DII EIP Motor Controller Software Design Specification

AUTHOR

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Department** |  | **Signature** |  | **Print Name** |  | **Date** |
| R&D |  |  |  | Ken Woodland |  |  |

SIGNATURES

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Department** |  | **Signature** |  | **Print Name** |  | **Date** |
| R&D |  |  |  |  |  |  |

Revision History

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rev.** |  | **Date** |  | **Initials** |  | **Description** |
| 0.1 |  | 28-Apr-2009 |  | KK |  | Initial draft |
| 0.2 |  | 04-Sep-2009 |  | KW |  | Updated |
| 0.3 |  | 17-Sep-2009 |  | KK |  | Updated |
| A |  | 16-Oct-2009 |  | DAT |  | Initial Release |
| A.1 |  | 19-Jul-2019 |  | KW |  | Add support for Motor Table and Serial Number RS485 commands for MDUs |
| A.2 |  | 28-Aug-2019 |  | KW |  | Formatted for better readability and added BLDC fault detail |
| A.3 |  | 10-Sep-2019 |  | KW |  | Corrected Program Page Timeout from 400ms to 500ms to reflect code,  Fixed typo in 2.4.4.1 d and reference in 1.2.3.3.2 |
| A.4 |  | 11-Sep-2019 |  | KW |  | Removed Boot RAM test as it was never part of the DII-EIP Boot |
| B |  | 26-Sep-2019 |  | DAT |  | Updated to Revision B |

Glossary

ARM System Controller processor

DSC Digital Signal Controller, specifically the Freescale MC56F8357

HallBus Allegro A3054 multiplexed two-wire hall-effect data bus

MC Motor Controller

MDU Motor Drive Unit

PWM Pulse Width Modulation

SC System Controller

SCI Serial Communication Interface

GPIO General Purpose Input/Output

FET Field effect transistor

Top FET The FET that controls energy flow into to the first winding of the winding pair

Bottom FET The FET that controls energy flow out of the second winding of the winding pair

CW Clockwise

CCW Counterclockwise

References

15000701 DYONICS II EIP Motor Controller Software Requirements Specification

15000721 DYONICS II EIP Inter Controller Protocol Specification

15000286 DYONICS II RS485 Accessory Protocol

15008940 Reliant RS485 Protocol Specification

15000307 Operation of Motor Controller Bootstrap Code

Table of Contents

[1 Overview 4](#_Toc20379785)

[1.1 Architecture 4](#_Toc20379786)

[1.2 Software Flow and Components 4](#_Toc20379787)

[1.2.1 Boot 4](#_Toc20379788)

[1.2.2 Main Program 5](#_Toc20379789)

[1.2.2.1 Initialization 5](#_Toc20379790)

[1.2.2.2 Main Loop 5](#_Toc20379791)

[1.2.3 Interrupts 5](#_Toc20379792)

[1.2.3.1 Watchdog 6](#_Toc20379793)

[1.2.3.2 Timer 6](#_Toc20379794)

[1.2.3.2.1 Interval Overflow 6](#_Toc20379795)

[1.2.3.2.2 Hardware edge 6](#_Toc20379796)

[1.2.3.3 Communication 6](#_Toc20379797)

[1.2.3.3.1 Parallel 6](#_Toc20379798)

[1.2.3.3.2 SCI 6](#_Toc20379799)

[2 Design Specifications 6](#_Toc20379800)

[2.1 Software Upgrade Interface 6](#_Toc20379801)

[2.2 System Controller Interface 7](#_Toc20379802)

[2.2.1 Communication Memory Map 7](#_Toc20379803)

[2.2.2 Parallel Inter Controller Communication 8](#_Toc20379804)

[2.3 Motor Port Interfaces 8](#_Toc20379805)

[2.3.1 Motor Types 9](#_Toc20379806)

[2.3.1.1 No Motor Configuration 9](#_Toc20379807)

[2.3.1.2 ASYNC (brushless) Configuration 9](#_Toc20379808)

[2.3.2 Motor Functions 9](#_Toc20379809)

[2.3.2.1 Asynchronous Motor Control 10](#_Toc20379810)

[2.3.2.1.1 Velocity Control 11](#_Toc20379811)

[2.3.2.1.2 Position Control 11](#_Toc20379812)

[2.3.2.1.2.1 Position Profile 12](#_Toc20379813)

