VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Xiaotian Li

MODEL-BASED DESIGN OF BRUSHLESS DC MOTOR CONTROL AND MOTION CONTROL MODELLING FOR ROBOCUP SSL ROBOTS

Technology and Communication

2015

ACKNOWLEDGEMENTS

Hereby I would like to give my sincere thanks to all the people who helped me and inspired me during my study in Degree Programme in Information Technology at Vaasan Ammattikorkeakoulu, Vaasa University of Applied Sciences.

I would like to thank my supervisor Dr. Yang Liu for his excellent guidance and help. During the past three years, I have not only learned professional knowledge from his lectures, but also benefited a lot from his the splendid way of thinking. He also provided us precious opportunities to join the Botnia Robocup team to work on promising projects during which I gained research experiences, cultivated my self-learning skill and found the path forward.

I am also very grateful to Dr. Chao Gao, Dr. Smail Menani, Mr. Jukka Matila, Dr. Ghodrat Moghadampour, Mr. Santiago Chavez, Mr. Ari Urpiola and all others staff of VAMK who helped me during my undergraduate study.

In addition, I am very thankful to all my fellows in our Botnia Robocup laboratory past and present for all those days we inspired and encouraged each other and devoted to challenging work, and for all the joys we had during the past years.

Finally, I would like to thank my family, especially my parents, for their support, understanding and patience. They are always the most important part of my life.

Vaasa, Finland

30/3/2015

Xiaotian Li

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Programme in Information Technology

ABSTRACT

Author	Xiaotian Li
Title	Model-Based Design of Brushless DC Motor Control and Motion Control Modelling for RoboCup SSL Robots
Year	2015
Language	English
Pages	104 + 4 Appendices
Name of Supervisor	Yang Liu

Over the recent years, the RoboCup competition has grown popular and attracted more and more domestic and international universities, and the levels of the teams increase every year. In Small Size League (SSL) competition, besides a good strategy system, the precision of the robots' actions is also of vital importance in order to achieve high performance. Thus, a highly accurate and stable motion control system is needed to drive the robots to move in accordance with the planned trajectory.

Currently, our Botnia robots has a reasonable strategy system, but the motion control system of our robots is not reliable which limits the performance of our robots in the game. Therefore, the aim of this thesis was to find the ways to improve the motion control system of our Botnia robots.

In this thesis, first the Model-Based Design method which is an efficient and cost-effective way to develop embedded systems was implemented to design the single brushless DC motor control built based on Arduino Mega 2560 and Altera DE2. Then, the whole model of the motion control system was built to simulate the motion control process and used to discuss the performance. The testing results were analysed and the future improvements suggested.

According to the testing results, the single BLDC motor control works well, so it can be further transplanted and implemented on the robots hardware which has the similar structure - ARM Cortex-M3 + Cyclone III FPGA. And according to the simulation results, the performance of the whole motion control system will benefit a lot if an accelerometer and a gyroscope are introduced to complete the overall speed control loop in the future.

Keywords Model-based Design, Brushless DC Motor, Motion Control

CONTENTS

ACKNOWLEDGEMENTS

ABSTRACT

1 INTRODUCTION 11

1.1 Objective 11

1.2 Overall Structure of Thesis 11

1.3 Background of Botnia SSL Robot Team 11

1.3.1 RoboCup 12

1.3.2 Small Size League 12

1.3.3 Structure of Botnia Robot System 12

1.4 Model-Based Design with MATLAB and Simulink 14

1.5 MATLAB/Simulink Support for Arduino 15

1.5.1 MATLAB Support for Arduino 15

1.5.2 Simulink Support for Arduino 16

1.6 HDL Coder 17

1.7 Brushless DC Motor 17

1.8 PID Controller 18

2 MODELLING AND SIMULATION 20

2.1 BLDC Motor 21

2.1.1 Structure of BLDC Motor 21

2.1.2 Operation of BLDC Motor 22

2.1.3 Mathematic Equations 25

2.1.4 Simulink Model 27

2.1.5 Commutation and Inverter 34

2.1.6 Simscape Model 40

2.1.7 Equivalent DC Motor Model 40

2.2 Modelling of Single BLDC Motor Control System 44

3 DESIGN OF SINGLE BLDC MOTOR CONTROL SYSTEM 48

3.1 Overall Structure of System 48

3.2 Motor Drive Circuit and Motor 49

3.3 Altera DE2 52

3.3.1 PWM Generation 53

3.3.2 Commutation ... 54
3.3.3 Display ... 58
3.3.4 Overall ... 68
3.4 Arduino Mega 2560 ... 73
3.4.1 Freqcount Block ... 75
3.4.2 Data Type Conversion ... 81
4 MODELLING OF MOTION CONTROL SYSTEM ... 83
4.1 Kinematic Model ... 84
4.2 Four Single Motor Control Systems ... 86
4.3 Velocity Decomposition ... 88
4.4 Velocity Composition ... 88
4.5 Trajectory ... 89
5 TESTING RESULTS AND ANALYSIS ... 91
5.1 Simulation Results of BLDC Motor ... 91
5.2 Single BLDC Motor Control System ... 93
5.3 Simulation Results of Motion Control System ... 94
6 DISCUSSION AND FUTURE RESEARCH ... 97
6.1 Rotational Speed Measurement ... 97
6.2 Deviation of Mass Centre ... 98
6.3 Slipping Wheels ... 98
6.4 Design of Entire Motion Control System and SoC Solution ... 99
7 CONCLUSION ... 100
REFERENCES ... 102
APPENDICES

LIST OF FIGURES AND TABLES

Figure 1.	Overall structure of Botnia robot system/4/	p. 13
Figure 2.	Botnia SR6/5/	p. 13
Figure 3.	Model-Based Design (1) /7/	p. 14
Figure 4.	Model-Based Design (2) /8/	p. 15
Figure 5.	Maxon EC 45 Flat datasheet/15/	p. 20
Figure 6.	BLDC motor structure/17/	p. 21
Figure 7.	Phase voltage, current and hall-sensor waveforms with respect to rotor electrical angle/19/	p. 23
Figure 8.	Interaction of rotor and stator/17/	p. 23
Figure 9.	BLDC drive scheme (1)/18/	p. 24
Figure 10.	BLDC drive scheme (2)/17/	p. 24
Figure 11.	Current generation subsystem	p. 28
Figure 12.	State ia and ib	p. 29
Figure 13.	EMF generation subsystem	p. 30
Figure 14.	Phase a, phase b and phase c	p. 31
Figure 15.	Mechanial subsystem	p. 32
Figure 16.	Model of BLDC motor	p. 33
Figure 17.	Circuit configuration in each interval/18/	p. 34
Figure 18.	Circuit configuration in the first interval	p. 35

Figure 19.	Commutation and inverter subsystem	p. 39
Figure 20.	BLDC drive model	p. 40
Figure 21.	Simscape BLDC motor model	p. 40
Figure 22.	DC motor equivalent circuit /20/	p. 41
Figure 23.	Equivalent DC motor model	p. 43
Figure 24.	Model of BLDC motor control system	p. 44
Figure 25.	Plant Identification – simulate I/O data	p. 46
Figure 26.	Plant Identification – Auto Estimate	p. 46
Figure 27.	PID tuner	p. 47
Figure 28.	Overall structure of BLDC motor control system	p. 48
Figure 29.	Motor drive circuit and motor	p. 49
Figure 30.	L6234 block diagram/21/	p. 50
Figure 31.	Pin functions/21/	p. 51
Figure 32.	Structure of control program run on Altera DE2	p. 52
Figure 33.	Model of commutaion logic module	p. 55
Figure 34.	Convert 10-bit binary to BCD	p. 59
Figure 35.	Position and index of each segment in a 7-segment display/23/	p. 60
Figure 36.	Model of 7-segment display logic	p. 61
Figure 37.	Model of display module	p. 62

Figure 38. Generated source files p. 63

Figure 39. PWM generation subsystem p. 68

Figure 40. Model of the whole system run on DE2 p. 69

Figure 41. Files for the whole program p. 70

Figure 42. Structure of control program run on Arduino Mega 2560 p. 74

Figure 43. Model for running on Arduino p. 74

Figure 44. Initialization pane p. 76

Figure 45. Data Property pane p. 77

Figure 46. Libraries pane p. 77

Figure 47. Discrete Update pane (Initialization part) p. 79

Figure 48. Outputs pane p. 80

Figure 49. Data type conversion subsystem p. 82

Figure 50. Structure of motion control system p. 83

Figure 51. Model of motion control system p. 84

Figure 52. Arrangement of the wheels p. 85

Figure 53. Model of four single motor control systems p. 87

Figure 54. Trajectory subsystem p. 90

Figure 55. Rotor speed p. 91

Figure 56. Electrical torque p. 92

Figure 57.	Phase currents	p. 92
Figure 58.	Zoomed view of phase currents	p. 93
Figure 59.	Closed-loop simulation result	p. 94
Figure 60.	Closed-loop testing result	p. 94
Figure 61.	Actual trajectory vs planned trajectory (1)	p. 95
Figure 62.	Actual trajectory vs planned trajectory (2)	p. 96
Figure 63.	Actual trajectory vs planned trajectory (3)	p. 96
Figure 64.	Deviation of mass centre /25/	p. 98
Figure 65.	Block diagram of the DE1-SoC board/27/	p. 99
Table 1.	Switching sequence	p. 25
Table 2.	Inverter output voltages (1)	p. 35
Table 3.	Inverter output voltages (2)	p. 36
Table 4.	Relationship between Hall sensor signals and inputs of L6234	p. 51
Table 5.	7-segment display logic	p. 60

LIST OF APPENDICES

APPENDIX 1. Freqcount_wrapper.cpp

APPENDIX 2. Extract_Bits.vhd

APPENDIX 3. HEX4_uni.vhd

APPENDIX 4. Iteration1.vhd

LIST OF ABBREVIATION

DC	Direct Current
BLDC	Brushless DC
SSL	Small Size League
FPGA	Field Programmable Gate Array
emf	Electromotive Force
rpm	Revolutions Per Minute
PCB	Printed Circuit Board
PID	Proportion Integration Differentiation
PWM	Pulse Width Modulation

1 INTRODUCTION

1.1 Objective

The main purpose of this thesis was to find appropriate methods to improve the motion control system of our Botnia SSL robots. The main method used here was the Model-Based Design method with MATLAB/Simulink which was implemented to design the single BLDC motor PID control. It allowed us to optimise PID controller's parameter by using PID tuner and monitor the data in real time as well. In addition, a model of the whole motion control system was built to simulate the control performance, so that we could find what should be improved in the future.

1.2 Overall Structure of Thesis

This thesis includes seven chapters. The structure of the thesis follows the procedure of the entire project. The first chapter gives a brief introduction to the background and related knowledge of this thesis. Chapter 2 contains the process to build the model of both an equivalent DC motor and the Maxon EC45 Flat BLDC motor which is the type of motor used on our robots. Chapter 3 mainly introduces the design of single BLDC motor control system including the building model of the system for simulation and model for running in the external mode on Arduino as well as the coding on FPGA (Altera DE2). And then, in Chapter 4, the building of the whole model of the motion control system is described. In Chapter 5, the testing results and simulation results are discussed and in Chapter 6, the problems of current system are discussed and future research direction is suggested. In the end, the last chapter concludes the entire thesis.

1.3 Background of Botnia SSL Robot Team

The Botnia SSL robot team is one of the most competitive teams in the RoboCup SSL competition and has been in the RoboCup world championship since 2006 and ranked top 10 in the Small-Size League. Our team is the only one qualified for RoboCup SSL competition among the whole Nordic countries and competed with top universities in the world such as Harvard, MIT, Carnegie Mellon, Georgia Tech, etc.

1.3.1 RoboCup

As an annual international robotics competition, RoboCup was founded in 1997. The objective is to stimulate the development of robotics and AI research, by offering a publicly appealing, but formidable challenge./1/

The main focus of the RoboCup competitions is the robot soccer game. In this area, the research goals concern cooperative multi-robot and multi-agent systems in dynamic adversarial environments. All robots in the competitions are fully autonomous. The ultimate goal of the RoboCup project is to develop a team of fully autonomous humanoid robots by 2050, which can win against the human world champion team in soccer./2/

1.3.2 Small Size League

One of the RoboCup league divisions is Small Size League robot soccer. In the Small Size League (F180), problems of intelligent multi-agent cooperation and control in a highly dynamic environment with a hybrid centralized/distributed system are focused./3/

1.3.3 Structure of Botnia Robot System

The overall structure of Botnia robot system is shown in Figure 1. The entire system is made up of several independent but correlative systems. The whole system can be divided into five parts, vision system, strategy system, referee system, wireless telecommunication system and multi-agent system. The motion control system is part of the multi-agent system and it is the last executor of the whole system to precisely and effectively execute the commands given by the strategy system.

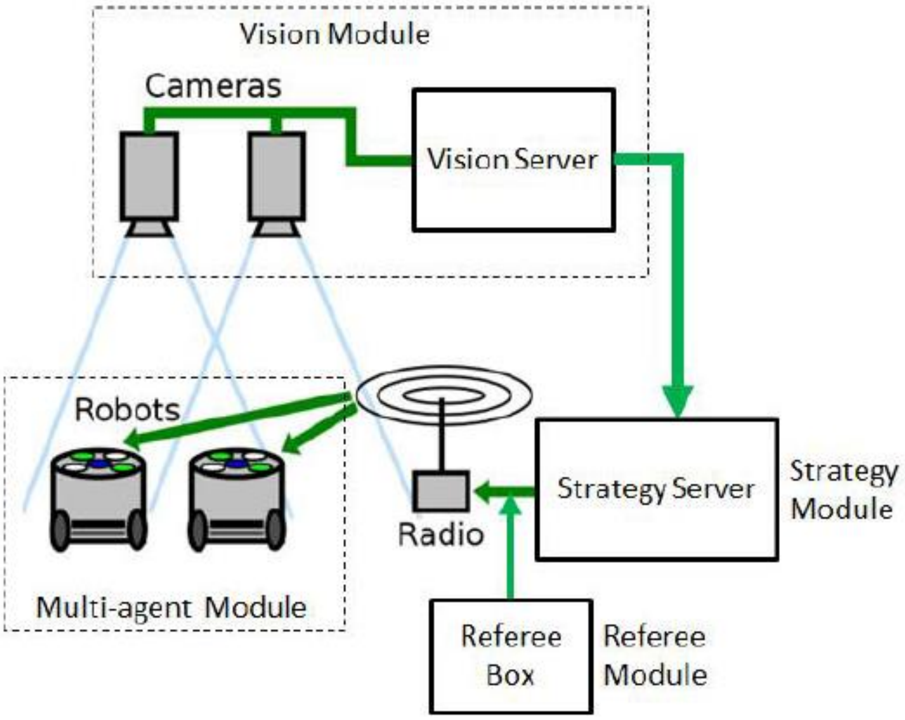


Figure 1. Overall structure of Botnia robot system/4/

Until now, six generations of Botnia robots have been developed. The robot modelled later is the sixth generation robot which is called “SR6”. Figure 2 shows the structure of SR6.

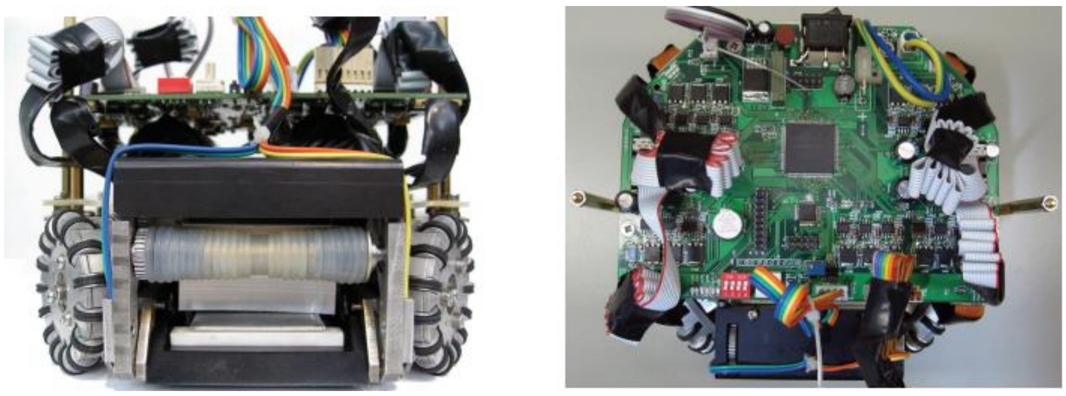


Figure 2. Botnia SR6/5/

Each robot uses four omnidirectional wheels which have become popular for mobile robots and allow a robot to drive on a straight path from any point to any other point on the floor, without having to rotate first. Moreover, the translational movement along a desired path can be combined with a rotation, so that the robot arrives to its destination at the correct angle. However, four wheels provide redundancy and the advantages in manoeuvrability and effectiveness are at the expense of more complex mechanical structure and increased complexity in control.

1.4 Model-Based Design with MATLAB and Simulink

Simulink provides a graphical editor, customizable block libraries, and solvers for modelling and simulating dynamic systems. It is integrated with MATLAB, enabling people to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis. /6/

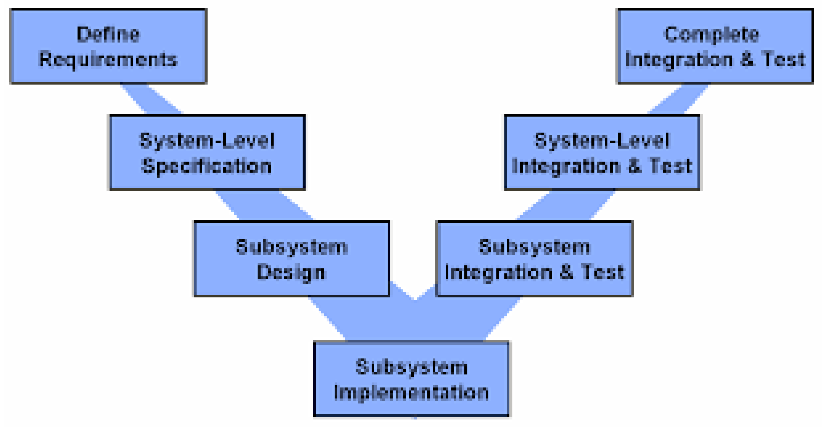

Define Requirements
System-Level Specification
Subsystem Design
Subsystem Implementation
Subsystem Integration & Test
System-Level Integration & Test
Complete Integration & Test

Figure 3. Model-Based Design (1) /7/

Simulink is a block diagram environment for multidomain simulation and Model-Based Design. It supports simulation, automatic code generation, and continuous test and verification of embedded systems. /6/

Figure 3 and Figure 4 give an intuitive way to describe Model-Based Design

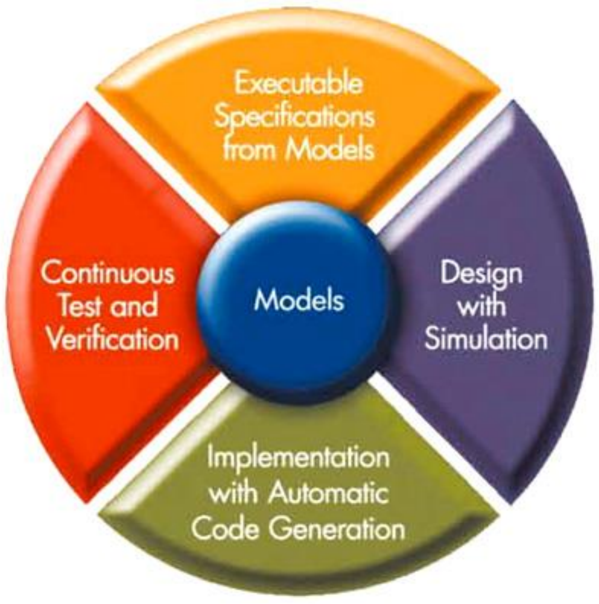

Executable Specifications from Models
Continuous Test and Verification
Models
Design with Simulation
Implementation with Automatic Code Generation

Figure 4. Model-Based Design (2) /8/

The general steps for Model-Based Design: /6/

Building the Model — Model hierarchical subsystems with predefined library blocks.

Simulating the Model — Simulate the dynamic behaviour of system and view results as the simulation runs.

Analysing Simulation Results — View simulation results and debug the simulation.

Managing Projects — Easily manage files, components, and large amounts of data for the project.

Connecting to Hardware — Connect the model to hardware for real-time testing and embedded system deployment.

1.5 MATLAB/Simulink Support for Arduino

1.5.1 MATLAB Support for Arduino

MATLAB Support Package for Arduino hardware enables one to use MATLAB® to communicate with the Arduino® board over a USB cable. This package is based on a server program running on the board, which listens to commands arriving via serial port, executes the commands, and, if needed, returns a result. This approach helps: /9/

- Start programming right away without any additional toolboxes.
- Work in MATLAB for interactive development and debugging.
- Interactively develop programs to acquire analog and digital data, and to control DC, servo, and stepper motors.
- Access peripheral devices and sensors connected over I2C or SPI.
- Run control loops at up to 25 Hz (not real time).
- Introduce mechatronics, signal processing, and electronics concepts in classroom labs

1.5.2 Simulink Support for Arduino

Simulink® can be used to program algorithms that run on the Arduino®, an inexpensive, open-source microcontroller board. Arduino boards feature Atmel® ATmega and ARM Cortex microprocessors and provide digital and analog connectivity and serial communications./10/

Simulink can be used to create algorithms for control system and robotics applications and then simulate to verify that the algorithms work during simulation. With the click of a button, the algorithms can be downloaded and run the algorithms on the embedded processor on the Arduino board. /10/

Simulink support for low cost hardware on the Arduino platform includes: /10/

- Automated installation and configuration
- Library of Simulink blocks that connect to Arduino I/O, such as digital input and output, analog input and output, serial receive and transmit, and servo read and write. Includes additional blocks for UDP send and receive, TCP/IP send and receive, and ThingSpeak write (requires Ethernet or Wifi Shield).
- Interactive parameter tuning and signal monitoring of running applications (not available on some boards)
- Model deployment for standalone operation

1.6 HDL Coder

HDL Coder™ generates portable, synthesizable Verilog® and VHDL® code from MATLAB® functions, Simulink® models, and Stateflow® charts. The generated HDL code can be used for FPGA programming or ASIC prototyping and design./11/

HDL Coder provides a workflow advisor that automates the programming of Xilinx® and Altera® FPGAs. One can control HDL architecture and implementation, highlight critical paths, and generate hardware resource utilization estimates. HDL Coder provides traceability between the Simulink model and the generated Verilog and VHDL code, enabling code verification for high-integrity applications adhering to DO-254 and other standards. /11/

The key Features are: /11/

- Target-independent, synthesizable VHDL and Verilog code
- Code generation support for MATLAB functions, System objects, and Simulink blocks
- Mealy and Moore finite-state machines and control logic implementations using Stateflow
- Workflow advisor for programming Xilinx and Altera application boards
- Resource sharing and retiming for area-speed trade-offs
- Code-to-model and model-to-code traceability for DO-254
- Legacy code integration

1.7 Brushless DC Motor

The brushed DC motor generates torque directly from DC power supplied to the motor by using internal commutation, stationary magnets (permanent or electromagnets), and rotating electrical magnets. This conventional DC motor has much attractive properties, so it has been used both in industry and daily life for many years. The advantages include its low cost, high reliability, high efficiency, linear propertied and simple control of motor speed. However, the disadvantages are also obvious. Its mechanical commutator decides its low life-span for high intensity uses and high maintenance.

Now, due to the rapid development of electronic device, brushless DC motor are widely used to replace the conventional DC motor. Brushless DC motor (BLDC motor) is a type of synchronous motors that are powered by a DC electric source via an integrated inverter which produces an AC electric signal to drive the motor. In this context, AC, alternating current, does not imply a sinusoidal waveform, but rather a bi-directional current with no restriction on waveform. Additional sensors and electronics control the inverter output amplitude and waveform (and therefore percent of DC bus usage/efficiency) and frequency (i.e. rotor speed). The rotor part of a brushless motor is often a permanent magnet synchronous motor, but can also be a switched reluctance motor, or induction motor. There are many different configurations of BLDC motors and the most common type is the three phase motor due to its efficiency and low torque ripple./12/

BLDC motors have some significant advantages over the conventional brushed dc motors: /13/

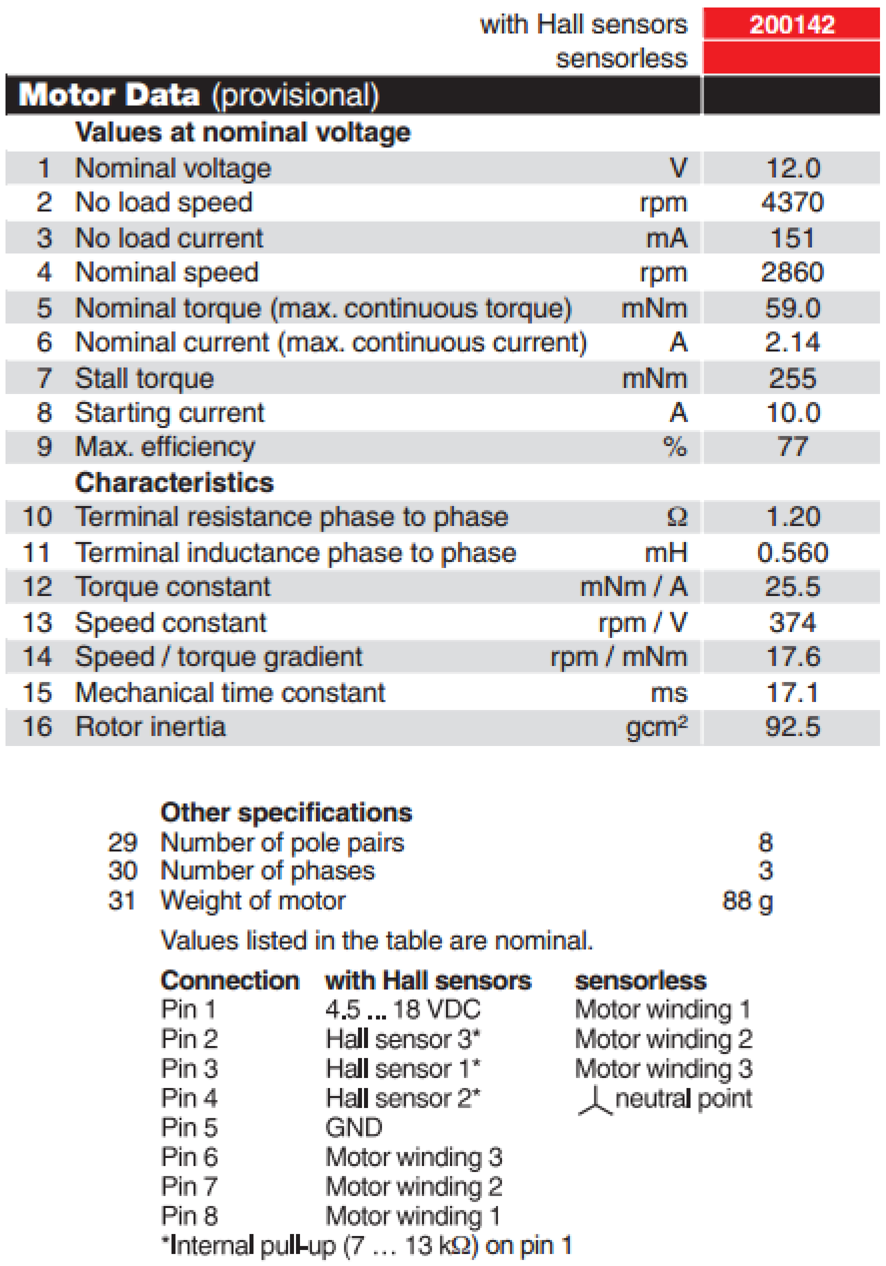
- Better speed vs torque characteristics
- Long operating life
- High efficiency
- Noiseless operation
- Higher speed range
- Higher torque-weight ratio
- High dynamic response

1.8 PID Controller

A proportional-integral-derivative controller (PID controller) is a control loop feedback mechanism (controller) widely used in industrial control systems (Programmable Logic Controllers, SCADA systems, Remote Terminal Units etc.). A PID controller calculates an "error" value as the difference between a measured process variable and a desired set point. The controller attempts to minimize the error in outputs by adjusting the process control inputs. /14/

2 MODELLING AND SIMULATION

Maxon EC 45 Flat BLDC motor 200142 is used on our robots and the main parameters of the motor are shown in Figure 5.



		with Hall sensors	200142
		sensorless	
Motor Data (provisional)			
	Values at nominal voltage		
1	Nominal voltage	V	12.0
2	No load speed	rpm	4370
3	No load current	mA	151
4	Nominal speed	rpm	2860
5	Nominal torque (max. continuous torque)	mNm	59.0
6	Nominal current (max. continuous current)	A	2.14
7	Stall torque	mNm	255
8	Starting current	A	10.0
9	Max. efficiency	%	77
	Characteristics		
10	Terminal resistance phase to phase	Ω	1.20
11	Terminal inductance phase to phase	mH	0.560
12	Torque constant	mNm / A	25.5
13	Speed constant	rpm / V	374
14	Speed / torque gradient	rpm / mNm	17.6
15	Mechanical time constant	ms	17.1
16	Rotor inertia	gcm^2	92.5

	Other specifications	
29	Number of pole pairs	8
30	Number of phases	3
31	Weight of motor	88 g

Values listed in the table are nominal.

