# Updating Field Firmware with PLC

**Author: Jeffrey Hushley**

**Associated Project: Yes**
**Associated Part Family: CY8CPLC20**
**Software Version: PSoC® Designer™ 5.1**

AN62792 describes how to update the user application code of devices in the field with a Cypress Powerline Communication (PLC) device without any external microcontroller or EEPROM. A transmitter project that sends out its user application code over the powerline and a receiver project that receives the data from the powerline and re-configures itself to the new application are attached to this application note.

## Contents

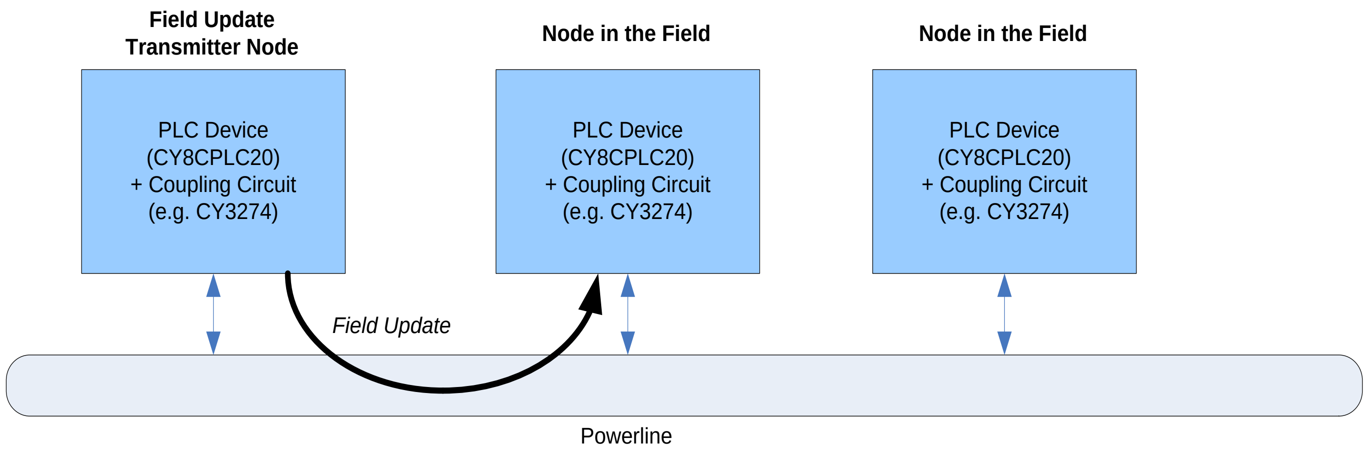
## Introduction

After a system is deployed to the field, it may require updates in the future to either add features or fix issues in the application. If the systems are connected on a communication bus, updates can be performed over the communication bus. This way, the update can be sent from one central location, saving time and money. With the Cypress PLC solution, this concept of a field update can be performed without adding any additional components. The reception of data from the powerline and the update of the application code are performed in one device.

This application note provides an overview of powerline communication, describes the operation of the device's CPU and how the code is read and updated. It describes how to write the application code so that it can be updated over PLC. It also walks through the attached example firmware projects that show how to transmit and receive a field update.

**Note** The example projects are designed to update only the application code. It is not designed to support field updates of the user modules (additional user modules or version updates of user modules). In other words, the first deployment can have any set of user modules, but these cannot be modified by the PLC field upgrade. Only the application code that uses these user modules can be updated.

Figure 1. Field Update System Diagram


Field Update Transmitter Node
Node in the Field
Node in the Field
PLC Device (CY8CPLC20) + Coupling Circuit (e.g. CY3274)
PLC Device (CY8CPLC20) + Coupling Circuit (e.g. CY3274)
PLC Device (CY8CPLC20) + Coupling Circuit (e.g. CY3274)
Field Update
Powerline

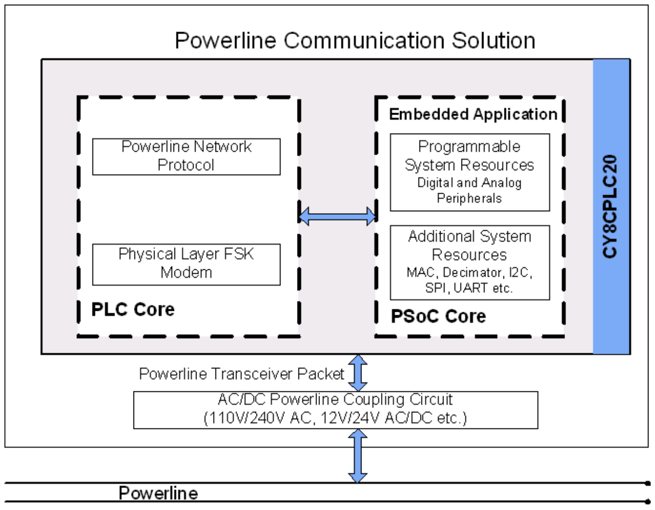
## Powerline Communication

Powerlines are widely available communication medium all over the world for PLC technology. The pervasiveness of Powerline also makes it difficult to predict the characteristics and operation of PLC products. Because of the variable quality of Powerlines around the world, implementing robust communication over Powerline has been an engineering challenge for years. The Cypress PLC solution enables secure and reliable communication over Powerline. Cypress PLC features that enable robust communication over Powerline include:

- Integrated Powerline PHY modem with optimized filters and amplifiers to work with lossy high voltage and low voltage Powerlines.
- Powerline optimized Network Protocol that supports bidirectional communication with acknowledgement based signaling. In case of data packet loss due to noise on the Powerline, the transmitter has the capability to retransmit the data.
- The Powerline Network Protocol also supports 8-bit CRC for error detection and data packet retransmission.
- A Carrier Sense Multiple Access (CSMA) scheme is built into the Network Protocol; it minimizes collision between packet transmissions on the Powerline, supports multiple masters, and enables reliable communication on a bigger network.

A block diagram of the PLC solution with the CY8CPLC20 programmable PLC chip is shown in Figure 2.

Figure 2. Cypress PLC Solution Block Diagram


Powerline Communication Solution
Powerline Network Protocol
Physical Layer FSK Modem
PLC Core
Embedded Application
Programmable System Resources
Digital and Analog Peripherals
Additional System Resources
MAC, Decimator, I2C, SPI, UART etc.
PSoC Core
CY8CPLC20
Powerline Transceiver Packet
AC/DC Powerline Coupling Circuit (110V/240V AC, 12V/24V AC/DC etc.)
Powerline

To interface a PLC device to the Powerline, a coupling circuit is required. The CY3274 High Voltage Programmable PLC Development Kit (DVK)contains the required coupling circuitry for high voltage powerline applications.

The high voltage kits CY3274 are designed with the filtering and power supply circuitry to operate on 110 V to 240 V AC powerlines. They are compliant to the following CENELEC and FCC standards.

- Powerline Signaling (EN50065-1:2001, FCC Part 15)
- Powerline Immunity (EN50065-2-1:2003, EN61000-3-2/3)

- Safety (EN60950)

The CY3274 kits are used to develop an embedded powerline networking application on the CY8CPLC20 programmable PLC device. They contain user interface options such as I$^2$C, RS232, GPIO, analog voltage, LCD display, and LED to develop a full application.

More information on PLC can be found at www.cypress.com/go/plc.

## CPU and Flash Operation

The PLC devices contain an M8C CPU, which executes the code stored in the flash memory of the device. The flash within the specified PSoC® devices is organized in 64-byte blocks. The PLC devices have 512 blocks numbered 0 through 511. User code, when running, can modify the data in these blocks.

Writing to flash requires that an entire 64-byte block be written, even if only 1 byte is to be modified. On the other hand, reading from flash can be performed on a byte-by-byte basis.

Each block is assigned a unique protection level whenever the PSoC programmer or a commercial production programmer programs the device. These levels of protection are shown in Table 1.

By default, the PSoC Designer™ hex file sets each block to level 3, Full Protection. Running firmware cannot change the protection levels. The protection level of each block can be configured in the PSoC Designer Workspace Explorer by editing the *flashsecurity.txt* file.

Table 1. Flash Protection Levels

| Level | Protection Description |
|---|---|
| 0 Unprotected | All reads enabled<br>All writes enabled |
| 1 Factory Upgrade | External reads disabled<br>All writes enabled |
| 2 Field Upgrade | External reads disabled<br>External writes disabled<br>Internal writes enabled |
| 3 Full Protection | External reads disabled<br>All writes disabled |

It is best to give the flash blocks a very high protection level.

In this application, the code that needs to be modified should have a level of 2, since the flash is written internally.

For more details, see the application note AN2015 - PSoC® 1 - Reading and Writing Flash.

