipolar transistors.

Table 2 : Sequence of rotating the motor in clockwise direction

| Sequence # | Hall Sensor Input | | | Active PWMs | | Phase Current | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | | | A | B | C |
| 1 | 0 | 0 | 1 | PWM1(Q1) | PWM4(Q4) | DC+ | Off | DC- |
| 2 | 0 | 0 | 0 | PWM1(Q1) | PWM2(Q2) | DC+ | DC- | Off |
| 3 | 1 | 0 | 0 | PWM5(Q5) | PWM2(Q2) | Off | DC- | DC+ |
| 4 | 1 | 1 | 0 | PWM5(Q5) | PWM0(Q0) | DC- | Off | DC+ |
| 5 | 1 | 1 | 1 | PWM3(Q3) | PWM0(Q0) | DC- | DC+ | Off |
| 6 | 0 | 1 | 1 | PWM3(Q3) | PWM4(Q4) | Off | DC+ | DC- |

Table 3: Sequence of rotating the motor in counter-clockwise direction

| Sequence # | Hall Sensor Input | | | Active PWMs | | Phase Current | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | | | A | B | C |
| 1 | 0 | 1 | 1 | PWM5(Q5) | PWM2(Q2) | Off | DC- | DC+ |
| 2 | 1 | 1 | 1 | PWM1(Q1) | PWM2(Q2) | DC+ | DC- | Off |
| 3 | 1 | 1 | 0 | PWM1(Q1) | PWM4(Q4) | DC+ | Off | DC- |
| 4 | 1 | 0 | 0 | PWM3(Q3) | PWM4(Q4) | Off | DC+ | DC- |
| 5 | 0 | 0 | 0 | PWM3(Q3) | PWM0(Q0) | DC- | DC+ | Off |
| 6 | 0 | 0 | 1 | PWM5(Q5) | PWM0(Q0) | DC- | Off | DC+ |

Table 2 and Table 3 shows the sequences in which these power switches should be switched based on the Hall sensor inputs, A, B and C. Table 2 is for clockwise rotation of the motor and Table 3 is for counter clockwise motor rotation. This is an example of Hall sensor signals having a 60 degree phase shift with respect to each other. As we have previously discussed in the “Hall Sensors” section, the Hall sensors may be at 60° or 120° phase shift to each other. When deriving a controller for a particular motor, the sequence defined by the motor manufacturer should be followed. Referring to Figure 6, if the signals marked by PWMx are switched ON or OFF according to the sequence, the motor will run at the rated speed. This is assuming that the DC bus voltage is equal to the motor rated voltage, plus any losses across the switches. To vary the speed, these signals should be Pulse Width Modulated (PWM) at a much higher frequency than the motor frequency. As a rule of thumb, the PWM frequency should be at least 10 times that of the maximum frequency of the motor. When the duty cycle of PWM is varied within the sequences, the average voltage supplied to the stator reduces, thus reducing the speed. Another advantage of having PWM is that, if the DC bus voltage is much higher than the motor rated voltage, the motor can be controlled by limiting the percentage of PWM duty cycle corresponding to that of the motor rated voltage. This adds flexibility to the controller to hook up motors with different rated voltages and match the average voltage output by the controller, to the motor rated voltage, by controlling the PWM duty cycle. There are different approaches of controls. If the PWM signals are limited in the microcontroller, the upper switches can be turned on for the entire time during the corresponding sequence and the corresponding lower switch can be controlled by the required duty cycle on PWM. The potentiometer, connected to the analog-to-digital converter channel in Figure 6, is for setting a speed reference. Based on this input voltage, the PWM duty cycle should be calculated.

### 1.3.2 CLOSED LOOP CONTROL

The speed/Torque can be controlled in a closed loop by measuring the actual speed of the motor, or current along the phases. The error in the set speed/Torque and actual speed/torque is calculated. A Proportional plus Integral plus Derivative (P.I.D.) controller can be used to amplify the speed/torque error and dynamically adjust the PWM duty cycle. For low-cost, low-resolution speed requirements, the Hall signals can be used to measure the speed feedback. A timer from the PIC18FXX31 can be used to count between two Hall transitions. With this count, the actual speed of the motor can be calculated. For high-resolution speed measurements, an optical encoder can be fitted onto the motor, which gives two signals with 90 degrees phase difference. Using these signals, both speed and direction of rotation can be determined. Also, most of the encoders give a third index signal, which is one pulse per revolution. This can be used for positioning applications. Optical encoders are available with different choices of Pulse per Revolution (PPR), ranging from hundreds to thousands.

### 1.3.3 FIELD WINDING CONTROL

Field winding in a TAISG machine are separately control using H-Bridge where we can control the amount of current on rotor winding and the direction of current. Figure 7 shows the detail block diagram as below.

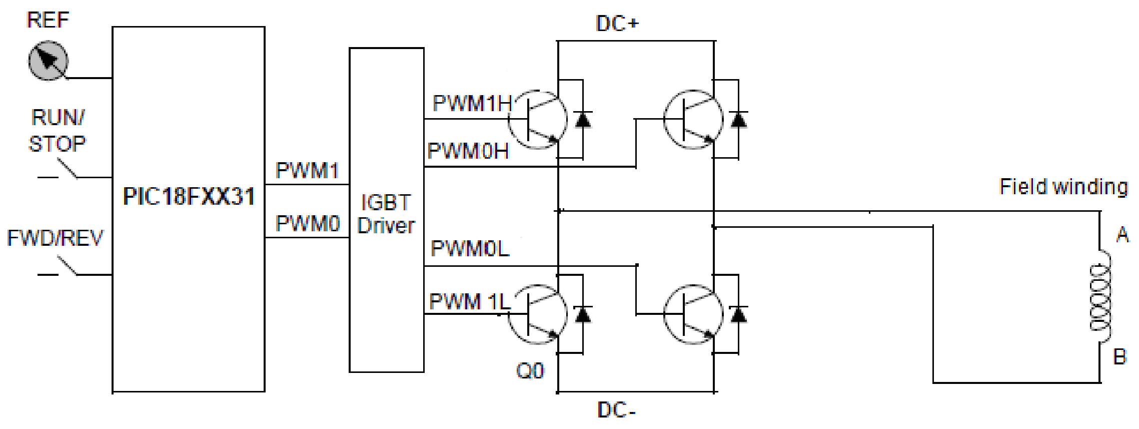


Figure 7 : Field winding control block diagram

IGBT or MOSFET driver as shown in the figure above is an H-Bridge driver which gives a complementary PWM on each leg of the high and low MOSFET/IGBT. While PWM1 is in PWM and PWM0 is low, the complementary PWM will appear on the PWM1H and PWM1L signal, and PWM0H will be low and PWM 1L will be high. Thereby, the current will be in the direction of A to B in the rotor/field winding, and the amount of current would be control using duty on PWM1.

During motoring operation, a full duty is applied which will give maximum rotor flux at specific current direction say A to B. Field duty can also be used to control rotor flux during alternating operation and field weakening operation.

### 1.3.4 HALL SENSOR

As mounting diagram shown below, magnetic cup is mounted on the rotor, and a fixed hall sensor is mounted on the body of the motor. Magnetic cup consist of alternate 6 North poles and 6 South poles magnet ring. When the magnetic cup rotate the three hall sensor give a sequence of  hall signal as shown in the consecutive figure below.

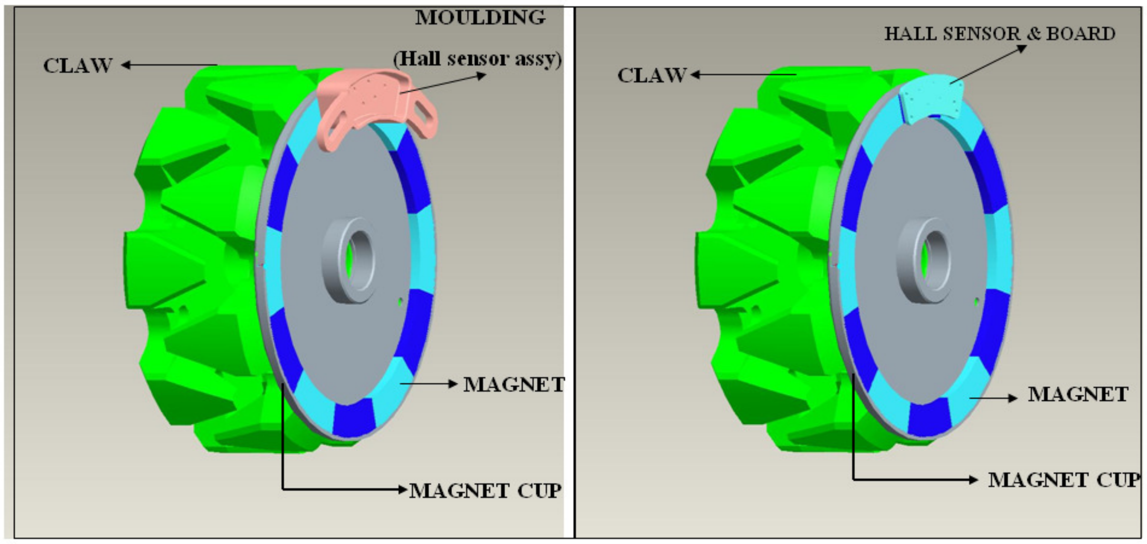


Figure 8 : Hall sensor mounting

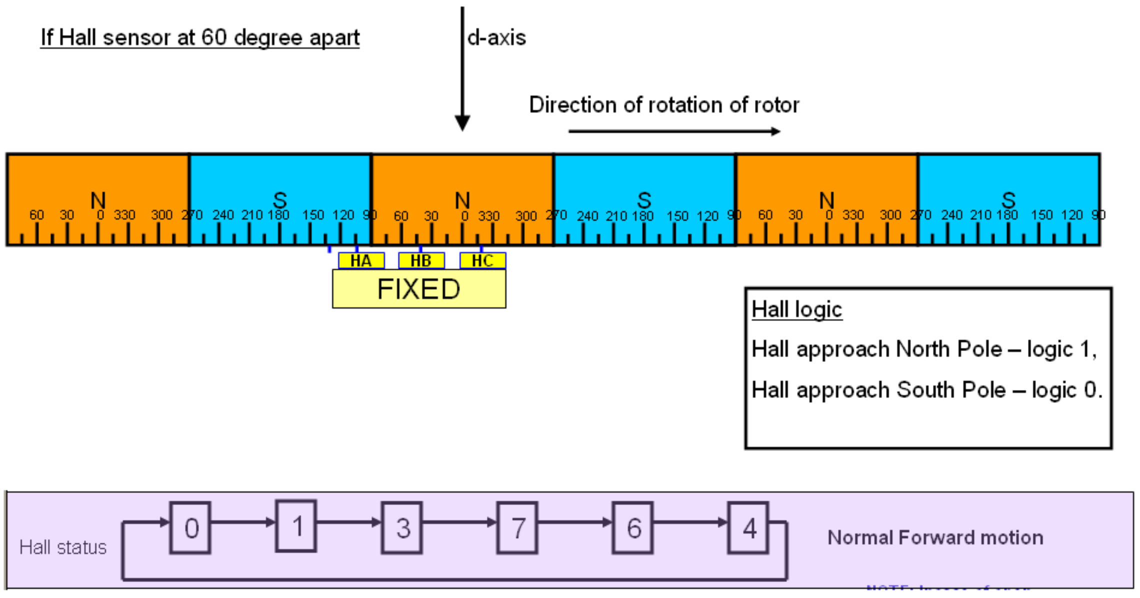

If Hall sensor at 60 degree apart
d-axis
Direction of rotation of rotor
N
S
N
S
N
S
HA
HB
HC
FIXED
Hall logic
Hall approach North Pole – logic 1,
Hall approach South Pole – logic 0.
Hall status
0
1
3
7
6
4
Normal Forward motion

Figure 9 : Hall sensor pattern at 60 degree apart

The hall sensor at 60 degree apart was used in our machine so the hall sequences will be 000(0), 001(1), 011(3), 111(7), 110(6), and 100(4). So, out of the 8 combination hall signals 101(5) and 010(2) will not be appearing. There will be 6 hall changes within one electrical cycle, so 36 hall changes in one mechanical rotation for 12 pole machine.

## 1.4 PROJECT FOLDER STRUCTURE

Specific folder structure is maintained in this project as shown below

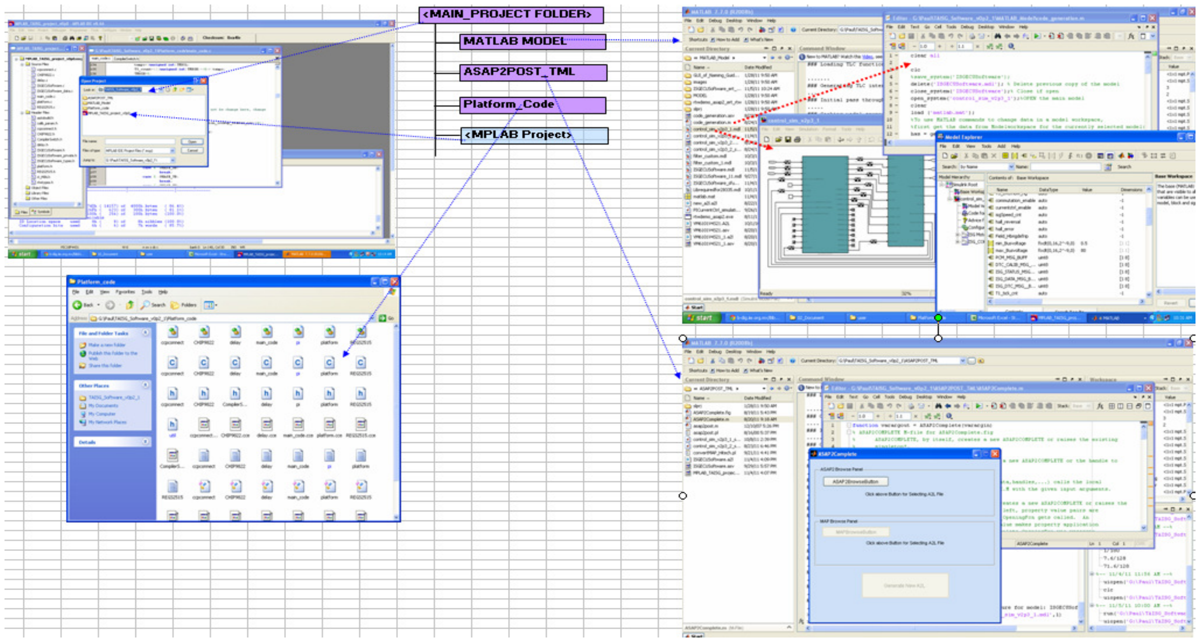


Figure 10: Project folder structure

The project folders are maintained in a hierarchical manner do to its complex tool chain, and are mentioned below:-

- **Main < project folder>:** This main folder can be given any name. This folder contains the MPLAB project and its aggregates. It contains sub-folder as mentioned below:-
    - **MATLAB model:** This folder contains MATLAB model and all MATLAB related files associated with the model
    - **ASAP2POST_TML**: This folder contain ASAP2 complete file generator M-script, GUI, and contain hex file and A2L files.
    - **Platform_code:** This folder contains the hand written code like platform code-hardware driver.c, ccp.c, etc.

# 2 CHAPTER 2: MODELING

## 2.1 Modeling:

Modeling is done on MATLAB Simulink [1] using native blocksets of Simulink library. Modeling a control system involved two parts – Plan model and Controller model. Controller model consist of the parts of system on focus, and the plan model consist of the remaining system which is on the scope of the design. Since the focus of this development is on motor controller PCB design and development, we segregate the Controller PCB part from the whole system as a one subsystem named ISG_CONTROLLER_PCB as show in Figure below, and the rest of the system including Power inverter, battery, engine and vehicle models sit on the plan part of the model. Figure below shows the topmost level model of the MATLAB Model, which consist of Plan model and controller PCB model.

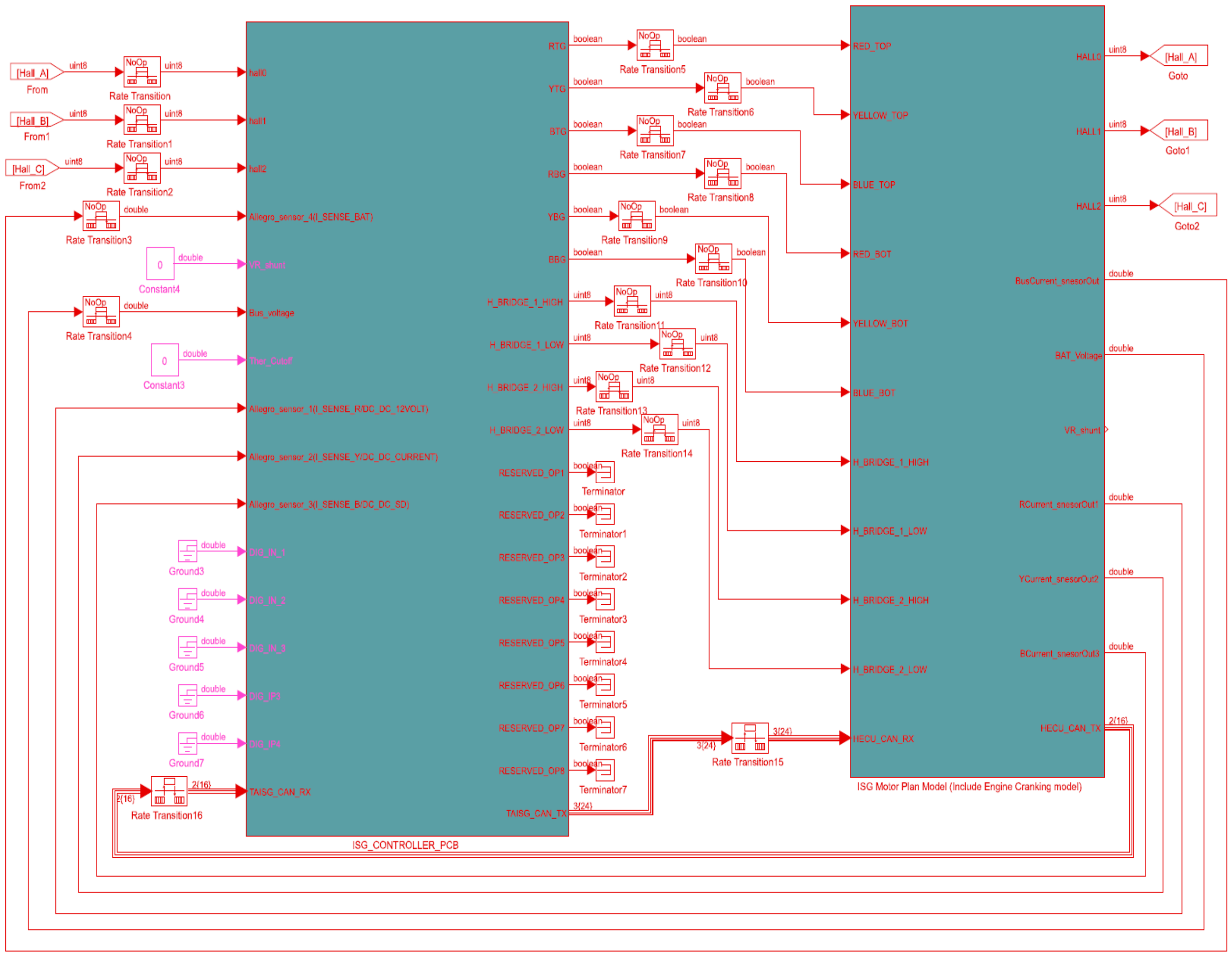


Figure 11 :Top-level model containing controller and plan

## 2.2 Modeling of Plan Model

Modeling is done on MATLAB Simulink using native blocksets of Simulink library. Plan model comprises of power inverter, motor, battery, engine, vehicle ECU, and the Vehicle subsystem as shown in [**Figure 12**]. SimPower System physical blocksets of Simulink library are used in the model for power MOSFETs, resistors, inductors, capacitors, and voltage sources. Apart from the motor, inverter and battery subsystem, the hybrid ECU and Engine cranking Load torque are included. Those hybrid ECU and Engine subsystems are modeled for future addition to simulate the whole vehicle dynamics for controller software validation.

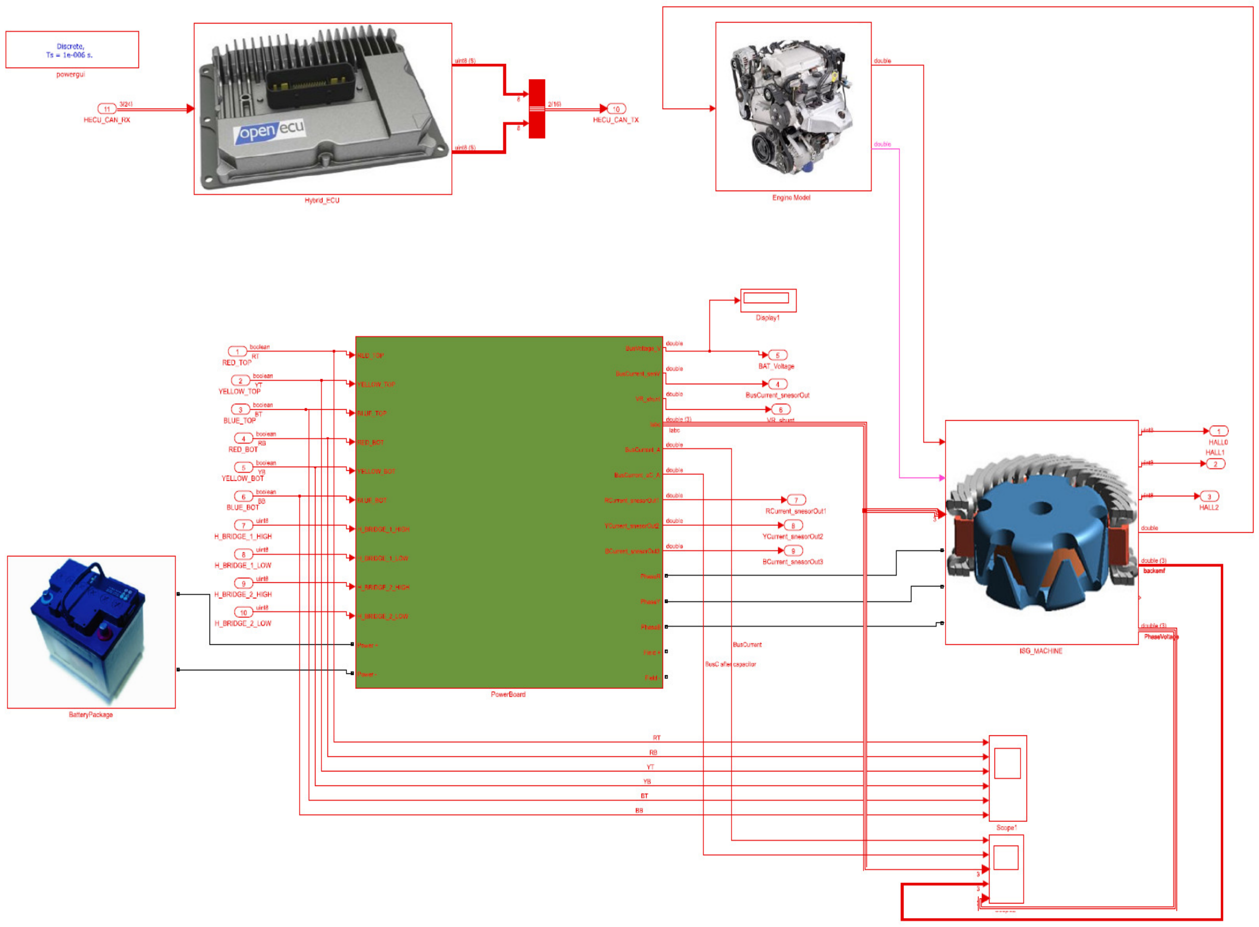


Figure 12 : Plan model comprises of different components of vehicle.

## 2.2.1 Power Inverter model

This model consists of 3 phase inverter, DC link capacitor, H-bridge, H-bridge shunt resistor, and current sensors. As shown in figure below, input signal to inverter are switching signal of MOSFET. Since these MOSFETs and Diodes are physical model of SimPower System it can be connected as like a wired connection in a physical world. The black color represents physical wire, and red lines represent Simulink signals. SimPower System sensors library blocks like current measurement blocks, voltage measurement blocks are used to give feedback signal to controller through current sensor subsystem as shown in gray box below.

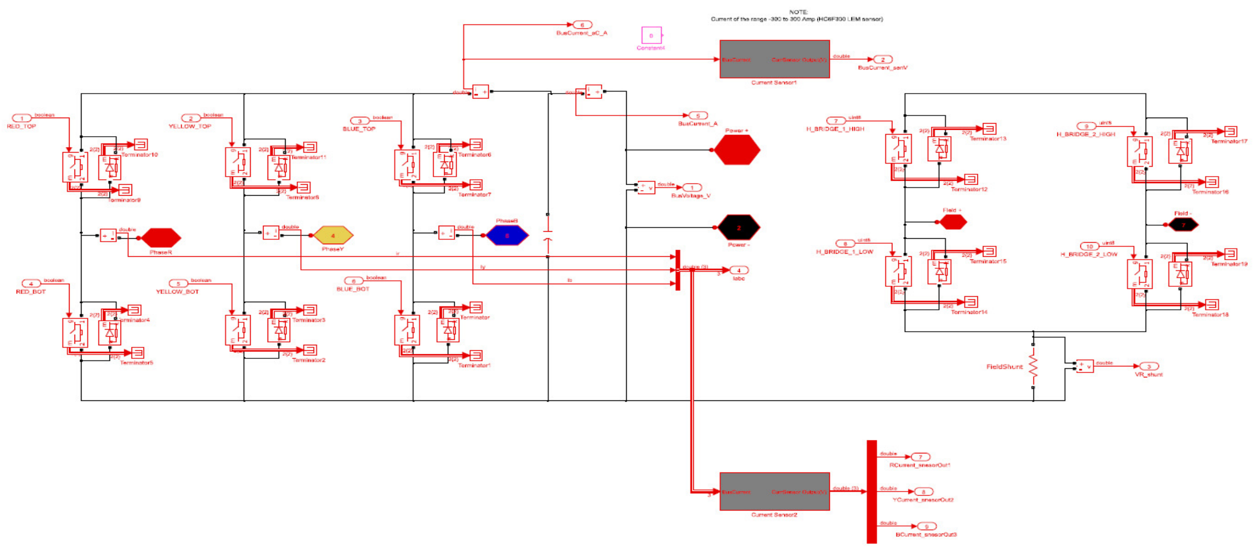


Figure 13 : Power Inverter model of the motor controller

### 2.2.1.1 Current sensor

Current sensor subsystem converts the ampere value to sensor output voltage level (0V to 5V) as shown below, where 2.5 V represent zero current.

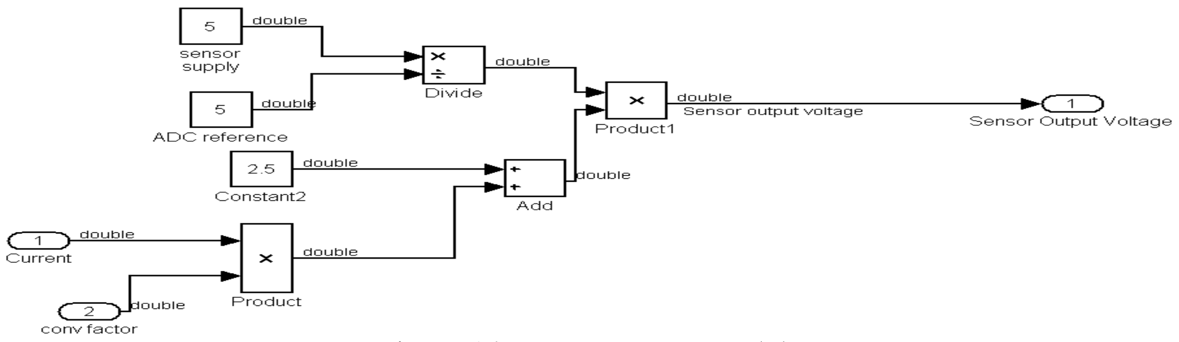


Figure 14: current sensor model

### 2.2.2 Motor model

Motor can be simulated as shown in figure below using SimPower system model and logical or mathematical Simulink subsystem model. Armature winding are modeled in star configuration using resistors, inductors and voltage sources as shown by black wire connection. Back emf is simulated using voltage source which are adjusted from Simulink signals subsystems depending upon speed of motor. Back emf is generated using mathematical equations from MAIN_SIM_LOOP as shown below. This subsystem consists of Simulink subsystem like electromagnetic torque, main simulation loop, hall-sensor and hall-sensor simulation subsystems.

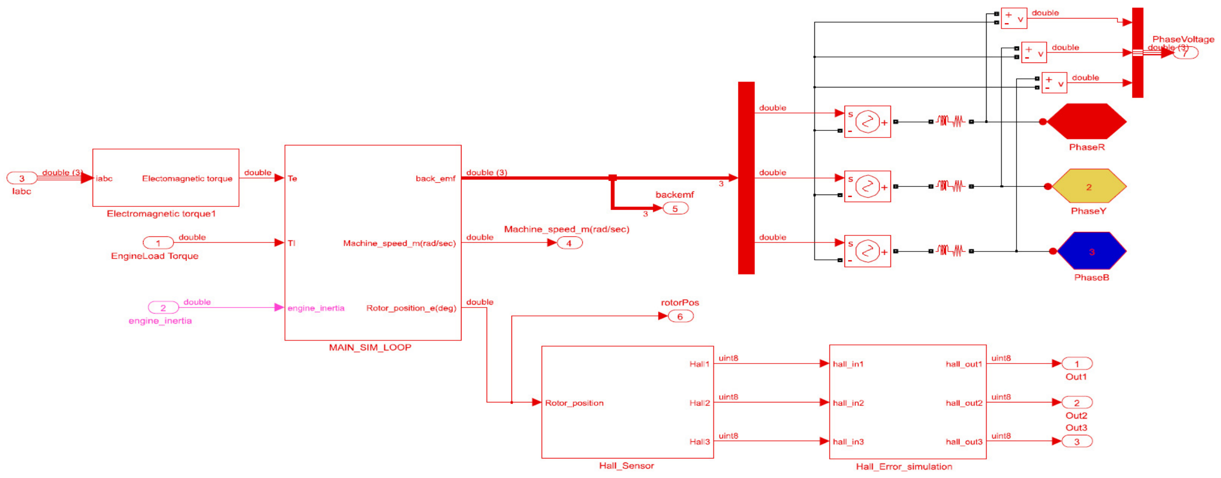


Figure 15 : Motor 3-phase star winding simulation with back emf generator.

#### 2.2.2.1 ASSUMPTIONS IN MOTOR MODELING:

- Magnetic circuit saturation is ignored.
- Stator resistance, self and mutual inductance of all phases are equal and constant.
- Hysteresis and eddy current losses are eliminated.
- All semiconductor switches are ideal.
- Mutual inductance is not considered.

### 2.2.2.2 Electromagnetic Torque

The torque generated by the machine depends on the phase current of each winding, flux pattern reference and machine constant.

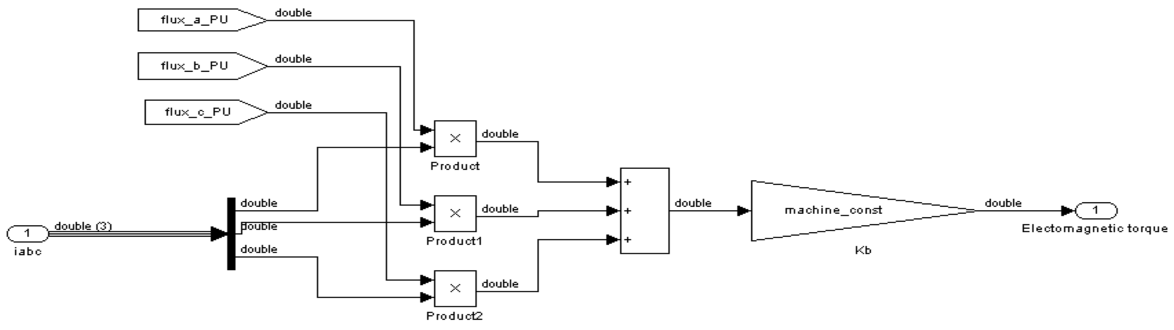


Figure 16: Electromagnetic torque model

The equation shown below is realized in the above model

Equation 1: Electromagnetic torque equation

$$T_e = K_m\left(I_a\phi_a + I_b\phi_b + I_c\phi_c\right)$$

Where,

| | |
|---|---|
| $T_e$ | - Electromagnetic Torque |
| $K_m$ | - Machine constant |
| $I_a, I_b, I_c$ | - Phase current |
| $\phi_a, \phi_b, \phi_c$ | - Flux pattern in per unit |

The flux pattern is generated from sinusoidal back-emf subsystem shown in Figure 17: main simulation loop subsystem.

### 2.2.2.3 Main simulation Loop subsystems

This subsystem consists of two subsystems – torque and speed loop, and sinusoidal back emf subsystem. Machine inertia is not considered and is left with constant zero as shown below

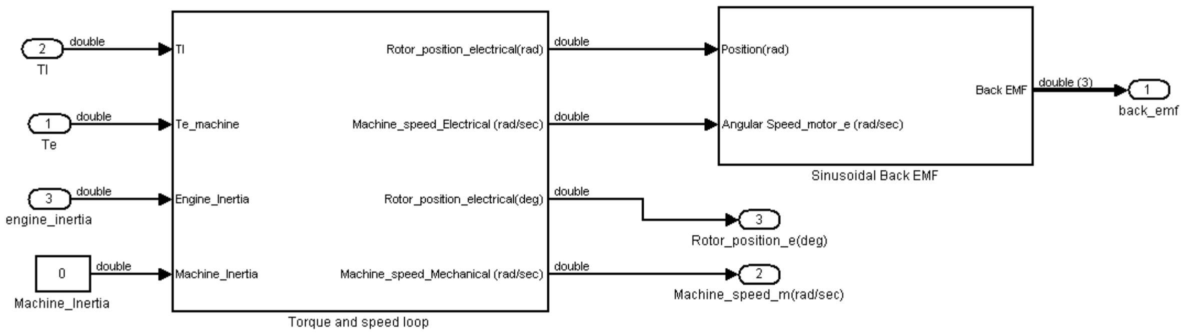


Figure 17: main simulation loop subsystem

#### *2.2.2.3.1 Torque and Speed Loop equation and model*

The core of the simulation lies on the load torque equation shown below

Equation 2 : Torque speed equation

$$T_e - T_\ell = J\frac{\partial\omega}{\partial t}$$

Where,

$T_e$ - Electromagnetic torque produces by the motor

$T_\ell$ - Load torque applied

$J$ - Total inertia which includes Engine inertia, the machine inertia and all other component on the shaft.

$\frac{\partial\omega}{\partial t}$ - Angular speed, where, $\omega = 2\pi n_s$ such that $n_s$ in rps (revolution per sec).

The above equation can be realized a MATLAB Simulink model as shown below.

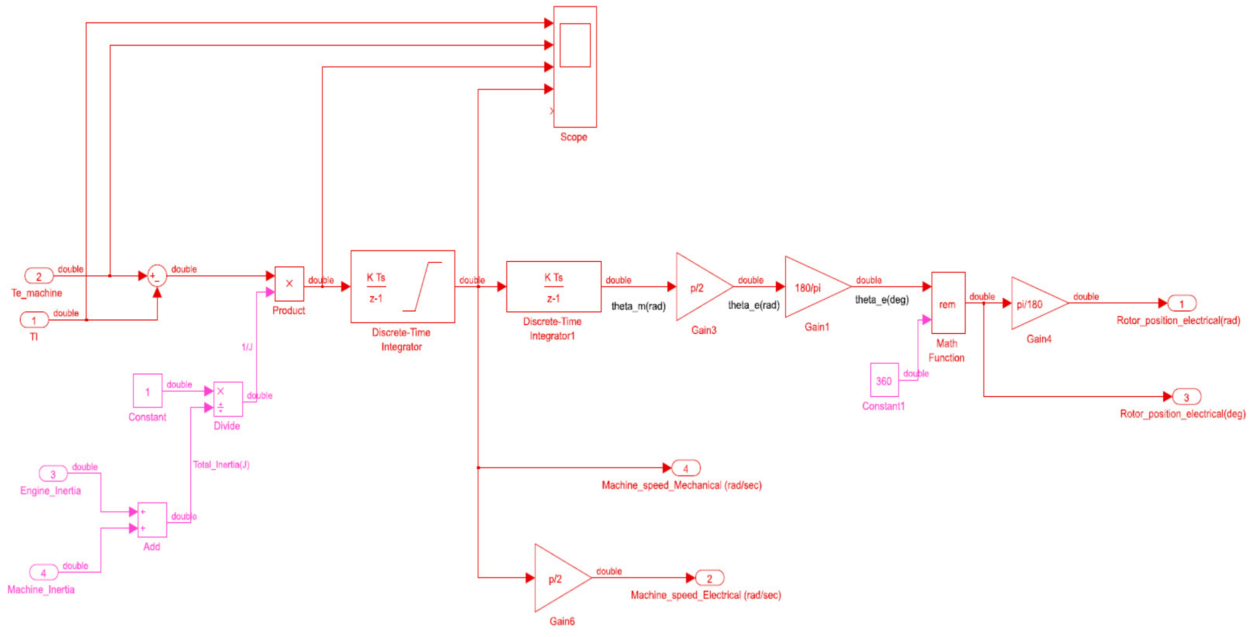


Figure 18: Torque speed equation realization in Simulink.

Load Torque $T_l$ is simulated from the frictional model of Engine, given in the Engine subsystem. Load Torque from engine $T_l$ varies with the speed of the crank shaft of the engine which is couple to ISG machine using belt and the belt ratio should also be consider in the model.

### *2.2.2.3.2 Sinusoidal back emf generation model*

The back emf pattern can be generated by a sinusoidal signal generator, where phase shift are applied for each phase as shown below. Position in radian is taken as an input which varies from 0 to 2*pi, and repeat itself. The frequency and amplitude will be decided by the speed of the machine.

$$E_a = K_e \omega_m F(\theta_e)$$
$$E_b = K_e \omega_m F(\theta_e - \frac{2\pi}{3})$$
$$E_c = K_e \omega_m F(\theta_e + \frac{2\pi}{3})$$

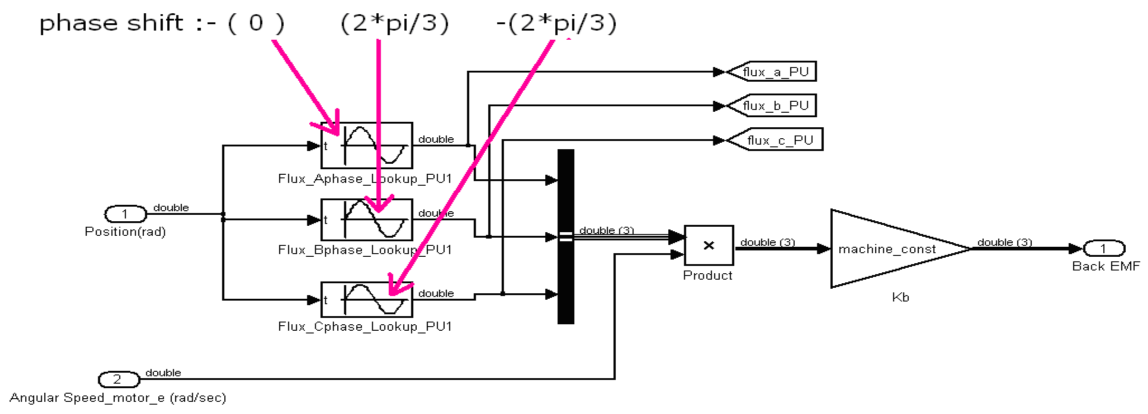


Figure 19 : sinusoidal back emf model

### 2.2.2.4 Hall sensor

Hall sensor can be simply model in Simulink using look-up table where the pattern of hall signal is generated at different rotor position from 0 to 360 degree. The input to this subsystem is a rotor position which varies from 0 to 360 repetitively depending on the machine speed.

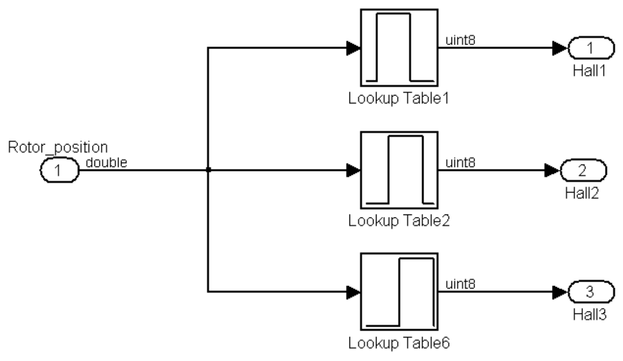


Figure 20 : Hall sensor model

#### 2.2.2.5 Hall error simulation

Hall sensor fault condition can be created by the model shown below. Using this subsystem fault can be injected during run-time of the simulation manually. Shorted to Ground and shorted to supply of hall sensor signal for each combination can be simulated.

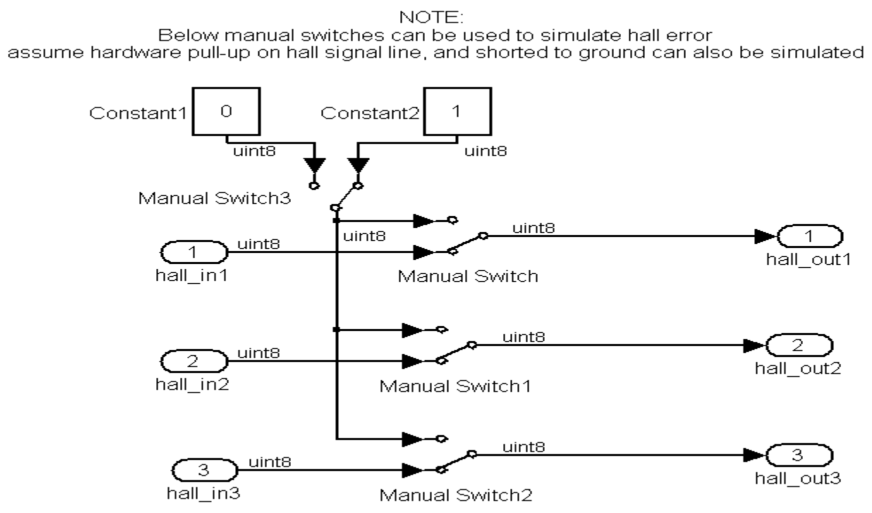
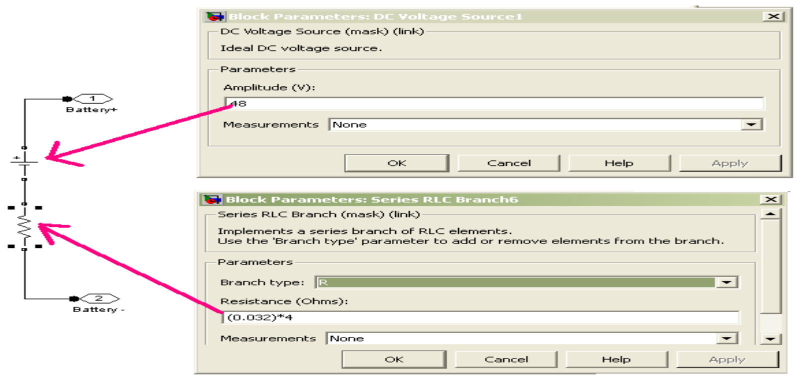

NOTE:
Below manual switches can be used to simulate hall error
assume hardware pull-up on hall signal line, and shorted to ground can also be simulated
Constant1
0
Constant2
1
uint8
Manual Switch3
1
hall_in1
Manual Switch
hall_out1
2
hall_in2
Manual Switch1
hall_out2
3
hall_in3
Manual Switch2
hall_out3

Figure 21: Hall error simulation

### 2.2.3 Battery model

Battery is modeled using an ideal voltage source with internal resistance of SimPower system block as shown below. Four 12V battery are used which has an internal resistance of 32 mOhm.