Connection	**with Hall sensors**	**sensorless**
Pin 1	4.5 ... 18 VDC	Motor winding 1
Pin 2	Hall sensor 3*	Motor winding 2
Pin 3	Hall sensor 1*	Motor winding 3
Pin 4	Hall sensor 2*	⅄ neutral point
Pin 5	GND	
Pin 6	Motor winding 3	
Pin 7	Motor winding 2	
Pin 8	Motor winding 1	

*Internal pull-up (7 … 13 kΩ) on pin 1

Figure 5. Maxon EC 45 Flat datasheet/15/

2.1 BLDC Motor

2.1.1 Structure of BLDC Motor

In general, a brushed DC motor has a rotating armature and fixed magnetic field and its commutation is mechanical. However, usually a brushless DC motor is an inside-out DC motor, of which the armature is in the stator and the magnets are on the rotor, and uses the position sensors and an inverter to commutate electrically, which makes it a virtually maintenance-free motor. The main components of the BLDC motor are the stator and rotor where the armature winding and the permanent magnets are installed respectively. Figure 6 shows the construction of brushless DC motor./16/

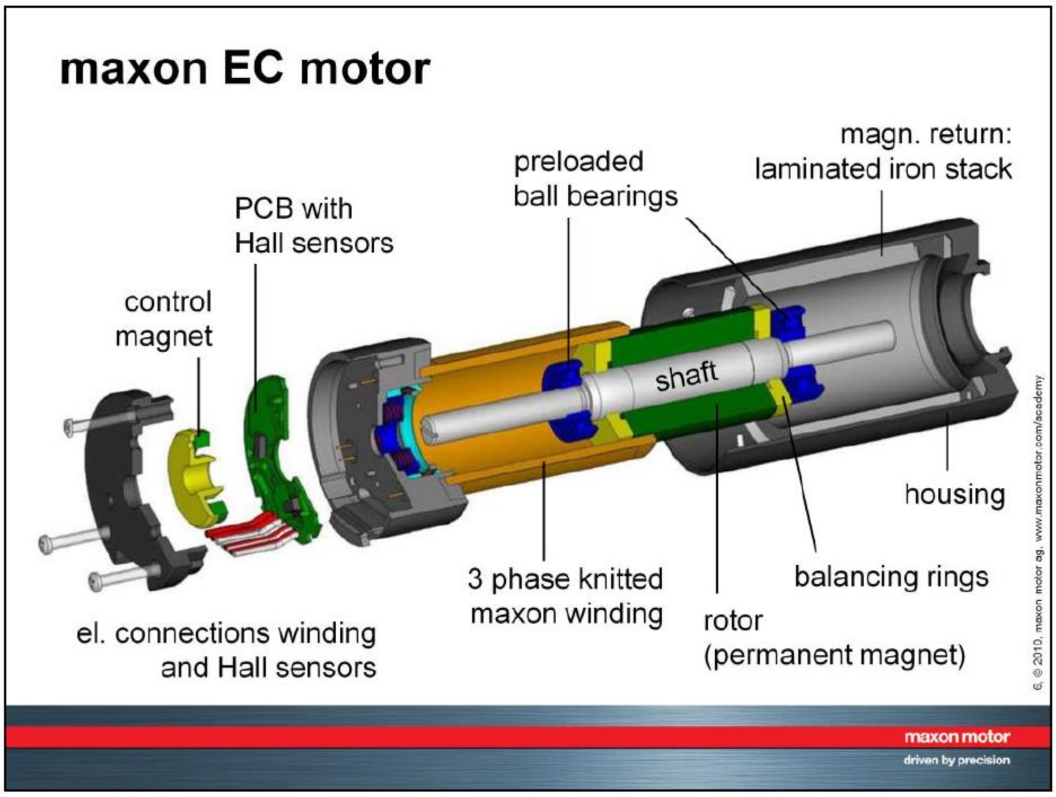

maxon EC motor
magn. return: laminated iron stack
preloaded ball bearings
PCB with Hall sensors
control magnet
shaft
housing
balancing rings
3 phase knitted maxon winding
rotor (permanent magnet)
el. connections winding and Hall sensors
6, © 2010, maxon motor ag, www.maxonmotor.com/academy
maxon motor
driven by precision

Figure 6. BLDC motor structure/17/

As we can see in Figure 6, the rotor with the permanent magnet mounted on the shaft at the centre. Dynamic balancing is achieved by removing material from the two balancing rings made of brass. Balancing is important for reducing vibration and noise and for increasing bearing life, particularly at the high speeds obtainable with brushless motors./17/

The stator contains the housing with the magnetic return. The magnetic return is made of a laminated iron stack in order to reduce the iron losses due to the rotating permanent magnet. Inside the iron stack we have the maxon winding, the three phases are contacted via the printed circuit board (PCB) to the electrical winding connections./17/

Rotor position feedback is often achieved by a system of three Hall sensor mounted on the PCB. The Hall sensors detect the magnetic field of a control magnet which is attached to the shaft. In some cases the magnetic field of the main permanent magnet is monitored directly. The Hall sensors have 5 additional electrical connections: 2 for the supply voltage and 3 for the Hall sensor signals./17/

2.1.2 Operation of BLDC Motor

The three phase BLDC motor is operated in a two-phased-on fashion, i.e. the two phases that produce the highest torque are energized while the third phase is off. Which two phases are energized depends on the rotor position. The signals form the position sensors produce a three digit number that changes every 60 ^{o}(electrical degrees) as shown in Figure 7. The figure also shows ideal current and back-emf waveforms./18/

Figure 8 shows a three-phase star-connected motor along with its phase energizing sequence. Each interval starts with the rotor and stator field lines 120 ^{o}apart and ends when they are 60 ^{o}apart. The maximum torque is reached when the field lines are perpendicular. The current commutation is done by a six-step inverter as shown in a simplified form in the figure. The switches are shown as bipolar junction transistors but MOSFET switches are more common. Table 1 shows the switching sequence, the current direction and the position sensor signals./18/

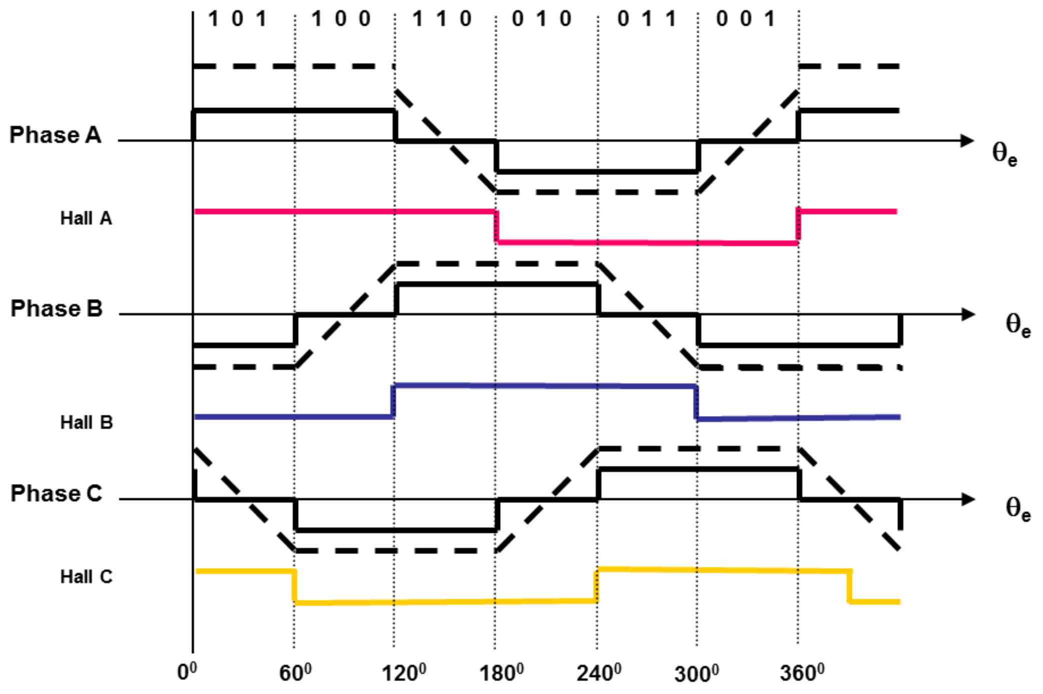


Figure 7. Phase voltage, current and hall-sensor waveforms with respect to rotor electrical angle/19/

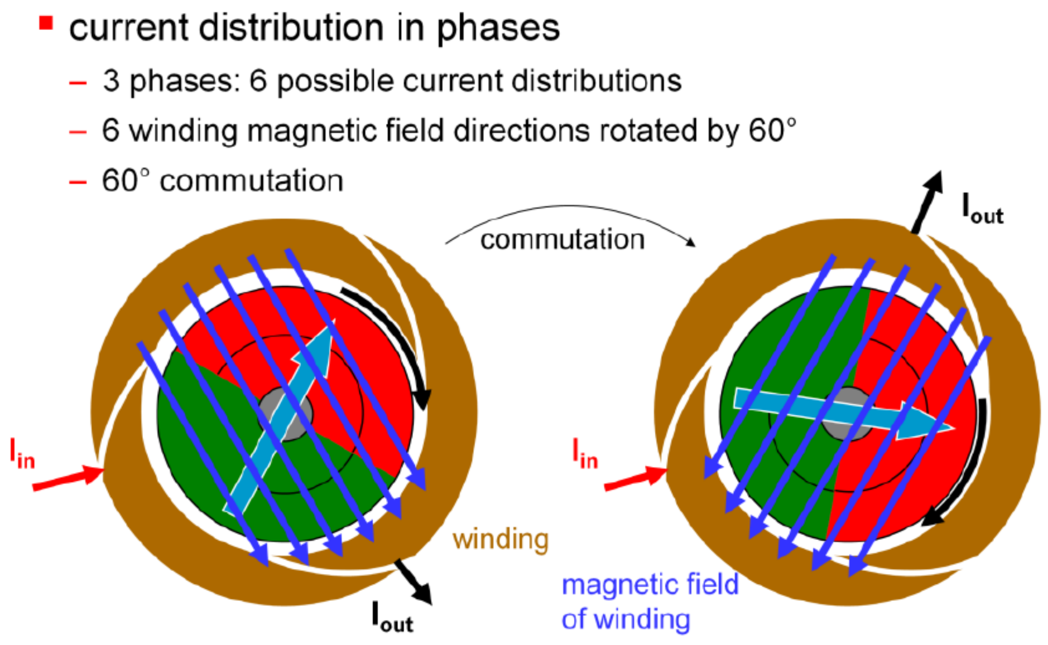

- current distribution in phases
  - 3 phases: 6 possible current distributions
  - 6 winding magnetic field directions rotated by 60°
  - 60° commutation

Figure 8. Interaction of rotor and stator/17/

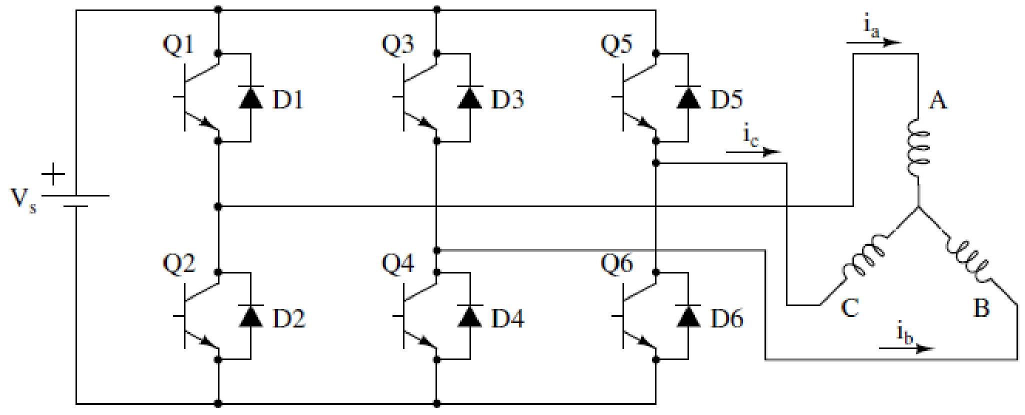


Figure 9. BLDC drive scheme (1)/18/

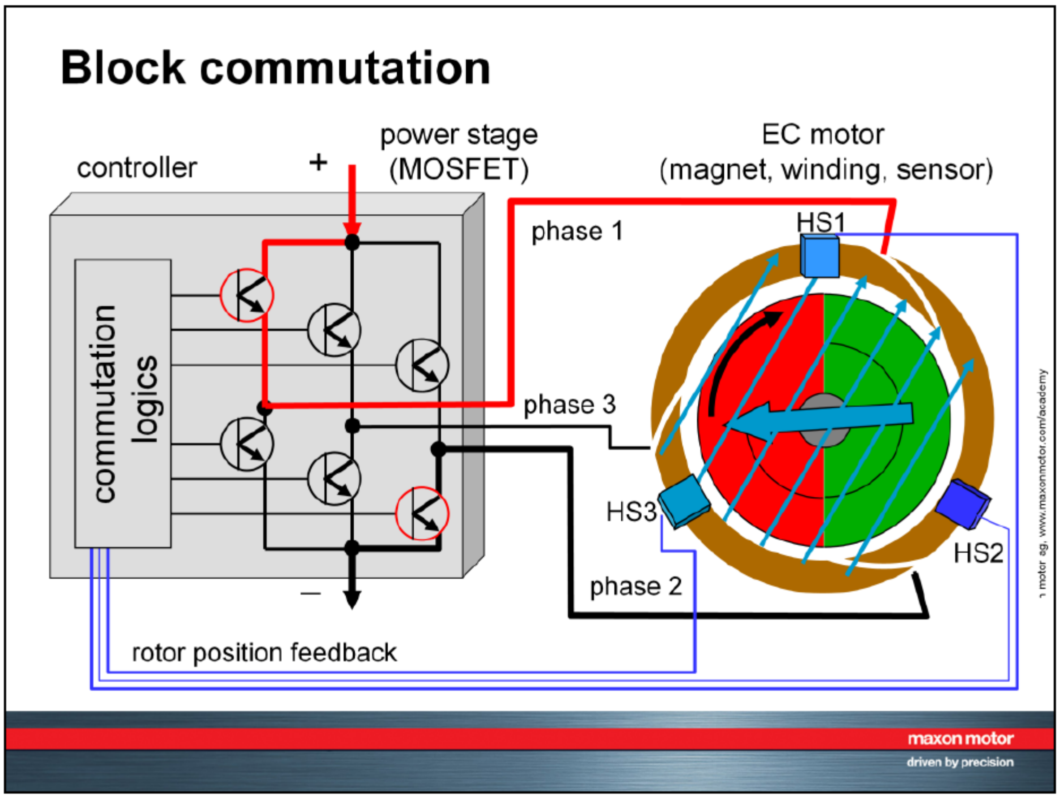


Figure 10. BLDC drive scheme (2)/17/

Table 1. Switching sequence

Electrical angle	Conductive phase	Hall sensors			Phase current			Switch closed	
		H1	H2	H3	Ia	Ib	Ic		
0 º-60 º	1	1	0	1	+	-	off	Q1	Q4
60 º-120 º	2	1	0	0	+	off	-	Q1	Q6
120 º-180 º	3	1	1	0	off	+	-	Q3	Q6
180 º-240 º	4	0	1	0	-	+	off	Q3	Q2
240 º-300 º	5	0	1	1	-	off	+	Q5	Q2
300 º-360 º	6	0	0	1	off	-	+	Q5	Q4

2.1.3 Mathematic Equations

The rotor and shaft are assumed to be rigid. Further, the model is assumed to be a viscous friction model. Therefore, the friction torque is proportional to the shaft angular velocity. Based on Newton's 2nd law and Kirchhoff's voltage law, the three phase star connected BLDC motor can be described by the following four governing equations:

$$T_e = b\dot{\theta}_m + J\ddot{\theta}_m + T_L \tag{1}$$

$$\mathrm{V}_{ab} = \mathrm{R}(\mathrm{i}_a - \mathrm{i}_b) + \mathrm{L}\frac{\mathrm{d}}{\mathrm{dt}}(\mathrm{i}_a - \mathrm{i}_b) + e_a - e_b \tag{2}$$

$$\mathrm{V}_{bc} = \mathrm{R}(\mathrm{i}_b - \mathrm{i}_c) + \mathrm{L}\frac{\mathrm{d}}{\mathrm{dt}}(\mathrm{i}_b - \mathrm{i}_c) + e_b - e_c \tag{3}$$

$$\mathrm{V}_{ca} = \mathrm{R}(\mathrm{i}_c - \mathrm{i}_a) + \mathrm{L}\frac{\mathrm{d}}{\mathrm{dt}}(\mathrm{i}_c - \mathrm{i}_a) + e_c - e_a \tag{4}$$

Here T_e, $\dot{\theta}_m$, b, J, T_L, V, R, I, L and e denote the electrical torque, the mechanical rotational speed, the viscous friction constant, the rotor inertia, the mechanical load torque, the phase-to-phase voltage, the phase resistance, the phase inductance and the phase back emf respectively. Notice that the voltage and current have following relationships:

$$\mathrm{V}_{ab} + \mathrm{V}_{bc} + \mathrm{V}_{ca} = 0 \tag{5}$$

$$\mathrm{i}_a + \mathrm{i}_b + \mathrm{i}_c = 0 \tag{6}$$

Therefore, in order to simply the modelling, only two voltage equations are needed. They can be derived as follows:

$$2\mathrm{V}_{ab} + \mathrm{V}_{bc} = 3\mathrm{Ri}_a + 3\mathrm{L}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{i}_a + 2e_a - e_b - e_c \tag{7}$$

$$-\mathrm{V}_{ab} + \mathrm{V}_{bc} = 3\mathrm{Ri}_b + 3\mathrm{L}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{i}_b + 2e_b - e_a - e_c \tag{8}$$

In general, the torque generated by a three phase star connected BLDC motor can also be expressed by the following equation:

$$T_e = (e_a\mathrm{i}_a + e_b\mathrm{i}_b + e_c\mathrm{i}_c)/\dot{\theta}_m \tag{9}$$

And the trapezoidal back emf can be expressed as:

$$e_a = \frac{k_e}{2}\dot{\theta}_m Tra(\theta_e) \tag{10}$$

$$e_b = \frac{k_e}{2}\dot{\theta}_m Tra(\theta_e - \frac{2}{3}\pi) \tag{11}$$

$$e_c = \frac{k_e}{2}\dot{\theta}_m Tra(\theta_e - \frac{4}{3}\pi) \tag{12}$$

Here, k_e is the back emf constant and θ_e is the electrical angle which is equal to the mechanical angle times the number of pole pairs ($\theta_e = p\theta_m$). Tra(θ_e) is the trapezoidal waveform function and one period of the function can be described as follow:

$$Tra(\theta_e) = \begin{cases} 1, \quad 0 \leq \theta_e < \frac{2}{3}\pi \\ 1 - \frac{6}{\pi}\left(\theta_e - \frac{2}{3}\pi\right), \quad \frac{2}{3}\pi \leq \theta_e < \pi \\ -1, \quad \pi \leq \theta_e < \frac{5}{3}\pi \\ -1 + \frac{6}{\pi}(\theta_e - \frac{5}{3}\pi), \quad \frac{5}{3}\pi \leq \theta_e < 2\pi \end{cases} \tag{13}$$

Further, we can simplify Equation 5 using Equation 6, 7, 8, and we can derive the following equation which makes the modelling more convenient:

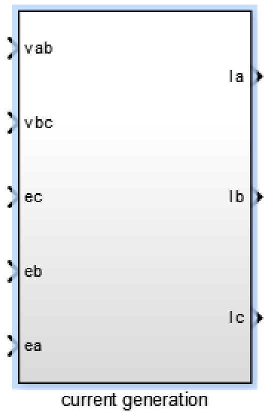
$$T_e = \frac{k_e}{2}(Tra(\theta_e)\mathrm{i}_a + Tra(\theta_e - \frac{2}{3}\pi)\mathrm{i}_b + Tra(\theta_e - \frac{4}{3}\pi)\mathrm{i}_c) \tag{14}$$

2.1.4 Simulink Model

According to the mathematical equations derived above, we can build the model of the BLDC using Simulink.

2.1.4.1 Current Generation Subsystem

According to the voltage equations (Equation 7, Equation 8) the current generation subsystem can be built as shown in Figure 11 and Figure 12. It takes in the phase-to-phase voltage Vab, Vbc, and back emfs ea, eb, ec, and generates the phase currents ia, ib and, ic.



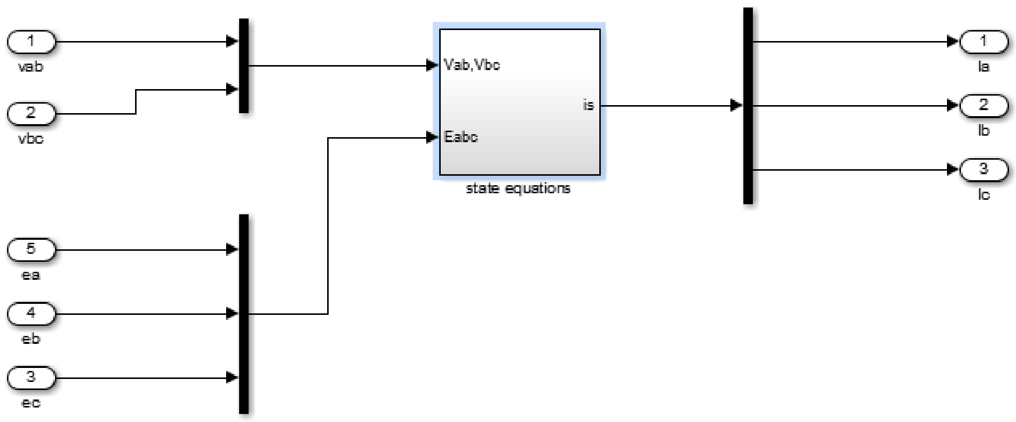


Figure 11. Current generation subsystem

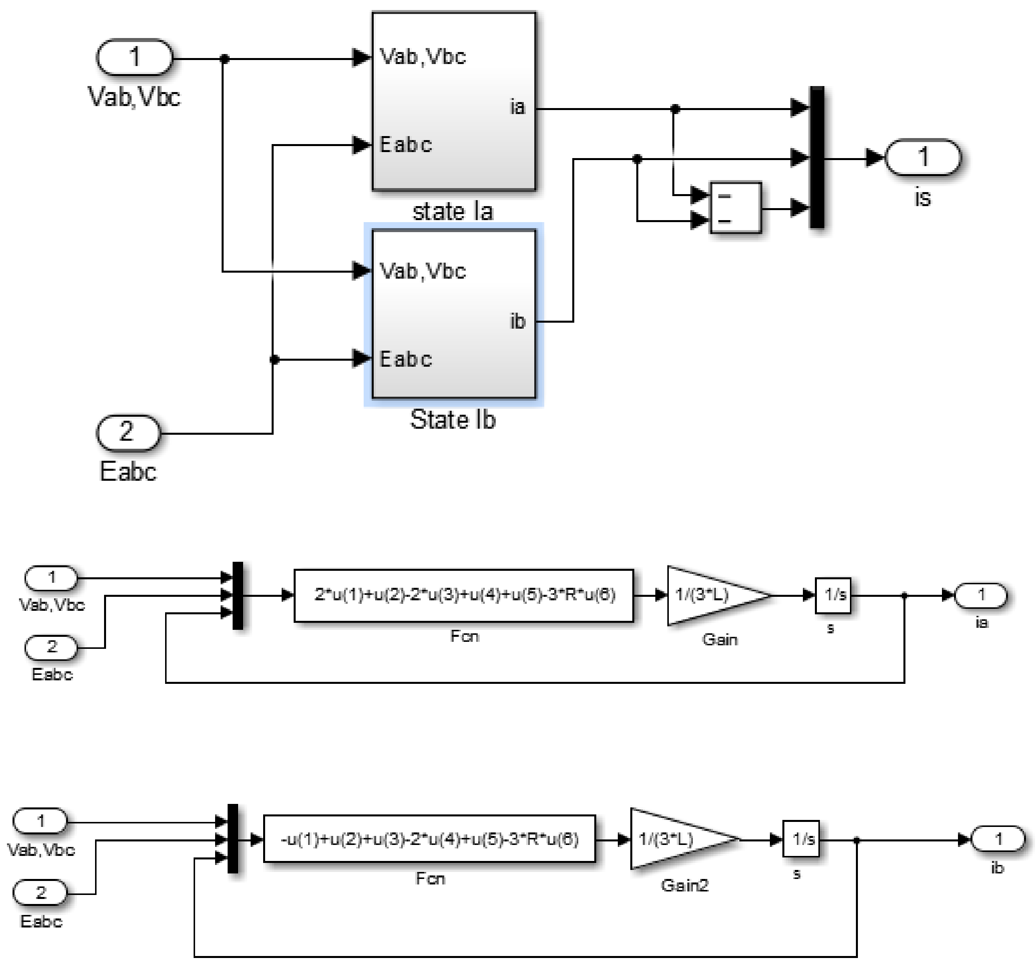


Figure 12. State ia and ib

2.1.4.2 EMF Generation Subsystem

The back emf waveform is a trapezoidal waveform which is a periodic function, therefore using a look-up table is a good way to represent it. The emf generation subsystem takes in the electrical angle theta e and the mechanical rotational speed and outputs the back emfs ea, eb, ec and the waveform function tra' which equals to the function Tra times half of the torque constant Ke ($Tra'(\theta_e) = \frac{k_e}{2} Tra(\theta_e)$).

Notice that it is easy to deduce that the motor torque and back emf constants are equal in SI units (Kt = Ke). Thus, only one parameter K can be used to represent both the back emf constant and the motor torque constant.

The details of the subsystem are shown in Figure 13 and Figure 14.

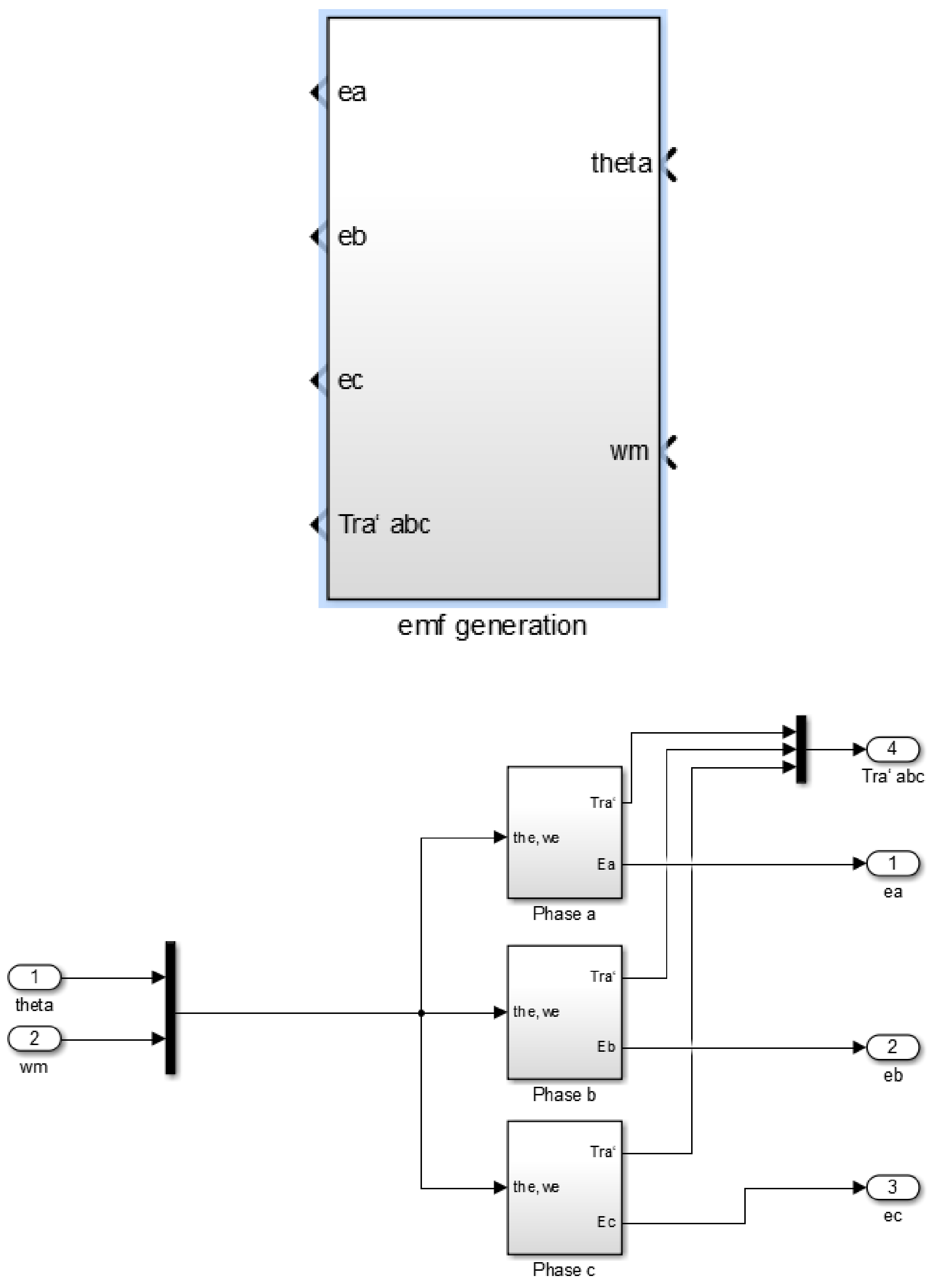


Figure 13. EMF generation subsystem

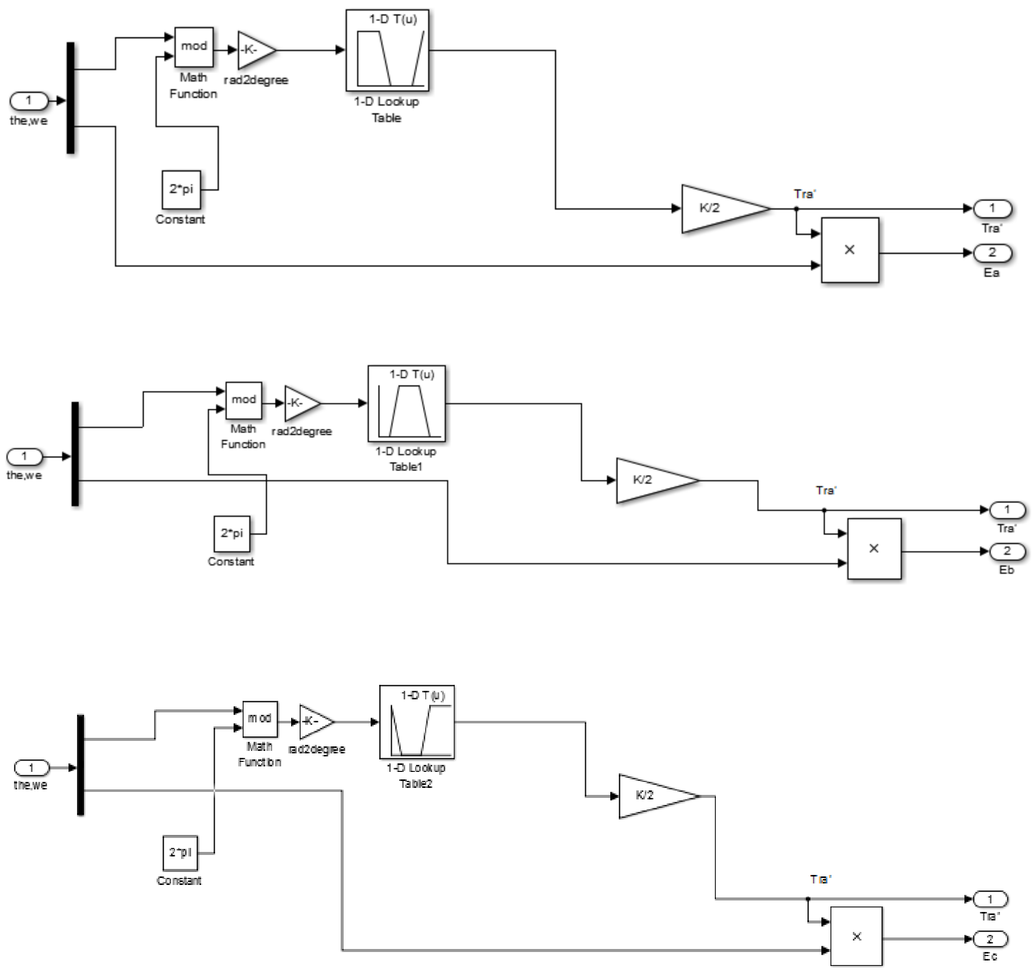


Figure 14. Phase a, phase b and phase c

2.1.4.3 Mechanical Subsystem

Based on the motion equation and torque equation, the mechanical subsystem was built. It takes in the phase currents, tra' waveforms, mechanical load torque and generates the electrical angle theta e, rotational speed wm, and the electrical torque te. The subsystem is shown in figure 15.

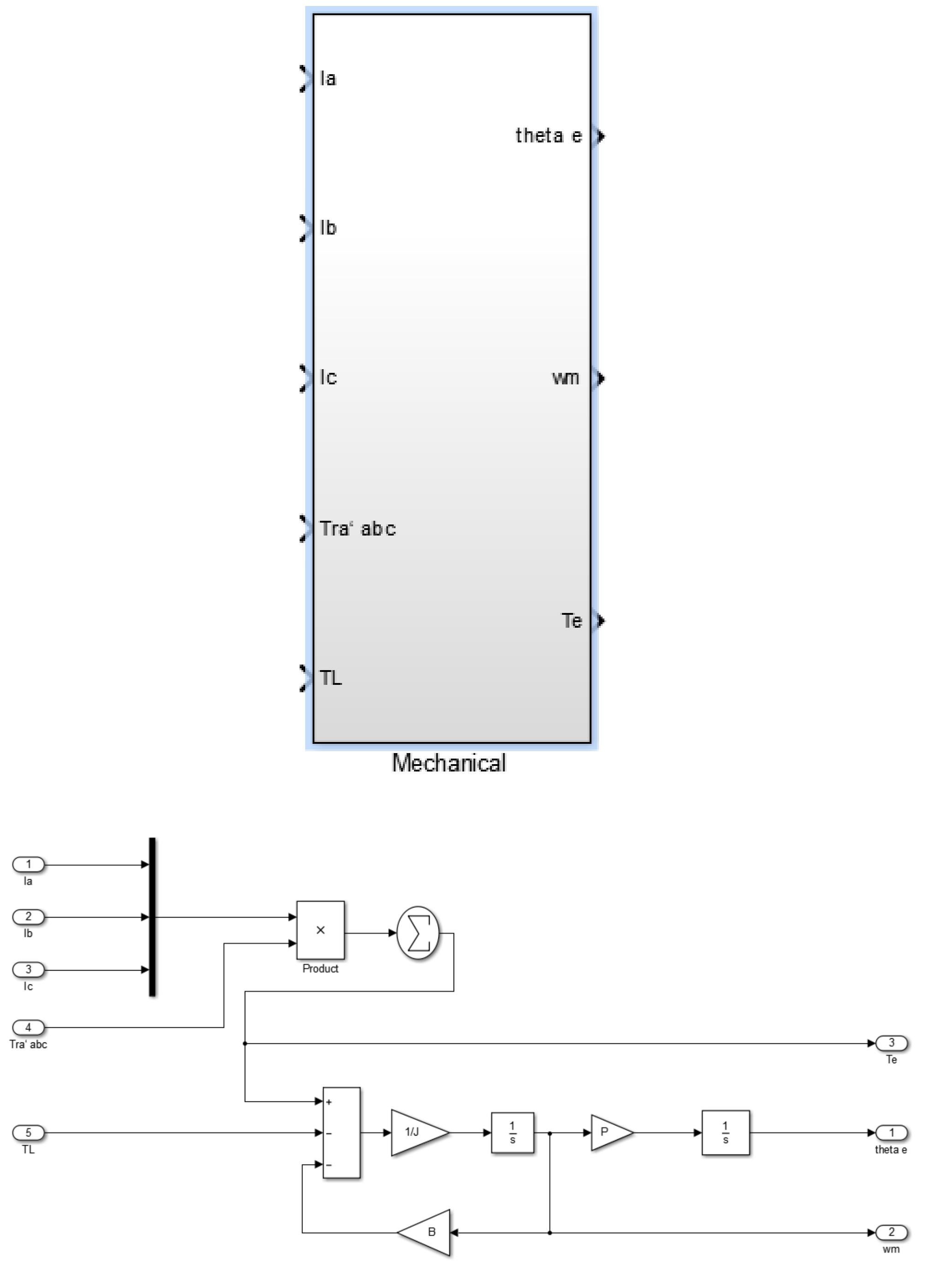
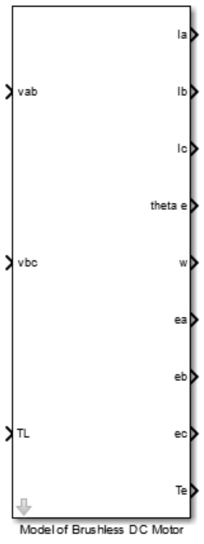
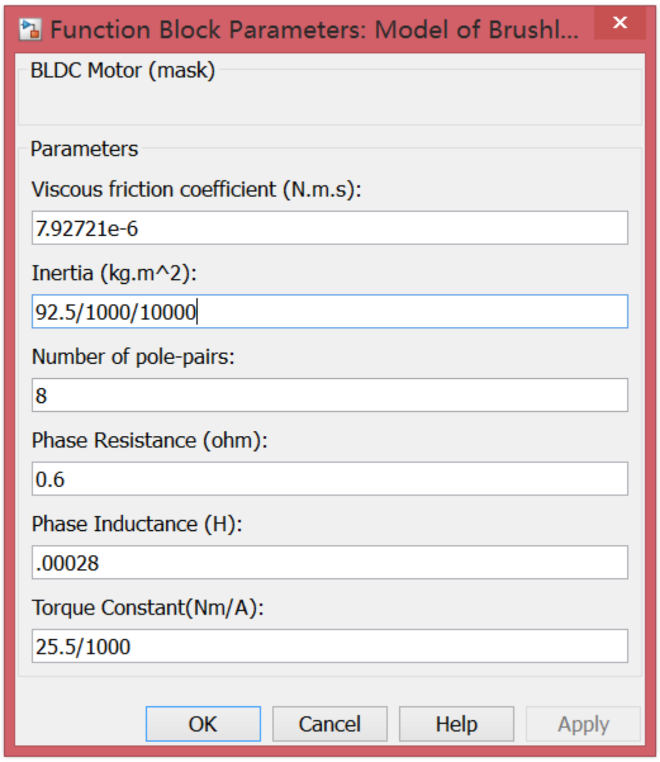


Figure 15. Mechanial subsystem

Now, the main parts of the BLDC motor are completed. We can make the connection between the subsystems and mask the BLDC motor block. The overall structure is shown in Firgure 16.