[2.3.2.1.3 Gain Variables 13](#_Toc20379814)

[2.3.2.1.4 PID Control Diagrams 13](#_Toc20379815)

[2.4 Accessory Port Interfaces 14](#_Toc20379816)

[2.4.1 Digital I/O 14](#_Toc20379817)

[2.4.1.1 External Inputs 15](#_Toc20379818)

[2.4.1.2 External Outputs 16](#_Toc20379819)

[2.4.2 Analog Inputs 17](#_Toc20379820)

[2.4.2.1 External 17](#_Toc20379821)

[2.4.2.2 Internal 18](#_Toc20379822)

[2.4.3 HallBus Inputs 18](#_Toc20379823)

[2.4.3.1 Memory 18](#_Toc20379824)

[2.4.3.2 Timing and Debounce 18](#_Toc20379825)

[2.4.4 Serial Interface 19](#_Toc20379826)

[2.4.4.1 RS-485 19](#_Toc20379827)

# Overview

The Motor Controller (MC) is a generic slave dual motor controller within a distributed control architecture with limited specialty functions (e.g., the motor closed loop controllers). It buffers external inputs for use by the application software resident on the System Controller (SC). The application then sends commands to the MC to control the functionality of various connected motors in the arthroscopic shaver handpieces, or Motor Drive Units (MDUs). General foundations for operation in support of the requirement based design specifications are listed here, in this overview.

## Architecture

The upper most modular layer can be represented diagrammatically as follows:

Figure 1

## Software Flow and Components

2.1 Boot Upgrade / Initialization

2.2 System Controller Interface

Configuration and Control Data Structures

2.3 Motor Controllers

2.4 Inputs

Upon power up or hardware reset, the MC software verifies the software integrity and checks for new software available in the upper bank of flash memory. Next, the DSC registers and program RAM are initialized. Lastly, the main program loop begins that monitors inputs and executes asynchronous subroutines that function outside of the interrupt system. Motor control is performed with timer and hardware triggered interrupts.

### Boot

It is the responsibility of the boot code to maintain the integrity of two identical copies of the newest MC software in the upper and lower halves of flash memory. In order to do this, the bootstrap scheme divides the program flash into two equal halves, each 128KB in size. Each half contains a boot image. This helps ensure that at least one usable boot image resides in program flash.

The system runs from the Lower Image FLASH and the Upper Image FLASH contains a mirror. Boot checks upper and lower banks for integrity and copies one bank to the other if necessary to correct corrupted data or copy a new software version to the lower bank for use. The boot code terminates with a status code and calls the Main Program. See 15000307, Operation of Motor Controller Bootstrap Code, for detailed operation.

|  |  |
| --- | --- |
| P$01\_FFFF  P$01\_0000 | Upper Image FLASH |
| P$00\_FFFF  P$00\_0000 | Lower Image FLASH |

Figure 2: Motor Controller Flash Layout

### Main Program

#### Initialization

The MC software initialization sequence proceeds in the follow order:

1. Variable initialization
2. Motor port configuration
3. DSC register initialization
4. Interrupt priority configuration
5. Two second Watchdog enable
6. Enable interrupts

#### Main Loop

The main foreground loop performs the following:

* Refreshes 2 second watchdog timer
* Reads and debounces digital inputs
* Updates analog inputs
* Handles latent SC communication tasks
* Handles latent CAN bus activities
* Handles changes in Motor port configuration

### Interrupts

Three types of interrupts are configured on the MC:

* Watchdog
* Timer
* Communications

**Note:** DSC generated interrupts that halt the processor also exist. These are beyond the scope of this document.

#### Watchdog

The two second watchdog timer is refreshed by the main program loop approximately every millisecond. Should the watchdog timer ever time out, complete DSC/system software reset will occur, leaving the MC in a safe “No Motor” state. Application software can detect this event by testing the wPortType variable for zero.

#### Timer

The majority of the MC motor control functionality takes place within the timer interrupts. These interrupts are configured to trigger either with a time interval overflow, or hardware edge (or both).

##### Interval Overflow

1. A five hundred microsecond multi purpose low priority timer:

* Services the one second Heartbeat communications integrity down counter
* Reads all ADCs
* Reads and debounces both HallBus ports
* RS485 (SCI0 & SCI1) buffer data retrieval
* Provides general purpose down counters

1. Two user definable interval timers, one for each motor port, used for synchronous and/or asynchronous motor control loops.

