FOC SDSoC

1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

Introduction

An implementation of a Field-Oriented Motor Control (FOC), developed in the Xilinx SDSoC Development Environment, is described. It is implemented as a C function in a SDSoC Platform sample program. To save FPGA resources, this implementation uses fixed-point numbers instead of floating-point numbers.

1.1 Block diagram

The block diagram of the FOC is shown on Fig. 1.1.

The FOC is controlled by arguments register block. The status register block reports encoder angle and rotation speed of the motor. The registers are accessible on an AXI Bus.

The hardware platform is interfaced to through two AXI Streams. The output AXI Stream contains three values, one for each phase of the three-phase inverter. The input AXI Stream should contain current sensor data from the first two phases, the encoder angle and the motor speed.

The FOC can operate in different modes as determined by the the function selector, which connects inputs to outputs according to the contents of the the register CONTROL_REG in the arguments register block. See CONTROL_REG for the list of functions available.

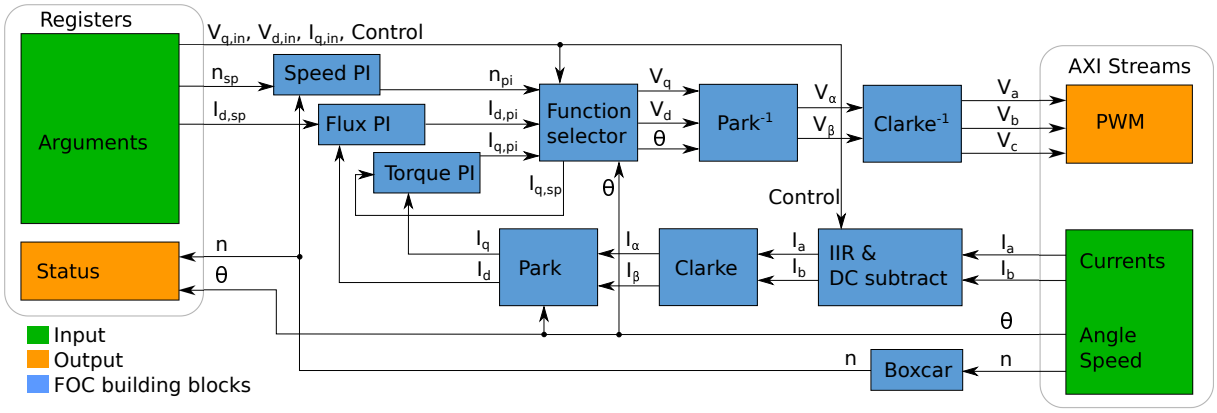

Figure 1.1 Block Diagram of the FOC.

Interface

The FOC is implemented in a form of a C function, that is to be called from a SDSoC project main function as follows:

```
foc(inputStream, outputStream, args, status);
```

The C header file foc.h contains all the necessary definitions for using the function foc(). See also the file main.cpp for the template C main function, which defines the SDSoC application.

Tools

For the documentation Doxygen is used; the C source includes Doxygen-formatted comments.

Tool	Version	Notes
Vivado SDSoC	2017.1	SDSoC Development Environment
Doxygen	1.8.11	Documentation extraction
MiKTeX	2.9	PDF generation

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Module Documentation

4.1 User-configurable macros

These macros should be configured by the user to match particular hardware setup.

Macros

- #define CPR 1000

  Number of encoder steps per one full revolution.
- #define PPR 2

  Number of pole pairs per phase of the motor; full sinus periods per revolution.
- #define DC_ACC_BITS 15

  Number of extra bits in the DC accumulators.
- #define FILTER_ORDER 5

  Order of the RPM boxcar filter.

4.1.1 Detailed Description

These macros should be configured by the user to match particular hardware setup.

Most important is to have CPR and PPR correctly set.

4.1.2 Macro Definition Documentation

4.1.2.1 CPR

```
#define CPR 1000
```

Number of encoder steps per one full revolution.

Important: Change the sine and cosine tables in the file sin_cos_table.h accordingly when changing this.

Definition at line 24 of file foc.h.

4.1.2.2 DC_ACC_BITS

```
#define DC_ACC_BITS 15
```

Number of extra bits in the DC accumulators.

The number of samples is 2 to the power of extra bits.

The DC level is determined when the motor is stopped (CONTROL_REG = 0) and the correction is applied when the motor is energized (CONTROL_REG != 0).

Definition at line 34 of file foc.h.

4.1.2.3 FILTER_ORDER

```
#define FILTER_ORDER 5
```

Order of the RPM boxcar filter.

Filter length is 2 to the power of filter order.

Definition at line 38 of file foc.h.

4.1.2.4 PPR

```
#define PPR 2
```

Number of pole pairs per phase of the motor; full sinus periods per revolution.

Definition at line 27 of file foc.h.

4.2 Arguments register block

Indices of the registers in the argument register block.

Macros

- #define ARGS_SIZE 16

  Number of argument registers of the FOC.
- #define CONTROL_REG 0

  Control register.
- #define FLUX_SP_REG 1

  Flux setpoint.
- #define FLUX_KP_REG 2

  Flux PI loop proportional factor.
- #define FLUX_KI_REG 3

  Flux PI loop integral factor.
- #define TORQUE_SP_REG 4

  Torque setpoint.
- #define TORQUE_KP_REG 5

  Torque PI loop proportional factor.
- #define TORQUE_KI_REG 6

  Torque PI loop integral factor.
- #define RPM_SP_REG 7

  Speed setpoint, in RPM.
- #define RPM_KP_REG 8

  Speed PI loop proportional factor.
- #define RPM_KI_REG 9

  Speed PI loop integral factor.
- #define ANGLE_SH_REG 10

  Angle shift, in the units of encoder steps.
- #define VD_REG 11

  Fixed Vd.
- #define VQ_REG 12

  Fixed Vq.
- #define FA_REG 13

  Filter coefficient A.
- #define FB_REG 14

  Filter coefficient B.
- #define TRIGGER_REG 14

  Trigger data capture.
- #define CONTROL2_REG 15

  Second control register.
- #define CONTROL_BIT_MODE 0

  Start of the mode bits in register CONTROL_REG.
- #define CONTROL_MAX_MODE 0x0Fu

  Maximum value for the mode bits in CONTROL_REG.
- #define CONTROL_BIT_FIXPERIOD 4

  Start of the fixed speed delay in the register CONTROL_REG.
- #define CONTROL_MAX_FIXPERIOD 0xFFFu

Maximum value for the fixed speed delay in the register CONTROL_REG.