Function Block Parameters: Model of Brushl...

BLDC Motor (mask)

Parameters

Viscous friction coefficient (N.m.s):

7.92721e-6

Inertia (kg.m^2):

92.5/1000/10000

Number of pole-pairs:

8

Phase Resistance (ohm):

0.6

Phase Inductance (H):

.00028

Torque Constant(Nm/A):

25.5/1000

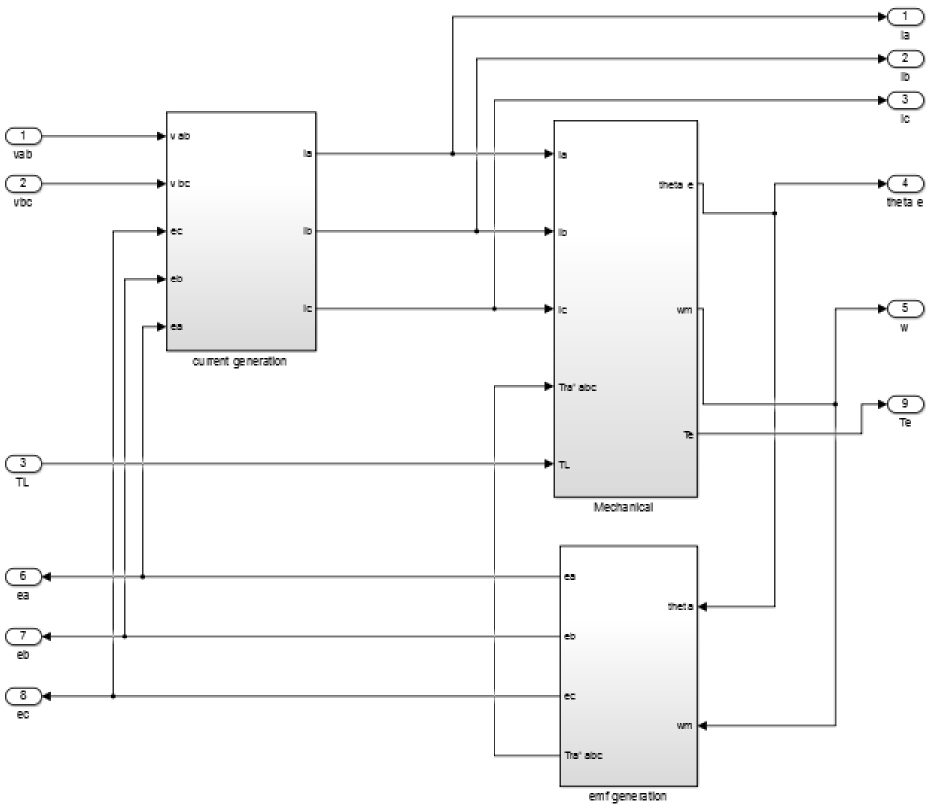


Figure 16. Model of BLDC motor

2.1.5 Commutation and Inverter

In order to make the BLDC run, we also need hall sensors to detect the position, a controller to implement the commutation logic and an inverter to output the phase-to-phase voltage and power the BLDC motor. Here, to simplify the construction, we use only one block to implement all the functions.

One thing that needs to be noticed is that the output voltages of the inverter is not all equal to the DC source voltage or 0, which powers the inverter. In practice, we do not actually need to know the values of phase-to-phase voltages, but in our model, we need to know the phase-to-phase voltages Vab and Vbc all the time.

To simplify, we do not consider the situations that there are switching signals during each interval. Figure 17 shows the equivalent circuit of the circuit shown in Figure 9 in each interval when the motor are rotating according to the sequence in Table 1. The switches and the diodes are assumed to be ideal devices.

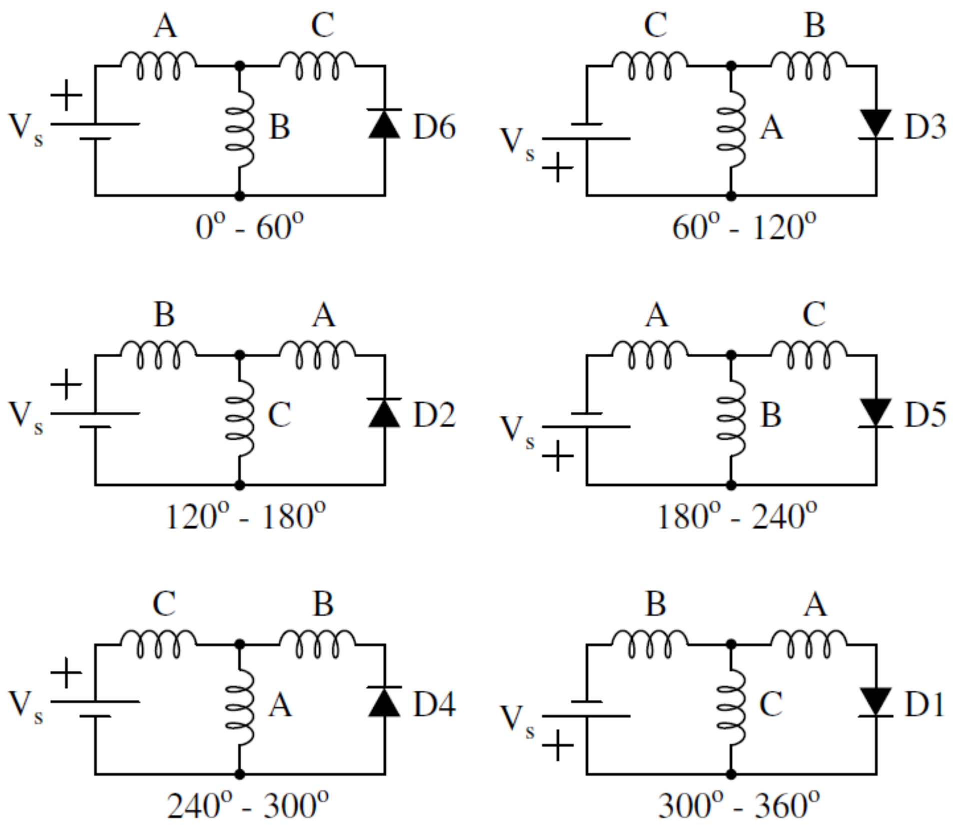


Figure 17. Circuit configuration in each interval/18/

When a phase is turned off, its phase current will go through a freewheeling path established through a diode and decay to zero. When the diode current is not zero, the diode voltage is 0. And when the diode current reaches zero, the diode voltage of diode should be equal to a value which can force the current to remain at zero in this interval. For example, the equivalent circuit of the first interval is shown in Figure 18 and we can calculate the Vab and Vbc by analysing the circuit.

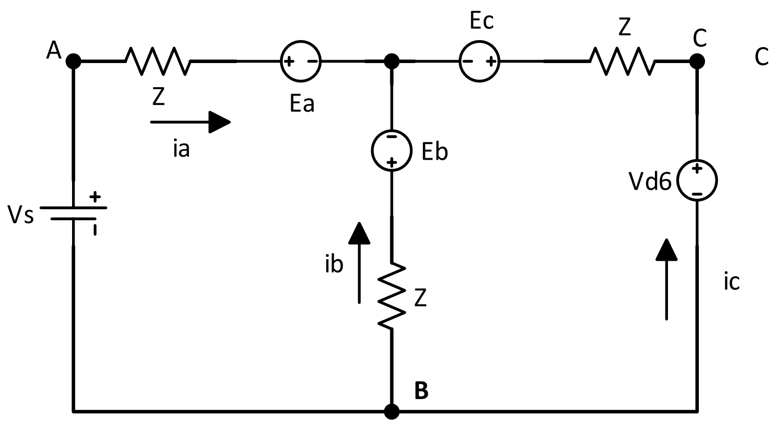


Figure 18. Circuit configuration in the first interval

It is obvious that the output voltages of the inverter depend on not only the DC source voltage, but also the back emfs and phase currents. The values of inverter voltages in all interval are calculated and shown in Table 2.

Table 2. Inverter output voltages (1)

Electrical angle	Diode current	V_{ab}	V_{bc}
0 º-60 º	$i_c > 0$ $i_c \leq 0$	V_s V_s	0 $\frac{1}{2}(-V_s + e_a + e_b - 2e_c)$
60 º-120 º	$i_b < 0$ $i_b \geq 0$	0 $\frac{1}{2}(V_s + e_a - 2e_b + e_c)$	V_s $\frac{1}{2}(V_s - e_a + 2e_b - e_c)$

120 º-180 º	$i_a > 0$ $i_a \leq 0$	$-\mathrm{V}_s$ $\frac{1}{2}(-\mathrm{V}_s + 2\mathrm{e}_a - \mathrm{e}_b - \mathrm{e}_c)$	V_s V_s
180 º-240 º	$i_c < 0$ $i_c \geq 0$	$-\mathrm{V}_s$ $-\mathrm{V}_s$	0 $\frac{1}{2}(\mathrm{V}_s + \mathrm{e}_a + \mathrm{e}_b - 2\mathrm{e}_c)$
240 º-300 º	$i_b > 0$ $i_b \leq 0$	0 $\frac{1}{2}(-\mathrm{V}_s + \mathrm{e}_a - 2\mathrm{e}_b + \mathrm{e}_c)$	$-\mathrm{V}_s$ $\frac{1}{2}(-\mathrm{V}_s - \mathrm{e}_a + 2\mathrm{e}_b - \mathrm{e}_c)$
300 º-360 º	$i_a < 0$ $i_a \geq 0$	V_s $\frac{1}{2}(\mathrm{V}_s + 2\mathrm{e}_a - \mathrm{e}_b - \mathrm{e}_c)$	$-\mathrm{V}_s$ $-\mathrm{V}_s$

Sometimes the armature current may be negative, for example, when the source voltage decreases to a lower voltage. In this situation, the circuit configuration is different and therefore, the values of inverter voltages in all interval are also different and the value are calculated and shown in Table 3.

Table 3. Inverter output voltages (2)

Electrical angle	Diode current	V_{ab}	V_{bc}
0 º-60 º	$i_c < 0$ $i_c \geq 0$	V_s V_s	$-\mathrm{V}_s$ $\frac{1}{2}(-\mathrm{V}_s + \mathrm{e}_a + \mathrm{e}_b - 2\mathrm{e}_c)$
60 º-120 º	$i_b > 0$ $i_b \leq 0$	V_s $\frac{1}{2}(\mathrm{V}_s + \mathrm{e}_a - 2\mathrm{e}_b + \mathrm{e}_c)$	0 $\frac{1}{2}(\mathrm{V}_s - \mathrm{e}_a + 2\mathrm{e}_b - \mathrm{e}_c)$
120 º-180 º	$i_a < 0$ $i_a \geq 0$	0 $\frac{1}{2}(-\mathrm{V}_s + 2\mathrm{e}_a - \mathrm{e}_b - \mathrm{e}_c)$	V_s V_s

180 º-240 º	$i_c > 0$ $i_c \leq 0$	$-\mathrm{V}_s$ $-\mathrm{V}_s$	V_s $\frac{1}{2}(\mathrm{V}_s + \mathrm{e}_a + \mathrm{e}_b - 2\mathrm{e}_c)$
240 º-300 º	$i_b < 0$ $i_b \geq 0$	$-\mathrm{V}_s$ $\frac{1}{2}(-\mathrm{V}_s + \mathrm{e}_a - 2\mathrm{e}_b + \mathrm{e}_c)$	0 $\frac{1}{2}(-\mathrm{V}_s - \mathrm{e}_a + 2\mathrm{e}_b - \mathrm{e}_c)$
300 º-360 º	$i_a > 0$ $i_a \leq 0$	0 $\frac{1}{2}(\mathrm{V}_s + 2\mathrm{e}_a - \mathrm{e}_b - \mathrm{e}_c)$	$-\mathrm{V}_s$ $-\mathrm{V}_s$

To implement the commutation logic and output correct values of Vab and Vbc, the following MATLAB function is used in our model. It takes in the position, source voltage, back emfs, and phase currents, and outputs the phase-to-phase supply voltage.

```
function y=commutation_inverter(pos,Vs,ea,eb,ec,Ia,Ib,Ic)
if (pos>=0)&&(pos<60)&&(Ic<=0)&&(Ib<=0)
   y = [Vs,0.5*(-Vs+ea+eb-2*ec)];
elseif (pos>=0)&&(pos<60)&&(Ic>0)&&(Ib<=0)
    y = [Vs,0];
elseif (pos>=0)&&(pos<60)&&(Ic>=0)&&(Ib>0)
   y = [Vs,0.5*(-Vs+ea+eb-2*ec)];
elseif (pos>=0)&&(pos<60)&&(Ic<0)&&(Ib>0)
    y = [Vs,-Vs];
elseif(pos>=60)&&(pos<120)&&(Ib>=0)&&(Ia>=0)
  y=[0.5*(Vs+ea-2*eb+ec),0.5*(Vs-ea+2*eb-ec)];
elseif(pos>=60)&&(pos<120)&&(Ib<0)&&(Ia>=0)
  y=[0,Vs];
elseif(pos>=60)&&(pos<120)&&(Ib<=0)&&(Ia<0)
  y=[0.5*(Vs+ea-2*eb+ec),0.5*(Vs-ea+2*eb-ec)];
elseif(pos>=60)&&(pos<120)&&(Ib>0)&&(Ia<0)
  y=[Vs,0];
elseif(pos>=120)&&(pos<180)&&(Ia<=0)&&(Ic<=0)
  y=[0.5*(-Vs+2*ea-eb-ec),Vs];
elseif(pos>=120)&&(pos<180)&&(Ia>0)&&(Ic<=0)
  y=[-Vs,Vs];
elseif(pos>=120)&&(pos<180)&&(Ia>=0)&&(Ic>0)
  y=[0.5*(-Vs+2*ea-eb-ec),Vs];
elseif(pos>=120)&&(pos<180)&&(Ia<0)&&(Ic>0)
  y=[0,Vs];
elseif(pos>=180)&&(pos<240)&&(Ic>=0)&&(Ib>=0)
  y = [-Vs,0.5*(Vs+ea+eb-2*ec)];
elseif(pos>=180)&&(pos<240)&&(Ic<0)&&(Ib>=0)
  y = [-Vs,0];
elseif(pos>=180)&&(pos<240)&&(Ic<=0)&&(Ib<0)
  y = [-Vs,0.5*(Vs+ea+eb-2*ec)];
elseif(pos>=180)&&(pos<240)&&(Ic>0)&&(Ib<0)
```

```
  y = [-Vs,Vs];
elseif(pos>=240)&&(pos<300)&&(Ib<=0)&&(Ia<=0)
  y=[0.5*(-Vs+ea-2*eb+ec),0.5*(-Vs-ea+2*eb-ec)];
elseif(pos>=240)&&(pos<300)&&(Ib>0)&&(Ia<=0)
  y=[0,-Vs];
elseif(pos>=240)&&(pos<300)&&(Ib>=0)&&(Ia>0)
  y=[0.5*(-Vs+ea-2*eb+ec),0.5*(-Vs-ea+2*eb-ec)];
elseif(pos>=240)&&(pos<300)&&(Ib<0)&&(Ia>0)
  y=[-Vs,0];
elseif(pos>=300)&&(pos<360)&&(Ia>=0)&&(Ic>=0)
 y=[0.5*(Vs+2*ea-eb-ec),-Vs];
elseif(pos>=300)&&(pos<360)&&(Ia<0)&&(Ic>=0)
 y=[Vs,-Vs];
elseif(pos>=300)&&(pos<360)&&(Ia<=0)&&(Ic<0)
 y=[0.5*(Vs+2*ea-eb-ec),-Vs];
elseif(pos>=300)&&(pos<360)&&(Ia>0)&&(Ic<0)
 y=[0,-Vs];
end
```

The whole subsystem thus can be modelled as shown in Figure 19.

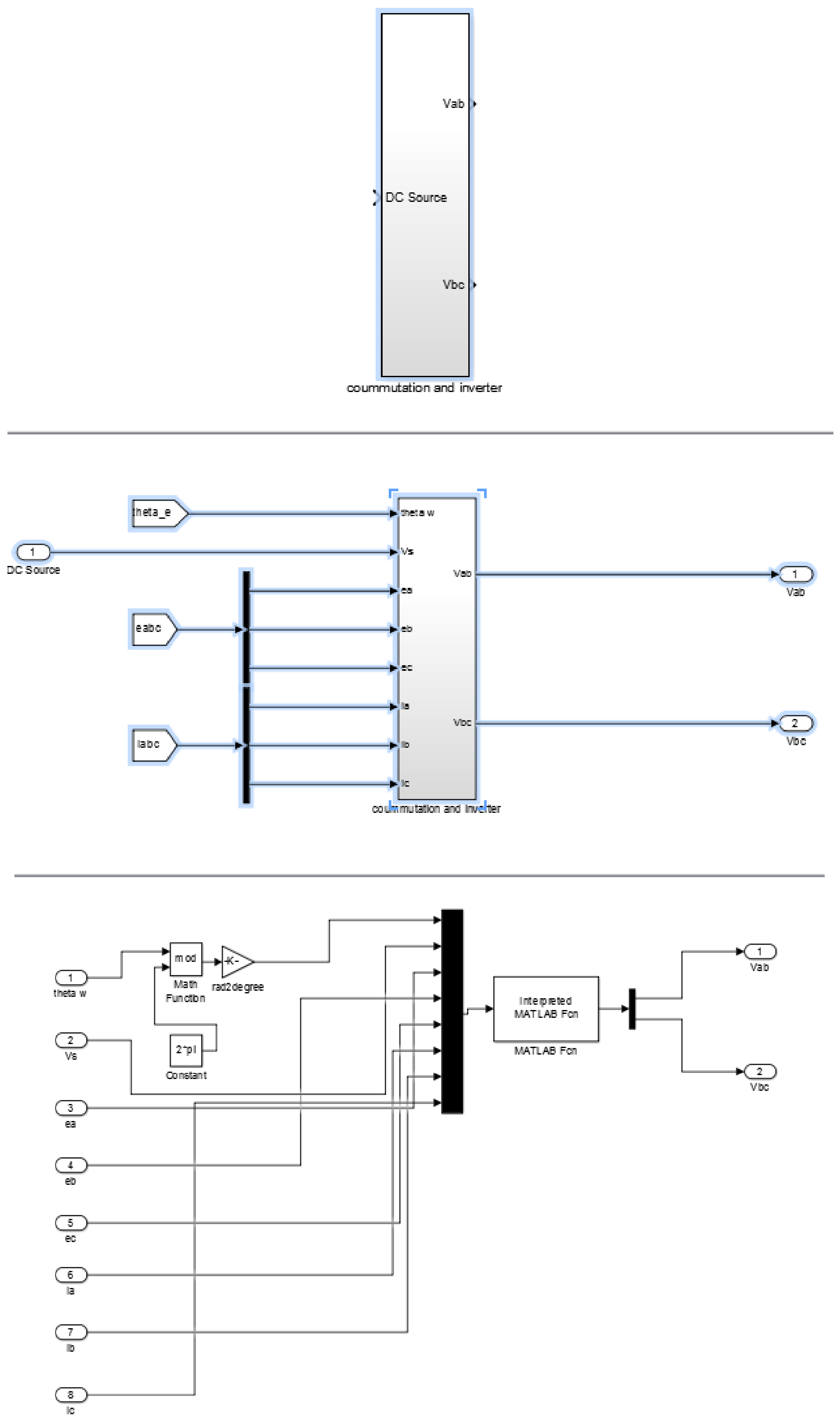


Figure 19. Commutation and inverter subsystem

Now the BLDC drive model can be finished as shown in Figure 20.

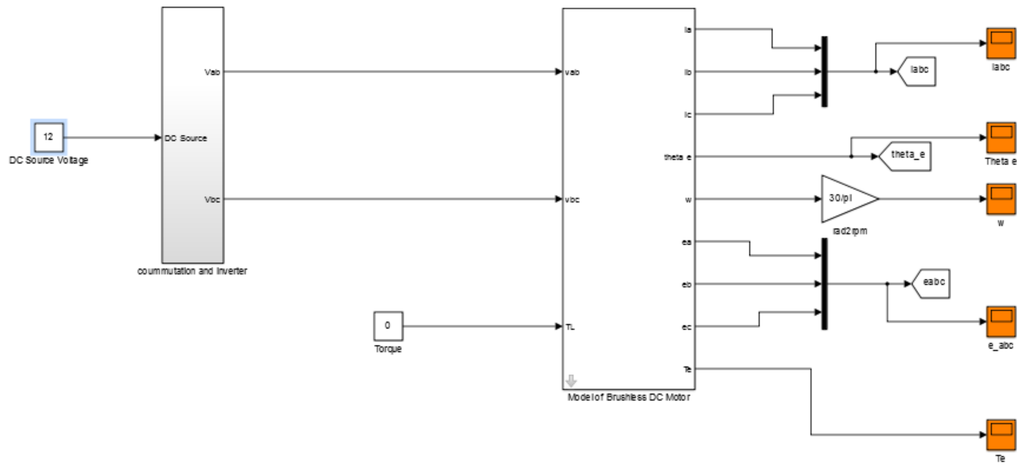


Figure 20. BLDC drive model

2.1.6 Simscape Model

There is an alternative way to build the brushless DC motor model with the physical modelling blocks of the Simscape library:

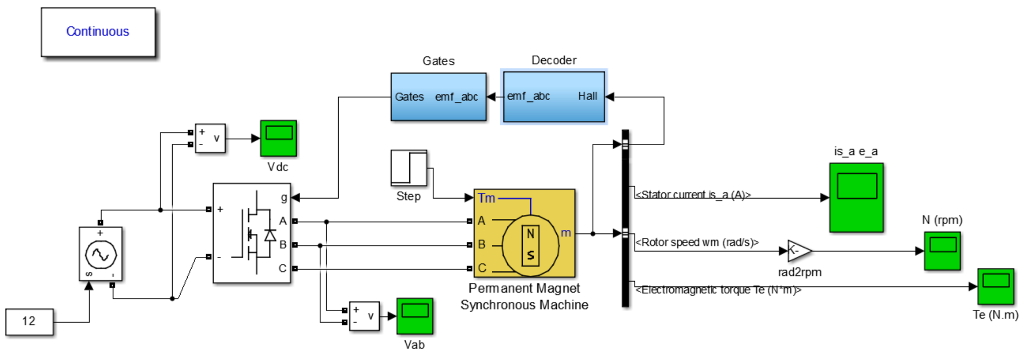


Figure 21. Simscape BLDC motor model

2.1.7 Equivalent DC Motor Model

The BLDC model built above is reasonable and accurate, however it is somewhat complex and it takes much time to run the simulation. In comparison, a DC motor has a relatively simple model and similar properties to a BLDC motor when having the same parameters. In Chapter 4, when modelling the motion control system, we need use an equivalent DC motor model to replace the BLDC motor model to greatly reduce the simulation time.

Figure 8 shows a simple model of the motor, the electric equivalent circuit of the armature and the diagram of the rotor.

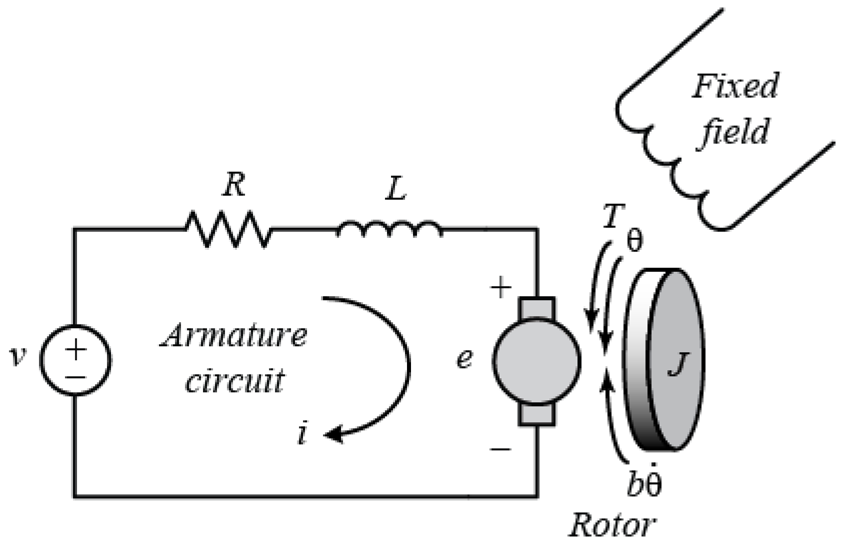


Figure 22. DC motor equivalent circuit /20/

From the figure above, the motor can be described by the following two governing equations based on Newton's 2nd law and Kirchhoff's voltage law:

$$T_e = b\dot{\theta} + J\ddot{\theta} + T_L \tag{15}$$

$$\mathrm{V} = \mathrm{Ri} + \mathrm{L}\frac{\mathrm{di}}{\mathrm{dt}} + e \tag{16}$$

Here T_e, $\dot{\theta}$, b, J, T_L, V, R, I, L and e denote the electrical torque, the rotational speed, the viscous friction constant, the rotor inertia, the mechanical load torque, the DC source voltage, the armature resistance, the armature inductance and the back emf respectively.

Usually, the torque generated by a DC motor is proportional to the armature current and the strength of the magnetic field. In ideal situation, the magnetic field is constant, so that the motor torque is proportional to only the armature current i by a constant factor Kt. In addition, the back emf, e, is proportional to the angular velocity of the shaft by a constant factor Ke. So we can derive the following equations (notice that Kt=Ke):

$$T_e = K_t i \tag{17}$$

$$\mathrm{e} = K_e \dot{\theta} \tag{18}$$

Therefore, the model can be built as shown in Figure 23.

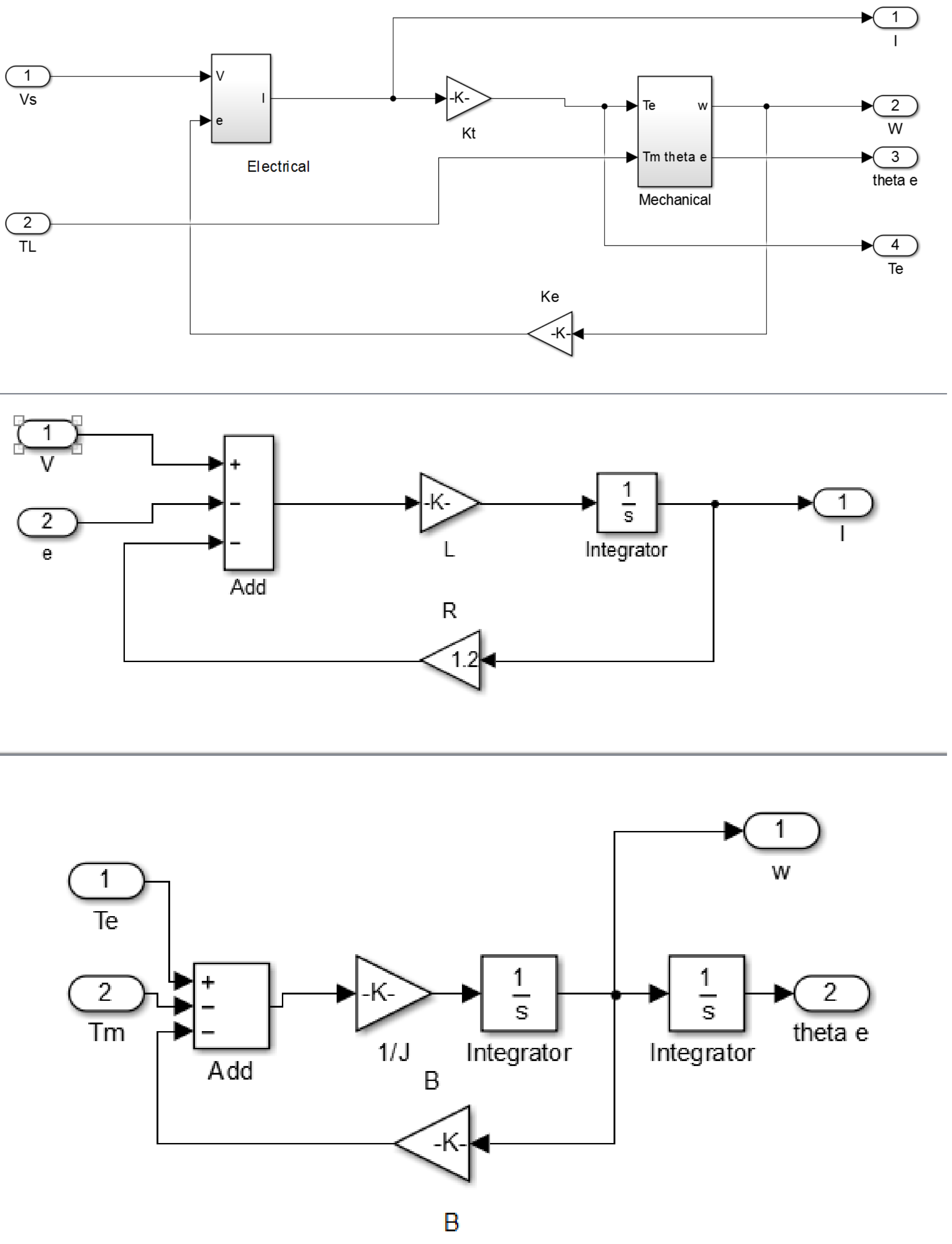


Figure 23. Equivalent DC motor model

2.2 Modelling of Single BLDC Motor Control System

As part of the Model-Based Design procedure to design the single BLDC Motor control system, we need to design and model the whole system with Simulink.

Figure 24 shows the whole model:

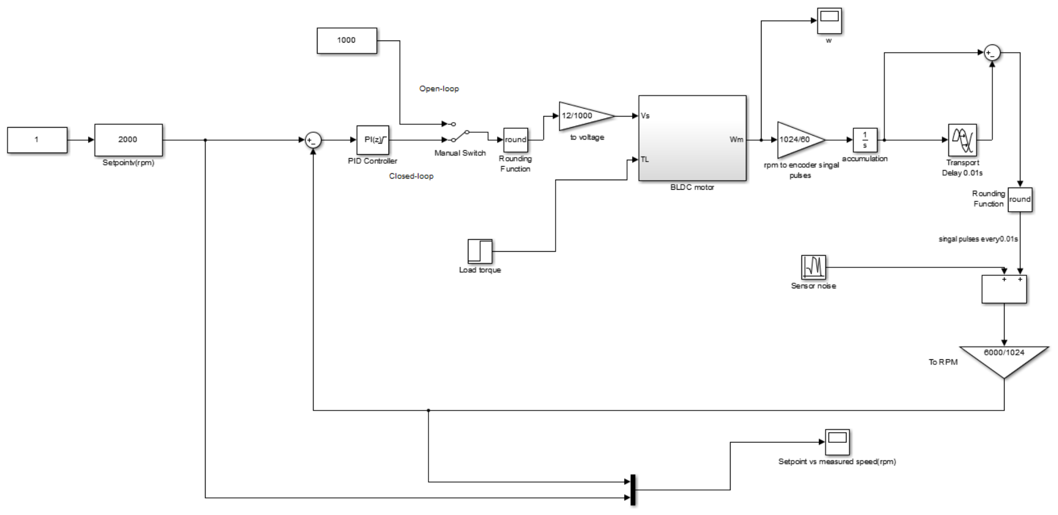


Figure 24. Model of BLDC motor control system

Based on the BLDC drive model shown is Figure 20, a PID controller is used for speed control to process the error between the set point and the feedback speed, so that the voltage into the motor is adjusted by the PID controller.

Here the PID controller is actually a PI controller. The sample time of the PI controller is 10ms which means the algorithm of the control loop is implemented every 10ms. The clamping method is used as the anti-windup method.

In practice, the average value of the input voltage to the motor is controlled with the Pulse Width Modulation (PWM) technique by periodically opening and closing the switches. In the pulse width modulation the switches are turned on at a constant switching frequency. The total time period of one cycle of output waveform is constant. The average input voltage is directly proportional to the ON time of chopper. The ratio of ON time to total time is defined as duty cycle.

Usually, the output of the PI controller is used to adjust the duty cycle of the PWM signal. In this model, we omit the PWM signal and directly use the Pi controller to adjust the average input voltage to the motor to obtain a simplified model.

In our design, the resolution of the PWM is chosen to be 1000, so the upper saturation limit and lower saturation limit are 1000 and 0 respectively and the output is rounded before converted to voltage. In addition, a switch is used so we can choose between open-loop and closed-loop.