##### Hardware edge

For each port, commutation for asynchronous (brushless) motors occurs in timer interrupts configured as edge triggered capture timers with overflow enabled. There is one interrupt service for each of the motor armature position Hall Effect sensors, for a total of six (three per port). These individual interrupts all call the same reentrant function to commutate the brushless motor (rotate the electromagnetic field around the stator), and accumulate position data for velocity calculations. As this routine is synchronous with the motor’s armature position when it is moving, it is also used for tracking the armature WindowLock position. The interval overflow trigger forced the interrupts whenever the motor is not moving or moving too slowly.

#### Communication

A parallel interface between the ARM and DSC handles the inter-processor communication. Serial communication with peripheral and/or accessory devices utilizes the SCI interrupts.

##### Parallel

A 16 bit parallel IO bus with some control and handshaking lines connect between the DSC and ARM processors for the passing of configuration and control data. (See section 2.2, below, and 15000721 Dyonics II EIP Inter Controller Protocol.)

##### SCI

The RS485 communications with accessory devices such as footswitches and MDUs is provided with the SCI0 and SCI1 interrupts. (See section 2.4.4, below, and 15000286 Dyonics II RS485 Accessory Protocol.)

# Design Specifications

The design specifications are linked to the software requirements by the requirements’ headings and section numbers used below.

## Software Upgrade Interface

Software upgrades are accomplished with a two step process via the parallel inter-processor interface and the Boot process. First, a new software version must be copied via the parallel bus to the upper bank of flash memory from the SC’s application software. Next (on the next MC reset) the boot code checks upper and lower flash memory banks for CRCs and software version numbers and provides the following primary functions:

|  |  |
| --- | --- |
| **If….** | **Then…** |
| Upper bank is missing or corrupt AND the lower bank is okay. | Copy the lower to the upper. |
| Upper bank is okay AND lower bank is corrupt. | Copy the upper to the lower. |
| Upper bank is okay AND the lower bank is okay AND the version numbers are different. | Copy the upper to the lower. Software upgrades occur by this mechanism. |
| Upper bank is okay AND lower bank is okay AND version numbers are the same. | Do nothing. |

The process of upgrading software falls into the following three scenarios:

1. Any valid software with a unique version number transferred to the upper flash memory bank through the parallel interface from the application software will be evaluated and copied down to the lower flash bank by the Boot process on the next power up or reset cycle.
2. Corrupted software transferred by the application will fail CRC and be replaced with the lower bank image (current software version).
3. Similarly, interrupted transfers will fail CRC and be replaced with the lower bank image.

## System Controller Interface

1. The communications interface uses a parallel bus to pass status, configuration and control data between the SC and memory map on the MC. The SC reads and writes to this space to configure, inquire about and control the MC. See Section 2.2.2, in this document.
2. Additionally, the parallel interface is used to transfer a new program image to the upper flash memory area if so directed by the SC, see Section 2.1 in this document.

### Communication Memory Map

Two memory sets are available to the application software via the parallel bus; i.e. RAM and flash memory. Writing to the flash memory block is the means provided for software upgrades, while reading and writing to RAM provides the means to configure and control motors. The global RAM structure for motor control and I/O is defined by:

/\* Status and control structure \*/

typedef struct {

SnQByte qVersion; // 32 bit version information

SnWord wPortType; // no motor / brushless motor for Ports A&B

Async tBldcA; // port A motor control parameters

Async tBldcB; // port B motor control parameters etc.

HallBus tHallBusA;

HallBus tHallBusB;

DigitalIn tDigital;

AnalogIn tAnalog;

Serial tSerial;

Temperature tTemperature;

SysInterrupt tInterrupt;

SnWord wHeartCount;

SnWord wAuxillary;

SnWord wBuffer[PROFILE\_BUFFER\_SIZE];

SnByte pbSerialNumberA[SERIAL\_NUMBER\_STORE];

SnByte pbSerialNumberB[SERIAL\_NUMBER\_STORE];

} Status\_Control;

### Parallel Inter Controller Communication

The MC is the slave on the parallel bus and responds to commands from the master SC. A command set provides the means to write to flash memory for software upgrades, write to the RS485, and to read and write to RAM for motor configuration and control. See 15000721 Dyonics II EIP Inter Controller Protocol for more detail.