- #define MODE_STOPPED 0u

  Motor stopped.
- #define MODE_SPEED 1u

  Speed control mode.
- #define MODE_MANUAL_TORQUE_FLUX 2u

  Manual torque and flux.
- #define MODE_MANUAL_TORQUE 3u

  Manual torque.
- #define MODE_SPEED_WITHOUT_TORQUE 4u

  Speed control without torque PI.
- #define MODE_TORQUE_WITHOUT_SPEED 5u

  Torque control without speed PI.
- #define MODE_MANUAL_TORQUE_FLUX_FIXED_SPEED 6u

  Manual torque and flux, fixed speed rotation.
- #define MODE_FIXED_POSITION 7u

  Motor position fixed; only phase C is powered.
- #define DATASOURCE_ADC 0

  ADC data.
- #define DATASOURCE_I_ALPHA_BETA 1

  Output of the Clarke transform, the values I_α and I_β.
- #define DATASOURCE_I_D_Q 2

  Output of the Park transform, the values I_d and I_q.
- #define DATASOURCE_V_D_Q 3

  Input to the inverse Park transform, the values V_d and V_q.
- #define DATASOURCE_V_ALPHA_BETA 4

  Output of the inverse Park transform, the values V_α and V_β.
- #define DATASOURCE_V_A_B_C 5

  Output of the inverse Clarke transform, the values V_a, V_b and V_c.
- #define DATASOURCE_PWM 6

  Direct PWM values.
- #define CONTROL2_BIT_ERROR_LIMIT 4u

  The bit position of the error limit in the register CONTROL2_REG.
- #define CONTROL2_BV_ERROR_LIMIT (0xFFFFu << CONTROL2_BIT_ERROR_LIMIT)

  Bitmask of the error limit in the register CONTROL2_REG.
- #define CONTROL2_BV_LED (1u << 20)

  Bit value of the user LED bit in the register CONTROL2_REG.
- #define CONTROL2_BV_RESET_ERROR (1u << 21)

  Bit value of the reset error bit in the register CONTROL2_REG.
- #define CONTROL2_BV_SPREAD_SPECTRUM (1u << 22)

  Bit value of the spread spectrum enable bit in the register CONTROL2_REG.
- #define CONTROL2_BIT_DECIMATION 24

  The bit position of the decimation in the register CONTROL2_REG.
- #define CONTROL2_MAX_DECIMATION 0xFFu

  Maximum value for the decimation factor.
- #define CONTROL2_BITMASK_DECIMATION (CONTROL2_MAX_DECIMATION << CONTROL2_BIT_↩ DECIMATION)

  The bitmask of the decimation value in the register CONTROL2_REG.

4.2.1 Detailed Description

Indices of the registers in the argument register block.

The size of the argument register block is determined by ARGS_SIZE.

4.2.2 Macro Definition Documentation

4.2.2.1 ANGLE_SH_REG

```
#define ANGLE_SH_REG 10
```

Angle shift, in the units of encoder steps.

Register number in the arguments register block.

Definition at line 122 of file foc.h.

4.2.2.2 ARGS_SIZE

```
#define ARGS_SIZE 16
```

Number of argument registers of the FOC.

Definition at line 48 of file foc.h.

4.2.2.3 CONTROL2_BIT_DECIMATION

```
#define CONTROL2_BIT_DECIMATION 24
```

The bit position of the decimation in the register CONTROL2_REG.

Definition at line 254 of file foc.h.

4.2.2.4 CONTROL2_BIT_ERROR_LIMIT

```
#define CONTROL2_BIT_ERROR_LIMIT 4u
```

The bit position of the error limit in the register CONTROL2_REG.

Definition at line 239 of file foc.h.

4.2.2.5 CONTROL2_BITMASK_DECIMATION

```
#define CONTROL2_BITMASK_DECIMATION (CONTROL2_MAX_DECIMATION << CONTROL2_BIT_DECIMATION)
```

The bitmask of the decimation value in the register CONTROL2_REG.

Definition at line 260 of file foc.h.

4.2.2.6 CONTROL2_BV_ERROR_LIMIT

```
#define CONTROL2_BV_ERROR_LIMIT (0xFFFFu << CONTROL2_BIT_ERROR_LIMIT)
```

Bitmask of the error limit in the register CONTROL2_REG.

Definition at line 242 of file foc.h.

4.2.2.7 CONTROL2_BV_LED

```
#define CONTROL2_BV_LED (1u << 20)
```

Bit value of the user LED bit in the register CONTROL2_REG.

Definition at line 245 of file foc.h.

4.2.2.8 CONTROL2_BV_RESET_ERROR

```
#define CONTROL2_BV_RESET_ERROR (1u << 21)
```

Bit value of the reset error bit in the register CONTROL2_REG.

Definition at line 248 of file foc.h.

4.2.2.9 CONTROL2_BV_SPREAD_SPECTRUM

```
#define CONTROL2_BV_SPREAD_SPECTRUM (1u << 22)
```

Bit value of the spread spectrum enable bit in the register CONTROL2_REG.

Definition at line 251 of file foc.h.

4.2.2.10 CONTROL2_MAX_DECIMATION

```
#define CONTROL2_MAX_DECIMATION 0xFFu
```

Maximum value for the decimation factor.

Definition at line 257 of file foc.h.

4.2.2.11 CONTROL2_REG

```
#define CONTROL2_REG 15
```

Second control register.

Register number in the arguments register block.

Layout of the second control register:

- Bits 0 .. 3: Source for the data capture, one of DATASOURCE_ADC, DATASOURCE_I_ALPHA_BETA, DATASOURCE_I_D_Q, DATASOURCE_I_D_Q, DATASOURCE_V_D_Q, DATASOURCE_V_ALPHA_BETA, DATASOURCE_V_A_B_C, DATASOURCE_PWM.
- Bits 4 .. 19: Error limit for the speed check, unsigned integer.,
- Bit 20: User led `LD3` on the ARTY Z7 board., see CONTROL2_BV_LED,
- Bit 21: Set to 1 to reset the built-in speed check; don't forget to reset it back to 0, see CONTROL2_BV_RESET_ERROR,
- Bit 22: Set to 1 to enable spread-spectrum mode, see CONTROL2_BV_SPREAD_SPECTRUM,
- Bits 23 .. 31: Unused, must be zero.

Important: This is valid in the Vivado HLS FOC project only.

Definition at line 170 of file foc.h.

4.2.2.12 CONTROL_BIT_FIXPERIOD

```
#define CONTROL_BIT_FIXPERIOD 4
```

Start of the fixed speed delay in the register CONTROL_REG.

Definition at line 179 of file foc.h.

4.2.2.13 CONTROL_BIT_MODE

```
#define CONTROL_BIT_MODE 0
```

Start of the mode bits in register CONTROL_REG.

Definition at line 173 of file foc.h.

4.2.2.14 CONTROL_MAX_FIXPERIOD

```
#define CONTROL_MAX_FIXPERIOD 0xFFFu
```

Maximum value for the fixed speed delay in the register CONTROL_REG.

Definition at line 182 of file foc.h.

4.2.2.15 CONTROL_MAX_MODE

```
#define CONTROL_MAX_MODE 0x0Fu
```

Maximum value for the mode bits in CONTROL_REG.

Definition at line 176 of file foc.h.

4.2.2.16 CONTROL_REG

```
#define CONTROL_REG 0
```

Control register.

Register number in the arguments register block.

Layout of the control register:

- Bits 0 .. 3: Mode of the FOC operation, one of MODE_STOPPED, MODE_SPEED, MODE_MANUAL_TO↩ RQUE_FLUX, MODE_MANUAL_TORQUE, MODE_SPEED_WITHOUT_TORQUE, MODE_TORQUE_WI↩ THOUT_SPEED, MODE_MANUAL_TORQUE_FLUX_FIXED_SPEED, MODE_FIXED_POSITION.
- Bits 4 .. 15: Delay in the fixed speed mode MODE_MANUAL_TORQUE_FLUX_FIXED_SPEED. The bigger the delay, the slower the motor rotates.

