COG Driver Programming Guide

Description	**AdapTag Development Kit: COG Driver Programming Guide**
Date	**2012/12/14**
Doc. No.	**8P003-00**
Revision	**01**
Model Name	**S0000AF1T1**

	Design Engineering		
	Approval	**Check**	**Design**
	李 2012.12.14 欣達	蘇 2012.12.14 倉民	劉 2012.12.14 佩柔

龍亭新技股份有限公司 Pervasive Displays, Inc.
No.71, Delun Rd., Rende Dist., Tainan City 71743, Taiwan (R.O.C.)
Tel: +886-6-279-5399 / Fax: +886-6-270-5857 / http://www.pervasivedisplays.com

Table of Contents

Revision History **4**
1. Introduction **5**
1.1 Overview 5
1.2 Kit Contents 5
1.3 System Diagram 6
1.4 The Project Code of AdapTag 6
1.4.1 Development Tools 6
1.4.2 Folder Structure 7
1.4.3 File Explanation 7
2. Preparing Image Data **9**
2.1 Preparing Image Packets 9
2.2 Bitmap Header 9
2.3 Image Data 11
2.4 Code Explanation 12
2.5 Insert Image Header 13
2.6 The Memory Allocation 14
2.6.1 The Memory Map in SRAM and Flash 14
2.6.2 The Steps of Storing Image Data 15
3. COG Timing Interface **17**
3.1 File Explanation 17
3.2 Code Explanation & Driving Flowchart 17
3.2.1 COG_Parameters Structure 19
3.2.2 ePaper Driving Process 19
3.2.3 Write to the Memory 19
3.2.4 Initialize Hardware 20
3.2.5 Power On 21
3.2.6 SPI command 22
3.2.7 Initialize Driver 23
3.2.8 Temperature Factor 25
3.2.9 Display 26
3.2.10 Power Off 31
3.2.11 Save Image to Flash 32
3.3 Project Configurations 33

4. REFERENCES 36
Glossary of Acronyms 37

Revision History

Version	Date	Page (New)	Section	Description
01	2012/12/14	All	All	First draft

1. Introduction

1.1 Overview

The AdapTag Development kit provides a working reference design for software development of sub 1GHz RF applications based on the TI CC430 MCU and the 1.44", 2" and 2.7" EPDs from Pervasive Displays. The AdapTag development kit consists of one AdapTag host connected to an EZ430 USB dongle and three AdapTag slave boards each connected to an EPD.

The purpose of this document is to give an overview of the AdapTag Development kit. This document also serves as an introduction to Pervasive Displays Inc. (PDI) EPD driving technology to explain the program code of COG driver to operate the EPD for TI CC430 MCU based solution. It applies to PDi's 1.44", 2", and 2.7" EPDs. All of the source codes are licensed under the Apache License, Version 2.0 (the "License"). You may not use the code except in compliance with the License.

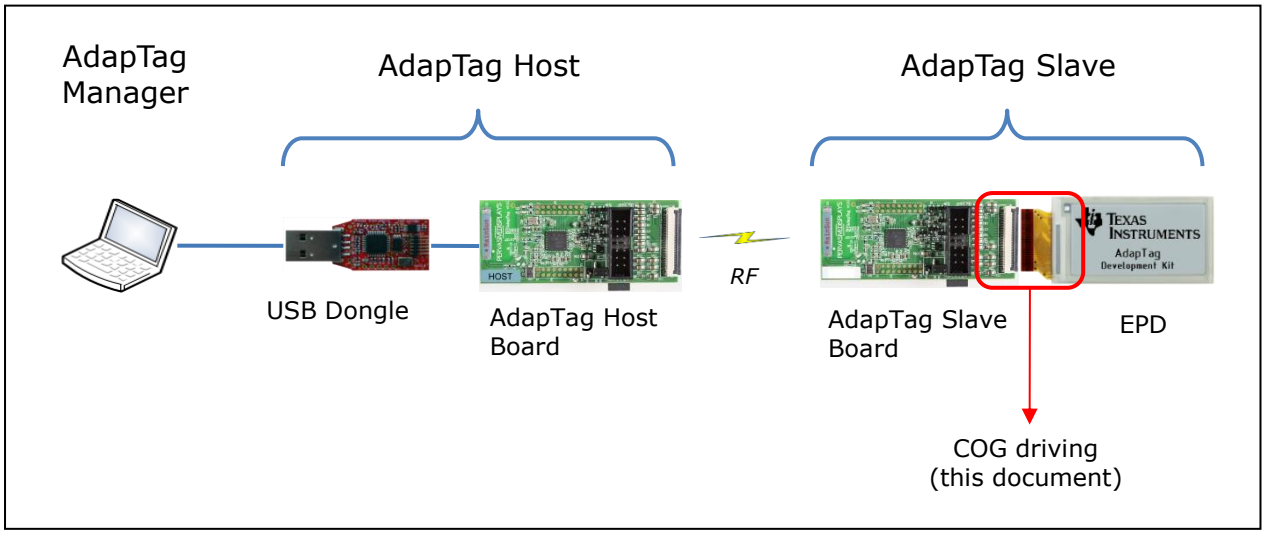
For more introductions on the COG driver, please refer to PDi's "E-paper Display COG Driver Interface Timing" which document number is 4P008-00 [9].

1.2 Kit Contents

Name	Photograph	Note
AdapTag Host board		1 piece. marked with blue label "HOST"
AdapTag Slave board		3 pieces. marked with white labels with 4 digits number (Slave ID) "1401", "2001" and "2701"
eZ430 USB Dongle		1 piece.
CR2450 Coin Cell Battery		3 pieces.

1.44", 2" and 2.7" EPD		3 pieces. 1 piece of each panel size
AdapTag Manager installer	Install AdapTag Manager software working with AdapTag kit	

1.3 System Diagram



1.4 The Project Code of AdapTag

1.4.1 Development Tools

[1] AdapTag Manager:
Developed by Microsoft® Visual Studio® 2010

AdapTag Manager is the GUI working with AdapTag Development Kit. It is also defined System Packet and functions that communicates with AdapTag Host.

[2] AdapTag Host & Slave:
Developed by Texas Instruments® Code Composer Studio™ (CCStudio) 5.0
The firmware codes are programmed in AdapTag Host and Slave board. They are also defined the Protocol Packet and functions that communicates between AdapTag Host and Slave.

1.4.2 Folder Structure

Switch to the project folder in CCStudio. You will find the directory structure as below:

Folder	Description
AdapTag Host	The project folder of Host
AdapTag Slave	The project folder of Slave
Common	Most of the project files are located here. (The project has defined "Predefined Symbols" to differentiate between host and slave in the codes.)
– ALPS Library	ALPS functions, parameters and structures
– DataHandle	CC430 UART driver and interface
– ePaper	COG driving waveform, process and driver
– General	Central control process and general functions
– Memory	Memory control, allocation and flash/ram driver
Library	The ALPS compiled libraries: AdapTagHostLibrary.lib and AdapTagSlaveLibrary.lib

1.4.3 File Explanation

In the "ePaper" folder, the COG driving codes are divided into three different functions which are **driver**, **process** and **controller**.

- **Display_Controller**.c:
  The main point to start process for initialing I/O, driving COG and storing image to memory. It provides overall steps matching the COG driving flow from power on, initialize driver, display image and power off. Developer is able to call the main function (e.g. `epaper_disp(EPD_Type, Previous_Image, New_Image)` for a completed display cycle.
- **Display_COG_Process**.c:
  This file consists of the functions for updating the EPD. Each function implements the step for COG driving flow. It also provides most of the EPD parameters to

separate difference sizes allowing developers easier to adjust initial parameter, resolution, frame time and so on.

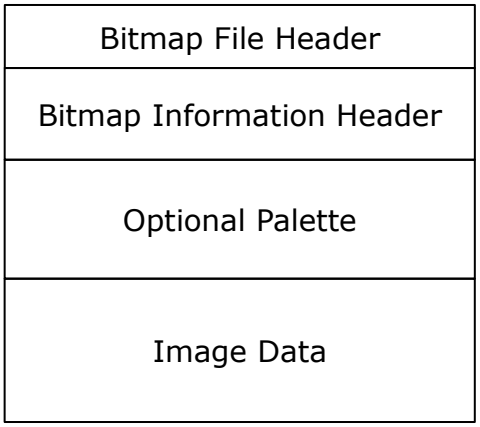
- **Display_Hardware_Driver**.c:
  This file includes the configuration and initialization of PWM, SPI and other I/Os. The SPI command and format are also defined here. The .h file defines meaningful I/O variables.

2. Preparing Image Data

The image pattern to COG supports 1 bit Bitmap (BMP) image format (Black/White) only. This section introduces how to structure and prepare the image data for COG to load on EPD.

2.1 Preparing Image Packets

A BMP image file consists of either 3 or 4 parts as shown in the diagram below. The first part is a "Bitmap File Header"; this is followed by a "Bitmap Information Header". If the image is indexed color then the "Palette" follows, and last of all is the "Image Data". The position of the image data with respect to the start of the file is contained in the header. Information such as the image width and height, the type of compression, the number of colors is contained in the information header.