## Motor Port Interfaces

1. Dual port design is provided via two separate timer interrupts, two independent configuration, control and data memory structures and several configuration dependent reentrant functions. The timer interrupts call the reentrant motor control function.

void PortA\_Timer\_Interrupt(void) // Port A motor control loop timer

void PortB\_Timer\_Interrupt(void) // Port B motor control loop timer

void ControlBldc(**Async \*ptBldc**, PWM \*ptPwm) // reentrant motor controller

The called motor control function uses the configuration, control and data memory structure specific to its hardware configuration. (For example ptBldc is the pointer to the motor specific memory structures for asynchronous brushless type motors.) These data structures are downloaded from the System Controller or the MDU via the RS485 Accessory Protocol (15000286) in response to the motor / device ID it compiles from digital, HallBus and serial data. The data structures for the motor type configured to either port are:

/\* Brushless motors \*/

typedef struct { // contains all command, configuration and control data

External tEx; // external motor specific motor data

Internal tIn; // internal utility variables

MotorProfile tMotorProfile;

} Async;

ptBldc is divided into three sections:

* External Data, “tEx” - Motor specific parameters that must be initialized by the SC each time a different motor is identified.
* Internal data, “tIn”, - Utility variable space utilized while controlling each motor.
* Position profile coordinate table, “tMotorProfile”.

1. Port drive FETs are initialized in the off state.
2. The MC tests for shorts with an algorithm that ramps PWM from 0 to 25% linearly for one half second while concurrently monitoring port current. If the measured current is greater than 1 amp during this half second period, the test terminates with the port disabled and sets flag ERROR\_ SHORT\_24.
3. The MC tests for the following operational faults and maintains this status in the two Async. tEx.wFault registers:

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Condition** |
| ERROR\_MOTOR\_TAC | 0x0001 | Set when the motor commutate gets on out of sequence hall pattern.  The MC does not clears the bit once set. |
| ERROR\_MOTOR\_STALL | 0x0002 | Set when the motor has power applied to it but is not moving.  The MC clears this bit when the condition goes away. |
| ERROR\_MOTOR\_IHIGH | 0x0004 | Set when motor exceeds 75%of measured stall current (fIhigh).  The MC clears this bit when the condition goes away. |
| ERROR\_MOTOR\_OVERFLOW | 0x0008 | Reserved (Not used) |
| ERROR\_MOTOR\_ILIM | 0x0010 | Set when the motor attempts to go over its set current limit (fIlim).  The MC does not clears the bit once set. |
| ERROR\_MOTOR\_TLIM | 0x0020 | Set when the motor goes into the torque limit set by fFrpmA and wFrpmB.  The MC clears this bit when the condition goes away. |
| ERROR\_SHORT\_24 | 0x0040 | Set when a motor phase short is detected. (See Section 2.2 c. above).  The MC does not clears the bit once set. |
| ERROR\_MOTOR\_TAC\_NOISE | 0x0080 | Set when the motor commutate detects a noisy hall signal.  The MC does not clears the bit once set. |
| ERROR\_MOTOR\_SET\_SPEED | 0x0100 | Set when the requested the motor speed is beyond the valid range.  The MC does not clears the bit once set. |
| ERROR\_MOTOR\_INT\_SAT | 0x0200 | Set when the Integral term of the PID exceeds the limit (sTlim).  The MC clears this bit when the condition goes away. |
| ERROR\_MOTOR\_DIRECTION | 0x0400 | Set when motor is turning in opposite direction from what it is driven.  The MC clears this bit when the condition goes away. |

1. A five hundred microsecond resolution interval programmable countdown timer (wHeartCount) will reset the MC to a quiescent state and sound the beeper it if loses communication with the main SC after the interval has elapsed. Default timing is 15 seconds.
2. If an RS485 MDU supports the Motor Table Command Request, the MDU downloads the motor configuration data instead of the System Controller. See 15000286 RS485 Accessory Protocol and section 2.2.4 below for more detail.