Block Parameters: DC Voltage Source1

DC Voltage Source (mask) (link)

Ideal DC voltage source.

Parameters

Amplitude (V): 48

Measurements: None

Block Parameters: Series RLC Branch6

Series RLC Branch (mask) (link)

Implements a series branch of RLC elements.
Use the 'Branch type' parameter to add or remove elements from the branch.

Parameters

Branch type: R

Resistance (Ohms): (0.032)*4

Measurements: None

Figure 22: Battery model

### 2.2.4 Engine model

The friction torque and inertia components of the engine are only modeled here. The cranking load behavior of the engine can be simulated using this model.

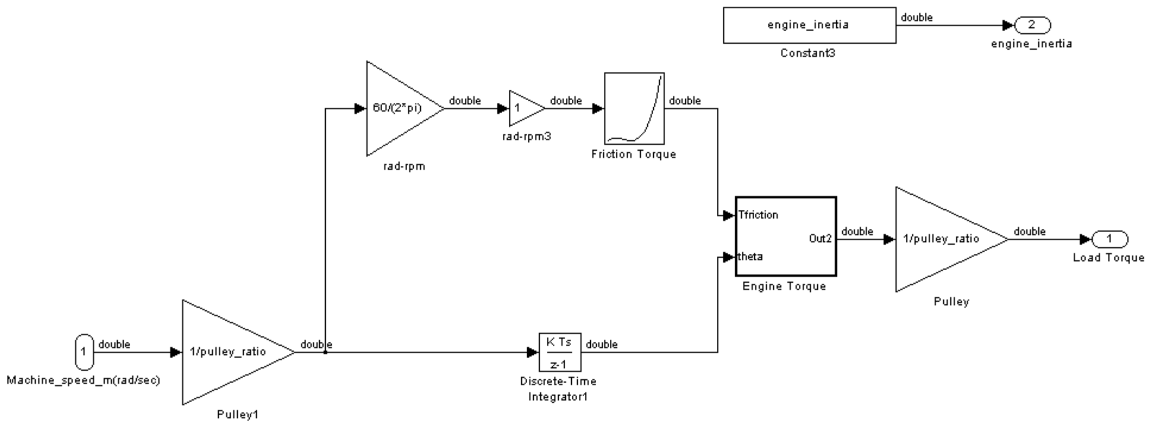


Figure 23: Engine model

### 2.2.5 Hybrid ECU model

Function of Hybrid is the main control system of the whole vehicle which send command to all other ECU in the vehicle. In this simulation CAN message is only simulated. Please note that motor controller are controlled only by Hybrid ECU through CAN message as shown below.

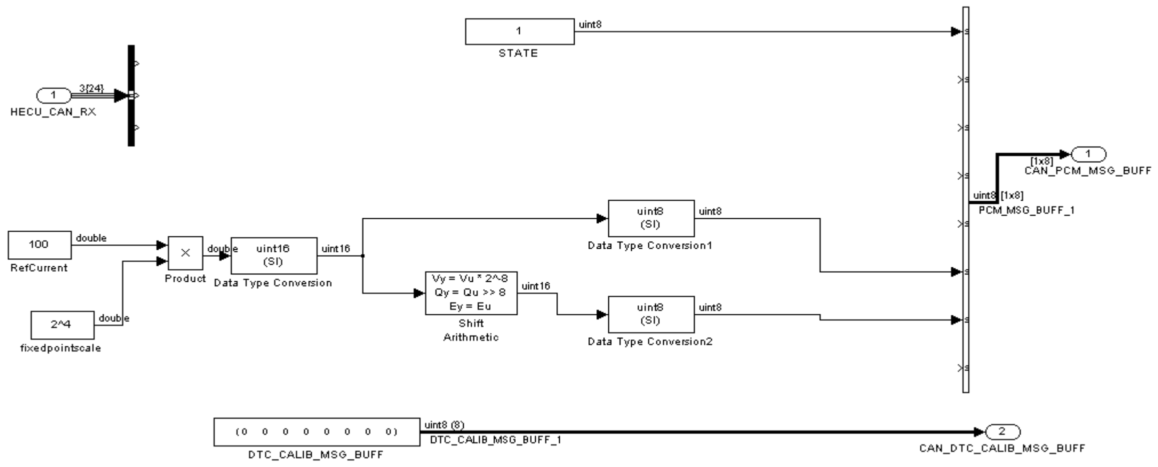


Figure 24: hybrid ECU model

## 2.3 Modeling of Controller

Modeling of the Controller includes PCB circuitry, Microcontroller and its Software algorithm. The complexity of this controller model is that the hardware component of the PCB are modeled to an extent of its logical functions with an aim to be able to auto-code directly from subsystem of the <model>.mdl file of the model used in the MIL simulation. Processor execution timing and interrupts are also modeled in MATLAB Simulink which are as per the specific configuration of the 18F microcontroller used. This is one of the most interesting parts of this dissertation project that it is one typical way of modeling, which could help designer working on the same model for both simulation and auto-coding. Part of software subsystem to be auto-coded will be discuss on the later part, and porting of generated code to the microcontroller platform code will also be discuss on Code generation chapter of this report.

Segregation of electrical circuitry part of the controller PCB are done as it is on the physical PCB, and part of the digital circuitry inside the microcontroller are all model on its logical aspect on module-wise subsystem. Software parts of the system are then model inside as one subsystem around the hardware and digital electronics subsystem model. This type of modeling would help to solve software bugs before actual implementation. It also gives an idea of the hardware-software interface, hardware functionality of the peripherals.

### 2.3.1 ASSUMPTION IN CONTROLLER MODELING

1. Active component of the electronics like capacitor, inductor, charging and discharging are not model. Only logical functions of the chips and the circuit are modeled.
2. Some logical functions of some chips are not considered for simplicity.
3. Sampling rate is fixed to 1 usec, fixed-step. This is done due to better PWM generation of 10/7.8 KHz (100/128 usec period), we can have 1/0.7v% resolution of duty.
4. Electromagnetic interference and electrostatic discharge are not considered.
5. Filter circuit functions like filtering delay etc are neglected in the model.
6. Configuration registers setting of MCU and its peripherals are fixed.

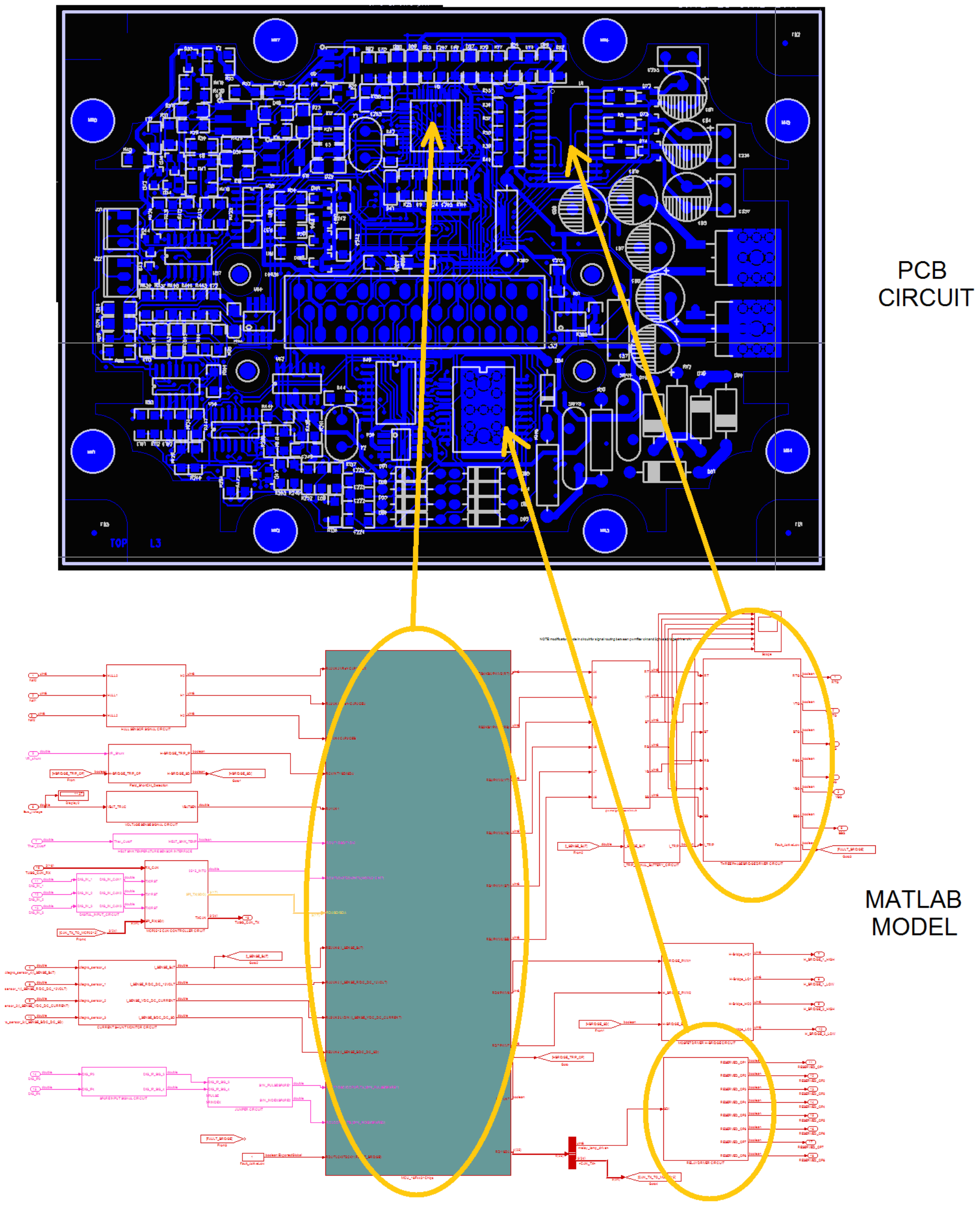


Figure 25 : Controller PCB model, and traceability

As shown in the figure above, modeling of functionality of PCB circuitry and IC are possible by looking into the actual the functional requirements in the simulation. All the PCB circuitries are modeled and are put to separate subsystems.

PCB consists of subsystem listed as below:-

1. Current shunt monitor.
2. Digital input circuit.
3. Field short circuit detection.
4. Hall sensor signal circuit.
5. Heat sink temperature sensor circuit.
6. I-Trip signal Battery circuit.
7. Jumper circuit.
8. MCP2515 CAN Controller circuit.
9. MCU 18F4431 Chips.
10. MOSFET driver H-Bridge circuit.
11. Relay driver circuit.
12. Spare input signal circuit.
13. Three-phase Bridge driver circuit.
14. PWM signal filter circuit.
15. Voltage sense signal circuit.

Since, we neglect the filtering logic in our modeling and simulation, we could directly connect input and output in many subsystem as shown in the next section.

### 2.3.2 Current shunts monitor Model

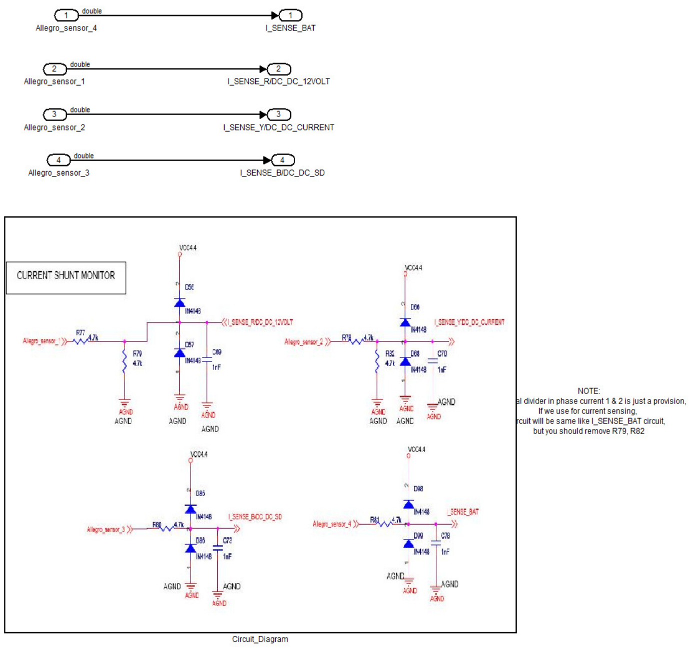


Figure 26 : Current shunt monitor circuit Model

### 2.3.3 Hall sensor signal circuit model

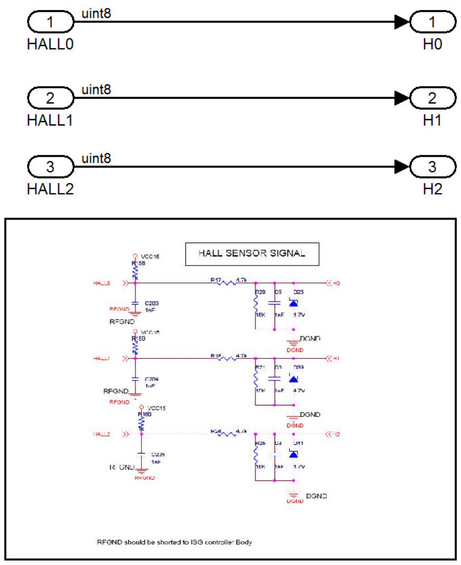
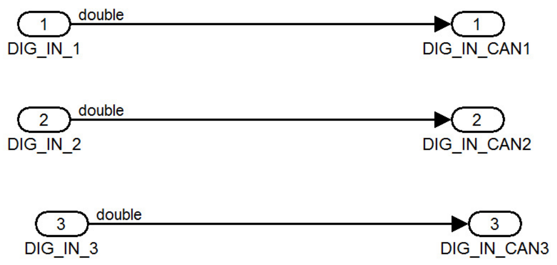


Figure 27 : Hall sensor signal circuit model

## 2.3.4 Digital Input Circuit Model



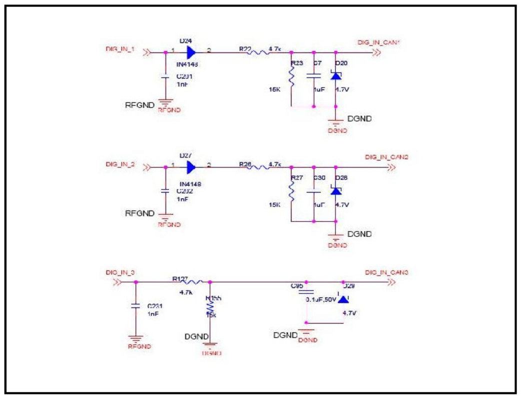
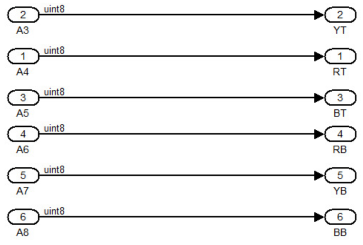


Figure 28 : Digital Input Circuit Model

## 2.3.5 PWM Signal Filter circuit Model



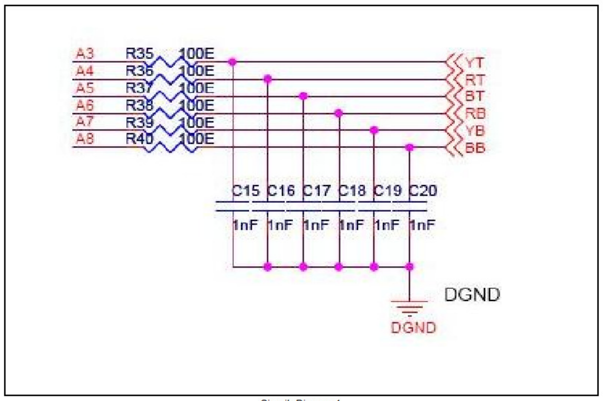


Figure 29 : PWM Signal Filter circuit Model

### 2.3.6 Spare input signal circuit model

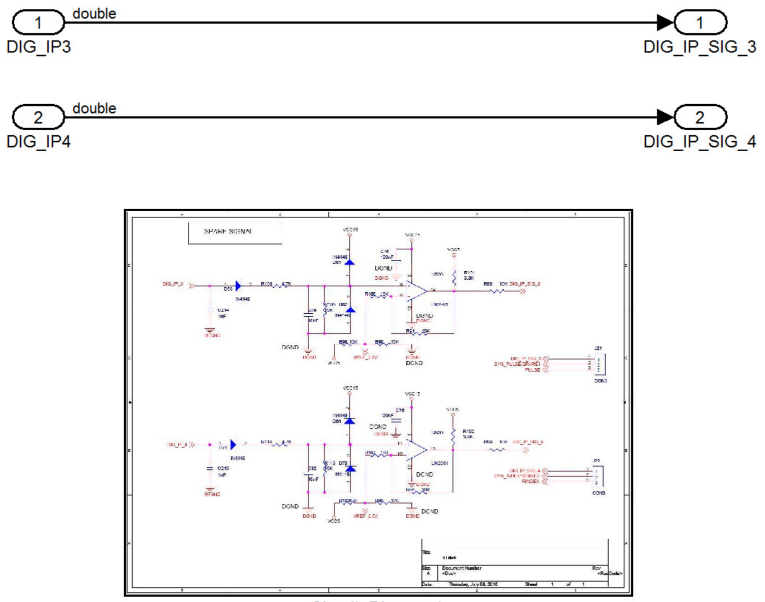


Figure 30 : Spare input signal circuit model

### 2.3.7 MCP2515 CAN Controller circuit model

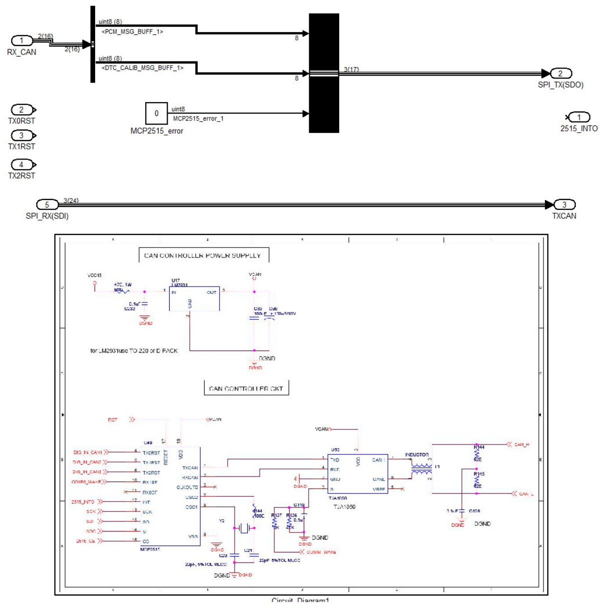


Figure 31: MCP2515 CAN Controller circuit model

## 2.3.8 Relay driver output circuit model

The main function of Relay driver is to latch out serial data from SPI to a parallel logic, which are modeled as shown below.

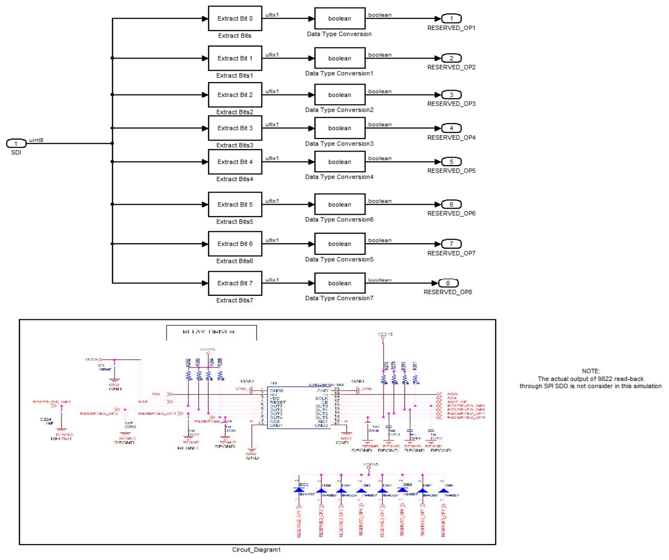


Figure 32 : Relay driver output circuit model

## 2.3.9 Field short circuit detection circuit model

As shown in figure 10, the field shunt resistor voltage is taken as input to this circuit which is a very low voltage, and is amplify and given to a comparator circuit. It is then compare with a know threshold voltage which is selected depending on trip current required. Since the H-Bridge driver chip shutdown does not latch the SD signal, microcontroller output latch is provided to latch through software.

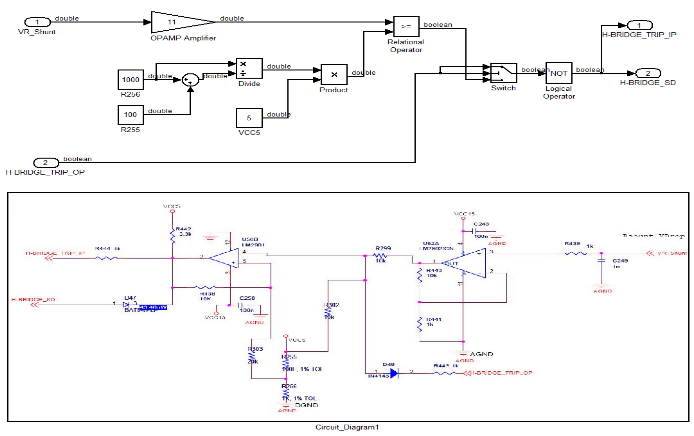


Figure 33 : Field short circuit detection circuit model

### 2.3.10 Heat Sink Temperature sensor interface model

Thermistor sensor from the power inverter heat sink is connected to the circuit shown below. Threshold is set using potential divider circuit which is model the same way in a MATLAB Simulink model.

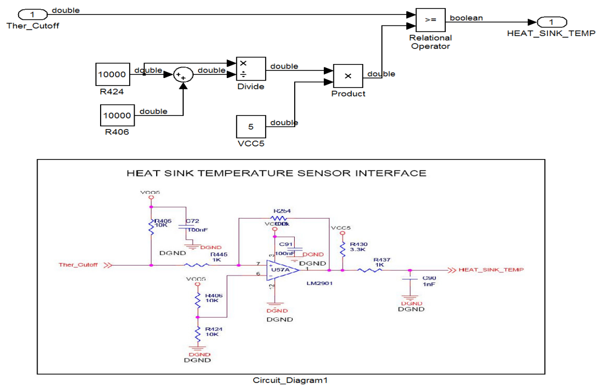


Figure 34 : Heat Sink Temperature sensor interface model

### 2.3.11 I Trip Signal Battery circuit model

This subsystem senses the specific threshold voltage (over voltage) of battery, which give trip signal to the 3-phase driver chip.

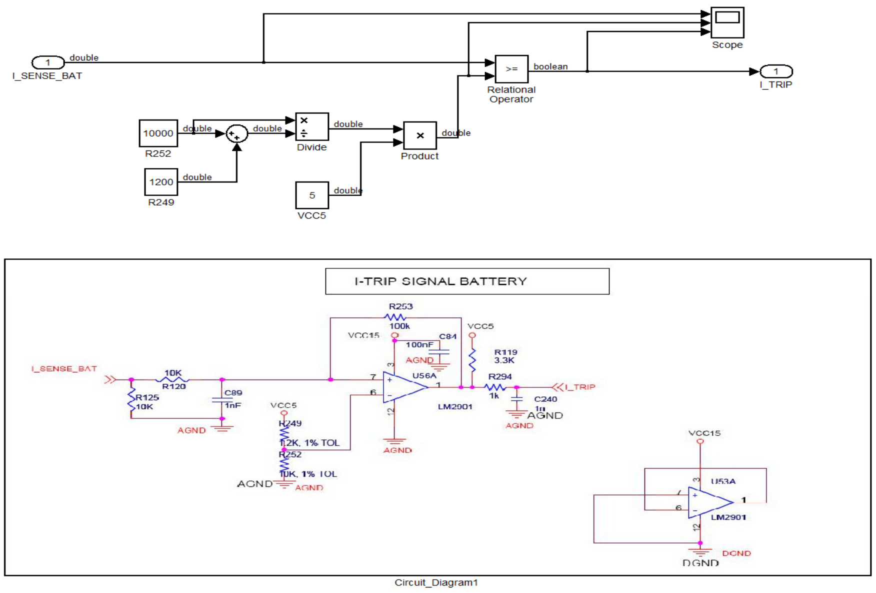


Figure 35 : I Trip Signal Battery circuit model

### 2.3.12 Jumper Circuit Model

Jumper on the PCB which route anyone of the signal exclusively to uC pin.

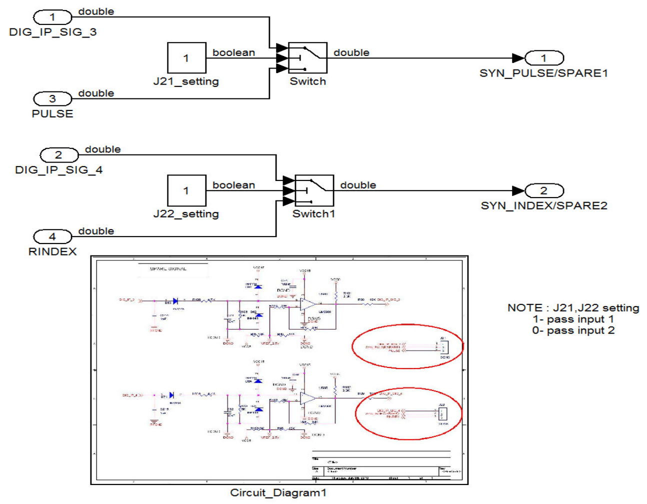


Figure 36 : Jumper Circuit Model

### 2.3.13 MOSFET driver H-Bridge circuit Model

Two FAN73832 are used to driver two leg of MOSFET in complimentary with dead-time, and with a shutdown pin controlled through uC and trip circuit as in figure 21. Shutdown logic model are shown in the subsequent figure 26 and 27.

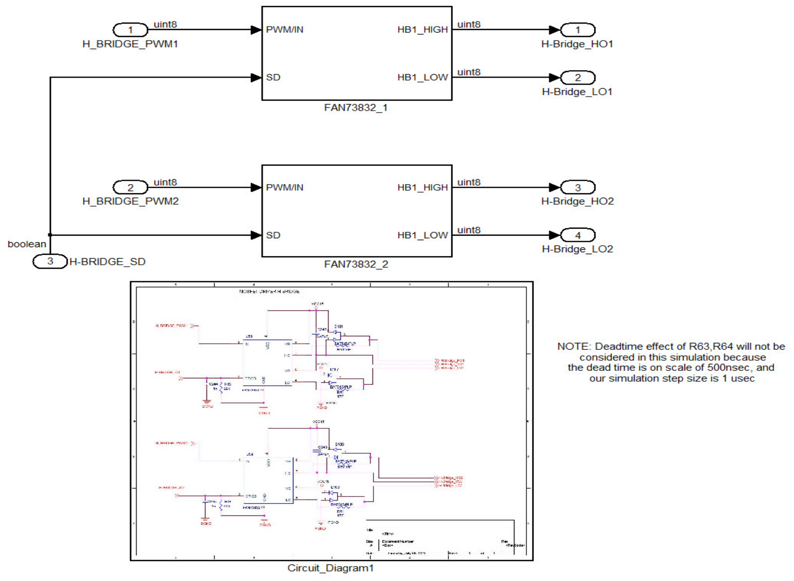


Figure 37 : MOSFET driver H-Bridge circuit Model

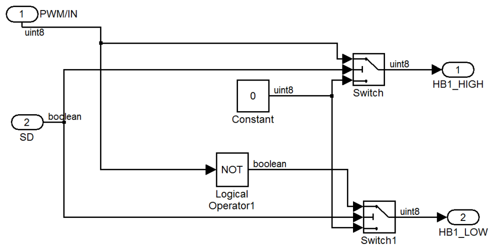


Figure 38 : FAN73832_1 shutdown logic model

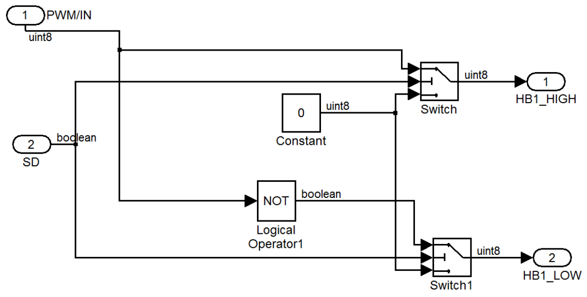


Figure 39 : FAN73832_2 shutdown logic model

### 2.3.14 Voltage Sense Circuit Model

Power board battery voltage signal are wire directly to the PCB, which are higher voltage (say 48V) and are needed to be divided down to uC sensible voltage level using potential divider circuit as shown below

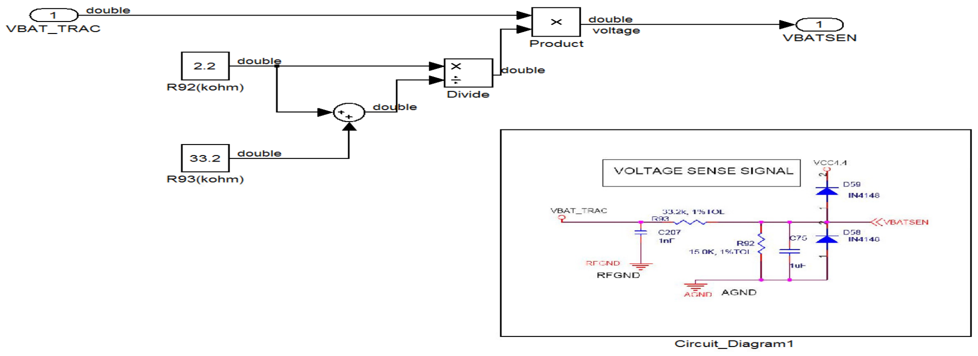


Figure 40 : Voltage Sense Circuit Model

## 2.3.15 Three Phase Bridge driver circuit Model

IR2132 the three phase driver chips are model on its functional excluding the bootstrap capacitor and diode. Fault latch and fault shutdown logic are modeled as shown below in the consecutive figures below.

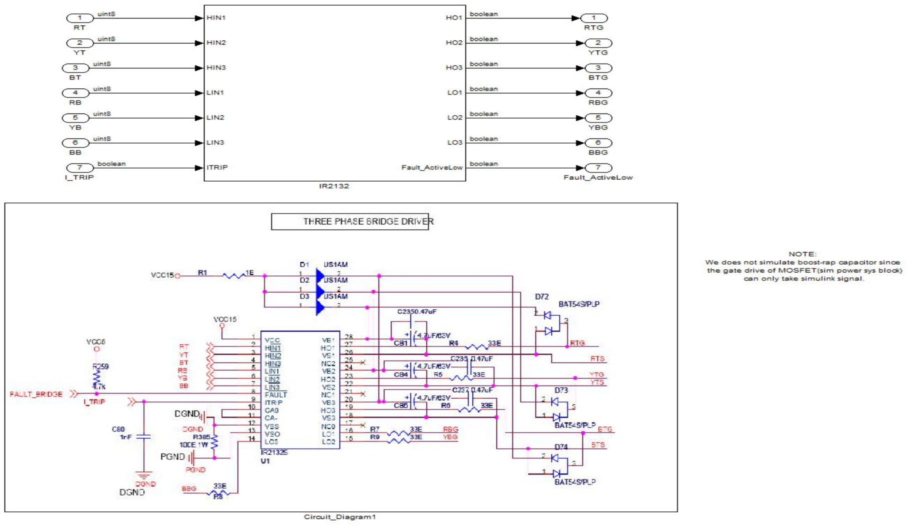

IR2132
HIN1
HIN2
HIN3
LIN1
LIN2
LIN3
ITRIP
HO1
HO2
HO3
LO1
LO2
LO3
Fault_ActiveLow
RT
YT
BT
RB
YB
BB
I_TRIP
RTG
YTG
BTG
RBG
YBG
BBG
THREE PHASE BRIDGE DRIVER
NOTE:
We does not simulate boost-rap capacitor since the gate drive of MOSFET(sim power sys block) can only take simulink signal.
FAULT_BRIDGE
Circuit_Diagram1

Figure 41 : Three Phase Bridge driver circuit Model

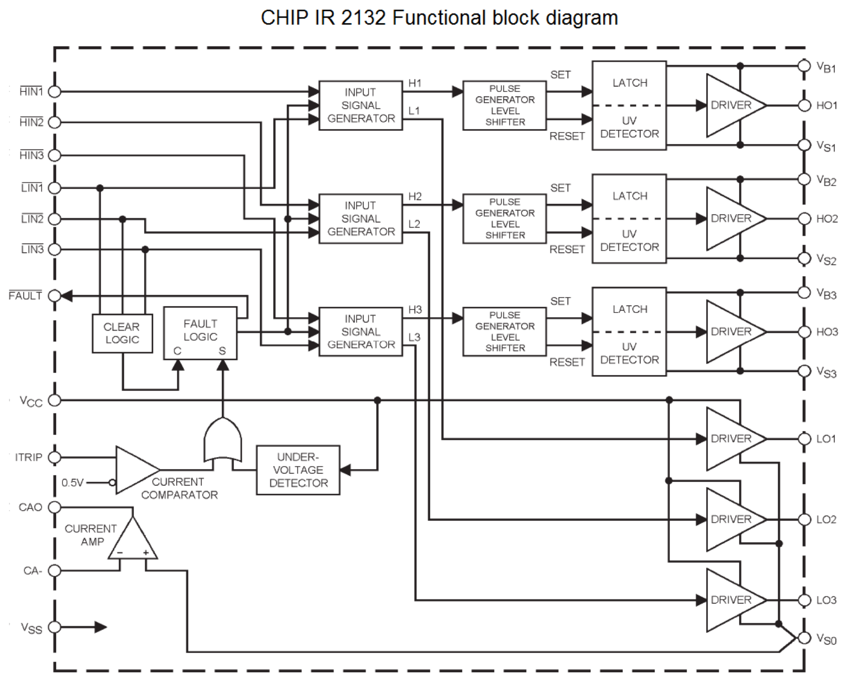


Figure 42 : IR3132 functional diagram

Fault Latch are reset by giving all 3 low input together for sometime as modeled below figure, state-chart is used to model fault state latch.

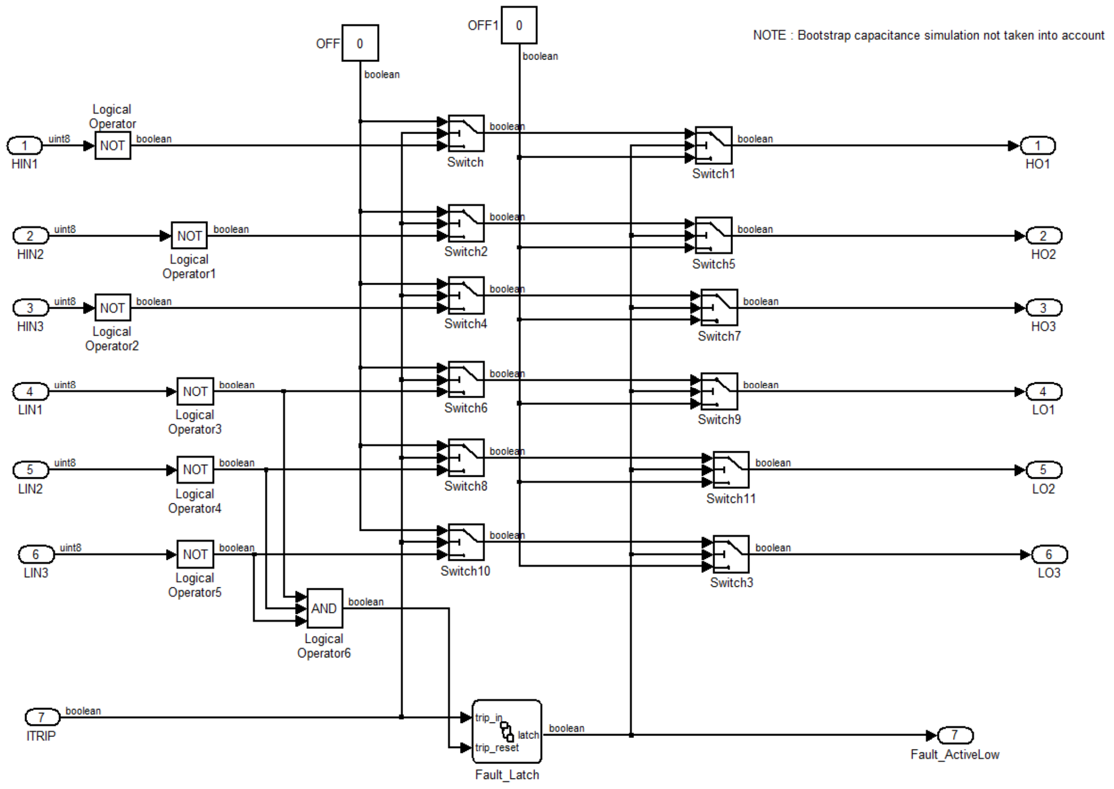


Figure 43 : IR2132 functional model

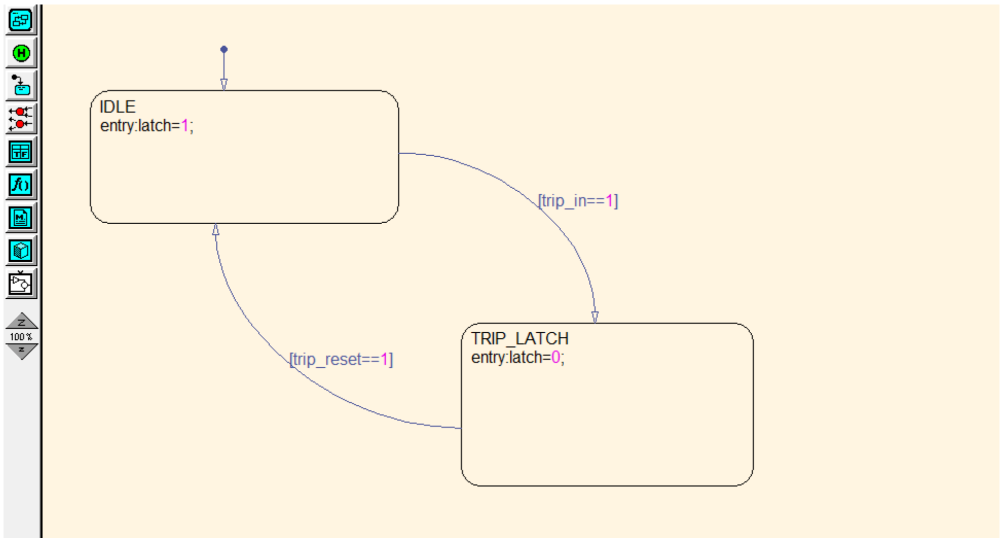


Figure 44 : IR2132 Latching model

## 2.3.16 MCU 18F4431 Chips Model

This subsystem consists of the CPU and the peripheral digital electronics inside the physical uC chips, as shown in the figure below. CPU functionalities are model as software which acts as the main control software from where uC C-code will be generated. Each of the peripheral modules is also modeled on a separate subsystem as shown in figure 33.

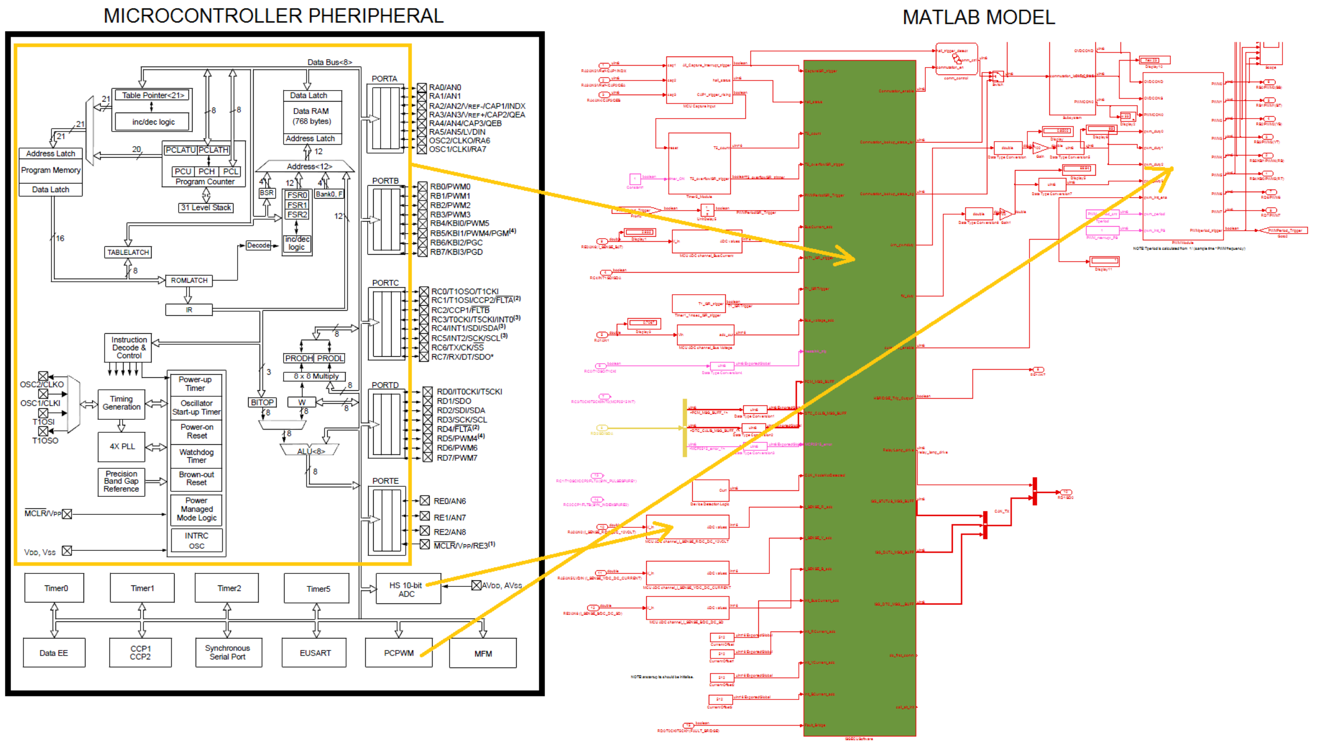


Figure 45 : Microcontroller model scheme

### 2.3.16.1 MCU ADC Block modeling

18F4431 chip ADC channel are group into 4 groups- A, B, C and D. Two channels at a time can be sampled together, and conversion can be done consecutively and can be routed to a 4 FIFO buffer as shown below. In this ADC modeling in MATLAB, we neglect this complexity of FIFO and channel multiplexing. So, direct conversion logic is modeled for each ADC channel.

