SMC_Actuators

Generated by Doxygen 1.8.20

Chapter 1

Change Log

SMC_Actuators_v1.0

- RS485 communication to SMC actuator controllers
- Uses CRC16MODBUS
- Current implementation requires threading for certain functions (alternative?)
- Still need testing

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 StepData Struct Reference

Sequence of steps stored in the controller.

Data Fields

- uint16_t **MoveMode**
- uint16_t Spd
  2 bytes, 1 = absolute, 2 = relative
- int Pos
  2 bytes, 0-65535 mm/s
- uint16_t Acc
  4 bytes, +-2147483647 0.01mm
- uint16_t Dec
  2 bytes, 0-65535 mm/s2
- uint16_t PushForce
  2 bytes, 0-65535 mm/s2
- uint16_t TrigLevel
  2 bytes, 0-100 %
- uint16_t PushSpd
  2 bytes, 0-100 %
- uint16_t MoveForce
  2 bytes, 0-65535 mm/s
- int AreaOut1
  2 bytes, 0-300 %
- int AreaOut2
  4 bytes, +-2147483647 0.01mm
- int InPos
  4 bytes, +-2147483647 0.01mm

4.1.1 Detailed Description

Sequence of steps stored in the controller.

The documentation for this struct was generated from the following file:

- C:/Users/jai_prajapati/Documents/SourceLibraries/Serial_LIB/SMCActuators_LIB/SMC_Actuators.h

Chapter 5

File Documentation

5.1 C:/Users/jai_prajapati/Documents/SourceLibraries/Serial_LIB/SMC↩ Actuators_LIB/SMC_Actuators.c File Reference

Actuator control library based on "LEC Serial Communication Information" document from SMC, for LEC_6 Series controllers.

Macros

- #define **TIMEOUT** 5.0
- #define **whileTO**(cond, timeOut, ...)
- #define **checkLim**(var, lowlim, hilim)

Functions

- void L2BE (int Input, int Size, uint8_t ∗buffer)

  Converts an int input to big endian notation and store it into buffer.
- void B2LE (uint8_t ∗Input, int Size, int ∗Buffer)

  Converts a uint8_t input to little endian notation (int) and store it into buffer.
- void **checkStepData** (struct StepData ∗StepData)
- int SMCQuery (char ∗SerialDeviceName, uint8_t Address, uint8_t Function, uint8_t Data[256], uint8_t Data↩ Size, uint8_t Reply[MAXREPLYLEN], char errmsg[ERRLEN])

  Compiles a message and sends it to the controller via serial communication (RS485) and populates a reply if the command is not a broadcast.
- int SMCGetErrMsg (uint8_t SMCErrCode, char errmsg[ERRLEN])

  Returns the error message from the documentation related to the error code.
- int SMCMotorOn (char ∗SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Turns motor on and finds origin if not already done.
- int SMCMotorOff (char ∗SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Turns motor off.
- int Initialize_SMC_Actuators (char ∗SerialConfigFile, int MainPanelHandle, char errmsg[ERRLEN])

  Initialize SMC_Actuators library.
- int SMCCheckError (char ∗SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

Checks the controller for errors, returns error code if there is an error, 0 otherwise.

- int SMCClearError (char *SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Sets the RESET output to try and clear the error. Times out after 5s.

- int SMCSetStep (char *SerialDeviceName, uint8_t Address, uint8_t Step, char errmsg[ERRLEN])

  Motor set step.

- int SMCRun (char *SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Run the specified step stored in the controller.

- int SMCRunStep (char *SerialDeviceName, uint8_t Address, uint8_t Step, char errmsg[ERRLEN])

  Run the specified step stored in the controller.

- int SMCStopStep (char *SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Stop running the current step.

- int SMCWriteStep (char *SerialDeviceName, uint8_t Address, uint8_t Step, struct StepData StepData, char errmsg[ERRLEN])

  Run the step stored in the controller.

- int SMCRunWithSpecified (char *SerialDeviceName, uint8_t Address, struct StepData StepData, char errmsg[ERRLEN])

  Run a one time command (Specified data)

- int SMCGetStateData (char *SerialDeviceName, uint8_t Address, int *CurPos, uint16_t *CurSpd, uint16_t *CurThrust, int *TargPos, uint16_t *StepNo, char errmsg[ERRLEN])

  Get the current state of the controller.