The mode selects the function of the FOC according to the following table:

Mode	V_q	V_d	Θ	$I_{q,sp}$	Description
0	0	0	0	$I_{q,in}$	Motor stopped. DC offset of the current
1	$I_{q,pi}$	$I_{d,pi}$	Θ_{in}	n_{pi}	Speed control
2	$V_{q,in}$	$V_{d,in}$	Θ_{in}	$I_{q,in}$	Manual torque and flux
3	$V_{q,in}$	$I_{d,pi}$	Θ_{in}	$I_{q,in}$	Manual torque
4	n_{pi}	$I_{d,pi}$	Θ_{in}	$I_{q,in}$	Speed control without torque control
5	$I_{q,pi}$	$I_{d,pi}$	Θ_{in}	$I_{q,in}$	Torque control without speed control
6	$V_{q,in}$	$V_{d,in}$	Internal counter	$I_{q,in}$	Manual torque and flux at low fixed speed
7	N/A	N/A	N/A	N/A	Motor position fixed; only phase C is powered

See the chapter Block diagram for the block diagram.

Definition at line 70 of file foc.h.

4.2.2.17 DATASOURCE_ADC

```
#define DATASOURCE_ADC 0
```

ADC data.

Definition at line 218 of file foc.h.

4.2.2.18 DATASOURCE_I_ALPHA_BETA

```
#define DATASOURCE_I_ALPHA_BETA 1
```

Output of the Clarke transform, the values I_α and I_β.

Definition at line 221 of file foc.h.

4.2.2.19 DATASOURCE_I_D_Q

```
#define DATASOURCE_I_D_Q 2
```

Output of the Park transform, the values I_d and I_q.

Definition at line 224 of file foc.h.

4.2.2.20 DATASOURCE_PWM

```
#define DATASOURCE_PWM 6
```

Direct PWM values.

Definition at line 236 of file foc.h.

4.2.2.21 DATASOURCE_V_A_B_C

```
#define DATASOURCE_V_A_B_C 5
```

Output of the inverse Clarke transform, the values V_a, V_b and V_c.

Definition at line 233 of file foc.h.

4.2.2.22 DATASOURCE_V_ALPHA_BETA

```
#define DATASOURCE_V_ALPHA_BETA 4
```

Output of the inverse Park transform, the values V_α and V_β.

Definition at line 230 of file foc.h.

4.2.2.23 DATASOURCE_V_D_Q

```
#define DATASOURCE_V_D_Q 3
```

Input to the inverse Park transform, the values V_d and V_q.

Definition at line 227 of file foc.h.

4.2.2.24 FA_REG

```
#define FA_REG 13
```

Filter coefficient A.

Register number in the arguments register block.

Format: Q16.16.

Important: This is valid in the Vivado SDSoC FOC project only.

Definition at line 142 of file foc.h.

4.2.2.25 FB_REG

```
#define FB_REG 14
```

Filter coefficient B.

Register number in the arguments register block.

Format: Q16.16.

Important: This is valid in the Vivado SDSoC FOC project only.

Definition at line 150 of file foc.h.

4.2.2.26 FLUX_KI_REG

```
#define FLUX_KI_REG 3
```

Flux PI loop integral factor.

Register number in the arguments register block.

Format: Q16.16.

Definition at line 86 of file foc.h.

4.2.2.27 FLUX_KP_REG

```
#define FLUX_KP_REG 2
```

Flux PI loop proportional factor.

Register number in the arguments register block. Format: Q16.16.

Definition at line 80 of file foc.h.

4.2.2.28 FLUX_SP_REG

```
#define FLUX_SP_REG 1
```

Flux setpoint.

Register number in the arguments register block. Unit: Resolution of the current ADC-s.

Definition at line 75 of file foc.h.

4.2.2.29 MODE_FIXED_POSITION

```
#define MODE_FIXED_POSITION 7u
```

Motor position fixed; only phase C is powered.

Value of the register CONTROL_REG.

Definition at line 214 of file foc.h.

4.2.2.30 MODE_MANUAL_TORQUE

```
#define MODE_MANUAL_TORQUE 3u
```

Manual torque.

Value of the register CONTROL_REG.

Definition at line 198 of file foc.h.

4.2.2.31 MODE_MANUAL_TORQUE_FLUX

```
#define MODE_MANUAL_TORQUE_FLUX 2u
```

Manual torque and flux.

Value of the register CONTROL_REG.

Definition at line 194 of file foc.h.

4.2.2.32 MODE_MANUAL_TORQUE_FLUX_FIXED_SPEED

```
#define MODE_MANUAL_TORQUE_FLUX_FIXED_SPEED 6u
```

Manual torque and flux, fixed speed rotation.

Value of the register CONTROL_REG.

Definition at line 210 of file foc.h.

4.2.2.33 MODE_SPEED

```
#define MODE_SPEED 1u
```

Speed control mode.

Value of the register CONTROL_REG.

Definition at line 190 of file foc.h.

4.2.2.34 MODE_SPEED_WITHOUT_TORQUE

```
#define MODE_SPEED_WITHOUT_TORQUE 4u
```

Speed control without torque PI.

Value of the register CONTROL_REG.

Definition at line 202 of file foc.h.

4.2.2.35 MODE_STOPPED

```
#define MODE_STOPPED 0u
```

Motor stopped.

Valu of the register CONTROL_REG.

Definition at line 186 of file foc.h.

4.2.2.36 MODE_TORQUE_WITHOUT_SPEED

```
#define MODE_TORQUE_WITHOUT_SPEED 5u
```

Torque control without speed PI.

Value of the register CONTROL_REG.

Definition at line 206 of file foc.h.

4.2.2.37 RPM_KI_REG

```
#define RPM_KI_REG 9
```

Speed PI loop integral factor.

Register number in the arguments register block.

Definition at line 118 of file foc.h.

4.2.2.38 RPM_KP_REG

```
#define RPM_KP_REG 8
```

Speed PI loop proportional factor.

Register number in the arguments register block.

Format: Q16.16.

Definition at line 114 of file foc.h.

4.2.2.39 RPM_SP_REG

```
#define RPM_SP_REG 7
```

Speed setpoint, in RPM.

Register number in the arguments register block.

Definition at line 108 of file foc.h.

4.2.2.40 TORQUE_KI_REG

```
#define TORQUE_KI_REG 6
```

Torque PI loop integral factor.

Register number in the arguments register block.

Format: Q16.16.

Definition at line 104 of file foc.h.

4.2.2.41 TORQUE_KP_REG

```
#define TORQUE_KP_REG 5
```

Torque PI loop proportional factor.

This is the index of the torque PI loop proportional factor register in the argument register block.

Format: Q16.16.

Definition at line 98 of file foc.h.

4.2.2.42 TORQUE_SP_REG

```
#define TORQUE_SP_REG 4
```

Torque setpoint.

Register number in the arguments register block.

Unit: Resolution of the current ADC-s.

Definition at line 92 of file foc.h.