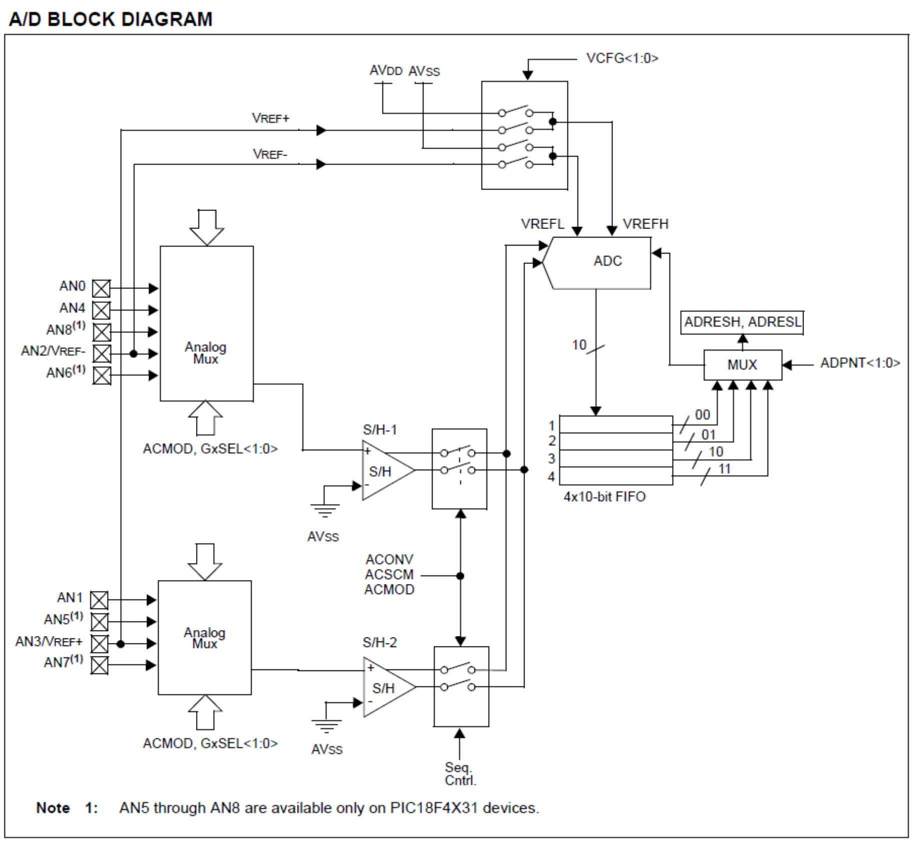


Figure 46 : MCU ADC block diagram

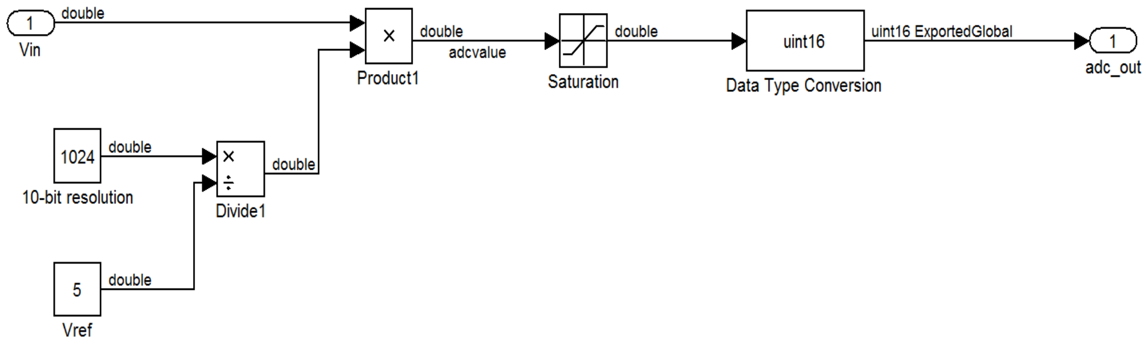


Figure 47 : ADC channel conversion and sampling model

ADC modeling is done for each 5 channels used which are used for sensing analog signal of currents and voltage signals. Table below highlights the ADC channels used with details, and modeling is same for each channel as given in above figure 35.

Table 4 : ADC channel assignment

| ADC Channel | ADC Grouping | Port Group | 18F Pin No | Signal symbols | Signal Descriptions |
|---|---|---|---|---|---|
| AN0 | A | RA0 | 19 | I_SENSE_R | R-phase current |
| AN5 | B | RA5 | 24 | I_SENSE_Y | Y-phase current |
| AN8 | A | RE2 | 27 | I_SENSE_B | B-phase current |
| AN7 | D | RE1 | 26 | VBAT_SEN | Bus Voltage |
| AN6 | C | RE0 | 25 | I_SENSE | Bus Current |

### 2.3.16.2 MCU Input Capture Block Modeling

Input channel (IC1) includes a special event trigger that can be configured for use in Velocity Measurement mode. Its block diagram is shown in Figure below. IC2 and IC3 are similar, but lack the special event trigger features or additional velocity-measurement logic. A representative block diagram is shown in Figure below. Please note that the time base is Timer5.

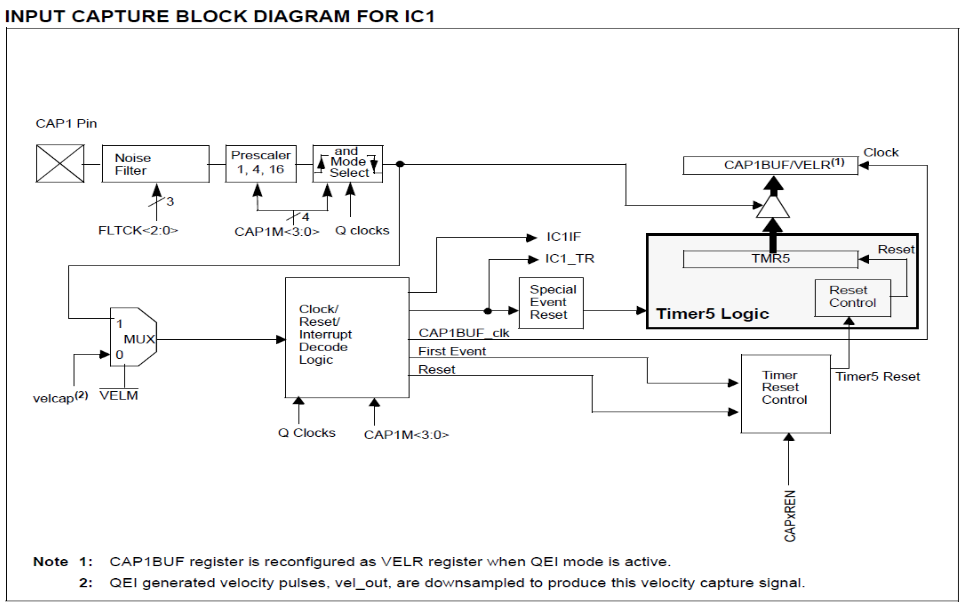


Figure 48 : MCU Input Capture block for IC1

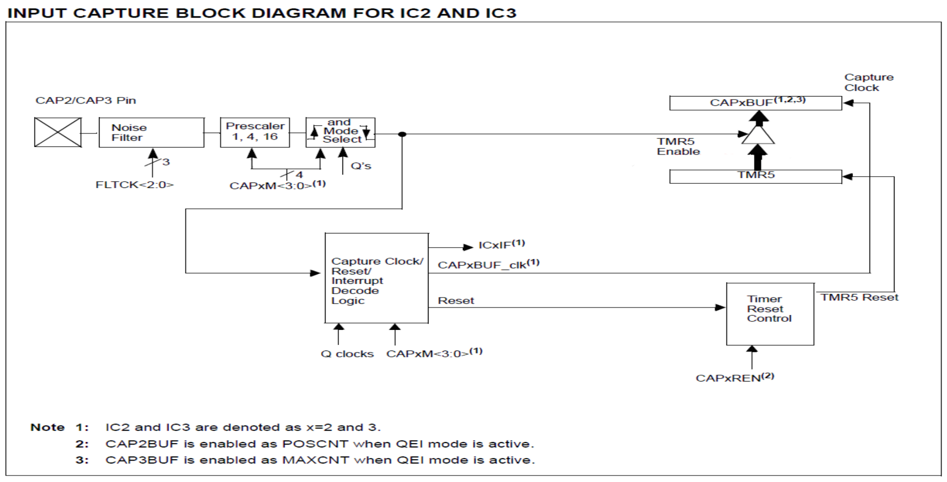

INPUT CAPTURE BLOCK DIAGRAM FOR IC2 AND IC3
CAP2/CAP3 Pin
Noise Filter
Prescaler 1, 4, 16
and Mode Select
FLTCK<2:0>
CAPxM<3:0>(1)
Q's
Capture Clock
CAPxBUF(1,2,3)
TMR5 Enable
TMR5
Capture Clock/ Reset/ Interrupt Decode Logic
ICxIF(1)
CAPxBUF_clk(1)
Reset
Timer Reset Control
TMR5 Reset
Q clocks
CAPxM<3:0>(1)
CAPxREN(2)
Note 1: IC2 and IC3 are denoted as x=2 and 3.
2: CAP2BUF is enabled as POSCNT when QEI mode is active.
3: CAP3BUF is enabled as MAXCNT when QEI mode is active.

Figure 49: MCU Input Capture block for IC2 and IC3

Here, capture triggering module and line-to-data are model, all other complexity are not model, but timer 5 special event reset is model in figure 44 using a state-chart, details can be found on Timer 5 module modeling in section 2.3.15.4.

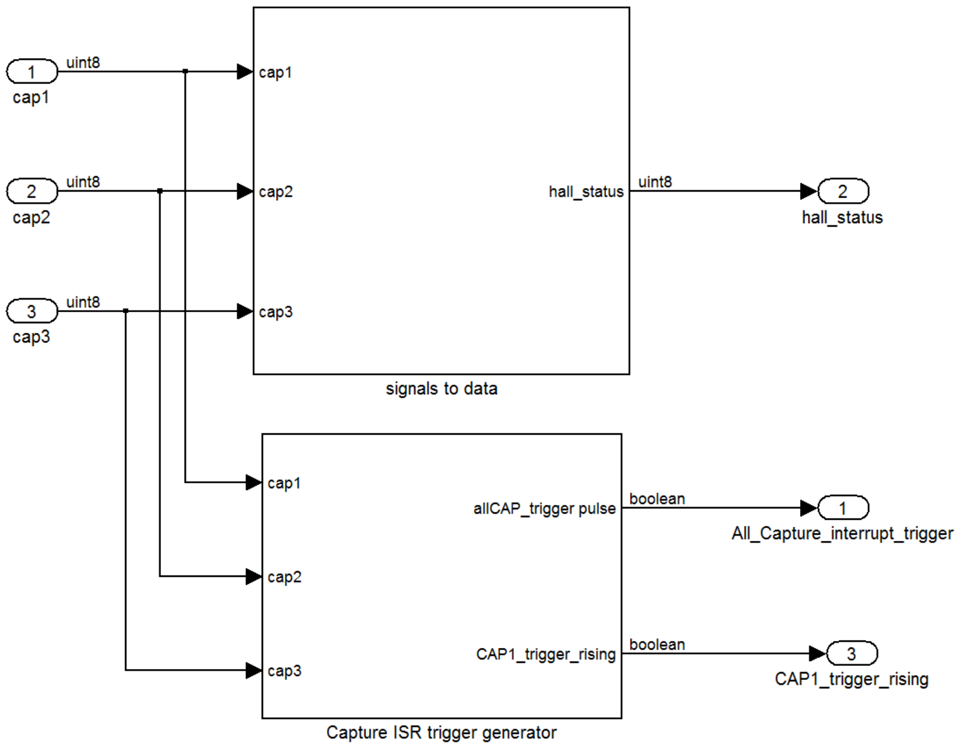


Figure 50: Input capture model

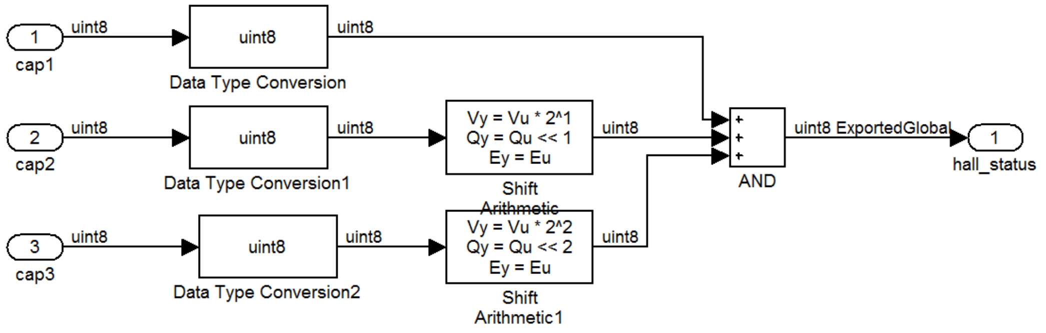


Figure 51: Signal line to data converter model

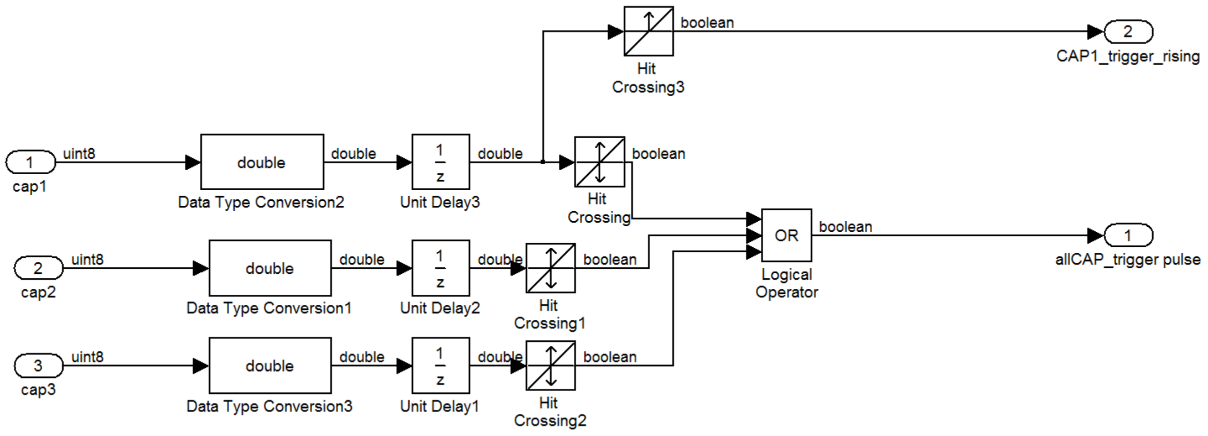


Figure 52: Capture trigger generator model

### 2.3.16.3 TIMER0 timer Module Modeling (for ISR 1 msec Interrupt)

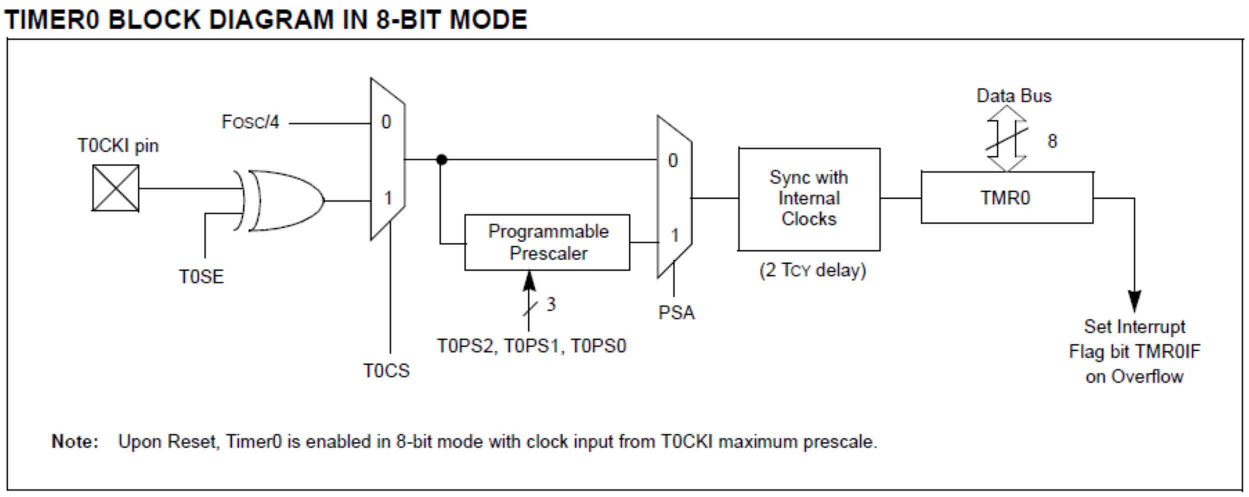
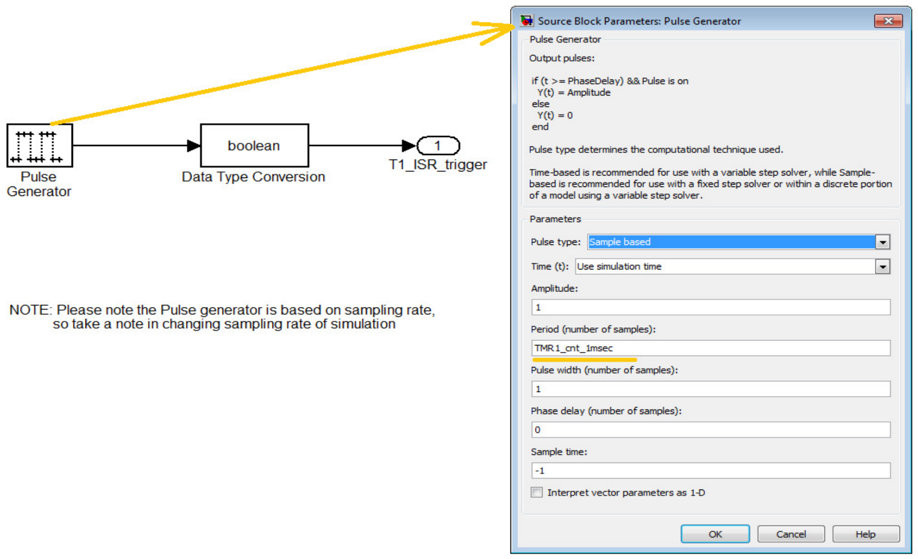

TIMER0 BLOCK DIAGRAM IN 8-BIT MODE
T0CKI pin
FOSC/4
0
1
T0SE
T0CS
Programmable Prescaler
3
T0PS2, T0PS1, T0PS0
0
1
PSA
Sync with Internal Clocks
(2 TCY delay)
Data Bus
8
TMR0
Set Interrupt Flag bit TMR0IF on Overflow
Note: Upon Reset, Timer0 is enabled in 8-bit mode with clock input from T0CKI maximum prescale.

Figure 53: MCU timer 0 block diagram

Timer 0 is modeled as a pulse generator which gives rising edge of pulse at specific interval of time, and is used to trigger the timer0 ISR subsystem inside the software foreground subsystem.


Pulse Generator
boolean
Data Type Conversion
1
T1_ISR_trigger
NOTE: Please note the Pulse generator is based on sampling rate, so take a note in changing sampling rate of simulation
Source Block Parameters: Pulse Generator
OK
Cancel
Help

Pulse Generator

Output pulses:

```
if (t >= PhaseDelay) && Pulse is on
  Y(t) = Amplitude
else
  Y(t) = 0
end
```

Pulse type determines the computational technique used.

Time-based is recommended for use with a variable step solver, while Sample-based is recommended for use with a fixed step solver or within a discrete portion of a model using a variable step solver.

Parameters

Pulse type: Sample based

Time (t): Use simulation time

Amplitude: 1

Period (number of samples): TMR1_cnt_1msec

Pulse width (number of samples): 1

Phase delay (number of samples): 0

Sample time: -1

☐ Interpret vector parameters as 1-D

Figure 54: Timer 0 interrupt generation model

## 2.3.16.4 TIMER5 MODULE MODELING

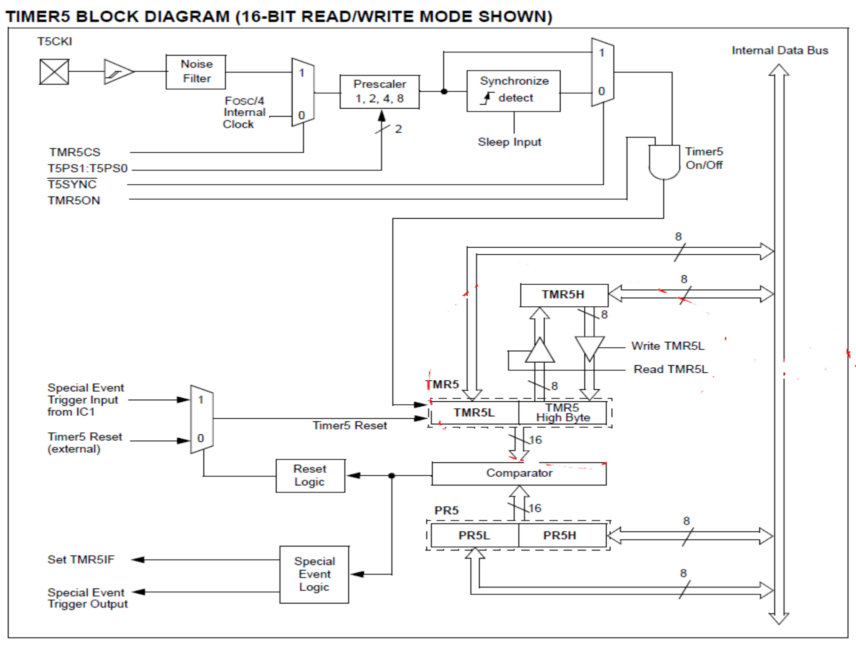


Figure 55 : MCU Timer 5 Block diagram

Timer 5 module block diagram as is inside the microcontroller is shown above. Timer 5 is used for hall sensor signal change time counter, which is used for speed calculation. Overflow and resetting of timer are taken care inside the software foreground subsystem, resetting is done on every instant of hall change, and overflow is taken care using timer5 overflow interrupt.

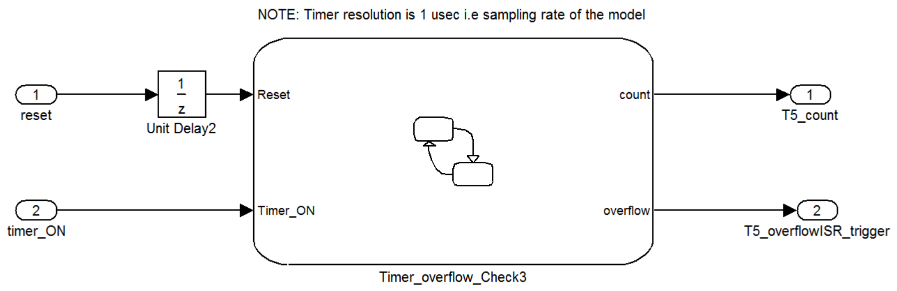


Figure 56 : Timer5 overflow and count model

This part of the model is not consider in the software since it works on the digital peripheral of the uC digital electronics, but are configure through special function register setting, during the initialization phase of uC peripheral registers. This subsystem is sampled at base model sampling rate i.e. 1usec, which is the resolution of the counter.

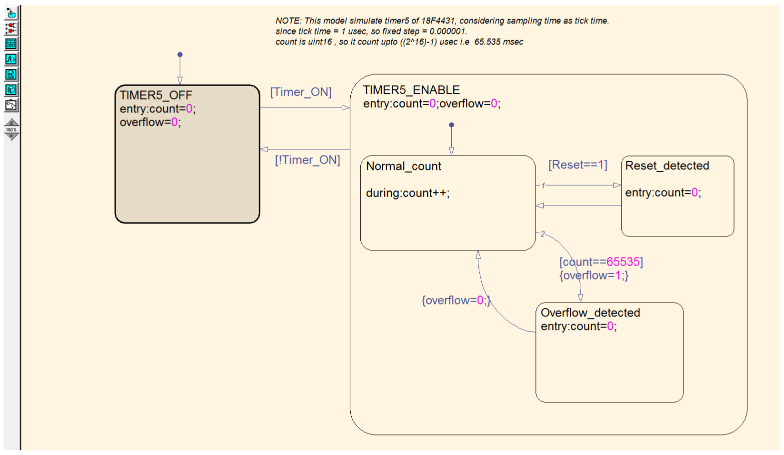

NOTE: This model simulate timer5 of 18F4431, considering sampling time as tick time.
since tick time = 1 usec, so fixed step = 0.000001.
count is uint16 , so it count upto ((2^16)-1) usec i.e 65.535 msec
TIMER5_OFF
entry:count=0;
overflow=0;
[Timer_ON]
[!Timer_ON]
TIMER5_ENABLE
entry:count=0;overflow=0;
Normal_count
during:count++;
[Reset==1]
Reset_detected
entry:count=0;
[count==65535]
{overflow=1;}
{overflow=0;}
Overflow_detected
entry:count=0;

Figure 57 : Timer 5 overflow and count state-chart

## 2.3.16.5 MCU Power Control PWM Module Modeling

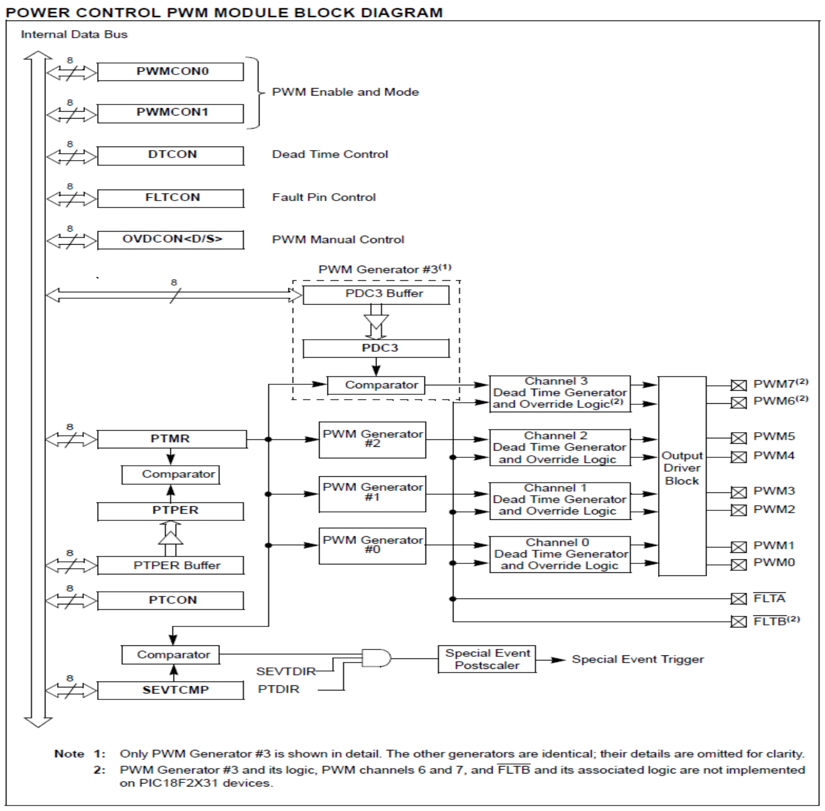

POWER CONTROL PWM MODULE BLOCK DIAGRAM
Internal Data Bus
PWMCON0
PWMCON1
PWM Enable and Mode
DTCON
Dead Time Control
FLTCON
Fault Pin Control
OVDCON<D/S>
PWM Manual Control
PWM Generator #3(1)
PDC3 Buffer
PDC3
Comparator
Channel 3 Dead Time Generator and Override Logic(2)
PTMR
PWM Generator #2
Channel 2 Dead Time Generator and Override Logic
Comparator
PTPER
PWM Generator #1
Channel 1 Dead Time Generator and Override Logic
PTPER Buffer
PWM Generator #0
Channel 0 Dead Time Generator and Override Logic
PTCON
Output Driver Block
PWM7(2)
PWM6(2)
PWM5
PWM4
PWM3
PWM2
PWM1
PWM0
FLTA
FLTB(2)
Comparator
SEVTDIR
PTDIR
SEVTCMP
Special Event Postscaler
Special Event Trigger
Note 1: Only PWM Generator #3 is shown in detail. The other generators are identical; their details are omitted for clarity.
2: PWM Generator #3 and its logic, PWM channels 6 and 7, and FLTB and its associated logic are not implemented on PIC18F2X31 devices.

Figure 58 : MCU PWM module block diagram

Power PWM module block diagram of 18F4431 is shown above, which consists of configuration and functional register, along with the PWM generator, dead-time generator and the output driver blocks. The special event trigger and dead-time generator are not model in this model. Complimentary logic and output override logic are digitally wired inside output driver block which are control through PWMCON0, OVDCOND and OVDCONS register. OVDCOND and OVDCONS registers are used to define the PWM override options. The OVDCOND register contains eight bits, POVD7:POVD0 that determine which PWM I/O pins will be overridden. The OVDCONS register contains eight bits, POUT7:POUT0, that determine the state of the PWM I/O pins when a particular output is overridden via the POVD bits. The POVD bits are active-low control bits. When the POVD bits are set, the corresponding POUT bit will have no effect on the PWM output. In other words, the pins corresponding to POVD bits that are set will have the duty PWM cycle set by the PDC registers. When one of the POVD bits is cleared, the output on the corresponding PWM I/O pin will be determined by the state of the POUT bit. When a POUT bit is set, the PWM pin will be driven to its active state. When the POUT bit is cleared, the PWM pin will be driven to its inactive state. The above mention logic is model as below.

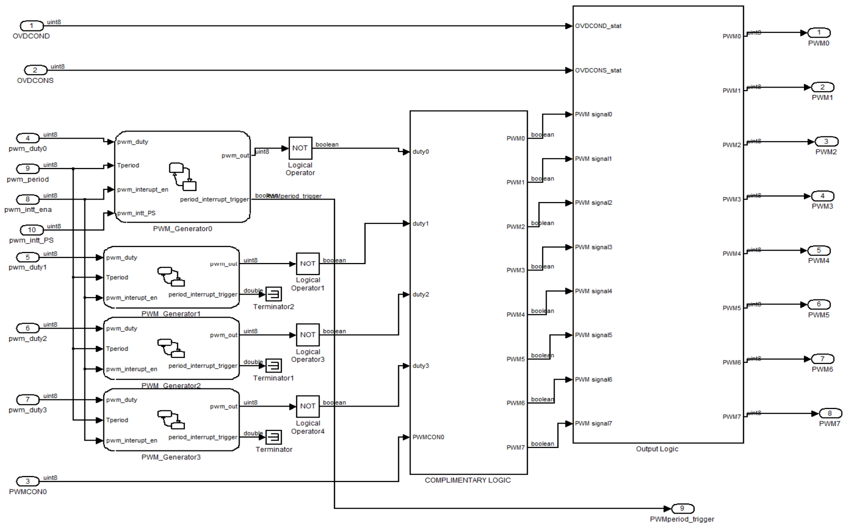


Figure 59 : PWM module model

The PWM generator block is model in state-chart as given with a base sampling rate of 1 usec. PWM period interrupt trigger pulse is also modeled, along with this PWM period could be set externally depending on the PWM frequency used for switching the MOSFET (say 10 KHz, 100usec).

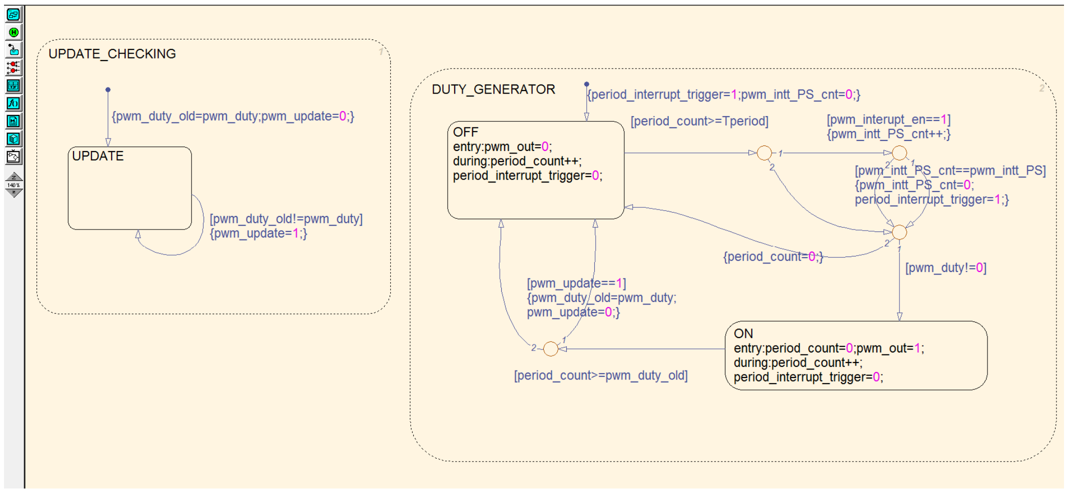


Figure 60 : PWM generator and Period interrupt triggering logic

The output override control block are model in consecutive figures 49 through 51 and are self explanatory. Complimentary logic is model as given below which are control by PWMCON0 (bit3, bit2, bit1 and bit0) register. If else signal routing block call switch in Simulink blockset is used to do that operation as shown.

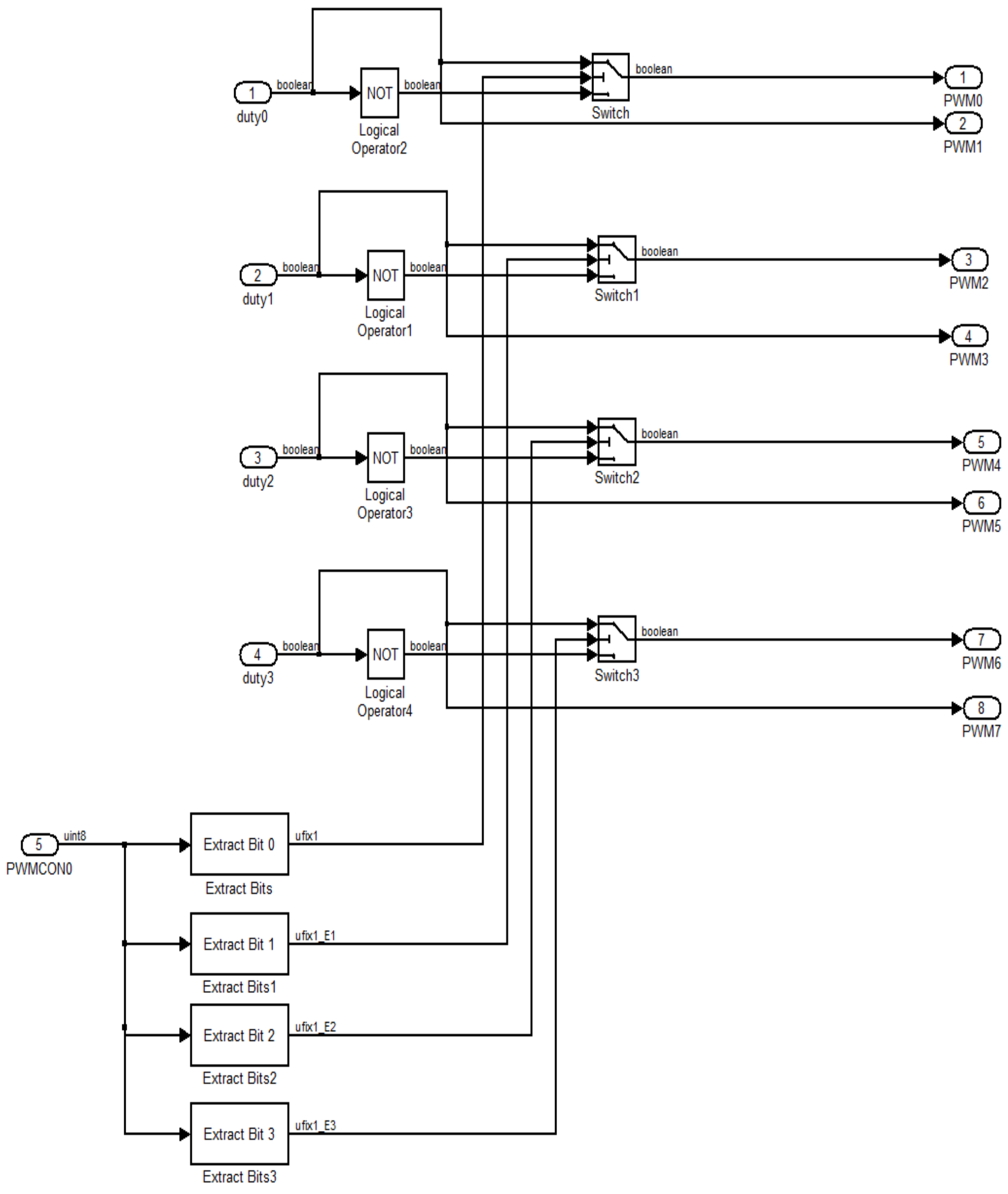


Figure 61 : Complementary logic block

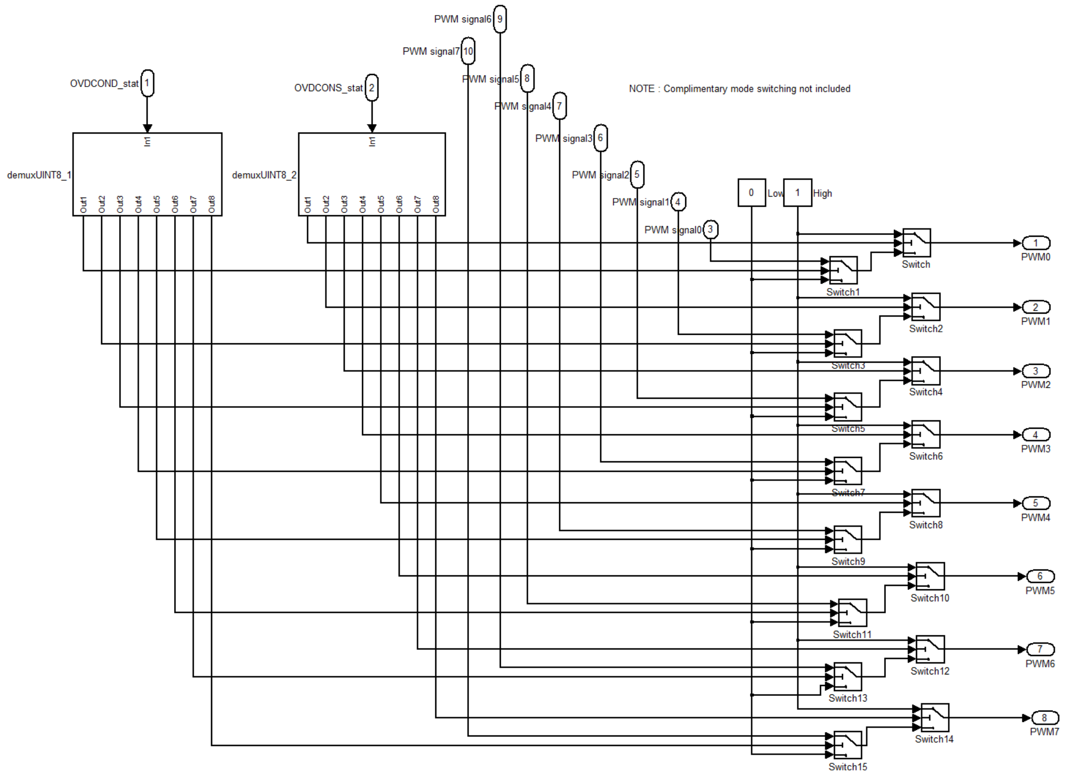


Figure 62 : Output Logic block

OVDCOND, OVDCONS are 8 bit register, where each bit field represents the override setting for each channel of PWM signal. So, demux block is model as given below to latch out data to signal lines for controlling PWM output override in figure 50.

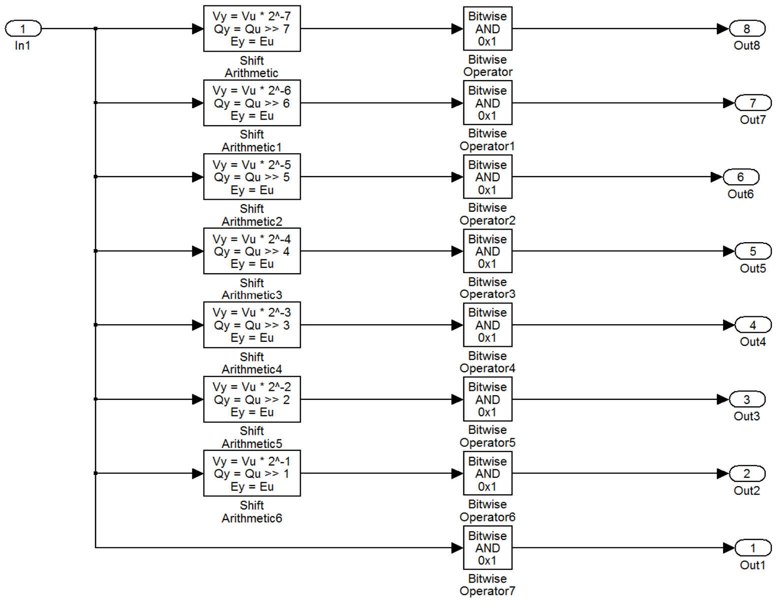


Figure 63 : De-multiplex Byte block

### 2.3.16.6 SOFTWARE MODEL

This subsystem is from where we generated the code which goes directly to the microcontroller. C code to be dump to microcontroller are auto-coded using real-time workshop from this level of subsystem. It consists of foreground and background subsystems which are communicated by global storage blocksets of Simulink, and some of the signal are loop back through unit delay block. Foreground are interrupt which are executed on timer or external event, background at task are function which are executed on the forever while loop inside the main function in C programming.

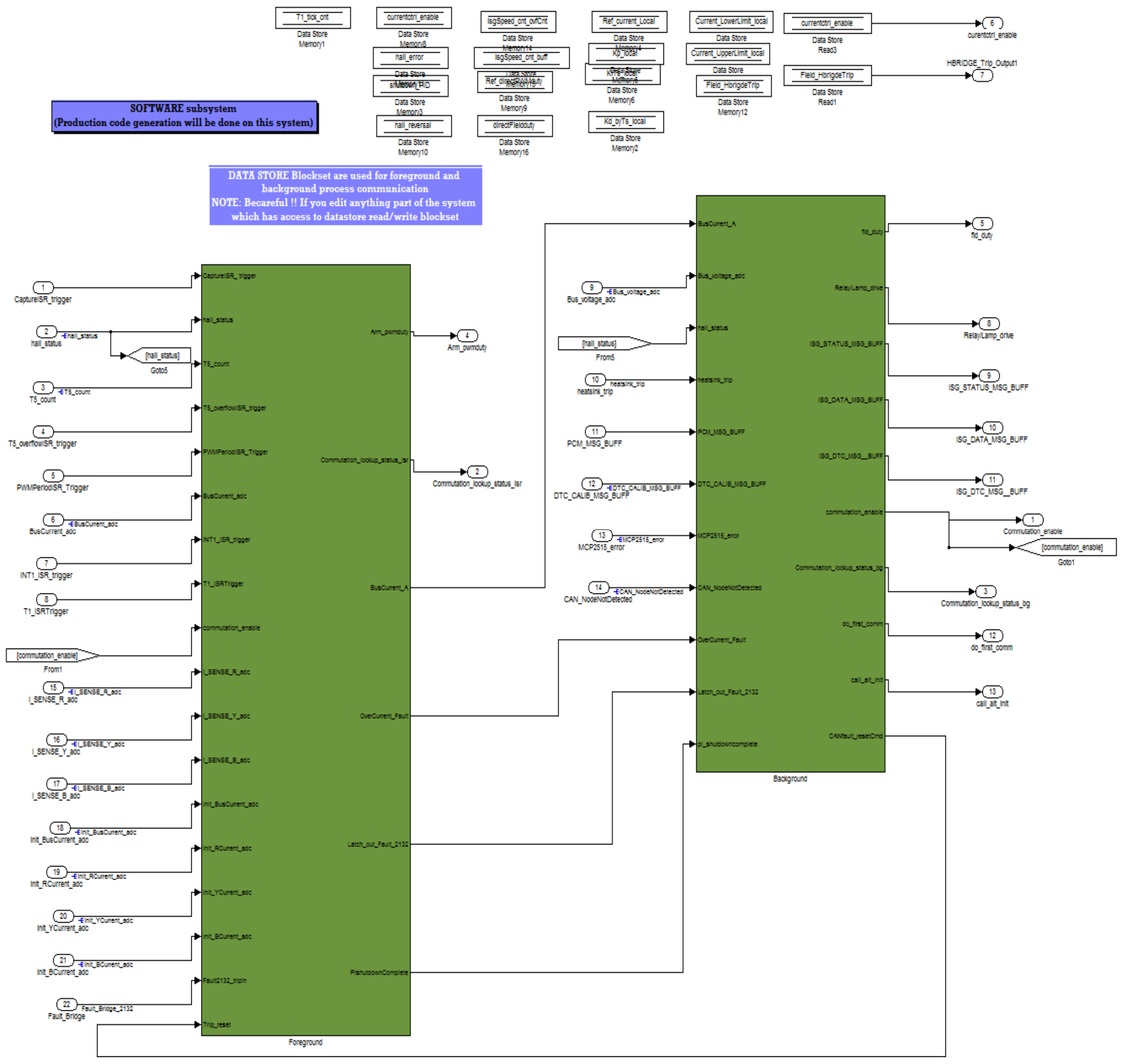

SOFTWARE subsystem
(Production code generation will be done on this system)
DATA STORE Blockset are used for foreground and background process communication
NOTE: Becareful !! If you edit anything part of the system which has access to datastore read/write blockset
Foreground
Background

Figure 64 : Software model (Background and Foreground Subsystem)

### 2.3.16.6.1 *FOREGROUND SUBSYSTEM MODEL*

Foreground subsystem consist of event based and timer based trigger subsystem. Event based trigger subsystem comprises of Hall event capture, PWM period event capture interrupt, and Trip event interrupt. Timer based trigger subsystem comprises of Timer0 periodic interrupt and Timer5 overflow interrupt. All subsystems in the foreground subsystem are trigger based subsystems.