- int SMCReadOutput (char *SerialDeviceName, uint8_t Address, enum StateChangeFlags Flag, uint16_t NumBitsToRead, uint8_t DataOut[48], char errmsg[ERRLEN])

  Function 0x01 of SMC controller, reads status of StateChangeFlags.

- int SMCReadInput (char *SerialDeviceName, uint8_t Address, enum StatusFlags Flag, uint16_t NumBitsToRead, uint8_t DataOut[16], char errmsg[ERRLEN])

  Function 0x02 of SMC controller, reads status of StatusFlags.

- int SMCReadData (char *SerialDeviceName, uint8_t Address, uint16_t DataStartAddress, uint16_t NumWordsToRead, uint16_t DataOut[1024], char errmsg[ERRLEN])

  Function 0x03 of SMC controller, reads specified words.

- int SMCForceOutput (char *SerialDeviceName, uint8_t Address, enum StateChangeFlags Flag, int State, char errmsg[ERRLEN])

  Function 0x05 of SMC controller, forces status of StateChangeFlags.

- int SMCEcho (char *SerialDeviceName, uint8_t *DataIn, uint8_t DataLen, uint8_t *DataOut, char errmsg[ERRLEN])

  Function 0x08 of SMC controller, echos the input data.

- int SMCWriteBatchOutput (char *SerialDeviceName, uint8_t Address, enum StateChangeFlags Flag, uint16_t NumBitsToWrite, uint8_t NumOfData, uint8_t *BatchData, char errmsg[ERRLEN])

  Function 0x0F of SMC controller, batch writes StateChangeFlags.

- int SMCWriteData (char *SerialDeviceName, uint8_t Address, uint16_t DataStartAddress, uint16_t NumWordsToWrite, uint8_t *BatchData, char errmsg[ERRLEN])

  Function 0x10 of SMC controller, writes specified words.

Variables

- static int **libInitialized** = 0

5.1.1 Detailed Description

Actuator control library based on "LEC Serial Communication Information" document from SMC, for LEC_6 Series controllers.

Author

Biye Chen

Copyright

Date

8/2/2019 5:39:59 PM

Date	Name	Rev.	Description
05-15-2013	Arxtron	1.↩ 0.0	Initial Release
09-25-2020	Chao Zhang	1.↩ 0.1	Seperate RunStep into two functions SetStep and Run
11-03-2020	Jai Prajapati	1.↩ 0.2	Update main with library template

5.1.2 Macro Definition Documentation

5.1.2.1 checkLim

```
#define checkLim(
               var,
               lowlim,
               hilim )
```

Value:

```
    var = (var<lowlim ? lowlim : var);\
    var = (var>hilim ? hilim : var)
```

5.1.2.2 whileTO

```
#define whileTO(
               cond,
               timeOut,
               ...  )
```

Value:

```
    startTime = Timer();\
    while (cond && (Timer()-startTime)<timeOut)\
    {\
        ProcessSystemEvents();\
        __VA_ARGS__\
    }\
    libErrChk ((Timer()-startTime)>timeOut,"%s\nFunction timed out",__func__);
```

5.1.3 Function Documentation

5.1.3.1 B2LE()

```
void B2LE (
            uint8_t * Input,
            int Size,
            int * Buffer )
```

Converts a uint8_t input to little endian notation (int) and store it into buffer.

Parameters

[IN]	Input Input byte array to be converted
[IN]	Size Size of the input (1-4)
[OUT]	Buffer Buffer to store the converted value

5.1.3.2 Initialize_SMC_Actuators()

```
int Initialize_SMC_Actuators (
            char * SerialConfigFile,
            int MainPanelHandle,
            char errmsg[ERRLEN] )
```

Initialize SMC_Actuators library.

Parameters

in	*SerialConfigFile*	XML Configuration file for Serial_LIB
in	*MainPanelHandle*	Parent panel handle for serial panel Pass 0 to create as parent panel

5.1.3.3 L2BE()

```
void L2BE (
            int Input,
            int Size,
            uint8_t * buffer )
```

Converts an int input to big endian notation and store it into buffer.