4.2.2.43 TRIGGER_REG

```
#define TRIGGER_REG 14
```

Trigger data capture.

Register number in the arguments register block.

Important: This is valid in the Vivado HLS FOC project only.

Definition at line 156 of file foc.h.

4.2.2.44 VD_REG

```
#define VD_REG 11
```

Fixed Vd.

Register number in the arguments register block.

Unit: Resolution of the PWM.

Definition at line 128 of file foc.h.

4.2.2.45 VQ_REG

```
#define VQ_REG 12
```

Fixed Vq.

Register number in the arguments register block.

Unit: Resolution of the PWM.

Definition at line 134 of file foc.h.

4.3 Status register block

Indices of the registers in the status register block.

Macros

- #define STATUS_SIZE 4
  Number of status registers of the FOC.
- #define ANGLE_REG 0
  Encoder angle, in encoder steps.
- #define RPM_REG 1
  Speed of rotation, in RPM.
- #define ID_REG 2
  Flux.
- #define IQ_REG 3
  Stator current.

4.3.1 Detailed Description

Indices of the registers in the status register block.

Size of the status register block is determined by STATUS_SIZE.

4.3.2 Macro Definition Documentation

4.3.2.1 ANGLE_REG

```
#define ANGLE_REG 0
```

Encoder angle, in encoder steps.

Register number in the status register block.

Definition at line 276 of file foc.h.

4.3.2.2 ID_REG

```
#define ID_REG 2
```

Flux.

Register number in the status register block.

Definition at line 284 of file foc.h.

4.3.2.3 IQ_REG

```
#define IQ_REG 3
```

Stator current.

Register number in the status register block.

Definition at line 288 of file foc.h.

4.3.2.4 RPM_REG

```
#define RPM_REG 1
```

Speed of rotation, in RPM.

Register number in the status register block.

Definition at line 280 of file foc.h.

4.3.2.5 STATUS_SIZE

```
#define STATUS_SIZE 4
```

Number of status registers of the FOC.

Definition at line 272 of file foc.h.

4.4 Mathematical constants

Mathematical constants used by the FOC.

Macros

- #define MAX_LIM 32767
  Maximum positive value for saturated arithmetic.
- #define MIN_LIM -32767
  Minimum negative value for saturated arithmetic.
- #define SQRT3A 0x000093CD
  The number $\frac{1}{\sqrt{3}}$ in the Q16.16 format.
- #define SQRT3C 0x0000DDB4
  The number $\sqrt{3}$ in the Q16.16 format.

4.4.1 Detailed Description

Mathematical constants used by the FOC.

4.4.2 Macro Definition Documentation

4.4.2.1 MAX_LIM

```
#define MAX_LIM 32767
```

Maximum positive value for saturated arithmetic.

Definition at line 297 of file foc.h.

4.4.2.2 MIN_LIM

```
#define MIN_LIM -32767
```

Minimum negative value for saturated arithmetic.

Definition at line 300 of file foc.h.

4.4.2.3 SQRT3A

```
#define SQRT3A 0x000093CD
```

The number $\frac{1}{\sqrt{3}}$ in the Q16.16 format.

Definition at line 303 of file foc.h.

4.4.2.4 SQRT3C

```
#define SQRT3C 0x0000DDB4
```

The number $\sqrt{3}$ in the Q16.16 format.

Definition at line 306 of file foc.h.

4.5 FOC function

Function implementing FOC.

Functions

- void foc (long long int ∗A, long long int ∗B, long long int ∗C, int args[ARGS_SIZE], int status[STATUS_SIZE])
  Implementation of the Field-Oriented Control in the FPGA.

4.5.1 Detailed Description

Function implementing FOC.

4.5.2 Function Documentation

4.5.2.1 foc()

```
void foc (
              long long int * A,
              long long int * B,
              long long int * C,
              int args[ARGS_SIZE],
              int status[STATUS_SIZE] )
```

Implementation of the Field-Oriented Control in the FPGA.

See the chapter Block diagram for the functional overview.

Parameters

A	Input stream. The data is structured as follows: • Bits 0 .. 15: Current on phase A, • Bits 16 .. 31: Current on phase B, • Bits 32 .. 47: RPM, • Bits 48 .. 63: Angle.
B	Output stream to the PWM block. The data is structured as follows: • Bits 0 .. 15: PWM on phase A, • Bits 16 .. 31: PWM on phase B, • Bits 32 .. 47: PWM on phase C, • Bits 48 .. 63: Reserved, always 0.
C	Output stream to the data capture IP core.
args	Control register block, size of ARGS_SIZE.
status	Status register block, size of STATUS_SIZE.