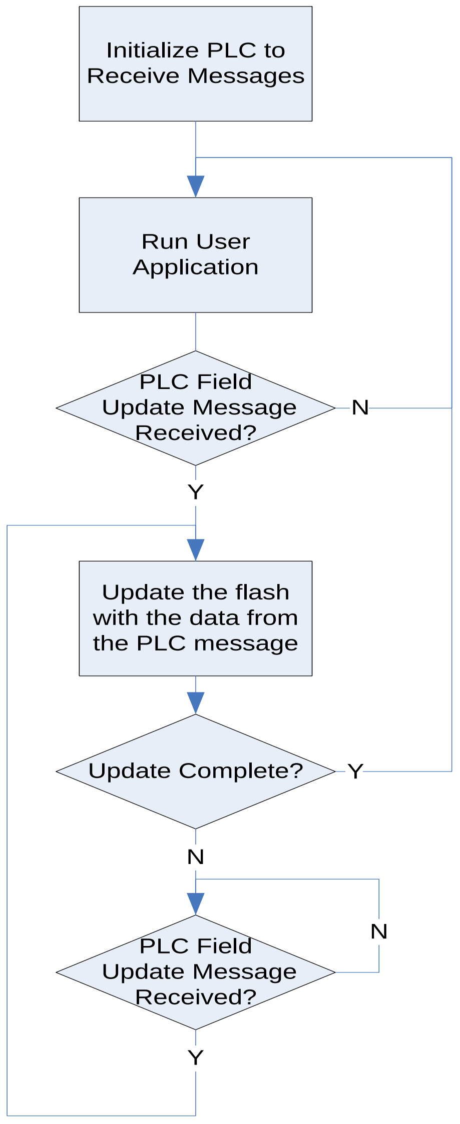
## Field Update Operation Overview

The flow chart in Figure 3 represents the basic operation of the device that receives the field update. After initializing, the device runs its normal application until it receives a field update message over the powerline. When it receives a message, it updates the flash that contains the user code with the received data. At this point, the code is only partially updated, so it is important that user code not be executed until the rest of the field update is received. The device continues to receive the remaining field update messages until the entire application code space is updated. Now, the new user code can be run.

At this point, there are many details that need to be addressed:

- How do I make sure that application code is stored in the same area in flash when I make an update?
- How is the user code sent and received over the powerline?
- How is the user code written to the flash?

Figure 3. Basic Field Update Operation


Initialize PLC to Receive Messages
Run User Application
PLC Field Update Message Received?
N
Y
Update the flash with the data from the PLC message
Update Complete?
Y
N
PLC Field Update Message Received?
N
Y

## Allocating Code in Flash and RAM

Since the user code is being updated from another device, it is necessary to make sure that it always starts from the same location in flash and that it does not overwrite any non-user code. In order to do this, the following steps need to be performed.

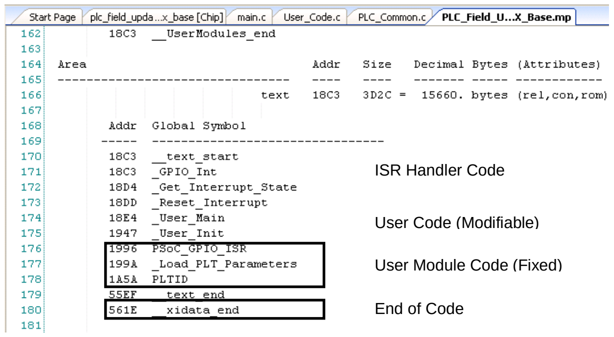
1. Create a field updateable project: Create a new project, place and configure all of the user modules and generate the project. Then, using Windows Explorer, copy the following files from the attached PLC_Field_Update_RX_Base project into the new project:
   - *main.c* (overwrite existing file)
     - *User_Code.c*
     - *ISR_Handler.c*
     - *PLC_Common.c*
     - *PLC_Flash_Update_RX.c*
     - *myData.asm*
     - *User_Code.h*
     - *ISR_Handler.h*
     - *PLC_Common.h*
     - *PLC_Flash_Update_RX.h*
     - *PLC_Flash_Update_Common.h*
     - *custom.lkp*

   To add the .c, .h, and .asm files to the PSoC Designer project, in Workspace Explorer, right-click the project folder and click “Add File…”. The *main.c* file is already in the project so it does not need to be added again. The file *custom.lkp* does not need to be added either.

2. Write your application code: Inside *User_Code.c*, all the user code should been written. Inside *ISR_Handler.c*, all of the code for processing any interrupt service routines (ISRs) should be written. The other *.c* files should not be modified (including *main.c*). After this, the code should be built to see where the user code can be allocated.

3. Determine where to place the code: The user application code must be allocated after all the user module code so that any future changes in user application code do not change the location of the user module code. The location of the code can be determined by opening the map file (extension .mp). In the menu, click Debug → Output Files → Map File. Search for the user code and ISR handler functions and see the address where the code starts and ends. This is the size of the modifiable code, which needs to be assigned to a fixed starting location in memory. In the example below, the ISR handler occupies 0x21 bytes (0x18C3 – 0x18E3) and the user code occupies 0xB2 bytes (0x18E4 – 0x1995).

Figure 4. Flash Location of ISR Handler and User Code



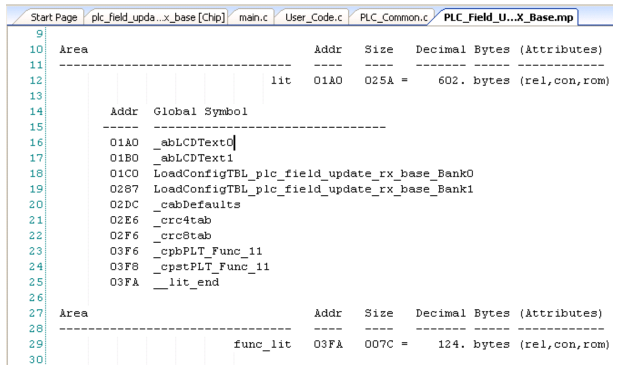
```
        18C3   __UserModules_end

Area                              Addr   Size   Decimal Bytes (Attributes)
--------------------------------  ----   ----   ------- ----- ------------
                            text  18C3   3D2C =  15660. bytes (rel,con,rom)

       Addr  Global Symbol
      -----  --------------------------------
       18C3  __text_start
       18C3  _GPIO_Int
       18D4  _Get_Interrupt_State
       18DD  _Reset_Interrupt
       18E4  _User_Main
       1947  _User_Init
       1996  PSoC_GPIO_ISR
       199A  _Load_PLT_Parameters
       1A5A  PLTID
       55EF  __text_end
       561E  __xidata_end
```

Next, identify the size of any constant arrays or structures (for example, constant strings) that are declared in the user code. In the example project, the user code has two constant strings that are 16 bytes each.

Figure 5. Flash Location of User Code Constants



```
Area                              Addr   Size   Decimal Bytes (Attributes)
--------------------------------  ----   ----   ------- ----- ------------
                             lit  01A0   025A =    602. bytes (rel,con,rom)

       Addr  Global Symbol
      -----  --------------------------------
       01A0  _abLCDText0
       01B0  _abLCDText1
       01C0  LoadConfigTBL_plc_field_update_rx_base_Bank0
       0287  LoadConfigTBL_plc_field_update_rx_base_Bank1
       02DC  _cabDefaults
       02E6  _crc4tab
       02F6  _crc8tab
       03F6  _cpbPLT_Func_11
       03F8  _cpstPLT_Func_11
       03FA  __lit_end

Area                              Addr   Size   Decimal Bytes (Attributes)
--------------------------------  ----   ----   ------- ----- ------------
                        func_lit  03FA   007C =    124. bytes (rel,con,rom)
```

Then, in the map file, identify the end of the user module code (end of the text AREA), which is equivalent to the value of `xidata_end` (see Figure 4). In the example project, the end is 0x561E. The starting address of the modifiable code should start after this. The modifiable code is shifted after this, and therefore we should subtract the size of the modifiable code to determine the starting address. Therefore, the starting address = (End of Code address) – (ISR Handler code size) – (User code size) – (User code constants). In this example, the starting address = 0x561E – 0x21 – 0xB2 – 0x20 = 0x552B. Since flash blocks are 64 (0x40) bytes each, the modifiable code should start at 0x5600.

4. Define areas for the user code in C: In the *User_Code.c* file, above where the functions are defined, uncomment the line:

```
#pragma text:my_code_area
```

5. The name `my_code_area` is given as an example and can be any name. It is critical that the first function in the user code is User_Main(). This is because the *main.c* code calls User_Main() and since the *main.c* code is fixed, the address of User_Main() must always be the same. All other user functions can be re-allocated for a field update because they call each other and all of the user code is updated by a field update. At the end of the user code, uncomment the line:

```
#pragma text:text // switch back to the text
AREA
```

This returns to the general area of flash where non-user code can be stored in flash.

Similarly, each ISR handler function (in this example GPIO_Int) must be at the beginning of its own AREA (for example, `my_isr_area`) because they are called by code that is fixed (e.g. *boot.asm*). Therefore, in *ISR_Handler.c*, above where the ISR is defined, uncomment the line:

```
#pragma text:my_isr_area
```

And at the end of the functions needed for that ISR, uncomment the line:

```
#pragma text:text // switch back to the text
AREA
```

6. Tell the compiler where to allocate the code: In the project folder that also contains *main.c*, there should be a file called *custom.lkp*, which was copied over in Step 1. In the file, add a line that will define the location of your user code. The line is as follows: `-bmy_code_area:<address start>.<address end>`

In this example, we allocate some extra space for the ISR handler section for future expansion. The user code section occupies the rest of the available flash memory.