Parameters

[IN]	Input Input integer to be converted
[IN]	Size Size of the buffer (1-4)
[OUT]	buffer Buffer to store the converted value

5.1.3.4 SMCCheckError()

```
int SMCCheckError (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Checks the controller for errors, returns error code if there is an error, 0 otherwise.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast

5.1.3.5 SMCClearError()

```
int SMCClearError (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Sets the RESET output to try and clear the error. Times out after 5s.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast

5.1.3.6 SMCEcho()

```
int SMCEcho (
            char * SerialDeviceName,
            uint8_t * DataIn,
            uint8_t DataLen,
```

```
            uint8_t * DataOut,
            char errmsg[ERRLEN] )
```

Function 0x08 of SMC controller, echos the input data.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	DataIn Arbitrary data
[IN]	DataLen Length of the arbitrary data
[OUT]	DataOut Echo of DataIn

5.1.3.7 SMCForceOutput()

```
int SMCForceOutput (
            char * SerialDeviceName,
            uint8_t Address,
            enum StateChangeFlags Flag,
            int State,
            char errmsg[ERRLEN] )
```

Function 0x05 of SMC controller, forces status of StateChangeFlags.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Flag StateChangeFlags
[IN]	State 0 = OFF, 1 = ON

5.1.3.8 SMCGetErrMsg()

```
int SMCGetErrMsg (
            uint8_t SMCErrCode,
            char errmsg[ERRLEN] )
```

Returns the error message from the documentation related to the error code.

Parameters

[IN]	SMCErrCode The third byte of the return message if there is an error

5.1.3.9 SMCGetStateData()

```
int SMCGetStateData (
            char * SerialDeviceName,
            uint8_t Address,
            int * CurPos,
            uint16_t * CurSpd,
            uint16_t * CurThrust,
            int * TargPos,
            uint16_t * StepNo,
            char errmsg[ERRLEN] )
```

Get the current state of the controller.

NOTE: May need to consider changing implementation of drive fns since they all have while loops within. Which means this function can't be called while other fns are running.

Either have this fn run in a separate thread, or have monitoring threads to turn off the motor once INP is reached. Either way, there needs to be threads created.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[OUT]	CurPos Current position +- 2147483647 0.01mm
[OUT]	CurSpd Current speed 0-65535 mm/s
[OUT]	CurThrust Current thrust 0-300 %
[OUT]	TargPos Target position +- 2147483647 0.01mm
[OUT]	StepNo Current step number 0-63

5.1.3.10 SMCMotorOff()

```
int SMCMotorOff (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Turns motor off.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast

5.1.3.11 SMCMotorOn()

```
int SMCMotorOn (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Turns motor on and finds origin if not already done.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast

5.1.3.12 SMCQuery()

```
int SMCQuery (
            char * SerialDeviceName,
            uint8_t Address,
            uint8_t Function,
            uint8_t Data[256],
            uint8_t DataSize,
            uint8_t Reply[MAXREPLYLEN],
            char errmsg[ERRLEN] )
```

Compiles a message and sends it to the controller via serial communication (RS485) and populates a reply if the command is not a broadcast.

Compiles a message and sends it to the controller via serial communication (RS485) and populates a reply if the command is not a broadcast. Message and reply integrity are checked via CRC16MODBUS. Reply is also checked for error SMC errors.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Function SMC functions, see documentation "LEC Serial Communication Information"
[IN]	Data The data related to the function being called
[IN]	DataSize The size of the Data portion of the message (No of bytes in Data)
[OUT]	Reply The reply based on the message sent

5.1.3.13 SMCReadData()

```
int SMCReadData (
            char * SerialDeviceName,
```

```
            uint8_t Address,
            uint16_t DataStartAddress,
            uint16_t NumWordsToRead,
            uint16_t DataOut[1024],
            char errmsg[ERRLEN] )
```

Function 0x03 of SMC controller, reads specified words.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	DataStartAddress Address of the first word to read
[IN]	NumWordsToRead Number of words to read starting from the flag address
[OUT]	DataOut Array of uint16_t storing the returned data words

5.1.3.14 SMCReadInput()

```
int SMCReadInput (
            char * SerialDeviceName,
            uint8_t Address,
            enum StatusFlags Flag,
            uint16_t NumBitsToRead,
            uint8_t DataOut[16],
            char errmsg[ERRLEN] )
```

Function 0x02 of SMC controller, reads status of StatusFlags.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Flag StatusFlags
[IN]	NumBitsToRead Number of bits to read starting from the flag address
[OUT]	DataOut Array of uint8_t storing the returned data bytes (not BITS!!)