Definition at line 115 of file foc.cpp.

```
116 {
117 #pragma HLS pipeline enable_flush
118 #pragma HLS interface axis port=A
119 #pragma HLS interface axis port=B
120 #pragma HLS interface axis port=C
121 #pragma HLS interface s_axilite port=args offset=64 bundle=args
122 #pragma HLS interface s_axilite port=status offset=128 bundle=args
123 #pragma HLS interface ap_ctrl_none port=return
124     //------------------------------------------------------------------------
125     // Decode Input stream
126     //------------------------------------------------------------------------
127     short Ia;                                       // Phase A current
128     short Ib;                                       // Phase B current
129     short RPM;                                      // RPM
130     unsigned short Angle;                           // Encoder count
131     int Angle_shift;                                // Angle shift between encoder index and motor rotor
    phase A
132     int Theta;                                      // Rotor Angle
133     int Control_Reg;                                // Copy of the control register
134     static uint4_t Mode_Prev = MODE_STOPPED; // Previous Control Register.
135     bool Mode_Change;
136     long long int   A_copy;
137     uint4_t Mode;
138     uint12_t FixPeriod;
139 
140     A_copy = *A;
141     Ia     = (A_copy & 0x000000000000FFFF);              // Extract phase A current from input stream
142     Ib     = (A_copy & 0x00000000FFFF0000) >> 16;        // Extract phase B current from input stream
143     RPM    = (A_copy & 0x0000FFFF00000000) >> 32;        // Extract RPM from input stream
144     Angle = (A_copy & 0xFFFF000000000000) >> 48;        // Extract encoder count from input stream
145     Angle_shift = args[ANGLE_SH_REG];               // Read angle shift parameter
146     Theta = (int)Angle - Angle_shift;               // Apply angle correction
147     Theta = (Theta < 0) ? (Theta + CPR) : Theta; // Correct negative angle
148     Theta = (Theta >= CPR) ? (Theta - CPR) : Theta;   // Correct angle overload
149     status[ANGLE_REG] = Theta;                     // Pass current Angle to Status
150     Control_Reg = args[CONTROL_REG];
151     Mode = (uint32_t)(Control_Reg >> CONTROL_BIT_MODE) &
    CONTROL_MAX_MODE;
152     FixPeriod = (uint32_t)(Control_Reg >> CONTROL_BIT_FIXPERIOD) &
    CONTROL_MAX_FIXPERIOD;
153     Mode_Change = Mode != Mode_Prev;
154 
155     //------------------------------------------------------------------------
156     // Input filters
157     //------------------------------------------------------------------------
158     // RPM Filtering (simple average filter)
159     int RPM_filtered;                               // Output of the RPM filter
160     int RPM_filtered_negative;                      // Output of the RPM filter
161     static int rpm_filt_acc;                        // Accumulator for the RPM filter
162     static int rpm_filt_mem[FILTER_LENGTH];            // Memory for the RPM filter
163     int fi;                                         // Loop index
164     int rpm_filt_res;                               // Result of the RPM filter
165 
166     rpm_filt_acc -= rpm_filt_mem[FILTER_LENGTH-1]; // Remove tail value from the accumulator
167     rpm_filt_acc += RPM;                            // Add new value to the accumulator
168     for(fi = (FILTER_LENGTH-1); fi > 0; fi--){     // Loop to shift rpm filter memory to the
    right
169         rpm_filt_mem[fi] = rpm_filt_mem[fi - 1];
170     }
171     rpm_filt_mem[0] = RPM;                          // Add head value to memory
172     rpm_filt_res = rpm_filt_acc >> FILTER_ORDER;    // Divide accumulator to memory size
173 
174     unsigned short filt_a;                          // Filter coefficient A
175     unsigned short filt_b;                          // Filter coefficient B
176     filt_a = (unsigned short)args[FA_REG];
177     filt_b = (unsigned short)args[FB_REG];
178 
179     // First IIR filter stage
180     int Ia_filtered;                                // Result of filter stage A
181     int Ib_filtered;                                // Result of filter stage B
182     int Y1a, Y1b;                                   // Partial result
183     static int Y1a_prev;                            // Variable to store previous value
184     static int Y1b_prev;                            // Variable to store previous value
185     int Ia1_filtered;                               // First stage result
186     int Ib1_filtered;                               // First stage result
187 
188     Y1a = Ia*filt_a + Y1a_prev*filt_b;              // Y(i) = Ia*A + Y(i-1)*B
189     Y1b = Ib*filt_a + Y1b_prev*filt_b;              // Y(i) = Ia*A + Y(i-1)*B
190     Ia1_filtered    = Y1a >> 15;                    // Remove fractional part
191     Ib1_filtered    = Y1b >> 15;                    // Remove fractional part
192     Y1a_prev = Ia1_filtered;                        // Store result for next round
193     Y1b_prev = Ib1_filtered;                        // Store result for next round
194 
195     // Second filter stage
```

```
196     int Y2a, Y2b;                                   // Partial result
197     static int Y2a_prev;                            // Variable to store previous value
198     static int Y2b_prev;                            // Variable to store previous value
199     int Ia2_filtered;                               // Second stage result
200     int Ib2_filtered;                               // Second stage result
201
202     Y2a = Ia1_filtered*filt_a + Y2a_prev*filt_b;    // Y(i) = Ia*A + Y(i-1)*B
203     Y2b = Ib1_filtered*filt_a + Y2b_prev*filt_b;    // Y(i) = Ia*A + Y(i-1)*B
204     Ia2_filtered    = Y2a >> 15;                    // Remove fractional part
205     Ib2_filtered    = Y2b >> 15;                    // Remove fractional part
206     Y2a_prev = Ia2_filtered;                        // Store result for next round
207     Y2b_prev = Ib2_filtered;                        // Store result for next round
208
209     Ia_filtered = Ia2_filtered;                     // Pass data to next step
210     Ib_filtered = Ib2_filtered;                     // Pass data to next step
211     RPM_filtered = rpm_filt_res;                    // Pass data to next step
212     RPM_filtered_negative = -RPM_filtered;
213     status[RPM_REG] = RPM_filtered_negative;                // Pass filtered RPM to Status
214
215     //-------------------------------------------------------------------------
216     // Calculate DC
217     //-------------------------------------------------------------------------
218     static int Ia_DC_acc = 0;                       // Ia Accumulator
219     static int Ib_DC_acc = 0;                       // Ib Accumulator
220     static int Ia_DC_val = 0;                       // Ia DC
221     static int Ib_DC_val = 0;                       // Ib DC
222     static int dc_cnt = 0;                          // Counter
223     static int Ia_corr;                             // Correction for phase A current
224     static int Ib_corr;                             // Correction for phase B current
225
226     if(dc_cnt >= (DC_ACC_SAMPLES-1)){             // End of accumulation
227         Ia_DC_val   = Ia_DC_acc >> DC_ACC_BITS;      // Divide
228         Ib_DC_val   = Ib_DC_acc >> DC_ACC_BITS;      // Divide
229         Ia_DC_acc   = Ia_filtered;                  // Reset Accumulation and load new value
230         Ib_DC_acc   = Ib_filtered;                  // Reset Accumulation and load new value
231         dc_cnt      = 0;
232     }
233     else{                                           // Accumulation
234         Ia_DC_acc   = Ia_DC_acc + Ia_filtered;      // Accumulate Ia
235         Ib_DC_acc   = Ib_DC_acc + Ib_filtered;      // Accumulate Ib
236         dc_cnt++;
237     }
238     if (Mode == MODE_STOPPED) {         // Save DC in Idle mode
239         Ia_corr = Ia_DC_val;                        // Save Ia correction
240         Ib_corr = Ib_DC_val;                        // Save Ib correction
241     }
242
243     //-------------------------------------------------------------------------
244     // Apply DC correction
245     //-------------------------------------------------------------------------
246     int Ia_AC;                                      // Corrected phase A current
247     int Ib_AC;                                      // Corrected phase B current
248
249     Ia_AC = Ia2_filtered - Ia_corr;                 // Apply Ia correction
250     Ib_AC = Ib2_filtered - Ib_corr;                 // Apply Ib correction
251
252     //-------------------------------------------------------------------------
253     // Clarke Direct
254     // Ialpha = Ia
255     // Ibeta = (Ia + 2Ib)/sqrt(3)
256     // Where Ia+Ib+Ic = 0
257     //-------------------------------------------------------------------------
258     int Ibd;                                        // Partial result
259     int Ialpha, Ibeta;                              // Transfom result
260
261     Ialpha = (int)Ia_AC;                            // Type conversion
262     Ibd = (int)Ia_AC + ((int)Ib_AC << 1);           // calculate Ia+2*Ib
263     Ibeta = Clip32((Ibd * SQRT3A) >> 16, MIN_LIM, MAX_LIM);                 // *
    1/SQRT(3)
264
265     //-------------------------------------------------------------------------
266     // Park Direct
267     // Id = Ialpha*cos(Theta) + Ibeta*sin(Theta)
268     // Iq = Ibeta*cos(Theta) - Ialpha*sin(Theta)
269     //-------------------------------------------------------------------------
270     int cos_theta, sin_theta;                       // Common Park Direct/Inverse
271     int Ia_cos, Ib_sin, Ib_cos, Ia_sin;             // Park Direct variables
272     int Id, Iq;                                     // Park Direct -> PI
273
274     cos_theta = (int)cos_table[Theta];             // Read con(theta)
275     sin_theta = (int)sin_table[Theta];             // Read sin(theta)
276     Ia_cos = (int)Ialpha * cos_theta;               // Ialpha*con(theta)
277     Ib_sin = (int)Ibeta * sin_theta;                // Ibeta*sin(theta)
278     Ib_cos = (int)Ibeta * cos_theta;                // Ibeta*cos(theta)
279     Ia_sin = (int)Ialpha * sin_theta;               // Ialpha*sin(theta)
280     Id = Clip32((Ia_cos + Ib_sin) >> 15, MIN_LIM, MAX_LIM);       // Remove fractional part
281     Iq = Clip32((Ib_cos - Ia_sin) >> 15, MIN_LIM, MAX_LIM);       // Remove fractional part
```

```
282     status[ID_REG] = ~Id;