**Note** The PLC Field Update code occupies the end of the flash (0x7C80 – 0x7FFF)

```
-bmy_ISR_area:0x5600.0x567F
-bmy_code_area:0x5680.0x7C7F
```

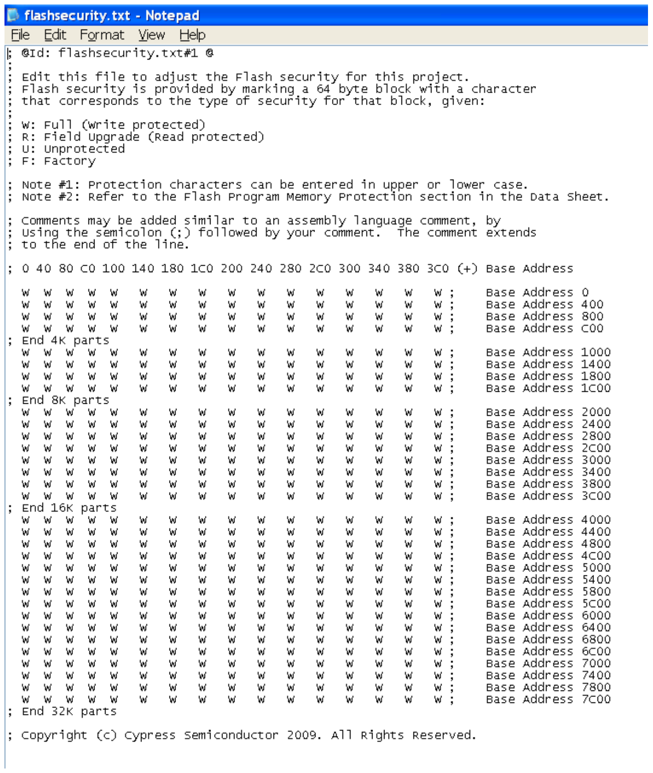
7. The constant arrays need to be allocated in flash memory. These can be allocated after the user code. In this example, the user code uses 0xB2 bytes and starts at 0x5680. Therefore, the constants can be placed after 0x5680 + 0xB2 = 0x5732. For this example, we use 0x5740. After determining the available memory location from the map file, assign the location using the pragma abs_address directive. For example, to allocate a constant string starting at address 0x5740 write the following:

```
#pragma abs_address: 0x5740
const char abText[8]="Example";
#pragma end_abs_address
```

8. Setting the security level in flash. In the project folder that also contains *main.c*, there is a file called

*flashsecurity.txt* (example shown in Figure 6). Since the section of user code needs to be writeable during code execution, it needs to be changed to the field upgrade protection level (“R”).

Figure 6. Example of flashsecurity.txt



```
; @Id: flashsecurity.txt#1 @
;
; Edit this file to adjust the Flash security for this project.
; Flash security is provided by marking a 64 byte block with a character
; that corresponds to the type of security for that block, given:
;
; W: Full (Write protected)
; R: Field Upgrade (Read protected)
; U: Unprotected
; F: Factory

; Note #1: Protection characters can be entered in upper or lower case.
; Note #2: Refer to the Flash Program Memory Protection section in the Data Sheet.

; Comments may be added similar to an assembly language comment, by
; Using the semicolon (;) followed by your comment.  The comment extends
; to the end of the line.

; 0 40 80 C0 100 140 180 1C0 200 240 280 2C0 300 340 380 3C0 (+) Base Address

  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 0
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 400
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 800
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address C00
; End 4K parts
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 1000
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 1400
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 1800
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 1C00
; End 8K parts
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 2000
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 2400
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 2800
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 2C00
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 3000
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 3400
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 3800
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 3C00
; End 16K parts
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 4000
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 4400
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 4800
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 4C00
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 5000
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 5400
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 5800
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 5C00
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 6000
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 6400
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 6800
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 6C00
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 7000
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 7400
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 7800
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 7C00
; End 32K parts

; Copyright (c) Cypress Semiconductor 2009. All Rights Reserved.
```

9. Finally, the data variables should be allocated to their own page so that future variables can be added without moving the non-user variables. To allocate variables in RAM, the following three steps need to be performed:

   a) Declare the variables in an assembly file (the file *myData.asm* is included in the attached receiver example project). An AREA needs to be created for all of the RAM variables. The variable needs to be exported and the name should start with an underscore so that it can be used in the C file. For example:

   ```
   AREA my_User_RAM (RAM, REL, CON)
   export _bMyVariable
   _bMyVariable:BLK 1
   ```

   b) Declare the variable in C by using the external declaration. For example:

   ```
   extern BYTE _bMyVariable;
   ```

   c) Allocate the RAM area in *custom.lkp*. The prefix for declaring the page location should be – B (capital B) followed by the area name, followed by a colon and the page number. For example:

   ```
   -Bmy_User_RAM:2
   ```

10. Setting the starting flash block to be updated over PLC. The starting flash block is determined by dividing the starting address by 64 (0x40). In this example, the starting address is 0x5600. Therefore, the starting flash block is 0x5600 / 0x40 = 0x158. In *PLC_Flash_Update_Common.h*, set the CODE_START_BLOCK constant to this value.

11. At this point, the code can be built and programmed onto the device.

## Sending the User Code with PLC

The PLC family of devices uses a memory array structure that sets the configuration of the PLC interface, contains the transmit data and receive data, and reports the status of operation.

The next sections describe how to transmit and receive a PLC packet using the memory array.

## Transmitting PLC Packets

To transmit a PLC packet, the following registers in the memory array need to be accessed. Only the bits that are important for this application are shown in Table 2.

. For details on these registers, see the PLT user module datasheet in PSoC Designer.

The TX Command ID is typically used to represent the type of data being transmitted. The command IDs 0x01 – 0x2f are reserved for internal commands. The command IDs 0x30 – 0xff are available for custom commands (for example, the field update command).

Since the maximum payload size is 31 bytes and each flash block is 64 bytes, the flash blocks need to be separated into multiple packets (for example. four packets of 16 bytes each). When the receiver receives the packets, it can combine them back into the one 64-byte array for writing to the flash.

After setting the configuration properties (PLC_Mode and TX_Config registers), destination address, command ID, data length, and data payload in the memory array, the PLT_SendMsg() API followed by the PLT_Poll() API should be called to send the message.

Table 2. PLC Memory Array Transmitter Registers

| Offset | Register Name | Access | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x01 | Local_LA_LSB | RW | 8-bit Logical Address | | | | | | | |
| 0x05 | PLC_Mode | RW | TX_Enable | RX_Enable | | | | | | |
| 0x06 | TX_Message_Length | RW | | | | Payload_Length_MASK | | | | |
| 0x07 | TX_Config | RW | TX_SA_Type | TX_DA_Type | | TX_Service _Type | TX_Retry | | | |
| 0x08 | TX_DA | RW | Remote Node Destination Address (8 bytes) | | | | | | | |
| 0x10 | TX_CommandID | RW | TX Command ID | | | | | | | |
| 0x11 | TX_Data | RW | TX Data (31 bytes) | | | | | | | |
| 0x69 | INT_Status | R | Status_Value_ Change | | Status_ UnableTo TX | Status_TX_ NO_ACK | | | | Status_TX_ Data_Sent |

## Processing Received PLC Messages

When a PLC message is received, the PLC memory array registers are updated. The registers that need to get set and the registers that get updated are shown in Table 3. For details on these registers, see the PLT user module datasheet in PSoC Designer.

Table 3. PLC Memory Array Receiver Registers

| Offset | Register Name | Access | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x01 | Local_LA_LSB | RW | 8-bit Logical Address | | | | | | | |
| 0x05 | PLC_Mode | RW | TX_Enable | RX_Enable | | | RX_Overri de | | | |
| 0x40 | RX_Message_INF O | RW | New_RX_Msg | RX_DA_Type | RX_SA_Ty pe | RX_Msg_Length | | | | |
| 0x41 | RX_SA | R | Remote Node Source Address(8 Bytes) | | | | | | | |
| 0x49 | RX_CommandID | R | RX Command ID | | | | | | | |
| 0x4a | RX_Data | R | RX Data (31 bytes) | | | | | | | |
| 0x69 | INT_Status | RW | Status_Value _ Change | | | | | Status_ RX_Pac k et_Drop p ed | Status_RX_ Data_Availa b le | |

## Writing To and Reading From Flash

After an entire 64-byte flash block of data is received, it can be written to the flash. There are two flash API functions related to reading and writing data to and from flash blocks while a user's program is running. These are `bFlashWriteBlock()` and `FlashReadBlock()` and are located in the PSoC project object library. The header files, *flashblock.h* and *flashblock.inc*, are found in the External Headers sub-directory in the Workspace Explorer of the PSoC Designer project.