5.1.3.15 SMCReadOutput()

```
int SMCReadOutput (
            char * SerialDeviceName,
            uint8_t Address,
            enum StateChangeFlags Flag,
            uint16_t NumBitsToRead,
```

```
            uint8_t DataOut[48],
            char errmsg[ERRLEN] )
```

Function 0x01 of SMC controller, reads status of StateChangeFlags.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Flag StateChangeFlags
[IN]	NumBitsToRead Number of bits to read starting from the flag address
[OUT]	DataOut Array of uint8_t storing the returned data bytes (not BITS!!)

5.1.3.16 SMCRun()

```
int SMCRun (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Run the specified step stored in the controller.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	step 0-63 step number to run

5.1.3.17 SMCRunStep()

```
int SMCRunStep (
            char * SerialDeviceName,
            uint8_t Address,
            uint8_t Step,
            char errmsg[ERRLEN] )
```

Run the specified step stored in the controller.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Step 0-63 step number to run

5.1.3.18 SMCRunWithSpecified()

```
int SMCRunWithSpecified (
```

```
            char * SerialDeviceName,
            uint8_t Address,
            struct StepData StepData ,
            char errmsg[ERRLEN] )
```

Run a one time command (Specified data)

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	StepData StepData structure containing all of the information required for a step

5.1.3.19 SMCSetStep()

```
int SMCSetStep (
            char * SerialDeviceName,
            uint8_t Address,
            uint8_t Step,
            char errmsg[ERRLEN] )
```

Motor set step.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Step 0-63 step number to run

5.1.3.20 SMCStopStep()

```
int SMCStopStep (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Stop running the current step.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast

5.1.3.21 SMCWriteBatchOutput()

```
int SMCWriteBatchOutput (
            char * SerialDeviceName,
            uint8_t Address,
            enum StateChangeFlags Flag,
            uint16_t NumBitsToWrite,
            uint8_t NumOfData,
            uint8_t * BatchData,
            char errmsg[ERRLEN] )
```

Function 0x0F of SMC controller, batch writes StateChangeFlags.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Flag StateChangeFlags
[IN]	NumBitsToWrite Number of bits to write starting from the flag address
[IN]	NumOfData Number of bytes of data to write
[IN]	BatchData The bytes of data to write, the flag (starting) bit corresponds to the LSB of the first byte (Eg. BatchData[0] = 0b00000001 => setting the flag bit to 1

5.1.3.22 SMCWriteData()

```
int SMCWriteData (
            char * SerialDeviceName,
            uint8_t Address,
            uint16_t DataStartAddress,
            uint16_t NumWordsToWrite,
            uint8_t * BatchData,
            char errmsg[ERRLEN] )
```

Function 0x10 of SMC controller, writes specified words.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	DataStartAddress Address of the first word to read
[IN]	NumWordsToWrite Number of word to write starting from the flag address
[IN]	BatchData The bytes of data to write. Keep in mind that all numerical data written should be in Big Endian form.

5.1.3.23 SMCWriteStep()

```
int SMCWriteStep (
```

```
char * SerialDeviceName,
uint8_t Address,
uint8_t Step,
struct StepData StepData ,
char errmsg[ERRLEN] )
```

Run the step stored in the controller.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Step 0-63 step number to write
[IN]	StepData StepData structure containing all of the information required for a step

5.2 C:/Users/jai_prajapati/Documents/SourceLibraries/Serial_LIB/SMC_Actuators_LIB/SMC_Actuators.h File Reference

Data Structures

- struct StepData

  Sequence of steps stored in the controller.

Macros

- #define **LEC6**
- #define **SENDDELAY** 0.02
- #define **MAXREPLYLEN** 2060

Enumerations

- enum StateChangeFlags {
  IN0 = 0x10, **IN1** = 0x11, **IN2** = 0x12, **IN3** = 0x13,
  IN4 = 0x14, **IN5** = 0x15, **HOLD** = 0x18, **SVON** = 0x19,
  DRIVE = 0x1A, RESET = 0x1B, SETUP = 0x1C, JOGN = 0x1D,
  JOGP = 0x1E, **SERIALINPUT** = 0x30 }