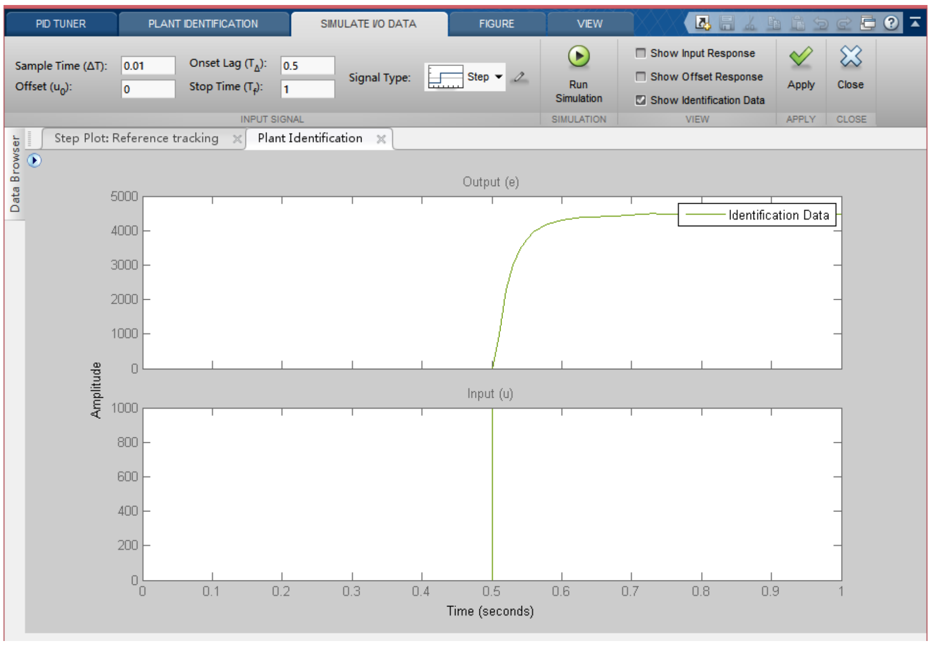
The set point is the rotational speed in rpm. The DC source voltage to the motor driver is 12V, so here we use 12V as the maximum voltage and we convert PI controller output to the correspondent voltage with a gain block.

In the simulation, the rotational speed can be directly measured, but in practice, the speed needs to be measured by counting the pulses generated by the encoder. Therefore, we need to convert the speed to the counted number of the pulses to simulate what will happen in the real situation. The resolution of the encoder used here is 1024, which means it generates 1024 pulses per revolution.

Then, the next step is to calculate the rotational speed of the motor. We calculate the number of the pulses in every 10ms, and the encoder noise is also taken into consideration and added, then the number of the pulses times 6000/1024 is the measured speed. The measured speed is calculated every 10ms which is also the sample time of the closed-loop control.

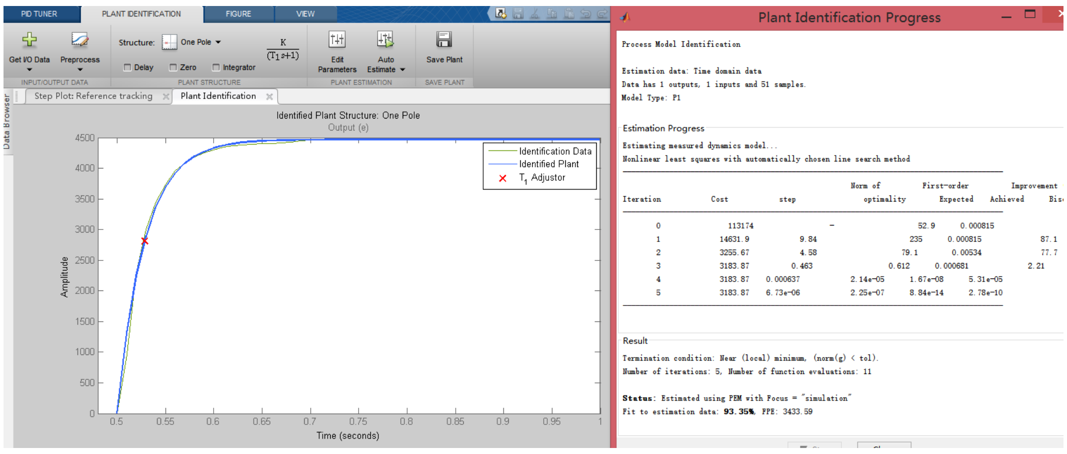
After completing to build the closed loop control structure, the next step is significant. To achieve reasonable, reliable and robust control performance, finding the optimal parameters of the PI controller is vital. Traditionally, PID controllers are tuned either manually or using rule-based methods, therefore finding the set of gains that ensures the best performance of the control system is a complex task and time-consuming. However, In MATLAB/Simulink, PID Tuner provides a fast and widely applicable single-loop PID tuning method for the Simulink PID Controller blocks. With this method, we can tune PID controller parameters to achieve a robust design with the desired response time.

We can tune the parameters by clicking the Tune button and the MATLAB will automatically compute the linearized model. Due to an interrupted MATLAB function used in commutation and inverter subsystem, the automatic linearization cannot be done. Therefore, we need to identify the plant using the input and output data obtained by simulating the model (shown in Figure 25 and Figure 26). The parameters can be tuned by manually adjusting design criteria and choose optimal parameters (shown in Figure 27). These parameter are used in the next chapter for the control system run on Arduino and DE2.



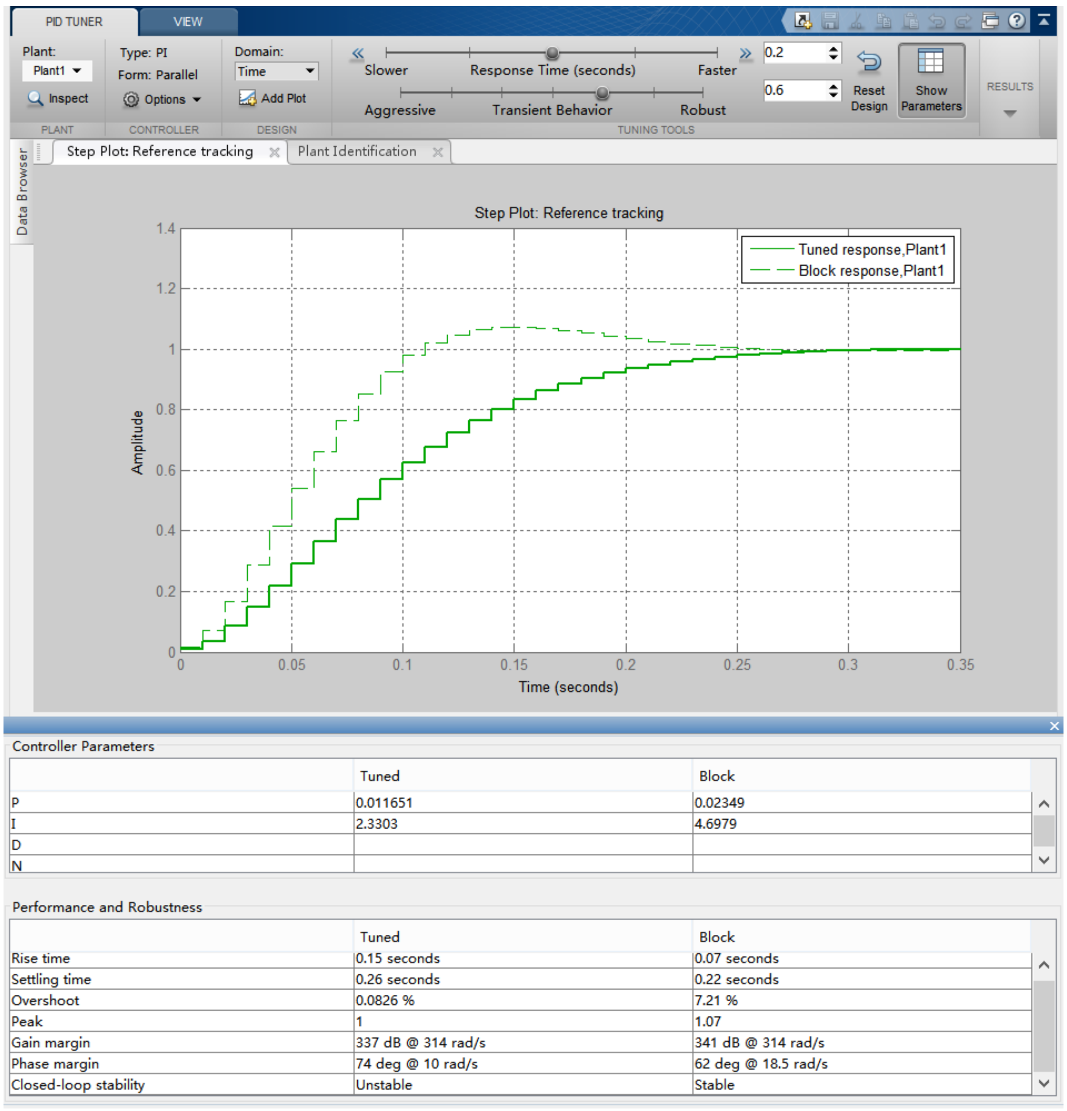
Sample Time (ΔT): 0.01
Offset (u_0): 0
Onset Lag (T_Δ): 0.5
Stop Time (T_f): 1
Signal Type: Step
☐ Show Input Response
☐ Show Offset Response
☑ Show Identification Data

Figure 25. Plant Identification – simulate I/O data



```
Plant Identification Progress

Process Model Identification

Estimation data: Time domain data
Data has 1 outputs, 1 inputs and 51 samples.
Model Type: P1

Estimation Progress
Estimating measured dynamics model...
Nonlinear least squares with automatically chosen line search method
----------------------------------------------------------------------------
                                    Norm of      First-order     Improvement
Iteration      Cost        step      optimality    Expected   Achieved    Bis
----------------------------------------------------------------------------
    0         113174          -               52.9    0.000815
    1        14631.9       9.84              235    0.000815          87.1
    2        3255.67       4.58             79.1    0.00534           77.7
    3        3183.87      0.463            0.612  0.000681           2.21
    4        3183.87   0.000637   2.14e-05   1.67e-08   5.31e-05
    5        3183.87   6.73e-06   2.25e-07   8.84e-14   2.78e-10
----------------------------------------------------------------------------

Result
Termination condition: Near (local) minimum, (norm(g) < tol).
Number of iterations: 5, Number of function evaluations: 11

Status: Estimated using PEM with Focus = "simulation"
Fit to estimation data: 93.35%, FPE: 3433.59
```

Figure 26. Plant Identification – Auto Estimate



Plant: Plant1

Type: PI

Form: Parallel

Domain: Time

0.2

0.6

Controller Parameters

	Tuned	Block
P	0.011651	0.02349
I	2.3303	4.6979
D		
N		

Performance and Robustness

	Tuned	Block
Rise time	0.15 seconds	0.07 seconds
Settling time	0.26 seconds	0.22 seconds
Overshoot	0.0826 %	7.21 %
Peak	1	1.07
Gain margin	337 dB @ 314 rad/s	341 dB @ 314 rad/s
Phase margin	74 deg @ 10 rad/s	62 deg @ 18.5 rad/s
Closed-loop stability	Unstable	Stable

Figure 27. PID tuner

The modelling of single BLDC motor control system is completed. The testing results of the simulation are discussed in Chapter 5.

3 DESIGN OF SINGLE BLDC MOTOR CONTROL SYSTEM

3.1 Overall Structure of System

The overall structure of single BLDC motor control system is shown in Figure 28:

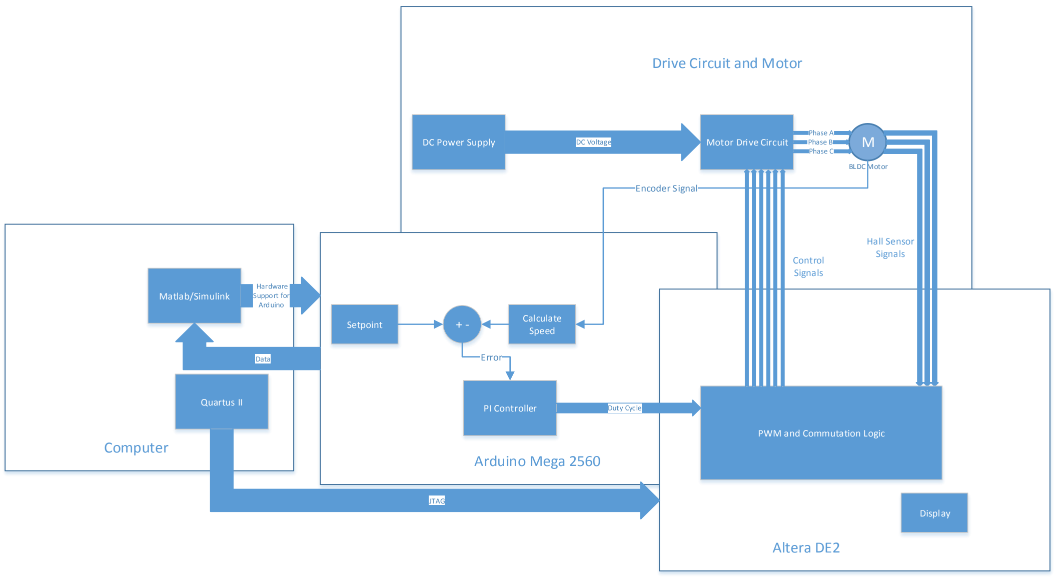


Figure 28. Overall structure of BLDC motor control system

As shown in Figure 28, the whole BLDC motor control system can be divided into four parts: the drive circuit and motor, the control part on Arduino Mega 2560, the control part on Altera DE2, and the computer. Note that the main part of the control system is divided into two parts and implemented on DE2 and Arduino Mega 2560 separately. The reason is that the commutation part and the PWM generation part need to run in an extremely high frequency and require high accuracy, so FPGA is usually the choice for that and Altera DE2 is chosen, while the speed control algorithm need a slower processing and can be easily implemented on microcontroller, so low-cost embedded hardware Arduino Mega 2560 is used. The DE2 is mainly used to generate the PWM signal of which the duty cycle is received from Arduino Mega 2560, and implement the commutation logic. The Arduino Mega 2560 is used for closed-loop speed control. It receives the encoder signal and calculated the motor speed which will be compared with the set point, then the adjusted PWM duty cycle which is used to control the motor speed is outputted by the PI control to Altera DE2. Ten parallel cables are used here to connect the Arduino Mega 2560 and the Altera DE2 to transmit the duty cycle in 10-bit binary format of which the resolution is 1000. The program running on Arduino is automatically generated by the Simulink hardware support for Arduino, and when the model from which the code is generated is running in the external mode on Arduino, we can tuning the parameters and monitor the feedback data in real time. This kind of configuration of the motor control system that the control algorithm is implemented by combining two kinds of hardware: the Arduino Mega 2560 with ATmega2560 microcontroller + Altera DE2 with Cyclone II FPGA is very similar to the one implemented on our robot: ARM Cortex-M3 + Cyclone III. Therefore this design method can be further implemented on our Botnia robot to improve the control performance.

3.2 Motor Drive Circuit and Motor

The details of motor drive circuit and motor part are shown in Figure 29.

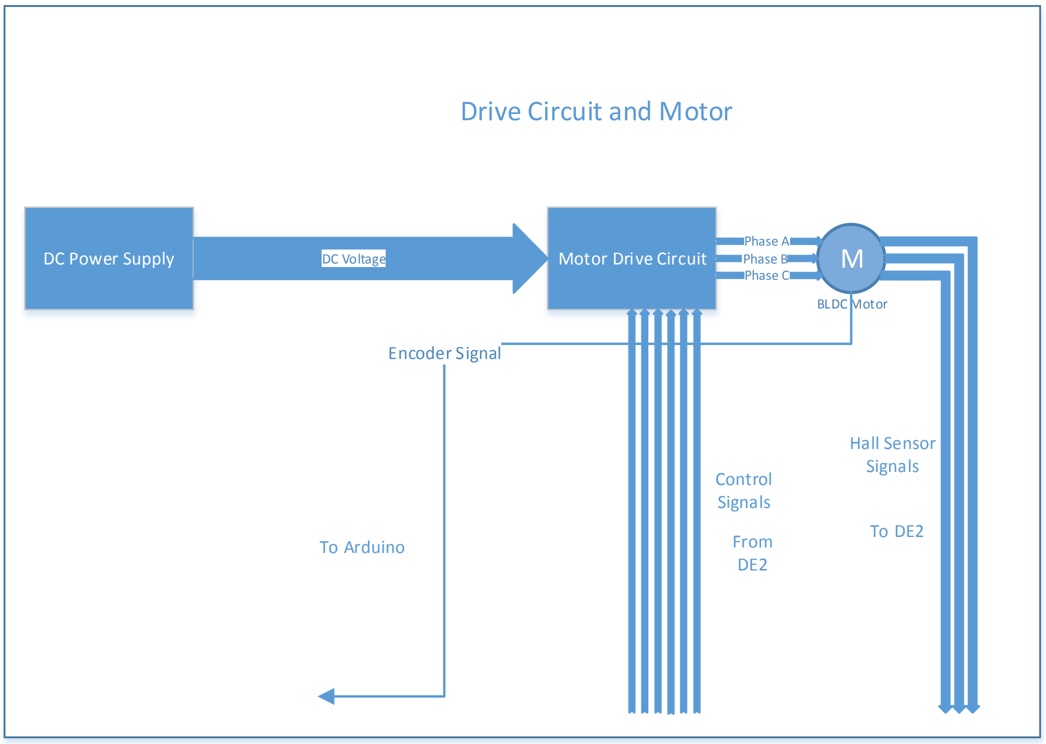


Figure 29. Motor drive circuit and motor

The motor drive circuit here is important. We can choose a ready-made three-phase motor driver chip L6234 with a breadboard and other components to build the motor drive circuit. The L6234 has three half bridges to drive a brushless DC motor. The BCD multipower technology is adopted so that isolated DMOS power transistors with CMOS and Bipolar circuits can be combined on the same chip. The block diagram of L6234 is shown in Figure 30.

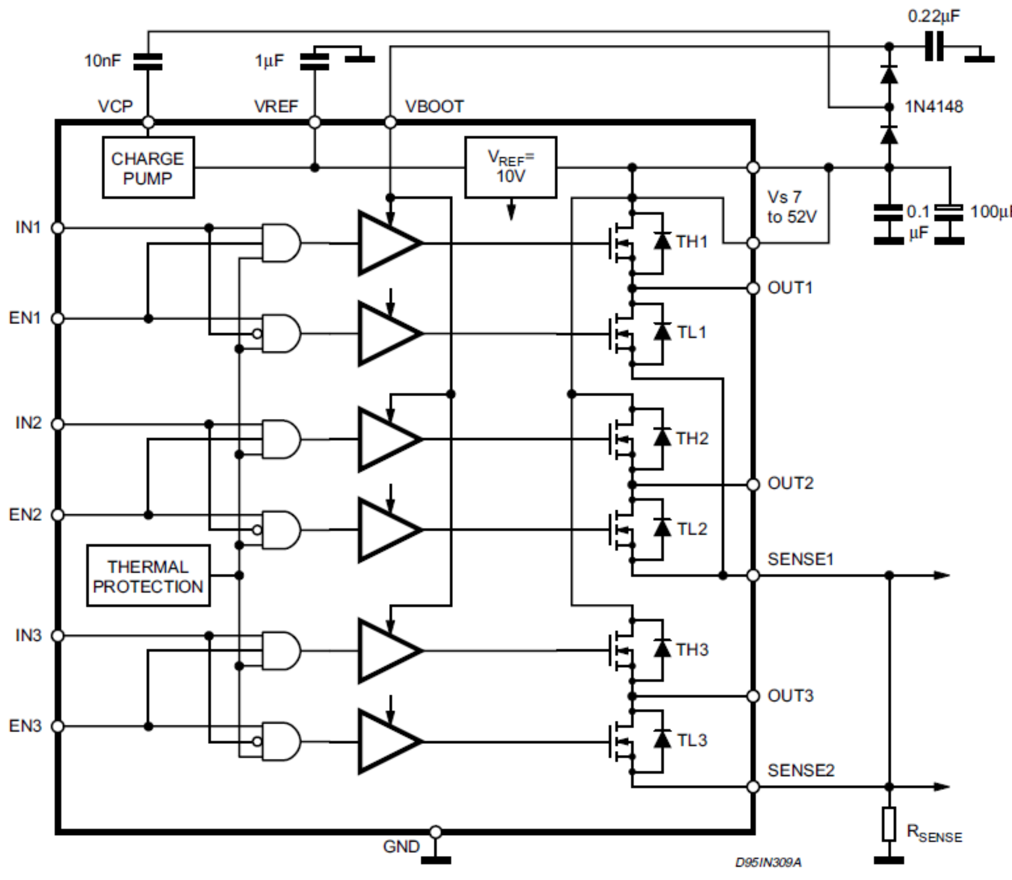
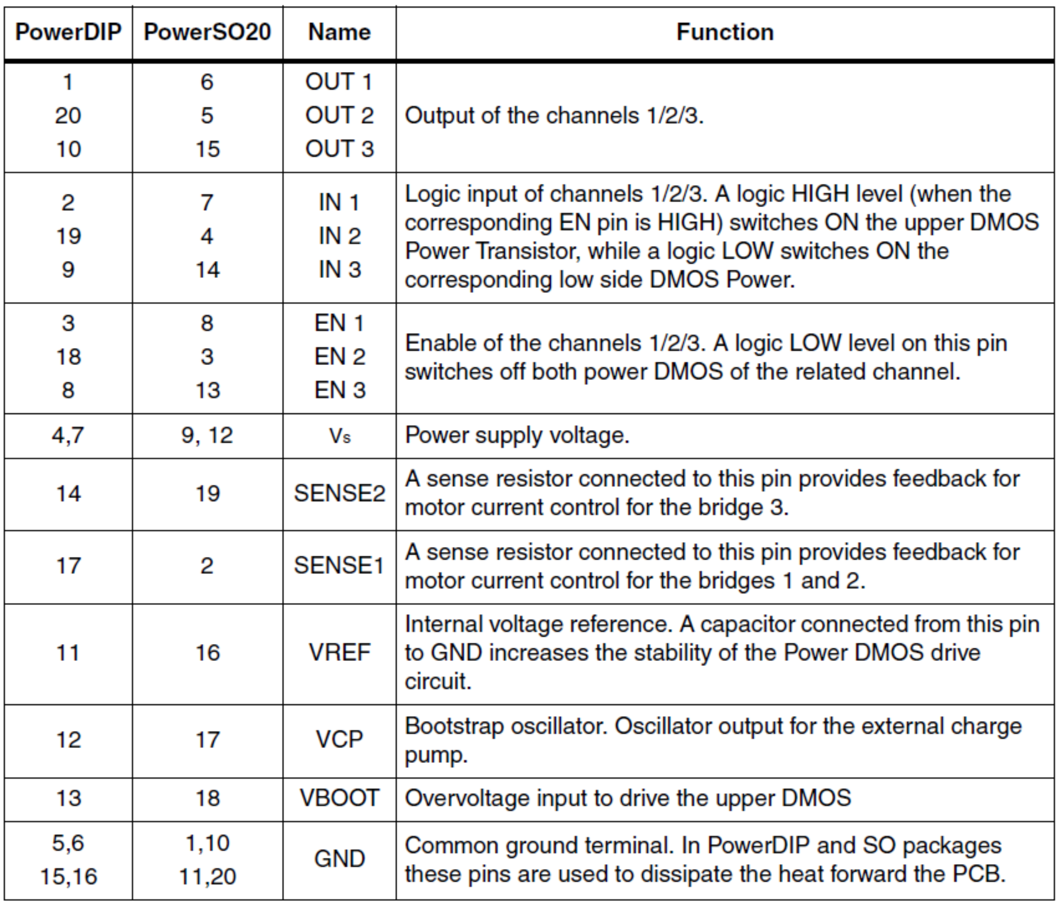


Figure 30. L6234 block diagram/21/

It has six MOSFETs and the construction is similar to the invert circuit shown in Figure 9 and Figure 10. However, the inputs of control (IN1, EN1, IN2, EN2, IN3, and EN3) of this motor drive cannot directly switch the MOSFETS because some logic gates are introduce between them. Figure 31 shows the functions of each pin.



PowerDIP	PowerSO20	Name	Function
1 20 10	6 5 15	OUT 1 OUT 2 OUT 3	Output of the channels 1/2/3.
2 19 9	7 4 14	IN 1 IN 2 IN 3	Logic input of channels 1/2/3. A logic HIGH level (when the corresponding EN pin is HIGH) switches ON the upper DMOS Power Transistor, while a logic LOW switches ON the corresponding low side DMOS Power.
3 18 8	8 3 13	EN 1 EN 2 EN 3	Enable of the channels 1/2/3. A logic LOW level on this pin switches off both power DMOS of the related channel.
4,7	9, 12	V_S	Power supply voltage.
14	19	SENSE2	A sense resistor connected to this pin provides feedback for motor current control for the bridge 3.
17	2	SENSE1	A sense resistor connected to this pin provides feedback for motor current control for the bridges 1 and 2.
11	16	VREF	Internal voltage reference. A capacitor connected from this pin to GND increases the stability of the Power DMOS drive circuit.
12	17	VCP	Bootstrap oscillator. Oscillator output for the external charge pump.
13	18	VBOOT	Overvoltage input to drive the upper DMOS
5,6 15,16	1,10 11,20	GND	Common ground terminal. In PowerDIP and SO packages these pins are used to dissipate the heat forward the PCB.

Figure 31. Pin functions/21/

According to the pin functions, we can obtain the relationship between the Hall sensor signals and the six inputs of L6234, which are implemented on DE2, as the commutation logic. The relationship is shown in Table 4.

Table 4. Relationship between Hall sensor signals and inputs of L6234

Electrical angle	Hall sensors			Logic input						Switch closed	
	H1	H2	H3	IN1	EN1	IN2	EN2	IN3	EN3		
0 º-60 º	1	0	1	1	1	0	1	-	0	TH1	TL2
60 º-120 º	1	0	0	1	1	-	0	0	1	TH1	TL3
120 º-180 º	1	1	0	-	0	1	1	0	1	TH2	TL3

180 º240 º	0	1	0	0	1	1	1	-	0	TH2	TL1
240 º300 º	0	1	1	0	1	-	0	1	1	TH3	TL1
300 º360 º	0	0	1	-	0	0	1	1	1	TH3	TL2

3.3 Altera DE2

Part of the control program is run on Altera DE2. Figure 32 shows the structure of the program run on Altera DE2.

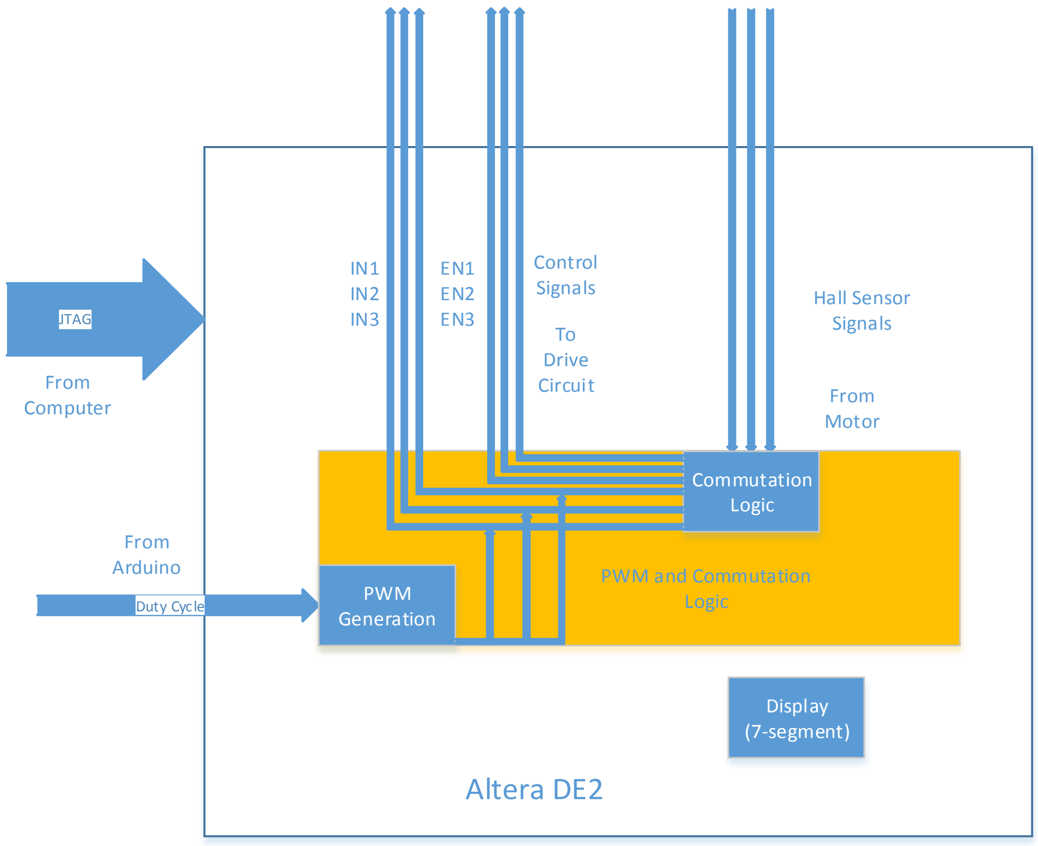


Figure 32. Structure of control program run on Altera DE2

The program contains three modules: the PWM generation module, the commutation logic module and the display module. The PWM generation module receives a 10-bit binary data from Arduino which ranges from 0 to 1000 and represents the duty cycle of the PWM, then the PWM generation module generates the PWM signal according to the received duty cycle. The commutation logic module receives the Hall sensor signals from the motor to obtain the rotor position and then implement the commutation logic to control the commutation process. The output signals of the commutation logic module are fed to the motor drive circuit, and three of them (IN1, IN2 and IN3) which control the three upper MOSFETs of the motor driver, are implemented in logical conjunction with the PWM signal. Therefore, the motor speed can be controlled by the PWM during each interval. The last module is the display module and it displays the value of the received 10-bit data using the 7-segment displays. Most of the codes are generated by HDL coder, except the PWM generation part which is easier to be written by hand. To generate the code using HDL coder, first the corresponding models are built for each module. After all the three model are ready, we can combine the codes and build the whole program, and then the JTAG programming mode is used to download the program onto the DE2 board.

3.3.1 PWM Generation

Altera DE2 has two internal clock inputs: a 50 MHz oscillator and 27 MHz (from TV decoder) for clock sources. It easy to generate the PWM using the 50 MHz clock source. First, we need to decide the frequency of PWM signal, and as a matter of experience the PWM frequency usually need to be at least 10 times the value of the maximum frequency of the motor and should not be less than 15 kHz. In order to make the calculation and coding easier, we can choose 50 kHz as the PWM frequency because the resolution of the PWM is 1000 and if we divide the 50 MHz clock source by 1000, it is 50 KHz.

Therefore, the following VHDL codes can be used to generate the PWM signal.

```
LIBRARY ieee;
USE  ieee.std_logic_1164.ALL;
USE  ieee.numeric_std.ALL;

ENTITY PWM_Generation IS
PORT(  clock_50mhz       : IN    std_logic;
       Duty_cycle     : IN    std_logic_vector(9 DOWNTO 0);
       PWM            : OUT   std_logic);
END PWM_Generation;

ARCHITECTURE rtl OF PWM_Generation IS
```

```
    SIGNAL count_50KHz                                        :
integer RANGE 0 TO 999 := 0;
    SIGNAL PWM_int                                            :
std_logic := '1';
    SIGNAL Duty_cycle_unsigned                     : unsigned(9
DOWNTO 0);
BEGIN
    Duty_cycle_unsigned <= unsigned(Duty_cycle);
    PROCESS
    BEGIN
-- divide by 1000
        WAIT UNTIL clock_50mhz'EVENT and clock_50mhz = '1';
            IF count_50KHz < 999 THEN
                count_50KHz <= count_50KHz + 1;
            ELSE
                count_50KHz <= 0;
            END IF;
            IF count_50KHz < Duty_cycle_unsigned THEN
                PWM_int <= '1';
            ELSE
                PWM_int <= '0';
            END IF;
            PWM <= PWM_int;
    END PROCESS;
END rtl;
```

Since the input duty cycle is a 10-bit data, we need to first convert it into unsigned integer to make the comparison possible. At the rising edge of the clock signal, the counter is increased by one until it reaches 999. And during every period of PWM, if the counter value is less than the input duty cycle, the PWM is ON, otherwise it is OFF.

3.3.2 Commutation

The commutation logic module can be implemented by using the commutation logic described in Table 4. The codes of this part are automatically generated by using HDL coder, thus we do not need to write the codes by hand. The first thing we need to do is to build the model of commutation logic module. The model can be easily used to verify the design of the commutation logic and further the model is used to generate the VHDL codes of the module. The model of this module is shown in Figure 33.