The following image data will use this 2 inch 1 bit bitmap image file as example.

2.2 Bitmap Header

✧ **File header**: 14 bytes

This block of bytes is at the start of the file and is used to identify the file. A typical application reads this block first to ensure that the file is actually a BMP file and that it is not damaged. The first two bytes of the BMP file format are the character 'B' (0x42) then the character 'M' (0x4D) in 1-byte ASCII encoding.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000h:	42	4D	BE	0A	00	00	00	00	00	00	3E	00	00	00	28	00
00000010h:	00	00	C8	00	00	00	60	00	00	00	01	00	01	00	00	00

Offset	Size (bytes)	Purpose	Example
0000h	2	Signature	0x42='B', 0x4D='M'
0002h	4	File size	0xABE=2,750 bytes
0006h	2	Reserved	
0008h	2	Reserved	
000Ah	4	File offset to pixel array	0x3E=62 bytes, starting address, of the byte where the bitmap image data (pixel array) can be found. If the image is one bit bitmap file, the starting address is always from 0x3E.

✧ **Information Header**: 40 bytes

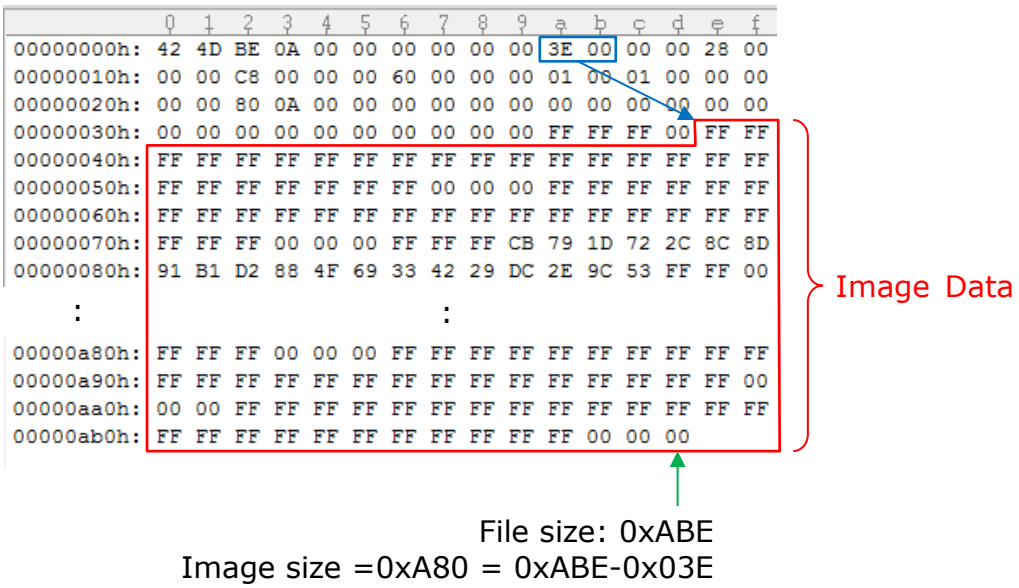
This block of bytes tells the application detailed information about the image, which will be used to display the image on the screen. The image info data that follows is 40 bytes in length. It is described in the structure given below. The fields of most interest below are the image width and height, the number of bits per pixel (should be 1, 4, 8 or 24), the number of planes (assumed to be 1 here), and the compression type (assumed to be 0 here).

```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 42 4D BE 0A 00 00 00 00 00 00 3E 00 00 00 28 00
00000010h: 00 00 C8 00 00 00 60 00 00 00 01 00 01 00 00 00
00000020h: 00 00 80 0A 00 00 00 00 00 00 00 00 00 00 00 00
00000030h: 00 00 00 00 00 00 00 00 00 00 FF FF FF 00 FF FF
00000040h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000050h: FF FF FF FF FF FF FF 00 00 00 FF FF FF FF FF FF
00000060h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

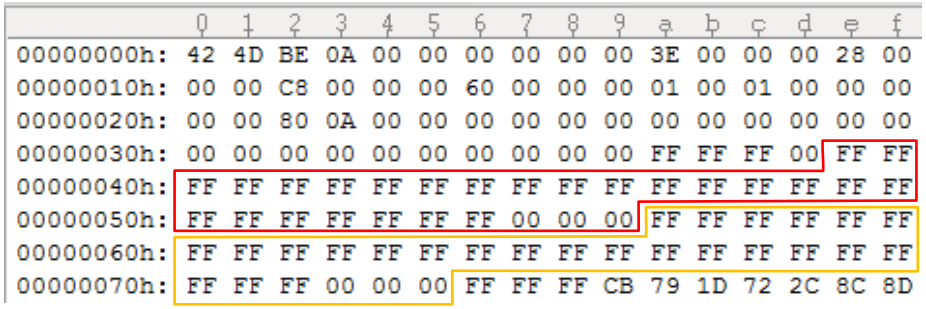
Offset	Size (bytes)	Purpose	Example
000Eh	4	Size of header	0x28=40 bytes
0012h	4	Bitmap width in pixels	0xC8=200 bytes=200 pixels
0016h	4	Bitmap height in pixels	0x60=96 bytes=96 pixels
001Ah	2	Number of color planes	0x01=1
001Ch	2	Number of bits per pixel, which is the color depth of the image	0x01=1
001Eh	4	Compression method	0, means no compression
0022h	4	Image size, the size of the image data	0x0A80=2,688 bytes
For more information of the others, please refer to further introduction of Bitmap file.			

2.3 Image Data

The image data that EPD needs is to remove the file header and information header.



The bits representing the bitmap pixels are packed in rows. The size of each row is rounded up to a multiple of 4 bytes (a 32-bit DWORD) by padding. The pixel number of 2 inch EPD is 200 by 96. 200 divided by 8 bit equals 25 bytes which means one Line on EPD needs 25 bytes image data. Padding bytes (not necessarily 0) will be appended to the end of the rows in order to bring up the length of the rows to a multiple of four bytes for 1 bit bitmap image file. The size of each row must be a multiple of 4 bytes, so you can find a row uses 28 bytes (see below figure).



The last three bytes are padding bytes which can be ignored.

The pixel of number of 1.44 inch EPD is 128 by 96. 128 divided by 8 equals 16 which is already a multiple of 4 bytes. There is no padding byte in row.

The pixel of number of 2.7 inch EPD is 264 by 176. 264 divided by 8 equals 33. There are 3 padding bytes.

2.4 Code Explanation

In "AdapTag_API.cs" file of [1], the `ConvertImageTo1bit` function supports converting source image file to 1 bit BMP file format by assigned EPD size.

```
public static Bitmap ConvertImageTo1bit(string ImageFilePath, string saveFilePath,
enumPanelSize Size)
```

For how to get the image data that just introduced above, the `GenSendBytes` function provides the approach to generate the image data that COG needs. The bitmap image file (the `BMPfile` below) can be converted by "ConvertImageTo1bit" function in advance.

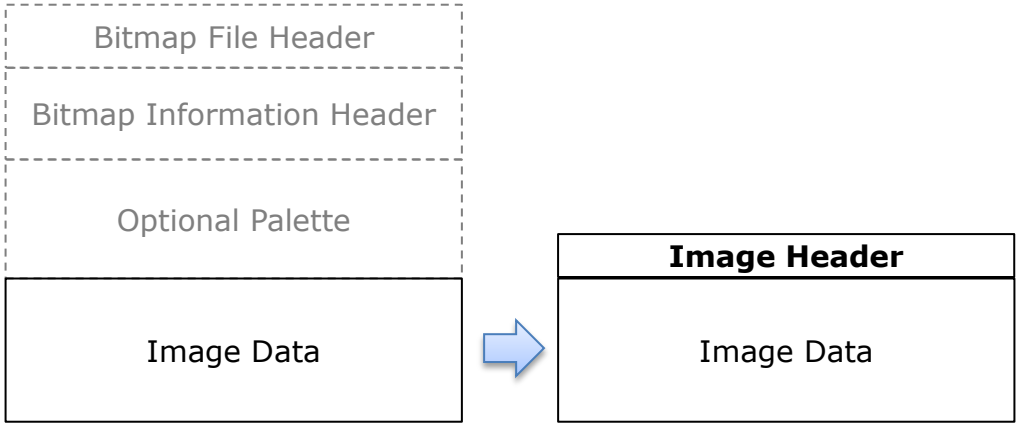
```
private byte[] GenSendBytes(string BMPfile)
{       //The file path of BMPfile must been one bit bitmap image format already
        int Real_H_Res = 0, Real_V_Res = 0;
        byte[] content;       //the byte array to store image data
        byte[] tmp;
        const byte BM_Header = 0x3e;//0x3e is the starting address of image data
        int BytesPerLine = 0;
        List<byte> res = new List<byte>(); //res List to store image data without padding bytes
        try
        {
                //Get the resolution from this bitmap
                Bitmap b = new Bitmap(ReadImage(BMPfile));
                Real_H_Res = b.Width;
                Real_V_Res = b.Height;
                b.Dispose();