Code 1 shows the function declarations for these two flash block APIs.

Code 2 shows the definition of the data structure that is passed into the `bFlashWriteBlock()` API function.

Code 3 shows the definition of the data structure that is passed into the `FlashReadBlock()` function. The user must allocate these data structures in SRAM memory to be initialized and then passed into the API functions. The `wARG_ReadCount` in Code 3 is used to specify any valid 16-bit number. This is because flash reads are not restricted to reading one flash block at a time. In fact, the flash read is done on a byte-by-byte basis.

The `wArg_BlockID` is the flash block to write to. Now that the starting address of the user code has been defined in *custom.lkp*, the starting flash block can be determined by dividing the `<address start>` by 64 (0x40).

Code 1. Flash Block API Function Declarations

```
   BYTE        bFlashWriteBlock(FLASH_WRITE_STRUCT * );
   void        FlashReadBlock(FLASH_READ_STRUCT * );
```

Code 2. Flash Block Write API Data Structure

```
typedef struct
{
    WORD            wARG_BlockId;              // 2 bytes: block ID [0…511] to operate upon
    BYTE *          pARG_FlashBuffer;  // 2 bytes: pointer from SRAM for data
    CHAR            cARG_Temperature           // 1 byte: temp. in Celsius (2s complement)
    BYTE            bDATA_PWErase;             // Local variable storage (reserved)
    BYTE            bDATA_PWProgram;           // Local variable storage (reserved)
    BYTE            bDATA_PWMultiplier;        // Local variable storage (reserved)
} FLASH_WRITE_STRUCT;
```

Code 3. Flash Block Read API Data Structure

```
typedef struct
{
    WORD            wARG_BlockId;              // 2 bytes: block ID [0…511] to operate upon
    BYTE *          pARG_FlashBuffer;  // 2 bytes: pointer to SRAM for data
    WORD            wARG_ReadCount             // 2 bytes: number of bytes to read
} FLASH_READ_STRUCT;
```

## Special Considerations

Calling `bFlashWriteBlock()` globally disables interrupts while erasing and writing the specified block. Calling `FlashReadBlock()` does not disable interrupts. Each PSoC device is encoded with an optimized write-pulse duration value. The nominal duration is 10 ms, but this value also depends on individual PSoC die characteristics and the temperature. Higher temperatures use a smaller duration and lower temperatures use a larger than nominal duration. For more details, refer to the specific PSoC device datasheets at http://www.cypress.com.

The supply voltage ($V_{DD}$) must be within the valid operating region during a `bFlashWriteBlock()` operation. It is best to properly use the power-on reset (POR) circuit so that a write operation does not occur if $V_{DD}$ decreases below minimum operating voltage. The code located in *boot.asm* (an automatically generated file) properly sets the correct POR level based on the CPU's operating frequency. It is best not to change the POR level in user code. If the voltage supply is not properly maintained during a write operation, a reset may occur, and the data within the block may not be written correctly and there is no indication of the write failure.

## Performing the Field Update

To update the firmware, the first step is to clone the original project to create a copy. To do this, go to **File → New Project**. After naming the new project and clicking **OK**, the next window should show **Clone Project** at the top. Click **Browse…** and find the *.soc* file in the original project's main directory, then click **Open**. Select **Use the same target device** option and click **OK**.

After creating the new project, perform the following steps to convert it to be able to transmit the field update:

1. Replace the *main.c* file from the attached transmitter example project
2. In *main.c*, set the logical address of this node by modifying the parameter to the Init_PLC() function
3. In *main.c*, set the destination address of the node that receives the update
4. Replace *PLC_Flash_Update_RX.c* (and *.h*) with *PLC_Flash_Update_TX.c* (and *.h*). In the Workspace Explorer, right-click the file to be removed and click "Exclude from project". To add a file, right-click the PLC_Field_Update_TX folder and click "Add File…"
5. Create a new application by modifying *User_Code.c* (and *.h*), *ISR_Handler.c*, and *myData.asm*, if necessary. Note that it is not possible to add user modules at this point because that section of code is not part of the field update.

6. If any constant arrays were defined, make sure that they are allocated just after the end of the user code (using the #pragma abs_address: directive described in step 7 in section 'Allocating Code in Flash and RAM'). If the user code size increases from the original code, but the constant arrays are located in the same spot, the linker may display an error "Trying to write to absolute address 0x5740 but it already contains a value". If this occurs, then either increase the number after #pragma abs_address until the build succeeds or comment out the lines:

```
#pragma abs_address:
#pragma end_abs_address
```

7. Then, find the end of my_code_area in the map file. Then, uncomment the above lines and set the value to the next available location after the end of my_code_area.

8. In the memory map file, determine the amount of flash used by the my_ISR_area, my_code_area, and any constant arrays (usually listed in the memory area). This should be the difference between the start of the my_ISR_area to then end of the constant arrays. Divide this value by 64 (0x40) and round up to get the number of flash blocks that require an update. Then, assign this value to USER_CODE_SIZE_BLOCKS in *User_Code.h*. If unsure of the exact value, it is okay to add 1 to this constant and update an extra block of code, even if it is not used.

9. Build the project and program the device. Upon power up, the new flash code is read from this new project and transmitted via PLC to the node requiring the update. Upon completion of the transmission, the receiving node runs the new application.

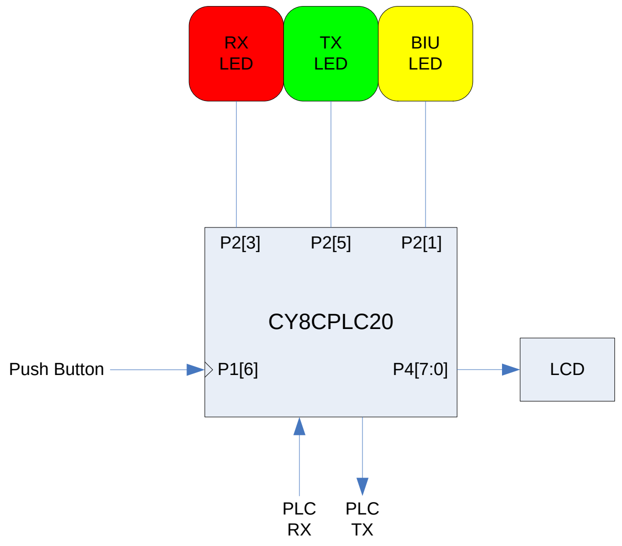
## Example Project

The example project is composed of a receiver project, which starts out as a simple push-button incrementing a value on an LCD, and a transmitter project, which transmits a PLC message when a push-button is pressed and displays the number of packets transmitted and received on the LCD. The transmitter project also transmits its flash contents of the user code to the receiver via PLC. Both projects display the status of the update on the LCD. Upon completion of the flash contents transfer, the receiver behaves the same as the transmitter.

## Receiver Project

The receiver project runs on the CY8CPLC20 PLC device. A block diagram of the receiver system is shown in Figure 7.

Figure 7. CY8CPLC20 Transmitter Block Diagram



The CY8CPLC20 device has the following inputs and outputs.

- **Push Button**: When the GPIO input (P1[6]) transitions from low to high, it generates an interrupt. When the interrupt occurs, the variable bButtonCount is incremented and displayed on the LCD.
- **PLC:** The field update is received from PLC. After the update is complete, the confirmation is sent via PLC. The PLC message types are as follows:
  - **Field Update Start (ID 0x50)**: Sent by the transmitter. This message informs the receiver that the field update is beginning. It contains the number of flash blocks that are transmitted. The LCD displays 'Updating'.
  - **Field Update Data (ID 0x51)**: Sent by the transmitter. This message contains the relative number of the flash block being updated (starting at 0x00), the relative packet number (0x00 – 0x03), followed by 16 bytes of the flash data.
  - **Field Updated Confirmation (ID 0x52)**: Sent by the receiver. This message contains no data and is sent to inform that the field update is successful.

The PLC memory array settings for these packets are shown in Table 6 through Table 8. The RX_Override bit is set to ‘0’ so that the message in the memory array is not overwritten until it is processed.