### Motor Types

Brushless motors are controlled with one of the two possible hardware configurations for the MC. Motors that require commutation as they rotate, such as brushless, are controlled by the ASYNC configuration. Hardware configuration is accomplished with:

void ConfigureMotorPorts(void)

High and low bytes of 16 bit wPortType define the configuration for both ports A and B with A defined in bits 0 - 6 and B in bits 8 - 14. Values for the individual port configuration bytes are defined as:

Table 1 – Port Configuration

|  |  |
| --- | --- |
| wPortType Binary Byte Value | Configuration Type |
| X0000000 | No Motor |
| X0000010 | ASYNC (i.e. brushless) |

The MSB of each byte in wPortType is the START/BUSY control bit for the respective port. Port configuration STARTs when the System Controller set one or both of these bits. The bits are cleared by ConfigureMotorPorts() after configuration is complete, thus providing BUSY status to the System Controller.

#### No Motor Configuration

The No Motor configuration disables digital port FET switching for synchronous motor phases, disables the asynchronous commutation interrupts and turns off the PWM outputs.

#### ASYNC (brushless) Configuration

The ASYNC configuration disables digital port FET switching for synchronous motor phases, enables asynchronous commutation edge triggered and timer overflow interrupts, and turns on simultaneous (PWM0 controls all) PWM control. Primarily designed for brushless type motors, the ASYNC configuration allows running brush type motors (self commutating) by configuring the commutation array for current flow through just one phase winding for any and all commutation input patterns.

### Motor Functions

Motor functionality depends on the type of motor being controlled. See subsections for the applicability of SRS items a – h.

#### Asynchronous Motor Control

During asynchronous motor control, electrical energy is applied to a pair of stator windings as determined by the armature position. As the armature moves, the electric field must be switched to another pair of stator winding such that the magnetic field produced is always pushing and pulling the armature around in the desired direction. This switching occurs in the edge triggered Commutate() interrupts. The pattern described by the three motor Hall Effect armature position sensors triggers the interrupt and forms an index into an array to control the motor phase drive FET devices, as defined by the motor design. As the armature rotates, the stator magnetic field is thereby rotated ahead of the armature, thus keeping it moving.

The amount of pull / push force applied to the armature is controlled by using Pulse Width Modulation (PWM) of the electrical energy to the Top FET and having the Bottom FET always “on” so the energy always has a path to return (ground).

SRS items a – h are provided for brushless motor designs. A front-end state process, defined by switch variable wMode and subsequent case table, provides the asynchronous motor functions.

switch (ptBldc->tEx.wMode) {

case MODE\_OFF: // 0 reset

case MODE\_REVERSE: // 1 reverse

case MODE\_FORWARD: // 2 forward

case MODE\_OSCILLATE: // 3 oscillate

case MODE\_POSITION: // 4 position

case MODE\_JOG: // 5 jog

case MODE\_LOCK: // 6 lock

case MODE\_STOP: // 7 WindowLock

case MODE\_DIAGNOSE: // 8 short circuit test

default:

}

1. MODE\_OFF stops motor operation and forces PWM output to zero.
2. MODE\_REVERSE runs motors in the reverse, or CCW direction when viewed from motor end, in the closed velocity loop mode.
3. MODE\_FORWARD runs motors in the forward, or CW direction when viewed from motor end, in the closed velocity loop mode.
4. MODE\_OSCILLATE runs motors in a closed velocity loop mode in an oscillating, or forward / reverse, fashion for a period established by the wDwell variable and at a velocity established by the sVelSet variable.
5. MODE\_POSITION runs motors in a closed position control loop over a course of application definable coordinates.
6. MODE\_JOG synchronously steps the asynchronous motor shaft around and resets the sWinLock output shaft position tracking variable.
7. MODE\_LOCK follows MODE\_STOP and provides a landing pad for the stopped motor armature at the WindowLock position for a period of time in seconds determined by the fStopTime variable. At the end of the fStopTime interval, MODE\_LOCK sets wMode to the OFF state.
8. MODE\_STOP targets the moving armature to stop at the WindowLock position using the position control algorithm.
9. MODE\_DIAGNOSE runs a half second phase to ground short circuit diagnostic test.
10. Default sets the PWM output to a zero state.

Running in a port timer interrupt, the MC is equipped with a high bandwidth subsidized PID controller function comprising independently programmable velocity and position gain coefficients. These gain variables allow the PID algorithm to support a vast variety of 24 volt DC motor sizes, gear ratios and inertial loads, with both open and closed loop operation.