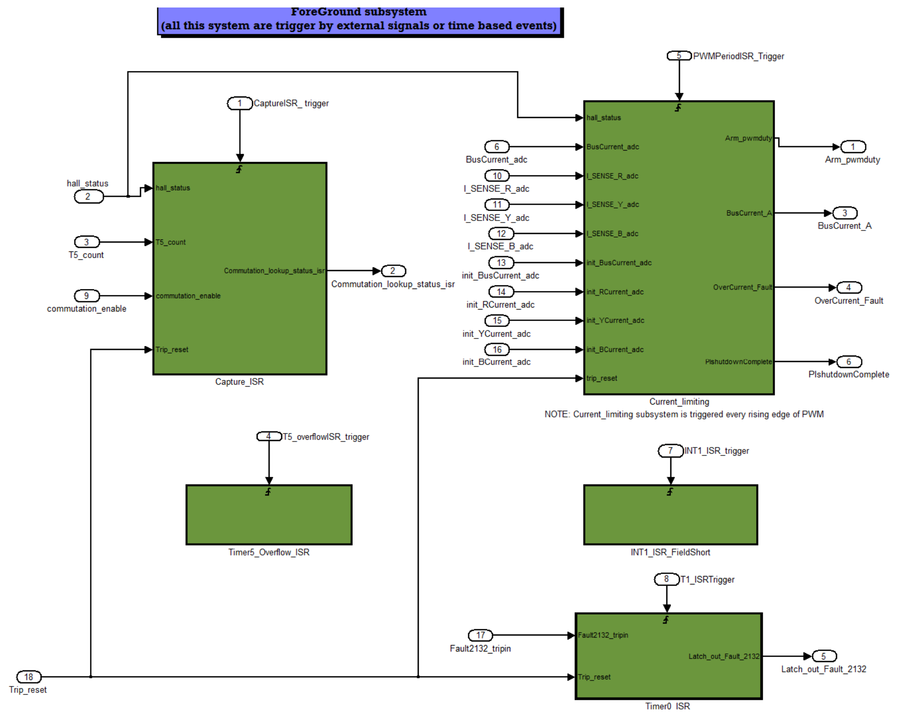


Figure 65 : Foreground subsystem model

#### 2.3.16.6.1.1 Capture ISR model

This subsystem is trigger on every hall change on any of the hall sensor signals generated through the peripheral subsystem of the uC. The source of this subsystem input could be referred in section MCU input capture block modeling, and MCU Timer 5 module. Under this subsystem there are four subsystems-Hall checks, Latch logic, commutation and update speed count.

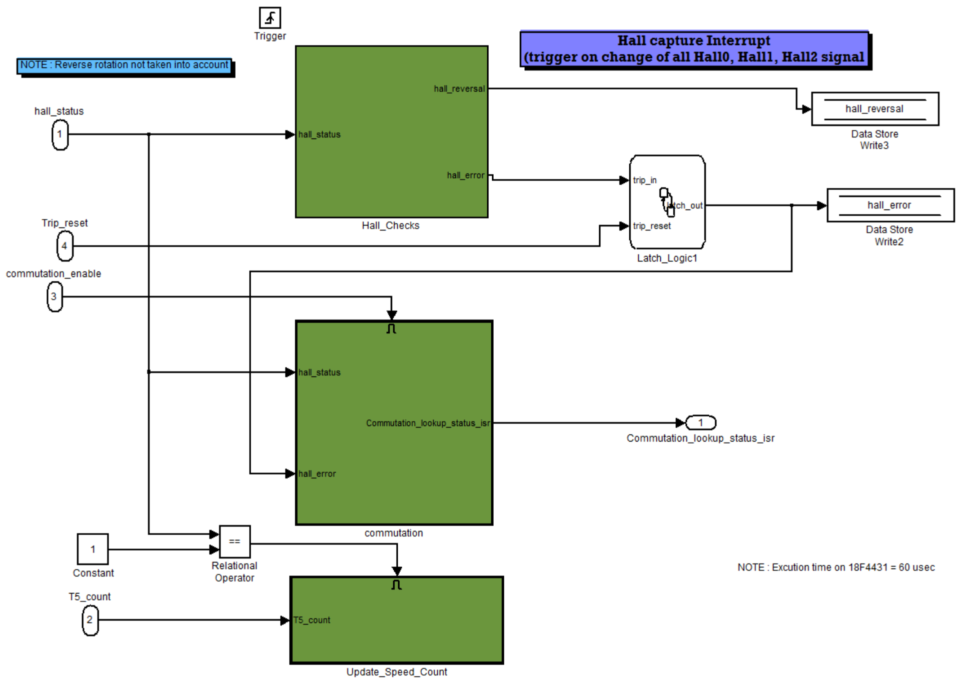


Figure 66: Capture ISR model

### 2.3.16.6.1.1.1 Hall Check model

Under this system the hall sequence is monitor all the time, whenever there is wrong sequence or reversal, output signal is made high accordingly.

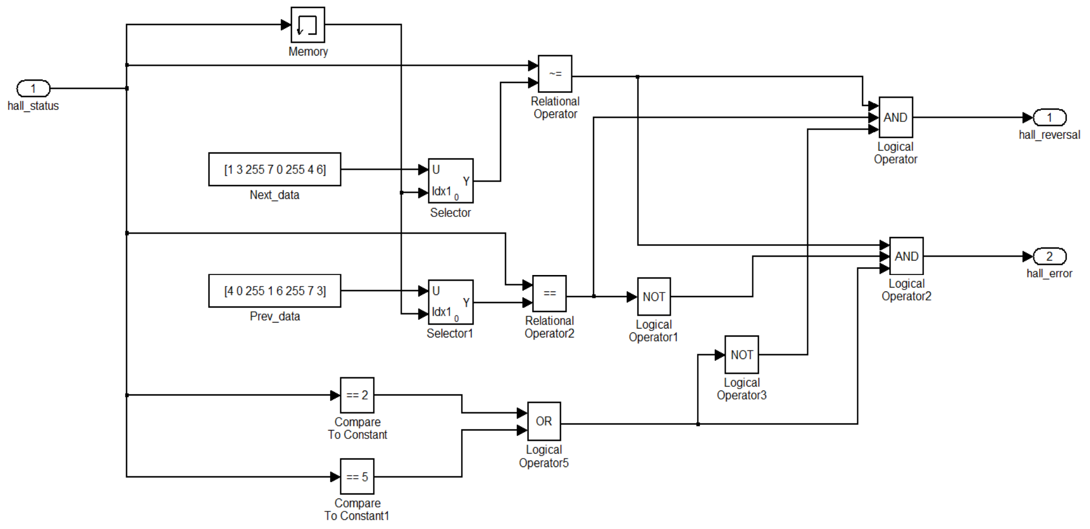
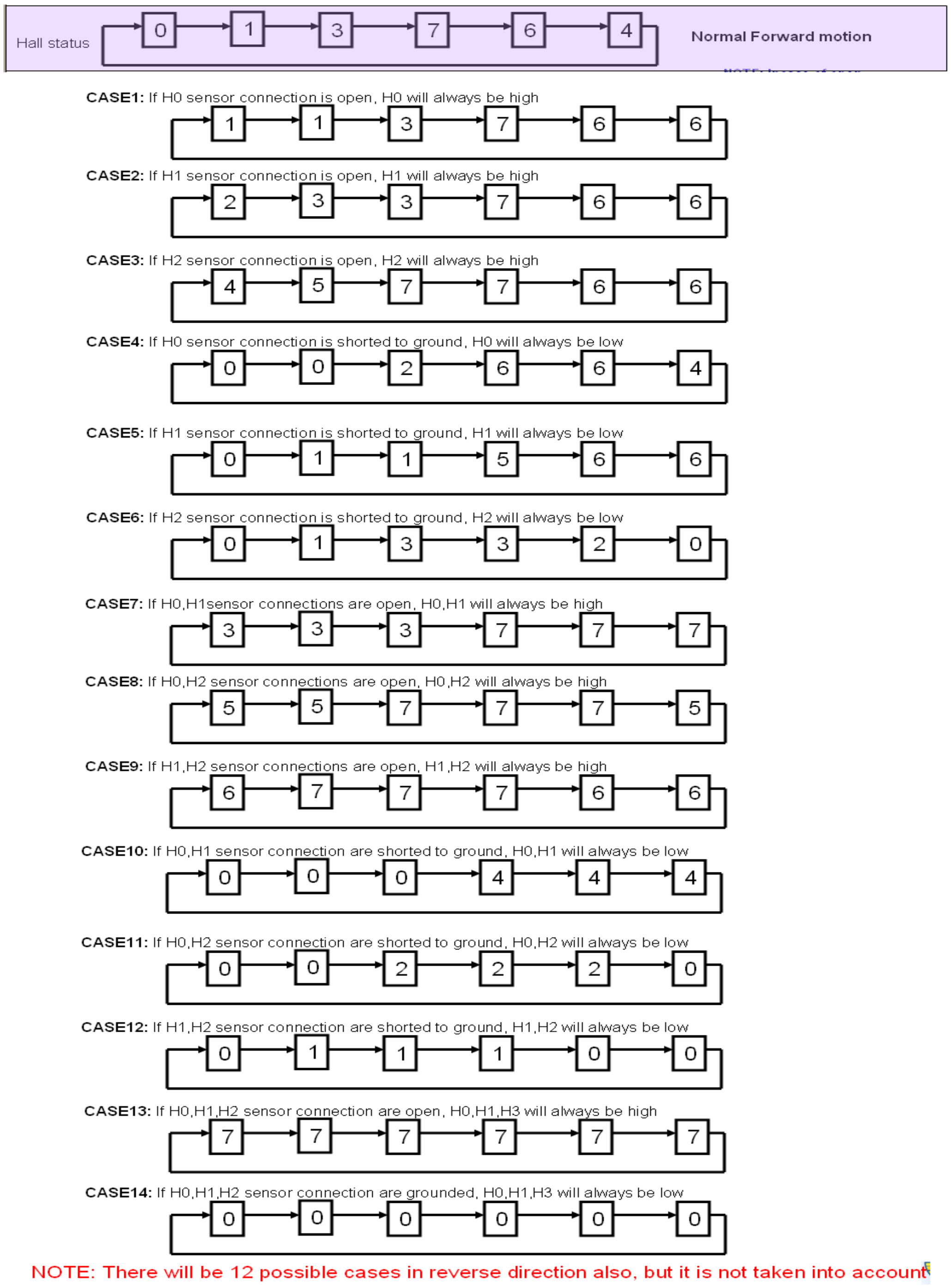


Figure 67 : Hall error check model


Hall status
0 1 3 7 6 4
Normal Forward motion
CASE1: If H0 sensor connection is open, H0 will always be high
1 1 3 7 6 6
CASE2: If H1 sensor connection is open, H1 will always be high
2 3 3 7 6 6
CASE3: If H2 sensor connection is open, H2 will always be high
4 5 7 7 6 6
CASE4: If H0 sensor connection is shorted to ground, H0 will always be low
0 0 2 6 6 4
CASE5: If H1 sensor connection is shorted to ground, H1 will always be low
0 1 1 5 6 6
CASE6: If H2 sensor connection is shorted to ground, H2 will always be low
0 1 3 3 2 0
CASE7: If H0,H1sensor connections are open, H0,H1 will always be high
3 3 3 7 7 7
CASE8: If H0,H2 sensor connections are open, H0,H2 will always be high
5 5 7 7 7 5
CASE9: If H1,H2 sensor connections are open, H1,H2 will always be high
6 7 7 7 6 6
CASE10: If H0,H1 sensor connection are shorted to ground, H0,H1 will always be low
0 0 0 4 4 4
CASE11: If H0,H2 sensor connection are shorted to ground, H0,H2 will always be low
0 0 2 2 2 0
CASE12: If H1,H2 sensor connection are shorted to ground, H1,H2 will always be low
0 1 1 1 0 0
CASE13: If H0,H1,H2 sensor connection are open, H0,H1,H3 will always be high
7 7 7 7 7 7
CASE14: If H0,H1,H2 sensor connection are grounded, H0,H1,H3 will always be low
0 0 0 0 0 0
NOTE: There will be 12 possible cases in reverse direction also, but it is not taken into account
Figure 68 : Hall error conditions analysis

Hall sensor signal are analyze on each of the probable condition when shorted to ground or shorted to supply as shown in figure above. Accordingly the hall check algorithm is modeled.

### 2.3.16.6.1.1.2 Latch logic model

This state-chart is same with the latch logic given before, it will hold the fault status until reset signal appear.

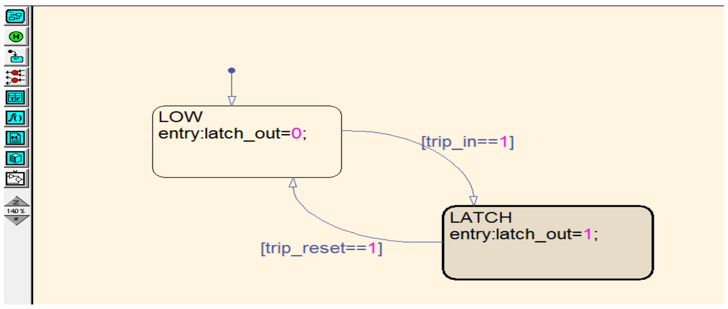
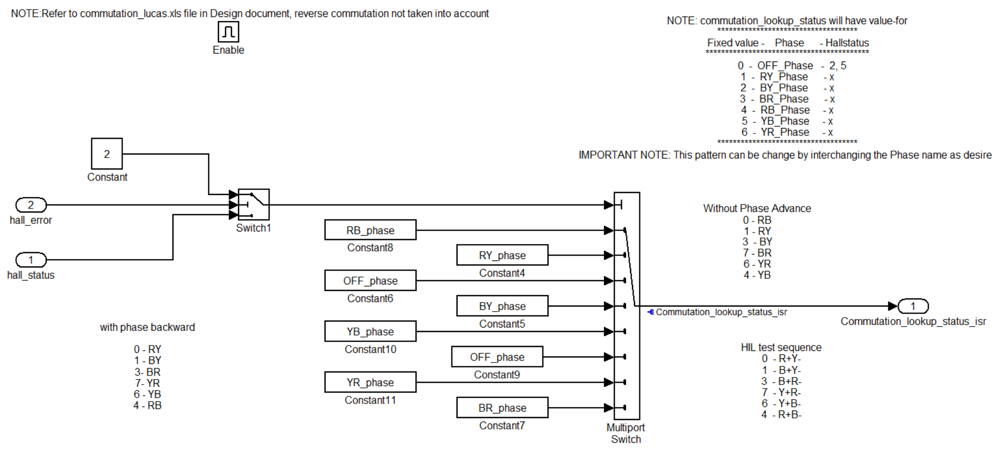


Figure 69 : Latch logic model

### 2.3.16.6.1.1.3 Commutation model

During motoring operation, the switching pattern should be change on every hall signal change. This subsystem switches the pattern of switch configuration depending on hall sensor signal. Occurrence of hall error will switch off all MOSFET. This subsystem is enabled only when commutation enable flag is raise from the background state control chart. Note that only pattern of MOSFET switching are only decided by PWM override which is controlled from this subsystem.



NOTE:Refer to commutation_lucas.xls file in Design document, reverse commutation not taken into account

NOTE: commutation_lookup_status will have value-for

| Fixed value - | Phase | - Hallstatus |
|---|---|---|
| 0 - | OFF_Phase | - 2, 5 |
| 1 - | RY_Phase | - x |
| 2 - | BY_Phase | - x |
| 3 - | BR_Phase | - x |
| 4 - | RB_Phase | - x |
| 5 - | YB_Phase | - x |
| 6 - | YR_Phase | - x |

IMPORTANT NOTE: This pattern can be change by interchanging the Phase name as desire

Without Phase Advance
0 - RB
1 - RY
3 - BY
7 - BR
6 - YR
4 - YB

with phase backward

0 - RY
1 - BY
3- BR
7- YR
6 - YB
4 - RB

HIL test sequence
0 - R+Y-
1 - B+Y-
3 - B+R-
7 - Y+R-
6 - Y+B-
4 - R+B-

Figure 70: Commutation model

### 2.3.16.6.1.1.4 Update speed count model

Timer 5 is used to measure the time between one electrical cycles of the machine, so this subsystem is enable on whenever hall status is 1 as shown in capture ISR model. Note that there will be 6 electrical cycle for mechanical one rotation for a 12 pole machine. T5 counter count a tick of 1 usec resolution in the microcontroller hardware. So, the accumulated count is read and noted to a global storage variable as shown below. But, there might be a case that, T5 16-bit counter overflow before hall status change to 1. In this case, a timer 5 overflow counter value will be updated on timer 5 overflow interrupt routine. 16 bit count value and number of overflow (8-bit) should be combined to 32-bit counter as shown below. So, speed count value of 32-bit is used in background to calculate speed in rpm.

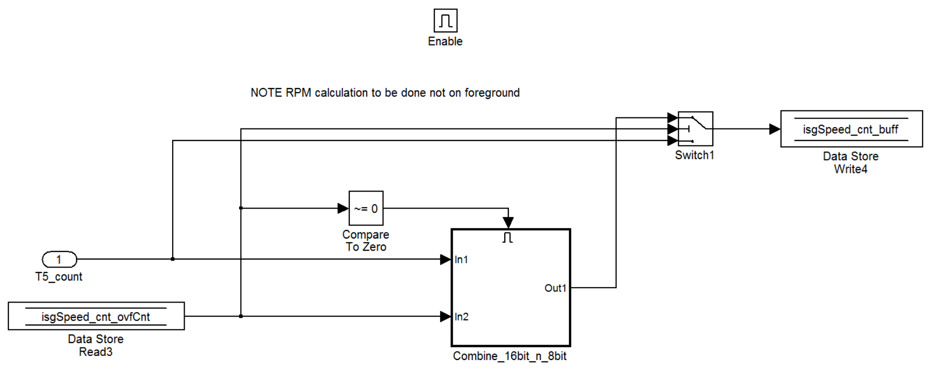


Figure 71: update speed count model

Whenever timer5 has been overflow or overflow count is not zero, the overflow counter will be reset.

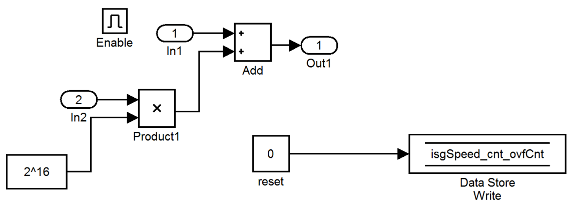


Figure 72: combine 16bit and 8bit to 32bit

### 2.3.16.6.1.2 Current Limiting Subsystem Model

This is the main control model which controlled the motor current using PID algorithm, and then controls torque. This subsystem will be trigger on every PWM period, so if the PWM period is 10/7.8 KHz, it will be trigger at instances of 100/128 usec. All phase current and bus current are sensed and converted to fixed-point engineering value (ampere) as shown in the figure below as a four small subsystems on left hand side. Other two subsystems are over current fault checking and the PID controller. Most subsystems in this current limiting subsystem are enabled at all time except that PID controller is enabled only when flagged by background state control. Any current faults will immediately shutdown the PID control which intern shut off the motor. Direct PWM duty control option is provided which directly control armature PWM duty over CAN interface for debugging and system checking during development.

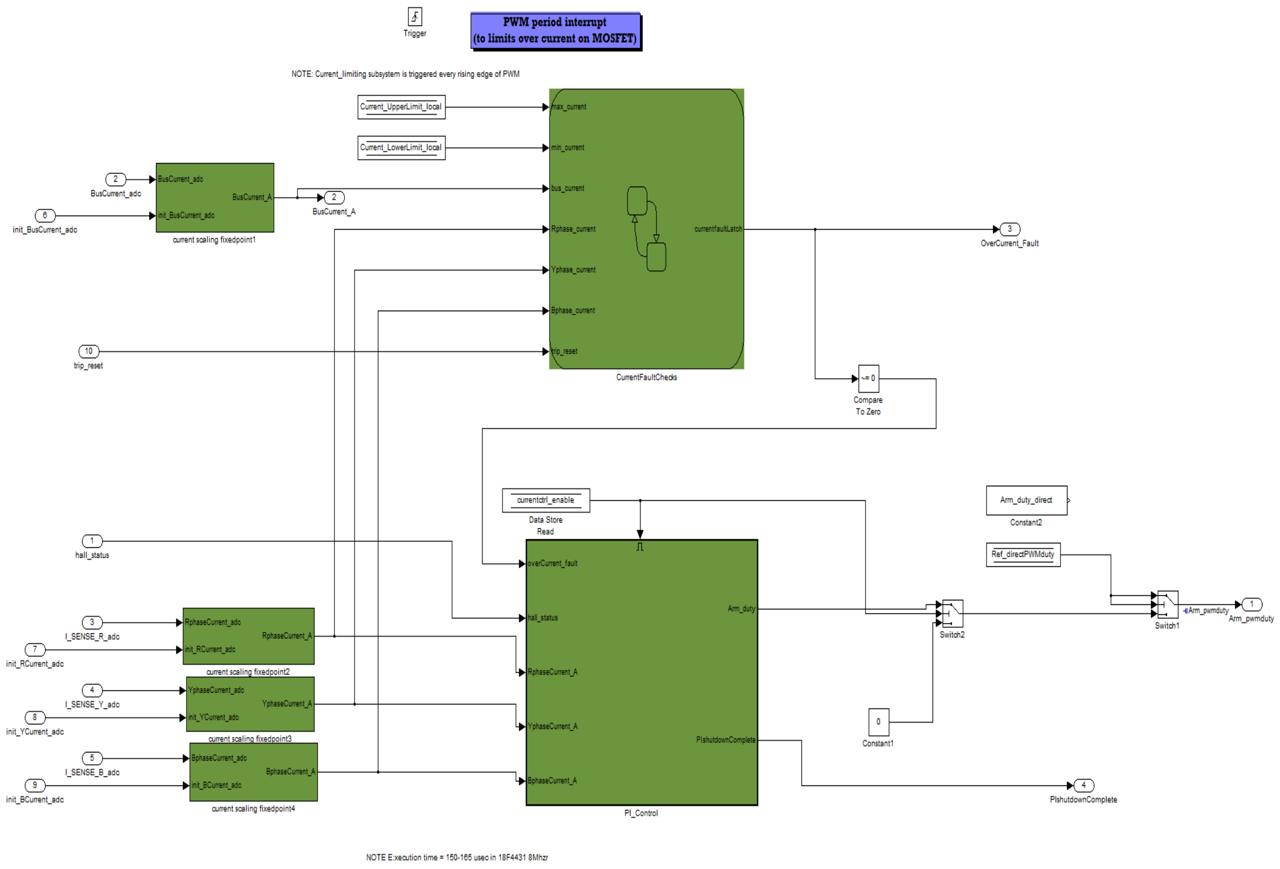


Figure 73: PWM interrupt or Current control model

Current limiting subsystem consists of current scaling block for each channel, fault checking and latch logic for each current channel, and the PID controller. Let us go through each of the block and study the functions of it one by one.

### 2.3.16.6.1.2.1 Current Scaling in Fixed point block model

ADC converted value need to be scaled to ampere value for the strategy, so fixed point scale value is chosen depending on the sensor specification, and are multiply with the ADC offset corrected value. ADC offset correction are done using initial start-up ADC value when the uC is first switch-on.

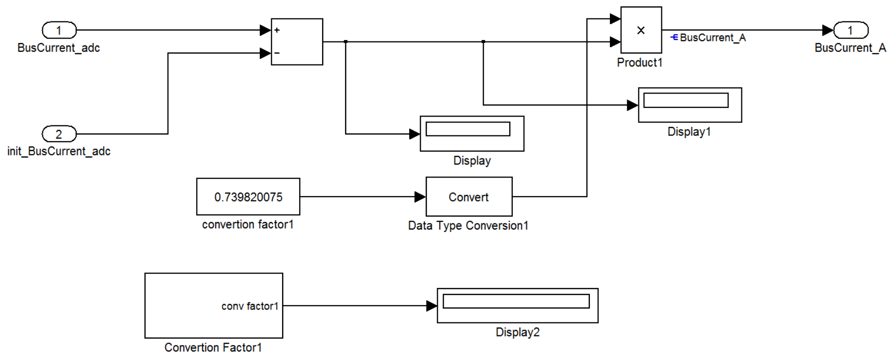
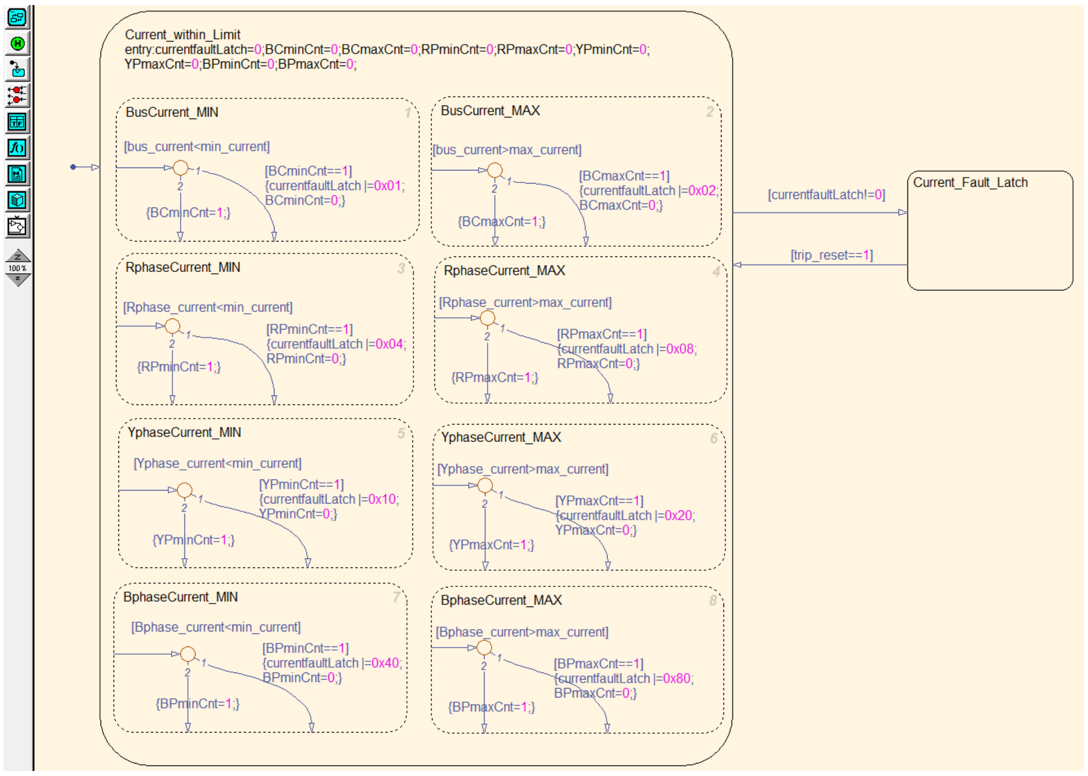


Figure 74: current scaling model

### 2.3.16.6.1.2.2 Current Fault checking and latching algorithm block model

Current value are check on both positive and negative extreme values, so if over current occurs on any current signal, a specific bit in the current fault variable is set. So, current fault can be uniquely identified. Each current limit checks are done on exclusive sub-state, which means that all checking are done paralleled within one sampling, and then if any fault occurs, it jumps to fault latch state and cannot jump back unless trip reset event occurs.


Current_within_Limit
entry:currentfaultLatch=0;BCminCnt=0;BCmaxCnt=0;RPminCnt=0;RPmaxCnt=0;YPminCnt=0;
YPmaxCnt=0;BPminCnt=0;BPmaxCnt=0;
BusCurrent_MIN
[bus_current<min_current]
[BCminCnt==1]
{currentfaultLatch |=0x01;
BCminCnt=0;}
{BCminCnt=1;}
BusCurrent_MAX
[bus_current>max_current]
[BCmaxCnt==1]
{currentfaultLatch |=0x02;
BCmaxCnt=0;}
{BCmaxCnt=1;}
RphaseCurrent_MIN
[Rphase_current<min_current]
[RPminCnt==1]
{currentfaultLatch |=0x04;
RPminCnt=0;}
{RPminCnt=1;}
RphaseCurrent_MAX
[Rphase_current>max_current]
[RPmaxCnt==1]
{currentfaultLatch |=0x08;
RPmaxCnt=0;}
{RPmaxCnt=1;}
YphaseCurrent_MIN
[Yphase_current<min_current]
[YPminCnt==1]
{currentfaultLatch |=0x10;
YPminCnt=0;}
{YPminCnt=1;}
YphaseCurrent_MAX
[Yphase_current>max_current]
[YPmaxCnt==1]
{currentfaultLatch |=0x20;
YPmaxCnt=0;}
{YPmaxCnt=1;}
BphaseCurrent_MIN
[Bphase_current<min_current]
[BPminCnt==1]
{currentfaultLatch |=0x40;
BPminCnt=0;}
{BPminCnt=1;}
BphaseCurrent_MAX
[Bphase_current>max_current]
[BPmaxCnt==1]
{currentfaultLatch |=0x80;
BPmaxCnt=0;}
{BPmaxCnt=1;}
[currentfaultLatch!=0]
[trip_reset==1]
Current_Fault_Latch
Figure 75: current fault checks and latch state-chart

## 2.3.16.6.1.2.3 PID CONTROL BLOCK MODEL

Note that we are controlling the block current; feedback to PID controller is a block current. The feedback block current is derived from the phase current depending on the hall status signal using mutiport switch as shown below. This feedback is filter using FIR filter so as to reduce commutation and switching transient for better feedback signal. Reference current is set through a global storage variable through CAN message from background. Reference filtering using exponential filter is used to reduce jerk in the motor during switch on and off the motor. Shutdown signal and hall error signal will automatically reset the reference block current to zero reference. The tunable parameter like Kp, KiTs and Kd/Ts are accessed using a global storage variable as shown in figure below.

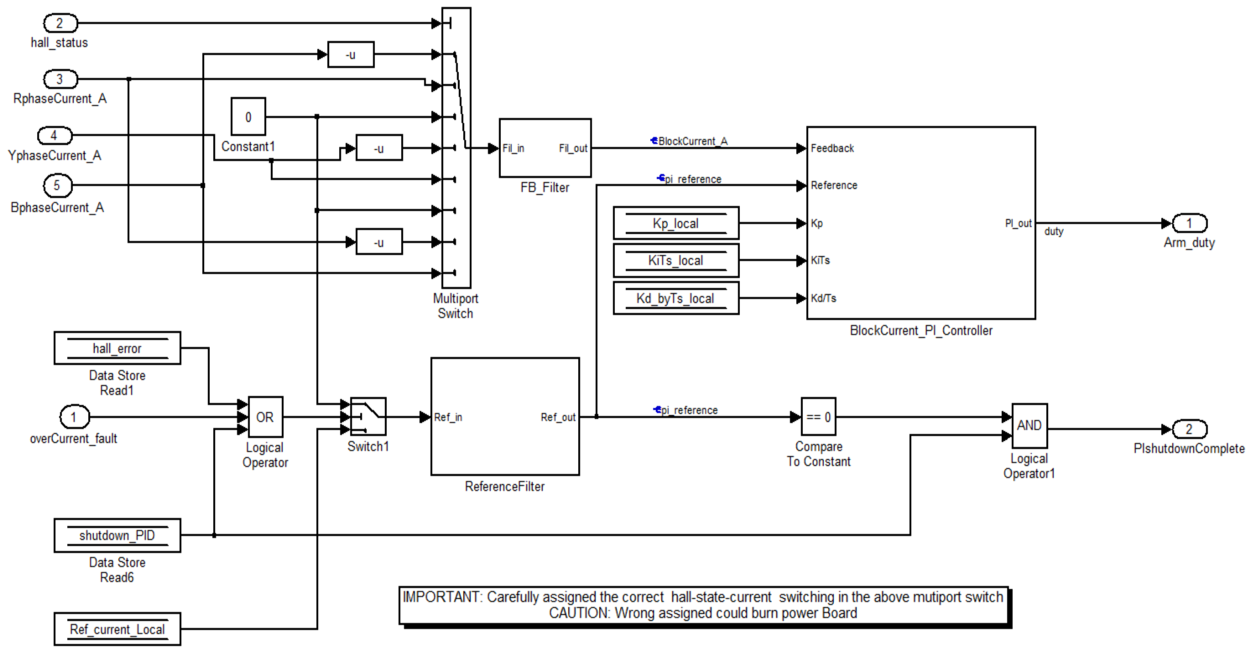


Figure 76: PID controller block model

### *2.3.16.6.1.2.3.1 Feedback Filter block model*

Block current feedback is filter as mentioned using 1st order FIR filter, and is implemented using unit sample delay, addition and shift arithmetic as shown below.

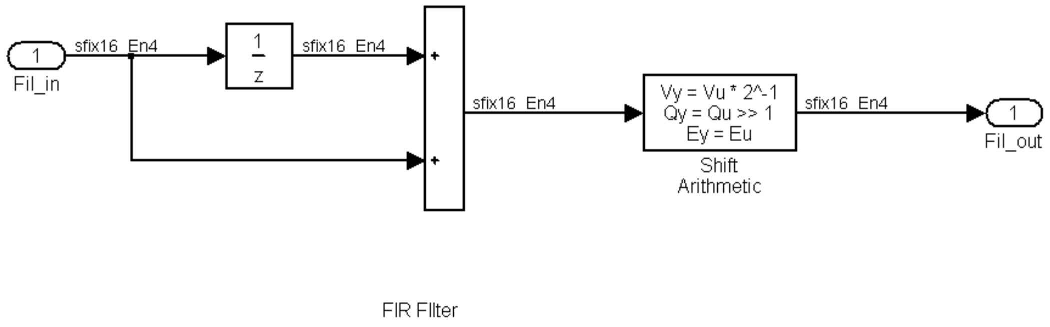


Figure 77 : Feedback FIR filter model

### 2.3.16.6.1.2.3.2 *Reference Filter block model*

Smooth reference signal is implemented using an exponential smoothening algorithm as shown below. Since our processor is an 8-bit fixed-point processor, we implement using shift arithmetic, addition-subtraction and unit delay blocks as shown below.

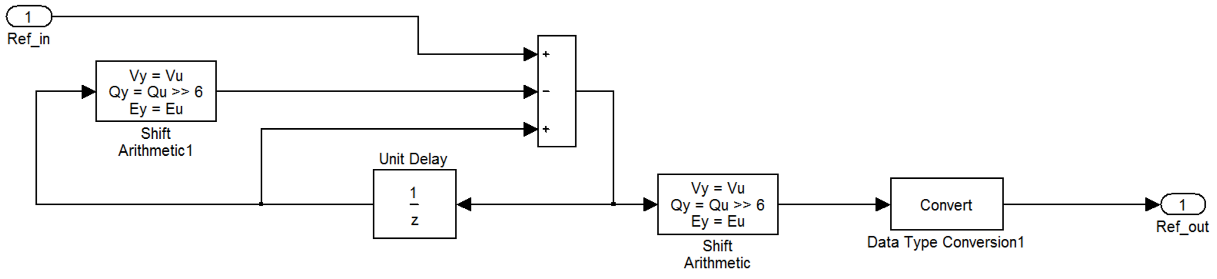


Figure 78: Reference ramp down model

### 2.3.16.6.1.2.3.3 *PID Control block model*

PID controller is implemented in a different way due to the computation constraint of the CPU. All variable are implemented in a fixed point number. To reducing the computational complexity, KiTs and Kd/Ts are directly given as a tunable parameter. Anti-windup reset is also implemented, along with control output saturation block.

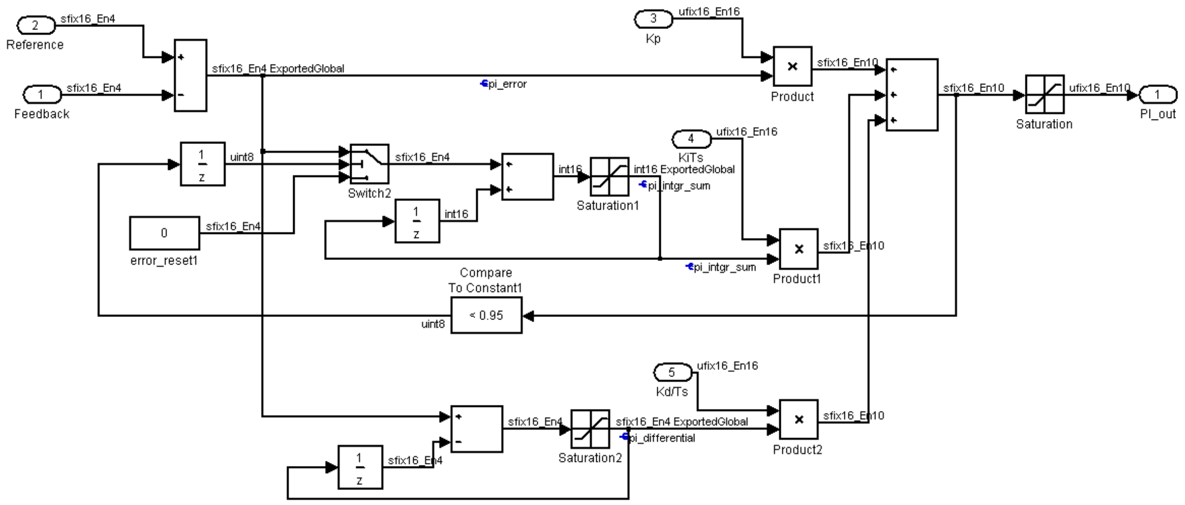


**Figure 79: PI controller model**

### 2.3.16.6.1.3 TIMER1 ISR model

Timer 1 is used for maintaining fixed time step by all algorithms in the background. Timer 1 is configured to interrupt the CPU every 1 msec interval which do the sampling of shown below. The 3phase bridge driver trip latch logic is also executed in this subsystem. Latch state-chart is shown in consecutive figure below.

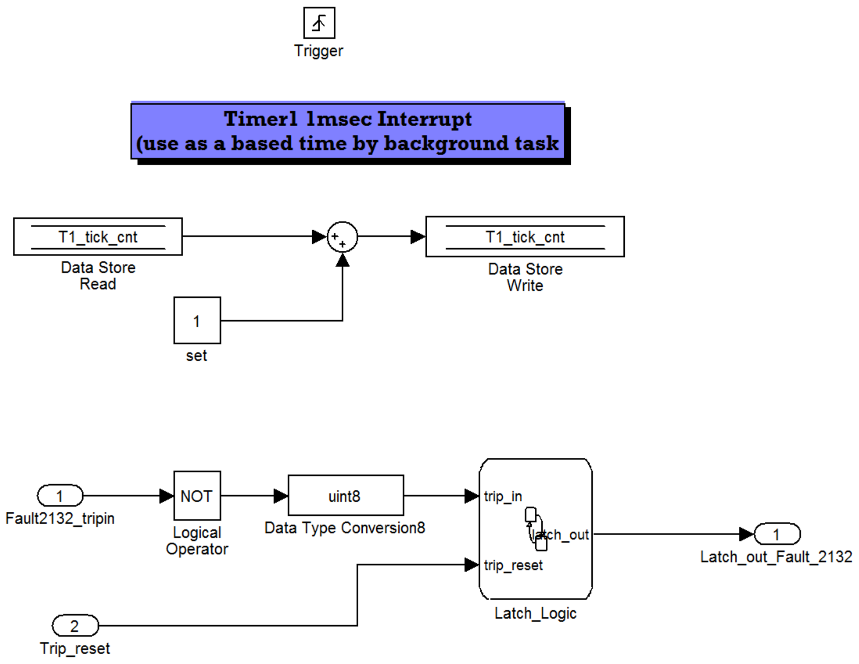


Figure 80: Timer 1 ISR model

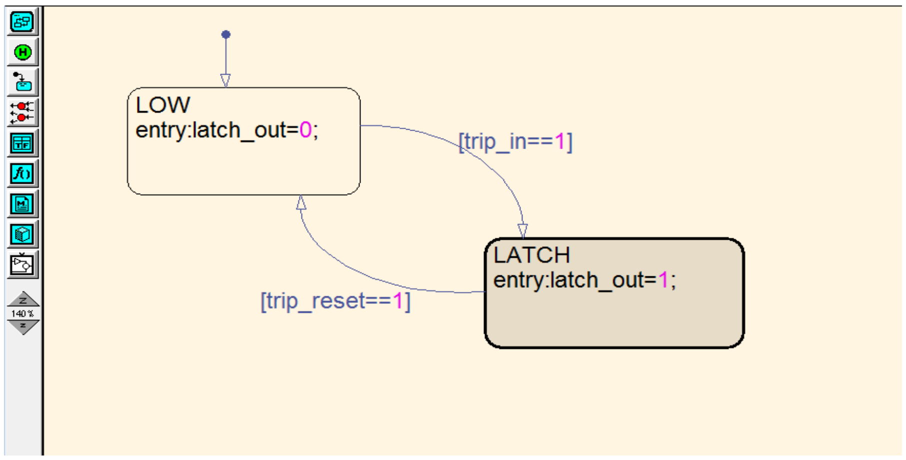
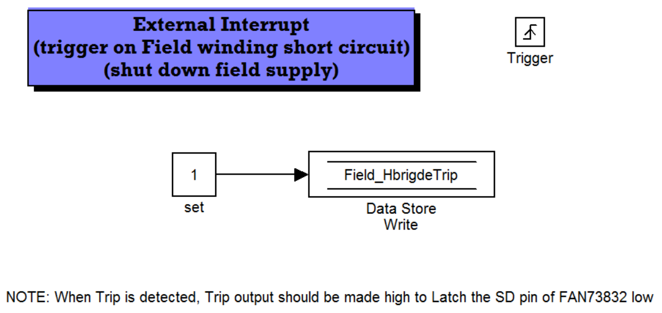


Figure 81: IR2132 Trip Latch state-chart

#### 2.3.16.6.1.4 INT1 ISR (Field short circuit detection) model

When field shunt resister voltage drop is high, it create an interrupt on INT1 pin of the uC through the hardware circuitry as shown in **Figure 33 : Field short circuit detection circuit model**, so a Field H-Bridge trip flag is set, which will shutdown the system from other subsystem.