                //Import the bitmap file to byte array
                FileStream fs = new FileStream(BMPfile, FileMode.Open, FileAccess.Read,
FileShare.ReadWrite, 8046);

                fs.Seek((long)BM_Header, SeekOrigin.Begin);
                content = new byte[fs.Length - BM_Header];//the size of content array is the file size
minus header size
                fs.Read(content, 0, content.Length);
                fs.Close();

                BytesPerLine = Real_H_Res / 8;      //200x96; 200/8=25 bytes per line
                int LineNumMod = (4 - BytesPerLine % 4) % 4;      // The size of each line must be a
multiple of 4 bytes

                int nIndex = 0;
                for (int i = 0; i < Real_V_Res; i++)
                {
                        for (int j = 0; j < BytesPerLine; j++)
                        {
                                res.Add(content[nIndex]);
                                nIndex++;
                        }
                        nIndex += LineNumMod;//Ignore padding bytes
                }
                tmp = new byte[Real_V_Res * BytesPerLine];//Image data List
                for (int j = 0; j < res.Count; j++)
                {
                        tmp[j] = (byte)res[j];       //List convert to byte array
                }
        }
        catch (Exception exp) {if (OnErrorHandler != null)OnErrorHandler("GenSendBytes :" +
exp.Message); }
        return tmp;
}
```

2.5 Insert Image Header

When sending image data packets to host from AdapTag Manager, the image packet will insert 8 bytes image header at the beginning of Image Data included Slave ID and panel size information in order to have the target slave updating on correct EPD content. The 8 bytes of image header are:

Byte 0, 1	Byte 2	Byte 3	Byte 4, 5	Byte 6, 7
Slave ID	State	Panel Type	Horizontal	Vertical

The value of byte 2 "State" is 0xFF. It means new image. When the image is updated onto EPD display, this byte in the flash memory will be changed to 0xAA. For more information on the memory allocation, please refer to section 2.6.



Bitmap file structure

The image header structure is defined at AdapTag_API.cs file in [1].