Table 4. Receiver Memory Array Settings for Field Update Start

| Offset | Register Name | Access | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x01 | Local_LA_LSB | RW | **0x02** | | | | | | | |
| 0x05 | PLC_Mode | RW | TX_Enable = ‘**1**’ | RX_Enable = ‘**1**’ | | | RX_Override = ‘**0**’ | | | |
| 0x40 | RX_Message_INFO | RW | New_RX_Msg = ‘**1**’ | RX_DA_Type =’**0’** | RX_SA_Type = ‘**0**’ (Logical) | RX_Msg_Length = ‘**00001’** | | | | |
| 0x41 | RX_SA | R | Remote Node Source Address(8 Bytes) = **0x01** | | | | | | | |
| 0x49 | RX_CommandID | R | **0x50** | | | | | | | |
| 0x4a | RX_Data[0] | R | Number of flash blocks that will be updated –coordinate | | | | | | | |

Table 5. Receiver Memory Array Settings for Field Update Data

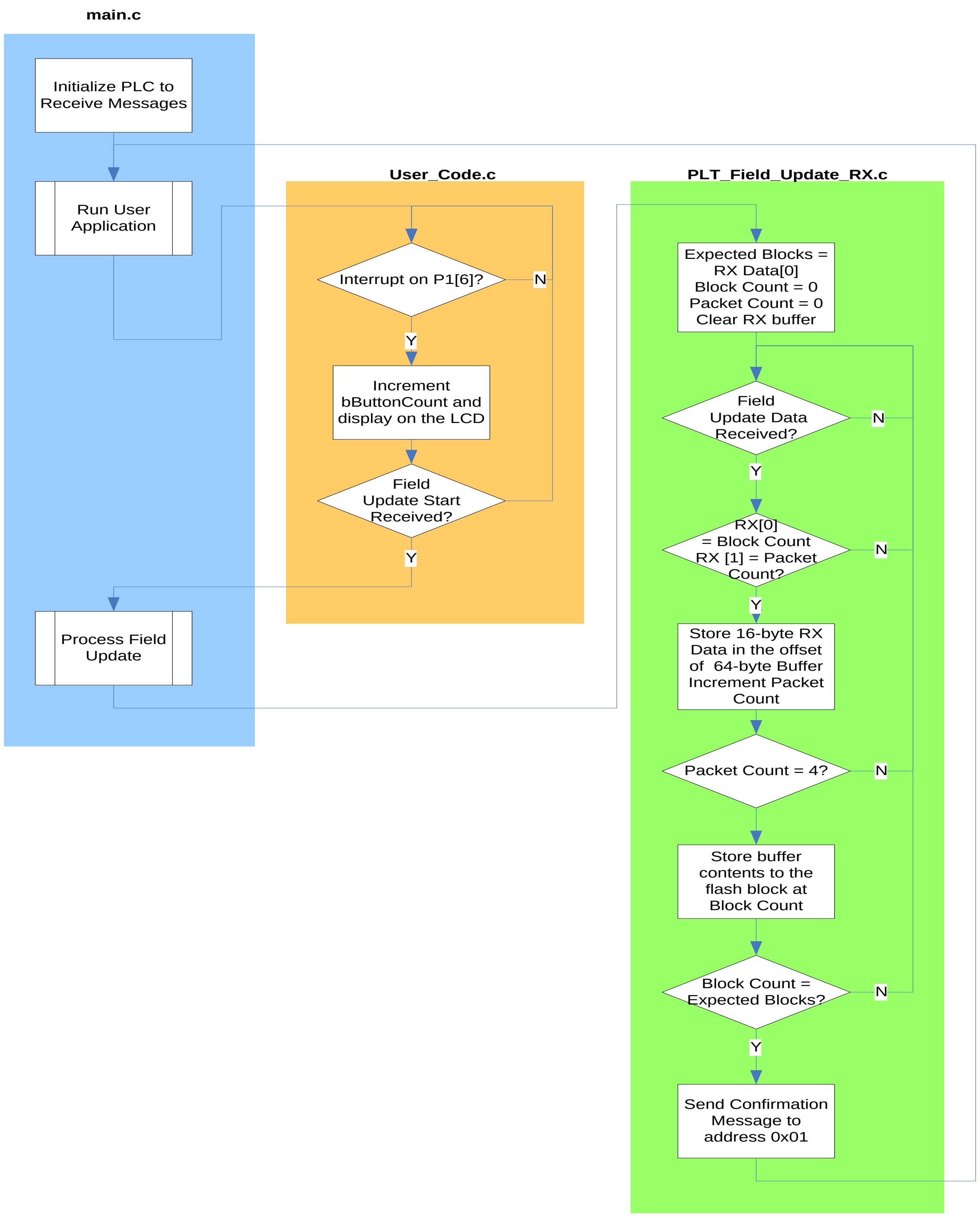
| Offset | Register Name | Access | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x40 | RX_Message_INFO | RW | New_RX_Msg = ‘**1**’ | RX_DA_Type =’**0’** (Direct) | RX_SA_Type = ‘**0**’ (Logical) | RX_Msg_Length = ‘**10010’** | | | | |
| 0x41 | RX_SA | R | Remote Node Source Address(8 Bytes) = **0x01** | | | | | | | |
| 0x49 | RX_CommandID | R | **0x51** | | | | | | | |
| 0x4a | RX_Data[0] | R | Flash Block Number | | | | | | | |
| 0x4b | RX_Data[1] | R | Packet Number (**0x00 – 0x03**) | | | | | | | |
| 0x4c | RX_Data[2-17] | R | Flash Data (16 bytes) | | | | | | | |

Table 6. Transmitter Memory Array Settings for Field Update Confirmation

| Offset | Register Name | Access | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x01 | Local_LA_LSB | RW | **0x02** | | | | | | | |
| 0x05 | PLC_Mode | RW | TX_Enable = ‘**1**’ | RX_Enable = ‘**1**’ | | | | | | |
| 0x06 | TX_Message_Length | RW | Send_Message = ‘**1**’ | | | Payload_Length_MASK = ‘**00000**’ | | | | |
| 0x07 | TX_Config | RW | TX_SA_Type = ‘**0**’ (Logical) | TX_DA_Type = ‘**00**’ (Direct Logical) | | TX_Service _Type = ‘**1**’ (ACK) | TX_Retry = ‘**0011**’ | | | |
| 0x08 | TX_DA | RW | **0x01** | | | | | | | |
| 0x06 | TX_Message_Length | RW | Send_Message = ‘**1**’ | | | Payload_Length_MASK = ‘**00000**’ | | | | |
| 0x10 | TX_CommandID | RW | **0x52** | | | | | | | |

A flow chart of the receiver algorithm is shown in Figure 8. Note that the code is separated across multiple files. The User_Code.c file contains the code that will be updated by the field update. The code in the other files (including *main.c*) is not updated. The files not shown are ISR_Handler.c (which handles the GPIO interrupt and updates a global variable bInterrupted that is read in User_Code.c) and PLC_Common.c (which contains common PLC functions).