External Interrupt
(trigger on Field winding short circuit)
(shut down field supply)
Trigger
1
set
Field_HbrigdeTrip
Data Store
Write
NOTE: When Trip is detected, Trip output should be made high to Latch the SD pin of FAN73832 low

#### 2.3.16.6.1.5 Timer5 Overflow ISR model

Timer 5 as mentioned before is used for speed calculation; the hall signal is used as a reference to read the count value from timer 5. But if hall change doesn't occur for sometimes the timer 5 will overflow which create an interrupt, and overflow count will be incremented as shown below.

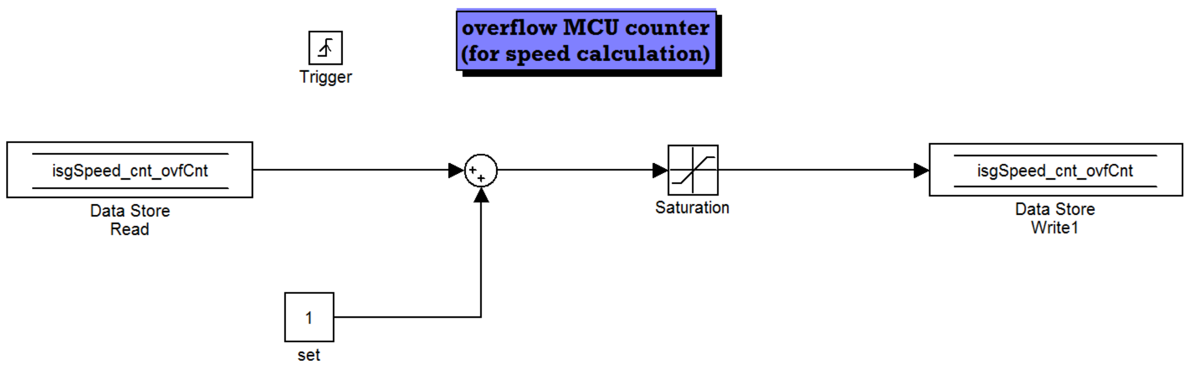


Figure 82: Timer 5 overflow ISR model

### *2.3.16.6.2 BACKGROUND SUBSYSTEM MODEL*

The while loop routine in C is consider as background which will be continuously looped/executed when there is no interrupt to be process by CPU. But in this model we executive two subsystem as shown below in a time slice scheduling algorithm. A time slice is control by a timer1 1msec interrupt.

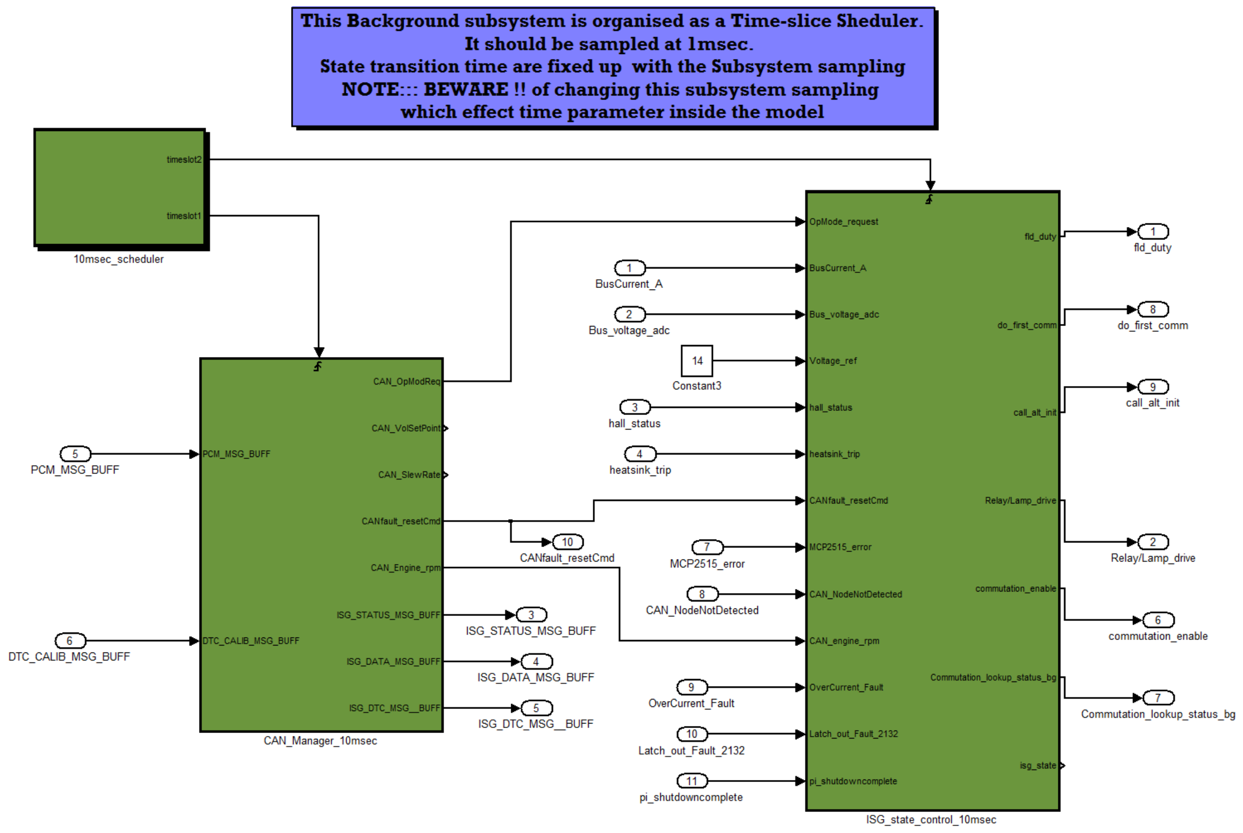


Figure 83: Background subsystem model

#### 2.3.16.6.2.1 Time slice Scheduler model

It scheduled the two subsystem- CAN manager and State control subsystems in a fixed time slice fashion of 50-50 for both. A slice of 5msec is provided for each subsystem. This time slicing distribute the CPU time for other task and ISR to execute with better throughput.

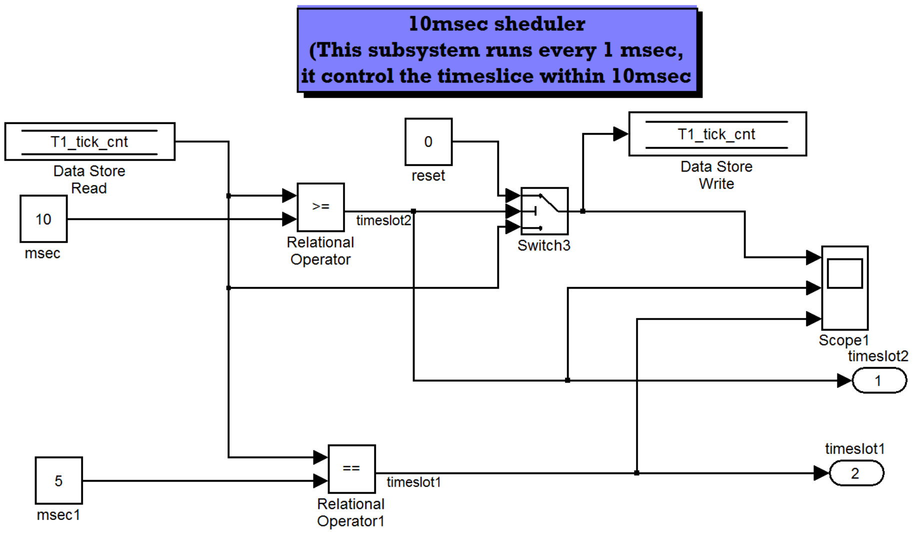


Figure 84: Time slice scheduler

### 2.3.16.6.2.2 CAN Manager Model

CAN message frame are process in this subsystem, all frames are array of bytes and are combined if required to the required data types. Reserve CAN frame for future used are also shown below.

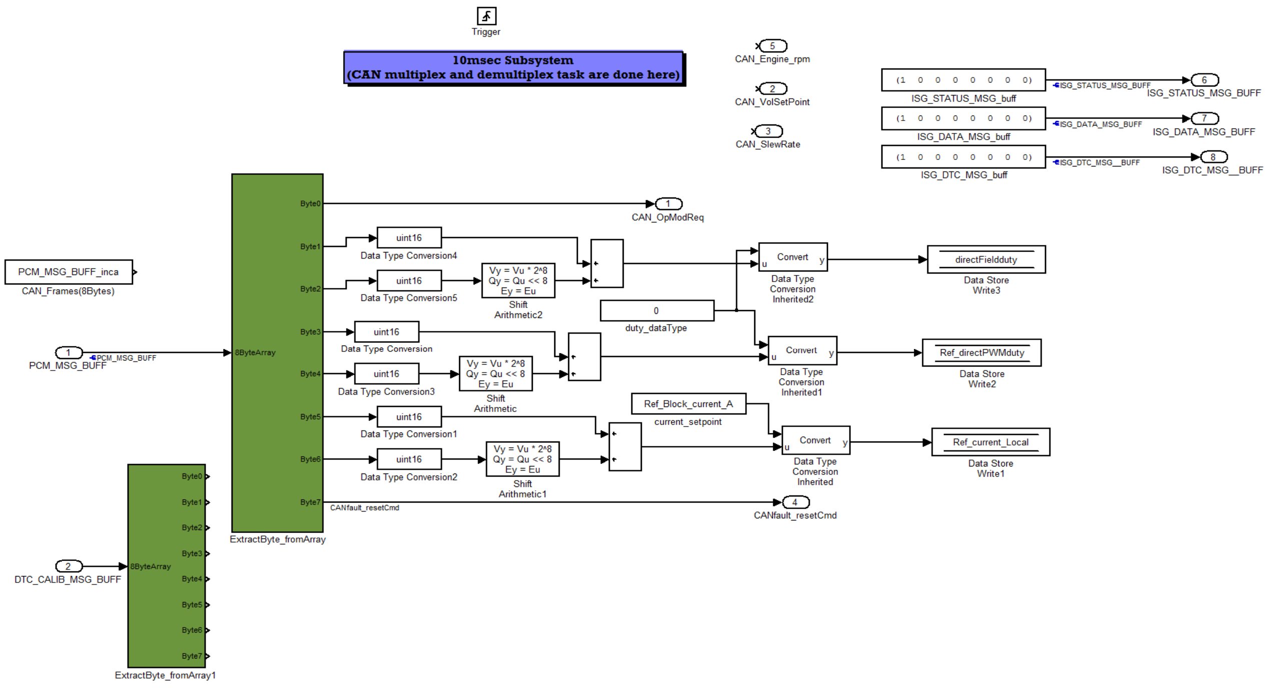


Figure 85: CAN manager model

### 2.3.16.6.2.3 State Control Subsystem model

This is the main control body of this software model. The state-flow based control is used where each of the system state are defined clearly with relevant to the control requirement. In this subsystem, fault manager is put before state control to have a proper fault control. Other background task like speed calculation, battery voltage scaling, relay driver output control are modeled as shown by same subsystem in the below figure.

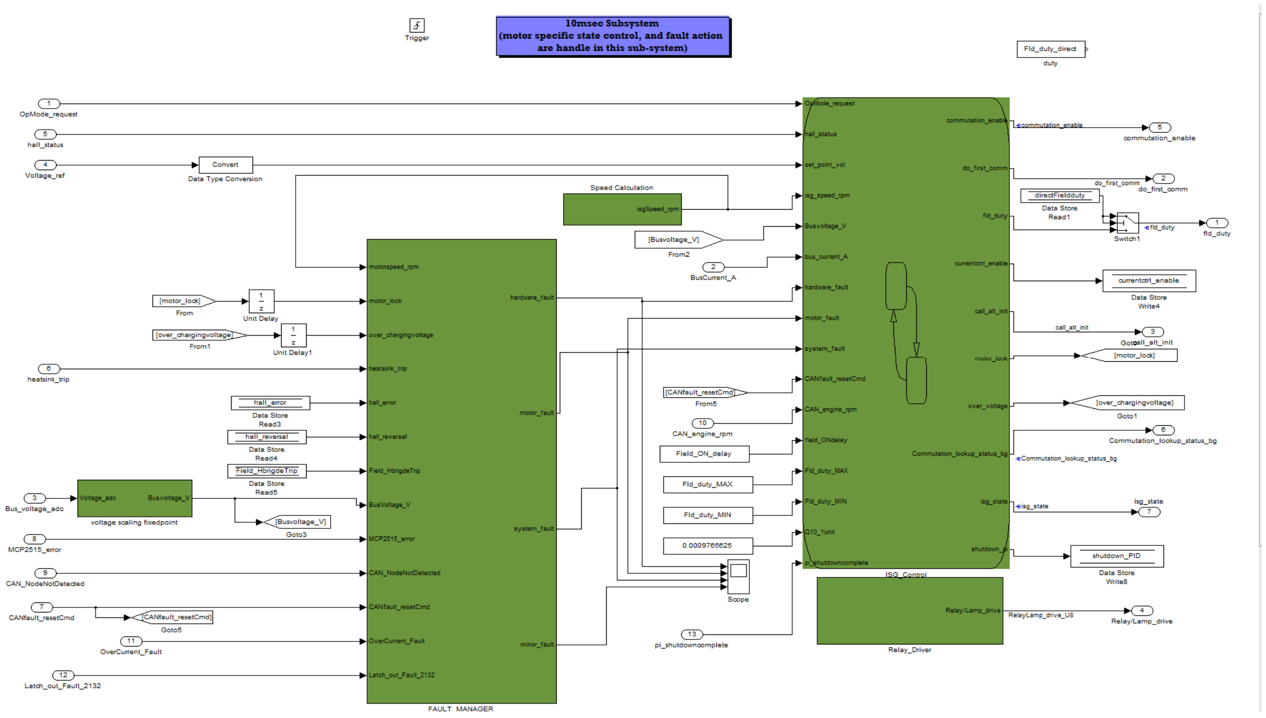


Figure 86: State control subsystem model

#### 2.3.16.6.2.3.1 Voltage scaling in fixed-point model

ADC value of the Bus voltage is converted using a specific scaling factor as shown below.

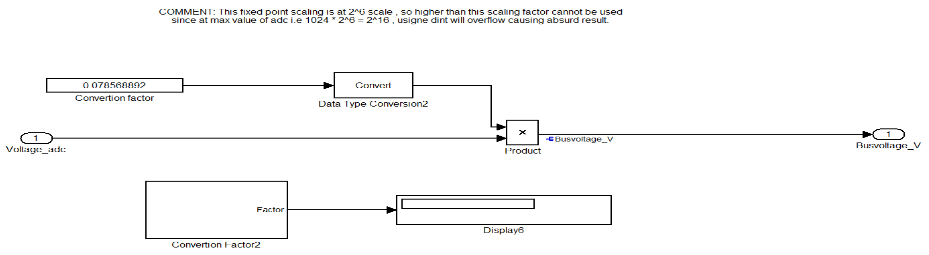
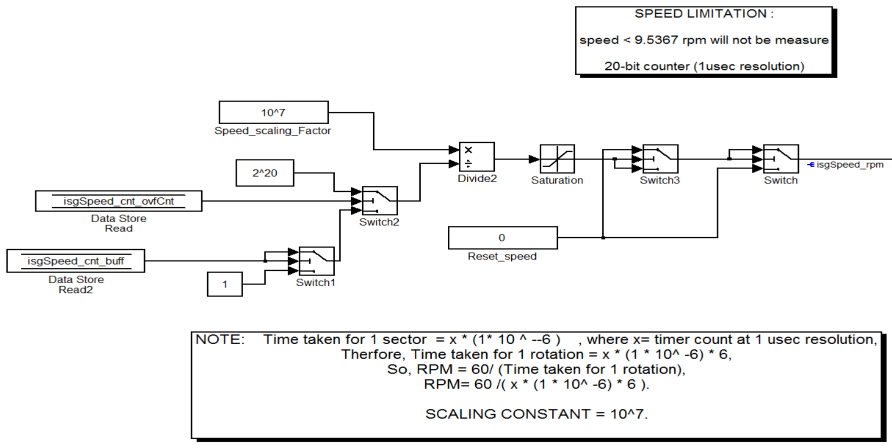

COMMENT: This fixed point scaling is at 2^6 scale , so higher than this scaling factor cannot be used since at max value of adc i.e 1024 * 2^6 = 2^16 , usigne dint will overflow causing absurd result.

Figure 87: Voltage scaling model

### 2.3.16.6.2.3.2 Speed calculation model

Speed in rpm (revolution per minute) is deduced in this subsystem from timer5 count value and overflow count value as shown below.



SPEED LIMITATION :

speed < 9.5367 rpm will not be measure

20-bit counter (1usec resolution)

NOTE: Time taken for 1 sector = x * (1* 10 ^ --6 ) , where x= timer count at 1 usec resolution,
Therfore, Time taken for 1 rotation = x * (1 * 10^ -6) * 6,
So, RPM = 60/ (Time taken for 1 rotation),
RPM= 60 /( x * (1 * 10^ -6) * 6 ).

SCALING CONSTANT = 10^7.

Figure 88: speed calculation model

### 2.3.16.6.2.3.3 Fault manager model

Faults are classified depending on the type and nature of fault. Fault manager is one of the important sub-systems in safety critical control system design. In this development, many of the hardware fault detection circuitry are design and implemented in the controller hardware, but function fault can also be identified in controller software. On the occurrence of fault, the state controls will shutdown the system depending upon the criticality of fault. Minor fault can also exist in a system where the system can still function with it.

Fault classifications are shown below:-

1. Hardware Fault

    a. Heat sink trip.

    b. Field H-Bridge Trip.

    c. IR2132 fault.

2. Motor Fault

    a. Motor Lock.

    b. Overcharge.

    c. Hall error.

    d. Over-speed.

3. System Fault.

    a. Over-voltage

    b. Under-voltage

    c. Bus over-current on positive direction.

    d. Bus over-current on negative direction.

    e. R-phase over-current on positive direction.

    f. R-phase over-current on negative direction.

    g. Y-phase over-current on positive direction.

    h. Y-phase over-current on negative direction.

    i. B-phase over-current on positive direction.

    j. B-phase over-current on negative direction.

4. Minor Fault

    a. Hall reversal

    b. CAN node not detected.

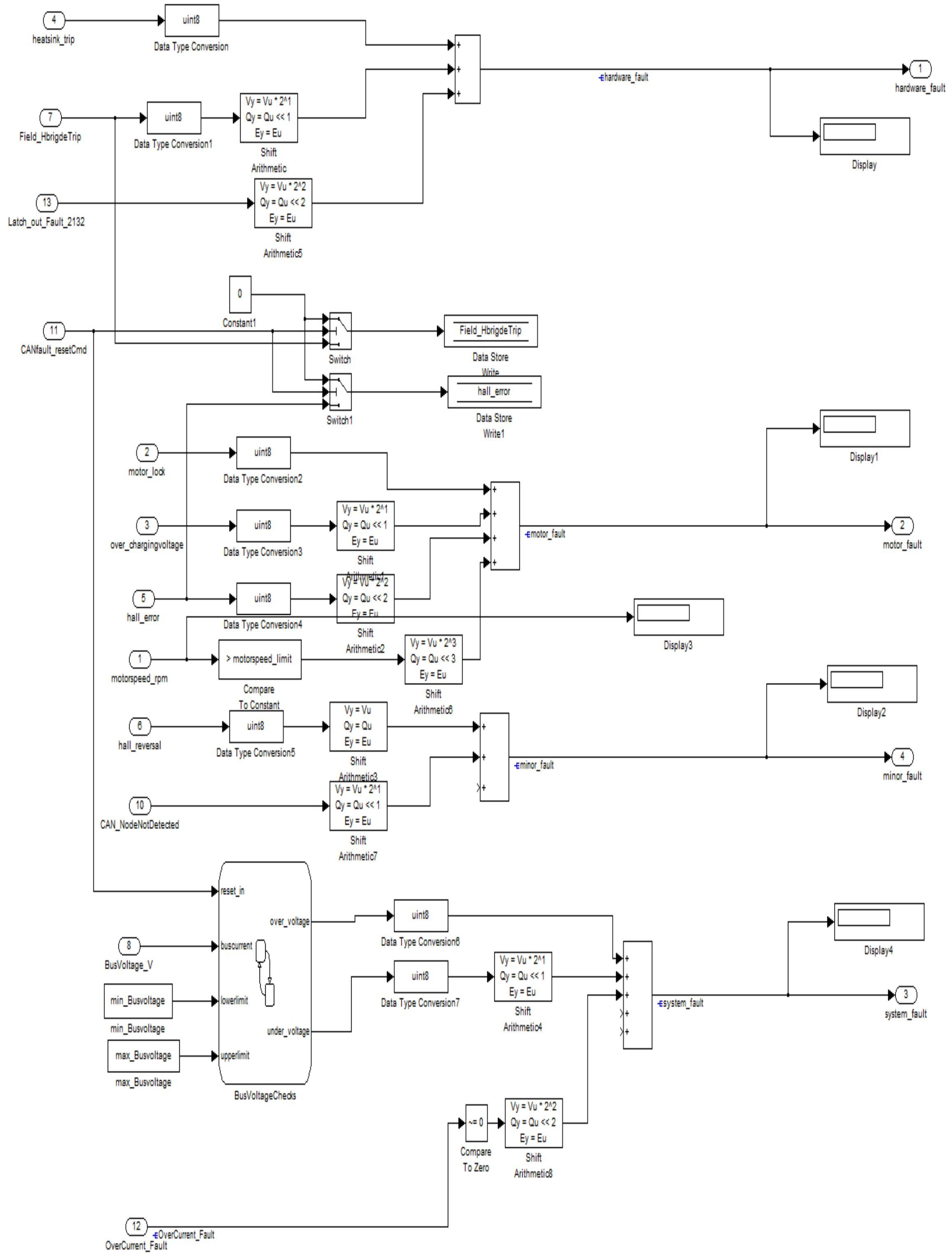


Figure 89: Fault manager model

### 2.3.16.6.2.3.4 ISG Control state-chart model

Control state-chart consists of three main state- idle state, motor state, and alternating state. The default entry or reset state is the idle state, which consist of fault reset sub-state. Whenever switch-off event or fault occurs in a system, it will come to this idle state. One intermediate state is put between the motor state and idle state, called current reference ramp down state, to have a smooth shutdown of PID controller by ramping down the reference signal exponential refer **Figure 78: Reference** . Note that this subsystem will be sampled every 10 msec instances.

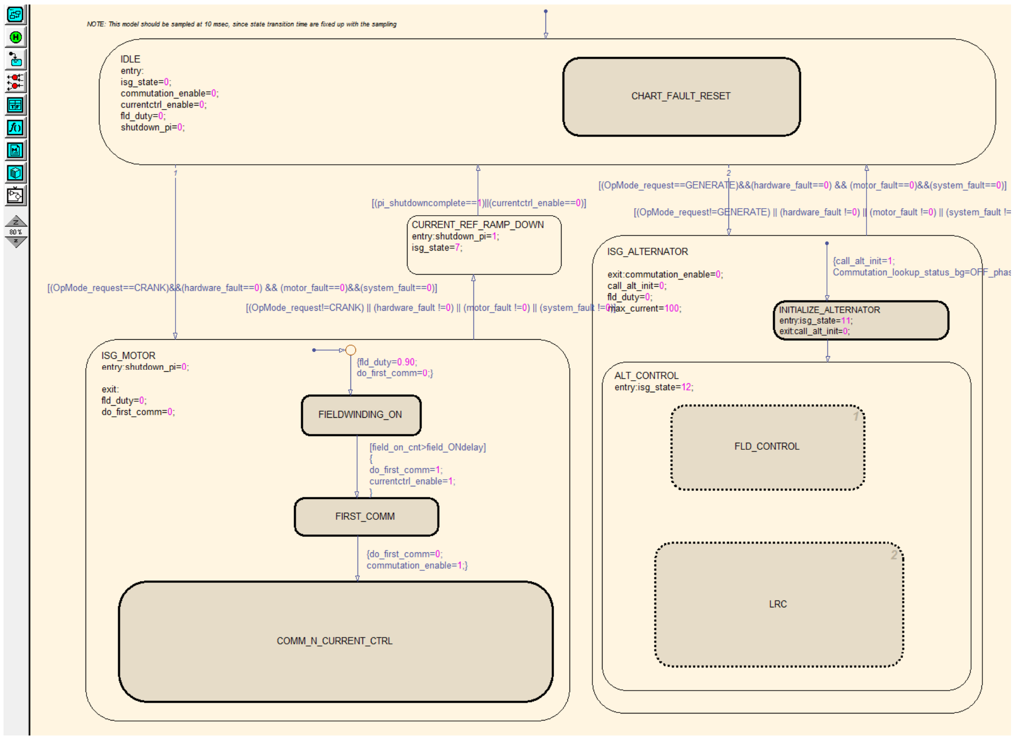

NOTE: This model should be sampled at 10 msec, since state transition time are fixed up with the sampling
IDLE
entry:
isg_state=0;
commutation_enable=0;
currentctrl_enable=0;
fld_duty=0;
shutdown_pi=0;
CHART_FAULT_RESET
1
2
[(pi_shutdowncomplete==1)||(currentctrl_enable==0)]
[(OpMode_request==GENERATE)&&(hardware_fault==0) && (motor_fault==0)&&(system_fault==0)]
[(OpMode_request!=GENERATE) || (hardware_fault !=0) || (motor_fault !=0) || (system_fault !=
CURRENT_REF_RAMP_DOWN
entry:shutdown_pi=1;
isg_state=7;
[(OpMode_request==CRANK)&&(hardware_fault==0) && (motor_fault==0)&&(system_fault==0)]
[(OpMode_request!=CRANK) || (hardware_fault !=0) || (motor_fault !=0) || (system_fault !=0)]
ISG_ALTERNATOR
exit:commutation_enable=0;
call_alt_init=0;
fld_duty=0;
max_current=100;
{call_alt_init=1;
Commutation_lookup_status_bg=OFF_phas
INITIALIZE_ALTERNATOR
entry:isg_state=11;
exit:call_alt_init=0;
ALT_CONTROL
entry:isg_state=12;
FLD_CONTROL
1
LRC
2
ISG_MOTOR
entry:shutdown_pi=0;
exit:
fld_duty=0;
do_first_comm=0;
{fld_duty=0.90;
do_first_comm=0;}
FIELDWINDING_ON
[field_on_cnt>field_ONdelay]
{
do_first_comm=1;
currentctrl_enable=1;
}
FIRST_COMM
{do_first_comm=0;
commutation_enable=1;}
COMM_N_CURRENT_CTRL
80 %

Figure 90: ISG state control chart

#### *2.3.16.6.2.3.4.1 IDLE state*

The isg state number is maintained throughout the operation for diagnostic purpose, the isg state value for this state is 0.

##### ***2.3.16.6.2.3.4.1.1 Fault Reset sub-chart***

This is a sub-state of idle state where fault can be reset in idle state on fault reset event from CAN message.

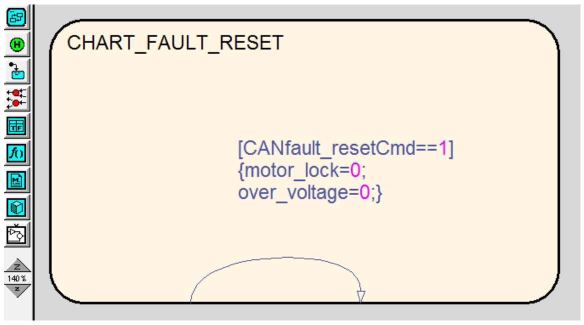


Figure 91: Fault reset chart

#### *2.3.16.6.2.3.4.2 Motor state*

When CAN command from Hybrid ECU is motor command, the state control jump from idle state to this motor state.

##### *2.3.16.6.2.3.4.2.1 Field-winding on sub-chart*

On entry to motor sub-state, it jumps directly to field winding ON sub-state and stay for fixed amount of time i.e. field on delay time, which is a calibration parameter. The isg state number is changed to 1.

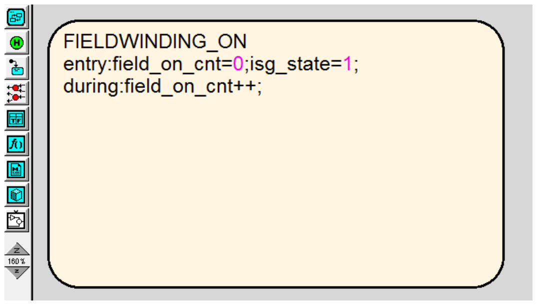


Figure 92: Field winding delay sub-chart

##### *2.3.16.6.2.3.4.2.2 First Commutation sub-chart*

When the motor is turn on, PWM output override should be first set from background so that duty set by Current control from foreground will be effective. So, depending on the hall status PWM output override is set in this sub-chart as shown below. And the isg state value is set to 2.

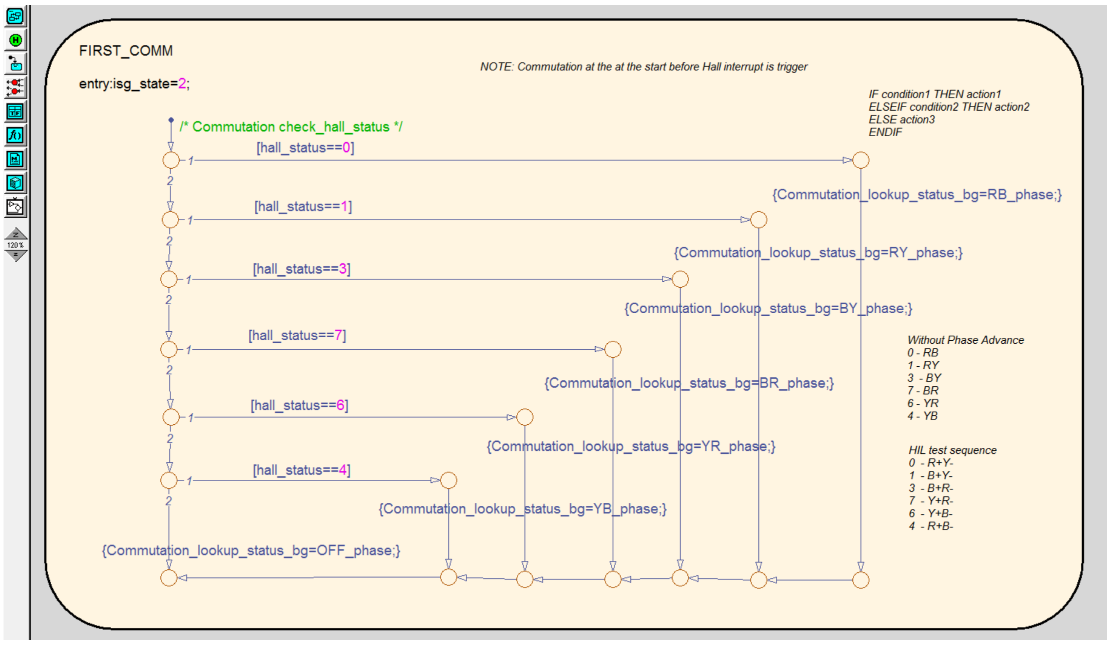

FIRST_COMM
entry:isg_state=2;
NOTE: Commutation at the at the start before Hall interrupt is trigger
IF condition1 THEN action1
ELSEIF condition2 THEN action2
ELSE action3
ENDIF
/* Commutation check_hall_status */
[hall_status==0]
{Commutation_lookup_status_bg=RB_phase;}
[hall_status==1]
{Commutation_lookup_status_bg=RY_phase;}
[hall_status==3]
{Commutation_lookup_status_bg=BY_phase;}
[hall_status==7]
{Commutation_lookup_status_bg=BR_phase;}
[hall_status==6]
{Commutation_lookup_status_bg=YR_phase;}
[hall_status==4]
{Commutation_lookup_status_bg=YB_phase;}
{Commutation_lookup_status_bg=OFF_phase;}
Without Phase Advance
0 - RB
1 - RY
3 - BY
7 - BR
6 - YR
4 - YB
HIL test sequence
0 - R+Y-
1 - B+Y-
3 - B+R-
7 - Y+R-
6 - Y+B-
4 - R+B-

Figure 93: First commutation sub-chart

### *2.3.16.6.2.3.4.2.3 Commutation and current control sub-chart*

Once first commutation is done the motor will rotate, and the foreground PID control come into effect to control the block current by updating armature PWM duty. And also change of commutation on hall status change will be handled by commutation subsystem in foreground at the same time. The background state control will be jumping to this sub-state, and continuously check the lock condition. The isg state will be set to 3, and if the machine rotate and speed appear it will jump to motor run sub-state, and then set isg state to 5. If the load torque is high with respect to the set reference current, lock can happen i.e. isg state 4.

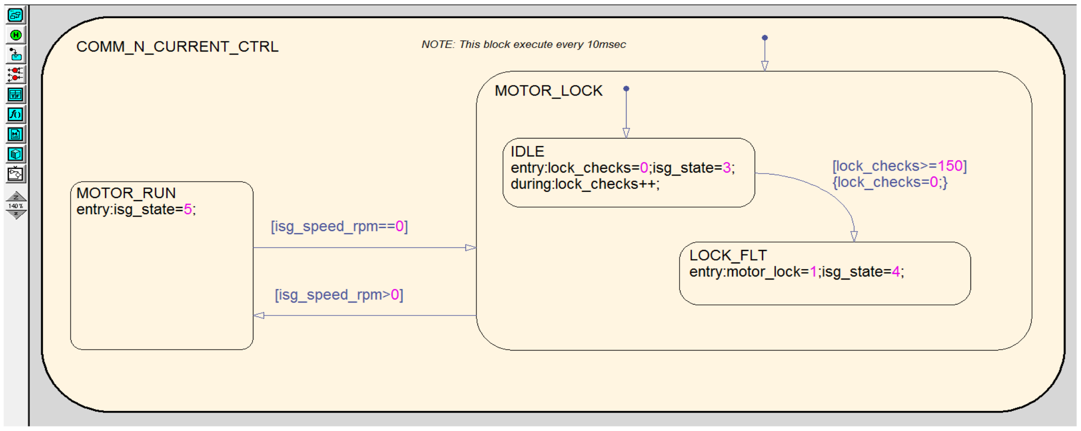


Figure 94: commutation and current control sub-chart

### *2.3.16.6.2.3.4.3 Alternator state*

Normal alternator operation is achieved by switching off all the MOSFET and controlling the field current by H-bridge. The back emf generated by the machine should be higher than the battery voltage, so that power can flow from engine to battery, or machine to battery. All motor functions like current control will be off, and only field current is control thereby controlling the charging voltage i.e. the Bus voltage. So, field control and load response control state-chart comes into action together during this operation. The isg state is set to 11 and then 12. State 11 is for initialization of alternating operation.

#### *2.3.16.6.2.3.4.3.1 Field Control sub-chart (Alternator mode)*

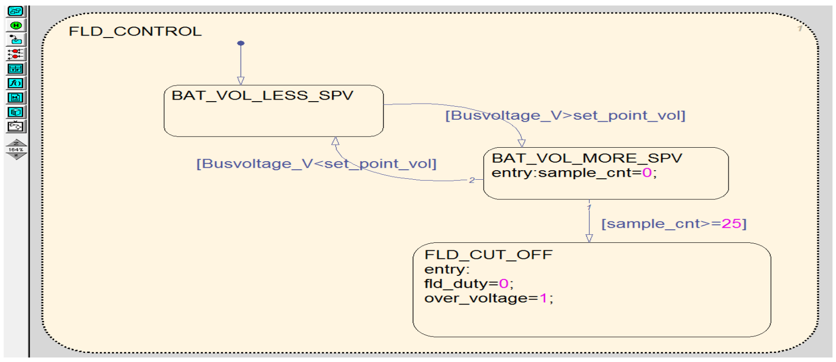


Figure 95: Field control sub-chart

#### *2.3.16.6.2.3.4.3.2 Load response control sub-chart (Alternator mode)*

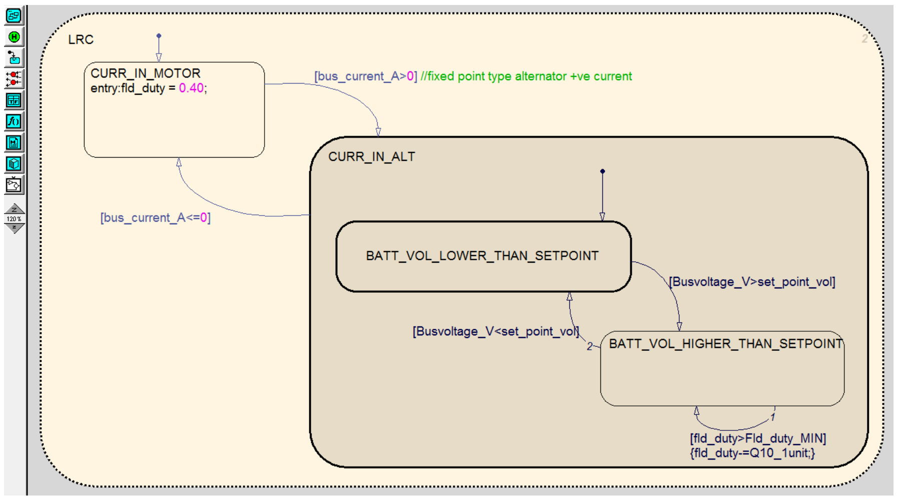

LRC
CURR_IN_MOTOR
entry:fld_duty = 0.40;
[bus_current_A>0] //fixed point type alternator +ve current
[bus_current_A<=0]
CURR_IN_ALT
BATT_VOL_LOWER_THAN_SETPOINT
[Busvoltage_V>set_point_vol]
[Busvoltage_V<set_point_vol]
BATT_VOL_HIGHER_THAN_SETPOINT
[fld_duty>Fld_duty_MIN]
{fld_duty-=Q10_1unit;}

Figure 96: Load response control sub-chart1

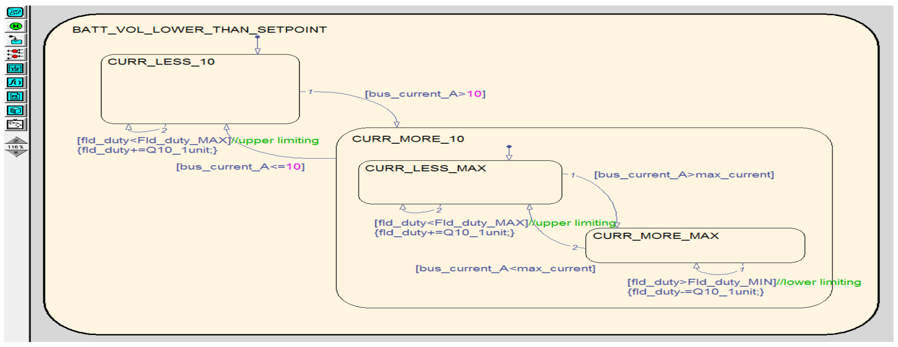

BATT_VOL_LOWER_THAN_SETPOINT
CURR_LESS_10
[bus_current_A>10]
[fld_duty<Fld_duty_MAX]//upper limiting
{fld_duty+=Q10_1unit;}
[bus_current_A<=10]
CURR_MORE_10
CURR_LESS_MAX
[bus_current_A>max_current]
[fld_duty<Fld_duty_MAX]//upper limiting
{fld_duty+=Q10_1unit;}
CURR_MORE_MAX
[bus_current_A<max_current]
[fld_duty>Fld_duty_MIN]//lower limiting
{fld_duty-=Q10_1unit;}

Figure 97: Load response control sub-chart2

The alternator mode of control is out of the scope of this present dissertation report, and also MIL and HIL validation is not done for this mode of operation. It will be a future improvement and on-going development, where synchronous rectification algorithm or boosting operation will also be done. So, we will skip at this point for the time being.

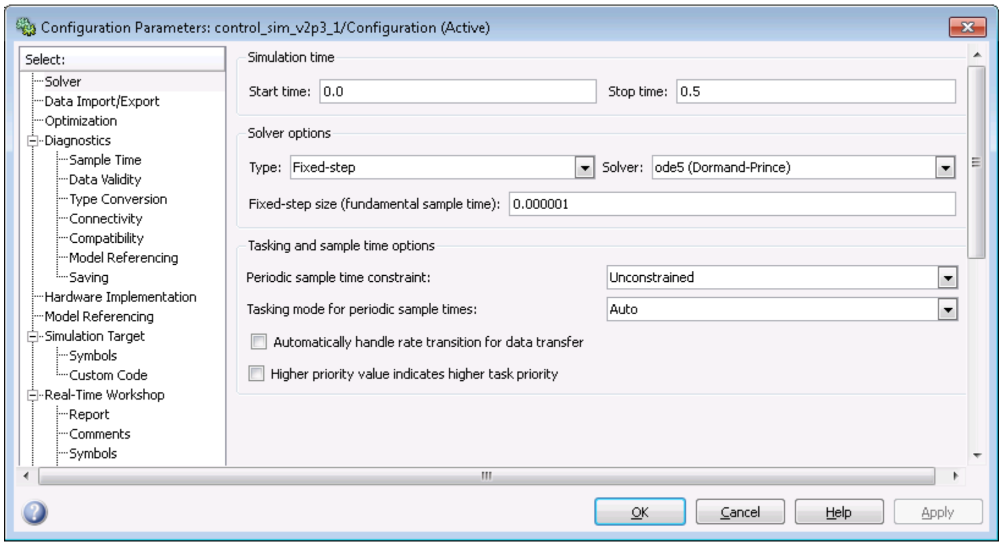
# 3 CHAPTER 3: MODEL-IN-LOOP SIMULATION

## 3.1 Concept of MBD for MIL

In Model-Based Design (MBD) there would be a system model at the center of the development process. The significant feature of this design is that it facilitates quicker and more cost-effective development of dynamic systems. With built-in mathematical functions and routines these tools are optimized for designing and analyzing control strategies through off-line simulation. Moreover these tools can be integrated with real-time hardware which means integrating traditional off-line simulation with real-world testing. MATLAB Simulink facilitates designing of the control algorithm and also helps in executing off-line simulation on the desktop. But it doesn't mean that with software simulations all the distinctive behaviors of an actual dynamic environment can be accounted for.

The Model-in-the-loop simulation captures the specified behavior of the model that is to be implemented in C code later on. These simulated results are validated with the requirements. Also it acts as the verification reference for the next stages of development cycle. Since the model acts as the design document, it reduces the defects slippage due to translation of requirements to design.

## 3.2 Simulation configuration


Configuration Parameters: control_sim_v2p3_1/Configuration (Active)

Simulation time
Start time: 0.0
Stop time: 0.5

Solver options
Type: Fixed-step
Solver: ode5 (Dormand-Prince)
Fixed-step size (fundamental sample time): 0.000001

Tasking and sample time options
Periodic sample time constraint: Unconstrained
Tasking mode for periodic sample times: Auto
☐ Automatically handle rate transition for data transfer
☐ Higher priority value indicates higher task priority

Figure 98: MIL configuration parameter

Model in loop (MIL) simulation is performed in MATLAB Simulink with the configuration as shown in **Figure 98: MIL configuration parameter** above. The fixed step sampling rate of 1 usec is used in the simulation with solver ode5 (Dormand-Price). The simulation is performed at fixed-step discrete environment. **1 usec** fixed-sampling is chosen to have a better PWM duty resolution and also for the based time count for the timer tick. SimPower system block is also given a good result with this sampling rate.

The limitation and assumption of the simulation are defined in the previous chapter in section 2.2.2.1 above and section 2.3.1 above.