```
public struct structImageInfo
{
        public UInt16 SlaveID;
        public byte State;
        public byte PanelType;
        public UInt16 HORIZONTAL;
        public UInt16 VERTICAL;
}
```

The source code of inserting image header is at the end of "SendData" function in AdapTag_API.cs file.

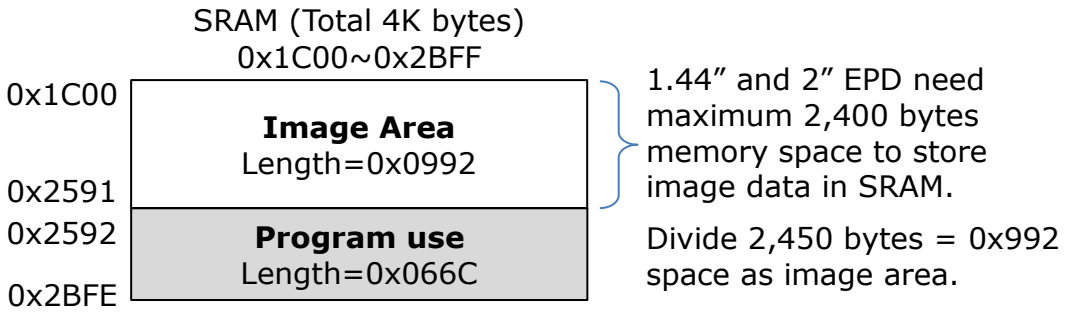
```
void SendData(UInt16 Address, byte cmd, byte[] payload, bool Start)
```

2.6 The Memory Allocation

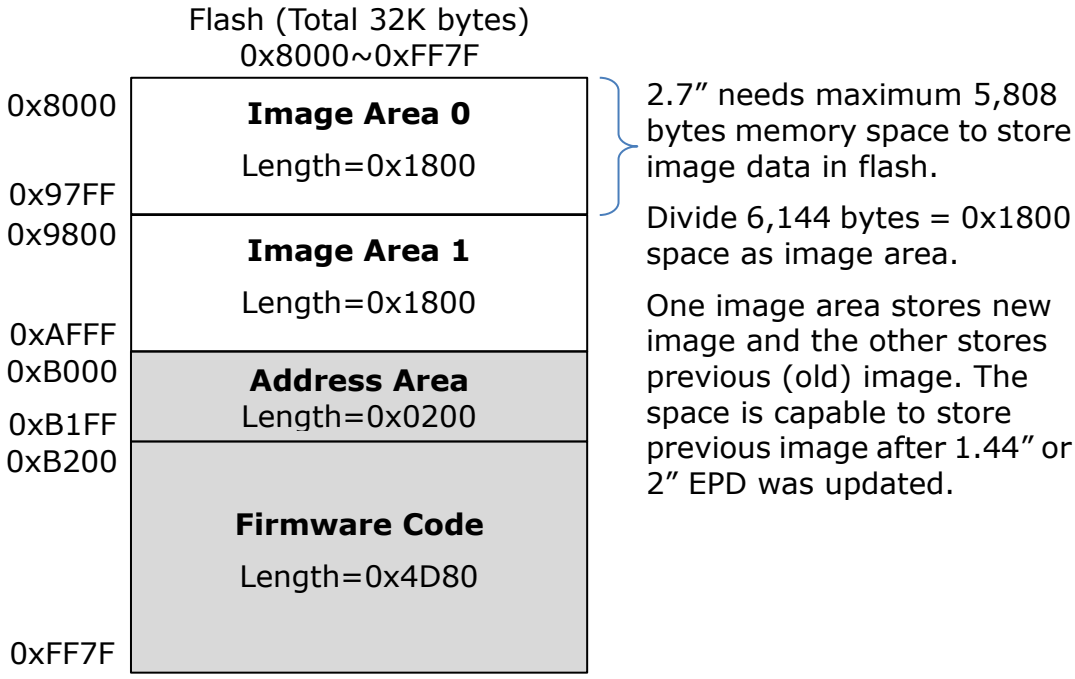
This section describes how this kit to store image data in memory. The CC430F5137 MCU has built-in 4K bytes SRAM and 32K bytes flash memory. The image data size of 1.44" EPD is 128 * 96 = 1,536 bytes and 2" EPD is 200 * 96 = 2,400 bytes. Both of 1.44" and 2" EPD are able to use internal 4K bytes SRAM to store new image data from radio. However, the data size of 2.7" is 264 * 176 = 5,808 bytes causes the built-in SRAM is unable to store a 2.7" image data, so 2.7" will use flash to store new image data.

2.6.1 The Memory Map in SRAM and Flash

✧ **SRAM**

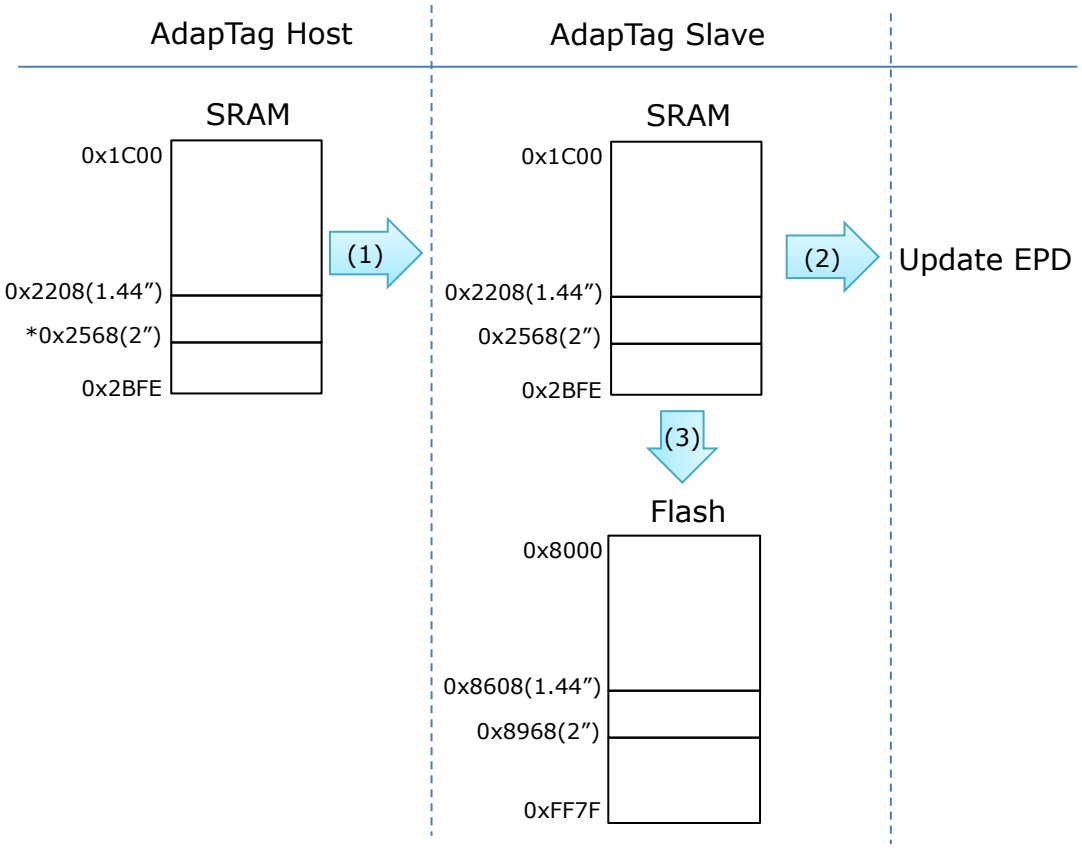


✧ **Flash**



2.6.2 The Steps of Storing Image Data

✧ **1.44” or 2”**



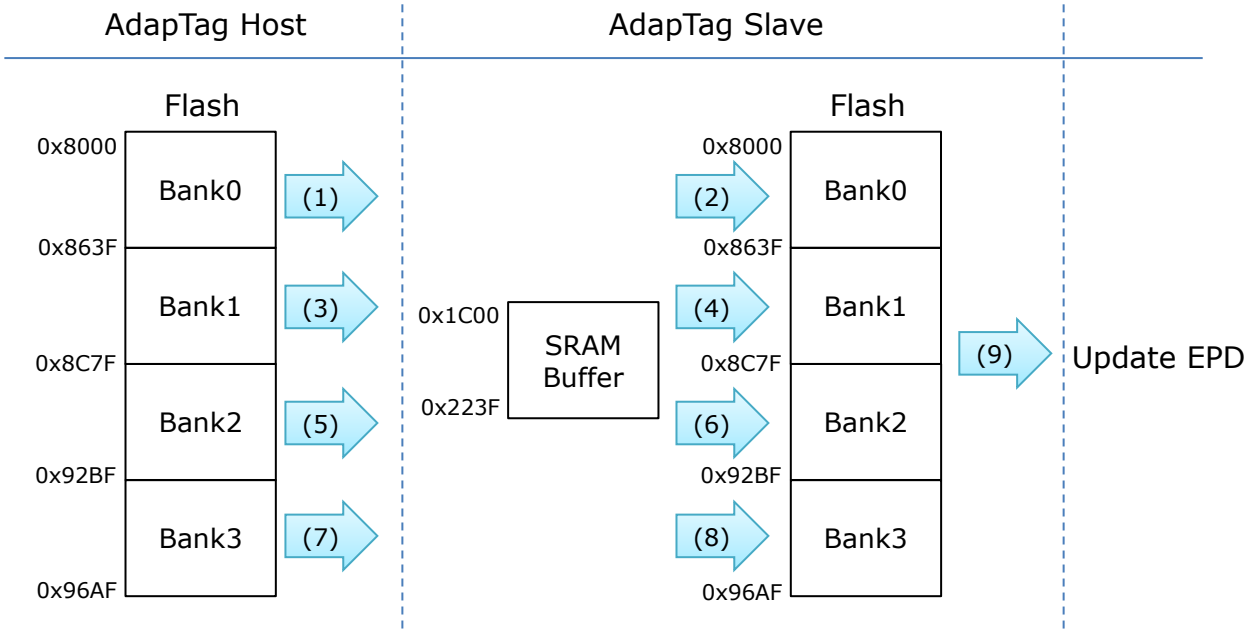
Note:

For 2” example: (200*96)/8 bit=0x960 bytes. 0x1C00 + 0x960 + **8** = 0x2568

The **8** bytes is added image header bytes.

The “State” byte in image header is oxFF (`ImageState_EMPTY` in MemoryMap.h of [2]) before updating on EPD display. After updating EPD, the image data needs to store in the flash memory for comparing with new image for compensate. The “State” byte will be changed to 0xAA (`ImageState_Action` in MemoryMap.h of [2]) which means it is old image.

✧ **2.7”**



The image data also has added the 8 bytes image header. The “State” byte in image header is oxFF (`ImageState_EMPTY`) before updating on EPD display. A 2.7” image data will be divided into four parts (bank0~3) for transmission. At the slave side, it uses SRAM memory as buffer to quick store image and then saves to flash memory. The new image data will be stored in either image area 0 or image area 1 depends on which area was cleared.

For PDi’s COG driving process [9], the update stage needs to compare with previous (old) image. The slave will determine where the “State” byte in image area 0 or area 1 is 0xAA (`ImageState_Action`) and run the COG updating stages and then proceed to update EPD display. After updating image on EPD display, the slave will clear the previous image area in order to have the new image to store here later. The slave will also change the “State” byte in current image area to 0xAA from 0xFF.

For the steps above, find the sample code `Save_Image()` in Display_Controller.c [2]. The function of changing “State” byte is `GetUseImageInfo` in ALPS_USE.c [2].

3. COG Timing Interface

This section provides an immediate implementation driving approach. It follows PDi's COG driving flow in the waveform document [9] and also gives sample source code to get Temperature and Voltage value of CC430.

The COG driving code in [2] is able to drive three different EPD sizes (1.44", 2" and 2.7") according to the `EPDType` enumeration. It has been divided into three files implements different function.

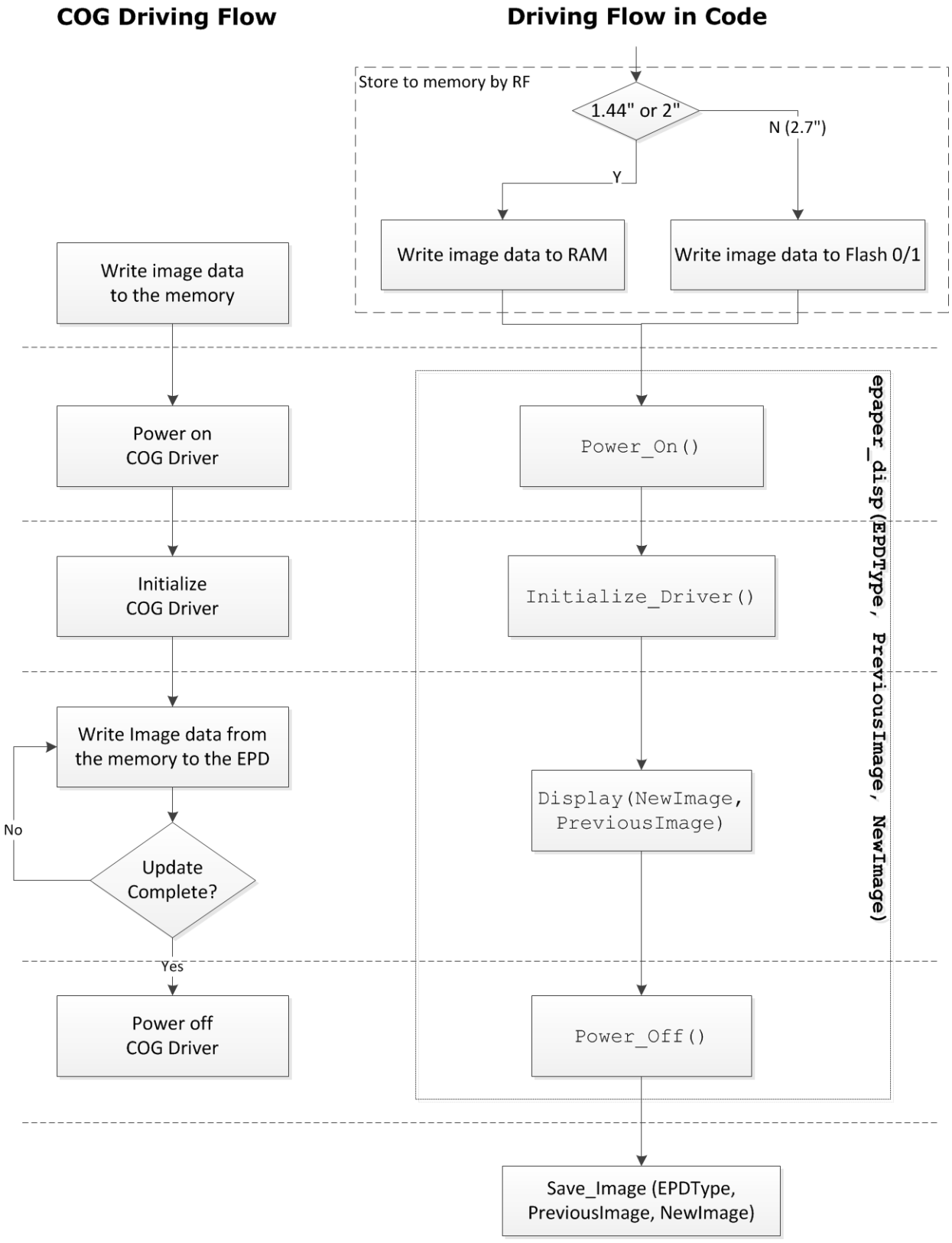
3.1 File Explanation

The three files below are in ePaper directory of [2].

- Display_Hardware_Driver:
  Most of the initialization and configuration are put at here. User can implement the driver layer of EPD if some variables need to adjust. The provided settings and functions are SPI configuration and initialization, I/O control, PWM configuration and EPD hardware initialization.
- Display_COG_Process:
  This file presents the waveform driving processes and updating stages. It is about the COG driving waveform mostly. The parameters of driving different EPD is defined at `COG_Parameters_t` structure in Display_COG_Process.h file which is easy for developer adjusting initial parameters, resolution, frame time of MCU and the size of data line.
- Display_Controller:
  The interface of Display Unit for external function wants to update EPD. `epaper_disp(EPDType, PreviousImage, NewImage)` is the main function for COG driving process.

3.2 Code Explanation & Driving Flowchart

Please request the COG Timing Interface document [9] from PDi for more details to compare the following functions and flowchart. The left flowchart below provides an overview of the actions necessary to update the EPD in the COG Driver Timing Interface document [9]. The right flowchart represents an overview of function flow to match with COG driving flow in [2].



3.2.1 COG_Parameters Structure

This structure is to define the size difference of 1.44", 2" and 2.7".

`ChannelSelect[8];`	The different SPI Channel Select at "Initialize COG Driver" stage
`VoltageLevel;`	The different SPI Gate and Source Voltage Level at "Initialize COG Driver" stage
`HORIZONTAL;`	The number of dots in a line. Image width.
`VERTICAL;`	The number of lines of image. Image height.
`DataLineSize;`	Even + Scan + Odd bytes
`FrameTime;`	Frame time of the COG
`StageTime;`	Stage Time of the COG

3.2.2 ePaper Driving Process

(Find the code below in Display_Controller.c [2])

