The University of Alabama in Huntsville
~ 08/29/2019 ~

PCCC.cpp
Reference Manual

A comprehensive documentation of pccc.cpp during the period of its development phase.

The PCCC.cpp file executes all of the EtherNet/IP Function supported by the OpenPLC. Within this report, each functionality, code structure, and version updates will be detailed.

Table of Contents

Requirements

The PCCC.cpp file is run through OpenPLC. OpenPLC is an open-source Programmable Logic Controller developed by Thiago Alves at the University of Alabama in Huntsville. The project is dedicated to provide a low cost industrial solution for automation and research. This required software is free to download. More information regarding OpenPLC may be viewed here.

Struct Declarations

The received data from the network is parsed into structures by its attribute. The structures contain pointer to the specific byte where the attribute begins. Due to numerous types in which data may be presented, various structs were utilized to hold data within the respected EtherNet/IP format to uphold the network's integrity.

Note: *The number of bytes taken by each attribute within each struct is denoted by //[?]*

Pccc_Header

This struct holds the data that is similar across the PCCC type packet. The position may vary, but these variables are consistent across the Command and Reply Packet Structure in PCCC. The temporary reply packet variables are also part of the structure.

Note: HD *stands for Header information in structures,* ***CMD*** *stands for Command packet information, and* ***RP*** *stands for Reply packet information.*

```
struct pccc_header
{
	unsigned char *Data;
	unsigned char *Data_Size;
	unsigned char *header_length = 5;
	unsigned char *HD_CMD_Code;
	unsigned char *HD_Status;
	unsigned char *HD_TransactionNum;
	unsigned char *HD_Data_Function_Code;
	unsigned char *HD_Ext_Status;

	unsigned char *RP_CMD_T;
	unsigned char *RP_STS_T;
	unsigned char *RP_TRNS_T;
	unsigned char temp = 0x4f;
```

```
      unsigned char *RP_CMD_Code = &temp;
      unsigned char temp2 = 0x00;
      unsigned char *DATA_CONST = &temp2;
};
```

Protected_Logical_Read_Command

This struct holds data associated with the communication command Protected Typed Logical Read with Three Address Fields (Command Code – 0x0f; Function Code – 0xA2) for both the Command Data Packet (denoted CMD) and the Reply Data Packet (denoted RP).

***Note: HD** stands for Header information in structures, **CMD** stands for Command packet information, and **RP** stands for Reply packet information.*

```
struct protected_logical_read_command
{
      unsigned char *CMD_Byte_Size;//[1]
      unsigned char *CMD_File_Num;//[1]
      unsigned char *CMD_File_Type;//[1]
      unsigned char *CMD_Element_Num;//[1]
      unsigned char *CMD_SubElement_Num;//[1]
      unsigned char *RP_EXT_Status;//[1]
      unsigned char *pccc_data_length;
};
```

Protected_Logical_Write_Command

This struct holds data associated with the communication command Protected Typed Logical Write with Three Address Fields (Command Code – 0x0f; Function Code – 0xAA) for both the Command Data Packet (denoted CMD) and the Reply Data Packet (denoted RP).

***Note: HD** stands for Header information in structures, **CMD** stands for Command packet information, and **RP** stands for Reply packet information.*

```
struct protected_logical_write_command
{
      unsigned char *CMD_Byte_Size;//[1]
      unsigned char *CMD_File_Num;//[1]
      unsigned char *CMD_File_Type;//[1]
      unsigned char *CMD_Element_Num;//[1]
      unsigned char *CMD_SubElement_Num;//[1]
      unsigned char *RP_EXT_Status;//[1]
      unsigned char *pccc_data_length;
};
```

Functions

The PCCC.cpp file contains numerous functions to accomplish various tasks. Therefore, this section will be broken into different types as may be seen below.

Note: *The number of bytes taken by each attribute within each struct is denoted by //[--]*

Command Protocol Switch

```
uint16_t Command_Protocol(pccc_header header,unsigned char *buffer, int buffer_size)
```

Command_Protocol ()

Description

This function determines the command that is being sent via the Command Packet so that it can switch to the appropriate function to create the Reply Packet.

Parameters

- **header –** Instance of struct object pccc_header containing needed data size
- **buffer –** The whole packet data of PCCC
- **buffer_size –** The size of the buffer

Return

Return -1 [if command not found]

OR

return temp_var1 [the length of the reply packet]

Parsing PCCC Data

```
int parsePCCCData(unsigned char *buffer, int buffer_size)
```

ParsePCCCData()

Description

This function was created to separate the values inside the buffer into the appropriate structure variables and calls on the Command_Protocol() function. Once we get enip.cpp and pccc.cpp passing the right data and data length, this function will no longer be necessary

Parameters

- **buffer_size –** Size of the data that is being passed
- **buffer –** The whole packet data of PCCC
-

Return

New_pccc_length

ProcessPCCCMessage

```
int parsePCCCData(unsigned char *buffer, int buffer_size)
```

processPCCCMessage()

Description

This function is the main call function for enip.cpp. The function takes in the data from enip.cpp and places the data in the appropriate structure variables. It also begins the call to the other functions in pccc.cpp to begin crafting the reply packet and determining its length.