283     status[IQ_REG] = ~Iq;
284 
285     //-------------------------------------------------------------------------
286     // RPM PI Controller
287     //-------------------------------------------------------------------------
288     static int32_t RPM_GiE_prev = 0;                    // Variable for previous value
289     int16_t RPM_Out;                                    // Partial results
290     RPM_Out = PI_Control(RPM_filtered, -args[RPM_SP_REG], args[
    RPM_KP_REG], args[RPM_KI_REG], Mode_Change, RPM_GiE_prev);
291 
292     //-------------------------------------------------------------------------
293     // Flux PI Controller
294     //-------------------------------------------------------------------------
295     static int32_t Flux_GiE_prev = 0;                   // Variable for previous value
296     int16_t Flux_Out;                                   // Partial results
297     Flux_Out = PI_Control(Id, -args[FLUX_SP_REG], args[FLUX_KP_REG], args[
    FLUX_KI_REG], Mode_Change, Flux_GiE_prev);
298 
299     //-------------------------------------------------------------------------
300     // Torque PI Controller
301     //-------------------------------------------------------------------------
302     static int32_t Torque_GiE_prev = 0;                 // Variable for previous value
303     int16_t Torque_Sp;                                  // PI parameters
304     int16_t Torque_Out;                                 // Partial results
305     Torque_Sp = (Mode == MODE_SPEED) ? RPM_Out : (-args[TORQUE_SP_REG]);
306     Torque_Out = PI_Control(Iq, Torque_Sp, args[TORQUE_KP_REG], args[
    TORQUE_KI_REG], Mode_Change, Torque_GiE_prev);
307 
308     //-------------------------------------------------------------------------
309     // Control
310     //-------------------------------------------------------------------------
311     volatile int Vd, Vq;                                // Control outs
312     static uint16_t gen_delay = 0;                          // Generator period counter
313     static uint16_t gen_angle = 0;                          // Generator angle counter
314 
315     // Simple angle generator for manual mode
316     // The motor should rotate regardless of the encoder output
317     if (gen_delay >= FixPeriod) {               // Period loop
318         gen_delay = 0;
319         if (gen_angle >= (CPR-1)) {              // Angle loop
320             gen_angle = 0;
321         }
322         else {
323             ++gen_angle;
324         }
325     }
326     else {
327         ++gen_delay;
328     }
329 
330     // Control Vd and Vq depending on work mode
331     switch (Mode) {
332         case MODE_STOPPED:                  // Motor stop
333             Vd = 0;                             // Set zero Vd
334             Vq = 0;                             // Set zero Vq
335             break;
336         case MODE_SPEED:                     // Work mode speed loop
337             Vd = Flux_Out;                      // Sorce Vd from Flux PI
338             Vq = Torque_Out;                    // Sorce Vq from Torque PI
339             break;
340         case MODE_MANUAL_TORQUE_FLUX:        // Manual Vd/Vq with real angle
341             Vd = args[VD_REG];                      // Sorce Vd from register
342             Vq = args[VQ_REG];                      // Sorce Vq from register
343             break;
344         case MODE_MANUAL_TORQUE:              // Manual torque
345             Vd = Flux_Out;                      // Sorce Vd from Flux PI
346             Vq = args[VQ_REG];                      // Sorce Vq from register
347             break;
348         case MODE_SPEED_WITHOUT_TORQUE:        // RPM loop (Torque PI bypass)
349             Vd = Flux_Out;                      // Sorce Vd from Flux PI
350             Vq = RPM_Out;                       // Sorce Vq from RPM PI
351             break;
352         case MODE_TORQUE_WITHOUT_SPEED:        // Disable RPM PI
353             Vd = Flux_Out;                      // Sorce Vd from Flux PI
354             Vq = Torque_Out;                    // Sorce Vq from Torque PI
355             break;
356                                                 // Manual mode with angle generator
357         case MODE_MANUAL_TORQUE_FLUX_FIXED_SPEED:
358             cos_theta = cos_table[gen_angle];  // Generated angle cos
359             sin_theta = sin_table[gen_angle];  // Generated angle sin
360             Vd = args[VD_REG];                      // Sorce Vd from register
361             Vq = args[VQ_REG];                      // Sorce Vq from register
362             break;
363         default:                                // Motor OFF
364             Vd = 0;                             // Set zero Vd
365             Vq = 0;                             // Set zero Vq
```

```
366             break;
367     }
368
369     //------------------------------------------------------------------------
370     // Park Inverse
371     // Valpha = Vd*cos(Theta) - Vq*sin(Theta)
372     // Vbeta = Vq*cos(Theta) + Vd*sin(Theta)
373     //------------------------------------------------------------------------
374     int Vd_cos, Vq_sin, Vq_cos, Vd_sin;             // Partial results
375     int Valpha, Vbeta;                              // Transfom result
376
377     /* It's already done in Park Direct
378     cos_theta = cos_table[Theta];
379     sin_theta = sin_table[Theta];
380     */
381     Vd_cos = Vd * cos_theta;                        // Vd*con(theta)
382     Vq_sin = Vq * sin_theta;                        // Vq*sin(theta)
383     Vq_cos = Vq * cos_theta;                        // Vq*con(theta)
384     Vd_sin = Vd * sin_theta;                        // Vd*sin(theta)
385     Valpha = Clip32((Vd_cos - Vq_sin) >> 15, MIN_LIM, MAX_LIM);   // Remove fractional part
386     Vbeta  = Clip32((Vq_cos + Vd_sin) >> 15, MIN_LIM, MAX_LIM);   // Remove fractional part
387
388     //------------------------------------------------------------------------
389     // Clarke Inverse
390     // Va = Valpha
391     // Vb = [-Valpha + sqrt(3)*Vbeta]/2
392     // Vc = [-Valpha - sqrt(3)*Vbeta]/2
393     //------------------------------------------------------------------------
394     int s3vb;                                   // Partial results
395     int Va, Vb, Vc;                             // Transfom result
396
397     Va = Valpha;                                // Va = Valpha
398     s3vb = Vbeta * SQRT3C;                        // (sqrt(3)*(2^15))*Vbeta
399     Vb = Clip32(((s3vb >> 15) - Valpha) >> 1, MIN_LIM, MAX_LIM);  // (-Valpha +
    sqrt(3)*Vbeta)/2
400     Vc = Clip32((0 - Valpha - (s3vb >> 15)) >> 1, MIN_LIM, MAX_LIM);  // (-Valpha -
    sqrt(3)*Vbeta)/2
401
402     //------------------------------------------------------------------------
403     // SVPWM
404     // Voff= [min(Va, Vb, Vc)+max(Va, Vb, Vc)]/2
405     // Vanew = Va - Voff
406     // Vbnew = Vb - Voff
407     // Vcnew = Vc - Voff
408     //------------------------------------------------------------------------
409     int Vmin, Vmax, Voff;                       // SVPWM internals
410     int Van, Vbn, Vcn;                          // Normalized SVPWM data
411
412     Vmin = (Va < Vb) ? Va : Vb;                 // min(Va,Vb)
413     Vmin = (Vc < Vmin) ? Vc : Vmin;             // min( ,Vc)
414     Vmax = (Va > Vb) ? Va : Vb;                 // max(Va,Vb)
415     Vmax = (Vc > Vmax) ? Vc : Vmax;             // max( ,Vc)
416     Voff = (Vmin + Vmax) >> 1;                  // Division
417     Van = Clip32(Va - Voff, MIN_LIM, MAX_LIM);    // Vanew = Va - Voff
418     Vbn = Clip32(Vb - Voff, MIN_LIM, MAX_LIM);    // Vbnew = Vb - Voff
419     Vcn = Clip32(Vc - Voff, MIN_LIM, MAX_LIM);    // Vcnew = Vc - Voff
420
421     //------------------------------------------------------------------------
422     // Update control register state.
423     //------------------------------------------------------------------------
424     Mode_Prev = Mode;
425
426     //------------------------------------------------------------------------
427     // Create output signal
428     //------------------------------------------------------------------------
429     if (Mode == MODE_FIXED_POSITION) {   // Set the motor to a fixed position.
430         Vcn = 0;                                // Zero Phase C
431         Vbn = 0;                                // Zero Phase B
432         Van = 7000;                             // Power Phase A
433     }
434
435     long long int pwm;                          // Result
436     pwm = MAKE_DATA4(Van, Vbn, Vcn, 0);
437     *B = pwm;                                   // SVPWM result
438     //------------------------------------------------------------------------
439     // Output the data stream, too.
440     //------------------------------------------------------------------------
441     long long int   Data_Out;
442     uint32_t        Control2;
443     uint16_t        Decimation_Max;
444     uint8_t         Data_Source;
445     static uint16_t Decimation_Counter = 0;
446
447     Control2 = args[CONTROL2_REG];
448     Data_Source = Control2 & 0x0Fu;
449     Decimation_Max = (Control2 >> CONTROL2_BIT_DECIMATION) & 0xFFu;
450
```

```
451     switch (Data_Source) {
452     case DATASOURCE_ADC:
453         Data_Out = MAKE_DATA4(GET_INT16(A_copy, 0), GET_INT16(A_copy, 1),
      RPM_filtered_negative, RPM_filtered_negative);
454         break;
455     case DATASOURCE_I_D_Q:
456         Data_Out = MAKE_DATA4(~Id, ~Iq, RPM_filtered_negative, RPM_filtered_negative);
457         break;
458     case DATASOURCE_V_D_Q:
459         Data_Out = MAKE_DATA4(Vd, Vq, 0, RPM_filtered_negative);
460         break;
461     case DATASOURCE_V_ALPHA_BETA:
462         Data_Out = MAKE_DATA4(Valpha, Vbeta, 0, RPM_filtered_negative);
463         break;
464     case DATASOURCE_V_A_B_C:
465         Data_Out = MAKE_DATA4(Va, Vb, Vc, RPM_filtered_negative);
466         break;
467     case DATASOURCE_PWM:
468         Data_Out = MAKE_DATA4(Van, Vbn, Vcn, RPM_filtered_negative);
469         break;
470     default:
471         Data_Out = MAKE_DATA4(0, 0, 0, RPM_filtered_negative);
472         break;
473     }
474 
475     if (Decimation_Counter >= Decimation_Max) {
476         Decimation_Counter = 0;
477         *C = Data_Out;
478     }
479     else {
480         ++Decimation_Counter;
481     }
482 
483     //-------------------------------------------------------------------------
484 //-------------------------------------------------------------------------
485 }
```