```
void epaper_disp (uint8_t EPDType, unsigned char *PreviousImage,unsigned char *NewImage )
{
       LED_Red_OFF();      //Turn off the LEDs
       LED_Green_OFF();
       ChangeFreq(EPDType);       //Set the frequency by EPD type
       Hardware_Init();    //Initial display hardware
       Power_On ();
       Initialize_Driver (EPDType);
       Display ((NewImage+Image_INFO),(PreviousImage+Image_INFO) );
       Power_Off ();

       if (NewImage != PreviousImage)
       {
              Save_Image (EPDType,PreviousImage,NewImage);
       }

       ChangeFreq(EPD_144);       //Set the frequency to 1.44" for saving energy
    }
```

3.2.3 Write to the Memory

See Section 2 in [9].

Please refer to section 2.1 to 0 in this document how to preparing image data and write to memory.

3.2.4 Initialize Hardware

See Section 3.1 in [9].

According to the "3.Power On COG Driver" stage [9], some initial state as follows: VCC/VDD, /RESET, /CS, /Border, SI, SCLK = 0. The `Hardware_Init()` represents the initial states above.

Display_Controller.c	Display_COG_Process.c	Display_hardware_Driver.c
`epaper_disp()`	`Hardware_Init()`	`Init_DisplayHardware()`
• COG driving flow	• Set time slot clock • Call driver layer `Init_DisplayHardware()`	• SPI initialization • VCC/VDD, /RESET, /CS, /Border, SI, SCLK = 0 • Temperature initialization • See the sample code below

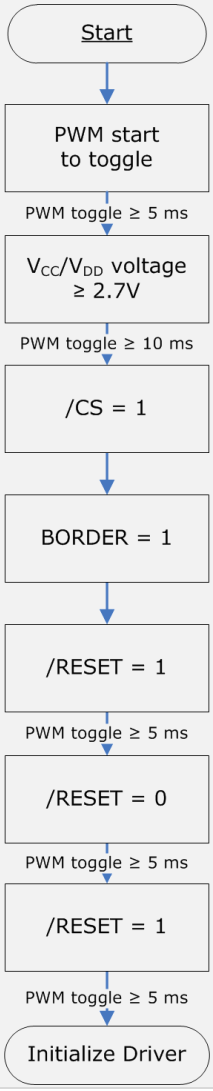
```
void Init_DisplayHardware (void)
{
	BITSET (PWM_PORT_DIR, PWM);
	BITSET (CS_PORT_DIR, CS);
	BITSET (RST_PORT_DIR, RST);
	BITSET (DISCHARGE_DIR, DISCHARGE);
	BITSET (PANELON_PORT_DIR, PANELON);
	BITCLR (DRIVERBSY_PORT_DIR, DRIVERBSY);
	BITSET (BORDER_PORT_DIR, PANELON);

	panelon_off();
	spi_init ();
	spi_detach ();
	init_epaper_pwm ();
	cs_low();
	pwm_low();
	rst_low();
	discharge_low();
	border_low();
	TemperatureInit();

}
```

3.2.5 Power On

See Section 3 in [9].

(Find the code below in Display_COG_Process.c [2])

```
void Power_On (void)
{
        discharge_low ();
        rst_low ();
        cs_low ();
        spi_attach ();

        pwm_high ();
        Delay (_5ms);
        pwm_low ();

        panelon_on ();

        pwm_high ();
        Delay (_10ms);
        pwm_low ();

        cs_high ();

        border_high();

        rst_high ();

        pwm_high ();
        Delay (_5ms);
        pwm_low ();

        rst_low ();

        pwm_high ();
        Delay (_5ms);
        pwm_low ();

        rst_high ();

        pwm_high ();
        Delay (_5ms);
        pwm_low ();
}
```

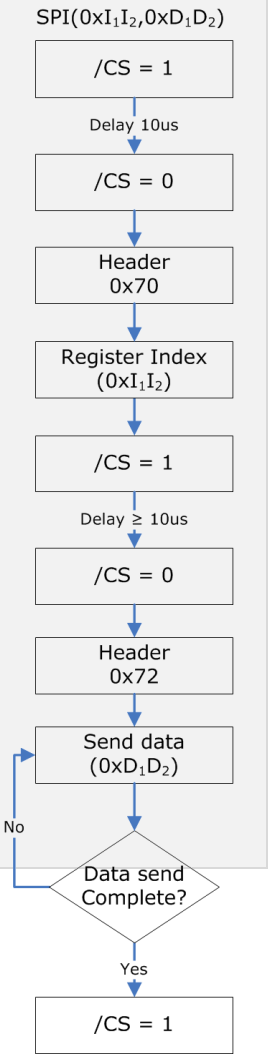


3.2.6 SPI command

See Section 1.6 in [9].

(Find the code below in Display_Hardware_Driver.c [2])

```
void SPI_Send (unsigned char Register, unsigned
char *Data, unsigned Length)
{
       cs_low ();
       SPI_Write (0x70);
       SPI_Write (Register);
       while ((UCA0STAT & UCBUSY))
       {
       }
       cs_high ();
       Wait_10us ();
       cs_low ();

       // setup dma
       __data16_write_addr((unsigned short) &DMA1SA,
(unsigned long) Data);
       DMA1SZ = Length;
       BITSET (DMA1CTL,(DMAEN));

       SPI_Write (0x72);
       while ((DMA1CTL & DMAIFG) == 0)
       {
       }
       BITCLR (DMA1CTL, DMAIFG);
       DMA1SZ = 0;
       while ((UCA0STAT & UCBUSY))
       {
       }
       cs_high ();
       Wait_10us ();
}
```



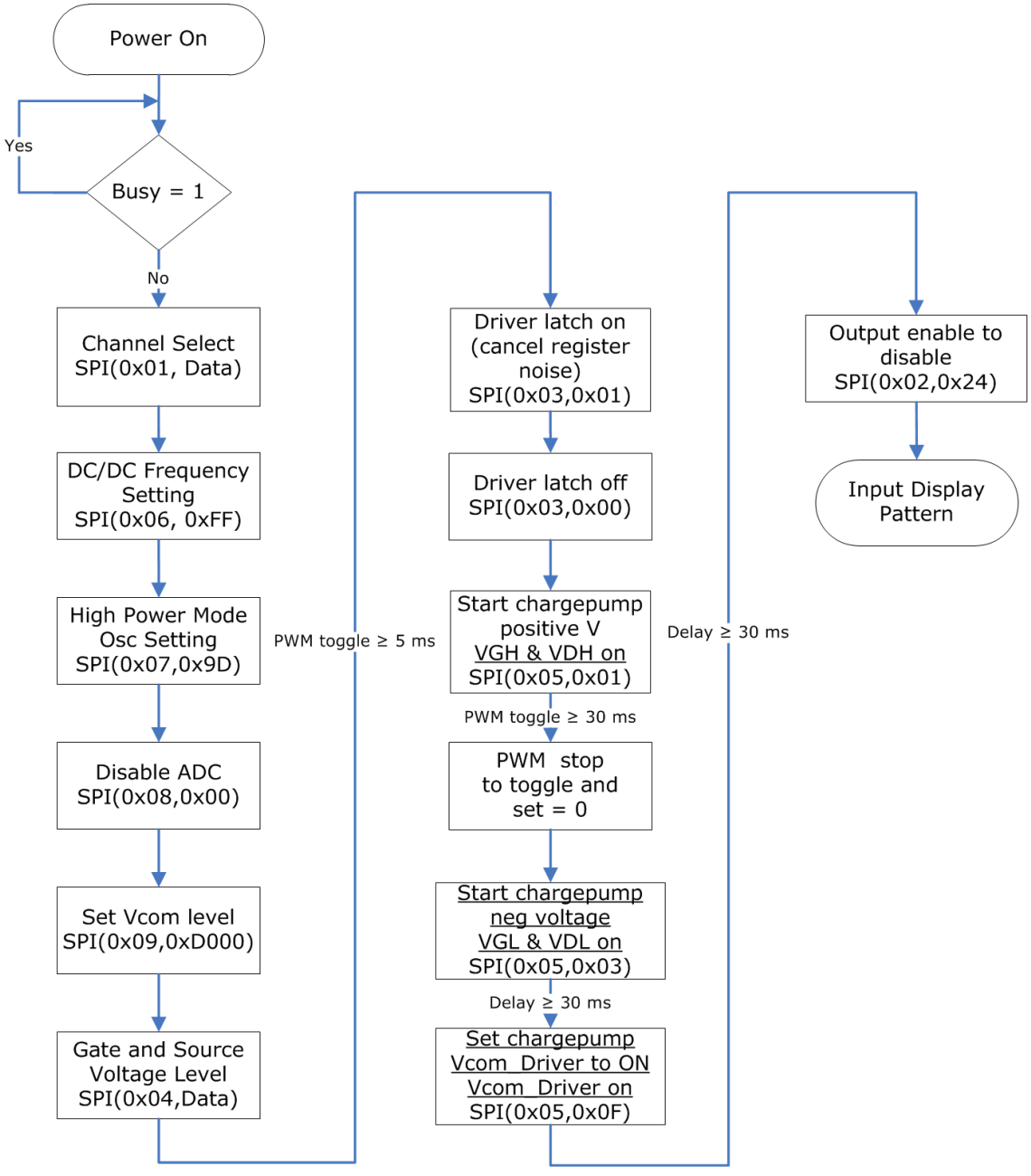
3.2.7 Initialize Driver

See Section 4 in [9].

(Find the code below in Display_COG_Process.c [2])