##### Velocity Control

1. The velocity control mode of operation allows motors to run in the desired direction at the application defined set speed +/‑0.5%.
2. Acceleration and deceleration rates are configurable via variables fAccel and fDecel to allow quick velocity changes to step input commands while softening the “feel” of the motor as it changes speeds.
3. wMode motor states utilizing the velocity control path through the PID algorithm:

* MODE\_OFF
* MODE\_FORWARD
* MODE\_REVERSE
* MODE\_OSCILLATE

See Figure 5 in 2.3.2.1.4.

##### Position Control

A position profile is defined by an array of coordinates in the form: % of time, position, with position being defined in revolutions of output shaft. The period of time (in seconds) to transact one complete move profile is contained in the fCycleTime register. The position profile structure template follows:

/\* Time and position coordinates for motor move \*/

typedef struct {

float X; // % time coordinate

float Y; // revolution distance coordinate

} Point;

/\* Time and position profile for motor move \*/

typedef struct {

SnWord Count; // number of point pairs

SnWord Fill;

Point P[2]; // number for stop profile

} Profile;

The Count variable contains the number of point pairs that define the move path. Repeating patterns must end where they begin to prevent creep (unless that is the intent). A simple oscillate profile for making two revolutions in each direction is the default and appears as:

% time Position

0.00 0.0

0.25 1.0625

0.50 2.125

0.75 1.0625

1. 0.0

with Count = 4.

Note: The extra .125 revolution is to accommodate backlash in the motor gear train and clearance in the motor drive fork.

Position is calculated in “TACs”, i.e., changes in the motor Hall Effect pattern. TACs = (Revolutions) x (Gearbox Ratio) x (Number of Poles) x (Number of Phases).

The position profile is downloaded from the application software via the parallel communication bus.

wMode motor states utilizing the position control path through the PID algorithm:

* MODE\_POSITION
* MODE\_LOCK
* MODE\_STOP

###### Position Profile

At each point in the profile, the algorithm calculates the timeand distanceto the next point, and with the knowledge of the motor’s current velocity, determines the accelerationrequired to reach the next point on time.





Figure 2 – Acceleration (shown for an oscillate mode)

Then, for each control loop the new target velocity calculates as:





Figure 3 – Velocity (shown for an oscillate mode)

And with the new target and previous velocities, the new target position calculates as:





Figure 4 – Position (shown for an oscillate mode)

##### Gain Variables

/\* Control loop gain coefficients \*/

typedef struct {

float fKf; // feed forward coefficient

float fKp; // proportional coefficient for erri

float fKi; // integral coefficient for erri

float fKd; // derivative coefficient for erri - erri-1

float fKa; // acceleration coefficient

float fKl; // load compensation coefficient

} Kgain;

Referring to the following PID block diagrams, the gain variables are used to scale the values contained in constituent control blocks, which are summed, then piped through the direction and limit filters of the PWM controller before outputting a PWM duty cycle to the motor drive hardware. Setting appropriate coefficients to zero allows the same variable structure and PID function to be used for both velocity and position control, as well as for different types of motors. For example, setting fKf = 1, and all others = 0 allows for simple open loop velocity control with states MODE\_FORWARD, MODE\_REVERSE and MODE\_OSCILLATE.

##### PID Control Diagrams



Figure 5 PID loop showing the velocity control path



Figure 6 PID loop showing the position control path

## Accessory Port Interfaces

The digital, analog and serial ports provide communications with accessory and peripheral devices.

### Digital I/O

The MC56F8357 contains 6 General Purpose Input/Output (GPIO) ports that can alternately be configured for peripheral function use. The MC utilizes some of these for onboard purposes, and makes the remaining ones available to external devices. Typical usage for the general-purpose digital inputs is for limit switches and/or jumpers used for external device recognition. The digital inputs are polled in the main program. Some port pins are delegated to their peripheral function. See Figure 7 for GPIO allocation and configuration.



Figure 7 - GPIO Map

#### External Inputs

In addition to the raw port status, the MC provides buffered port status with additional utility over the raw input data via some of the following configurable variables in the *DigitalIn* data structure:

/\* DSC digital inputs \*/

typedef struct {

SnWord wStateData[6]; // processed input state data

SnWord wNewData[6]; // debouncing momentary status

SnWord wOldData[6]; // last new data

SnWord wActive[6]; // 1 = active digital input

SnWord wAssert[6]; // 1 = assert event

SnWord wActiveLow[6]; // 1 = active low input (inverts input)

SnWord wInputType[6]; // 1 = push on / push off

SnWord wDebounce[6]; // number of consecutive identical reads

SnWord wEvent; // events and errors

} DigitalIn;