  State Change Flags (Y10-Y3F bits), can be read or written.
- enum StatusFlags {
  OUT0 = 0x40, **OUT1** = 0x41, **OUT2** = 0x42, **OUT3** = 0x43,
  OUT4 = 0x44, **OUT5** = 0x45, **BUSY** = 0x48, SVRE = 0x49,
  SETON = 0x4A, INP = 0x4B, AREA = 0x4C, WAREA = 0x4D,
  ESTOP = 0x4E, **ALARM** = 0x4F }

  Status Flags (X40-X4F bits), can only be read.
- enum StateData {
  CurrPos = 0x9000, CurrSpd = 0x9002, CurrThrust = 0x9003, TargPos = 0x9004,
  DriveDataNo = 0x9006, EquipName = 0x000E }

  State Data (D9000-D9006 and D000E words)
- enum SpecifiedData { **StartOp** = 0x9100 }

  Specified Data used for running a move command manually.

Functions

- void **GetStandardErrMsg** (int error, char errmsg[ERRLEN])
- int CVICALLBACK **FunctionSelect** (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
- int CVICALLBACK **RunFunction** (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
- int Initialize_SMC_Actuators (char *SerialConfigFile, int MainPanelHandle, char errmsg[ERRLEN])

  Initialize SMC_Actuators library.
- int SMCMotorOn (char *SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Turns motor on and finds origin if not already done.
- int SMCMotorOff (char *SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Turns motor off.
- int SMCCheckError (char *SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Checks the controller for errors, returns error code if there is an error, 0 otherwise.
- int SMCClearError (char *SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Sets the RESET output to try and clear the error. Times out after 5s.
- int SMCSetStep (char *SerialDeviceName, uint8_t Address, uint8_t Step, char errmsg[ERRLEN])

  Motor set step.
- int SMCRun (char *SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Run the specified step stored in the controller.
- int SMCRunStep (char *SerialDeviceName, uint8_t Address, uint8_t Step, char errmsg[ERRLEN])

  Run the specified step stored in the controller.
- int SMCStopStep (char *SerialDeviceName, uint8_t Address, char errmsg[ERRLEN])

  Stop running the current step.
- int SMCWriteStep (char *SerialDeviceName, uint8_t Address, uint8_t Step, struct StepData StepData, char errmsg[ERRLEN])

  Run the step stored in the controller.
- int SMCRunWithSpecified (char *SerialDeviceName, uint8_t Address, struct StepData StepData, char errmsg[ERRLEN])

  Run a one time command (Specified data)
- int SMCReadOutput (char *SerialDeviceName, uint8_t Address, enum StateChangeFlags Flag, uint16_t NumBitsToRead, uint8_t DataOut[48], char errmsg[ERRLEN])

  Function 0x01 of SMC controller, reads status of StateChangeFlags.
- int SMCReadInput (char *SerialDeviceName, uint8_t Address, enum StatusFlags Flag, uint16_t NumBitsToRead, uint8_t DataOut[16], char errmsg[ERRLEN])

  Function 0x02 of SMC controller, reads status of StatusFlags.
- int SMCReadData (char *SerialDeviceName, uint8_t Address, uint16_t DataStartAddress, uint16_t NumWordsToRead, uint16_t DataOut[1024], char errmsg[ERRLEN])

  Function 0x03 of SMC controller, reads specified words.
- int SMCForceOutput (char *SerialDeviceName, uint8_t Address, enum StateChangeFlags Flag, int State, char errmsg[ERRLEN])

  Function 0x05 of SMC controller, forces status of StateChangeFlags.
- int SMCEcho (char *SerialDeviceName, uint8_t *DataIn, uint8_t DataLen, uint8_t *DataOut, char errmsg[ERRLEN])

  Function 0x08 of SMC controller, echos the input data.
- int SMCWriteBatchOutput (char *SerialDeviceName, uint8_t Address, enum StateChangeFlags Flag, uint16_t NumBitsToWrite, uint8_t NumOfData, uint8_t *BatchData, char errmsg[ERRLEN])

  Function 0x0F of SMC controller, batch writes StateChangeFlags.
- int SMCWriteData (char *SerialDeviceName, uint8_t Address, uint16_t DataStartAddress, uint16_t NumWordsToWrite, uint8_t *BatchData, char errmsg[ERRLEN])

  Function 0x10 of SMC controller, writes specified words.
- int SMCGetStateData (char *SerialDeviceName, uint8_t Address, int *CurPos, uint16_t *CurSpd, uint16_t *CurThrust, int *TargPos, uint16_t *StepNo, char errmsg[ERRLEN])

  Get the current state of the controller.