Parameters

- **buffer_size –** Size of the data that is being passed
- **buffer –** The whole packet data of PCCC

Return

New_pccc_length

Protected_Logical_Read_Reply

```
uint16_t Protected_Logical_Read_Reply(pccc_header, unsigned char *buffer, int buffer_size
```

Protected_Logical_Read_Reply()

Description

This function creates the reply packet for the *Protected Typed Logical Read with Three Address Fields* (Command Code – 0x0f; Function Code – 0xA2) and determines the length of the reply packet. The length of the reply packet is the value that is returned at the end of the function.

Parameters

- **header –** The struct pccc_header
- **buffer –** The whole data packet of PCCC
- **buffer_size –** The size of the buffer

Return

len [reply packet length]

Protected_Logical_Write_Reply

```
uint16_t Protected_Logical_Write_Reply(pccc_header, unsigned char *buffer, int buffer_size);
```

Protected_Logical_Write_Reply()

Description

This function creates the reply packet for the *Protected Typed Logical Write with Three Address Fields* (Command Code – 0x0f; Function Code – 0xAA) and determines the length of the reply packet. The length of the reply packet is the value that is returned at the end of the function.

Parameters

- **header –** The struct pccc_header
- **buffer –** The whole data packet of PCCC
- **buffer_size –** The size of the buffer

Return

len [reply packet length]

PCCC_ReadCoil

```
void Pccc_ReadCoils(unsigned char *buffer, int buffer_size);
```

Pccc_ReadCoils()

Description

This function accesses the data inside the PLC Address and stores the information in an output buffer. Then it writes that data into the buffer so that the reply packet can be crafted. This is for digital information.

Parameters

- **buffer –** The whole data packet of PCCC
- **buffer_size –** The size of the buffer

Return

Void

PCCC_ReadDiscreteInputs

```
void Pccc_ReadDiscreteInputs(unsigned char *buffer, int buffer_size);
```

Pccc_ReadDiscreteInputs()

Description

This function accesses the data inside the PLC Address and stores the information in an input buffer. Then it writes that data into the buffer so that the reply packet can be crafted. This is for digital information.

Parameters

- **buffer –** The whole data packet of PCCC
- **buffer_size –** The size of the buffer

Return

Void

PCCC_ReadHoldingRegisters

```
void Pccc_ReadHoldingRegisters(unsigned char *buffer, int buffer_size);
```

Pccc_ReadHoldingRegisters ()

Description

This function accesses the data inside the PLC Address and stores the information in an output buffer. Then it writes that data into the buffer so that the reply packet can be crafted. This is for analog information.

Parameters

- **buffer –** The whole data packet of PCCC
- **buffer_size –** The size of the buffer

Return

Void

PCCC_WriteCoil

```
void Pccc_WriteCoil(unsigned char *buffer, int buffer_size);
```

Pccc_WriteCoil ()

Description

This function accesses the data inside the PLC Address and writes either a 0 or 1 to the data in PCCC buffer. This decision is based on the contents of the PLC Address. This is for digital information.

Parameters

- **buffer –** The whole data packet of PCCC
- **buffer_size –** The size of the buffer

Return

Void

PCCC_WriteRegister

```
void Pccc_WriteRegister(unsigned char *buffer, int buffer_size);
```

Pccc_WriteRegister ()

Description

This function accesses the data inside the PLC Address and writes that data to equivalent PCCC buffer. This is for analog information.

Parameters

- **buffer –** The whole data packet of PCCC
- **buffer_size –** The size of the buffer

Return

Void

Word_Pccc

```
int word_pccc(unsigned char byte1, unsigned char byte2)
```

word_pccc ()

Description

This function executes the OR operation of two bytes that is passed through the function and returns the result.

Parameters

- **byte1 –** First byte that is passed through the function
- **byte2–** Second byte that is passed through the function

Return

Void

An_Word_Pccc

```
int an_word_pccc(unsigned char byte1, unsigned char byte2)
```

an_word_pccc ()

Description

This function concatenates two bytes and returns the result. This is use in the reading and writing of integer values.

Parameters

- **byte1 –** First byte that is passed through the function
- **byte2–** Second byte that is passed through the function

Return

Void

#Define Functions

The PCCC.cpp file contains numerous #define functions to accomplish various bit and byte manipulation throughout the program. Therefore, this section will give a brief description on what each of these #define functions accomplish.