Digital input resides for application use in the wStateData variable and is produced by the function:

static SnWord ReadPortData(DigitalPort\* ptPort, DigitalInput wIndex)

Four variables control the modes of operation of the inputs:

1. wActive – Enables processing of port pin through the ReadPortData function.
2. wDebounce – Validates the states of the port values by specifying the number of consecutive same value reads before the values are made available in the wStateData variable. Noisy inputs that won’t settle within the specified debounce count accept the noisy value and return the error code ERROR\_DIGITAL\_DEBOUNCE in the wEvent variable.
3. wActiveLow – Specifying a 1 in the desired bit position inverts the logic state of the port bit allowing active low inputs to produce a logical high in the respective wStateData bit.
4. wInputType – Momentary or toggle state status (0/1 resp.) allows for momentary type switches to operate in a push on/push off mode. Note: clearing the wStateData bit after it is read allows for single action latching operation.

The remaining variables are either used internally for processing, or unused and reserved for future use. The following table shows the physical layout of available digital inputs:

|  |  |
| --- | --- |
| User Port Pins | Description |
| GPIOD3 | Spare inputs (may by used as outputs, see 2.4.1.2) |
| GPIOD10 - 11 |
| GPIOE10 – 13 |

Table 2 - Digital Inputs

#### External Outputs

Additionally, User Port Pins are available for use as outputs. Accomplish this by writing a 1 to the appropriate bits in ptGPIOX->wDDR.

### Analog Inputs

The MC56F8357 contains two 8 channel 12-bit analog to digital converters, ANA and ANB. All analog channels are read by the MC at a 500 microsecond rate, initiated by general purpose timer interrupt TMRD0. Channels 0-7 of ANA are store in offsets 0-7 and channels 0-7 of ANB are stored in offsets 8-15 in the AnalogIn memory structure (below).

/\* DSC analog inputs \*/

typedef struct {

SnSWord sData[16]; // raw adc data

SnSWord sAverage[16]; // prescaled average input data

SnSWord sOffset[16]; // offset from zero

SnWord wInvert[16]; // TRUE = negate value - offset

SnWord wCount[16]; // number of values to average

float fGain[16]; // gain for 4095 max value

SnWord wAssert[16]; // 1 = assert interrupt

} AnalogIn;

The sAverage variable contains a scaled average value normalized to an offset and selectably inverted to allow decreasing analog voltages to produce increasing digital values. Every 500 microseconds, each analog channel value is independently summed in preparation for calculating its average. The number of summed values is determined by the value in the wCount variable. (Note: care should be taken when selecting a wCount value as it determines the time (in 500 microsecond intervals) for the sAverage value to update.) The fGain variable is multiplied times the average to give up to a 16 bit signed value. wAssert is currently unused.

Analog channel mapping is as follows:

|  |  |  |
| --- | --- | --- |
| Analog Channel | AnalogIn memory  Structure index | Description |
| ANA0 | 0 | Port A 24 volt current |
| ANA1 | 1 | DSC core temperature sensor |
| ANA2 | 2 | On board temperature sensor |
| ANA3 | 3 | Port A HallBus |
| ANA4 | 4 | General purpose input (Port A Drill speed) |
| ANA5 | 5 | VrefA port A 24 volt current limit control voltage |
| ANA6 | 6 | VrefB port B 24 volt current limit control voltage |
| ANA7 | 7 | General purpose input (Port B Drill Speed) |
| ANB0 | 8 | Port B 24 volt current |
| ANB1 | 9 | Port B HallBus |
| ANB2 – 4 | 10 – 12 | General purpose inputs (Foot FWD, REV, OSC) |
| ANB5 – 7 | 13 – 15 | NC |

Table 3 - Analog Inputs

#### External

External analog inputs are used for variable speed triggers and foot pedals on the motor controlling accessories. These are designated “general purpose” in the above table.

#### Internal

The internal analog to digital channels are hard mapped to system information circuit points per the above table.

### HallBus Inputs

The HallBus specialty function resides in the 500 microsecond general purpose timer interrupt TMRD0. Two separate HallBus channels are provided for legacy device support, one for port A, and one for port B and the footswitch.