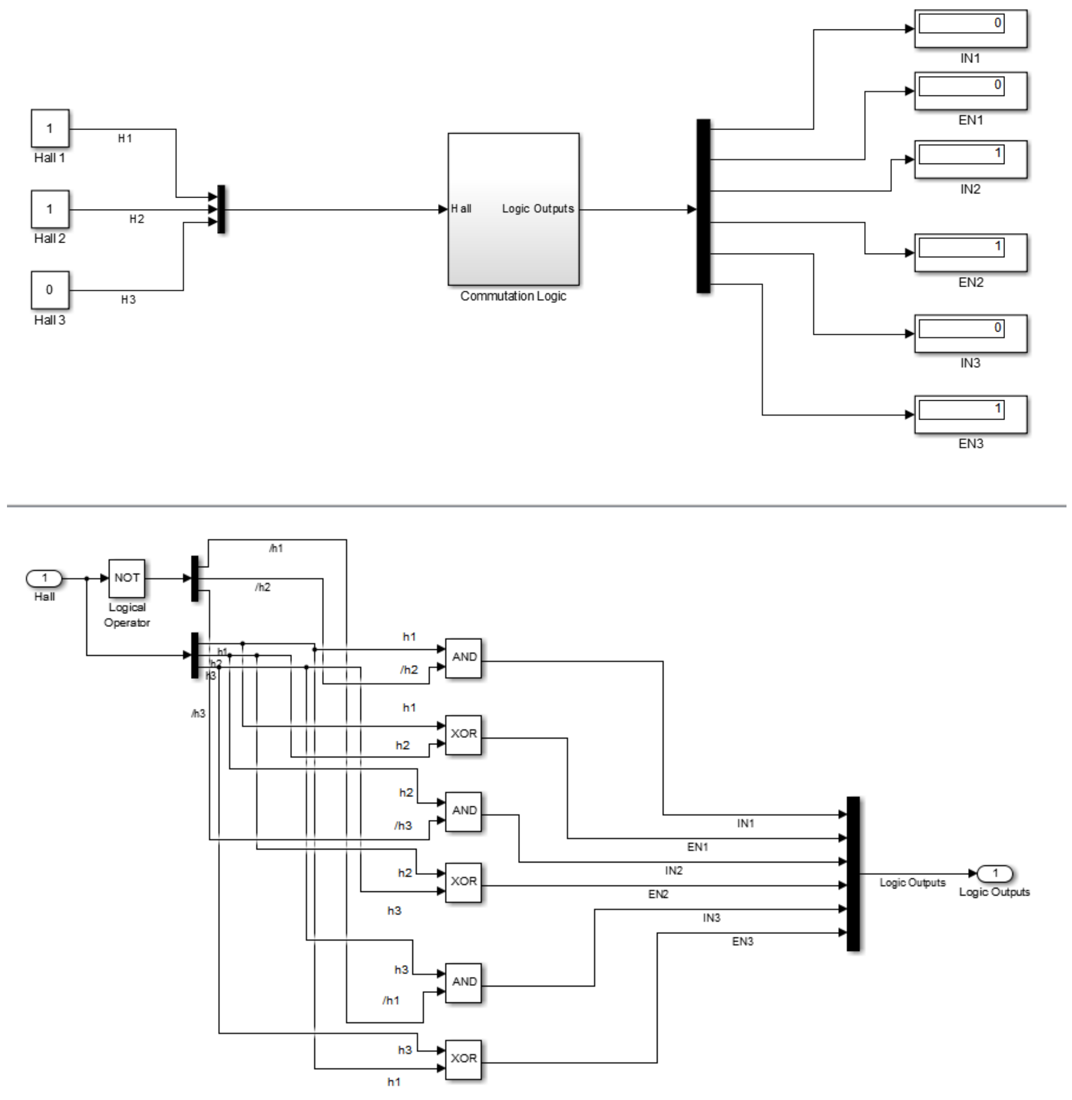


Figure 33. Model of commutaion logic module

After the design of commmutation logic module is verifed, we can just right click the commuctioan logic subsystem, go to the HDL coder and click generate the HDL code for the subsystem. If there are no errors reported, the codes are generated. Here we choose to gererate the VHDL code and this choice can be found in Model Configurtaion Parameters. The generated codes are shown as follows:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Commutation_Logic IS
```

```
  PORT( Hall                              :   IN    std_logic_vector(0
TO 2);  -- boolean [3]
      Logic_Outputs                       :   OUT   std_logic_vector(0
TO 5)  -- boolean [6]
      );
END Commutation_Logic;


ARCHITECTURE rtl OF Commutation_Logic IS

  -- Signals
  SIGNAL H1                               : std_logic;
  SIGNAL Hall_1                            : std_logic;
  SIGNAL Hall_1_1                          : std_logic;
  SIGNAL H2                               : std_logic;
  SIGNAL Hall_2                            : std_logic;
  SIGNAL Hall_2_1                          : std_logic;
  SIGNAL H3                               : std_logic;
  SIGNAL Hall_0                            : std_logic;
  SIGNAL Hall_0_1                          : std_logic;
  SIGNAL H1_1                             : std_logic;
  SIGNAL IN1                              : std_logic;
  SIGNAL H1_2                             : std_logic;
  SIGNAL EN1                              : std_logic;
  SIGNAL H2_1                             : std_logic;
  SIGNAL IN2                              : std_logic;
  SIGNAL H2_2                             : std_logic;
  SIGNAL EN2                              : std_logic;
  SIGNAL H3_1                             : std_logic;
  SIGNAL IN3                              : std_logic;
  SIGNAL H3_2                             : std_logic;
  SIGNAL EN3                              : std_logic;

BEGIN
  -- h1
  --
  -- /h2
  --
  -- h1
  --
  -- h2
  --
  -- h2
  --
  -- /h3
  --
  -- h2
  --
  -- h3
  --
  -- h3
  --
  -- /h1
  --
  -- h3
  --
  -- h1

  H1 <= Hall(0);
  Hall_1 <= Hall(1);

  Hall_1_1 <=  NOT Hall_1;

  H2 <= Hall(1);

  Hall_2 <= Hall(2);

  Hall_2_1 <=  NOT Hall_2;

  H3 <= Hall(2);

  Hall_0 <= Hall(0);

  Hall_0_1 <=  NOT Hall_0;

  H1_1 <= H1 AND Hall_1_1;

  IN1 <= H1_1;

  H1_2 <= H1 XOR H2;

  EN1 <= H1_2;

  H2_1 <= H2 AND Hall_2_1;

  IN2 <= H2_1;

  H2_2 <= H2 XOR H3;

  EN2 <= H2_2;

  H3_1 <= H3 AND Hall_0_1;

  IN3 <= H3_1;

  H3_2 <= H3 XOR H1;

  EN3 <= H3_2;

  Logic_Outputs(0) <= IN1;
  Logic_Outputs(1) <= EN1;
  Logic_Outputs(2) <= IN2;
  Logic_Outputs(3) <= EN2;
  Logic_Outputs(4) <= IN3;
  Logic_Outputs(5) <= EN3;

END rtl;
```

3.3.3 Display

The function of the display module is to display the value of input data from Arduino using 7-segment displays. Since the input data is in 10-bit binary format, we first need to convert the binary data into binary-coded decimal (BCD) notation so that we can further extract the bits which can represents the unit, ten, hundred, thousand respectively and the decimal digits are displayed on 7-segment displays.

The double dabble algorithm (also known as the shift and add 3 algorithm) is used to convert the binary data into BCD notation. The details of the algorithm can found on Wikipedia./22/

In our case, the data to be converted is a 10-bit binary data ranging from 0 to 1000, so the algorithm need to iterate 10 times. However, in the first three iterations, no BCD digits will be greater than four, so we can assume the fourth iteration as first iteration and initial the register accordingly and this will decrease the iteration times to 7. The model of the converting algorithm is shown in Figure 34.

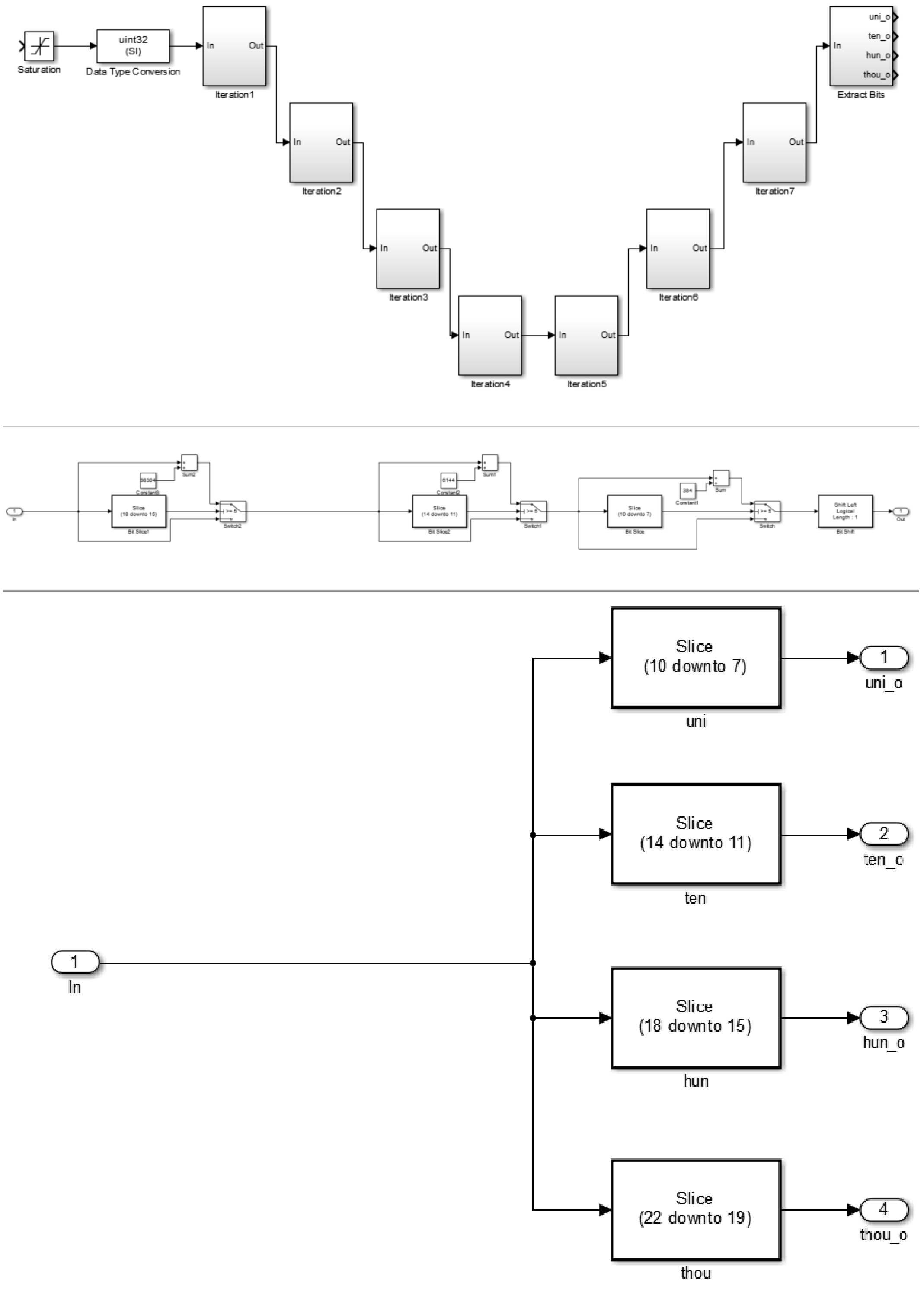


Figure 34. Convert 10-bit binary to BCD

After the convertion is done, we need to display each digit on 7-segment display. DE2 has 8 7-segment displays, and the configuration of each 7-segment display is shown in Figure 34.

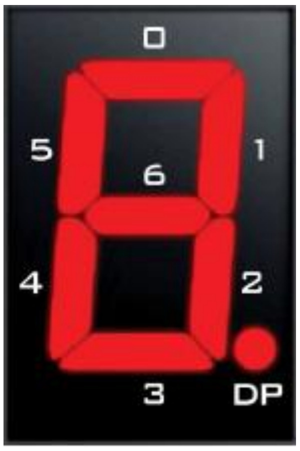


Figure 35. Position and index of each segment in a 7-segment display/23/

The logic to display decimal digit is described in Table 5. Notice that when the segment is ON when the logic input is LOW.

Table 5. 7-segment display logic

BCD	Decimal	Segments						
		0	1	2	3	4	5	6
0000	0	0	0	0	0	0	0	1
0001	1	1	0	0	1	1	1	1
0010	2	0	0	1	0	0	1	0
0011	3	0	0	0	0	1	1	0
0100	4	1	0	0	1	1	0	0
0101	5	0	1	0	0	1	0	0

0110	6	1	1	0	0	0	0	0
0111	7	0	0	0	1	1	1	1
1000	8	0	0	0	0	0	0	0
1001	9	0	0	0	1	1	0	0

As shown in Table 5, the model of display logic can be built which is shown is Figure 35.

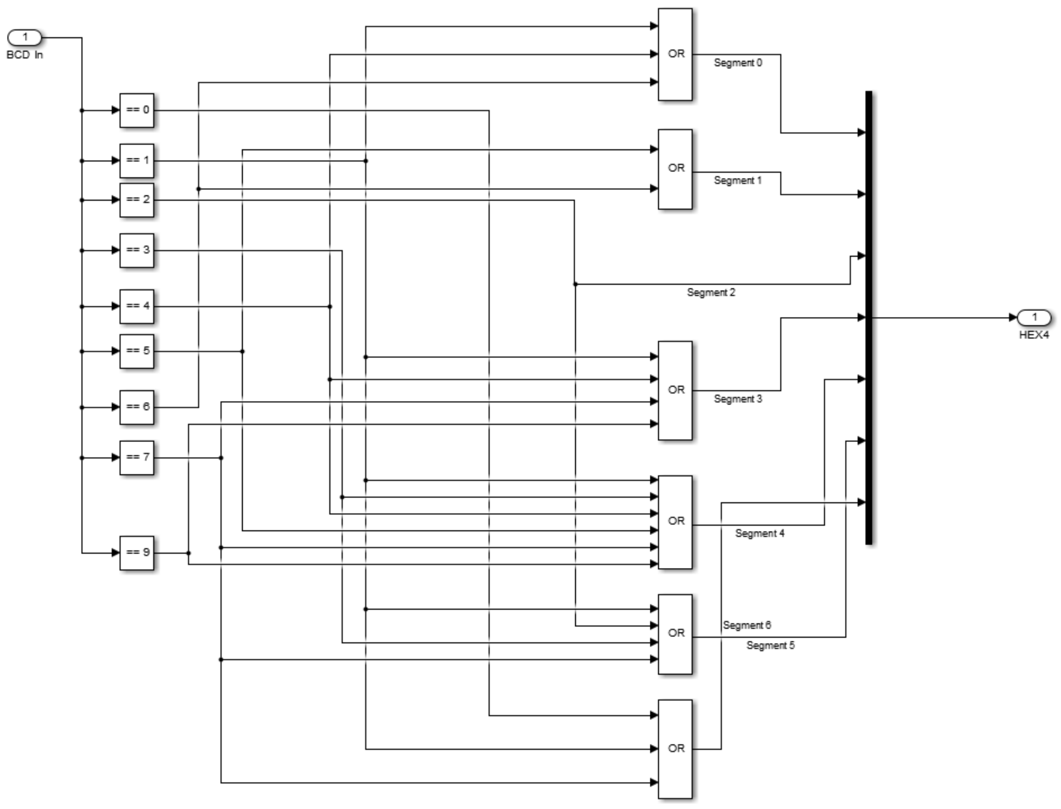


Figure 36. Model of 7-segment display logic

Now we can complete the model of the whole module which is shown in Figure 36 and use it to generate the VHDL codes.

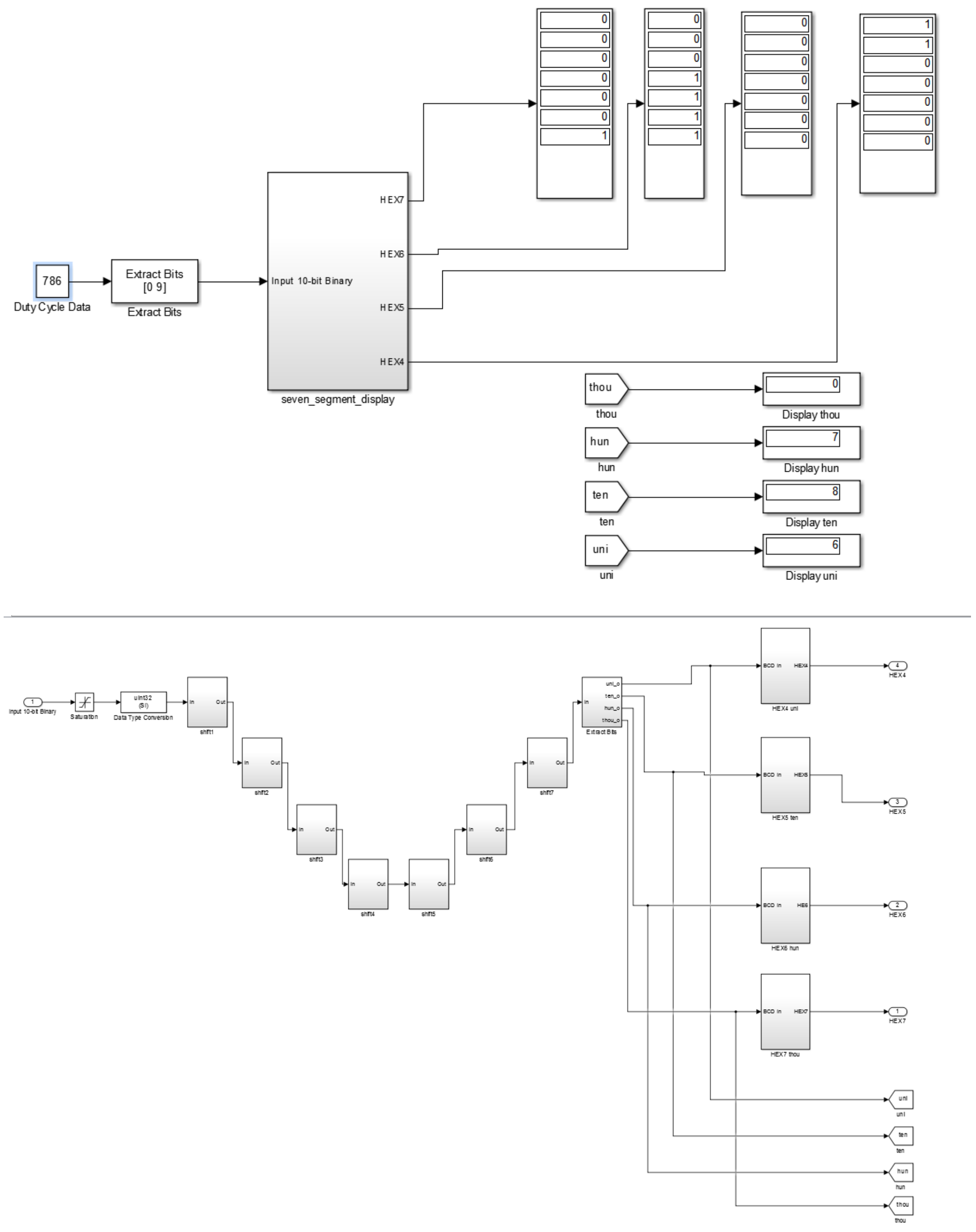


Figure 37. Model of display module

The generated VHDL source files are shown in Figure 36. Among them, the seven_segment_display is the Top-Level VHDL source file. This design method can save a lot of time.

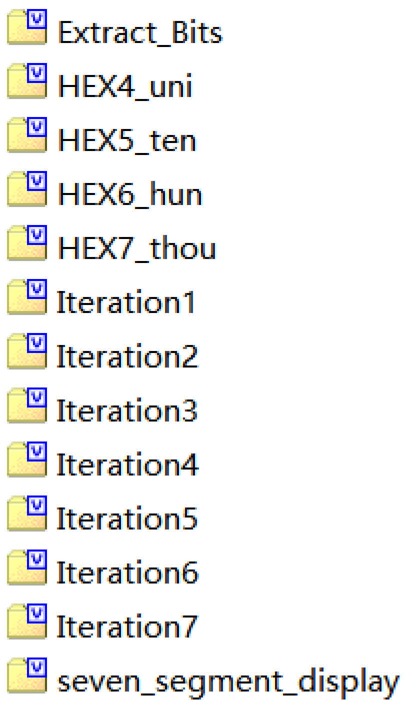

Extract_Bits
HEX4_uni
HEX5_ten
HEX6_hun
HEX7_thou
Iteration1
Iteration2
Iteration3
Iteration4
Iteration5
Iteration6
Iteration7
seven_segment_display

Figure 38. Generated source files

The content of seven_segment_display VHDL file is shown as follows.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY seven_segment_display IS
  PORT( Input_10_bit_Binary                :   IN
std_logic_vector(9 DOWNTO 0);  -- ufix10
       HEX7                             :   OUT   std_logic_vector(0
TO 6);  -- boolean [7]
       HEX6                             :   OUT   std_logic_vector(0
TO 6);  -- boolean [7]
       HEX5                             :   OUT   std_logic_vector(0
TO 6);  -- boolean [7]
       HEX4                             :   OUT   std_logic_vector(0
TO 6)  -- boolean [7]
       );
END seven_segment_display;


ARCHITECTURE rtl OF seven_segment_display IS

  -- Component Declarations
  COMPONENT Iteration1
    PORT( In_rsvd                          :   IN
std_logic_vector(31 DOWNTO 0);  -- uint32
         Out_rsvd                          :   OUT
std_logic_vector(31 DOWNTO 0)  -- uint32
         );
  END COMPONENT;

  COMPONENT Iteration2
    PORT( In_rsvd                          :   IN
std_logic_vector(31 DOWNTO 0);  -- uint32
          Out_rsvd                         :   OUT
std_logic_vector(31 DOWNTO 0)  -- uint32
          );
  END COMPONENT;

  COMPONENT Iteration3
    PORT( In_rsvd                          :   IN
std_logic_vector(31 DOWNTO 0);  -- uint32
          Out_rsvd                         :   OUT
std_logic_vector(31 DOWNTO 0)  -- uint32
          );
  END COMPONENT;

  COMPONENT Iteration4
    PORT( In_rsvd                          :   IN
std_logic_vector(31 DOWNTO 0);  -- uint32
          Out_rsvd                         :   OUT
std_logic_vector(31 DOWNTO 0)  -- uint32
          );
  END COMPONENT;

  COMPONENT Iteration5
    PORT( In_rsvd                          :   IN
std_logic_vector(31 DOWNTO 0);  -- uint32
          Out_rsvd                         :   OUT
std_logic_vector(31 DOWNTO 0)  -- uint32
          );
  END COMPONENT;

  COMPONENT Iteration6
    PORT( In_rsvd                          :   IN
std_logic_vector(31 DOWNTO 0);  -- uint32
          Out_rsvd                         :   OUT
std_logic_vector(31 DOWNTO 0)  -- uint32
          );
  END COMPONENT;

  COMPONENT Iteration7
    PORT( In_rsvd                          :   IN
std_logic_vector(31 DOWNTO 0);  -- uint32
          Out_rsvd                         :   OUT
std_logic_vector(31 DOWNTO 0)  -- uint32
          );
  END COMPONENT;

  COMPONENT Extract_Bits
    PORT( In_rsvd                          :   IN
std_logic_vector(31 DOWNTO 0);  -- uint32
          uni_o                           :   OUT   std_logic_vector(3
DOWNTO 0);  -- ufix4
          ten_o                           :   OUT   std_logic_vector(3
DOWNTO 0);  -- ufix4
          hun_o                           :   OUT   std_logic_vector(3
DOWNTO 0);  -- ufix4
```

```
        thou_o                          :   OUT   std_logic_vector(3
DOWNTO 0)  -- ufix4
        );
  END COMPONENT;

  COMPONENT HEX7_thou
    PORT( BCD_In                          :   IN    std_logic_vector(3
DOWNTO 0);  -- ufix4
        HEX7                            :   OUT   std_logic_vector(0
TO 6)  -- boolean [7]
        );
  END COMPONENT;

  COMPONENT HEX6_hun
    PORT( BCD_In                          :   IN    std_logic_vector(3
DOWNTO 0);  -- ufix4
        HE6                             :   OUT   std_logic_vector(0
TO 6)  -- boolean [7]
        );
  END COMPONENT;

  COMPONENT HEX5_ten
    PORT( BCD_In                          :   IN    std_logic_vector(3
DOWNTO 0);  -- ufix4
        HEX5                            :   OUT   std_logic_vector(0
TO 6)  -- boolean [7]
        );
  END COMPONENT;

  COMPONENT HEX4_uni
    PORT( BCD_In                          :   IN    std_logic_vector(3
DOWNTO 0);  -- ufix4
        HEX4                            :   OUT   std_logic_vector(0
TO 6)  -- boolean [7]
        );
  END COMPONENT;

  -- Component Configuration Statements
  FOR ALL : Iteration1
    USE ENTITY work.Iteration1(rtl);

  FOR ALL : Iteration2
    USE ENTITY work.Iteration2(rtl);

  FOR ALL : Iteration3
    USE ENTITY work.Iteration3(rtl);

  FOR ALL : Iteration4
    USE ENTITY work.Iteration4(rtl);

  FOR ALL : Iteration5
    USE ENTITY work.Iteration5(rtl);

  FOR ALL : Iteration6
    USE ENTITY work.Iteration6(rtl);

  FOR ALL : Iteration7
    USE ENTITY work.Iteration7(rtl);
  FOR ALL : Extract_Bits
    USE ENTITY work.Extract_Bits(rtl);

  FOR ALL : HEX7_thou
    USE ENTITY work.HEX7_thou(rtl);

  FOR ALL : HEX6_hun
    USE ENTITY work.HEX6_hun(rtl);

  FOR ALL : HEX5_ten
    USE ENTITY work.HEX5_ten(rtl);

  FOR ALL : HEX4_uni
    USE ENTITY work.HEX4_uni(rtl);

  -- Signals
  SIGNAL Input_10_bit_Binary_unsigned     : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL Saturation_out1                 : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL Data_Type_Conversion_out1        : unsigned(31 DOWNTO
0);  -- uint32
  SIGNAL y                              : std_logic_vector(31
DOWNTO 0);  -- ufix32
  SIGNAL y_1                            : std_logic_vector(31
DOWNTO 0);  -- ufix32
  SIGNAL y_2                            : std_logic_vector(31
DOWNTO 0);  -- ufix32
  SIGNAL y_3                            : std_logic_vector(31
DOWNTO 0);  -- ufix32
  SIGNAL y_4                            : std_logic_vector(31
DOWNTO 0);  -- ufix32
  SIGNAL y_5                            : std_logic_vector(31
DOWNTO 0);  -- ufix32
  SIGNAL y_6                            : std_logic_vector(31
DOWNTO 0);  -- ufix32
  SIGNAL y_7                            : std_logic_vector(3 DOWNTO
0);  -- ufix4
  SIGNAL y_8                            : std_logic_vector(3 DOWNTO
0);  -- ufix4
  SIGNAL y_9                            : std_logic_vector(3 DOWNTO
0);  -- ufix4
  SIGNAL y_10                           : std_logic_vector(3 DOWNTO
0);  -- ufix4
  SIGNAL HEX7_thou_out1                  : std_logic_vector(0 TO
6);  -- boolean [7]
  SIGNAL HEX6_hun_out1                   : std_logic_vector(0 TO
6);  -- boolean [7]
  SIGNAL HEX5_ten_out1                   : std_logic_vector(0 TO
6);  -- boolean [7]
  SIGNAL HEX4_uni_out1                   : std_logic_vector(0 TO
6);  -- boolean [7]

BEGIN
  u_Iteration1 : Iteration1
    PORT MAP( In_rsvd =>
std_logic_vector(Data_Type_Conversion_out1),  -- uint32
              Out_rsvd => y  -- uint32
              );
```

```
u_Iteration2 : Iteration2
  PORT MAP( In_rsvd => y,  -- uint32
            Out_rsvd => y_1  -- uint32
            );

u_Iteration3 : Iteration3
  PORT MAP( In_rsvd => y_1,  -- uint32
            Out_rsvd => y_2  -- uint32
            );

u_Iteration4 : Iteration4
  PORT MAP( In_rsvd => y_2,  -- uint32
            Out_rsvd => y_3  -- uint32
            );

u_Iteration5 : Iteration5
  PORT MAP( In_rsvd => y_3,  -- uint32
            Out_rsvd => y_4  -- uint32
            );

u_Iteration6 : Iteration6
  PORT MAP( In_rsvd => y_4,  -- uint32
            Out_rsvd => y_5  -- uint32
            );

u_Iteration7 : Iteration7
  PORT MAP( In_rsvd => y_5,  -- uint32
            Out_rsvd => y_6  -- uint32
            );

u_Extract_Bits : Extract_Bits
  PORT MAP( In_rsvd => y_6,  -- uint32
            uni_o => y_7,  -- ufix4
            ten_o => y_8,  -- ufix4
            hun_o => y_9,  -- ufix4
            thou_o => y_10  -- ufix4
            );

u_HEX7_thou : HEX7_thou
  PORT MAP( BCD_In => y_10,  -- ufix4
            HEX7 => HEX7_thou_out1  -- boolean [7]
            );

u_HEX6_hun : HEX6_hun
  PORT MAP( BCD_In => y_9,  -- ufix4
            HE6 => HEX6_hun_out1  -- boolean [7]
            );

u_HEX5_ten : HEX5_ten
  PORT MAP( BCD_In => y_8,  -- ufix4
            HEX5 => HEX5_ten_out1  -- boolean [7]
            );

u_HEX4_uni : HEX4_uni
  PORT MAP( BCD_In => y_7,  -- ufix4
            HEX4 => HEX4_uni_out1  -- boolean [7]
            );
  Input_10_bit_Binary_unsigned <= unsigned(Input_10_bit_Binary);


  Saturation_out1 <= to_unsigned(2#1111101000#, 10) WHEN
Input_10_bit_Binary_unsigned > 1000 ELSE
      Input_10_bit_Binary_unsigned;

  Data_Type_Conversion_out1 <= resize(Saturation_out1, 32);

  HEX7 <= HEX7_thou_out1;

  HEX6 <= HEX6_hun_out1;

  HEX5 <= HEX5_ten_out1;

  HEX4 <= HEX4_uni_out1;

END rtl;
```

3.3.4 Overall

After finishing the three modules, we need to combine them in one Quartus II project. One approach is that we can write the Top-Level VHDL source file manually, and then add both the manually written source file (PWM generation) and the automatically generated source files (commutation module and display module). Here we implement in another way which is based on the model, so the Top-Level VHDL source file can be automatically generated. Firstly, we can build the whole model of the program run on DE2 according to the structure shown in Figure 32. The main parts of the model are the three modules we discussed above. We can directly use the model of the display module and the model of the commutation module as the subsystems while the PWM generation module is an empty subsystem as shown in Figure 37. We only need to make the name of the inputs, outputs and the system identical to the ones used in our written codes, thus we can use our manually written source file to replace the automatically generated source file later.

1
clock_50mhz
2
Duty_cycle
1
PWM

Figure 39. PWM generation subsystem

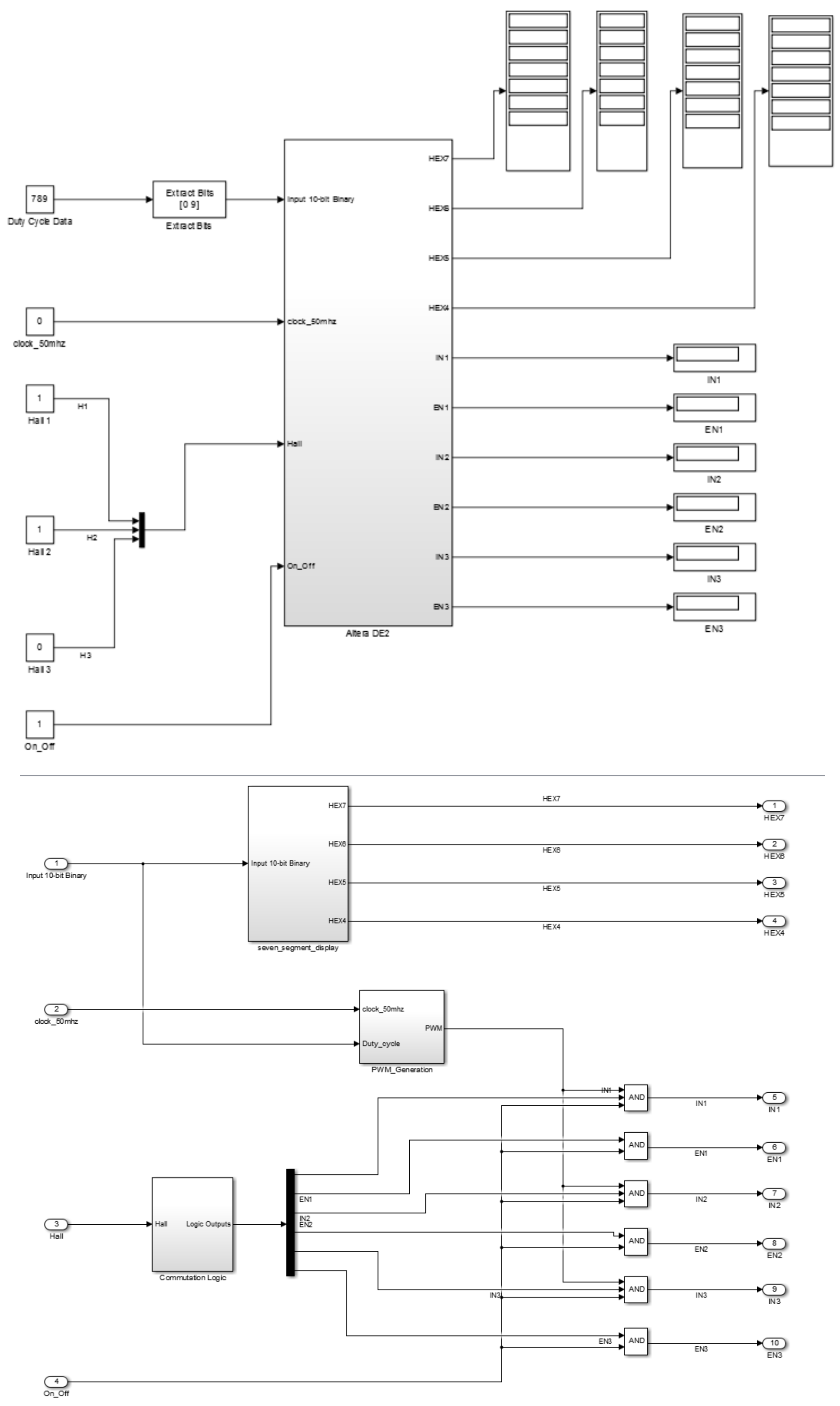
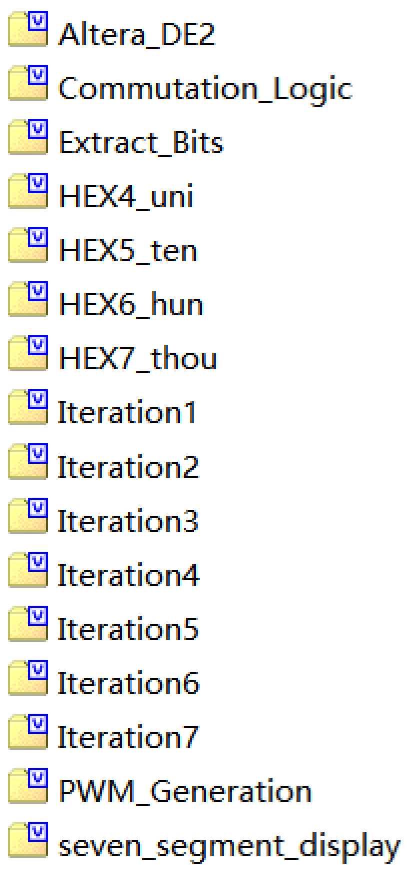


Figure 40. Model of the whole system run on DE2

Figure 38 shows the model of the whole program. In addition to the functions described above, an input On_Off is added. It works as a switch to switch ON or OFF the motor. All automatically generated files from this model are shown in Figure 39. We can replace the PWM_Generation source file with our manually written one.