#define bitRead

```
#define bitRead(value, bit) (((value) >> (bit)) & 0x01)
```

bitRead()

Description

This function reads the specified **bit** in the **value** passed through in order to achieve a reading of a single bit instead of the whole byte/word/etc.

Parameters

- **value –** The data to execute the operations specified in the define on.
- **bit –** The specified bit you wish to change.

#define bitSet

```
#define bitSet(value, bit) ((value) |= (1UL << (bit)))
```

bitSet()

Description

This function sets a specific **bit** in the **value** passed through in order to achieve a bit setting of a single bit. This function is necessary in the bitWrite function.

Parameters

- **value –** The data to execute the operations specified in the define on.
- **bit –** The specified bit you wish to change.

#define bitClear

```
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
```

bitClear()

Description

This function clears a specific **bit** in the **value** passed through in order to achieve a bit clearing of a single bit. This function is necessary in the **bitWrite** function.

Parameters

- **value –** The data to execute the operations specified in the define on.
- **bit –** The specified bit you wish to change.

#define bitWrite

```
#define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) : bitClear(value, bit)
```

bitWrite()

Description

This function takes in a **value**, writes the **bitvalue** to a specific **bit** in the **value** passed through in order to achieve a bit write of a single bit.

Parameters

- **value –** The data to execute the operations specified in the define on.
- **bit –** The specified bit you wish to change.
- **bitvalue –** The bit data value that will be written to value

#define lowByte

```
#define lowByte(w) ((unsigned char) ((w) & 0xff))
```

lowByte()

Description

This function takes a word that is passed in and grabs the low byte of the word.

Parameters

- **W (Word-16 bits) –** Data that has a size of 16-bits.

#define highByte

```
#define highByte(w) ((unsigned char) ((w) >> 8))
```

highByte()

Description

This function takes a word that is passed in and grabs the high byte of the word.

Parameters

- **W (Word-16 bits) –** Data that has a size of 16-bits.

Version Updates

Latest Update: Version 0.6

Version 0.1 (??/??/????) The first version of ENIP.cpp documented featuring:

- Support of command code 0x65

Version 0.2 (05/16/2019) Additional functionality added:

- Support of command code 0x6f with Enip Type: Unknown
- Output file functionality testing

Version 0.3 (05/30/2019) Additional functionality added:

- Support of command code 0x6f with Enip Type: Unconnected
- Output file test for Type: Unconnected
- Added mechanism to determine which Enip Type is exhibited
- Added mechanism to select corresponding response based on Enip Type
- enip_Data struct implemented
- Some function parameters have been modified

Version 0.4 (06/28/2019) Additional functionality added:

- Support of command code 0x6f with Enip Type: Connected
- Support of command code 0x70 with Enip Type: Connected
- Output file test for Type: Connected

Version 0.5 (07/09/2019) Additional functionality added:

- Refactored command code functions
- Added support for PCCC.cpp to allow for variable and dynamic PCCC response messages
- Removed hard coded PCCC response

Version 0.6 (07/09/2019) Additional functionality added:

- Refactored code into three files
  - enipStruct.h
  - outputFileFunctions.cpp
  - enip.cpp

Version 0.7 (08/01/2019) Additional functionality added:

- *Work in Progress*
  - pccc.cpp
- pccc.cpp – develop functionality
  - Read/Write from PLC Address
  - Digital/Analog Read/Write
  - Craft complete response packet

Version 0.8 (08/16/2019) Additional functionality added:

- *Work in Progress*
  - ReadInputReg - Read Analog Input
  -
- Pccc.cpp – adding functionality
  - Read Digital Input and Output
  - Write Digital Input and Output

Version 0.9 (08/28/2019) Additional functionality added:

- *Work in Progress*
  -
- Pccc.cpp – adding functionality
  - Floating Point Supported (32-bit)
  - Integers Supported (16-bit)
- **Pccc.cpp-Removed Function(Analog Input)**
  - **Removed Analog Input functionality from pccc.cpp. PCCC as a protocol does not have a way to access that part of the PLC's memory. It was also discovered that Analog Input is read only and in theory is more protected due to this fact.**

Future Work

This list details the work that is currently being worked on and discusses the future works for the OpenPLC Project and PCCC Protocol.

- Error Handling
  - Check to make sure pccc buffer size is not to small
  - Add Status (STS) bit functionality
    - This bit will allow us to receive errors that the PLC will throw
  - Add Extended Status (EXT STS) bit functionality
    - This bit will allow us to receive more detailed errors from the PLC via the response packet
- Functionality
  - 64-bit memory (ML) support
  - Cycle Times: Have OpenPLC operate at the same speed as the Allen Bradley in terms of clock speed
  - Timeouts: Have OpenPLC and Protocols determine Timeouts based on unresponsive system/user.
- Unconnected Type
  - Develop a tool to test Unconnected Type to verify that this type is supported with the current code