```
void Initialize_Driver (uint8_t EPDIndex)
{
	uint8_t SendBuffer[2];
	uint16_t	k;
	EPD_Type_Index=EPDIndex;
	//Data Line Clear
	for (k = 0; k <= __LineDataSize; k ++)
	{
		COG_Line.uint8[k] = 0x00;
	}
	Driver_TypeSelect(EPDIndex);
	k=0;
	//Temperature
	SetTemperature_Factor(EPDIndex);
	while (DRIVERBSY_IS_HIGH)
	{
		if((k++)>=0xFFF) return;
	}

	// SPI (0x01, 0x0000, 0x0000, 0x01ff, 0xe000)
	SPI_Send (0x01, (uint8_t *)&COG_Parameters[EPDIndex]. ChannelSelect, 8);

	// SPI (0x06, 0xff)
	SPI_Send_Byte (0x06, 0xff);

	// SPI (0x07, 0x9d)
	SPI_Send_Byte (0x07, 0x9d);

	// SPI (0x08, 0x00)
	SPI_Send_Byte (0x08, 0x00);

	// SPI (0x09, 0xd000)
	SendBuffer[0] = 0xd0;
	SendBuffer[1] = 0x00;
	SPI_Send (0x09, SendBuffer, 2);

	// SPI (0x04, 0x03)
	SPI_Send_Byte (0x04,COG_Parameters[EPDIndex]. VoltageLevel);

	// SPI (0x03, 0x01)
	SPI_Send_Byte (0x03, 0x01);

	// SPI (0x03, 0x00)
	SPI_Send_Byte (0x03, 0x00);

	pwm_high ();
	Delay (_5ms);
	pwm_low ();

	// SPI (0x05, 0x01)
	SPI_Send_Byte (0x05, 0x01);

	pwm_high ();
	Delay (_30ms);
	pwm_low ();

	// SPI (0x05, 0x03)
	SPI_Send_Byte (0x05, 0x03);
```

```
        Delay (_30ms);

        // SPI (0x05, 0x0f)
        SPI_Send_Byte (0x05, 0x0f);

        Delay (_30ms);

        // SPI (0x02, 0x24)
        SPI_Send_Byte (0x02, 0x24);
}
```



3.2.8 Temperature Factor

See Section 5.3 in [9].

(Find the code below in Display_COG_Process.c [2])

```
void SetTemperature_Factor(uint8_t EPD_Type_Index)
{
	int8_t Temperature;

	Temperature=GetTemperature();
	if (Temperature < -10)
	{
		StageTime =GetTime(COG_Parameters[EPD_Type_Index].StageTime) * 17;
	}
	else if (Temperature < -5)
	{
		StageTime =GetTime(COG_Parameters[EPD_Type_Index].StageTime) * 12;
	}
	else if (Temperature < 5)
	{
		StageTime =GetTime(COG_Parameters[EPD_Type_Index].StageTime) * 8;
	}
	else if (Temperature < 10)
	{
		StageTime = GetTime(COG_Parameters[EPD_Type_Index].StageTime) * 4;
	}
	else if (Temperature < 15)
	{
		StageTime =GetTime(COG_Parameters[EPD_Type_Index].StageTime) * 3;
	}
	else if (Temperature < 20)
	{
		StageTime =GetTime(COG_Parameters[EPD_Type_Index].StageTime) * 2;
	}
	else if (Temperature < 40)
	{
		StageTime = GetTime(COG_Parameters[EPD_Type_Index].StageTime) * 1;
	}
	else
	{
		StageTime =GetTime(COG_Parameters[EPD_Type_Index].StageTime) * 7)/10;
	}
}
```

Temperature (℃)	TF*7 V110/V220
≤-10	17
-5 ≥ T > -10	12
5 ≥ T > -5	8
10 ≥ T > 5	4
15 ≥ T > 10	3
20 ≥ T > 15	2
40 ≥ T > 20	1
> 40	0.7

3.2.9 Display

See Section 5 in [9].

(Find the code below in Display_COG_Process.c [2])

```
void Display (unsigned char *Picture,unsigned char *PreviousImage)
{
	//COG Process
	Display_Stage_1 (PreviousImage);
	Display_Stage_2 (PreviousImage);
	Display_Stage_3 (Picture);
	Display_Stage_4 (Picture);
}
```

For more details on the source code of the four stages above, please find them in Display_COG_Process.c. The source code below is for the stage 1 (compensate) only.

✧ Stage 1
(Find the code below in Display_COG_Process.c [2])

```
void Display_Stage_1 (uint8_t *PreviousPicture)
{
	uint16_t	i, j;
	uint8_t		TempByte; // Temporary storage for image data check
	uint16_t	StartClock;
	uint8_t		*DataLineOddPrt;
	Currentframe=COG_Parameters[EPD_Type_Index].FrameTime;
#if(_MeasureStageMark==1)
	LED_Red_Trigger();
#endif
	DisplayOrgPtr=PreviousPicture;
	cnt=0;
	StartClock = TA1R;
	do
	{
#if(_MeasureStageMark==1)
	  LED_Green_Trigger();
#endif
	PreviousPicture=DisplayOrgPtr;
	for (i = 0; i < COG_Parameters[EPD_Type_Index].VERTICAL; i++)// for every line
	{
		// SPI (0x04, 0x03)
		SPI_Send_Byte (0x04, COG_Parameters[EPD_Type_Index].VoltageLevel);

		DataLineOddPrt=(uint8_t
*)&DataLineOdd[COG_Parameters[EPD_Type_Index].HORIZONTAL-1];
		for (j = 0; j < COG_Parameters[EPD_Type_Index].HORIZONTAL; j++)
		{
			TempByte = (*PreviousPicture++);
			DataLineEven[j]	= ((TempByte & 0x80) ? BLACK3  : WHITE3)
					| ((TempByte & 0x20) ? BLACK2  : WHITE2)
					| ((TempByte & 0x08) ? BLACK1  : WHITE1)
					| ((TempByte & 0x02) ? BLACK0  : WHITE0);

			*(DataLineOddPrt--) = ((TempByte & 0x01) ? BLACK3  : WHITE3)
					| ((TempByte & 0x04) ? BLACK2  : WHITE2)
					| ((TempByte & 0x10) ? BLACK1  : WHITE1)
					| ((TempByte & 0x40) ? BLACK0  : WHITE0);
			}
```

```
                    DataLineScan[(i>>2)]= ScanTable[(i%4)];
                    // SPI (0x0a, line data....)
                    SPI_Send (0x0a, (uint8_t *)&COG_Line.uint8,
COG_Parameters[EPD_Type_Index].DataLineSize);

                    // SPI (0x02, 0x25)
                    SPI_Send_Byte (0x02, 0x2F);

                    DataLineScan[(i>>2)]=0;
             }

             if(COG_Parameters[EPD_Type_Index].FrameTime>0)
             {
                    while(Currentframe>(TA1R - StartClock));
             }
             Currentframe=(TA1R - StartClock)+COG_Parameters[EPD_Type_Index].FrameTime ;
             cnt++;
       }while (StageTime>Currentframe);
#if(_MeasureStageMark==1)
       LED_Red_Trigger();
#endif

        while(StageTime>(TA1R - StartClock));