#### Memory

Data from these channels are contained in the HallBus memory structure:

/\* Hall bus for legacy shaver and footswitch data \*/

typedef struct {

SnWord wDeviceExist; // device present in corresponding bit pattern

SnWord wDeviceActive; // activated hall devices

SnWord wDeviceLatch; // push-on / push-off state data

SnWord wDeviceAssert; // devices that assert system interrupts

SnWord wHallBusVq; // bus quiescent voltage during reset

SnWord wDeltaCnt; // incremented for every hall change

} HallBus;

1. wDeviceExists contains a 1 in the bit position corresponding to the physical address of the hall-effect devices present on the bus.
2. wDeviceActive contains a 1 in the bit position indicating a device with an address matching the number of the corresponding bit position is activated. (Momentary status.)
3. wDeviceLatch contains a 1 in the bit position indicating a device with an address matching the number of the corresponding bit position was activated. Subsequent activation clears the wDeviceLatch bit, providing push-on/push-off operation.
4. wDeviceAssert is currently unused.
5. wHallBusVq contains the digital value of the quiescent voltage of the hall bus, which is a function of bus loading. This enables measurement of resistors bridged across the bus, and can be used for a course form of analog device ID.
6. wDeltaCnt is used for test purposes and increments by one thousand for every debounced HallBus state change and increments by one for every HallBus state change that the debounce rejects.

#### Timing and Debounce

HallBus devices, two in total, are read sequentially at the 500 microsecond timer rate. If a HallBus state read would be less than 400 us from the last HallBus state read, the read will be delayed until the next 500 microsecond timer interval. At the end of the scan (9 ms including the reset overhead), the HallBus wDeviceExist and wDeviceActive values are compared to the previously scanned values. HallBus values are processed whenever four consecutive sets of scanned values are the same.

### Serial Interface

The RS-485 serial interface provides communications with external devices.

#### RS-485

Serial communications with external accessories transfers across three differential RS-485 channels, converging into the two SCI channels of the DSC. They are comprised of two conventional ports, for handheld devices in ports A and B, and one isolated port for floor contacting devices in port A. The RS-485 TRCVR\_HI line is shared with a GPIO line due to limited connector pins, thereby requiring switching between the two depending upon the need. GPIOA10 – 13 for port A, and GPIOD4 – 5 for port B, are used for this purpose and multiplexing between the channels. Baud rate is determined by the application software. See 15000286 Dyonics II RS485 Accessory Protocol for details.

1. All command requests except the Program Page, Motor Table and Serial Number consist of an 8-bit request byte and a 16 bit response word. Each command request byte is sent from the SC to the MC where it is then transmitted over the RS485 bus to the MDU or footswitch. The MDU or footswitch then transmits back the 16 bit response to the MC and the response is relayed back to SC. If there is no response to the command request within 3ms, the MC will notify the SC there was an error.
2. The Program Page command request consists of a 132-byte request and a 16 bit response word. The Program Page command request is sent from the SC to the MC where it is then transmitted over the RS485 bus to the MDU or footswitch. The MDU or footswitch then transmits back the 16 bit response to the MC and the response is relayed back to SC. If there is no response to the command request within 500ms, the MC will notify the SC there was an error.
3. The Motor Table command request consists of an 8-bit request byte and a 35-byte response. The Motor Table command request is sent from the SC to the MC where it is then transmitted over the RS485 bus to the Reliant MDU. The Reliant MDU then transmits back the 35-byte response to the MC where the MC verifies the CRC byte and expands the 34-byte Motor Table structure to the appropriate Communication Memory Map BLDC external data structure (either tBldcA.tExt or tBldcB.tExt depending on the MDU request port). If there is no response to the command request within 10ms or the CRC byte was not valid or the Motor Table data was not valid then the MC will not update BLDC external data structure and will notify the SC there was an error. See 15008940 Reliant RS485 Protocol Specification for details.
4. The Serial Number command request consists of an 8-bit request byte and a 12-byte response. The Serial Number command request is sent from the SC to the MC where it is then transmitted over the RS485 bus to the Reliant MDU. The Reliant MDU then transmits back the 12-byte response to the MC. The MC then verifies the CRC byte and copies the 11-byte Serial Number to the Communication Memory Map (either pbSerialNumberA or pbSerialNumberB depending on the MDU request port). If there is no response to the command request within 10ms or the CRC byte was not valid then the MC will not copy the Serial Number data and will notify the SC there was an error. See 15008940 Reliant RS485 Protocol Specification for detail.