Chapter 5

File Documentation

5.1 doxygen/src/main_page.dox File Reference

5.2 foc/foc.cpp File Reference

Implementation of the function foc().

```
#include <ap_int.h>
#include <stdint.h>
#include "foc.h"
#include "sin_cos_table.h"
```

Macros

- #define DC_ACC_SAMPLES (1 << DC_ACC_BITS)

  Number of samples to be accumulated in the DC filters.
- #define FILTER_LENGTH (1 << FILTER_ORDER)

  Length of the speed averaging filter.
- #define MAX_LIM_E 16777215

  Maximum positive value for the integral error.
- #define MIN_LIM_E -16777215

  Minimum negative value for the integral error.
- #define MAKE_DATA4(s0, s1, s2, s3)

  Create 64-bit value out of four 16-bit ones.
- #define GET_INT16(data64, int16index) ((int16_t)((data64 >> (int16index*16)) & 0xFFFF))

Typedefs

- typedef ap_uint< 4 > uint4_t

  A 4-bit unsigned integer.
- typedef ap_uint< 12 > uint12_t

  A 12-bit unsigned integer.
- typedef ap_int< 48 > int48_t

  A 48-bit signed integer type.

Functions

- void foc (long long int *A, long long int *B, long long int *C, int args[ARGS_SIZE], int status[STATUS_SIZE])
  Implementation of the Field-Oriented Control in the FPGA.

5.2.1 Detailed Description

Implementation of the function foc().

This file contains the function implementing the Field-Oriented Control.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.2.2 Macro Definition Documentation

5.2.2.1 DC_ACC_SAMPLES

```
#define DC_ACC_SAMPLES (1 << DC_ACC_BITS)
```

Number of samples to be accumulated in the DC filters.

Definition at line 16 of file foc.cpp.

5.2.2.2 FILTER_LENGTH

```
#define FILTER_LENGTH (1 << FILTER_ORDER)
```

Length of the speed averaging filter.

Definition at line 19 of file foc.cpp.

5.2.2.3 GET_INT16

```
#define GET_INT16(
              data64,
              int16index ) ((int16_t)((data64 >> (int16index*16)) & 0xFFFF))
```

Definition at line 112 of file foc.cpp.

5.2.2.4 MAKE_DATA4

```
#define MAKE_DATA4(
              s0,
              s1,
              s2,
              s3 )
```

Value:

```
(((long long int)(s3)      << 48) & 0xFFFF000000000000LL) | \
    (((long long int)(s2)      << 32) & 0x0000FFFF00000000LL) | \
    (((long long int)(s1)      << 16) & 0x00000000FFFF0000LL) | \
    ( (long long int)(s0)             & 0x000000000000FFFFLL)
```

Create 64-bit value out of four 16-bit ones.

Definition at line 105 of file foc.cpp.

5.2.2.5 MAX_LIM_E

```
#define MAX_LIM_E 16777215
```

Maximum positive value for the integral error.

Definition at line 22 of file foc.cpp.

5.2.2.6 MIN_LIM_E

```
#define MIN_LIM_E -16777215
```

Minimum negative value for the integral error.

Definition at line 24 of file foc.cpp.

5.2.3 Typedef Documentation

5.2.3.1 int48_t

```
typedef ap_int<48> int48_t
```

A 48-bit signed integer type.