5.2.1 Detailed Description

Author

Biye Chen

Copyright

Date

8/2/2019 5:39:59 PM

5.2.2 Enumeration Type Documentation

5.2.2.1 StateChangeFlags

`enum StateChangeFlags`

State Change Flags (Y10-Y3F bits), can be read or written.

Enumerator

DRIVE	Motor On/Off.
RESET	Start/stop motion.
SETUP	Error/alarm reset.
JOGN	Return to origin.

5.2.2.2 StateData

`enum StateData`

State Data (D9000-D9006 and D000E words)

Enumerator

CurrSpd	4 bytes, +-2147483647 0.01mm
CurrThrust	2 bytes, 0-65535 mm/s
TargPos	2 bytes, 0-300 %
DriveDataNo	4 bytes, +-2147483647 0.01mm
EquipName	2 bytes, 0-63 step no.

5.2.2.3 StatusFlags

```
enum StatusFlags
```

Status Flags (X40-X4F bits), can only be read.

Enumerator

SVRE	Servo is moving.
SETON	Servo Ready, on when SVON=1.
INP	On when return to origin is done.
AREA	In position, on when operation is complete.
WAREA	On when between Area1 and Area2.

5.2.3 Function Documentation

5.2.3.1 Initialize_SMC_Actuators()

```
int Initialize_SMC_Actuators (
            char * SerialConfigFile,
            int MainPanelHandle,
            char errmsg[ERRLEN] )
```

Initialize SMC_Actuators library.

Parameters

in	*SerialConfigFile*	XML Configuration file for Serial_LIB
in	*MainPanelHandle*	Parent panel handle for serial panel Pass 0 to create as parent panel

5.2.3.2 SMCCheckError()

```
int SMCCheckError (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Checks the controller for errors, returns error code if there is an error, 0 otherwise.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast

5.2.3.3 SMCClearError()

```
int SMCClearError (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Sets the RESET output to try and clear the error. Times out after 5s.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast

5.2.3.4 SMCEcho()

```
int SMCEcho (
            char * SerialDeviceName,
            uint8_t * DataIn,
            uint8_t DataLen,
            uint8_t * DataOut,
            char errmsg[ERRLEN] )
```

Function 0x08 of SMC controller, echos the input data.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	DataIn Arbitrary data
[IN]	DataLen Length of the arbitrary data
[OUT]	DataOut Echo of DataIn

5.2.3.5 SMCForceOutput()

```
int SMCForceOutput (
            char * SerialDeviceName,
```

```
            uint8_t Address,
            enum StateChangeFlags Flag,
            int State,
            char errmsg[ERRLEN] )
```

Function 0x05 of SMC controller, forces status of StateChangeFlags.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Flag StateChangeFlags
[IN]	State 0 = OFF, 1 = ON

5.2.3.6 SMCGetStateData()

```
int SMCGetStateData (
            char * SerialDeviceName,
            uint8_t Address,
            int * CurPos,
            uint16_t * CurSpd,
            uint16_t * CurThrust,
            int * TargPos,
            uint16_t * StepNo,
            char errmsg[ERRLEN] )
```

Get the current state of the controller.

NOTE: May need to consider changing implementation of drive fns since they all have while loops within. Which means this function can't be called while other fns are running.

Either have this fn run in a separate thread, or have monitoring threads to turn off the motor once INP is reached. Either way, there needs to be threads created.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[OUT]	CurPos Current position +- 2147483647 0.01mm
[OUT]	CurSpd Current speed 0-65535 mm/s
[OUT]	CurThrust Current thrust 0-300 %
[OUT]	TargPos Target position +- 2147483647 0.01mm
[OUT]	StepNo Current step number 0-63

5.2.3.7 SMCMotorOff()

```
int SMCMotorOff (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Turns motor off.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast

5.2.3.8 SMCMotorOn()

```
int SMCMotorOn (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Turns motor on and finds origin if not already done.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast

5.2.3.9 SMCReadData()

```
int SMCReadData (
            char * SerialDeviceName,
            uint8_t Address,
            uint16_t DataStartAddress,
            uint16_t NumWordsToRead,
            uint16_t DataOut[1024],
            char errmsg[ERRLEN] )
```