## 3.3 SIMULATION RESULTS

System model are checked here, and a specific functions or integrity of whole model are checked. The controller model is validated to functional level with respect to phase block current control. Some fault conditions are also simulated here. Control parameter like the PID controller tunable parameter like Kp, Ki, Kd are tune in MIL level simulation. The plan dynamics are observe and accordingly controller are modeled as shown in chapter 2 of this report.

### 3.3.1 Lock condition from start

By setting the load torque very high with respect to the block current reference, current control at lock condition is observed as shown below.

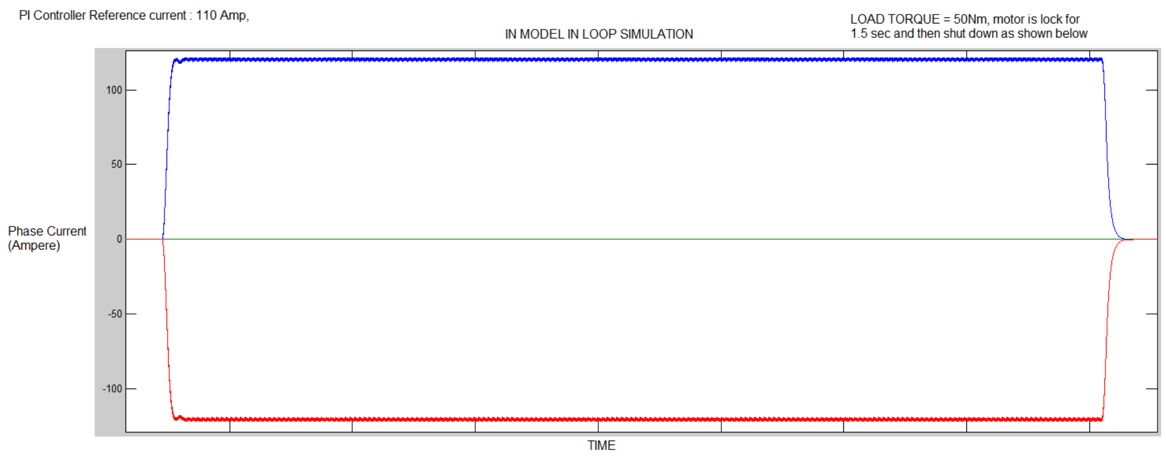


Figure 99: Lock condition from start (MIL)

### 3.3.2 Motor torque higher than load torque

When the Load torque set in the plan model is lesser than the torque produce by the specific controlled phase current, the motor will rotate showing a change in phase current direction due to commutation cause by rotation. Below result shows the phase current at 20Nm Load torque and block current reference to 150 Ampere, it has been observed that commutation causes spike in the current, but is latter controlled by PID controller.

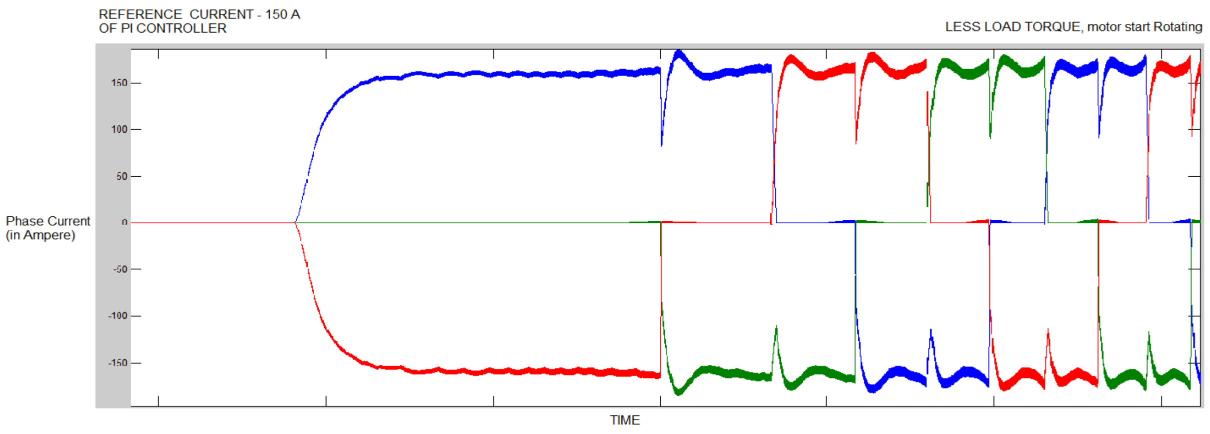


**Figure 100: motor running (MIL)**

### 3.3.3 Lock condition while running

When load torque is made suddenly high during the running condition of the machine, rotor can get locked as shown below and the controller shutdown once it detect lock fault.

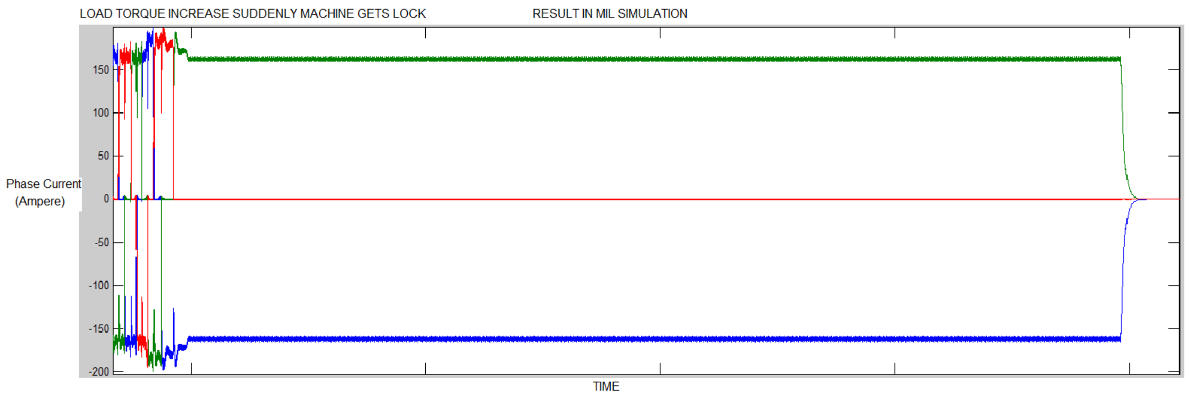
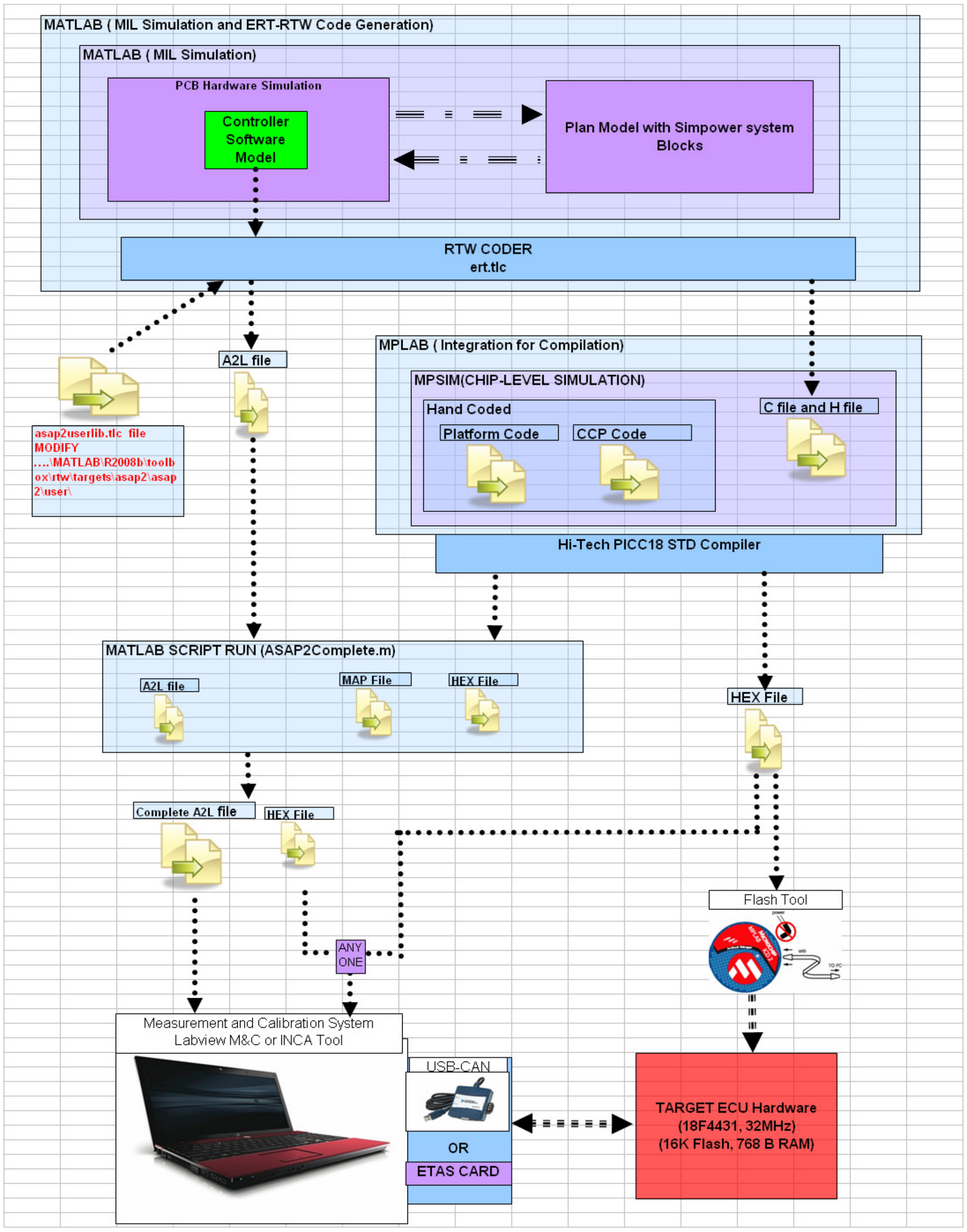


Figure 101: Motor lock while running (MIL)

# 4 CHAPTER 4: BUILD PROCESS

## 4.1 Tool chain

The tool chain of this project is shown in the figure below.



MATLAB ( MIL Simulation and ERT-RTW Code Generation)
MATLAB ( MIL Simulation)
PCB Hardware Simulation
Controller Software Model
Plan Model with Simpower system Blocks
RTW CODER
ert.tlc
asap2userlib.tlc file MODIFY ....\MATLAB\R2008b\toolbox\rtw\targets\asap2\asap2\user\
A2L file
MPLAB ( Integration for Compilation)
MPSIM(CHIP-LEVEL SIMULATION)
Hand Coded
Platform Code
CCP Code
C file and H file
Hi-Tech PICC18 STD Compiler
MATLAB SCRIPT RUN (ASAP2Complete.m)
A2L file
MAP File
HEX File
HEX File
Complete A2L file
HEX File
ANY ONE
Flash Tool
Measurement and Calibration System
Labview M&C or INCA Tool
USB-CAN
OR
ETAS CARD
TARGET ECU Hardware
(18F4431, 32MHz)
(16K Flash, 768 B RAM)

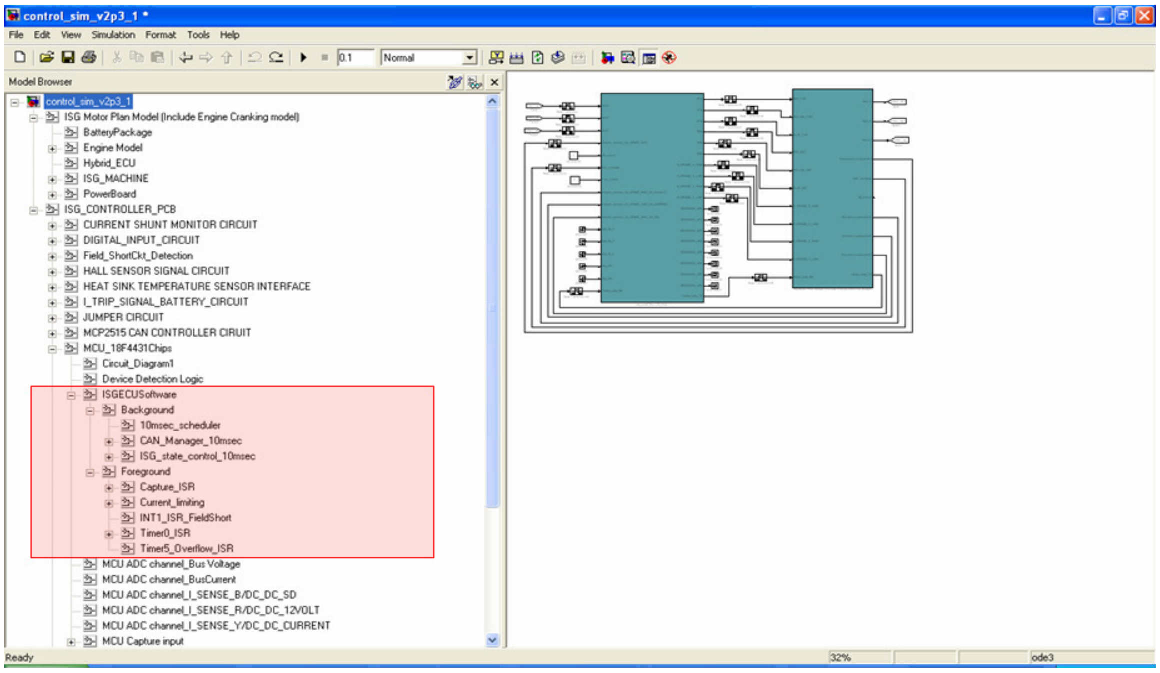
Figure 102: Development tool chain

## 4.2 Real-time workshop build process

Auto-coding is done using embedded real-time workshop; the same MIL model is used in which a specific M-script **[Appendix I]** is run before code generation to copy out software part of the model to a new model from where actual code is generated. The generated code is located at specific location which consists of the C source code, and A2L file which contains a measurement and calibration variable data description file with address token. The generated code along with hand-code are combine in MPLAB integrated development environment, and are compiled using Hi-tech PICC18 C compiler, and then flash to uC using ICD2/ ICD3 in-circuit debugger. Map file generated after compiling contains the address of variable in RAM as well as on Flash memory. The A2L file containing the address token must resolved by some method discussed later topic in this chapter.

### 4.2.1 Subsystem for code generation

As mentioned, code generation is done from a specific part of the model subsystem (ISGECUSoftware) as highlighted in the figure below. Other part of the model will not be used for code generation; it is used only for MIL simulation.



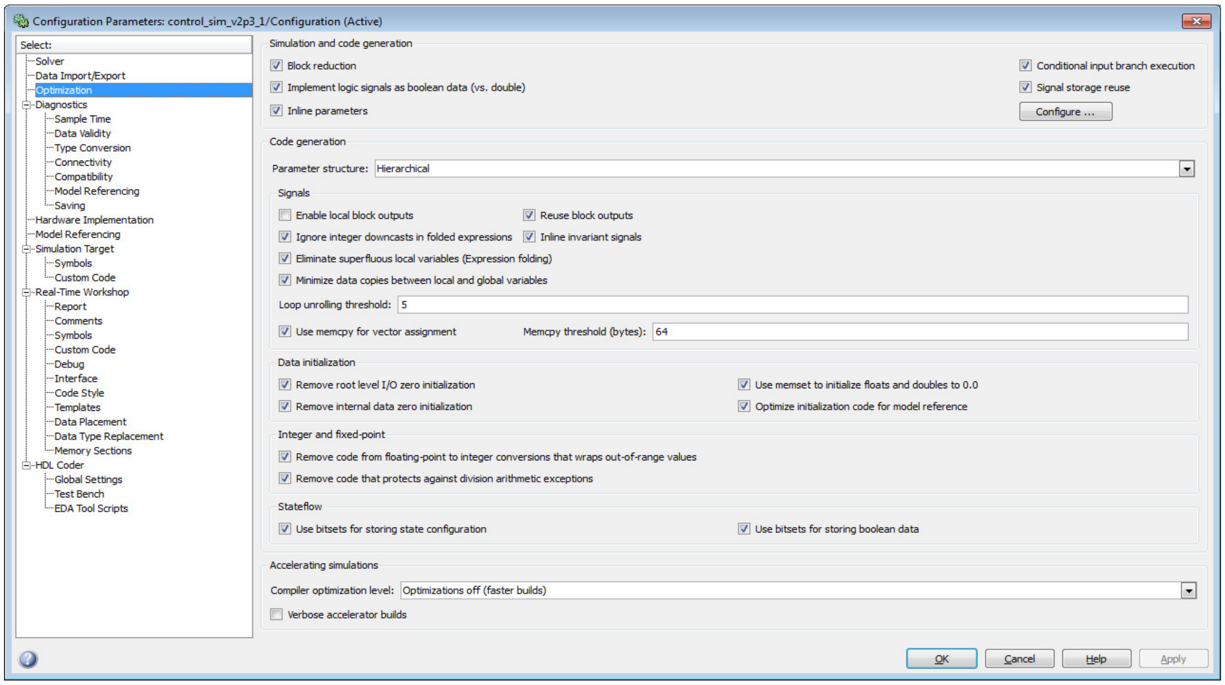
```
control_sim_v2p3_1
  ISG Motor Plan Model (Include Engine Cranking model)
    BatteryPackage
    Engine Model
    Hybrid_ECU
    ISG_MACHINE
    PowerBoard
  ISG_CONTROLLER_PCB
    CURRENT SHUNT MONITOR CIRCUIT
    DIGITAL_INPUT_CIRCUIT
    Field_ShortCkt_Detection
    HALL SENSOR SIGNAL CIRCUIT
    HEAT SINK TEMPERATURE SENSOR INTERFACE
    I_TRIP_SIGNAL_BATTERY_CIRCUIT
    JUMPER CIRCUIT
    MCP2515 CAN CONTROLLER CIRUIT
    MCU_18F4431Chips
      Circuit_Diagram1
      Device Detection Logic
      ISGECUSoftware
        Background
          10msec_scheduler
          CAN_Manager_10msec
          ISG_state_control_10msec
        Foreground
          Capture_ISR
          Current_limiting
          INT1_ISR_FieldShort
          Timer0_ISR
          Timer5_Overflow_ISR
      MCU ADC channel_Bus Voltage
      MCU ADC channel_BusCurrent
      MCU ADC channel_I_SENSE_B/DC_DC_SD
      MCU ADC channel_I_SENSE_R/DC_DC_12VOLT
      MCU ADC channel_I_SENSE_Y/DC_DC_CURRENT
      MCU Capture input
```

Figure 103: subsystem for code generation

### 4.2.2 Source code and A2L file generation

#### 4.2.2.1 Configure optimization for code generation

Optimization option of MATLAB for block simulation and code generation are configure from the optimization dialog of configuration parameter. All most all optimization options are checked as shown below for better code footprint.



Simulation and code generation

- ✓ Block reduction
- ✓ Implement logic signals as boolean data (vs. double)
- ✓ Inline parameters
- ✓ Conditional input branch execution
- ✓ Signal storage reuse

Code generation

Parameter structure: Hierarchical

Signals

- ☐ Enable local block outputs
- ✓ Reuse block outputs
- ✓ Ignore integer downcasts in folded expressions
- ✓ Inline invariant signals
- ✓ Eliminate superfluous local variables (Expression folding)
- ✓ Minimize data copies between local and global variables

Loop unrolling threshold: 5

- ✓ Use memcpy for vector assignment

Memcpy threshold (bytes): 64

Data initialization

- ✓ Remove root level I/O zero initialization
- ✓ Use memset to initialize floats and doubles to 0.0
- ✓ Remove internal data zero initialization
- ✓ Optimize initialization code for model reference

Integer and fixed-point

- ✓ Remove code from floating-point to integer conversions that wraps out-of-range values
- ✓ Remove code that protects against division arithmetic exceptions

Stateflow

- ✓ Use bitsets for storing state configuration
- ✓ Use bitsets for storing boolean data

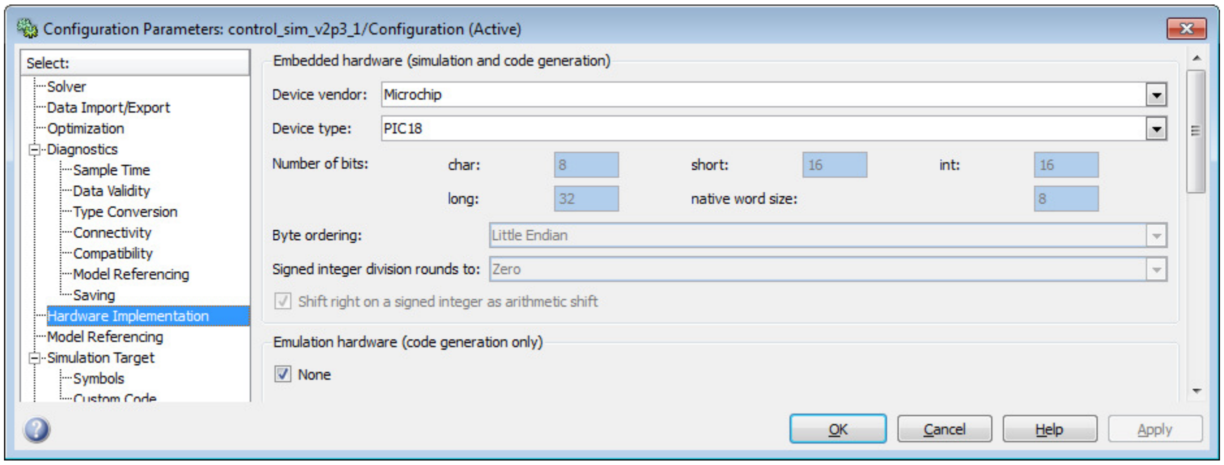
Accelerating simulations

Compiler optimization level: Optimizations off (faster builds)

- ☐ Verbose accelerator builds

Figure 104: configure optimization

Since microchip PIC18 was used, Embedded Target preference is set as below.



Embedded hardware (simulation and code generation)

Device vendor: Microchip

Device type: PIC18

Number of bits: char: 8 short: 16 int: 16

long: 32 native word size: 8

Byte ordering: Little Endian

Signed integer division rounds to: Zero

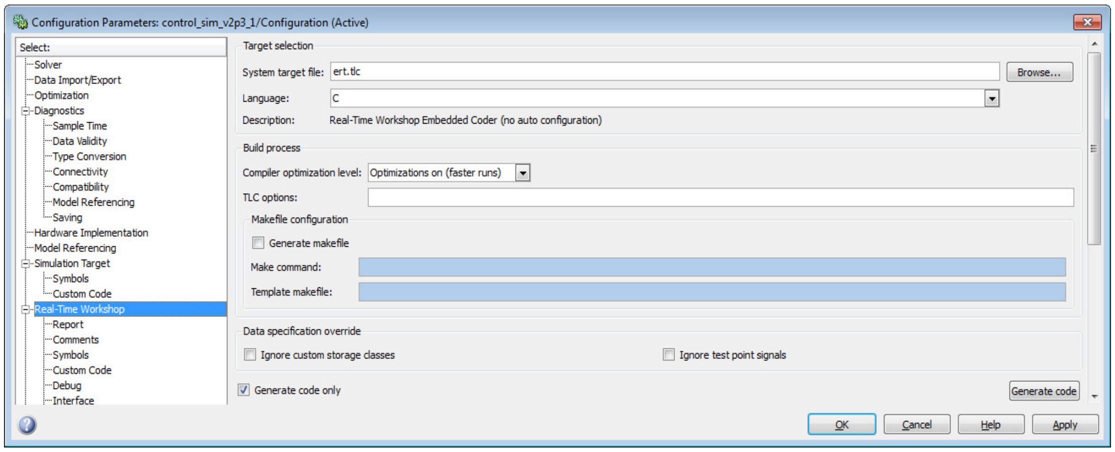
- ✓ Shift right on a signed integer as arithmetic shift

Emulation hardware (code generation only)

- ✓ None

Figure 105: configure hardware implementation

Code generation in MATLAB RTW is fully control by target language compiler of MATLAB which can also be configured depending on requirement using TLC programming. Ready-made ert.tlc is available on MATLAB, which is used directly here as shown.



Configuration Parameters: control_sim_v2p3_1/Configuration (Active)

Target selection
System target file: ert.tlc
Language: C
Description: Real-Time Workshop Embedded Coder (no auto configuration)

Build process
Compiler optimization level: Optimizations on (faster runs)
TLC options:
Makefile configuration
☐ Generate makefile
Make command:
Template makefile:

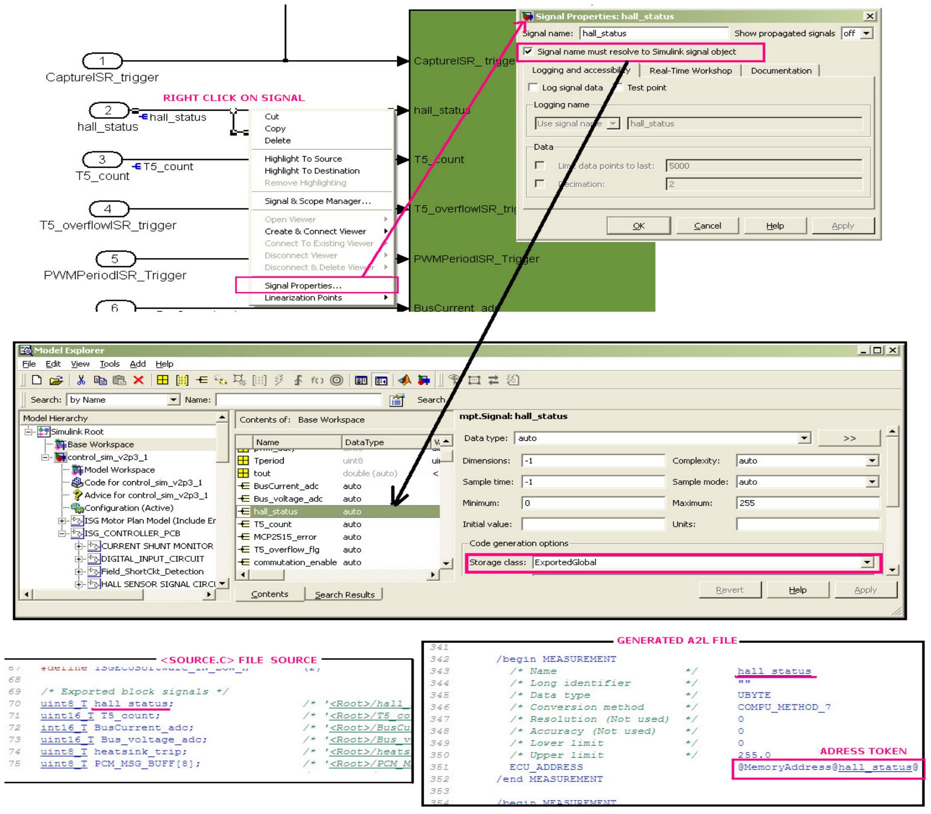
Data specification override
☐ Ignore custom storage classes
☐ Ignore test point signals

☑ Generate code only

Figure 106: Configure RTW target link compiler

### 4.2.2.2 Configure measurement variable

Signal name should be assigned for all measurement variables using signal properties widget as shown in figure below.



Signal Properties: hall_status
Signal name: hall_status
Show propagated signals: off
☑ Signal name must resolve to Simulink signal object

mpt.Signal: hall_status
Data type: auto
Dimensions: -1
Complexity: auto
Sample time: -1
Sample mode: auto
Minimum: 0
Maximum: 255
Initial value:
Units:
Code generation options
Storage class: ExportedGlobal

```
/* Exported block signals */
uint8_T hall_status;             /* '<Root>/hall_
uint16_T T5_count;               /* '<Root>/T5_co
int16_T BusCurrent_adc;          /* '<Root>/BusCu
uint16_T Bus_voltage_adc;        /* '<Root>/Bus_v
uint8_T heatsink_trip;           /* '<Root>/heats
uint8_T PCM_MSG_BUFF[8];         /* '<Root>/PCM_M
```

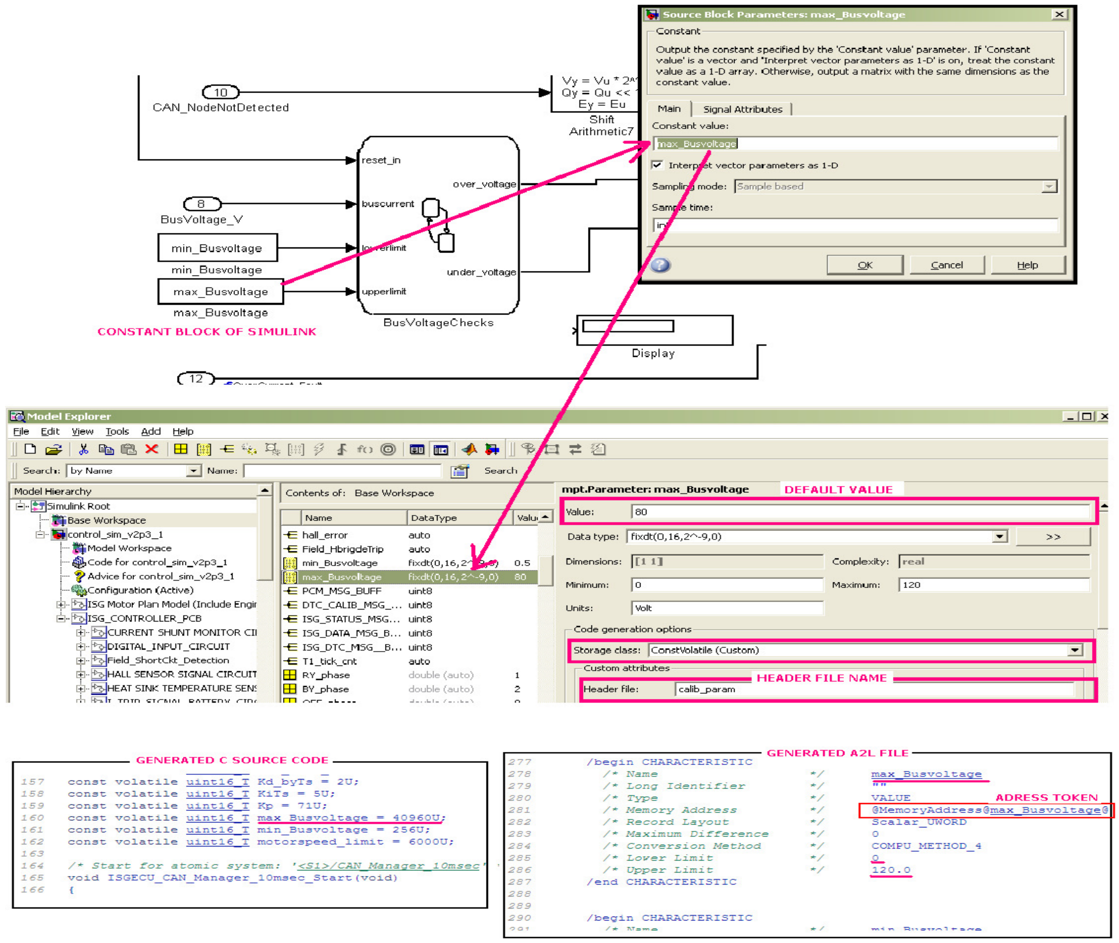
```
    /begin MEASUREMENT
      /* Name                    */      hall_status
      /* Long identifier         */      ""
      /* Data type               */      UBYTE
      /* Conversion method       */      COMPU_METHOD_7
      /* Resolution (Not used)   */      0
      /* Accuracy (Not used)     */      0
      /* Lower limit             */      0
      /* Upper limit             */      255.0
      ECU_ADDRESS                        @MemoryAddress@hall_status@
    /end MEASUREMENT
```

Figure 107: configure measurement variables

Note that the same signal should be created on model explorer widget before selecting checkbox (signal name must resolve from Simulink signal object) and object should be created under mpt.Signal object. Signal properties like dimension, complexity, sampling time, sampling mode, minimum, maximum, initial value, units and storage class can be configured from model explorer widget itself. Storage class for measurement variables should be defined as exported global, which will be generated as a global variable in C.

### 4.2.2.3 Configure Calibration parameter

Calibration parameters are used in the model using constant block whose name are resolve from model explorer widget, using mpt.Parameter object whose properties like value, dimension, complexity, minimum, maximum, units, storage class and header file name. It can be assigned as shown in figure below. Note that storage class should be assigned as constant volatile which will be allocated to flash memory or as an initialize RAM variable.



Source Block Parameters: max_Busvoltage

Constant

Output the constant specified by the 'Constant value' parameter. If 'Constant value' is a vector and 'Interpret vector parameters as 1-D' is on, treat the constant value as a 1-D array. Otherwise, output a matrix with the same dimensions as the constant value.

Main | Signal Attributes

Constant value: max_Busvoltage

[✓] Interpret vector parameters as 1-D

Sampling mode: Sample based

Sample time: inf

Model Hierarchy
- Simulink Root
  - Base Workspace
  - control_sim_v2p3_1
    - Model Workspace
    - Code for control_sim_v2p3_1
    - Advice for control_sim_v2p3_1
    - Configuration (Active)
    - ISG Motor Plan Model (Include Engir
    - ISG_CONTROLLER_PCB
      - CURRENT SHUNT MONITOR CI
      - DIGITAL_INPUT_CIRCUIT
      - Field_ShortCkt_Detection
      - HALL SENSOR SIGNAL CIRCUIT
      - HEAT SINK TEMPERATURE SENS

Contents of: Base Workspace

| Name | DataType | Value |
|---|---|---|
| hall_error | auto | |
| Field_HbrigdeTrip | auto | |
| min_Busvoltage | fixdt(0,16,2^-9,0) | 0.5 |
| max_Busvoltage | fixdt(0,16,2^-9,0) | 80 |
| PCM_MSG_BUFF | uint8 | |
| DTC_CALIB_MSG_... | uint8 | |
| ISG_STATUS_MSG... | uint8 | |
| ISG_DATA_MSG_B... | uint8 | |
| ISG_DTC_MSG__B... | uint8 | |
| T1_tick_cnt | auto | |
| RY_phase | double (auto) | 1 |
| BY_phase | double (auto) | 2 |

mpt.Parameter: max_Busvoltage

| | | | |
|---|---|---|---|
| Value: | 80 | | |
| Data type: | fixdt(0,16,2^-9,0) | | |
| Dimensions: | [1 1] | Complexity: | real |
| Minimum: | 0 | Maximum: | 120 |
| Units: | Volt | | |

Code generation options

Storage class: ConstVolatile (Custom)

Custom attributes

Header file: calib_param

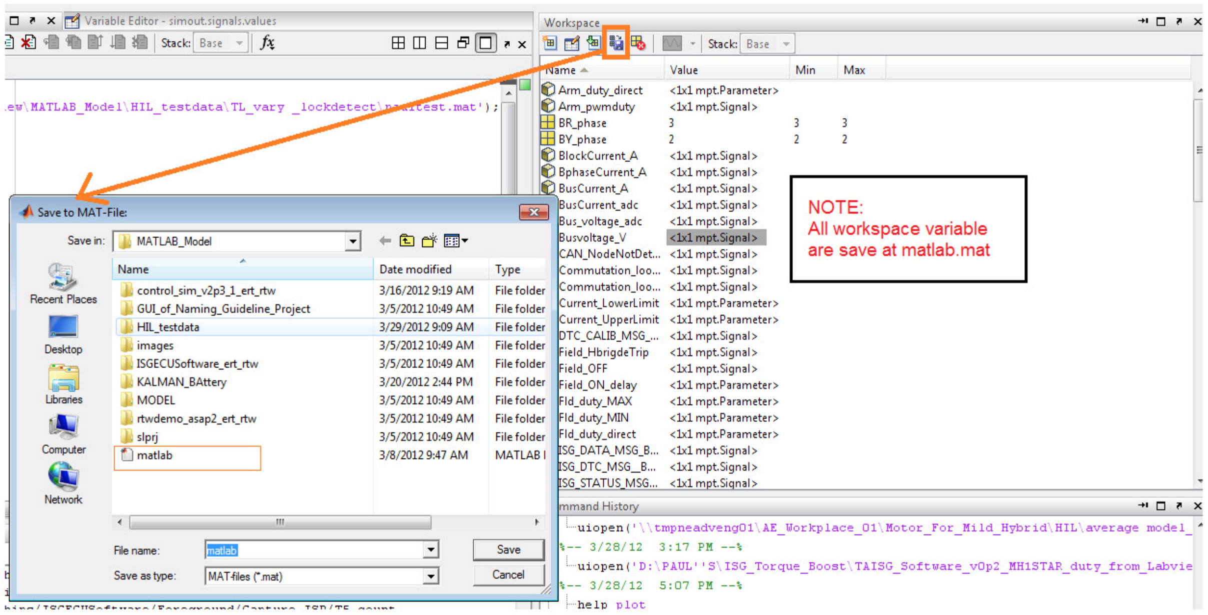
```
const volatile uint16_T Kd_byTs = 2U;
const volatile uint16_T KiTs = 5U;
const volatile uint16_T Kp = 71U;
const volatile uint16_T max_Busvoltage = 40960U;
const volatile uint16_T min_Busvoltage = 256U;
const volatile uint16_T motorspeed_limit = 6000U;

/* Start for atomic system: '<S1>/CAN_Manager_10msec'
void ISGECU_CAN_Manager_10msec_Start(void)
{
```

```
/begin CHARACTERISTIC
  /* Name                  */      max_Busvoltage
  /* Long Identifier       */      ""
  /* Type                  */      VALUE
  /* Memory Address        */      @MemoryAddress@max_Busvoltage@
  /* Record Layout         */      Scalar_UWORD
  /* Maximum Difference    */      0
  /* Conversion Method     */      COMPU_METHOD_4
  /* Lower Limit           */      0
  /* Upper Limit           */      120.0
/end CHARACTERISTIC


/begin CHARACTERISTIC
```

Figure 108: Configure Calibration parameters

### 4.2.2.4 Saving workspace variable

All variables and parameters which are created on model explorer are all located at the workspace, which are save to a specific mat file as shown below.



```
.ew\MATLAB_Model\HIL_testdata\TL_vary _lockdetect\n...test.mat');
```

| Name | Value | Min | Max |
|---|---|---|---|
| Arm_duty_direct | <1x1 mpt.Parameter> | | |
| Arm_pwmduty | <1x1 mpt.Signal> | | |
| BR_phase | 3 | 3 | 3 |
| BY_phase | 2 | 2 | 2 |
| BlockCurrent_A | <1x1 mpt.Signal> | | |
| BphaseCurrent_A | <1x1 mpt.Signal> | | |
| BusCurrent_A | <1x1 mpt.Signal> | | |
| BusCurrent_adc | <1x1 mpt.Signal> | | |
| Bus_voltage_adc | <1x1 mpt.Signal> | | |
| Busvoltage_V | <1x1 mpt.Signal> | | |
| CAN_NodeNotDet... | <1x1 mpt.Signal> | | |
| Commutation_loo... | <1x1 mpt.Signal> | | |
| Commutation_loo... | <1x1 mpt.Signal> | | |
| Current_LowerLimit | <1x1 mpt.Parameter> | | |
| Current_UpperLimit | <1x1 mpt.Parameter> | | |
| DTC_CALIB_MSG_... | <1x1 mpt.Signal> | | |
| Field_HbrigdeTrip | <1x1 mpt.Signal> | | |
| Field_OFF | <1x1 mpt.Signal> | | |
| Field_ON_delay | <1x1 mpt.Parameter> | | |
| Fld_duty_MAX | <1x1 mpt.Parameter> | | |
| Fld_duty_MIN | <1x1 mpt.Parameter> | | |
| Fld_duty_direct | <1x1 mpt.Parameter> | | |
| ISG_DATA_MSG_B... | <1x1 mpt.Signal> | | |
| ISG_DTC_MSG__B... | <1x1 mpt.Signal> | | |
| ISG_STATUS_MSG... | <1x1 mpt.Signal> | | |

NOTE:
All workspace variable are save at matlab.mat

| Name | Date modified | Type |
|---|---|---|
| control_sim_v2p3_1_ert_rtw | 3/16/2012 9:19 AM | File folder |
| GUI_of_Naming_Guideline_Project | 3/5/2012 10:49 AM | File folder |
| HIL_testdata | 3/29/2012 9:09 AM | File folder |
| images | 3/5/2012 10:49 AM | File folder |
| ISGECUSoftware_ert_rtw | 3/5/2012 10:49 AM | File folder |
| KALMAN_BAttery | 3/20/2012 2:44 PM | File folder |
| MODEL | 3/5/2012 10:49 AM | File folder |
| rtwdemo_asap2_ert_rtw | 3/5/2012 10:49 AM | File folder |
| slprj | 3/5/2012 10:49 AM | File folder |
| matlab | 3/8/2012 9:47 AM | MATLAB I |

File name: matlab
Save as type: MAT-files (*.mat)

```
uiopen('\\tmpneadveng01\AE_Workplace_01\Motor_For_Mild_Hybrid\HIL\average model_
%-- 3/28/12  3:17 PM --%
uiopen('D:\PAUL''S\ISG_Torque_Boost\TAISG_Software_v0p2_MH1STAR_duty_from_Labvie
%-- 3/28/12  5:07 PM --%
help plot
```

Figure 109: saving workspace variable

### 4.2.2.5 Configure A2L Header for 18F4431 ECU

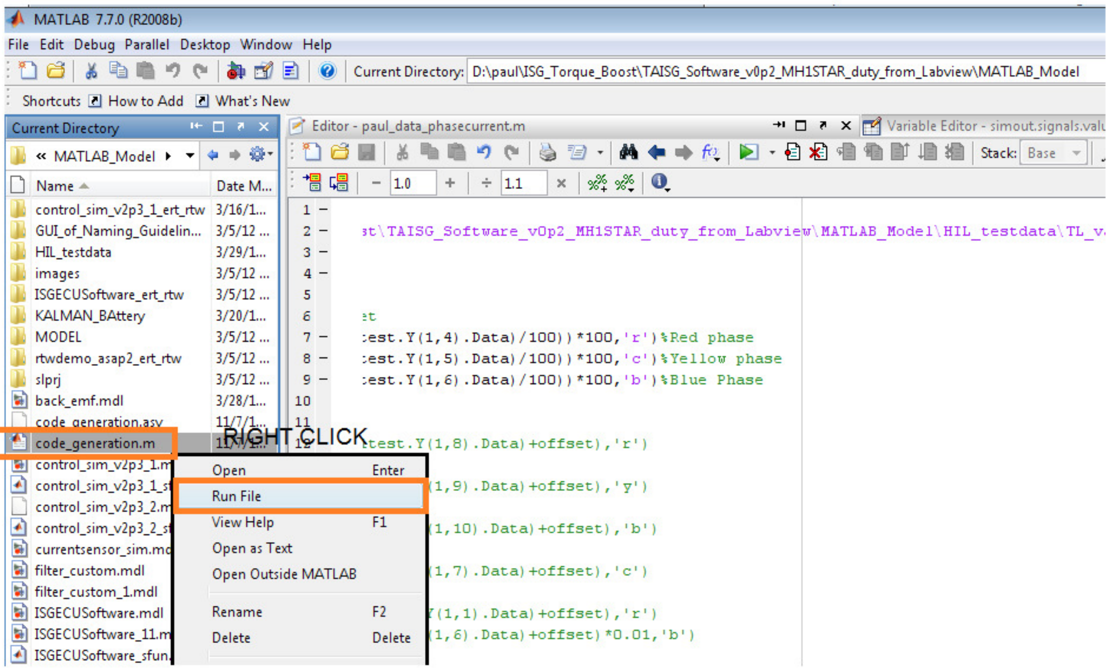
A2L header file for the ECU like CCP version, CAN message baud rate, ID of CRO and DTO, memory alignment, RAM and FLASH address range specification etc. are added or define in <asap2userlib.tlc> located at … \toolbox\rtw\targets\asap2\asap2\user in the MATLAB installation directory.