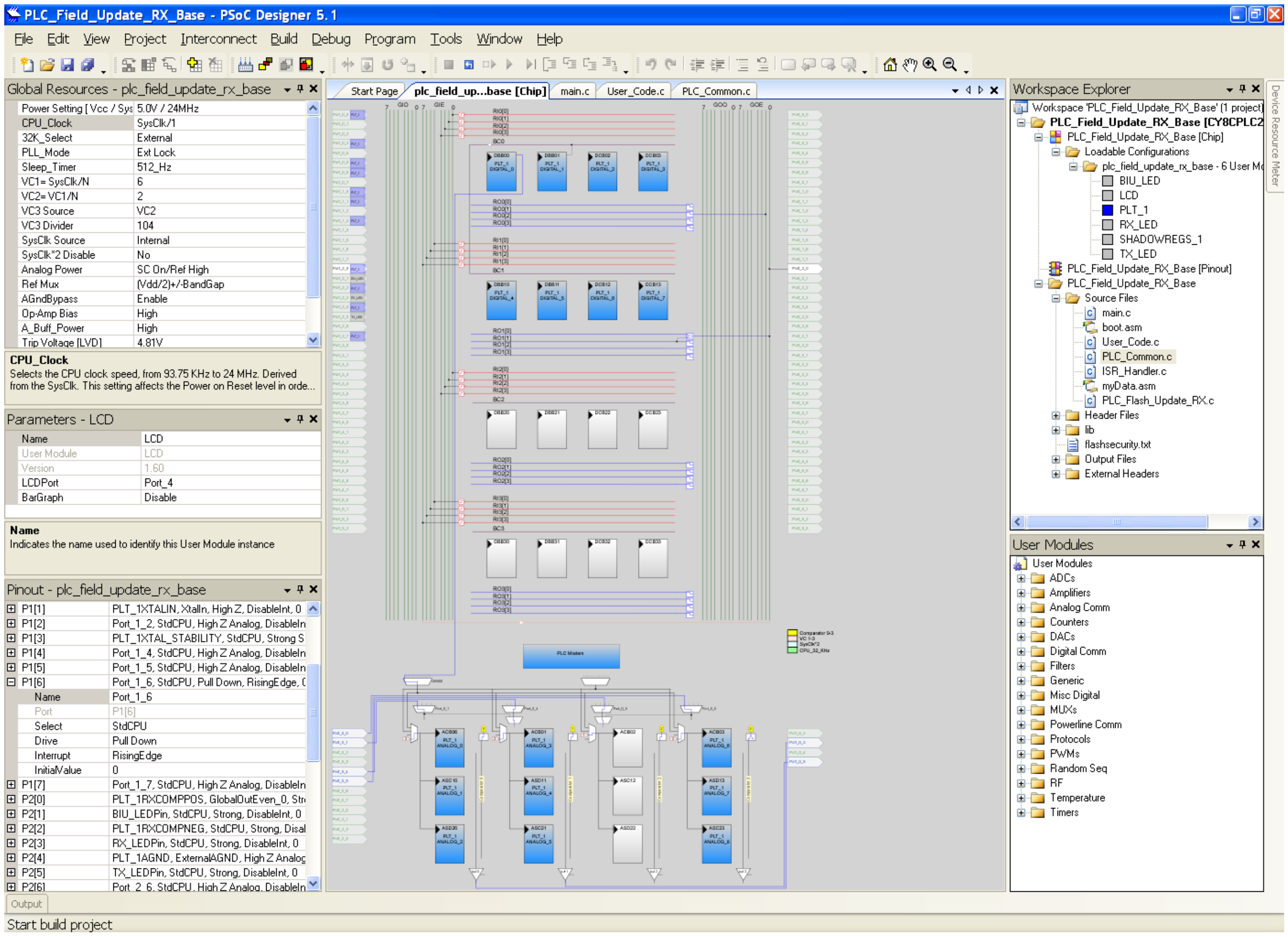
Figure 8. Receiver Flow Chart


main.c
Initialize PLC to Receive Messages
Run User Application
Process Field Update
User_Code.c
Interrupt on P1[6]?
N
Y
Increment bButtonCount and display on the LCD
Field Update Start Received?
Y
PLT_Field_Update_RX.c
Expected Blocks = RX Data[0] Block Count = 0 Packet Count = 0 Clear RX buffer
Field Update Data Received?
N
Y
RX[0] = Block Count RX [1] = Packet Count?
N
Y
Store 16-byte RX Data in the offset of 64-byte Buffer Increment Packet Count
Packet Count = 4?
N
Store buffer contents to the flash block at Block Count
Block Count = Expected Blocks?
N
Y
Send Confirmation Message to address 0x01

## PSoC Designer Project

The project was created with PSoC Designer 5.1. Figure 9 shows the chip level view of the project. The **FSK Modem + Network Stack** user module option is selected for the PLT user module. The configuration of the LCD user module is shown in the same figure. The BIU_LED, RX_LED, and TX_LED user modules are placed at P2[1], P2[3], and P2[5], respectively and are all set to Active High. Port pin P1[6] Drive is set to **Pull Down** and the interrupt is set to **Rising Edge**.

Figure 9. PLC Receiver PSoC Designer Project


PLC_Field_Update_RX_Base - PSoC Designer 5.1
File Edit View Project Interconnect Build Debug Program Tools Window Help
Start Page
plc_field_up...base [Chip]
main.c
User_Code.c
PLC_Common.c
PLC Modem
Device Resource Meter

Global Resources - plc_field_update_rx_base

| | |
|---|---|
| Power Setting [ Vcc / Sys | 5.0V / 24MHz |
| CPU_Clock | SysClk/1 |
| 32K_Select | External |
| PLL_Mode | Ext Lock |
| Sleep_Timer | 512_Hz |
| VC1= SysClk/N | 6 |
| VC2= VC1/N | 2 |
| VC3 Source | VC2 |
| VC3 Divider | 104 |
| SysClk Source | Internal |
| SysClk*2 Disable | No |
| Analog Power | SC On/Ref High |
| Ref Mux | (Vdd/2)+/-BandGap |
| AGndBypass | Enable |
| Op-Amp Bias | High |
| A_Buff_Power | High |
| Trip Voltage [LVD] | 4.81V |

**CPU_Clock**
Selects the CPU clock speed, from 93.75 KHz to 24 MHz. Derived from the SysClk. This setting affects the Power on Reset level in orde...

Parameters - LCD

| | |
|---|---|
| Name | LCD |
| User Module | LCD |
| Version | 1.60 |
| LCDPort | Port_4 |
| BarGraph | Disable |

**Name**
Indicates the name used to identify this User Module instance

Pinout - plc_field_update_rx_base

| | |
|---|---|
| P1[1] | PLT_1XTALIN, XtalIn, High Z, DisableInt, 0 |
| P1[2] | Port_1_2, StdCPU, High Z Analog, DisableIn |
| P1[3] | PLT_1XTAL_STABILITY, StdCPU, Strong S |
| P1[4] | Port_1_4, StdCPU, High Z Analog, DisableIn |
| P1[5] | Port_1_5, StdCPU, High Z Analog, DisableIn |
| P1[6] | Port_1_6, StdCPU, Pull Down, RisingEdge, ( |
| Name | Port_1_6 |
| Port | P1[6] |
| Select | StdCPU |
| Drive | Pull Down |
| Interrupt | RisingEdge |
| InitialValue | 0 |
| P1[7] | Port_1_7, StdCPU, High Z Analog, DisableIn |
| P2[0] | PLT_1RXCOMPPOS, GlobalOutEven_0, Str |
| P2[1] | BIU_LEDPin, StdCPU, Strong, DisableInt, 0 |
| P2[2] | PLT_1RXCOMPNEG, StdCPU, Strong, Disal |
| P2[3] | RX_LEDPin, StdCPU, Strong, DisableInt, 0 |
| P2[4] | PLT_1AGND, ExternalAGND, High Z Analog |
| P2[5] | TX_LEDPin, StdCPU, Strong, DisableInt, 0 |
| P2[6] | Port_2_6, StdCPU, High Z Analog, DisableIn |

Workspace Explorer

- Workspace 'PLC_Field_Update_RX_Base' (1 project)
  - PLC_Field_Update_RX_Base [CY8CPLC2
    - PLC_Field_Update_RX_Base [Chip]
      - Loadable Configurations
        - plc_field_update_rx_base - 6 User M
          - BIU_LED
          - LCD
          - PLT_1
          - RX_LED
          - SHADOWREGS_1
          - TX_LED
    - PLC_Field_Update_RX_Base [Pinout]
    - PLC_Field_Update_RX_Base
      - Source Files
        - main.c
        - boot.asm
        - User_Code.c
        - PLC_Common.c
        - ISR_Handler.c
        - myData.asm
        - PLC_Flash_Update_RX.c
      - Header Files
      - lib
      - flashsecurity.txt
      - Output Files
      - External Headers

User Modules

- User Modules
  - ADCs
  - Amplifiers
  - Analog Comm
  - Counters
  - DACs
  - Digital Comm
  - Filters
  - Generic
  - Misc Digital
  - MUXs
  - Powerline Comm
  - Protocols
  - PWMs
  - Random Seq
  - RF
  - Temperature
  - Timers

Output

Start build project

The firmware is written in C, with the exception of the PLT interrupt routines (in *PLT_1INT.asm*), which are modified to drive transmit, receive, and band-in-use LEDs.

The following files are created:

*User_Code.c*: This is where all of the user’s application code is written. There should be only one function that is called from *main.c* because the calling function needs to always have the same address. This should be the first function in the file so that the starting address does not change whenever there are code modifications. In this case, the function is called User_Main(). Note that there is also an interrupt function that modifies a global variable (bInterrupted), which is used in the User_Main() function.

*ISR_Handler.c*: This is where the user ISRs is processed. The variable bInterrupted needs to be accessible by the code in User_Code.c .

*PLC_Flash_Update_RX.c*: This is where the received PLC packet is processed and the flash is updated.

*PLC_Common.c*: This contains common PLC functions.

*myData.asm*: Contains the user variables, which are allocated to page 2 of the RAM

There is a .h header file associated with each of the .c files.

#### Allocating in Flash Memory

The code was allocated to the flash according to the guidelines defined in section “Allocating Code in Flash and RAM”. The file *custom.lkp* was updated accordingly. In addition, the file *flashsecurity.hex* was updated to set the corrected protection levels for the flash that contains the user code.

**Note** If the hardware configuration is modified (for example, User Modules added), it may extend the initial AREAs and cause overlap with the user-defined AREAs. In this case, the AREAs will need to be shifted.

### Implementation on Hardware

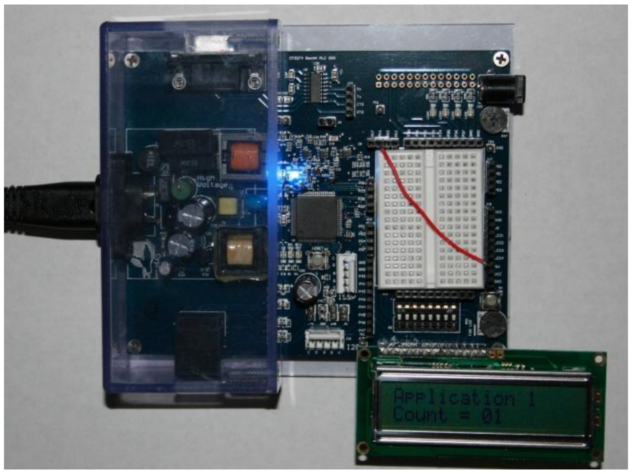
This section describes how to implement the receiver project in hardware.