#if(_MeasureStageMark==1)
       LED_Red_Trigger();
#endif
}
```

For example:

Previous image	11	10
New images	11	10
Stage 1	10	11
Stage 2	00	10
Stage 3	00	11
Stage 4	11	10

bit1 bit0	Input
1 1	Black(B)
1 0	White(W)
0 0	Nothing(N)

This table is created by the input data from the [9]. User needs to input the four stages' data likes the bits above. One Dot/pixel is comprised of 2 bits.

Here is an example for stage 1 to get Even and Odd data.

TempByte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	1	0	1	1	0	1	0	0

```
#define BLACK0 = 0x03, BLACK1 = 0x0C, BLACK2= 0x30, BLACK3= 0xC0,
WHITE0 = 0x02, WHITE1 = 0x08, WHITE2 = 0x20, WHITE3 = 0x80
NOTHING = 0x00
(Display_COG_Process.h)
```

Code	Result
(TempByte & 0x80) ? BLACK3 : WHITE3	BLACK3 = 0xC0 = 1100 0000
(TempByte & 0x20) ? BLACK2 : WHITE2	BLACK2 = 0x30 = 0011 0000
(TempByte & 0x08) ? BLACK1 : WHITE1	WHITE1 = 0x08 = 0000 1000
(TempByte & 0x02) ? BLACK0 : WHITE0	WHITE0 = 0x02 = 0000 0010
DataLineEven = 1100 0000 | 0011 0000 | 0000 1000 | 0000 0010 = **1111 1010**	
(TempByte & 0x01) ? BLACK3 : WHITE3	WHITE3 = 0x80 = 1000 0000
(TempByte & 0x04) ? BLACK2 : WHITE2	BLACK2 = 0x30 = 0011 0000
(TempByte & 0x10) ? BLACK1 : WHITE1	BLACK1 = 0x0C = 0000 1100
(TempByte & 0x40) ? BLACK0 : WHITE0	WHITE0 = 0x02 = 0000 0010
DataLineOdd = 1000 0000 | 0011 0000 | 0000 1100 | 0000 0010 = **1011 1110**	

TempByte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	1	0	1	1	0	1	0	0
Color (white(W)=1, black(B)=0)	W	B	W	W	B	W	B	B
Stage 1 Compensate	B	W	B	B	W	B	W	W
Input	11	10	11	11	10	11	10	10
Even data	11		11		10		10	
Odd data		10		11		11		10

Example for stage 2: White

TempByte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	1	0	1	1	0	1	0	0

Code	Result
(TempByte & 0x80) ? WHITE3 : NOTHING	WHITE3 = 0x80 = 1000 0000
(TempByte & 0x20) ? WHITE2 : NOTHING	WHITE2 = 0x20 = 0010 0000
(TempByte & 0x08) ? WHITE1 : NOTHING	NOTHING = 0x00 = 0000 0000
(TempByte & 0x02) ? WHITE0 : NOTHING	NOTHING = 0x00 = 0000 0000
DataLineEven = 1000 0000 | 0010 0000 | 0000 0000 | 0000 0000 = **1010 0000**	
(TempByte & 0x01) ? WHITE3 : NOTHING	NOTHING = 0x00 = 0000 0000
(TempByte & 0x04) ? WHITE2 : NOTHING	WHITE2 = 0x20 = 0010 0000
(TempByte & 0x10) ? WHITE1 : NOTHING	WHITE1 = 0x08 = 0000 1000
(TempByte & 0x40) ? WHITE0 : NOTHING	NOTHING = 0x00 = 0000 0000
DataLineOdd = 0000 0000 | 0010 0000 | 0000 1000 | 0000 0000 = **0010 1000**	

TempByte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	1	0	1	1	0	1	0	0
Color (white(W)=1, black(B)=0)	W	B	W	W	B	W	B	B
Stage 1 Compensate	B	W	B	B	W	B	W	W
Stage 2 White	W	W	W	W	W	W	W	W
Input	10	00	10	10	00	10	00	00
Even data	10		10		00		00	
Odd data		00		10		10		00

✧ Frame Time in Stage

For matching the stage time by MCU frame time (e.g. stage time=480ms for 1.44" and 2" when MCU's frame time is less than 50ms), using a do-while and count of the total frame time for each stage has the same stage time. At the end of each Display_Stage_#, the sample code is as below. The stage time and frame time can be adjusted at `COG_Parameters` structure in `Display_COG_Process.c`

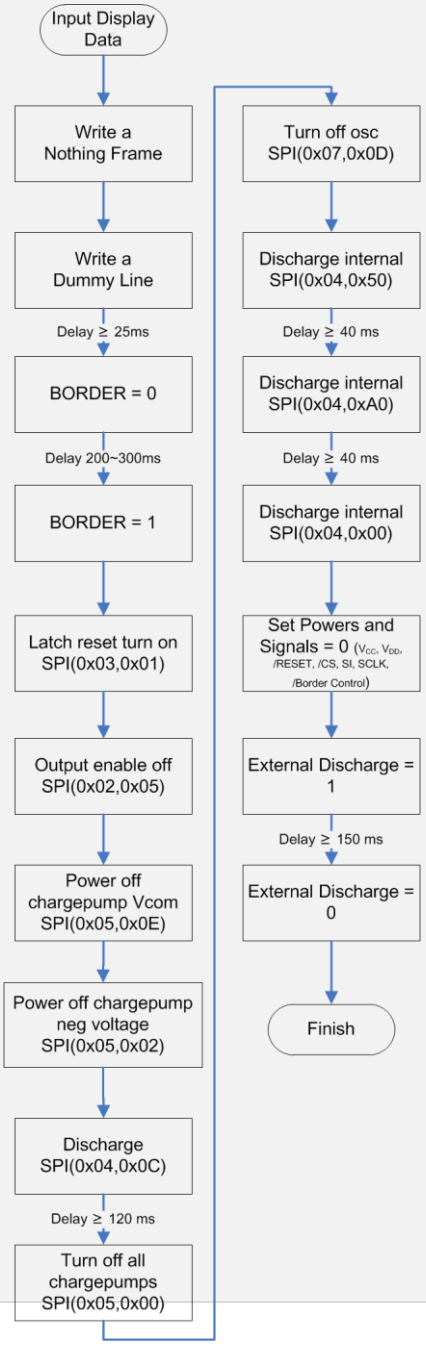
```
void Display_Stage_1 (uint8_t *PreviousPicture)
{

	do
	{

		if(COG_Parameters[EPD_Type_Index].FrameTime>0)
		{
			while(Currentframe>(TA1R - StartClock));
		}
		Currentframe=(TA1R - StartClock)+COG_Parameters[EPD_Type_Index].FrameTime;
		cnt++;
	}while (StageTime>Currentframe);
}
```

3.2.10 Power Off

See Section 6 in [9].

(Find the code below in Display_COG_Process.c [2])

```
void Power_Off (void)
{

        Display_Nothing ();

        Dummy_line();

        Delay (_25ms);

        border_low();
        Delay (_200ms);
        border_high();

        // SPI (0x03, 0x01)
        SPI_Send_Byte (0x03, 0x01);
        // SPI (0x02, 0x05)
        SPI_Send_Byte (0x02, 0x05);
        // SPI (0x05, 0x0E)
        SPI_Send_Byte (0x05, 0x0E);
        // SPI (0x05, 0x02)
        SPI_Send_Byte (0x05, 0x02);
        // SPI (0x04, 0x0C)
        SPI_Send_Byte (0x04, 0x0C);
        Delay (_120ms);
        // SPI (0x05, 0x00)
        SPI_Send_Byte (0x05, 0x00);
        // SPI (0x07, 0x0D)
        SPI_Send_Byte (0x07, 0x0D);
        // SPI (0x04, 0x50)
        SPI_Send_Byte (0x04, 0x50);
        Delay (_40ms);
        // SPI (0x04, 0xA0)
        SPI_Send_Byte (0x04, 0xA0);
        Delay (_40ms);
        // SPI (0x04, 0x00)
        SPI_Send_Byte (0x04, 0x00);

        rst_low ();
        spi_detach ();
        cs_low ();
        panelon_on ();
        border_low();