**Example:**

```
/begin TP_BLOB
/* CCP version        */ 0x201
/* Blob version       */ 0x202
/* CAN msg ID - send */ 0x7F1   /* CCP_CRO*/
/* CAN msg ID - recv */ 0x7F2   /*CCP_DTO*/
/* station address    */ 0x200
/* byte order         */ 1

/begin CAN_PARAM
      /* crystal frequency */ 0x1F4
      /* timing register 0 */ 0x41
      /* timing register 1 */ 0x25
/end CAN_PARAM
……………………………………………………………………………………
```

### 4.2.2.6 Build all for code generation

As mentioned code generation is done from a subsystem of main model. To accomplish this code generation from a specific subsystem of model with work space variable, an m-script [**Appendix I**] has been written which automate the code build process. This m-script is run as shown below.


MATLAB 7.7.0 (R2008b)
File Edit Debug Parallel Desktop Window Help
Current Directory: D:\paul\ISG_Torque_Boost\TAISG_Software_v0p2_MH1STAR_duty_from_Labview\MATLAB_Model
Shortcuts How to Add What's New
Current Directory
MATLAB_Model
Editor - paul_data_phasecurrent.m
Variable Editor - simout.signals.valu
Stack: Base
1.0
1.1
RIGHT CLICK
Open Enter
Run File
View Help F1
Open as Text
Open Outside MATLAB
Rename F2
Delete Delete

| Name | Date M... |
|---|---|
| control_sim_v2p3_1_ert_rtw | 3/16/1... |
| GUI_of_Naming_Guidelin... | 3/5/12 ... |
| HIL_testdata | 3/29/1... |
| images | 3/5/12 ... |
| ISGECUSoftware_ert_rtw | 3/5/12 ... |
| KALMAN_BAttery | 3/20/1... |
| MODEL | 3/5/12 ... |
| rtwdemo_asap2_ert_rtw | 3/5/12 ... |
| slprj | 3/5/12 ... |
| back_emf.mdl | 3/28/1... |
| code_generation.asv | 11/7/1... |
| code_generation.m | 11/7/1... |
| control_sim_v2p3_1.m | |
| control_sim_v2p3_1_s | |
| control_sim_v2p3_2.m | |
| control_sim_v2p3_2_s | |
| currentsensor_sim.md | |
| filter_custom.mdl | |
| filter_custom_1.mdl | |
| ISGECUSoftware.mdl | |
| ISGECUSoftware_11.m | |
| ISGECUSoftware_sfun | |

```
st\TAISG_Software_v0p2_MH1STAR_duty_from_Labview\MATLAB_Model\HIL_testdata\TL_v



et
:est.Y(1,4).Data)/100))*100,'r')%Red phase
:est.Y(1,5).Data)/100))*100,'c')%Yellow phase
:est.Y(1,6).Data)/100))*100,'b')%Blue Phase


test.Y(1,8).Data)+offset),'r')

(1,9).Data)+offset),'y')

(1,10).Data)+offset),'b')

(1,7).Data)+offset),'c')

Y(1,1).Data)+offset),'r')
(1,6).Data)+offset)*0.01,'b')
```

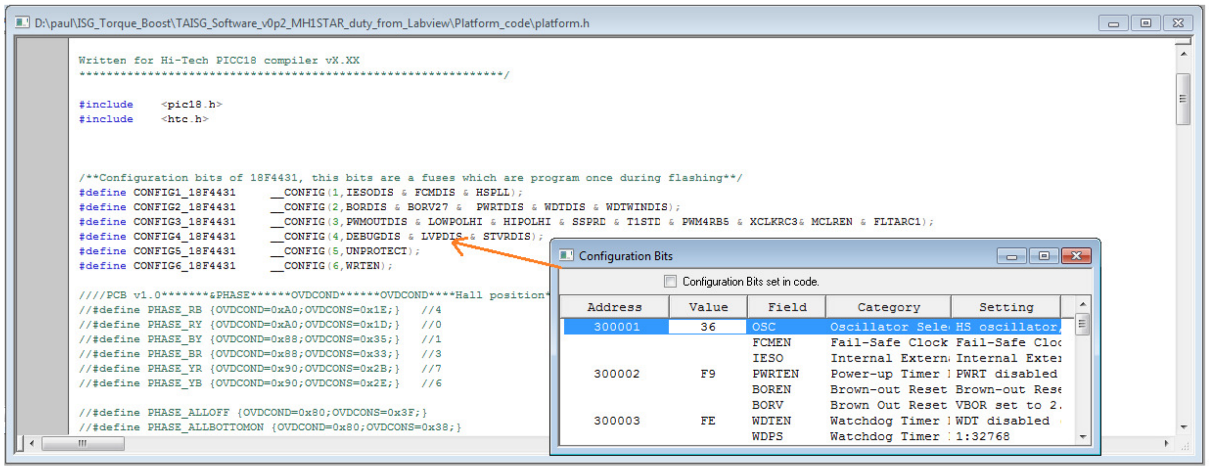
Figure 110: build using M-script

## 4.3 Integration of source code in MPLAB

All source codes, some generated from MATLAB and some hand-written code are integrated together in MPLAB IDE environment. Generated codes of RTW lies on a specific directory on MATLAB model folder refer section 1.4 above, and the platform code in specific location. As the main MPLAB project lies in the main directory, all codes are link from here. Hi-tech C STD compiler is used to compiled and linked, and then generated executable hex and map file for further use in the tool chain. Hex file will be dump to microcontroller and map file will be used for complete A2L file generation shown in the later section of this report.

### 4.3.1 Configuration fuses of microchip 18F4431

There is specific fuse setting to be done during flashing the code to microchip product, those fuses setting can be coded by using Hi-tech C compiler directives as shown below


D:\paul\ISG_Torque_Boost\TAISG_Software_v0p2_MH1STAR_duty_from_Labview\Platform_code\platform.h
Configuration Bits
Configuration Bits set in code.

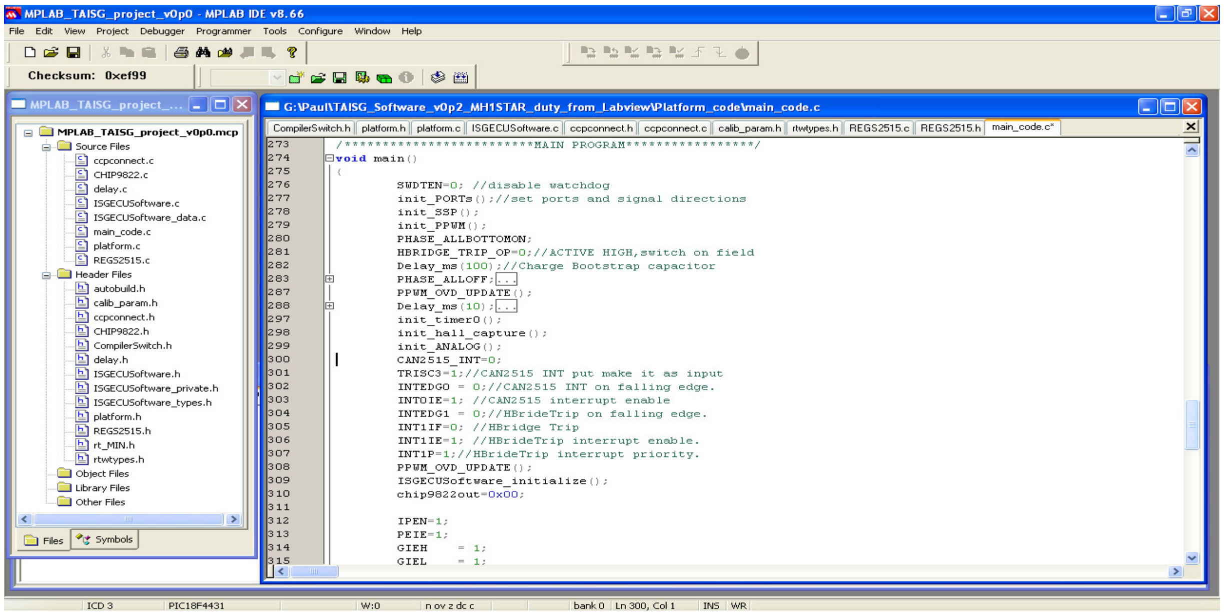
```
Written for Hi-Tech PICC18 compiler vX.XX
****************************************************************/

#include    <pic18.h>
#include    <htc.h>



/**Configuration bits of 18F4431, this bits are a fuses which are program once during flashing**/
#define CONFIG1_18F4431     __CONFIG(1,IESODIS & FCMDIS & HSPLL);
#define CONFIG2_18F4431     __CONFIG(2,BORDIS & BORV27 &  PWRTDIS & WDTDIS & WDTWINDIS);
#define CONFIG3_18F4431     __CONFIG(3,PWMOUTDIS & LOWPOLHI & HIPOLHI & SSPRD & T1STD & PWM4RB5 & XCLKRC3& MCLREN & FLTARC1);
#define CONFIG4_18F4431     __CONFIG(4,DEBUGDIS & LVPDIS & STVRDIS);
#define CONFIG5_18F4431     __CONFIG(5,UNPROTECT);
#define CONFIG6_18F4431     __CONFIG(6,WRTEN);

////PCB v1.0*******&PHASE******OVDCOND******OVDCOND****Hall position*
//#define PHASE_RB {OVDCOND=0xA0;OVDCONS=0x1E;}   //4
//#define PHASE_RY {OVDCOND=0xA0;OVDCONS=0x1D;}   //0
//#define PHASE_BY {OVDCOND=0x88;OVDCONS=0x35;}   //1
//#define PHASE_BR {OVDCOND=0x88;OVDCONS=0x33;}   //3
//#define PHASE_YR {OVDCOND=0x90;OVDCONS=0x2B;}   //7
//#define PHASE_YB {OVDCOND=0x90;OVDCONS=0x2E;}   //6

//#define PHASE_ALLOFF {OVDCOND=0x80;OVDCONS=0x3F;}
//#define PHASE_ALLBOTTOMON {OVDCOND=0x80;OVDCONS=0x38;}
```

| Address | Value | Field | Category | Setting |
|---|---|---|---|---|
| 300001 | 36 | OSC | Oscillator Sele | HS oscillator, |
| | | FCMEN | Fail-Safe Clock | Fail-Safe Cloc |
| | | IESO | Internal Extern | Internal Exte |
| 300002 | F9 | PWRTEN | Power-up Timer | PWRT disabled |
| | | BOREN | Brown-out Reset | Brown-out Rese |
| | | BORV | Brown Out Reset | VBOR set to 2. |
| 300003 | FE | WDTEN | Watchdog Timer | WDT disabled |
| | | WDPS | Watchdog Timer | 1:32768 |

Figure 111: fuse setting of 18F

#### 4.3.1.1 Initialization code

All initialization function of MCU peripherals are done at the starting of the main as the usual C coding methodology as shown below.

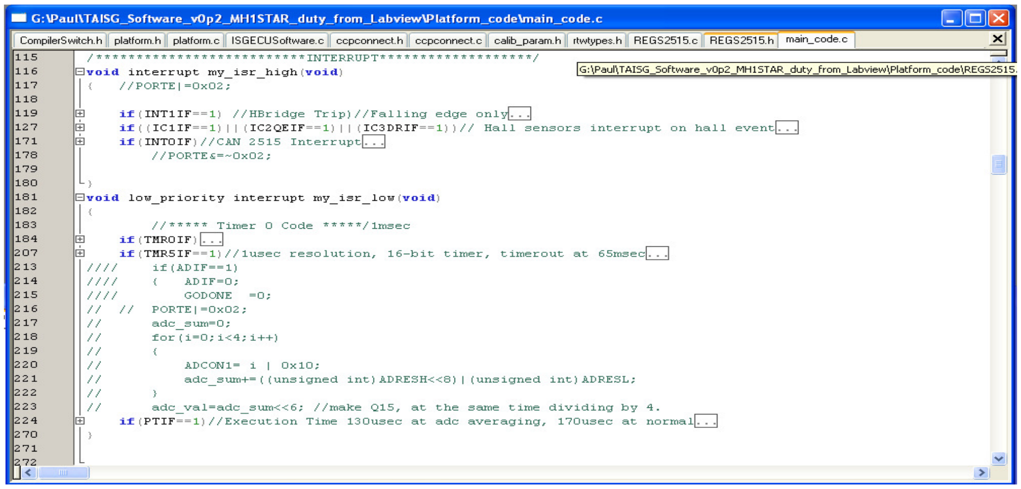


```
/************************MAIN PROGRAM*****************/
void main()
{
        SWDTEN=0; //disable watchdog
        init_PORTs();//set ports and signal directions
        init_SSP();
        init_PPWM();
        PHASE_ALLBOTTOMON;
        HBRIDGE_TRIP_OP=0;//ACTIVE HIGH,switch on field
        Delay_ms(100);//Charge Bootstrap capacitor
        PHASE_ALLOFF;...
        PPWM_OVD_UPDATE();
        Delay_ms(10);...
        init_timer0();
        init_hall_capture();
        init_ANALOG();
        CAN2515_INT=0;
        TRISC3=1;//CAN2515 INT put make it as input
        INTEDG0 = 0;//CAN2515 INT on falling edge.
        INT0IE=1; //CAN2515 interrupt enable
        INTEDG1 = 0;//HBrideTrip on falling edge.
        INT1IF=0; //HBridge Trip
        INT1IE=1; //HBrideTrip interrupt enable.
        INT1P=1;//HBrideTrip interrupt priority.
        PPWM_OVD_UPDATE();
        ISGECUSoftware_initialize();
        chip9822out=0x00;

        IPEN=1;
        PEIE=1;
        GIEH    = 1;
        GIEL    = 1;
```

Figure 112: initialization of peripherals

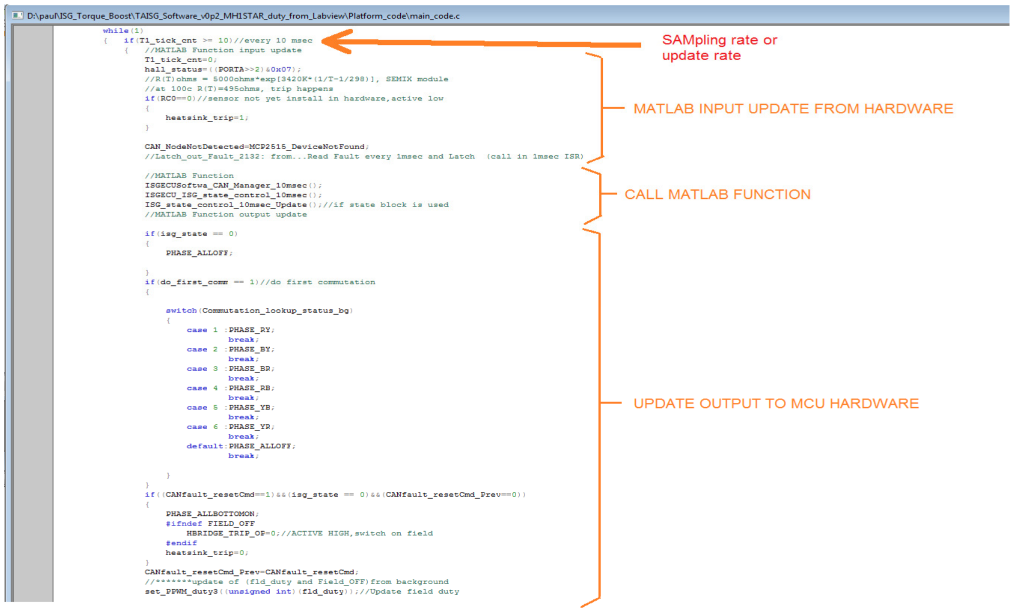
### 4.3.2 Interrupt code – integration

The Two interrupt priorities- low and high priority functions are shown below. All interrupts are pre-configured and interrupt flag are checked, and depending on the interrupt event the specific subroutine code will be executed. The subsystem model in foreground of Simulink model which has been generated as a one C function, will be called inside this flag (not seen - code folding ON). So, inside this the input to the subsystem will be updated and then function will be called and then out will be updated as shown in figure **Figure 114: Integration of input and output of subsystem** below.


G:\Paul\TAISG_Software_v0p2_MH1STAR_duty_from_Labview\Platform_code\main_code.c
CompilerSwitch.h
platform.h
platform.c
ISGECUSoftware.c
ccpconnect.h
ccpconnect.c
calib_param.h
rtwtypes.h
REGS2515.c
REGS2515.h
main_code.c
G:\Paul\TAISG_Software_v0p2_MH1STAR_duty_from_Labview\Platform_code\REGS2515.

```
/**************************INTERRUPT******************/
void interrupt my_isr_high(void)
{    //PORTE|=0x02;

     if(INT1IF==1) //HBridge Trip)//Falling edge only...
     if((IC1IF==1)||(IC2QEIF==1)||(IC3DRIF==1))// Hall sensors interrupt on hall event...
     if(INT0IF)//CAN 2515 Interrupt...
          //PORTE&=~0x02;

}
void low_priority interrupt my_isr_low(void)
{
          //***** Timer 0 Code *****/1msec
     if(TMR0IF)...
     if(TMR5IF==1)//1usec resolution, 16-bit timer, timerout at 65msec...
////      if(ADIF==1)
////      {    ADIF=0;
////           GODONE  =0;
//   //   PORTE|=0x02;
//        adc_sum=0;
//        for(i=0;i<4;i++)
//        {
//             ADCON1= i | 0x10;
//             adc_sum+=((unsigned int)ADRESH<<8)|(unsigned int)ADRESL;
//        }
//        adc_val=adc_sum<<6; //make Q15, at the same time dividing by 4.
     if(PTIF==1)//Execution Time 130usec at adc averaging, 170usec at normal...
}
```

Figure 113: ISR subsystems integration

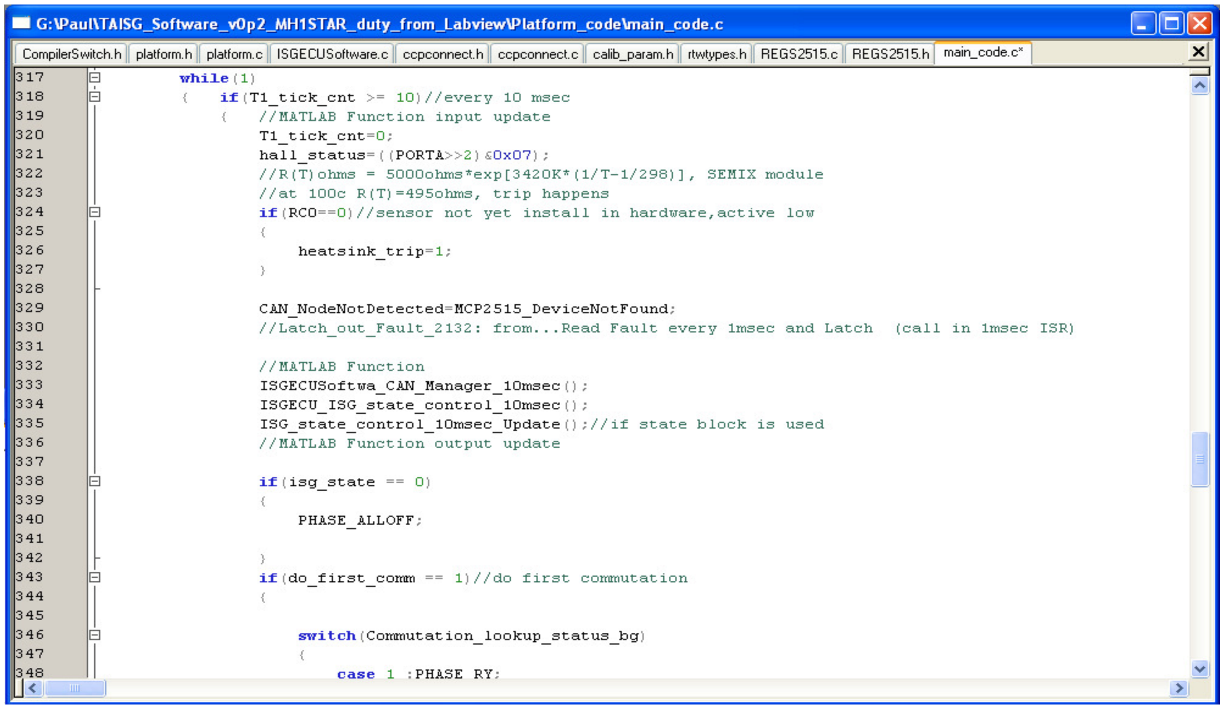

D:\paul\ISG_Torque_Boost\TAISG_Software_v0p2_MH1STAR_duty_from_Labview\Platform_code\main_code.c
SAMpling rate or
update rate
MATLAB INPUT UPDATE FROM HARDWARE
CALL MATLAB FUNCTION
UPDATE OUTPUT TO MCU HARDWARE

```
while(1)
{   if(T1_tick_cnt >= 10)//every 10 msec
    {   //MATLAB Function input update
        T1_tick_cnt=0;
        hall_status=((PORTA>>2)&0x07);
        //R(T)ohms = 5000ohms*exp[3420K*(1/T-1/298)], SEMIX module
        //at 100c R(T)=495ohms, trip happens
        if(RC0==0)//sensor not yet install in hardware,active low
        {
            heatsink_trip=1;
        }

        CAN_NodeNotDetected=MCP2515_DeviceNotFound;
        //Latch_out_Fault_2132: from...Read Fault every 1msec and Latch  (call in 1msec ISR)

        //MATLAB Function
        ISGECUSoftwa_CAN_Manager_10msec();
        ISGECU_ISG_state_control_10msec();
        ISG_state_control_10msec_Update();//if state block is used
        //MATLAB Function output update

        if(isg_state == 0)
        {
            PHASE_ALLOFF;

        }
        if(do_first_comm == 1)//do first commutation
        {

            switch(Commutation_lookup_status_bg)
            {
                case 1 :PHASE_RY;
                        break;
                case 2 :PHASE_BY;
                        break;
                case 3 :PHASE_BR;
                        break;
                case 4 :PHASE_RB;
                        break;
                case 5 :PHASE_YB;
                        break;
                case 6 :PHASE_YR;
                        break;
                default:PHASE_ALLOFF;
                        break;

            }
        }
        if((CANfault_resetCmd==1)&&(isg_state == 0)&&(CANfault_resetCmd_Prev==0))
        {
            PHASE_ALLBOTTOMON;
            #ifndef FIELD_OFF
                HBRIDGE_TRIP_OP=0;//ACTIVE HIGH,switch on field
            #endif
            heatsink_trip=0;
        }
        CANfault_resetCmd_Prev=CANfault_resetCmd;
        //*******update of (fld_duty and Field_OFF)from background
        set_PPWM_duty3((unsigned int)(fld_duty));//Update field duty
```

Figure 114: Integration of input and output of subsystem

### 4.3.3 Background code - integration

The while loop routine in C main file is directly the background subsystem in our MATLAB model, so integration of the background subsystem in MPLAB model is done by calling the C functions of the background subsystem in the while loop of C main file as shown below. All input subsystem are update from hardware peripheral of the uC and then function are called, and then output of subsystem are again updated to the peripheral hardware of the uC.



G:\Paul\TAISG_Software_v0p2_MH1STAR_duty_from_Labview\Platform_code\main_code.c

CompilerSwitch.h | platform.h | platform.c | ISGECUSoftware.c | ccpconnect.h | ccpconnect.c | calib_param.h | rtwtypes.h | REGS2515.c | REGS2515.h | main_code.c*

```
        while(1)
        {   if(T1_tick_cnt >= 10)//every 10 msec
            {   //MATLAB Function input update
                T1_tick_cnt=0;
                hall_status=((PORTA>>2)&0x07);
                //R(T)ohms = 5000ohms*exp[3420K*(1/T-1/298)], SEMIX module
                //at 100c R(T)=495ohms, trip happens
                if(RC0==0)//sensor not yet install in hardware,active low
                {
                    heatsink_trip=1;
                }

                CAN_NodeNotDetected=MCP2515_DeviceNotFound;
                //Latch_out_Fault_2132: from...Read Fault every 1msec and Latch  (call in 1msec ISR)

                //MATLAB Function
                ISGECUSoftwa_CAN_Manager_10msec();
                ISGECU_ISG_state_control_10msec();
                ISG_state_control_10msec_Update();//if state block is used
                //MATLAB Function output update

                if(isg_state == 0)
                {
                    PHASE_ALLOFF;

                }
                if(do_first_comm == 1)//do first commutation
                {

                    switch(Commutation_lookup_status_bg)
                    {
                        case 1 :PHASE RY;
```

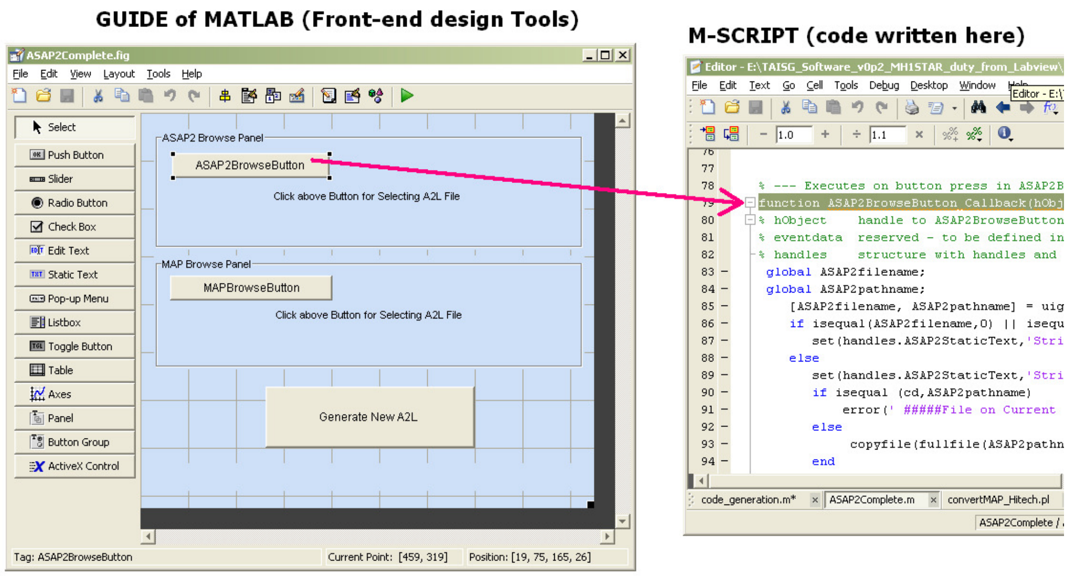
Figure 115: background code integration

### 4.3.4 Building in MPLAB and Flash to ECU

Building of the whole code is done in MPLAB and the hex code is flash using the ICD3 as shown in the tool chain diagram above. Note that ASAP2 driver in ECU is implement along with this source code which is called the CAN Calibration protocol (CCP). The ready-made driver code has been plug-in to the build process with slight modification. Flashing through CAN is not included with the implemented CCP protocol.

## 4.4 Complete A2L file generation

For generating complete A2L file, *asap2Complete.m* [**Appendix II**] file should be run, but this script required a specific format of map file for resolving address token in the A2L file generated on RTW build. So, this specific map file format is generated from original map file using Perl script [**Appendix III**]. To automate the whole process a GUI is design using GUIDE (graphical user interface development environment) as shown below.


GUIDE of MATLAB (Front-end design Tools)
M-SCRIPT (code written here)
ASAP2Complete.fig
File Edit View Layout Tools Help
Select
Push Button
Slider
Radio Button
Check Box
Edit Text
Static Text
Pop-up Menu
Listbox
Toggle Button
Table
Axes
Panel
Button Group
ActiveX Control
ASAP2 Browse Panel
ASAP2BrowseButton
Click above Button for Selecting A2L File
MAP Browse Panel
MAPBrowseButton
Click above Button for Selecting A2L File
Generate New A2L
Tag: ASAP2BrowseButton
Current Point: [459, 319]
Position: [19, 75, 165, 26]
Editor - E:\TAISG_Software_v0p2_MH1STAR_duty_from_Labview\
File Edit Text Go Cell Tools Debug Desktop Window Help
code_generation.m*
ASAP2Complete.m
convertMAP_Hitech.pl
ASAP2Complete /

```
% --- Executes on button press in ASAP2B
function ASAP2BrowseButton_Callback(hObj
% hObject    handle to ASAP2BrowseButton
% eventdata  reserved - to be defined in
% handles    structure with handles and
 global ASAP2filename;
 global ASAP2pathname;
    [ASAP2filename, ASAP2pathname] = uig
    if isequal(ASAP2filename,0) || isequ
       set(handles.ASAP2StaticText,'Stri
    else
       set(handles.ASAP2StaticText,'Stri
       if isequal (cd,ASAP2pathname)
           error(' #####File on Current
       else
            copyfile(fullfile(ASAP2pathn
       end
```

Figure 116: ASAP2 complete GUIDE development

When the script runs, it prompt for file browser- A2L, map, and hex file as shown. Once all files are given, a complete A2L file will be generated by pressing generate button.

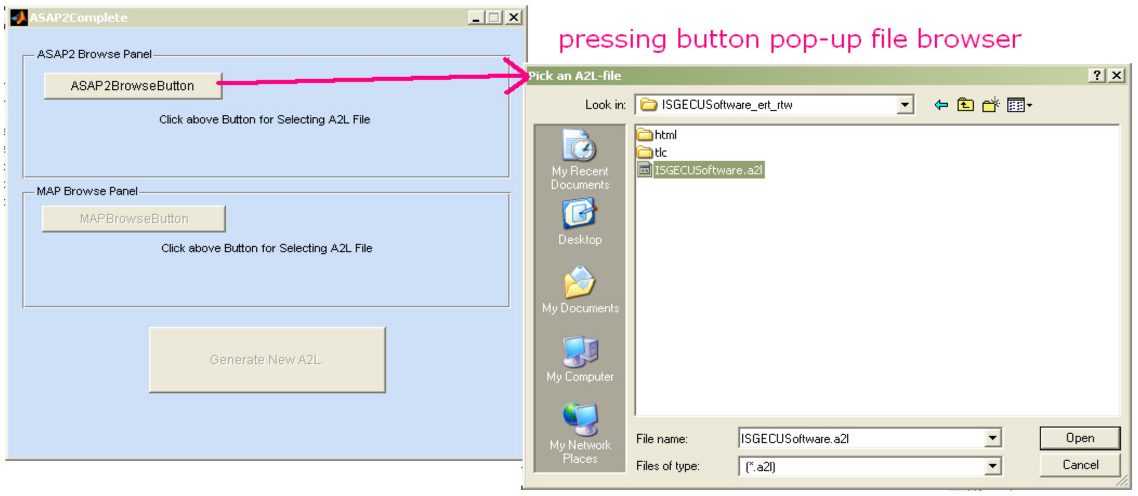


**Figure 117: ASAP2 complete GUI when run**

## 4.5 Run-time control

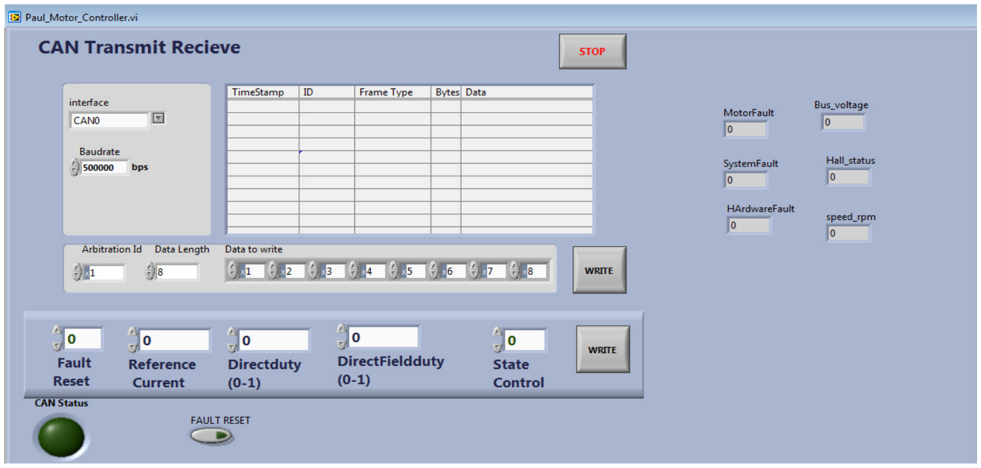
Control, measure and calibrate are the basic important word which comes in development of any complex control software. Here we will discuss how the control is implemented for HIL testing, and also how measurement and calibration of ECU are done.

### 4.5.1 Measurement and calibration

INCA tool or Labview M&C[1] tool can be used as a tool for measurement and calibration through CAN interface using CCP protocol. The tools offer a wide variety of functions including precalibration of function models on the PC, ECU flash programming, measurement data analysis, calibration data management, and automated optimization of ECU parameters. The generated calibration and measurement data can be processed and evaluated continuously. PID controller parameter tuning has been done extensively using this tool.

### 4.5.2 Hybrid ECU simulating Node

GUI is designed in Labview as shown below for a manual CAN command message sending node like a Hybrid ECU in vehicle. So an interface connection using a CAN-USB device is used as shown in tool chain figure above.



interface CAN0
Baudrate 500000 bps
Arbitration Id 1
Data Length 8
Data to write 1 2 3 4 5 6 7 8
Fault Reset 0
Reference Current 0
Directduty (0-1) 0
DirectFieldduty (0-1) 0
State Control 0
MotorFault 0
Bus_voltage 0
SystemFault 0
Hall_status 0
HArdwareFault 0
speed_rpm 0

Figure 118: Hybrid ECU simulator

[1] Labview measurement and Calibration tool

# 5 CHAPTER 5: HARDWARE IN LOOP SIMULATION

## 5.1 Introduction and challenges in HIL testing

Hardware-in-the-loop (HIL) simulation is a technique that is used in the development and test of complex real-time embedded systems. HIL simulation provides an effective platform by adding the complexity of the plant under control to the test platform. The complexity of the plant under control is included in test and development by adding a mathematical representation of all related dynamic systems. These mathematical representations are referred to as the "plant simulation". The embedded system to be tested interacts with this plant simulation.

A HIL simulation must include electrical emulation of sensors and actuators. These electrical emulations act as the interface between the plant simulation and the embedded system under test. The value of each electrically emulated sensor is controlled by the plant simulation and is read by the embedded system under test (feedback). Likewise, the embedded system under test implements its control algorithms by outputting actuator control signals. Changes in the control signals result in changes to variable values in the plant simulation.

In many cases, the most effective way to develop an embedded system is to connect the embedded system to the real plant. In other cases, HIL simulation is more efficient. The metric of development and test efficiency is typically a formula that includes the following factors: 1. Cost, 2. Duration, 3. Safety, 4. Feasibility

Cost of the approach will be a measure of the cost of all tools and effort. The duration of development and test affects the time-to-market for a planned product. The safety factor and duration are typically equated to a cost measure. Specific conditions that warrant the use of HIL simulation include the following:

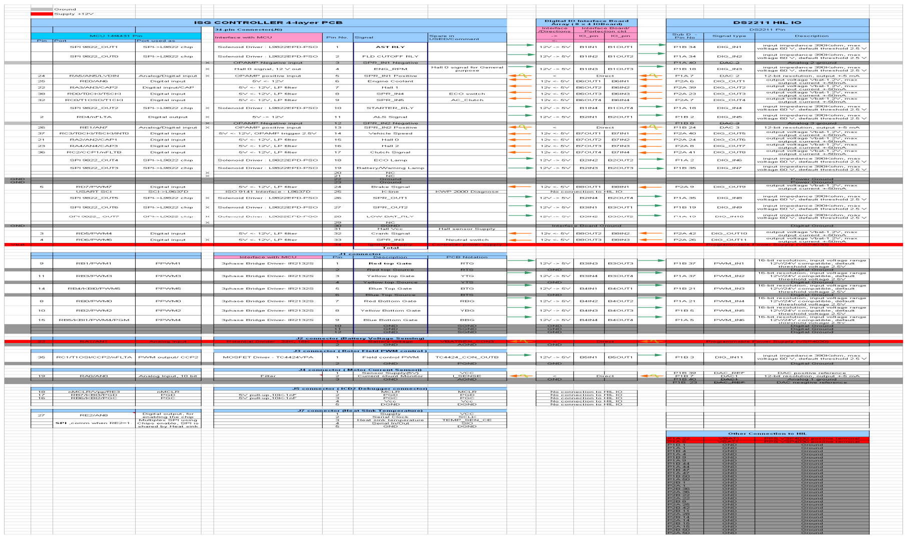
- Enhancing the quality of Testing
- Tight development schedules
- High-burden-rate plant

- Early process human factors development

In hardware-in-the-loop (HIL) testing, the designer can verify the production system controller by simulating the real-time behavior and characteristics of the final system without the physical hardware or operational environment. While the system is simulated in real-time on a test computer the control code can be run on the target controller hardware. Though it is possible to connect the target hardware with the actual motor, testing against a simulated motor offers several advantages. When compared to a physical plant, a desktop simulator, often called a hardware-in-the-loop (HIL) tester, is far more cost-efficient, and easier to reproduce. The simulated motor also can simulate a variety of operating conditions or even fault conditions, such as engine stall, that would be difficult, costly, and/or dangerous with the actual plant. If measured data from HIL simulation deviates from Model-in-the-Loop simulation, the most likely cause is a bug in the target compiler or a problem with the processor.