Function 0x03 of SMC controller, reads specified words.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	DataStartAddress Address of the first word to read
[IN]	NumWordsToRead Number of words to read starting from the flag address
[OUT]	DataOut Array of uint16_t storing the returned data words

5.2.3.10 SMCReadInput()

```
int SMCReadInput (
            char * SerialDeviceName,
            uint8_t Address,
            enum StatusFlags Flag,
            uint16_t NumBitsToRead,
            uint8_t DataOut[16],
            char errmsg[ERRLEN] )
```

Function 0x02 of SMC controller, reads status of StatusFlags.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Flag StatusFlags
[IN]	NumBitsToRead Number of bits to read starting from the flag address
[OUT]	DataOut Array of uint8_t storing the returned data bytes (not BITS!!)

5.2.3.11 SMCReadOutput()

```
int SMCReadOutput (
            char * SerialDeviceName,
            uint8_t Address,
            enum StateChangeFlags Flag,
            uint16_t NumBitsToRead,
            uint8_t DataOut[48],
            char errmsg[ERRLEN] )
```

Function 0x01 of SMC controller, reads status of StateChangeFlags.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Flag StateChangeFlags
[IN]	NumBitsToRead Number of bits to read starting from the flag address
[OUT]	DataOut Array of uint8_t storing the returned data bytes (not BITS!!)

5.2.3.12 SMCRun()

```
int SMCRun (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Run the specified step stored in the controller.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	step 0-63 step number to run

5.2.3.13 SMCRunStep()

```
int SMCRunStep (
            char * SerialDeviceName,
            uint8_t Address,
            uint8_t Step,
            char errmsg[ERRLEN] )
```

Run the specified step stored in the controller.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Step 0-63 step number to run

5.2.3.14 SMCRunWithSpecified()

```
int SMCRunWithSpecified (
            char * SerialDeviceName,
            uint8_t Address,
            struct StepData StepData ,
            char errmsg[ERRLEN] )
```

Run a one time command (Specified data)

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	StepData StepData structure containing all of the information required for a step

5.2.3.15 SMCSetStep()

```
int SMCSetStep (
            char * SerialDeviceName,
            uint8_t Address,
            uint8_t Step,
            char errmsg[ERRLEN] )
```

Motor set step.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Step 0-63 step number to run

5.2.3.16 SMCStopStep()

```
int SMCStopStep (
            char * SerialDeviceName,
            uint8_t Address,
            char errmsg[ERRLEN] )
```

Stop running the current step.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast

5.2.3.17 SMCWriteBatchOutput()

```
int SMCWriteBatchOutput (
            char * SerialDeviceName,
            uint8_t Address,
            enum StateChangeFlags Flag,
            uint16_t NumBitsToWrite,
            uint8_t NumOfData,
            uint8_t * BatchData,
            char errmsg[ERRLEN] )
```

Function 0x0F of SMC controller, batch writes StateChangeFlags.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Flag StateChangeFlags
[IN]	NumBitsToWrite Number of bits to write starting from the flag address
[IN]	NumOfData Number of bytes of data to write
[IN]	BatchData The bytes of data to write, the flag (starting) bit corresponds to the LSB of the first byte (Eg. BatchData[0] = 0b00000001 => setting the flag bit to 1

5.2.3.18 SMCWriteData()

```
int SMCWriteData (
            char * SerialDeviceName,
            uint8_t Address,
            uint16_t DataStartAddress,
            uint16_t NumWordsToWrite,
            uint8_t * BatchData,
            char errmsg[ERRLEN] )
```

Function 0x10 of SMC controller, writes specified words.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	DataStartAddress Address of the first word to read
[IN]	NumWordsToWrite Number of word to write starting from the flag address
[IN]	BatchData The bytes of data to write. Keep in mind that all numerical data written should be in Big Endian form.

5.2.3.19 SMCWriteStep()

```
int SMCWriteStep (
            char * SerialDeviceName,
            uint8_t Address,
            uint8_t Step,
            struct StepData StepData ,
            char errmsg[ERRLEN] )
```

Run the step stored in the controller.

Parameters

[IN]	SerialDeviceName Name of the controller found in configuration\Serial.xml
[IN]	Address 1-255 for Controller ID, 0 for broadcast
[IN]	Step 0-63 step number to write
[IN]	StepData StepData structure containing all of the information required for a step

Index