Definition at line 33 of file foc.cpp.

5.2.3.2 uint12_t

```
typedef ap_uint<12> uint12_t
```

A 12-bit unsigned integer.

Definition at line 30 of file foc.cpp.

5.2.3.3 uint4_t

```
typedef ap_uint<4> uint4_t
```

A 4-bit unsigned integer.

Definition at line 27 of file foc.cpp.

5.3 foc/foc.h File Reference

Main header file required for using Field-Oriented Control.

```
#include "ap_int.h"
```

Macros

- #define CPR 1000

  Number of encoder steps per one full revolution.
- #define PPR 2

  Number of pole pairs per phase of the motor; full sinus periods per revolution.
- #define DC_ACC_BITS 15

  Number of extra bits in the DC accumulators.
- #define FILTER_ORDER 5

  Order of the RPM boxcar filter.
- #define ARGS_SIZE 16

  Number of argument registers of the FOC.
- #define CONTROL_REG 0

  Control register.
- #define FLUX_SP_REG 1

  Flux setpoint.
- #define FLUX_KP_REG 2

  Flux PI loop proportional factor.
- #define FLUX_KI_REG 3

  Flux PI loop integral factor.
- #define TORQUE_SP_REG 4

  Torque setpoint.
- #define TORQUE_KP_REG 5

  Torque PI loop proportional factor.
- #define TORQUE_KI_REG 6

  Torque PI loop integral factor.
- #define RPM_SP_REG 7

  Speed setpoint, in RPM.
- #define RPM_KP_REG 8

  Speed PI loop proportional factor.
- #define RPM_KI_REG 9

  Speed PI loop integral factor.
- #define ANGLE_SH_REG 10

  Angle shift, in the units of encoder steps.
- #define VD_REG 11

  Fixed Vd.
- #define VQ_REG 12

  Fixed Vq.
- #define FA_REG 13

  Filter coefficient A.
- #define FB_REG 14

  Filter coefficient B.
- #define TRIGGER_REG 14

  Trigger data capture.
- #define CONTROL2_REG 15

  Second control register.
- #define CONTROL_BIT_MODE 0

  Start of the mode bits in register CONTROL_REG.
- #define CONTROL_MAX_MODE 0x0Fu

  Maximum value for the mode bits in CONTROL_REG.
- #define CONTROL_BIT_FIXPERIOD 4

Start of the fixed speed delay in the register CONTROL_REG.

- #define CONTROL_MAX_FIXPERIOD 0xFFFu

  Maximum value for the fixed speed delay in the register CONTROL_REG.
- #define MODE_STOPPED 0u

  Motor stopped.
- #define MODE_SPEED 1u

  Speed control mode.
- #define MODE_MANUAL_TORQUE_FLUX 2u

  Manual torque and flux.
- #define MODE_MANUAL_TORQUE 3u

  Manual torque.
- #define MODE_SPEED_WITHOUT_TORQUE 4u

  Speed control without torque PI.
- #define MODE_TORQUE_WITHOUT_SPEED 5u

  Torque control without speed PI.
- #define MODE_MANUAL_TORQUE_FLUX_FIXED_SPEED 6u

  Manual torque and flux, fixed speed rotation.
- #define MODE_FIXED_POSITION 7u

  Motor position fixed; only phase C is powered.
- #define DATASOURCE_ADC 0

  ADC data.
- #define DATASOURCE_I_ALPHA_BETA 1

  Output of the Clarke transform, the values I_α and I_β.
- #define DATASOURCE_I_D_Q 2

  Output of the Park transform, the values I_d and I_q.
- #define DATASOURCE_V_D_Q 3

  Input to the inverse Park transform, the values V_d and V_q.
- #define DATASOURCE_V_ALPHA_BETA 4

  Output of the inverse Park transform, the values V_α and V_β.
- #define DATASOURCE_V_A_B_C 5

  Output of the inverse Clarke transform, the values V_a, V_b and V_c.
- #define DATASOURCE_PWM 6

  Direct PWM values.
- #define CONTROL2_BIT_ERROR_LIMIT 4u

  The bit position of the error limit in the register CONTROL2_REG.
- #define CONTROL2_BV_ERROR_LIMIT (0xFFFFu << CONTROL2_BIT_ERROR_LIMIT)

  Bitmask of the error limit in the register CONTROL2_REG.
- #define CONTROL2_BV_LED (1u << 20)

  Bit value of the user LED bit in the register CONTROL2_REG.
- #define CONTROL2_BV_RESET_ERROR (1u << 21)

  Bit value of the reset error bit in the register CONTROL2_REG.
- #define CONTROL2_BV_SPREAD_SPECTRUM (1u << 22)

  Bit value of the spread spectrum enable bit in the register CONTROL2_REG.
- #define CONTROL2_BIT_DECIMATION 24

  The bit position of the decimation in the register CONTROL2_REG.
- #define CONTROL2_MAX_DECIMATION 0xFFu

  Maximum value for the decimation factor.
- #define CONTROL2_BITMASK_DECIMATION (CONTROL2_MAX_DECIMATION << CONTROL2_BIT_↩ DECIMATION)

  The bitmask of the decimation value in the register CONTROL2_REG.
- #define STATUS_SIZE 4

Number of status registers of the FOC.

- #define ANGLE_REG 0

  Encoder angle, in encoder steps.
- #define RPM_REG 1

  Speed of rotation, in RPM.
- #define ID_REG 2

  Flux.
- #define IQ_REG 3

  Stator current.
- #define MAX_LIM 32767

  Maximum positive value for saturated arithmetic.
- #define MIN_LIM -32767

  Minimum negative value for saturated arithmetic.
- #define SQRT3A 0x000093CD

  The number $\frac{1}{\sqrt{3}}$ in the Q16.16 format.
- #define SQRT3C 0x0000DDB4

  The number $\sqrt{3}$ in the Q16.16 format.

Functions

- void foc (long long int *A, long long int *B, long long int *C, int args[ARGS_SIZE], int status[STATUS_SIZE])

  Implementation of the Field-Oriented Control in the FPGA.

5.3.1 Detailed Description

Main header file required for using Field-Oriented Control.

This file contains all the necessary definitions for using the Field-Oriented Control C function.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.4 foc/sin_cos_table.h File Reference

Sinus and cosinus tables for foc function.

Variables

- short sin_table [1000]

  Lookup table for the sine function in the Q16.16 format.
- short cos_table [1000]

  Lookup table for the cosine function in the Q16.16 format.

5.4.1 Detailed Description

Sinus and cosinus tables for foc function.

This file contains the lookup tables used by the foc() function.

Important: This file has to be updated whenever CPR has been changed.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.4.2 Variable Documentation

5.4.2.1 cos_table

```
short cos_table[1000]
```

Lookup table for the cosine function in the Q16.16 format.

Important: Update this table when the encoder resolution has changed, i.e. when CPR has changed.

Definition at line 72 of file sin_cos_table.h.

5.4.2.2 sin_table

```
short sin_table[1000]
```

Lookup table for the sine function in the Q16.16 format.

Important: Update this table when the encoder resolution has been changed, i.e. when CPR has been changed.

Definition at line 17 of file sin_cos_table.h.

Index