## 5.2 dSPACE System setup

dSPACE system is used for setting-up hardware in loop simulation. All the signals interface requirement are drawn on excel spread sheet as shown in figure below. Pin assignment for each of the signal on controller PCB and dSPACE systems are made. Note that this HIL is a signal level HIL where high current or high power is not involved.



| | |
|---|---|
| ▨ | Ground |
| 🟥 | Supply +12V |

| | ISG CONTROLLER 4-layer PCB | | | | | | | | Digital IO Interface Board Array ( 8 x 4 IOBoard) | | | DS2211 HIL IO | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | 34-pin Connector(J6) | | | | Interface /Directions | Interface Board/ Portection ckt | | DS2211 Pin | | |
| | MCU 14f4431 Pin | | | | Interface with MCU | Pin No. | Signal | Spare in USED/Comment | -> <- | IO_pin | IO_pin | Sub D - Pin No | Signal type | Description |
| | Pin | Port | Port used as | | | | | | | | | | | |
| | | SPI 9822_OUT1 | SPI->L9822 chip | | Solenoid Driver : L9822EPD-PSO | 1 | AST RLY | | 12V -> 5V | B1IN1 | B1OUT1 | P1B 34 | DIG_IN1 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | | SPI 9822_OUT0 | SPI->L9822 chip | | Solenoid Driver : L9822EPD-PSO | 2 | FLD CUTOFF RLY | | 12V -> 5V | B1IN2 | B1OUT2 | P1A 34 | DIG_IN2 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | | | | X | OPAMP Negative input | 3 | SPR_IN1 Negative | | | | | P1A 40 | DAC 2 | Analog 2 ground |
| | | | | X | Hall 0 signal, 12 V out | 4 | ENG_RPM | Hall 0 signal for General purpose | 12V -> 5V | B1IN3 | B1OUT3 | P1B 18 | DIG_IN3 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | 24 | RA5/AN5/LVDIN | Analog/Digital input | X | OPAMP positive input | 5 | SPR_IN1 Positive | | < | Direct | | P1A 7 | DAC 2 | 12-bit resolution, output +-5 mA |
| | 25 | RE0/AN6 | Digital input | | 5V <- 12V | 6 | Engine Coolant | | 12v <- 5V | B6OUT1 | B6IN1 | P2A 6 | DIG_OUT1 | output voltage Vbat-1.2V, max output current +-50mA |
| | 22 | RA3/AN3/CAP2 | Digital input/CAP | | 5V <- 12V, LP filter | 7 | Hall 1 | | 12v <- 5V | B6OUT2 | B6IN2 | P2A 39 | DIG_OUT2 | output voltage Vbat-1.2V, max output current +-50mA |
| | 38 | RD0/T0CKI/T5CKI | Digital input | | 5V <- 12V, LP filter | 8 | SPR_IN4 | ECO switch | 12v <- 5V | B6OUT3 | B6IN3 | P2A 23 | DIG_OUT3 | output voltage Vbat-1.2V, max output current +-50mA |
| | 32 | RC0/T1OSO/T1CKI | Digital input | | 5V <- 12V, LP filter | 9 | SPR_IN5 | AC_Clutch | 12v <- 5V | B6OUT4 | B6IN4 | P2A 7 | DIG_OUT4 | output voltage Vbat-1.2V, max output current +-50mA |
| | | SPI 9822_OUT2 | SPI->L9822 chip | X | Solenoid Driver : L9822EPD-PSO | 10 | STARTER_RLY | | 12V -> 5V | B1IN4 | B1OUT4 | P1A 16 | DIG_IN4 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | 2 | RD4/nFLTA | Digital output | X | 5V -> 12V | 11 | ALS Signal | | 12V -> 5V | B2IN1 | B2OUT1 | P1B 2 | DIG_IN5 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | | | | X | OPAMP Negative input | 12 | SPR_IN2 Negative | | | | | P1B 8 | DAC 3 | Analog 3 ground |
| | 26 | RE1/AN7 | Analog/Digital input | X | OPAMP positive input | 13 | SPR_IN2 Positive | | < | Direct | | P1B 24 | DAC 3 | 12-bit resolution, output +-5 mA |
| | 37 | RC3/T0CKI/T5CKI/INT0 | Digital input | | 5V <- 12V, OPAMP trigger 2.5V | 14 | Vehicle Speed | | 12v <- 5V | B7OUT1 | B7IN1 | P2A 40 | DIG_OUT5 | output voltage Vbat-1.2V, max output current +-50mA |
| | 21 | RA2/AN2/CAP1 | Digital input | | 5V <- 12V, LP filter | 15 | Hall 0 | | 12v <- 5V | B7OUT2 | B7IN2 | P2A 24 | DIG_OUT6 | output voltage Vbat-1.2V, max output current +-50mA |
| | 23 | RA4/AN4/CAP3 | Digital input | | 5V <- 12V, LP filter | 16 | Hall 2 | | 12v <- 5V | B7OUT3 | B7IN3 | P2A 8 | DIG_OUT7 | output voltage Vbat-1.2V, max output current +-50mA |
| | 36 | RC2/CCP1/nFLTB | Digital input | | 5V <- 12V, LP filter | 17 | Clutch Signal | | 12v <- 5V | B7OUT4 | B7IN4 | P2A 41 | DIG_OUT8 | output voltage Vbat-1.2V, max output current +-50mA |
| | | SPI 9822_OUT4 | SPI->L9822 chip | | Solenoid Driver : L9822EPD-PSO | 18 | ECO Lamp | | 12V -> 5V | B2IN2 | B2OUT2 | P1A 2 | DIG_IN6 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | | SPI 9822_OUT3 | SPI->L9822 chip | | Solenoid Driver : L9822EPD-PSO | 19 | Battery/Warning Lamp | | 12V -> 5V | B2IN3 | B2OUT3 | P1B 35 | DIG_IN7 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | | | | X | | 20 | NC | | | | | | | |
| | | | | X | | 21 | NC | | | | | | | |
| GND | | | | | | 22 | Ground | | | | | | | Power Ground |
| GND | | | | | | 23 | Ground | | | | | | | Power Ground |
| | 5 | RD7/PWM7 | Digital input | | 5V <- 12V, LP filter | 24 | Brake Signal | | 12v <- 5V | B8OUT1 | B8IN1 | P2A 9 | DIG_OUT9 | output voltage Vbat-1.2V, max output current +-50mA |
| | | USART SCI | SCI->L9637D | | ISO 9141 Interface - L9637D | 25 | K-line | KWP 2000 Diagnose | No connection to HIL IO | | | | | |
| | | SPI 9822_OUT5 | SPI->L9822 chip | X | Solenoid Driver : L9822EPD-PSO | 26 | SPR_OUT1 | | 12V -> 5V | B2IN4 | B2OUT4 | P1A 35 | DIG_IN8 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | | SPI 9822_OUT6 | SPI->L9822 chip | X | Solenoid Driver : L9822EPD-PSO | 27 | SPR_OUT2 | | 12V -> 5V | B3IN1 | B3OUT1 | P1B 19 | DIG_IN9 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | | SPI 9822_ OUT7 | SPI->L9822 chip | X | Solenoid Driver : L9822EPD-PSO | 28 | LOW BAT_RLY | | 12V -> 5V | B3IN2 | B3OUT2 | P1A 19 | DIG_IN10 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | | | | X | | 29 | NC | | | | | | | |
| GND | | | | | | 30 | SGND | | Interface Board Ground | | | | | Digital Ground |
| | | | | | | 31 | Hall Vcc | Hall sensor Supply | | | | | | |
| | 3 | RD5/PWM4 | Digital input | | 5V <- 12V, LP filter | 32 | Crank Signal | | 12v <- 5V | B8OUT2 | B8IN2 | P2A 42 | DIG_OUT10 | output voltage Vbat-1.2V, max output current +-50mA |
| | 4 | RD6/PWM6 | Digital input | X | 5V <- 12V, LP filter | 33 | SPR_IN3 | Neutral switch | 12v <- 5V | B8OUT3 | B8IN3 | P2A 26 | DIG_OUT11 | output voltage Vbat-1.2V, max output current +-50mA |
| Vbat | | | | | | 34 | Ignition Supply | Controller power supply | | | | | Programmable Power Supply (VSP4030) | |
| | | | | | | | Total | | | | | | | |
| | | | | | J1 connector | | | | | | | | | |
| | | | | | Interface with MCU | Pin | Description | PCB Notation | | | | | | |
| | 9 | RB1/PWM1 | PPWM1 | | 3phase Bridge Driver- IR2132S | 1 | Red top Gate | RTG | 12V -> 5V | B3IN3 | B3OUT3 | P1B 37 | PWM_IN1 | 16-bit resolution, input voltage range 12V/24V compatible, default threshold voltage 2.5V |
| | | | | | | 2 | Red top Source | RTS | GND | | | | | Digital Ground |
| | 11 | RB3/PWM3 | PPWM3 | | 3phase Bridge Driver- IR2132S | 3 | Yellow top Gate | YTG | 12V -> 5V | B3IN4 | B3OUT4 | P1A 37 | PWM_IN2 | 16-bit resolution, input voltage range 12V/24V compatible, default threshold voltage 2.5V |
| | | | | | | 4 | Yellow top Source | YTS | GND | | | | | Digital Ground |
| | 14 | RB4/KBI0/PWM5 | PPWM5 | | 3phase Bridge Driver- IR2132S | 5 | Blue Top Gate | BTG | 12V -> 5V | B4IN1 | B4OUT1 | P1B 21 | PWM_IN3 | 16-bit resolution, input voltage range 12V/24V compatible, default threshold voltage 2.5V |
| | | | | | | 6 | Blue Top Source | BTS | GND | | | | | Digital Ground |
| | 8 | RB0/PWM0 | PPWM0 | | 3phase Bridge Driver- IR2132S | 7 | Red Bottom Gate | RBG | 12V -> 5V | B4IN2 | B4OUT2 | P1A 21 | PWM_IN4 | 16-bit resolution, input voltage range 12V/24V compatible, default threshold voltage 2.5V |
| | 10 | RB2/PWM2 | PPWM2 | | 3phase Bridge Driver- IR2132S | 8 | Yellow Bottom Gate | YBG | 12V -> 5V | B4IN3 | B4OUT3 | P1B 5 | PWM_IN5 | 16-bit resolution, input voltage range 12V/24V compatible, default threshold voltage 2.5V |
| | 15 | RB5/KBI1/PWM4/PGM | PPWM4 | | 3phase Bridge Driver- IR2132S | 9 | Blue Bottom Gate | BBG | 12V -> 5V | B4IN4 | B4OUT4 | P1A 5 | PWM_IN6 | 16-bit resolution, input voltage range 12V/24V compatible, default threshold voltage 2.5V |
| | | | | | | 10 | GND | SGND | GND | | | | | Digital Ground |
| | | | | | | 11 | GND | SGND | GND | | | | | Digital Ground |
| | | | | | | 12 | GND | SGND | GND | | | | | Digital Ground |
| | | | | | J2 connector (Battery Voltage Sensing) | | | | | | | | | |
| | 22 | RA1/AN1 | Analog Input | | Potential Divider - 33K, 15K | 1 | Battery voltage | VBATSEN_CON3 | < | Direct | | | Programmable Power Supply (VSP4030) | |
| | | | | | | 2 | GND | AGND | GND | | | | | |
| | | | | | J3 connector ( Rotor Field PWM control ) | | | | | | | | | |
| | 36 | RC1/T1OSI/CCP2/nFLTA | PWM output/ CCP2 | | MOSFET Driver - TC4424VPA | 1 | Field control PWM | TC4424_CON_OUTB | 12V -> 5V | B5IN1 | B5OUT1 | P1B 3 | DIG_IN11 | input impedance 390Kohm, max voltage 60 V, default threshold 2.5 V |
| | | | | | | 2 | GND | PGND | GND | | | | | Digital Ground |
| | | | | | J4 connector ( Motor Current Sensor)) | | | | | | | | | |
| | | | | | | 1 | Sensor Supply(5V) | VCC | | | | P1B 39 | DAC_REF | DAC positive reference |
| | 19 | RA0/AN0 | Analog Input, 10 bit | | Filter | 2 | Current shunt Monitor | I_SENSE | < | Direct | | P1B 7 | DAC1 | 12-bit resolution, output +-5 mA |
| | | | | | | 3 | GND | AGND | GND | | | P1B 40 | DAC1 | Analog 1 ground |
| | | | | | | | | | | | | P1B 23 | DAC_REF | DAC neagtive reference |
| | | | | | J5 connector ( ICD2 Debugger connector) | | | | | | | | | |
| | 18 | nMCLR/Vpp/RE3 | nMCLR | | | 1 | MCLR | MCLR | No connection to HIL IO | | | | | |
| | 17 | RB7/KBI3/PGD | PGD | | 5V pull-up,10K-1nF | 2 | PGD | PGD | No connection to HIL IO | | | | | |
| | 16 | RB6/KBI2/PGC | PGC | | 5V pull-up,10K-1nF | 3 | PGC | PGC | No connection to HIL IO | | | | | |
| | | | | | | 4 | Vcc | Vcc | No connection to HIL IO | | | | | |
| | | | | | | 5 | DGND | DGND | No connection to HIL IO | | | | | |
| | | | | | J7 connector (Heat Sink Temperature) | | | | | | | | | |
| | 27 | RE2/AN8 | Digital output, for enabling the chip | | | 1 | Supply | VCC | | | | | | |
| | | | | | | 2 | Serial Clock | SCLK | | | | | | |
| | | SPI ,comm when RE2=1. | Multiplex SPI using Chips enable, SPI is shared by Heat sink | | | 3 | Heat sink temperature | TEMP_SEN_CE | | | | | | |
| | | | | | | 4 | Serial In/Out | SIO | | | | | | |
| | | | | | | 5 | GND | DGND | | | | | | |

| Other Connection to HIL | | |
|---|---|---|
| P1A 22 | VBAT1 | PPS VSP4030 positive terminal |
| P1A 6 | VBAT1 | PPS VSP4030 positive terminal |
| P1B 1 | GND | Ground |
| P1A 1 | GND | Ground |
| P1B 4 | GND | Ground |
| P1A 4 | GND | Ground |
| P1B 9 | GND | Ground |
| P1A 9 | GND | Ground |
| P1B 44 | GND | Ground |
| P1A 44 | GND | Ground |
| P1B 47 | GND | Ground |
| P1A 47 | GND | Ground |
| P1B 50 | GND | Ground |
| P1A 50 | GND | Ground |
| P2B 1 | GND | Ground |
| P2A 1 | GND | Ground |
| P2B 36 | GND | Ground |
| P2A 36 | GND | Ground |
| P2B 22 | GND | Ground |
| P2A 22 | GND | Ground |
| P2B 8 | GND | Ground |
| P2A 25 | GND | Ground |
| P2B 42 | GND | Ground |
| P2B 11 | GND | Ground |
| P2A 28 | GND | Ground |
| P2B 14 | GND | Ground |
| P2A 14 | GND | Ground |
| P2B 15 | GND | Ground |
| P2A 15 | GND | Ground |
| P2B 50 | GND | Ground |
| P2A 50 | GND | Ground |

Figure 119: Hardware interface connection

The block diagram of the test setup is shown below. Since SimPower system model used in MIL simulation cannot be directly flash into the dSPACE system due to fast sampling requirement. A new model equivalent to SimPower system model has been made, and is used for this HIL simulation. The details of switch model to average model conversion are out of the scope of this report. Note that we are using an average plan model in our dSPACE system.

dSPACE HIL system is control through Laptop running Control desk software for monitoring and controlling of Plan parameter at run-time. After the build process the code is flash to the ECU using ICD3 programmer as shown in the diagram, ECU is named as 'In-house ISG Drive Controller', another Laptop running INCA and LAbview is used to control and monitor the parameter of the controller ECU.

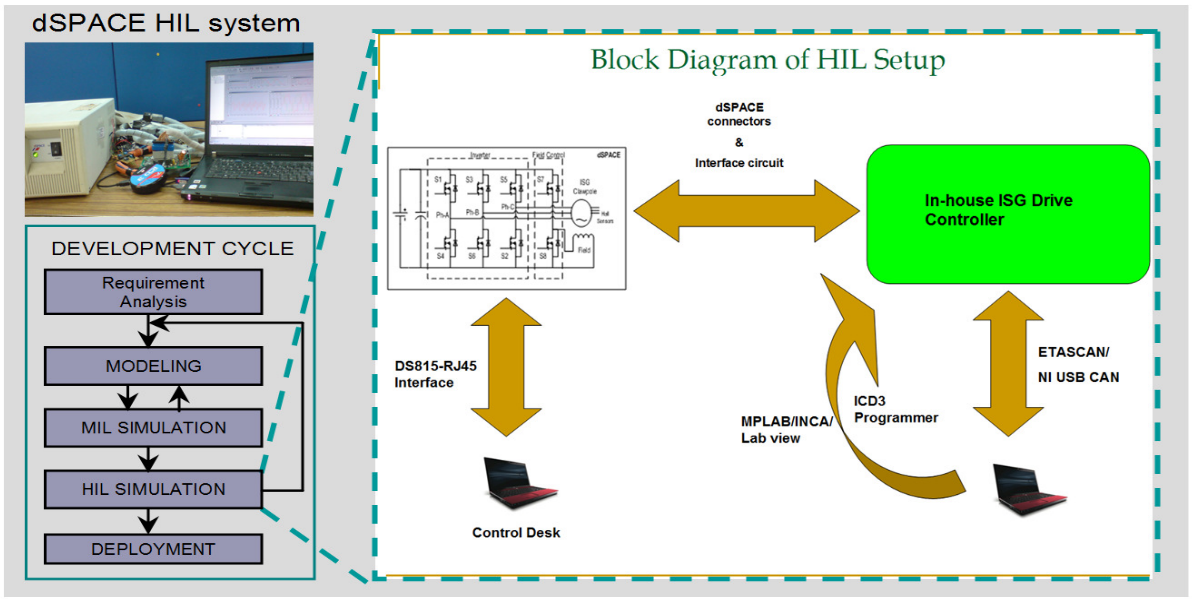


Figure 120: dSPACE setup block diagram

## 5.3 TEST RESULT

- Switching pattern of the PWM signals (7.8/10 kHz switching frequency) generated by the controller based on hall sensor signals is verified. Motor stalls if the pattern is incorrect.
- Current control works fine with Fixed-point PID controller

- PI Tuning considering:-
    - ❑ Lock (high Load Torque= 60-100 Nm).
    - ❑ Rotating with fixed Load Torque and fixed reference current.
    - ❑ Varying load torque with fixed reference current.
    - ❑ Varying reference current (causes speed variation) at constant Load condition.
- All the condition shown below replicated the MIL simulation result with a slight different in the waveform which is an effect of discrete sampling.

### 5.3.1 Lock condition from start

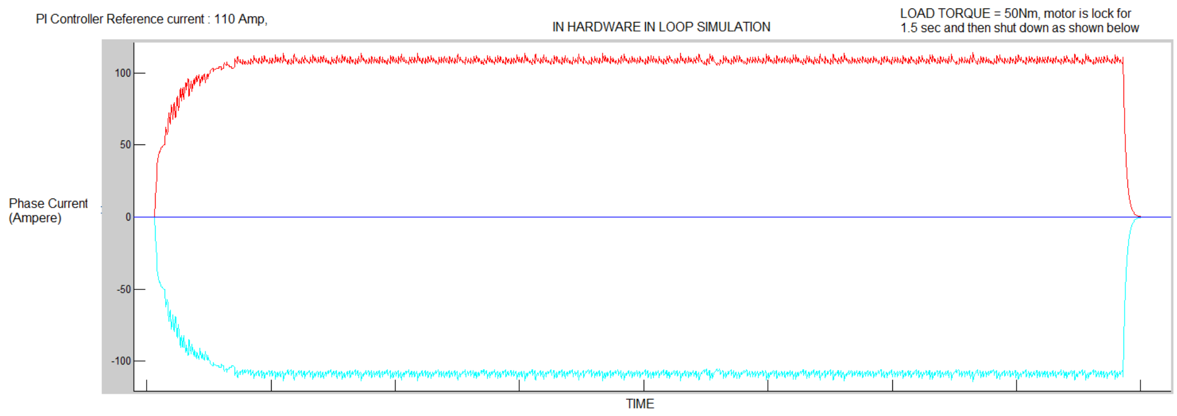


Figure 121: Lock condition from start (HIL)

### 5.3.2 Motor torque higher than load torque

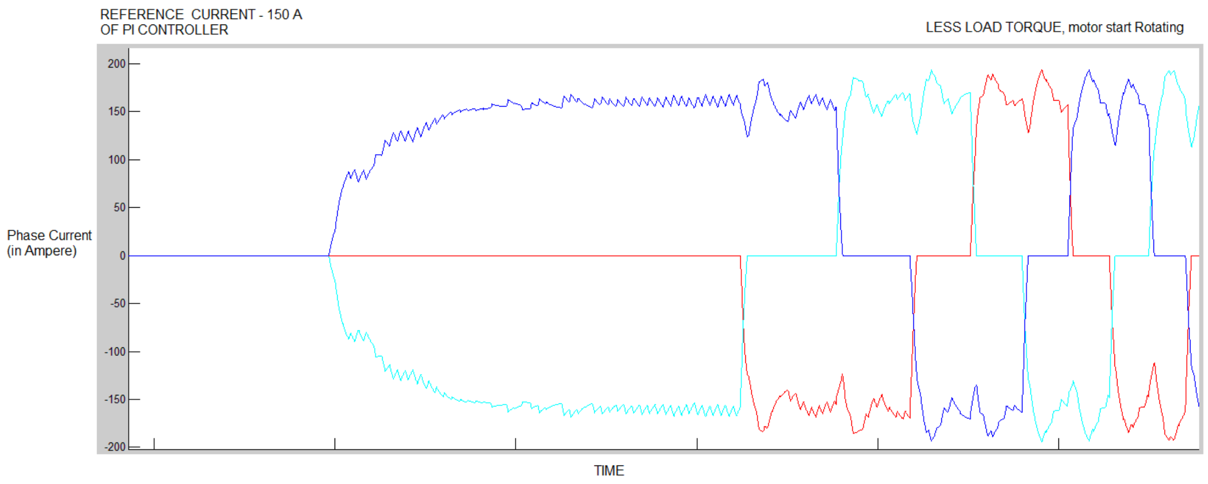


Figure 122: motor running (HIL)

### 5.3.3 Lock condition while running

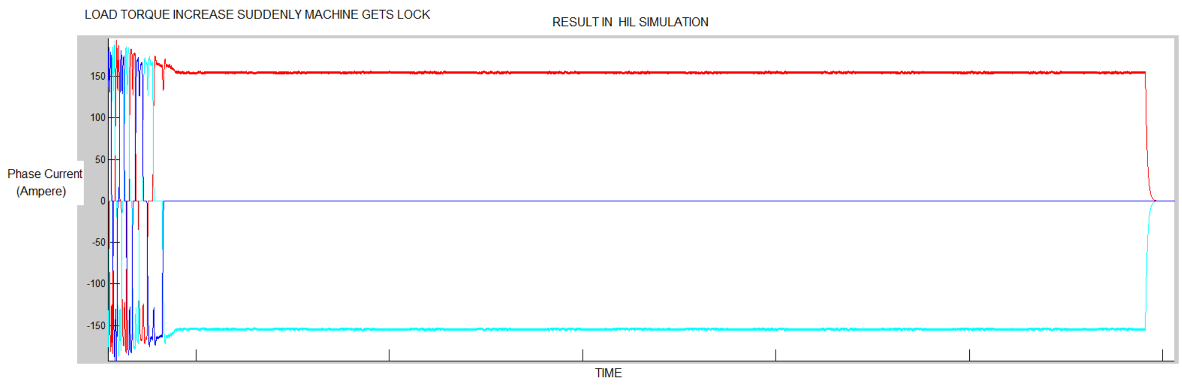


Figure 123 : motor lock while running (HIL)

### 5.3.4 Comparison of control at different rpm

Analysis has been done at two different rpm of the machine, the nature of block current control behavior is observed. It has been found that sampling rate of the PID controller effect the accuracy of control at different speed. The block phase current control at lower rpm is much better compare to a higher rpm as shown by figure below.

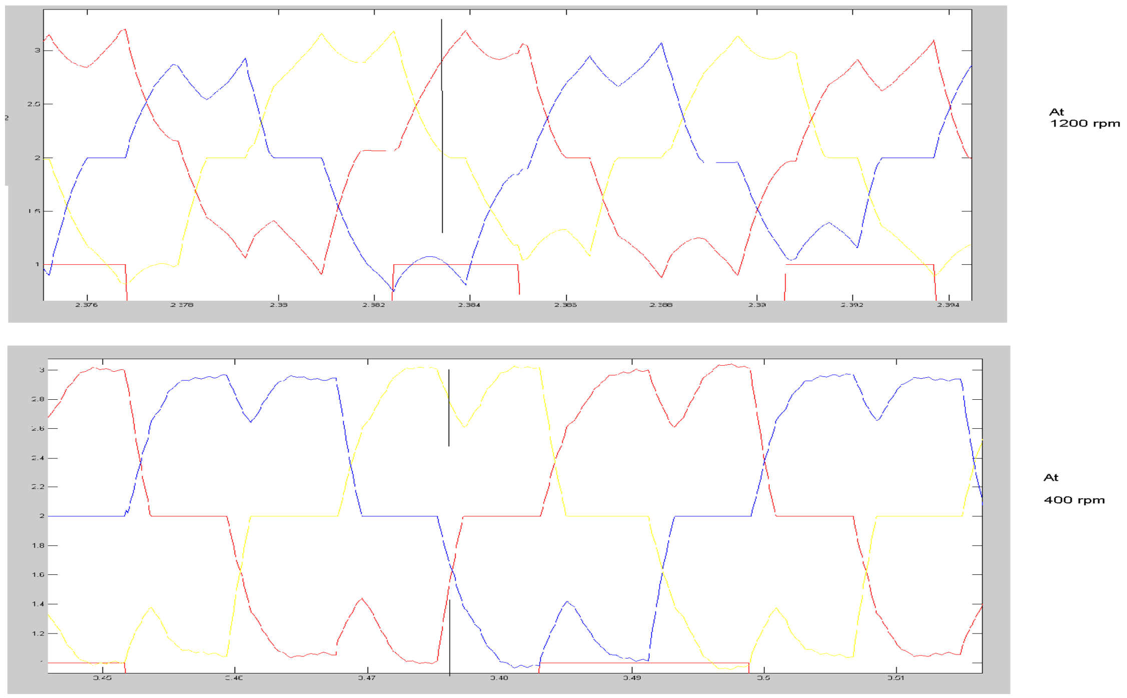


Figure 124: control behavior at different rpm

### 5.3.5 Fault test

Fault is classified uniquely as listed below and test conditions are created through dSPACE Plan model and the following test has been observed, and are all passed.

- Hardware Fault
  - ❑ Heat-sink Trip. (Controller over temperature)
  - ❑ H-Bridge Trip.
  - ❑ 2132_Fault Trip.
  - ❑ Motor over Temperature.
- Motor Fault.
  - ❑ Motor Lock.
  - ❑ Overcharging Voltage.
  - ❑ Hall error.
  - ❑ Over Speed.
- System Fault.
  - ❑ Bus under Voltage.
  - ❑ Bus over Voltage.
  - ❑ CAN Fault.
  - ❑ Current Fault.
    - Bus Negative over Current
    - Bus Positive over Current
    - RPhase Negative over Current
    - RPhase Positive over Current
    - YPhase Negative over Current
    - YPhase Positive over Current
    - BPhase Negative over Current
    - BPhase Positive over Current
- Minor Fault.
  - ❑ Hall reversal.
  - ❑ CAN Node not detect

It is observed that the controller doesn’t latch the fault when over-current is detected. These types of problems have been resolved by making Latching logic in software. Any Fault occurrences will shutdown the system immediately except for minor fault. Software response for all the fault conditions has been checked.

# 6 CHAPTER 6: CONCLUSION

## 6.1 Result comparison of HIL and MIL

Each of the waveform observed in the MIL and HIL has the same Load torque and control reference. Since, we are developing a motor controller software in which current is the main control parameter, so all result are made on the observation of the armature winding phase currents. The torque produce by the motor is also directly proportional to the amount of phase current applied.  The nature of the waveform between MIL and HIL result is shown in the consecutive section below on three different cases. If we observe the nature of HIL result, it is clearly showing that the nature of their waveform is curlier than MIL waveform due to lower sampling rate. The MIL offline simulation was done at 1 usec sampling, but the HIL with dSPACE runs at 100usec sampling which is at the order of 100 differences. But note that control algorithm runs at the same rate in time in both MIL and HIL.

### 6.1.1 Lock condition

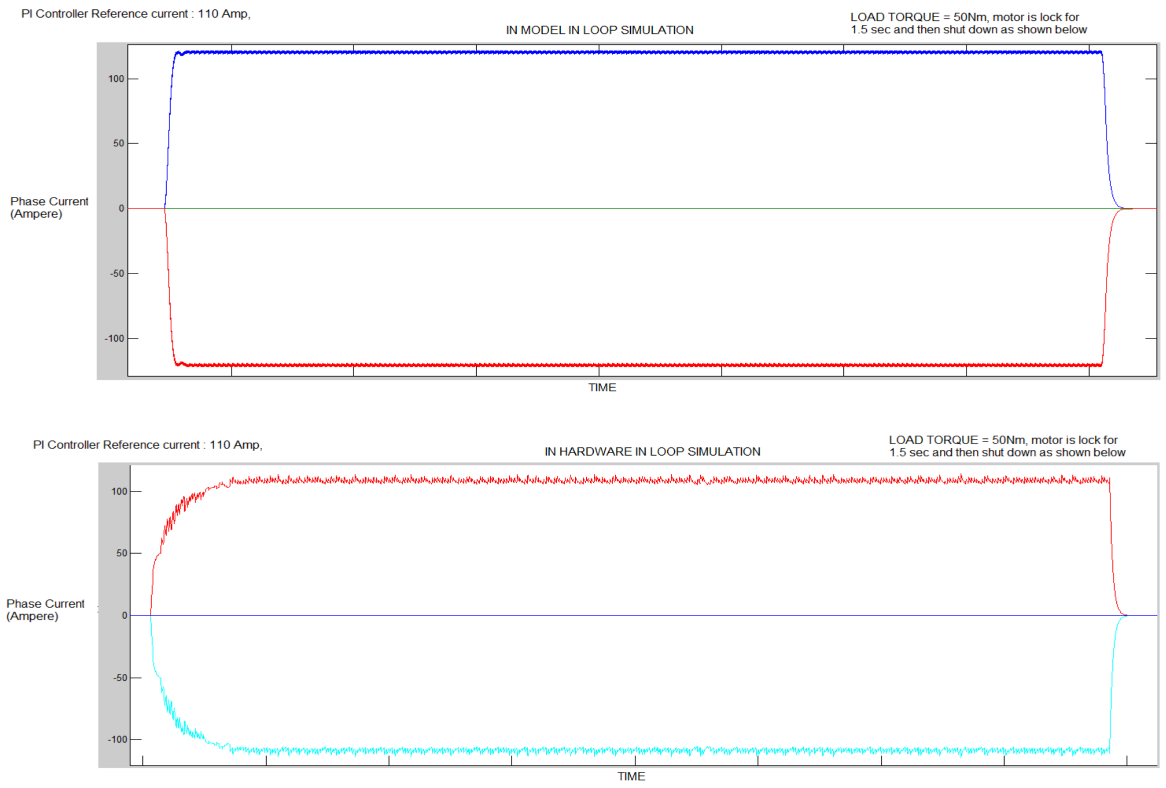


Figure 125: Lock conditions compare MIL and HIL

### 6.1.2 Motor torque higher than load torque

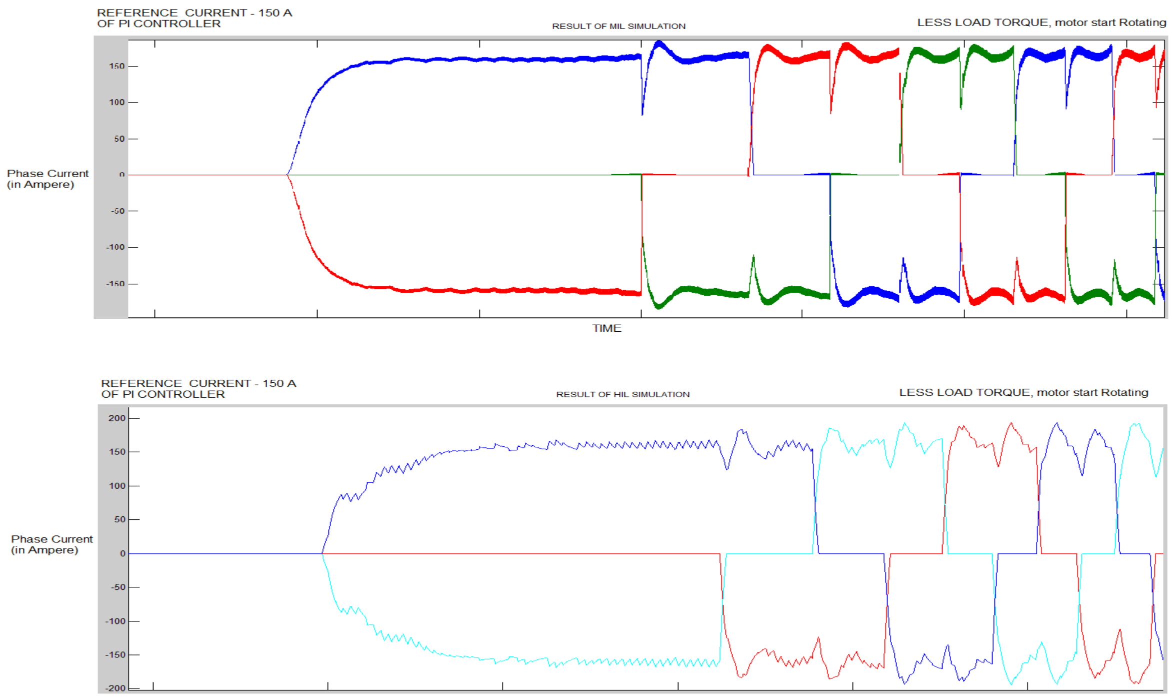


Figure 126: motor running compare MIL and HIL

### 6.1.3 Lock condition while running

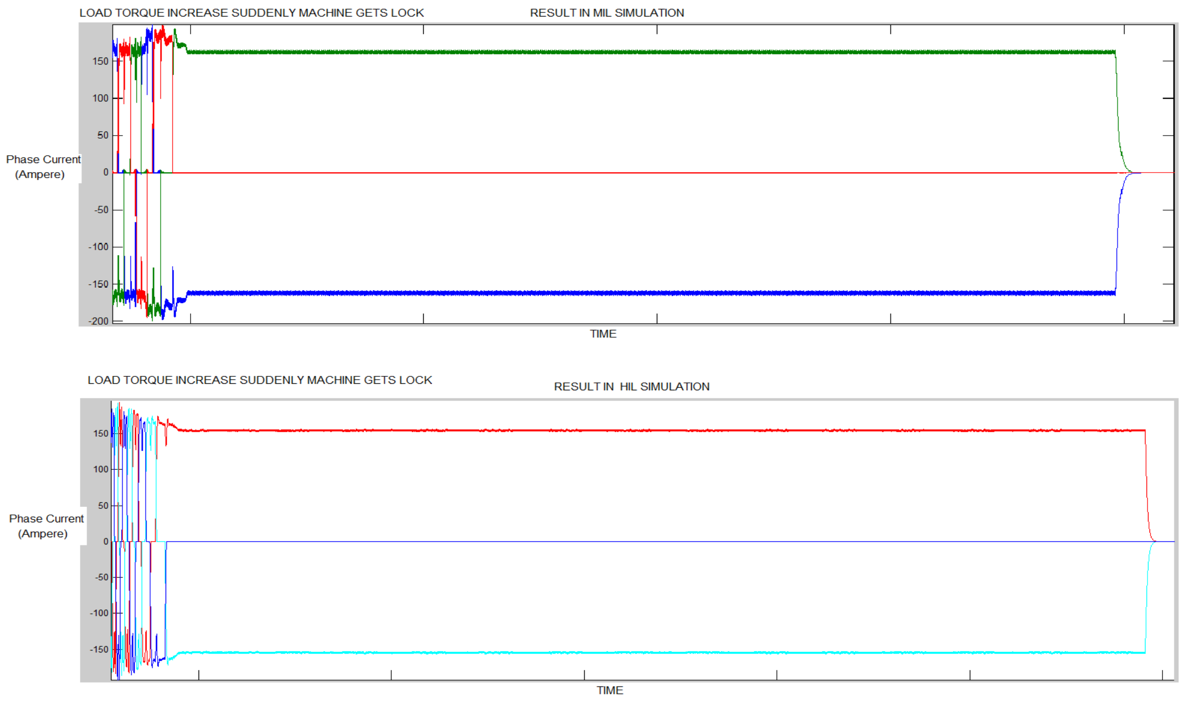


Figure 127: motor lock while running compare MIL and HIL

## 6.2 DISCUSSION

We have discussed all the development process which we will be coming across in the model based development and validation of motor controller software, before an actual deployment to Test Bench or Vehicle. All throughout this development and validation activity, the software bugs in the model has been continuously corrected. It has been observed that MIL can be a very good validation tool for functional behavior of the system, due to this reason I have hit hard the modeling section of this report on modeling of microcontroller functionality which is a unique approached, and also helps in porting the generated source codes in the microcontroller IDE environment. Porting of a generated code of different ISR function can be difficult, but this approach helps in easy porting since we consider the microcontroller execution instants during modeling or MIL itself. Also, HIL validation help to find many hardware issue in the circuit as well as in the platform code. Fault injection into the plan which cannot be generated or dangerous at the actual physical environment like over-current testing can be done.

## 6.3 FUTURE WORK

This development has cover only the motoring function of the TAISG machine functionalities, but the other functions like generating mode (normal charging, synchronous rectification, and boosting operation) are the future scope of this work. But with this tool chain setup in hand, it would really fasten the development of other functions of the system which are to be implemented and tested. Also, this HIL setup can be used as a based setup for further full hybrid HIL simulation.

# APPENDIX I

**FILENAME: code_generation.m (m-script)**

```
clear all
clc

delete('ISGECUSoftware.mdl'); % Delete previous copy of the model
close_system('ISGECUSoftware');% Close if open
open_system('control_sim_v2p3_1');%OPEN the main model
clear
load ('matlab.mat');
%To use MATLAB commands to change data in a model workspace,
%first get the data from Modelworkspace for the currently selected model:
hws = get_param('control_sim_v2p3_1', 'modelworkspace');
hws.DataSource = 'MAT-File';
hws.FileName = 'MWparams';
hws.saveToSource; % MWparams.mat file will be save in working directory
hws.DataSource = 'MDL-File'; %Attached back modelworkspace variable with
Model

%CREATE new Model from where we are going to generate the code
new_system('ISGECUSoftware');
save_system('ISGECUSoftware');

%COPY CONFIGURATION SETTING to a new Model
%The following example creates a copy of ModelA's active configuration
%object and attaches it to ModelB, changing the name if necessary to be
unique.
%The code is the same whether the object is a configuration set or
configuration reference.
myConfigObj = getActiveConfigSet('control_sim_v2p3_1');
newConfigObj = attachConfigSetCopy('ISGECUSoftware', myConfigObj, true);

%COPY a Subsystem named 'ISGECUSoftware' from Base model
%control_sim_v2p3_1.mdl to new model 'ISGECUSoftware.mdl'
Simulink.SubSystem.copyContentsToBlockDiagram('control_sim_v2p3_1/ISG_CONTR
OLLER_PCB/MCU_18F4431Chips/ISGECUSoftware', 'ISGECUSoftware');

%CLOSE the Main Model
save_system('control_sim_v2p3_1');
close_system('control_sim_v2p3_1');
%save, close and open new model create from specific subsystem
save_system('ISGECUSoftware');
close_system('ISGECUSoftware');
open_system('ISGECUSoftware');

%Set ACTIVE COPNFIGURATION TAKEN FROM MAIN MODEL
setActiveConfigSet(gcs, 'Configuration1');
save_system('ISGECUSoftware');

%LOAD Model workspace data to new Model 'ISGECUSoftware.mdl' Model
workspace
hws = get_param('ISGECUSoftware', 'modelworkspace');
hws.DataSource = 'MAT-File';
hws.FileName = 'MWparams';
hws.reload;
hws.DataSource = 'MDL-File'; %Attached modelworkspace variable with Model
delete MWparams.mat %delete the local copy of modelworkspace variable
%load workspace_variables.mat;

%DELETE TRIGGER BLOCK for Code Optimization
Total_Trigger_blocks=find_system('ISGECUSoftware', 'blocktype',
'TriggerPort');
delete_block(Total_Trigger_blocks);

%DELETE 10msec_scheduler BLOCK for Code Optimization
delete_block('ISGECUSoftware/Background/10msec_scheduler');
save_system('ISGECUSoftware');



%RTW code Generation
rtwbuild('ISGECUSoftware');
save_system('ISGECUSoftware');
close_system('ISGECUSoftware');
%delete ISGECUSoftware.mdl;
```

# APPENDIX II

**FILENAME: asap2Complete.m (M-script)**

```
function varargout = ASAP2Complete(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ASAP2Complete_OpeningFcn, ...
                   'gui_OutputFcn',  @ASAP2Complete_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before ASAP2Complete is made visible.
function ASAP2Complete_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ASAP2Complete (see VARARGIN)

% Choose default command line output for ASAP2Complete
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
set(handles.GenerateButton,'Enable','off');
set(handles.ASAP2BrowseButton,'Enable','on');
set(handles.MAPBrowseButton,'Enable','off');
% UIWAIT makes ASAP2Complete wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --- Outputs from this function are returned to the command line.
function varargout = ASAP2Complete_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in ASAP2BrowseButton.
function ASAP2BrowseButton_Callback(hObject, eventdata, handles)
% hObject    handle to ASAP2BrowseButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 global ASAP2filename;
 global ASAP2pathname;
    [ASAP2filename, ASAP2pathname] = uigetfile('*.a2l', 'Pick an A2L-
file');
    if isequal(ASAP2filename,0) || isequal(ASAP2pathname,0)
       set(handles.ASAP2StaticText,'String','User pressed cancel');
    else
       set(handles.ASAP2StaticText,'String',['User selected ',
fullfile(ASAP2pathname, ASAP2filename)]);
       if isequal (cd,ASAP2pathname)
           error(' #####File on Current Directory****');
       else
            copyfile(fullfile(ASAP2pathname, ASAP2filename),ASAP2filename);
       end
       set(handles.ASAP2BrowseButton,'Enable','off');
       set(handles.MAPBrowseButton,'Enable','on');
    end
```

```
% --- Executes on button press in MAPBrowseButton.
function MAPBrowseButton_Callback(hObject, eventdata, handles)
% hObject    handle to MAPBrowseButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 global MAPfilename;
 global MAPpathname;
 PerlFile  = 'convertMAP_Hitech.pl';
    [MAPfilename, MAPpathname] = uigetfile('*.map', 'Pick an MAP-file');
    if isequal(MAPfilename,0) || isequal(MAPpathname,0)
       set(handles.MAPStaticText,'String','User pressed cancel');
    else
       set(handles.MAPStaticText,'String',['User selected ',
fullfile(MAPpathname, MAPfilename)]);
        set(handles.GenerateButton,'Enable','on');
        set(handles.ASAP2BrowseButton,'Enable','off');
        set(handles.MAPBrowseButton,'Enable','off');
        if isequal (cd,MAPpathname)
           error(' #####File on Current Directory****');
        else
            copyfile(fullfile(MAPpathname, MAPfilename),MAPfilename);
            % Call PerlFile from operating system
            result = perl(PerlFile,MAPfilename);
            %Copy hex-file also for just for easy reference
            [hexfilename, hexpathname] = uigetfile('*.hex', 'Pick an HEX-
file');
            if isequal(hexfilename,0) || isequal(hexpathname,0)
               disp('User pressed cancel')
            else
               disp(['User selected ', fullfile(hexpathname,
hexfilename)]);
               copyfile(fullfile(hexpathname, hexfilename),hexfilename);
            end
        end
   end


% --- Executes on button press in GenerateButton.
function GenerateButton_Callback(hObject, eventdata, handles)
% hObject    handle to GenerateButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 global MAPfilename;
 global ASAP2filename;

  asap2post(ASAP2filename, ['temp_',MAPfilename]);
    disp('******* A2L File Created in working directory *******');
    set(handles.genstatusStaticText,'String','******* A2L File Created in
working directory *******');
    set(handles.GenerateButton,'Enable','off');
    %Delete after all
delete(['temp_',MAPfilename]);
delete(MAPfilename);
%delete(ASAP2filename);
```

# APPENDIX III

**FILENAME: convertMAP_Hitech.pl (perl Script)**

```
#***Hitech C compiler MAP file Compatible to MATLAB A2L file generation****
#******* RUN THIS SCRIPT BY INPUTING MAP FILE OF HI-TECH COMPILER*********
#***********USE THIS OUTPUT FILE MAP FILE TO asap2post.m*****************
# ****** CREATED ON : 13-AUG-2011, 11:45PM IST     ***********************

$MAPFileName = $ARGV[0];# File input name'MPLAB_TAISG_project_v0p0.map'; #
$outputFileName = "temp_" . $MAPFileName;      # File output name

open(MAPFILE, $MAPFileName)
  || die "PERL Error: Couldn't open MAP file: ", $MAPFileName, ".\n";
undef $/; $MAPFileString = <MAPFILE>; $/ = "\n";
close(MAPFILE);

# - Replace consecutive white-space characters with a single space
$MAPFileString =~ s/\s+/\n/g;
@FileArray =split(/\n/, $MAPFileString);
# - Convert MAPFileString to MAPFile Hash Table

open(OUTPUTFILE, (">" . $outputFileName))
  || die "PERL Error: Couldn't open output file: ", $outputFileName, "\n";

$Length=scalar(@FileArray);
print "Length of Array =",$Length,"\n";
$n=0;
$count=0;
$foundSymbol=0;
while ($n < $Length) {
    if(($FileArray[$n] eq "Symbol")&& ($foundSymbol==0)) {
        print "Found Symbol Table at ",$n,"----",$FileArray[$n],"\n";
        $SymbolIndexnumber=$n;
        $foundSymbol=1;
     }
     if(($FileArray[$n] eq "Table") && ($foundSymbol==1))
     {  print "Found Symbol Table at ",$n,"----",$FileArray[$n],"\n";
        $SymbolIndexnumber=$n;
        $foundSymbol=2;
     }
     if(($foundSymbol==2))
     {
            push(@NewFileArray,$FileArray[$n]);
            #print "New Array [",$count,"]----",$NewFileArray[$count],"\n";
            $count++;
     }

     $n++;
}
if($foundSymbol!=2){
    print "\n***********Wrong Input file********Not a Hi-tech PICC18 MAP
file*****\n";
}
else
{

    $NewLength=scalar(@NewFileArray);
    print "\nLength of New Array =",$NewLength,"\n";

    $NewLength=$count;
    $n=0;
    $count=0;
    while ($n < $NewLength) {
        print "New Array [",$n,"]----",$NewFileArray[$n],"\n";
        if(($NewFileArray[$n] eq "bss")||($NewFileArray[$n] eq
"bigbss")||($NewFileArray[$n] eq "data")||($NewFileArray[$n] eq
"bigdata")||($NewFileArray[$n] eq "const"))
        {
            $NewFileArray[$n-1] =~ s/^_//g;
            push(@FinalFileArray,$NewFileArray[$n-1]); # Push Variable Name
            push(@FinalFileArray,"\t"); #Push Tab
            $ecuadress="0x".$NewFileArray[$n+1]; #augment '0x'
            push(@FinalFileArray,$ecuadress); # Push Address of Variable
            push(@FinalFileArray,"\n"); #Push new line
            $count=$count+4;
            print "###*****###",$count,"   index",$n,"\n";

        }
        $n++;
    }
    #NewFileArray contain contain only the required data but still types
name need to remove
    $NewLength=$count;
    $n=0;
    print "\n\n*******************SEMI-FINAL****************\n\n";
    while ($n < $NewLength) {
        print "Final Array [",$n,"]----",$FinalFileArray[$n],"\n";
        $n++;
    }
    print "\n\n******************FINAL****************\n\n";
    $Finalbuff=join("",@FinalFileArray);
    print "\n\n";
    print $Finalbuff;
    print "\n\n";
    print %Finalbuff;

    print OUTPUTFILE $Finalbuff; #Print to FILE
   rename $outputFileName, $MAPFileName;
    print "\n\nComplete MAP file generation....... \n";
    close(OUTPUTFILE);
}
```

# REFERENCES

1. MATHWORKS SIMULINK HELP, www.**mathworks**.com/**help/**
2. VECTOR ECU Calibration, www.vector.com
3. ETAS INCA, www.etas.com
4. MPLAB IDE, www.microchip.com
5. Hi-tech C Compiler, www.htsoft.com
6. MAAB modeling guidelines.
7. MISRA C coding standard.
8. Gabriela Nicolescu, Pieter J. Mosterman http://www.amazon.com/Model-Based-Embedded-Computational-Analysis-Synthesis/dp/1420067842 : Model-Based Design for Embedded Systems (Computational Analysis, Synthesis, and Design of Dynamic Systems), Taylor & Francis Group, LCC 2010.
9. Tata motors Ltd, "TAISG_functional_requirements_V2.0.pdf", Advance Engineering department, ERC TATA motors Ltd, Pimpri, Pune, 2011.
10. Stefan Baldursson, "BLDC Motor Modeling and Control – A Matlab®/Simulink® Implementation", Master Thesis work. May, 2005.
11. A.Tashakori, M. Ektesabi, and N. Hosseinzadeh "Modeling of BLDC Motor with Ideal Back-EMF for Automotive Applications", July 2011, London, UK.
12. Wonbok Hong, Wootaik Lee, Byoung-Kuk Lee, "Dynamic simulation of brushless DC motor drives considering phase commutation for automotive applications", Electric Machines & Drives Conference, 2007. IEMDC '07. IEEE International, 3-5 May 2007, Antalya, Turkey.