Altera_DE2
Commutation_Logic
Extract_Bits
HEX4_uni
HEX5_ten
HEX6_hun
HEX7_thou
Iteration1
Iteration2
Iteration3
Iteration4
Iteration5
Iteration6
Iteration7
PWM_Generation
seven_segment_display

Figure 41. Files for the whole program

The content of the Top-Level VHDL source file is as follows:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Altera_DE2 IS
  PORT( Input_10_bit_Binary                 :   IN
std_logic_vector(9 DOWNTO 0); -- ufix10
        clock_50mhz                        :   IN    std_logic;
        Hall                              :   IN    std_logic_vector(0
TO 2);  -- boolean [3]
        On_Off                             :   IN    std_logic;
        HEX7                              :   OUT   std_logic_vector(0
TO 6);  -- boolean [7]
       HEX6                          :   OUT   std_logic_vector(0
TO 6);  -- boolean [7]
       HEX5                          :   OUT   std_logic_vector(0
TO 6);  -- boolean [7]
       HEX4                          :   OUT   std_logic_vector(0
TO 6);  -- boolean [7]
       IN1                           :   OUT   std_logic;
       EN1                           :   OUT   std_logic;
       IN2                           :   OUT   std_logic;
       EN2                           :   OUT   std_logic;
       IN3                           :   OUT   std_logic;
       EN3                           :   OUT   std_logic
       );
END Altera_DE2;


ARCHITECTURE rtl OF Altera_DE2 IS

  -- Component Declarations
  COMPONENT seven_segment_display
    PORT( Input_10_bit_Binary            :   IN
std_logic_vector(9 DOWNTO 0);  -- ufix10
         HEX7                        :   OUT   std_logic_vector(0
TO 6);  -- boolean [7]
         HEX6                        :   OUT   std_logic_vector(0
TO 6);  -- boolean [7]
         HEX5                        :   OUT   std_logic_vector(0
TO 6);  -- boolean [7]
         HEX4                        :   OUT   std_logic_vector(0
TO 6)  -- boolean [7]
         );
  END COMPONENT;

  COMPONENT PWM_Generation
    PORT( clock_50mhz                  :   IN    std_logic;
         Duty_cycle                  :   IN    std_logic_vector(9
DOWNTO 0);  -- ufix10
         PWM                         :   OUT   std_logic
         );
  END COMPONENT;

  COMPONENT Commutation_Logic
    PORT( Hall                         :   IN    std_logic_vector(0
TO 2);  -- boolean [3]
         Logic_Outputs                :   OUT   std_logic_vector(0
TO 5)  -- boolean [6]
         );
  END COMPONENT;

  -- Component Configuration Statements
  FOR ALL : seven_segment_display
    USE ENTITY work.seven_segment_display(rtl);

  FOR ALL : PWM_Generation
    USE ENTITY work.PWM_Generation(rtl);

  FOR ALL : Commutation_Logic
    USE ENTITY work.Commutation_Logic(rtl);
  -- Signals
  SIGNAL PWM_Generation_out1            : std_logic;
  SIGNAL Commutation_Logic_out1          : std_logic_vector(0 TO
5);  -- boolean [6]
  SIGNAL IN1_1                        : std_logic;
  SIGNAL PWM_Generation_out1_1           : std_logic;
  SIGNAL EN1_1                        : std_logic;
  SIGNAL EN1_2                        : std_logic;
  SIGNAL IN2_1                        : std_logic;
  SIGNAL PWM_Generation_out1_2           : std_logic;
  SIGNAL EN2_1                        : std_logic;
  SIGNAL EN2_2                        : std_logic;
  SIGNAL IN3_1                        : std_logic;
  SIGNAL PWM_Generation_out1_3           : std_logic;
  SIGNAL EN3_1                        : std_logic;
  SIGNAL EN3_2                        : std_logic;

BEGIN
  u_seven_segment_display : seven_segment_display
    PORT MAP( Input_10_bit_Binary => Input_10_bit_Binary,  --
ufix10
              HEX7 => HEX7,  -- boolean [7]
              HEX6 => HEX6,  -- boolean [7]
              HEX5 => HEX5,  -- boolean [7]
              HEX4 => HEX4  -- boolean [7]
              );

  u_PWM_Generation : PWM_Generation
    PORT MAP( clock_50mhz => clock_50mhz,
              Duty_cycle => Input_10_bit_Binary,  -- ufix10
              PWM => PWM_Generation_out1
              );

  u_Commutation_Logic : Commutation_Logic
    PORT MAP( Hall => Hall,  -- boolean [3]
              Logic_Outputs => Commutation_Logic_out1  -- boolean
[6]
              );

  IN1_1 <= Commutation_Logic_out1(0);

  PWM_Generation_out1_1 <= On_Off AND (PWM_Generation_out1 AND
IN1_1);

  IN1 <= PWM_Generation_out1_1;

  EN1_1 <= Commutation_Logic_out1(1);

  EN1_2 <= EN1_1 AND On_Off;

  EN1 <= EN1_2;

  IN2_1 <= Commutation_Logic_out1(2);

  PWM_Generation_out1_2 <= On_Off AND (PWM_Generation_out1 AND
IN2_1);

  IN2 <= PWM_Generation_out1_2;
  EN2_1 <= Commutation_Logic_out1(3);

  EN2_2 <= EN2_1 AND On_Off;

  EN2 <= EN2_2;

  IN3_1 <= Commutation_Logic_out1(4);

  PWM_Generation_out1_3 <= On_Off AND (PWM_Generation_out1 AND
IN3_1);

  IN3 <= PWM_Generation_out1_3;

  EN3_1 <= Commutation_Logic_out1(5);

  EN3_2 <= EN3_1 AND On_Off;

  EN3 <= EN3_2;

END rtl;
```

After the pin assignment is made, the whole program can be build and run on DE2.

3.4 Arduino Mega 2560

The rest part of the control program including the PI controller is run on Arduino Mega 2560. Figure 40 shows the structure of the program run on Arduino Mega 2560.

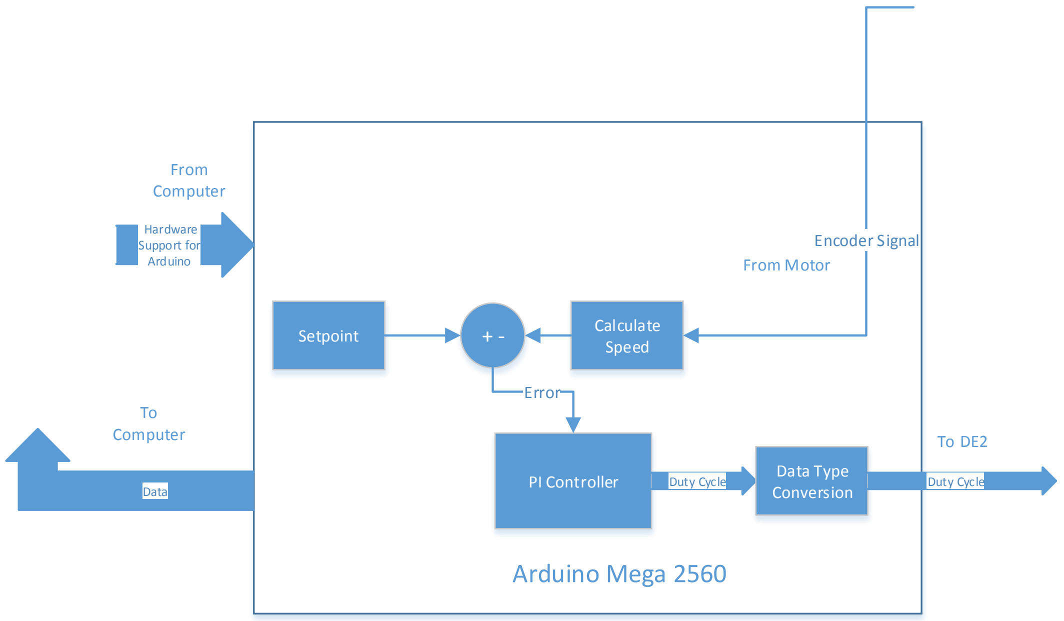


Figure 42. Structure of control program run on Arduino Mega 2560

The main part of the program is the PI controller, it receives the error between the set point and the measured speed and then adjusts the duty cycle of PWM to control the speed and eliminate the error. The measured speed is obtained from the encoder signal by counting the single pulses every 10ms. The output data of the PI controller is converted into 10-bit binary and then transmitted to DE2. Since the Simulink hardware support for Arduino can automatically generate the code from the model and deploy the program on Arduino for standalone operation or in the external mode for interactive parameter tuning and signal monitoring, the model is identical to the program. The model built for running on Arduino Mega 2560 is shown in Figure 41.

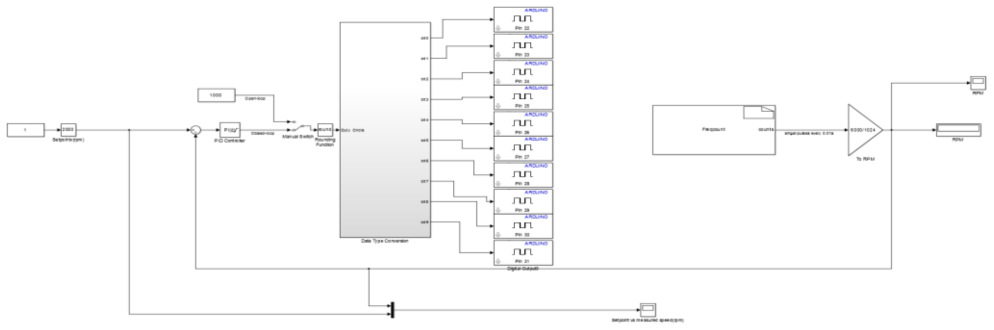


Figure 43. Model for running on Arduino

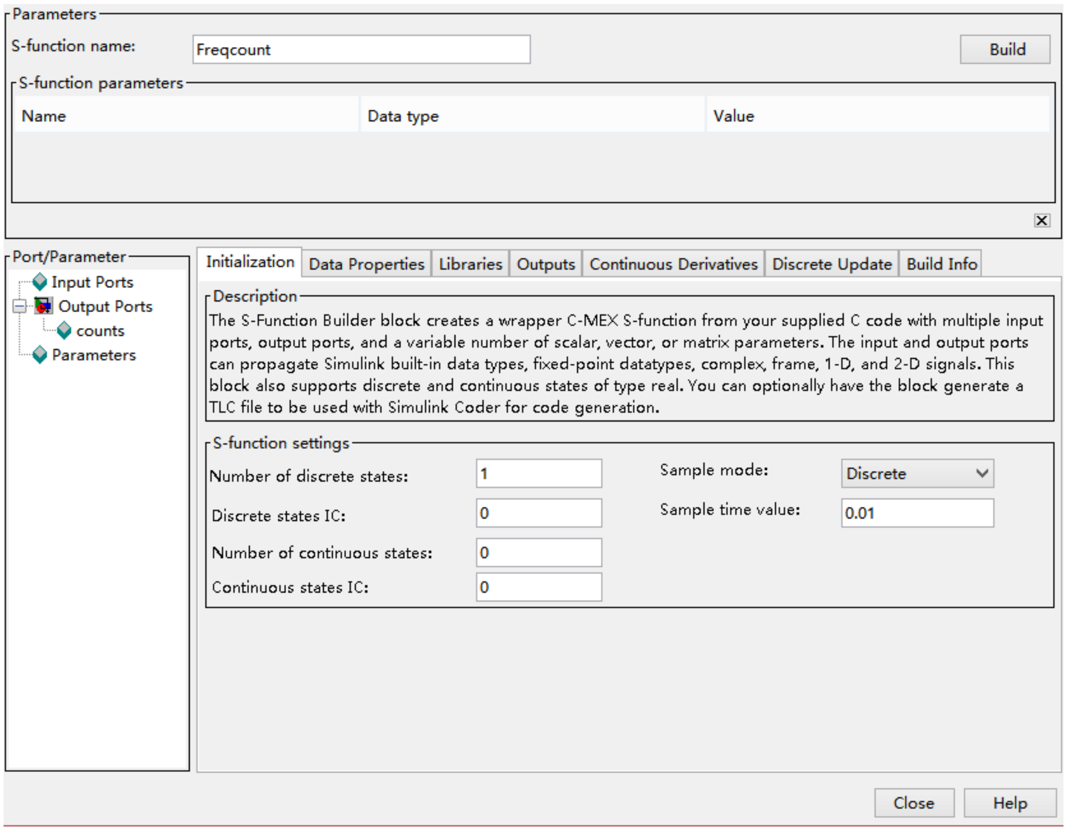
This model has almost the same structure with the control part of the model in Figure 24 and the two PI controllers use the same parameter. Since the Simulink hardware support for Arduino does not contain the specific block count the encoder pulses, we should build our own Simulink device diver block for Arduino (Frecount in Figure 41) to implement the functionality. How to build the Frecount block for counting the pulses is explained later. The 10-bit data for the duty cycle can be outputted by the Arduino Digital Output block in the hardware support library.

3.4.1 Freqcount Block

Since R2012a version, it has been possible to run Simulink models on a set of supported target hardware. However, even more and more features are supported, sometimes we still need build a driver that is not included in the support packages. Some approaches are available including using the MATLAB Function black and the S-Function Builder block. In general, the S-Function builder approach is better for developers who are more familiar with C than the MATLAB Function block and need to quickly develop simple drivers that do not have a lot of parameters, while the MATLAB Function approach works better for developing more complex drivers and blocks that have to be redistributed and masked/24/. Since the Freqcount block is less complex, we choose S-Function builder approach to solve it.

To begin with, the S-Function Builder should be opened and it was named “Freqcount”. The first pane is the Initialization pane as shown in Figure 30. The Initialization pane establishes the block sample time and its number of continuous and discrete states. Normally the driver blocks execute in discrete time and have no continuous states. In this case, we have one discrete state and the sample time is set to 10ms so that the output is updated every 10ms.

The implementation of a driver block requires that we set at least a single discrete-time state, which must be initialized to 0. One could add more states if needed, but the first element of the discrete state vector (xD[0]) must be initialized to 0 in order for the initialization part (introduced later) to work. /24/



S-function name: Freqcount

Name	Data type	Value

Description

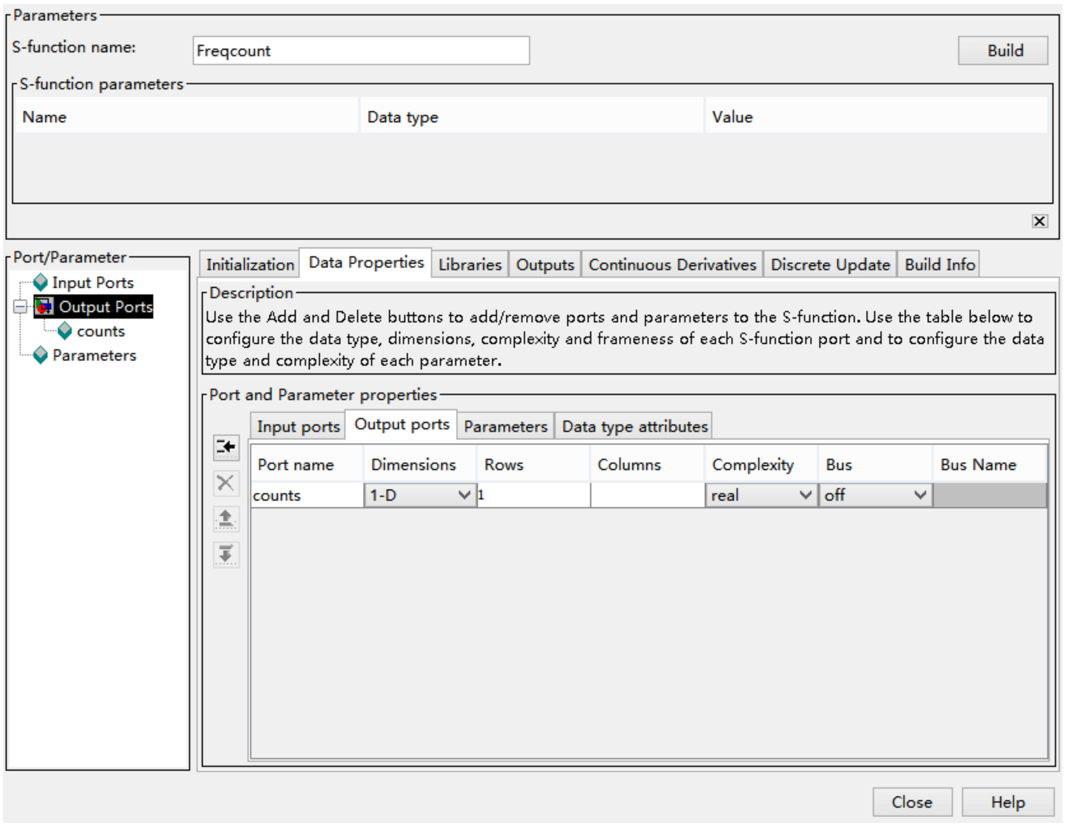
The S-Function Builder block creates a wrapper C-MEX S-function from your supplied C code with multiple input ports, output ports, and a variable number of scalar, vector, or matrix parameters. The input and output ports can propagate Simulink built-in data types, fixed-point datatypes, complex, frame, 1-D, and 2-D signals. This block also supports discrete and continuous states of type real. You can optionally have the block generate a TLC file to be used with Simulink Coder for code generation.

S-function settings

Number of discrete states:	1	Sample mode:	Discrete
Discrete states IC:	0	Sample time value:	0.01
Number of continuous states:	0		
Continuous states IC:	0		

Figure 44. Initialization pane

The second pane is Data Property pane as shown in Figure 43. Here we can define the number and dimensions of input and output ports as well as the parameters passed to the S-Function block. In this case, we do not need an input, and an output “counts” is created to output the number of counted pulses.


Parameters
S-function name: Freqcount

S-function parameters

Name	Data type	Value

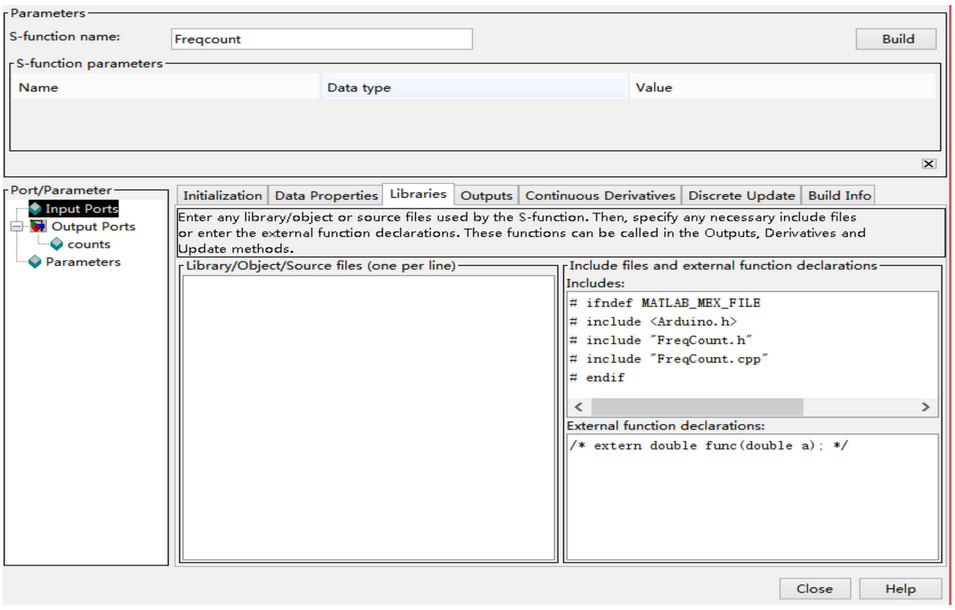
Description
Use the Add and Delete buttons to add/remove ports and parameters to the S-function. Use the table below to configure the data type, dimensions, complexity and frameness of each S-function port and to configure the data type and complexity of each parameter.

Port and Parameter properties

Port name	Dimensions	Rows	Columns	Complexity	Bus	Bus Name
counts	1-D	1		real	off	

Figure 45. Data Property pane

In the Libraries pane, we can specify external libraries, include files, and global variables that are needed by the custom code written in the other panes:


Parameters
S-function name: Freqcount

S-function parameters

Name	Data type	Value

Enter any library/object or source files used by the S-function. Then, specify any necessary include files or enter the external function declarations. These functions can be called in the Outputs, Derivatives and Update methods.

Library/Object/Source files (one per line)

Include files and external function declarations
Includes:

```
# ifndef MATLAB_MEX_FILE
# include <Arduino.h>
# include "FreqCount.h"
# include "FreqCount.cpp"
# endif
```

External function declarations:

```
/* extern double func(double a); */
```

Figure 46. Libraries pane

In our case, only the upper right "Includes" field needs to be considered (the Library/Object/Source field on the left hand side expects a lib-file, e.g. .lib, .a, .so, .obj, .cpp, .c, files having .h or .hpp extension are ignored). The following is the code in the "Includes" field:

```
# ifndef MATLAB_MEX_FILE
# include <Arduino.h>
# include "FreqCount.h"
# include "FreqCount.cpp"
# endif
```

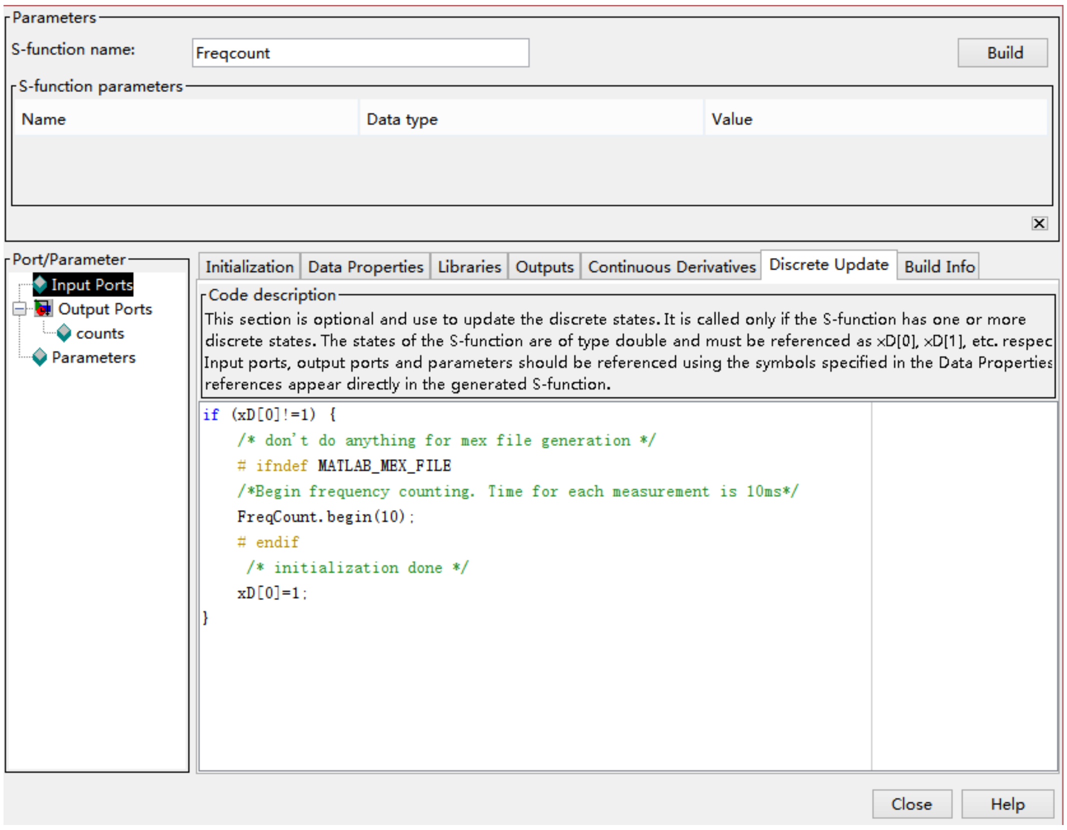
Please note that the conditional compilation instruction # ifndef MATLAB_MEX_FILE used here prevents all the code that follows (until # endif) from being included in the compilation when the MATLAB_MEX_FILE identifier is defined. /24/

When a MEX-file is generated from the S-Function Block (in order for the whole model to be simulated in Simulink), the identifier "MATLAB_MEX_FILE" is defined at compilation time. /24/

As a result, when generating the executable for the simulation, the lines between will not be included in the compilation. On the other hand, when an executable that needs to run on the target hardware is generated, the identifier "MATLAB_MEX_FILE" will not be defined, and as a consequence the lines will be included in the compilation. /24/

The library files included here are the Arduino.h which contains declaration for almost all the fundamental functions for Arduino and an external FreqCount library placed in the current MATLAB folder. All the functions used in this block are declared by the FreqCount library.

The next step is to make the initialization part in the Discrete Update pane as shown in Figure 45. In general, the Discrete Update pane defines the evolution laws for the discrete state vector. However, here it is used to run the initialization code.



This section is optional and use to update the discrete states. It is called only if the S-function has one or more discrete states. The states of the S-function are of type double and must be referenced as xD[0], xD[1], etc. respec
Input ports, output ports and parameters should be referenced using the symbols specified in the Data Properties
references appear directly in the generated S-function.

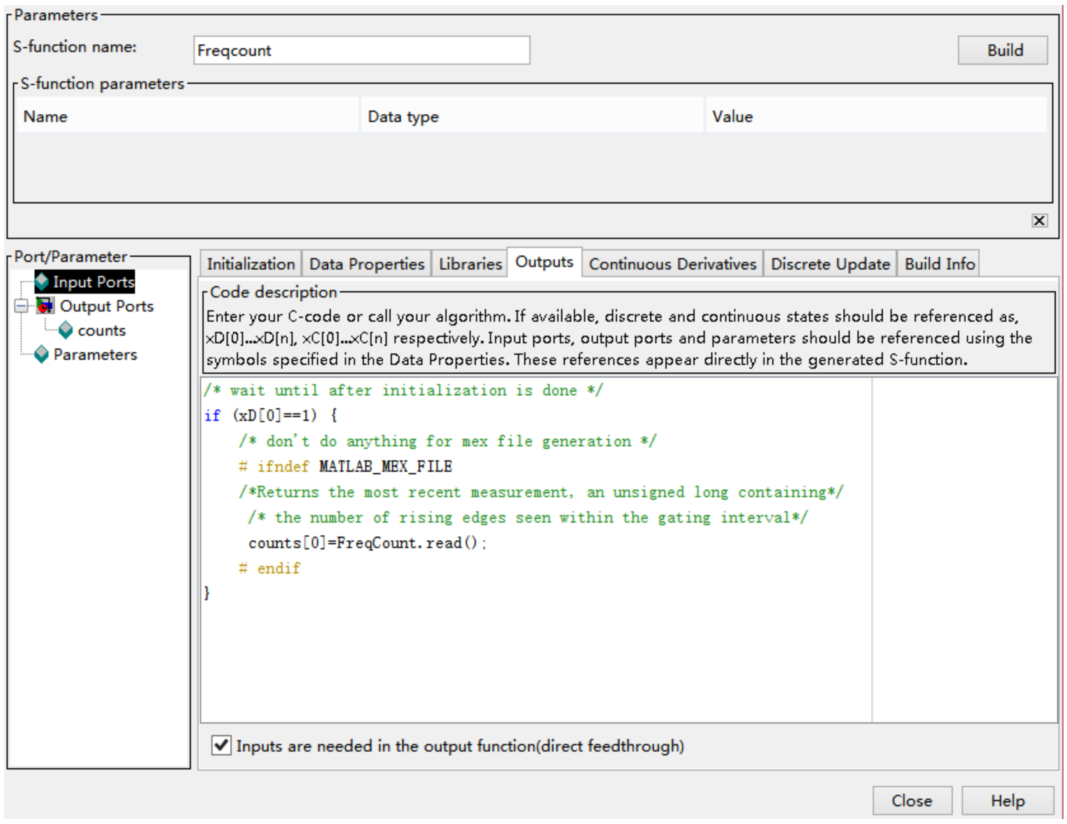
```
if (xD[0]!=1) {
    /* don't do anything for mex file generation */
    # ifndef MATLAB_MEX_FILE
    /*Begin frequency counting. Time for each measurement is 10ms*/
    FreqCount.begin(10);
    # endif
     /* initialization done */
    xD[0]=1;
}
```

Figure 47. Discrete Update pane (Initialization part)

The initial condition for the discrete state is 0 (this is set up by the initialization pane seen in the previous page), therefore the first time this Discrete Update function is called xD[0] is 0 and the code inside the brackets following the "if" condition is executed. The last line inside the brackets sets xD[0] to 1, which prevents anything inside the brackets from being executed ever again. /24/

Here, the function FreqCount.begin(10) is used to initialise the frequency counting. The time for each measurement is 10m, so the counted pulses number is updated every 10ms.

Then we go to the Outputs pane to define the actions that the block performs (in general on its outputs) when it is executed, as shown in Figure 46:



S-function name: Freqcount

Enter your C-code or call your algorithm. If available, discrete and continuous states should be referenced as, xD[0]..xD[n], xC[0]..xC[n] respectively. Input ports, output ports and parameters should be referenced using the symbols specified in the Data Properties. These references appear directly in the generated S-function.