#### PLC Receiver Hardware

The PLC receiver project was designed to run on the CY3274 High Voltage PLC Development Kit. Follow these steps to set up the system:

1. Connect a jumper wire from SW to P1[6]. See Figure 10 for the location of the headers.
2. Connect the LCD daughter card to the LCD connector.
3. Connect the board to the powerline. The blue power LED should turn ON.
4. Program the firmware with the MiniProg in Reset mode.
5. Disconnect the programmer and reset the board.
6. Press the push-button and observe the LCD. The Count should increment to 01.

Figure 10. CY3274 Receiver Hardware Setup



## Transmitter Project

The transmitter project runs on the CY8CPLC20 PLC device. The hardware is the same as the receiver project since only a field update is expected for this example. Therefore, the block diagram of the transmitter system is the same as the receiver, which was shown in Figure 7.

The CY8CPLC20 device has the following inputs and outputs for the transmitter project:

- **Push Button**: When the GPIO input (P1[6]) transitions from low to high, it generates an interrupt. When the interrupt occurs, the following occurs:
  - The variable bButtonCount is incremented and displayed on the LCD
  - A normal data message (ID 0x09) is transmitted on the powerline. If it is successful, the bSuccessCount is incremented and displayed on the LCD
- **PLC:** The field update is received from PLC. After the update is complete, the confirmation is sent via PLC. The PLC message types are as follows:
  - **Normal Data Message (ID 0x09)**: When a normal data message is received, the variable bRXCount is incremented and displayed on the LCD.
  - **Field Update Start (ID 0x50)**: Sent by the transmitter. This message informs the receiver that the field update is beginning. It contains the number of flash blocks that will be transmitted. The LCD will display “Updating”.
  - **Field Update Data (ID 0x51)**: Sent by the transmitter. This message contains the relative number of the flash block being updated (starting at 0x00), the relative packet number (0x00 – 0x03), followed by 16 bytes of the flash data.
  - **Field Updated Confirmation (ID 0x52)**: Sent by the receiver. This message contains no data and is sent to inform that the field update was successful.

The PLC memory array setting for these packets is shown in Table 6 through Table 8. The RX_Override bit is set to ‘0’ so that the message in the memory array is not overwritten until it is processed.

Table 7. Transmitter Memory Array Settings for Field Update Start

| Offset | Register Name | Access | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x01 | Local_LA_LSB | RW | **0x01** | | | | | | | |
| 0x05 | PLC_Mode | RW | TX_Enable = ‘**1**’ | RX_Enabl e = ‘**1**’ | | | | | | |
| 0x06 | TX_Message_Length | RW | Send_Message = ‘**1**’ | | | Payload_Length_MASK = ‘**00001**’ | | | | |
| 0x07 | TX_Config | RW | TX_SA_Type = ‘**0**’ (Logical) | TX_DA_Type = ‘**00**’ (Direct Logical) | | TX_Service_Type= ‘**1**’ (ACK) | TX_Retry = ‘**0011**’ | | | |
| 0x08 | TX_DA | RW | **0x02** | | | | | | | |
| 0x10 | TX_CommandID | RW | **0x50** | | | | | | | |
| 0x11 | TX_Data[0] | RW | Number of flash blocks that will be updated -coordinate | | | | | | | |

Table 8. Transmitter Memory Array Settings for Field Update Data

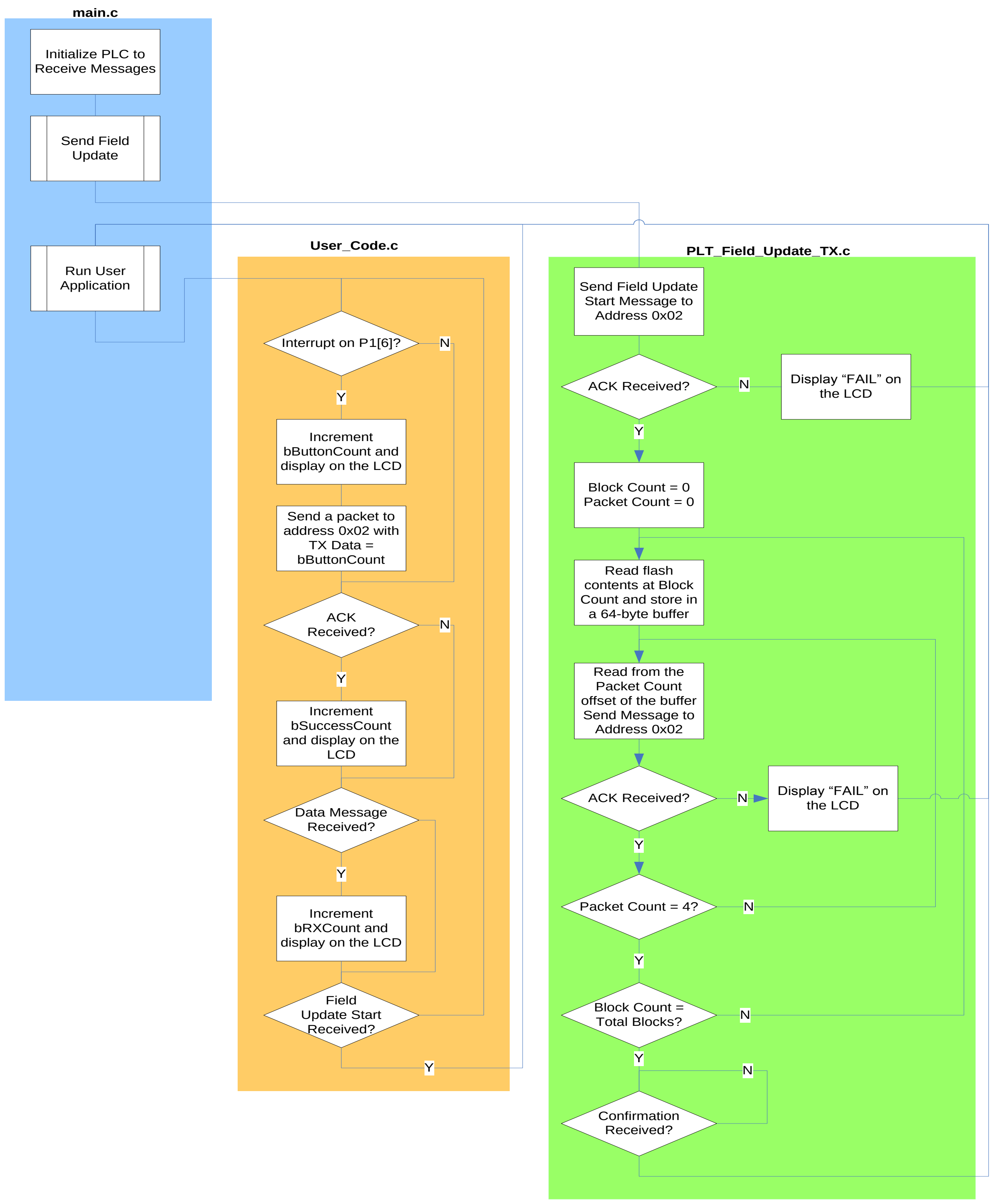
| Offset | Register Name | Access | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x01 | Local_LA_LSB | RW | **0x01** | | | | | | | |
| 0x05 | PLC_Mode | RW | TX_Enable = ‘**1**’ | RX_Enable = ‘**1**’ | | | | | | |
| 0x06 | TX_Message_Length | RW | Send_Message= ‘**1**’ | | | Payload_Length_MASK= ‘**10010**’ | | | | |
| 0x07 | TX_Config | RW | TX_SA_Type = ‘**0**’ (Logical) | TX_DA_Type = ‘**00**’ (Direct Logical) | | TX_Service _Type = ‘**1**’ (ACK) | TX_Retry = ‘**0011**’ | | | |
| 0x08 | TX_DA | RW | **0x02** | | | | | | | |
| 0x10 | TX_CommandID | RW | **0x51** | | | | | | | |
| 0x11 | TX_Data[0] | RW | Flash Block Number | | | | | | | |
| 0x12 | TX_Data[1] | RW | Packet Number (**0x00 – 0x03**) | | | | | | | |
| 0x13 – 0x23 | TX_Data[2] | RW | Flash Data (16  bytes) | | | | | | | |

Table 9. Receiver Memory Array Settings for Field Update Confirm

| Offset | Register Name | Access | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x40 | RX_Message_INFO | RW | New_RX_Msg = ‘**1**’ | RX_DA_Type =’**0’** (Direct) | RX_SA_Type = ‘**0**’ (Logical) | RX_Msg_Length = ‘**00000’** | | | | |
| 0x41 | RX_SA | R | Remote Node Source Address(8 Bytes) = **0x02** | | | | | | | |
| 0x49 | RX_CommandID | R | **0x52** | | | | | | | |
| 0x4a | RX_Data[0] | R | Flash Block Number | | | | | | | |
| 0x4b | RX_Data[1] | R | Packet Number (**0x00 – 0x03**) | | | | | | | |

A flow chart of the transmitter algorithm is shown in the following figure. Note that the code is separated across multiple files. The *User_Code.c* file contains the code that will be sent to the receiver for the field update. The code in the other files (including *main.c*) is not sent. The files not shown are *ISR_Handler.c* (which handles the GPIO interrupt and updates a global variable bInterrupted that is read in *User_Code.c*) and *PLC_Common.c* (which contains common PLC functions).