```
/* wait until after initialization is done */
if (xD[0]==1) {
    /* don't do anything for mex file generation */
    # ifndef MATLAB_MEX_FILE
    /*Returns the most recent measurement, an unsigned long containing*/
     /* the number of rising edges seen within the gating interval*/
     counts[0]=FreqCount.read();
    # endif
}
```

☑ Inputs are needed in the output function(direct feedthrough)

Figure 48. Outputs pane

The code in the brackets follows the condition xD[0]==1. Since xD[0] is 0 at the beginning and is then set to 1 by the first discrete update call, this means that the code in the brackets is executed only after the initialization code has already been executed /24/. Here the FreqCount.read() function used to return the measurement and pass the to the output counts.

Now we can go to the Build Info pane and make sure the “Geneta wrapper TLC” checkbox is checked and the “Enable access to SimStruct” is unchecked, and then click on the “Build” button to build it.

If everything goes well six files will be generated: a wrapper file, a simulation-only S-function file, a simulation-only MEX-file, two configuration files, and a TLC-file. The wrapper file contains the code from the dialog box panes which is also referenced both in the MEX-file (for simulation) and in the TLC-file (used to generate the executable which will run on the target). /24/

Since the external library FreqCount is in C++, some steps are necessary to make sure that the compiler and linker know how to handle interoperating C and C++ source files.

First, we need to rename the Freqcount_wrapper.c file generated by S-Function Builder to Freqcount _wrapper.cpp. Then open the file and add **extern "C"** right before the definition of functions.

Now the Freqcount block is completed. Appendix 1 can be referred to see how includes, functions, output part and initialization part are located in the wrapper file.

3.4.2 Data Type Conversion

The output of the PI controller is a decimal number ranging from 0 to 1000. In order to make it easy to transmit the data, we can convert the data into stored integer and extract each bit of the stored integer. Then 10 Arduino Digital Output blocks are used to output the value of the 10 bits respectively, as shown in Figure 41. The data type conversion subsystem is shown in Figure 48. Further, the 10 digital output pins of Arduino are connected to 10 input pins on DE2 to transmit the data to control the PWM generation.

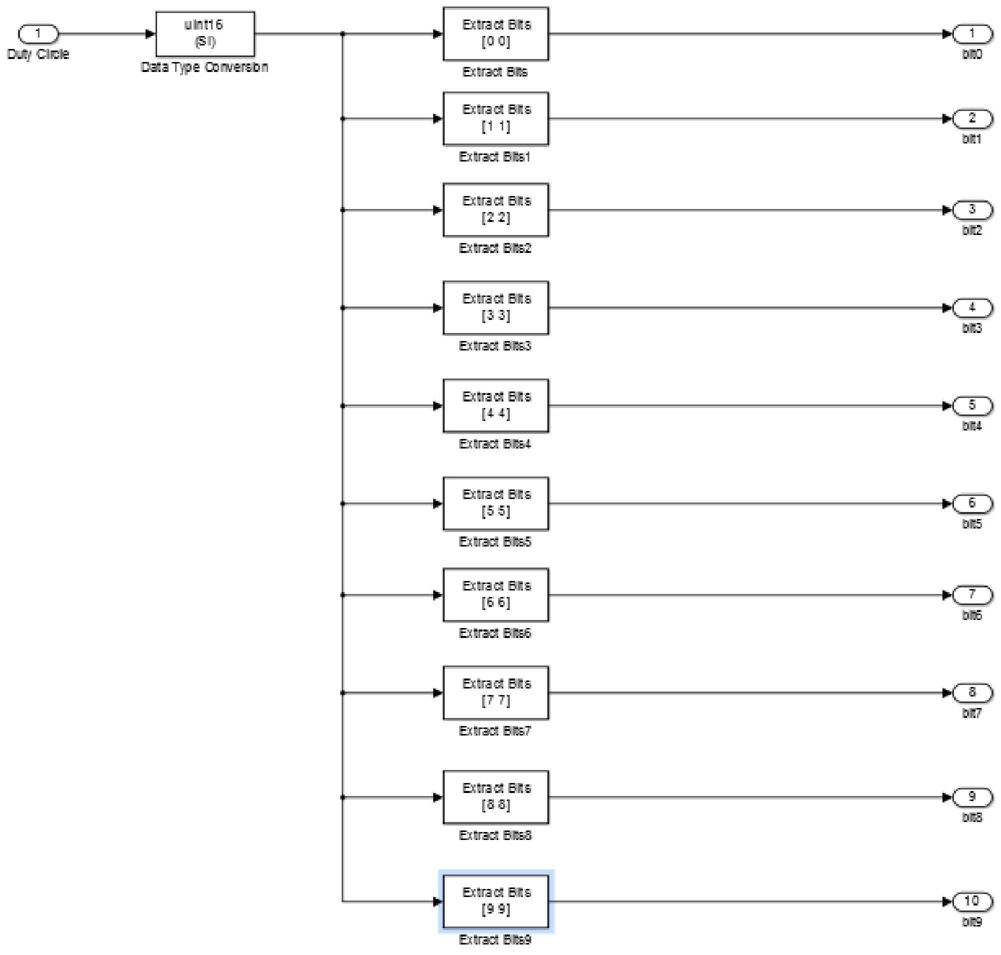


Figure 49. Data type conversion subsystem

Now, the design of single BLDC motor control is finished and the testing results of are discussed in Chapter 5. This design method can be further implemented to design the motion control system discussed in the next chapter based on the Simulink model.

4 MODELLING OF MOTION CONTROL SYSTEM

In the previous chapters, the design of the single BLDC motor speed control system which is a single degree of freedom control system was described. Since the single motor control is optimised, the motion control system of the robot can be discussed as a whole. Figure 48 shows the structure of motion control system. Based on kinematic model, the velocity of the robot (Vx, Vy, w) is decomposed into the speeds of four wheels. In general, the actual speed of the robot is measured by accelerometer and gyroscope while in the model the speed of the robot is derived by the velocity composition of the 4 wheels' speed. Actually the structure shown in Figure 48 contains two closed loops for speed control. The inner one is included in the single motor control system to control the speed of the motor and the outer loop is for the speed control of the robot as a whole. Now, our Botnia robots only have the inner closed loop, and how the absence of the outer loop can influence the overall control performance is shown in Chapter 5.

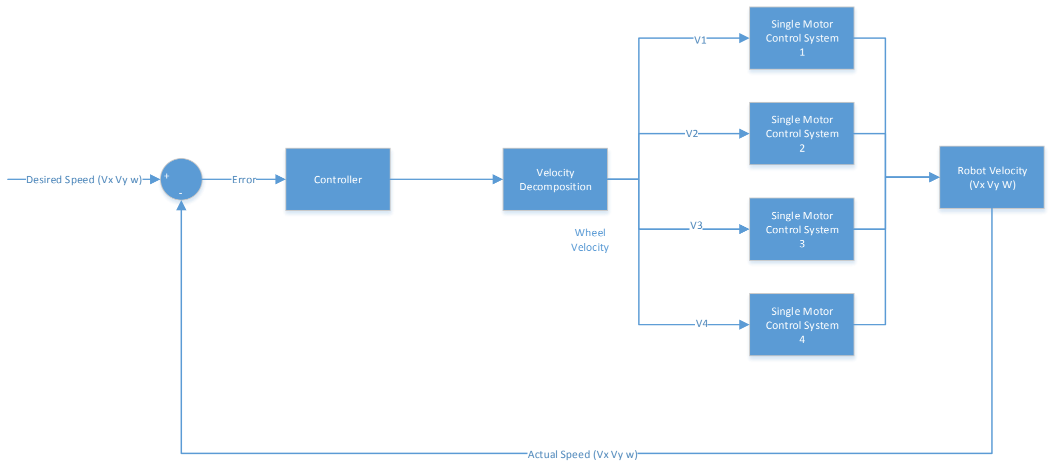


Figure 50. Structure of motion control system

According to the structure of motion control system, the model of the motion control system is built as shown in Figure 49. The speed decomposition and composition based on the kinematic model is implemented by MATLAB Functions. The four motor control systems are under the subsystem named four single motor control systems. The desired robot speed and the actual robot speed with respect to the robot's own coordinate system are further converted to the desired robot trajectory and the actual robot trajectory with respect to the absolute coordinate system. By comparing the two trajectories, we can check the performance of the motion control system.

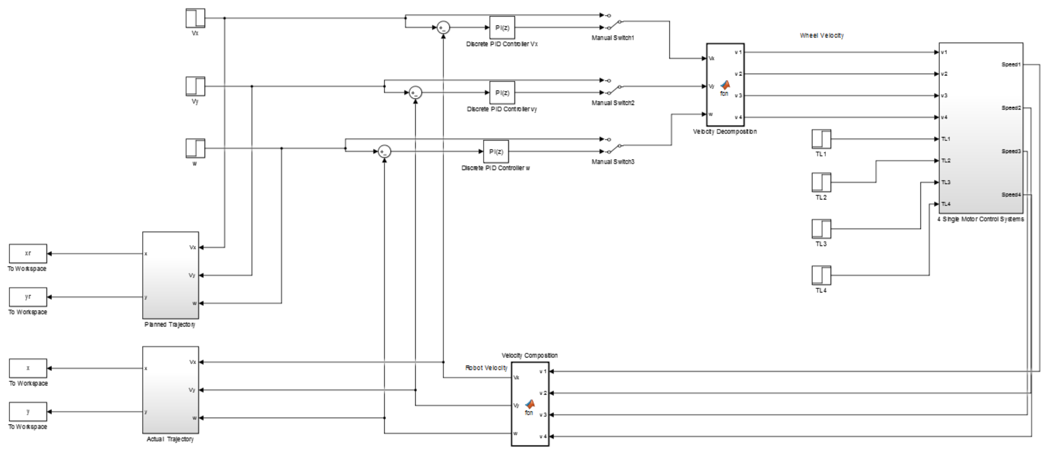


Figure 51. Model of motion control system

Here three PI controllers are used to adjust the Vx, Vy, w respectively. Due to the feedback of the three speeds are coupled, the parameters of each PI controller are optimised using the same method described in Chapter 2 when other two controllers are not connected into the system (the other two loops are open loops).

4.1 Kinematic Model

The fundamental of the speed composition and decomposition is the kinematic model of the robot. In order to obtain the kinematic model, the arrangement of the wheels as well as the speeds need to be analysed. Figure 50 shows the arrangement of the wheels of our Botnia robot with the coordinate systems ($\delta_1 = 45°, \delta_1 = 54°, R = 0.078m$):

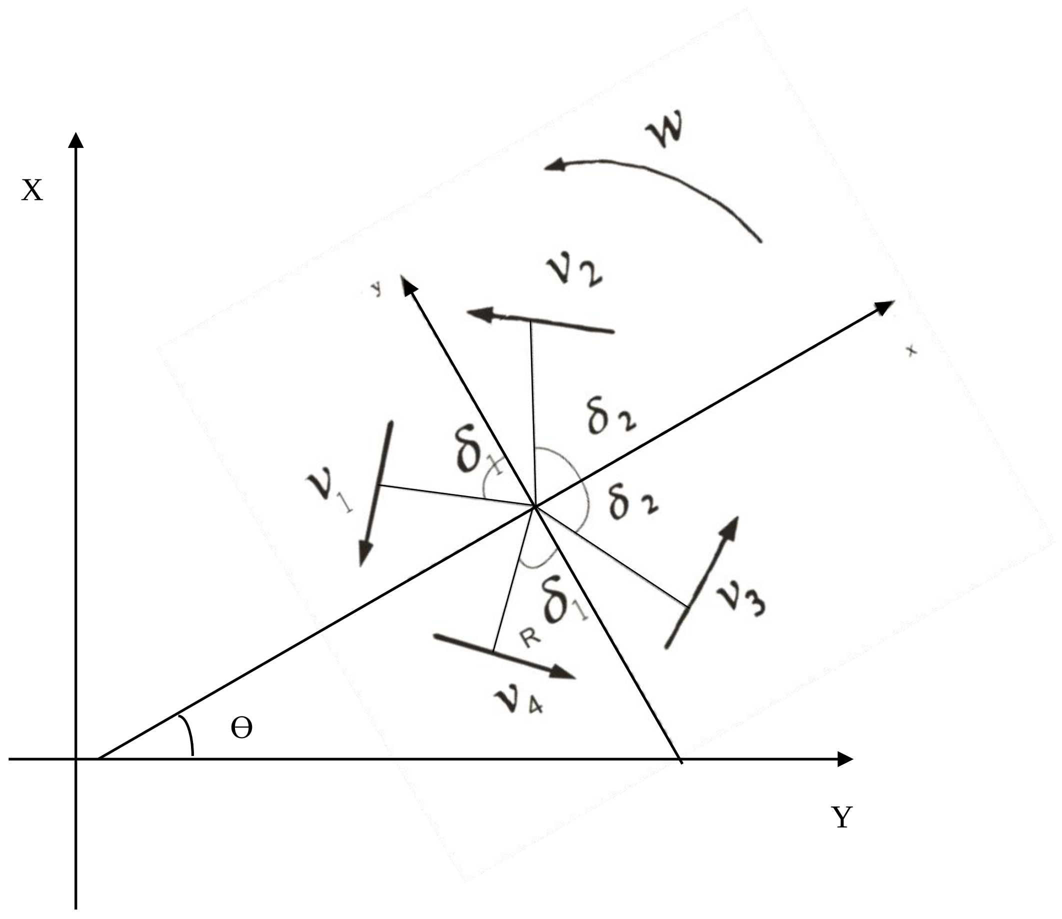


Figure 52. Arrangement of the wheels

By analysing the geometrical relationships shown in Figure 50, we can obtain the following four equations which described how the four wheel velocities V1, V2, V3, and V4 are related with Vx, Vy, and w:

$$V_1 = -V_x \cos\delta_1 - V_y \sin\delta_1 + \omega R \tag{19}$$

$$V_2 = -V_x \sin\delta_2 + V_y \cos\delta_2 + \omega R \tag{20}$$

$$V_3 = V_x \sin\delta_2 + V_y \cos\delta_2 + \omega R \tag{21}$$

$$V_4 = V_x \cos\delta_1 - V_y \sin\delta_1 + \omega R \tag{22}$$

The above equations can be written as:

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{pmatrix} = \begin{pmatrix} -\cos\delta_1 & -\sin\delta_1 & R \\ -\sin\delta_2 & \cos\delta_2 & R \\ \sin\delta_2 & \cos\delta_2 & R \\ \cos\delta_1 & -\sin\delta_1 & R \end{pmatrix} \begin{pmatrix} V_x \\ V_y \\ \omega \end{pmatrix} \quad (23)$$

We assume that:

$$V = \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{pmatrix} \quad (24)$$

$$M(\theta) = \begin{pmatrix} -\cos\delta_1 & -\sin\delta_1 & R \\ -\sin\delta_2 & \cos\delta_2 & R \\ \sin\delta_2 & \cos\delta_2 & R \\ \cos\delta_1 & -\sin\delta_1 & R \end{pmatrix} \quad (25)$$

$$S = \begin{pmatrix} V_x \\ V_y \\ \omega \end{pmatrix} \quad (26)$$

Then the following equation can also be obtained:

$$S = M^{-1}(\theta)V \quad (27)$$

The Equation 11 and Equation 12 are the mathematical kinematic model of the robot.

Since $M(\theta)$ is not a square matrix which means the system is redundant, there is not only one solution for $M^{-1}(\theta)$.

4.2 Four Single Motor Control Systems

This subsystem contains four independent single motor control systems representing the four wheels respectively. The model of the subsystem is shown in Figure 51. We assume the gear ratio is 1 and the radius of the wheel is 0.02m. Where NA is the number of teeth on the input gear, NB is the number of teeth on the output gear and R is the radius of the wheel, the following equation which describes the relationship between rotational speed of motor and the linear velocity of the wheel is formed:

$$\omega = \frac{30N_B}{\pi N_A} V/R \tag{28}$$

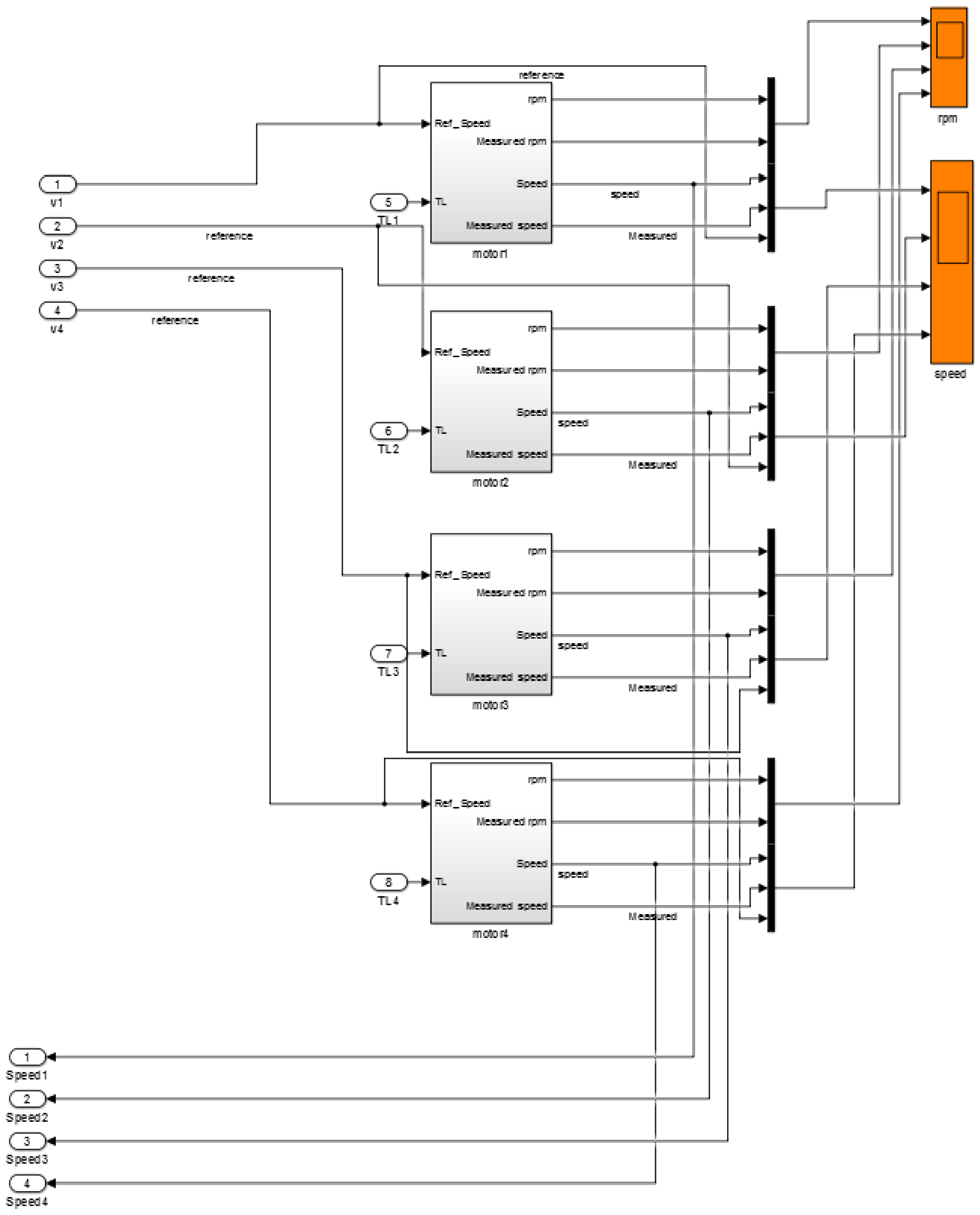


Figure 53. Model of four single motor control systems

4.3 Velocity Decomposition

In our situation, the radius of the robot R is 0.078m, $\delta_1 = 45°, \delta_1 = 54°$. Therefore, according to the kinematic model, the MATLAB function for velocity decomposition can be written as follow:

```
function [v1,v2,v3,v4] = fcn(Vx,Vy,w)
v1 = -cos(45*pi/180)*Vx-sin(45*pi/180)*Vy+0.078*w;
v2 = -sin(54*pi/180)*Vx+cos(54*pi/180)*Vy+0.078*w;
v3 = sin(54*pi/180)*Vx+cos(54*pi/180)*Vy+0.078*w;
v4 = cos(45*pi/180)*Vx-sin(45*pi/180)*Vy+0.078*w;
```

4.4 Velocity Composition

$M(\theta)$ could have several pseudo-inverse matrixes, and the following is one of them:

$$M^{-1}(\theta) = \begin{pmatrix} -\frac{1}{4\cos\delta_1} & -\frac{1}{4\sin\delta_2} & \frac{1}{4\sin\delta_2} & \frac{1}{4\cos\delta_1} \\ -\frac{1}{4\sin\delta_1} & \frac{1}{4\cos\delta_2} & -\frac{1}{4\cos\delta_2} & -\frac{1}{4\sin\delta_1} \\ \frac{\sin\delta_1+\cos\delta_2}{8R\sin\delta_1} & \frac{\sin\delta_1+\cos\delta_2}{8R\sin\delta_1} & \frac{\sin\delta_1+\cos\delta_2}{8R\sin\delta_1} & \frac{\sin\delta_1+\cos\delta_2}{8R\sin\delta_1} \end{pmatrix} \tag{29}$$

Therefore, according to the kinematic model the following equation can be derived:

$$V_x = -\frac{(V_1 - V_4)}{4\cos\delta_1} - \frac{(V_2 - V_3)}{4\sin\delta_2} \tag{30}$$

$$V_y = -\frac{(V_1 + V_4)}{4\sin\delta_1} + \frac{(V_2 + V_3)}{4\cos\delta_2} \tag{31}$$

$$\omega = \frac{\sin\delta_1 + \cos\delta_2}{8R}\left(\frac{(V_1 + V_4)}{\sin\delta_1} + \frac{(V_2 + V_3)}{\cos\delta_2}\right) \tag{32}$$

Then the corresponding MATLAB function can be written as follow:

```
function [Vx,Vy,w] = fcn(v1,v2,v3,v4)
Vx=-(v1-v4)/(4*cos(45*pi/180))-(v2-v3)/(4*sin(54*pi/180));
Vy=-(v1+v4)/(4*sin(45*pi/180))+(v2+v3)/(4*cos(54*pi/180));
w=((v1+v4)/(sin(45*pi/180))+(v2+v3)/(cos(54*pi/180)))*(sin(45*pi
/180)+cos(54*pi/180))/(8*0.078);
```

4.5 Trajectory

In order to obtain the trajectory of the robot with regard to the absolute coordinate system, we need to analyse the relationship between the absolute coordinate system and the robot's own coordinate system shown in Figure 50 and find the speed according to the absolute coordinate system. Then the following equations can be derived:

$$V_X(t) = V_x(t)\cos(\theta(t)) - V_y(t)\sin(\theta(t)) \tag{33}$$

$$V_Y(t) = V_y(t)\cos(\theta(t)) + V_x(t)\sin(\theta(t)) \tag{34}$$

$$\omega(t) = \omega(t) \tag{35}$$

$$X(t) = \int V_X(t)\,dt \tag{36}$$

$$Y(t) = \int V_Y\,(t)dt \tag{37}$$

$$\theta(t) = \int \omega(t)\,dt \tag{38}$$

Therefore, the model can be built as shown in Figure 51 and the code can be written as follow:

```
function [VX,VY] = fcn(Vx,Vy,theta)
VX=Vx*cos(theta)-Vy*sin(theta);
VY=Vx*sin(theta)+Vy*cos(theta);
```

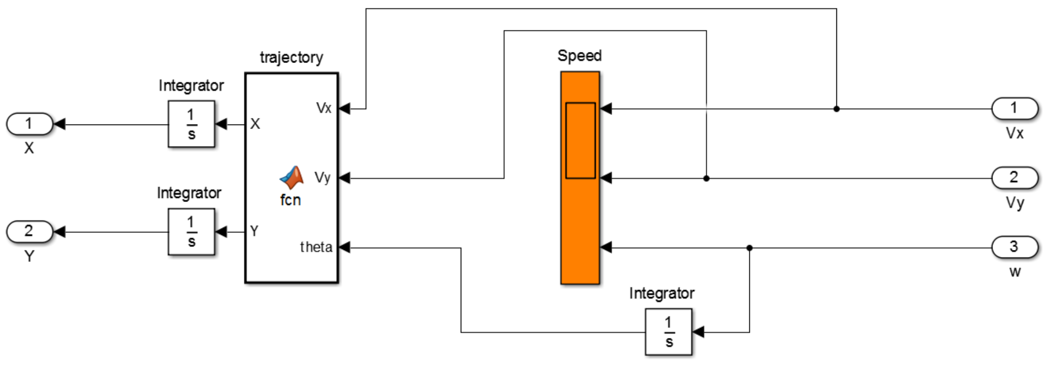


Figure 54. Trajectory subsystem

The modelling of motion motor control system is completed. The testing results of the simulation are discussed in Chapter 5.

5 TESTING RESULTS AND ANALYSIS

In this chapter, the testing results are presented and analysed.

5.1 Simulation Results of BLDC Motor

The simulation time is 1 second and the source voltage is 12V at the beginning, then decreased to 6V at 0.5 seconds. A load torque of 30 mNm is applied at 0.8 seconds. Figure 53 shows the actual motor speed, and the electrical torque is shown in Figure 54. Figure 55 shows the phase currents and the zoomed view is shown in Figure 56. We can see that the no-load speed and no-load current are quite close to the values in Figure 5.

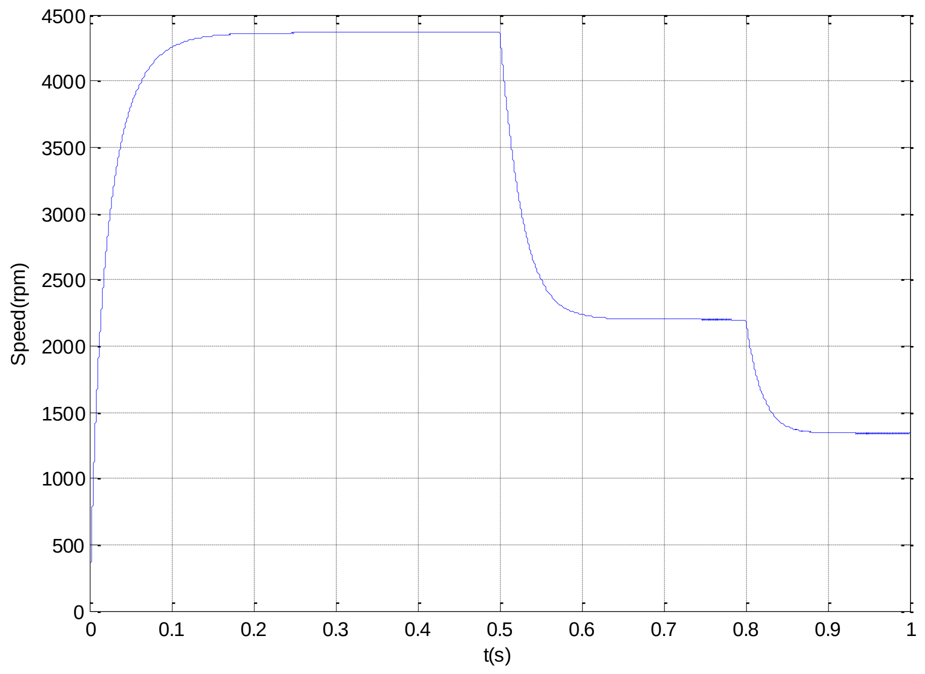


Figure 55. Rotor speed

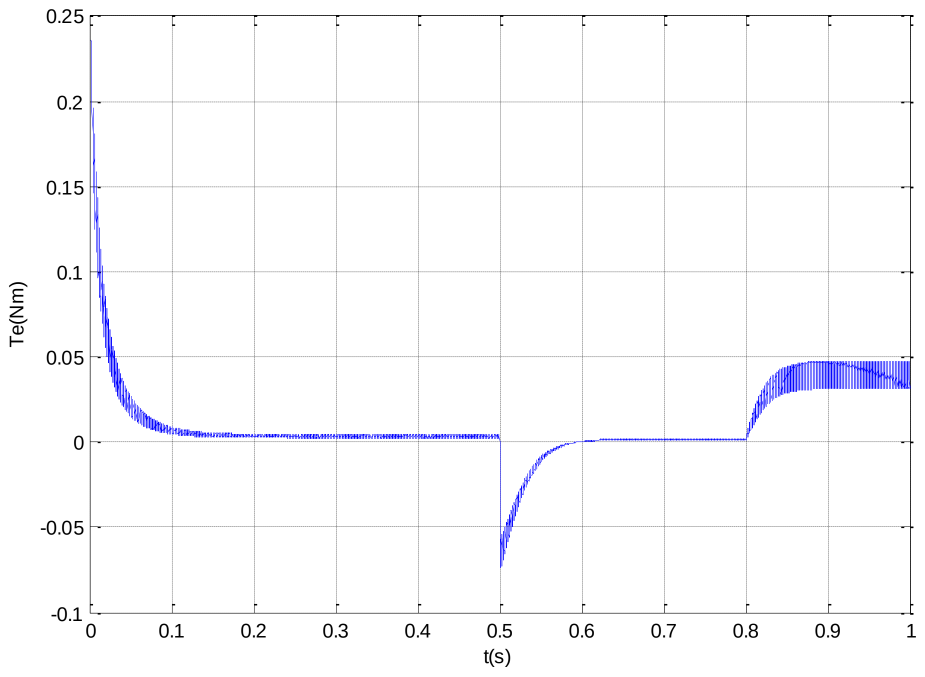


Figure 56. Electrical torque

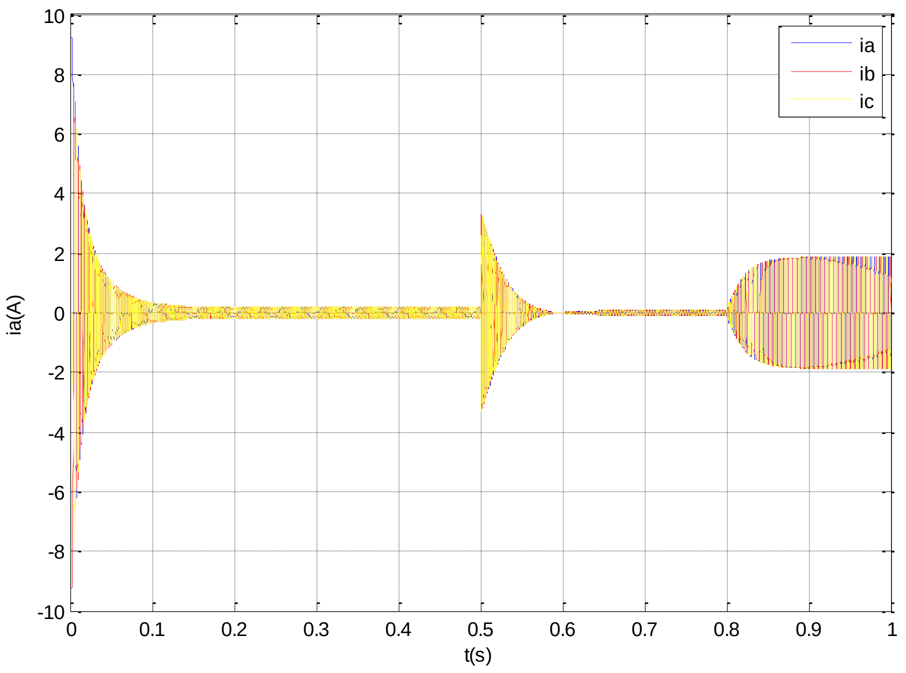


Figure 57. Phase currents

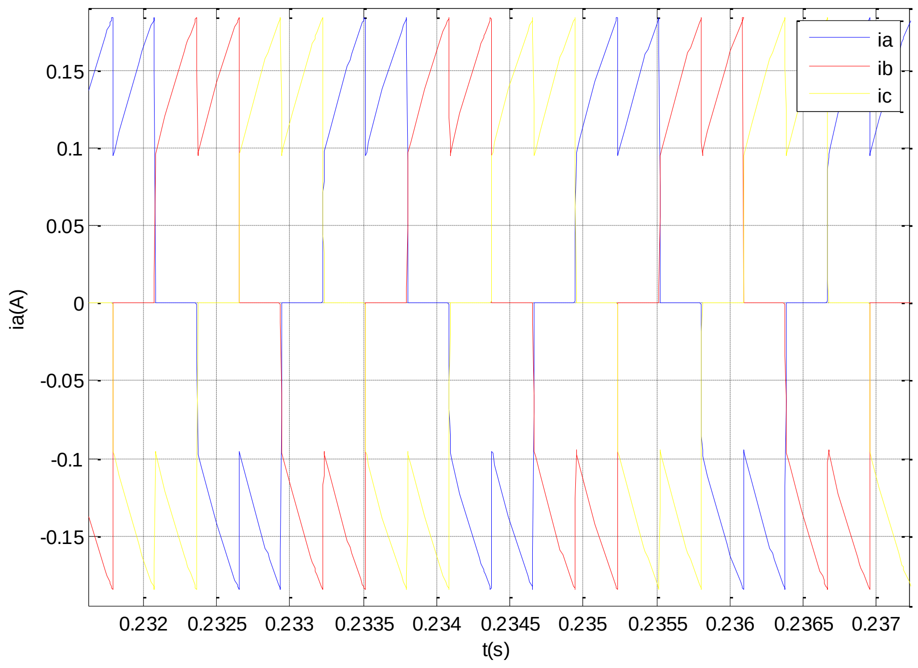


Figure 58. Zoomed view of phase currents

5.2 Single BLDC Motor Control System

Figure 57 shows the simulation result of the single BLDC motor control system. The simulation time is 25 seconds. The set point is set to 2000 rpm from 3 seconds to 10 seconds and then set to 1200 rpm from 10 seconds to 17 seconds and from 17 seconds, set to 400 rpm to the end. A load torque of 30 mNm is applied from 5 seconds to 7.5 seconds and then the same load torque is applied from 12 seconds to 15 seconds and from 20.5 seconds to 24 seconds. We can see how the PID controller attempts to minimize the error.

The testing result of the motor control system is shown in Figure 58. The running time is 25 seconds. The set points are the same as in the simulation. The moments to apply the torque are also similar. The closed-loop testing result is quite similar to the simulation result. This proves that our single BLDC motor control system has a reliable performance.

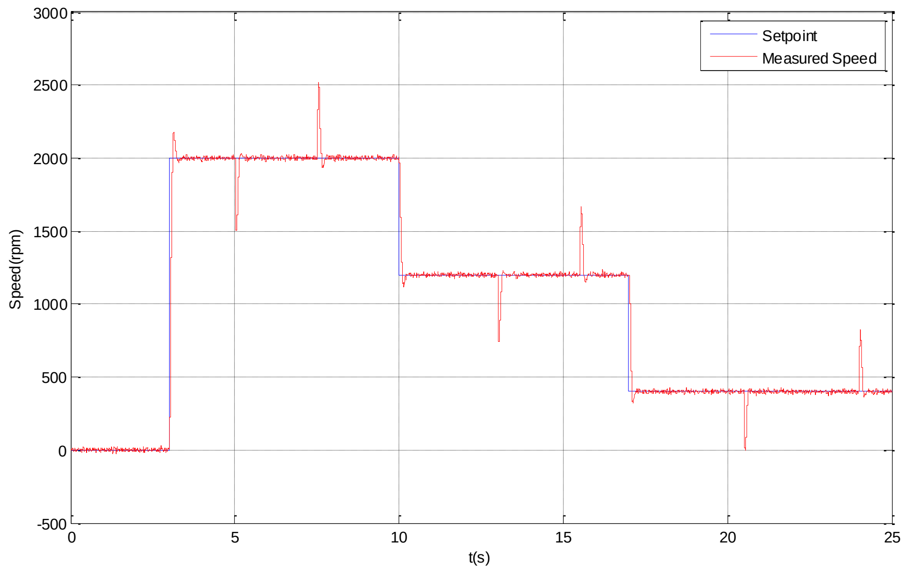


Figure 59. Closed-loop simulation result

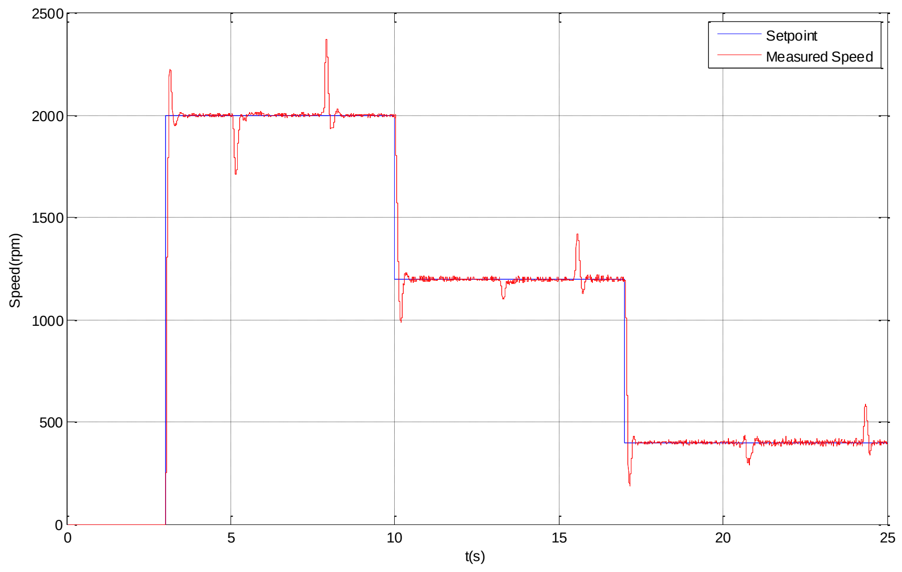


Figure 60. Closed-loop testing result

5.3 Simulation Results of Motion Control System

The simulation results of motion control system are mainly used to compare the actual trajectory and the planned trajectory to discuss the control performance of the whole system. The simulation time is 20 seconds. The desired Vx is 3m/s from the beginning to 5 seconds and 1.3m/s from 5 seconds to the end. The desired Vy is 2m/s from the beginning to 10 seconds and 3.5m/s from 10 seconds to the end. The desired w is 0rad/s from the beginning to 5 seconds and 0.9rad/s from 5 seconds to the end. The following three figures show the actual trajectories and planned trajectories in three different situations. The first one shows the situation when the PID controllers for both the whole robot velocities and the single BLDC motors are optimized. The second one shows when the whole system only has the PID controllers for each motor. The last one shows when there is no PID controllers at all. It can be seen from the second figure that the motion control system already has a reasonable control performance. However, in order to achieve higher performance, the velocity controllers for the overall speeds are needed.

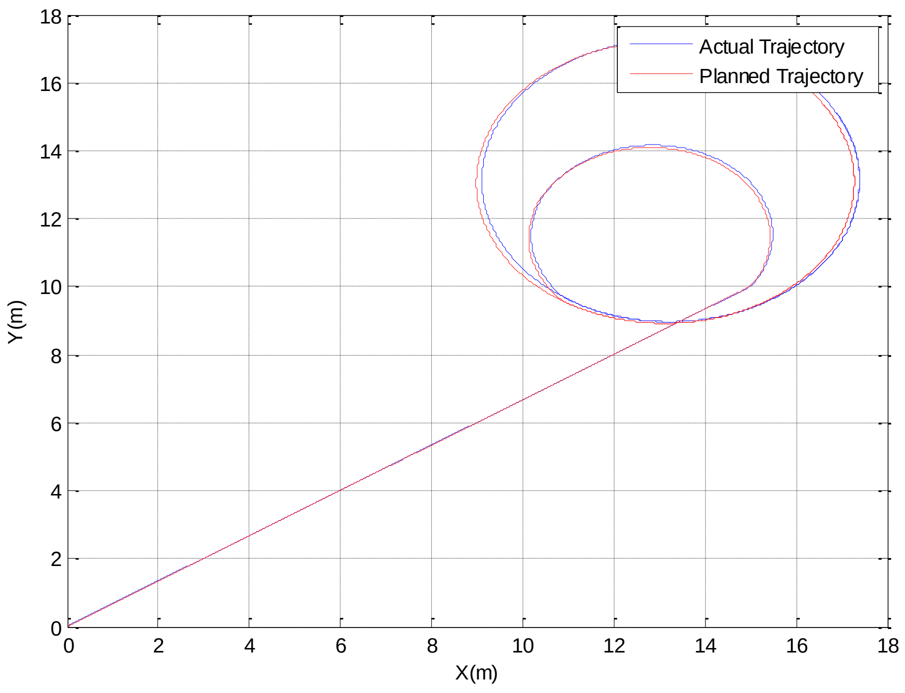


Figure 61. Actual trajectory vs planned trajectory (1)

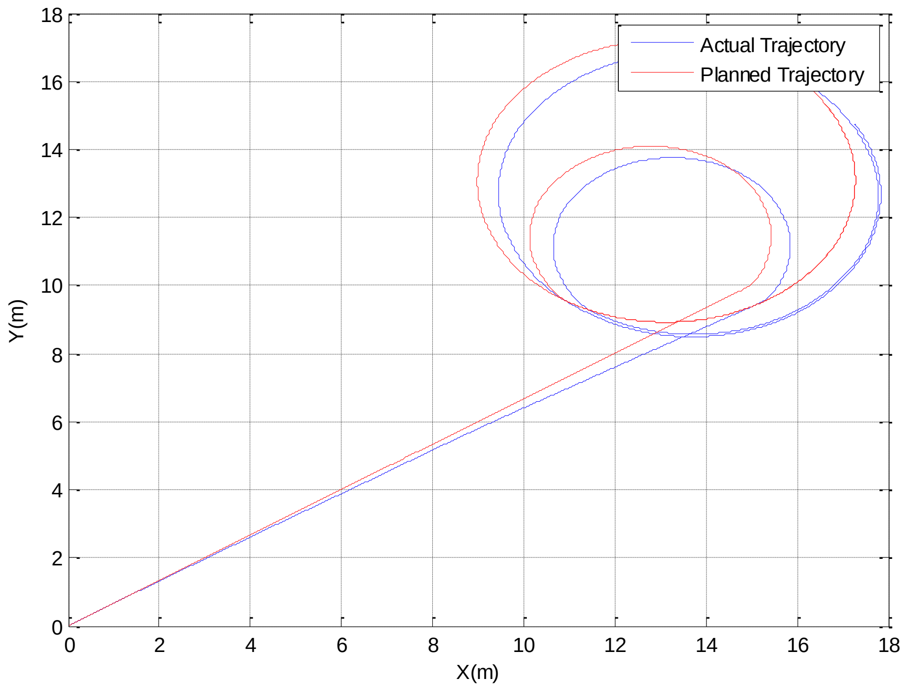


Figure 62. Actual trajectory vs planned trajectory (2)

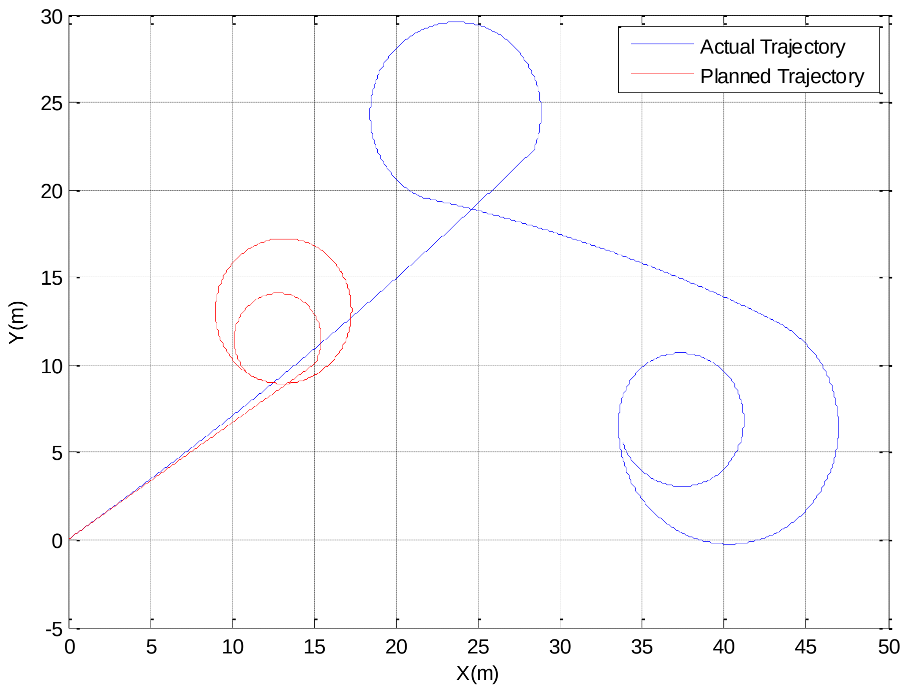


Figure 63. Actual trajectory vs planned trajectory (3)

6 DISCUSSION AND FUTURE RESEARCH

The single BLDC motor control system has a reasonable and reliable control performance and the model of the motion control system. However, the RoboCup competition is highly competitive, therefore, there is still some future work based on this project need to be done to improve the robot system.

6.1 Rotational Speed Measurement

In order to achieve high control performance, one of the most important factors is the precision of the feedback and many things can influence the precision of the feedback. In this project, we only consider motor rotating in one direction, so the method to measure the rotational speed is only suitable for one direction rotation since the only one signal from the quadrature encoder is used. To obtain the rotational speed as well as the rotational direction we need to use both two signals. However, this causes more hardware consumption and Arduino only may not be powerful enough to implement it precisely.

Another aspect which can influence the precision of the feedback is how we calculate the frequency. In our case, the measured speed is calculated by counting the pulses in every 10ms, and this is so called the M method. It has a high accuracy when the speed is high. Another method, the T method, is to calculate the speed by measuring the time interval between two pulses, and it has a high accuracy when the speed is low. Moreover, an M/T method is the synthesis of the two methods, but it is more difficult to implement. In order to achieve a high precision and high control dynamic, we need to choose a proper method according to the situations.

In addition, the encoder noises need to be considered and filters may be needed to eliminate the noises.

6.2 Deviation of Mass Centre

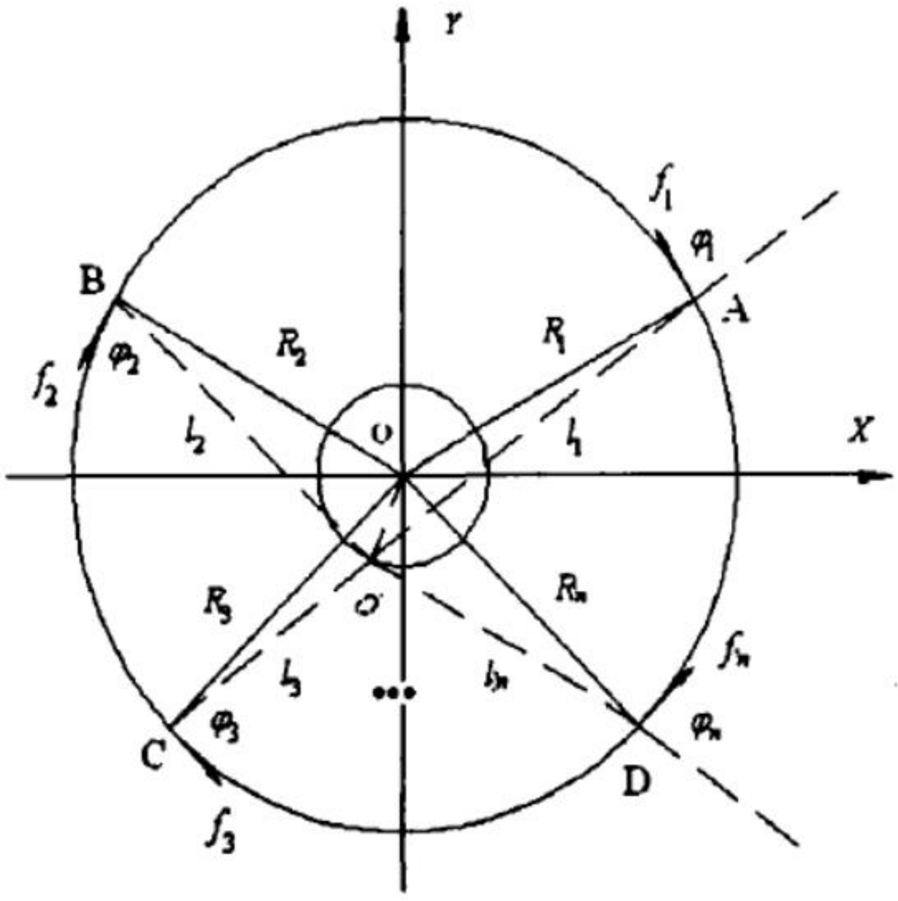


Figure 64. Deviation of mass centre /25/

In Chapter 4, the kinematic model gives an ideal model of the robot when only the geometric centre is considered. When considering further the acceleration of the robot and the force of each motor, the mass centre needs to be taken into consideration and it is not the geometric centre in most situations.

6.3 Slipping Wheels

Because of the fact that we use the pseudo-inverse matrix to do the speed composition, we need to confirm whether the wheels are slipping. The four wheels are a redundant system, so that the speed of one motor can be deduced by the speeds of other three motors when there is no slippage.

For example, if we trust the speed of first three motors, the speed of fourth motor is a linear combination of the other three. If the linear combination does not correspond to the measurement, then we know that a wheel is slipping. We do not know exactly which, but usually the fastest wheel is the one slipping./26/

6.4 Design of Entire Motion Control System and SoC Solution

In this thesis project, only the control system for the single BLDC motor was designed and improved by Model-Based Design method. And this method is proved to be efficient and effective. The next step is to design and improve the entire motion control system using the same method based on the model built in Chapter 4. The future design is highly suggested to be implemented using SoC solution and Altera DE1-SoC board is a good option since it integrates an ARM-based hard processor system (HPS) consisting of the latest dual-core Cortex-A9 embedded cores, peripherals and memory interfaces tied seamlessly with the FPGA fabric using a high-bandwidth interconnect backbone. The Block diagram of the DE1-SoC board is shown in Figure 65. The programmable logic paired with a high-performance processor system will be powerful enough to implement and improve the control system for the single BLDC motor as well as entire motion control system.

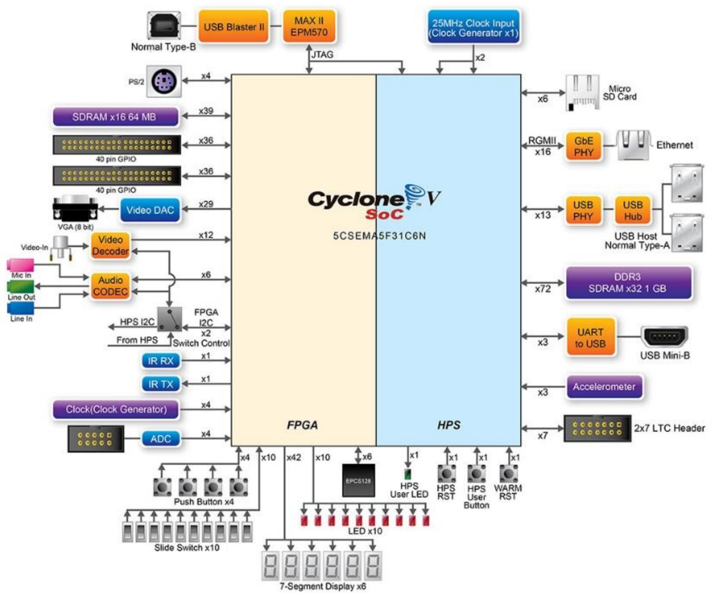


Figure 65. Block diagram of the DE1-SoC board/27/

7 CONCLUSION

In this thesis, the Model-Based Design method which is an efficient and cost-effective way to develop embedded systems was implemented successfully to design the single BLDC motor control system of the motion control system of Botnia robot with MATLAB/Simulink, Arduino Mega 2560 and Altera DE2. In the meantime, the model of the motion control system of SSL robots was built successfully to simulate the motion control process to discuss the performance and can be used for further design and deeper research.

In the design of the single BLDC control system, the Model-Based Design method proved to be a time saving and effective way to design embedded system. First, the model of the BLDC motor and the control system were built for simulation and most importantly for PID tuning to obtain optimal parameters for PID controller. Based on the model, the motor control system was separated into two parts and implemented on Arduino Mega 2560 and Altera DE2 separately. For the part of control system on Altera DE2, the model of the system was built to simulate the system and verify the design, and then the HDL coder was used to generate the VHDL code from the model. For the other part of the control system on Arduino, Simulink hardware support for Arduino was used, so that we only needed to build the model of the system and the code could be automatically generated and deployed on Arduino. This method can make one free from coding and ensure the consistency between the code and the model to make it simple to verify the design. For the modelling of the motion control system, the model was built based on an ideal kinematic model to simulate the control process. It consists of the following parts: four BLDC control systems, velocity decomposition model, velocity composition model and trajectory generation models. The desired and actual trajectories are used to discuss the control performance.

The objective of the thesis to find the ways to improve the motion control system of our Botnia robots was achieved, yet some further improvements need to be implemented. Firstly, the motor is only considered rotating in one direction, so the method to measure the rotational speed is only suitable for one direction rotation.

This must be improved so that we can measure the rotational speed as well as the rotational direction. Further, more efforts are needed to achieve higher measurement precision. As for the motion control system modelling, the deviation of mass centre and slipping wheels should be taken into consideration as well.

In the future, the single BLDC motor control system can be transplanted on our robots. Moreover, the entire motion control system will be redeveloped and improved by the same method based on the model of the motion control system. More students will join our Botnia Robosoccer Team for high achievement and continuous improvement. We wish all the students working on this project the best for their future study and career.

REFERENCES

/1/ RoboCup. Wikipedia. Accessed 20.04.2015.
http://en.wikipedia.org/wiki/RoboCup

/2/ Robocup Soccer. Robocup2014. Accessed 20.04.2015.
http://www.robocup2014.org/?page_id=51

/3/ Robocup Soccer. Robocup2015. Accessed 20.04.2015.
http://www.robocup2015.org/show/league/8.html

/4/ James, B., Stefan, Z., Mike, L. & Manuela, V. 2008. CMDragons: Dynamic Passing and Strategy on a Champion Robot Soccer Team. 2008 IEEE International Conference on Robotics and Automation Pasadena, vol., no., pp.4074, 4079.

/5/ Botnia Robot Flyer. VAMK. Accessed 20.04.2015.
http://robotics.puv.fi/uploads/images/Botnia_robot_flyer.pdf

/6/ Simulink - Simulation and Model-based Design. MathWorks. Accessed 20.04.2015. http://se.mathworks.com/products/simulink/

/7/ Development of embedded systems using Model-Based Design (1/4). Accessed 20.04.2015. http://www.cipher.pe.kr/tt/cipher/167

/8/ Lennon, Tony. April 08, 2008 . A model-based approach to developing mechatronic systems. Embedded. Accessed 20.04.2015.
http://www.embedded.com/design/prototyping-and-development/4007549/A-model-based-approach-to-developing-mechatronic-systems

/9/ Arduino Support from MATLAB. MathWorks. Accessed 20.04.2015.
http://se.mathworks.com/hardware-support/arduino-matlab.html

/10/ Arduino Support from Simulink. MathWorks. Accessed 20.04.2015.
http://se.mathworks.com/hardware-support/arduino-simulink.html

/11/ HDL Coder. MathWorks. Accessed 20.04.2015.
http://se.mathworks.com/products/hdl-coder/

/12/ Brushless DC electric motor. Wikipedia. Accessed 20.04.2015.
http://en.wikipedia.org/wiki/Brushless_DC_electric_motor

/13/ Padmaraja Tedamale. Brushless DC (BLDC) Motor Fundamentals. Microchip Technology Inc. Accessed 20.04.2015.
http://ww1.microchip.com/downloads/en/AppNotes/00885a.pdf

/14/ PID controller. Wikipedia. Accessed 20.04.2015.
http://en.wikipedia.org/wiki/PID_controller

/15/ EC 45 flat. Maxon Motor. Accessed 20.04.2015.
http://wiki.robojackets.org/mediawiki/images/e/e0/07_197_e.pdf

/16/ He, Shilei. 2013. Modeling of Motion Control System for the Soccer Robot. VAMK. Accessed 20.04.2015.
http://www.theseus.fi/bitstream/handle/10024/59267/He_Shilei.pdf?sequence=1

/17/ Maxon EC Motor. Maxon Motor. Accessed 20.04.2015.
http://www.maxonmotor.com/medias/sys_master/8803451338782/maxonECmotor-Notes.pdf?attachment=true

/18/ Baldursson,Stefán . May, 2005. BLDC Motor Modelling and Control – A Matlab®/Simulink® Implementation. Accessed 20.03.2015.
http://webfiles.portal.chalmers.se/et/MSc/BaldurssonStefanMSc.pdf

/19/ Rajne, Milan. Nov 8, 2013. Generate your own commutation table: Trapezoidal control 3-phase BLDC motors using hall sensors. Texas Instruments. Accessed 20.04.2015.
https://e2e.ti.com/blogs_/b/motordrivecontrol/archive/2013/11/08/generate-your-own-commutation-table-trapezoidal-control-3-phase-bldc-motors-using-hall-sensors

/20/ DC Motor Speed: Simulink Modeling. Control Tutorials for MATLAB and Simulink. Accessed 20.03.2015.
http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SimulinkModeling

/21/ L6234. STMicroelectronics. Accessed 20.04.2015.
http://www.st.com/web/en/resource/technical/document/datasheet/CD00000046.pdf

/22/ Double dabble. Wikipedia. Accessed 20.04.2015.
http://en.wikipedia.org/wiki/Double_dabble

/23/ DE2 Development and Education Board User Manual. Altera. Accessed 20.04.2015.
ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Boards/DE2/DE2_User_Manual.pdf

/24/ Campa, Giampiero. Writing a Simulink Device Driver block: a step by step guide. MATLAB CENTRAL. Accessed 20.04.2015.
http://www.mathworks.com/matlabcentral/fileexchange/39354-device-drivers

/25/ Yu,Huayu . 2007. The Kinematic and Dynamic Analysis Based on F-180 Soccer Robot. Dalian University of Technology.

/26/ Rojas, Raul. Omnidirectional Control. Freie Universitat Berlin.

/27/ DE1-SoC Board. Terasic Inc. Accessed 20.04.2015. http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836&PartNo=2#section

APPENDICES

APPENDIX 1. Freqcount_wrapper.cpp

```
/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO
_END */
# ifndef MATLAB_MEX_FILE
# include <Arduino.h>
# include "FreqCount.h"
# include "FreqCount.cpp"
# endif
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO
_BEGIN */
#define u_width
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO
_END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO
_BEGIN */

/*
 * Output functions
 *
 */
extern "C" void Freqcount_Outputs_wrapper(real_T *counts,
           const real_T *xD)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO
_END */
/* wait until after initialization is done */
if (xD[0]==1) {
    /* don't do anything for mex file generation */
    # ifndef MATLAB_MEX_FILE
    /*Returns the most recent measurement, an unsigned long
containing*/
     /* the number of rising edges seen within the gating
interval*/
     counts[0]=FreqCount.read();
    # endif
}
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO
_BEGIN */
}

/*
  * Updates function
  *
  */
extern "C" void Freqcount_Update_wrapper(const real_T *counts,
           real_T *xD)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO
_END */
if (xD[0]!=1) {
    /* don't do anything for mex file generation */
    # ifndef MATLAB_MEX_FILE
    /*Begin frequency counting. Time for each measurement is
10ms*/
    FreqCount.begin(10);
    # endif
     /* initialization done */
    xD[0]=1;
}
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO
_BEGIN */
}
```

APPENDIX 2. Extract_Bits.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Extract_Bits IS
  PORT( In_rsvd                           :   IN
std_logic_vector(31 DOWNTO 0);  -- uint32
        uni_o                            :   OUT   std_logic_vector(3
DOWNTO 0);  -- ufix4
        ten_o                            :   OUT   std_logic_vector(3
DOWNTO 0);  -- ufix4
        hun_o                            :   OUT   std_logic_vector(3
DOWNTO 0);  -- ufix4
        thou_o                           :   OUT   std_logic_vector(3
DOWNTO 0)  -- ufix4
        );
END Extract_Bits;


ARCHITECTURE rtl OF Extract_Bits IS

  -- Signals
  SIGNAL In_unsigned                      : unsigned(31 DOWNTO 0);
-- uint32
  SIGNAL uni_out1                        : unsigned(3 DOWNTO 0);  --
ufix4
  SIGNAL ten_out1                        : unsigned(3 DOWNTO 0);  --
ufix4
  SIGNAL hun_out1                        : unsigned(3 DOWNTO 0);  --
ufix4
  SIGNAL thou_out1                       : unsigned(3 DOWNTO 0);  -
- ufix4

BEGIN
  In_unsigned <= unsigned(In_rsvd);

  uni_out1 <= In_unsigned(10 DOWNTO 7);

  uni_o <= std_logic_vector(uni_out1);

  ten_out1 <= In_unsigned(14 DOWNTO 11);

  ten_o <= std_logic_vector(ten_out1);

  hun_out1 <= In_unsigned(18 DOWNTO 15);

  hun_o <= std_logic_vector(hun_out1);

  thou_out1 <= In_unsigned(22 DOWNTO 19);

  thou_o <= std_logic_vector(thou_out1);

END rtl;
```

APPENDIX 3. HEX4_uni.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY HEX4_uni IS
  PORT( BCD_In                          :   IN    std_logic_vector(3
DOWNTO 0);  -- ufix4
        HEX4                            :   OUT   std_logic_vector(0
TO 6)  -- boolean [7]
        );
END HEX4_uni;


ARCHITECTURE rtl OF HEX4_uni IS

  -- Signals
  SIGNAL BCD_In_unsigned                  : unsigned(3 DOWNTO 0);
-- ufix4
  SIGNAL Compare_To_Constant_out1         : std_logic;
  SIGNAL Compare_To_Constant3_out1        : std_logic;
  SIGNAL Compare_To_Constant5_out1        : std_logic;
  SIGNAL Compare_To_Constant4_out1        : std_logic;
  SIGNAL Compare_To_Constant_out1_1       : std_logic;
  SIGNAL Segment_0                        : std_logic;
  SIGNAL Compare_To_Constant4_out1_1      : std_logic;
  SIGNAL Segment_1                        : std_logic;
  SIGNAL Segment_2                        : std_logic;
  SIGNAL Compare_To_Constant6_out1        : std_logic;
  SIGNAL Compare_To_Constant9_out1        : std_logic;
  SIGNAL Compare_To_Constant_out1_2       : std_logic;
  SIGNAL Segment_3                        : std_logic;
  SIGNAL Compare_To_Constant2_out1        : std_logic;
  SIGNAL Compare_To_Constant_out1_3       : std_logic;
  SIGNAL Segment_4                        : std_logic;
  SIGNAL Compare_To_Constant_out1_4       : std_logic;
  SIGNAL Segment_5                        : std_logic;
  SIGNAL Compare_To_Constant7_out1        : std_logic;
  SIGNAL Compare_To_Constant7_out1_1      : std_logic;
  SIGNAL Segment_6                        : std_logic;
  SIGNAL Mux_out1                         : std_logic_vector(0 TO
6);  -- boolean [7]

BEGIN
  BCD_In_unsigned <= unsigned(BCD_In);


  Compare_To_Constant_out1 <= '1' WHEN BCD_In_unsigned = 1 ELSE
      '0';


  Compare_To_Constant3_out1 <= '1' WHEN BCD_In_unsigned = 4 ELSE
      '0';


  Compare_To_Constant5_out1 <= '1' WHEN BCD_In_unsigned = 6 ELSE
```

```
      '0';


  Compare_To_Constant4_out1 <= '1' WHEN BCD_In_unsigned = 5 ELSE
      '0';

  Compare_To_Constant_out1_1 <= Compare_To_Constant5_out1 OR
 (Compare_To_Constant_out1 OR Compare_To_Constant3_out1);

  Segment_0 <= Compare_To_Constant_out1_1;

  Compare_To_Constant4_out1_1 <= Compare_To_Constant4_out1 OR
Compare_To_Constant5_out1;

  Segment_1 <= Compare_To_Constant4_out1_1;


  Segment_2 <= '1' WHEN BCD_In_unsigned = 2 ELSE
      '0';


  Compare_To_Constant6_out1 <= '1' WHEN BCD_In_unsigned = 7 ELSE
      '0';


  Compare_To_Constant9_out1 <= '1' WHEN BCD_In_unsigned = 9 ELSE
      '0';

  Compare_To_Constant_out1_2 <= Compare_To_Constant9_out1 OR
 (Compare_To_Constant6_out1 OR (Compare_To_Constant_out1 OR
Compare_To_Constant3_out1));

  Segment_3 <= Compare_To_Constant_out1_2;


  Compare_To_Constant2_out1 <= '1' WHEN BCD_In_unsigned = 3 ELSE
      '0';

  Compare_To_Constant_out1_3 <= Compare_To_Constant9_out1 OR
 (Compare_To_Constant6_out1 OR (Compare_To_Constant4_out1 OR
 (Compare_To_Constant3_out1 OR (Compare_To_Constant_out1 OR
Compare_To_Constant2_out1))));

  Segment_4 <= Compare_To_Constant_out1_3;

  Compare_To_Constant_out1_4 <= Compare_To_Constant6_out1 OR
 (Compare_To_Constant2_out1 OR (Compare_To_Constant_out1 OR
Segment_2));

  Segment_5 <= Compare_To_Constant_out1_4;


  Compare_To_Constant7_out1 <= '1' WHEN BCD_In_unsigned = 0 ELSE
      '0';

  Compare_To_Constant7_out1_1 <= Compare_To_Constant6_out1 OR
 (Compare_To_Constant7_out1 OR Compare_To_Constant_out1);
```

```
  Segment_6 <= Compare_To_Constant7_out1_1;

  Mux_out1(0) <= Segment_0;
  Mux_out1(1) <= Segment_1;
  Mux_out1(2) <= Segment_2;
  Mux_out1(3) <= Segment_3;
  Mux_out1(4) <= Segment_4;
  Mux_out1(5) <= Segment_5;
  Mux_out1(6) <= Segment_6;

  HEX4 <= Mux_out1;

END rtl;
```

APPENDIX 4. Iteration1.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Iteration1 IS
  PORT( In_rsvd                           :   IN
std_logic_vector(31 DOWNTO 0);  -- uint32
        Out_rsvd                          :   OUT
std_logic_vector(31 DOWNTO 0)  -- uint32
        );
END Iteration1;


ARCHITECTURE rtl OF Iteration1 IS

  -- Signals
  SIGNAL In_unsigned                       : unsigned(31 DOWNTO 0);
-- uint32
  SIGNAL Bit_Slice1_out1                    : unsigned(3 DOWNTO 0);
-- ufix4
  SIGNAL switch_compare_1                   : std_logic;
  SIGNAL Constant3_out1                     : unsigned(31 DOWNTO 0);
-- uint32
  SIGNAL Sum2_out1                         : unsigned(31 DOWNTO 0);
-- uint32
  SIGNAL Switch2_out1                       : unsigned(31 DOWNTO 0);
-- uint32
  SIGNAL Bit_Slice2_out1                    : unsigned(3 DOWNTO 0);
-- ufix4
  SIGNAL switch_compare_1_1                  : std_logic;
  SIGNAL Constant2_out1                     : unsigned(31 DOWNTO 0);
-- uint32
  SIGNAL Sum1_out1                         : unsigned(31 DOWNTO 0);
-- uint32
  SIGNAL Switch1_out1                       : unsigned(31 DOWNTO 0);
-- uint32
  SIGNAL Bit_Slice_out1                     : unsigned(3 DOWNTO 0);  -
- ufix4
  SIGNAL switch_compare_1_2                  : std_logic;
  SIGNAL Constant1_out1                     : unsigned(31 DOWNTO 0);
-- uint32
  SIGNAL Sum_out1                          : unsigned(31 DOWNTO 0);  -
- uint32
  SIGNAL Switch_out1                        : unsigned(31 DOWNTO 0);
-- uint32
  SIGNAL Bit_Shift_out1                     : unsigned(31 DOWNTO 0);
-- uint32

BEGIN
  In_unsigned <= unsigned(In_rsvd);

  Bit_Slice1_out1 <= In_unsigned(18 DOWNTO 15);


  switch_compare_1 <= '1' WHEN Bit_Slice1_out1 >= 5 ELSE
```

```
      '0';

  Constant3_out1 <= to_unsigned(16#00018000#, 32);

  Sum2_out1 <= In_unsigned + Constant3_out1;


  Switch2_out1 <= In_unsigned WHEN switch_compare_1 = '0' ELSE
      Sum2_out1;

  Bit_Slice2_out1 <= Switch2_out1(14 DOWNTO 11);


  switch_compare_1_1 <= '1' WHEN Bit_Slice2_out1 >= 5 ELSE
      '0';

  Constant2_out1 <= to_unsigned(16#00001800#, 32);

  Sum1_out1 <= Switch2_out1 + Constant2_out1;


  Switch1_out1 <= Switch2_out1 WHEN switch_compare_1_1 = '0'
ELSE
      Sum1_out1;

  Bit_Slice_out1 <= Switch1_out1(10 DOWNTO 7);


  switch_compare_1_2 <= '1' WHEN Bit_Slice_out1 >= 5 ELSE
      '0';

  Constant1_out1 <= to_unsigned(16#00000180#, 32);

  Sum_out1 <= Switch1_out1 + Constant1_out1;


  Switch_out1 <= Switch1_out1 WHEN switch_compare_1_2 = '0' ELSE
      Sum_out1;

  Bit_Shift_out1 <= Switch_out1 sll 1;

  Out_rsvd <= std_logic_vector(Bit_Shift_out1);

END rtl;
```