Figure 11: Transmitter Flow Chart


main.c
Initialize PLC to Receive Messages
Send Field Update
Run User Application
User_Code.c
Interrupt on P1[6]?
N
Y
Increment bButtonCount and display on the LCD
Send a packet to address 0x02 with TX Data = bButtonCount
ACK Received?
N
Y
Increment bSuccessCount and display on the LCD
Data Message Received?
Y
Increment bRXCount and display on the LCD
Field Update Start Received?
Y
PLT_Field_Update_TX.c
Send Field Update Start Message to Address 0x02
ACK Received?
N
Display "FAIL" on the LCD
Y
Block Count = 0 Packet Count = 0
Read flash contents at Block Count and store in a 64-byte buffer
Read from the Packet Count offset of the buffer Send Message to Address 0x02
ACK Received?
N
Display "FAIL" on the LCD
Y
Packet Count = 4?
N
Y
Block Count = Total Blocks?
N
Y
N
Confirmation Received?

## PSoC Designer Project

The transmitter project has the same configuration as shown in Figure 9. It is necessary that the configuration is the same and it is recommended that the transmitter project is cloned from the receiver project so that the code is allocated in the same location.

To perform the field update, the guidelines described in section 'Performing the Field Update' were followed. The flash and RAM is allocated in the same locations and the only differences are:

1. The *PLC_Field_Update_RX.c* (and *.h*) file is replaced with *PLC_Field_Update_TX.c* (and *.h*) so that the field update can be sent, not received
2. The *User_Main function in User_Code.c* is modified to have a different functionality (described above)
3. The constant string arrays in *User_Code.c* are modified for the new functionality shown by the LCD display.
4. The location (#pragma abs_address) of the constant string arrays was changed from 0x5740 to 0x57C0 because the code size of User_Main and User_Init increased. The end of the area my_code_area changed from 0x5732 to 0x57A2, which would have caused a conflict with the strings.
5. The code in *main.c* is modified so that the field update can be performed.
6. The new variables in *User_Code.c* are declared in myData.asm
7. The `USER_CODE_SIZE_BLOCKS` parameter in *User_Code.h* was set to 8 based on the code size:
    a. start = 0x5600 (my_ISR_area),
    b. end = 0x57E0 (end of abLCDText1)
    c. # of blocks = (0x57E0 – 0x5600) / 64 = 7.5

## Implementation on Hardware

This section describes how to implement the transmitter project in hardware. A second board is needed for evaluating this project.

### PLC Transmitter Hardware

The PLC transmitter project is designed to run on the CY3274 High Voltage PLC Development Kit. Follow these steps to set up the system:

1. Connect a jumper wire from SW to P1[6].
2. Connect the LCD daughter card to the LCD connector.
3. Connect the board to the powerline. The blue power LED should turn ON.
4. Program the firmware with the MiniProg in Reset mode.
5. Disconnect the programmer and reset the board.
6. Observe the transmitter and receiver LCDs. The following sequence should occur:
    - The transmitter will display ("Send Update XXYY", where XX represents the flash block number and YY represents the packet number for that block. Recall that each 64-byte block is broken into four 16-byte data packets. Figure 12 shows what the transmitter's LCD will display when it has sent packet 2 of flash block 1. Figure 13 shows what the receiver's LCD will display for the same packet. The receiver's LCD also displays how many total flash blocks that it expects receive. In this case, it is 8.
    - If the receiver is not within range of the transmitter, then the transmitter's LCD will display "Fail" as shown in Figure 14.
    - If the field update was a success, then the transmitter and receiver will both display "Application 2" as shown in Figure 15.
    - To test out the update, press the pushbutton SW on both of the boards. If communication is successful, then both boards will show that 1 packet was acknowledged out of 1 transmitted ("TX =01/01") and 1 packet was received ("RX = 01") as shown in Figure 16.

Figure 12. Transmitter Field Update Display

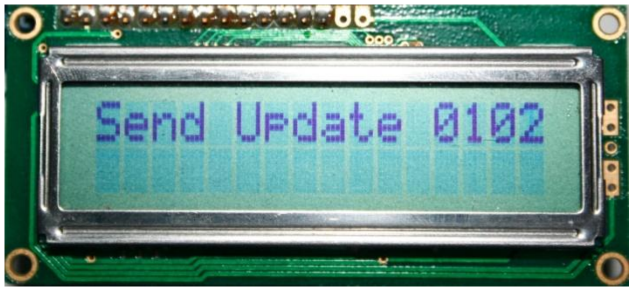


Figure 13. Receiver Field Update Display

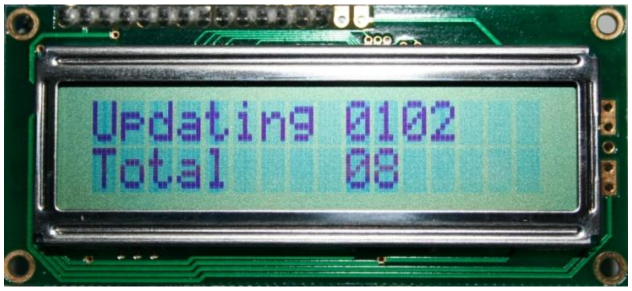


Figure 14. Transmitter Field Update Failure Display

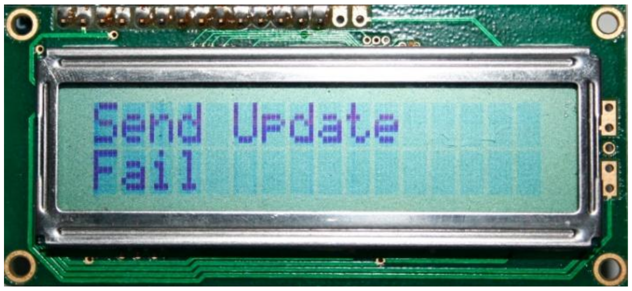


Figure 15. Transmitter and Receiver Successful Update Display

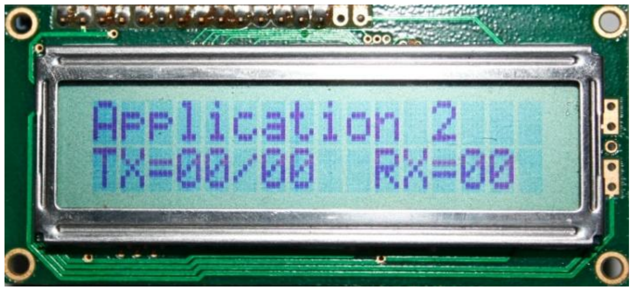
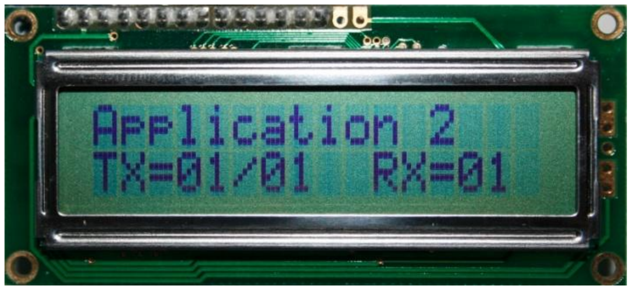


Figure 16. Application 2 Packet Transmit and Receive



## Document History

Document Title: Updating Field Firmware with PLC - AN62792

Document Number: 001-62792

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | 2969819 | FRE | 07/05/10 | New Application Note. |
| *A | 3177822 | FRE | 02/20/2011 | Updated Software Version as PSoC® Designer™ 5.1 SP1<br>Updated Allocating Code in Flash and RAM.<br>Updated Performing the Field Update.<br>Updated Receiver Project.<br>Updated Transmitter Project.<br>Updated PSoC Designer Project. |
| *B | 3379910 | ADIY | 9/22/2011 | Removed reference to CY8CLED16P01, CY3276 and CY3277.<br>Updated Figure 1 and Figure 2.<br>Updated template. |
| *C | 4542402 | ROIT | 10/17/2014 | Removed reference to CY3275 kit (obsolete) |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

### Products

| | |
|---|---|
| Automotive | cypress.com/go/automotive |
| Clocks & Buffers | cypress.com/go/clocks |
| Interface | cypress.com/go/interface |
| Lighting & Power Control | cypress.com/go/powerpsoc<br>cypress.com/go/plc |
| Memory | cypress.com/go/memory |
| PSoC | cypress.com/go/psoc |
| Touch Sensing | cypress.com/go/touch |
| USB Controllers | cypress.com/go/usb |
| Wireless/RF | cypress.com/go/wireless |

### PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

### Cypress Developer Community

Community | Forums | Blogs | Video | Training

### Technical Support

cypress.com/go/support

| | | |
|---|---|---|
| Cypress Semiconductor | Phone | : 408-943-2600 |
| 198 Champion Court | Fax | : 408-943-4730 |
| San Jose, CA 95134-1709 | Website | : www.cypress.com |