        discharge_high ();
        Delay (_150ms);
        discharge_low ();
}
```



3.2.11 Save Image to Flash

The 1.44” and 2” EPD will erase the image area 0 and then store the image data to image area 0 in flash memory. For 2.7”, it will erase the image area that “State” byte equals to 0xAA.

(Find the code below in Display_Controller.c [2])

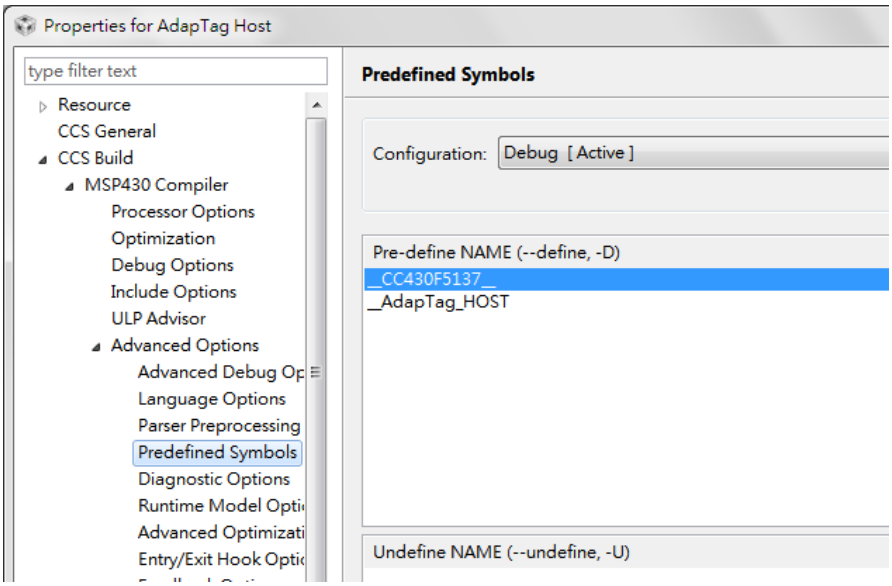
```
void Save_Image (uint8_t EPDType,unsigned char *PreviousImage,unsigned char *NewImage )
{
	uint16_t i,j;
	switch(EPDType)
	{
		case ImageType_1_44:
		case ImageType_2_00:
			In_Flash_Image0_Erase();
			for(i=0;i< COG_Parameters[EPDType].VERTICAL+1;i++)
			{
				j=(uint16_t)(COG_Parameters[EPDType].HORIZONTAL*i);
		In_Flash_Image0_Write(j,NewImage+j,COG_Parameters[EPDType].HORIZONTAL);
			}
			break;
		case ImageType_2_70:
			if(PreviousImage==(uint8_t *)Image0_FlashAddress)
			{
				In_Flash_Image0_Erase();
			}
			else if(PreviousImage==(uint8_t *)Image1_FlashAddress)
			{
				In_Flash_Image1_Erase();
			}
			break;
	}
}
```

3.3 Project Configurations

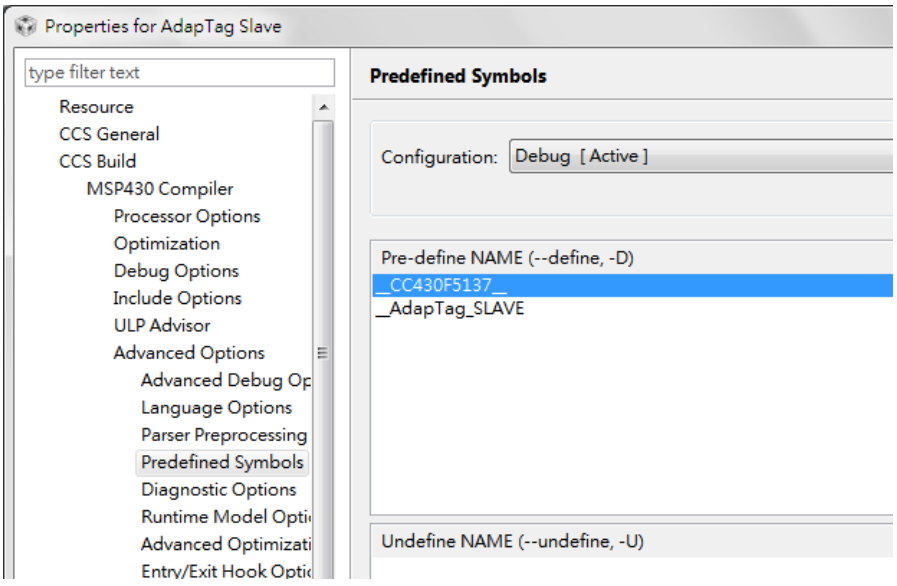
Every project will have a set of properties, which include settings for the compiler, linker, debugger, and more. To view the project properties, right click on the project name at the top of the file list and select "Properties".

✧ Predefined Symbols

- Switch the left side menu of pop-up window to "CCS Build \ MSP430 Compiler \ Advanced Options \ Predefined Symbols"
- AdapTag Host
  There is "__CC430F5137__" and "__AdapTag_HOST" pre-defined names.

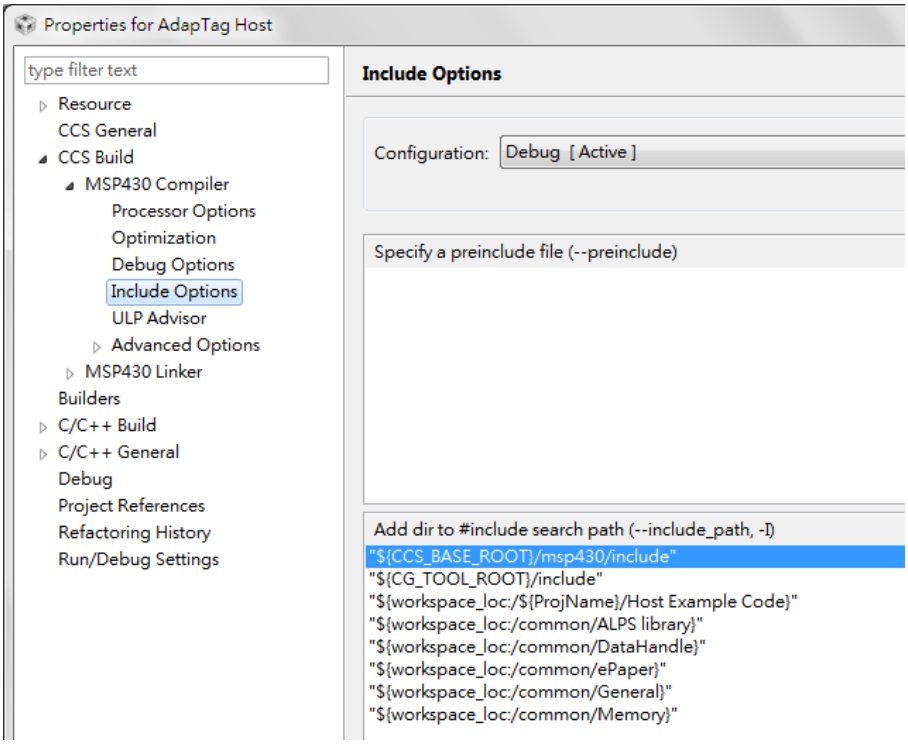


- AdapTag Slave
  There is "__CC430F5137__" and "__AdapTag_SLAVE" pre-defined names.

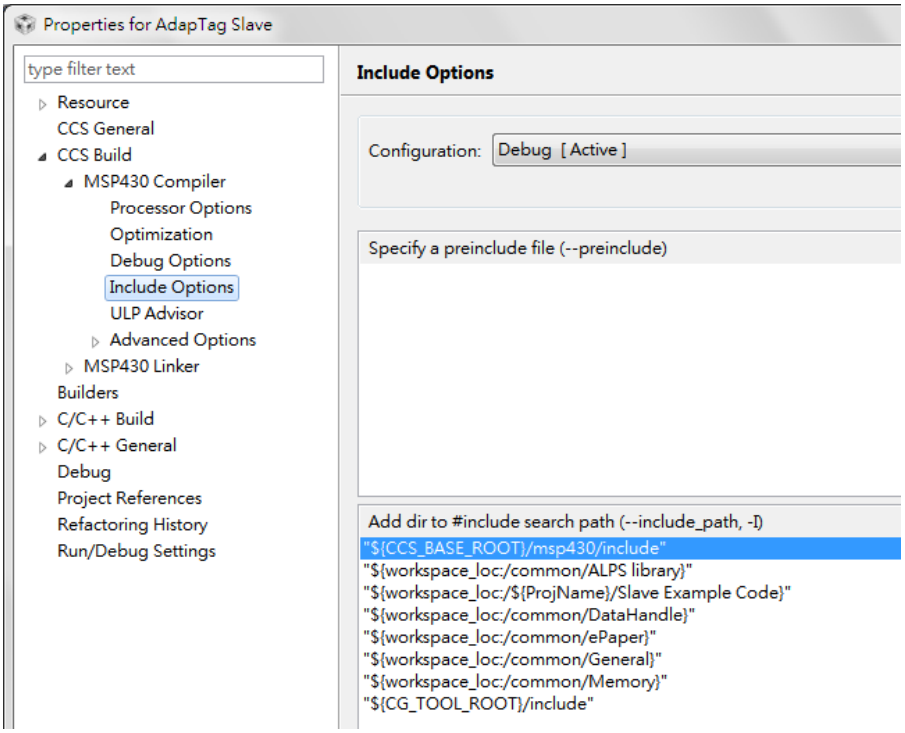


✧ Include Options

- Switch the left side menu to "CCS Build \ MSP430 Compiler \ Include Options"
- AdapTag Host
  Add below directories to list for `#include` search path.

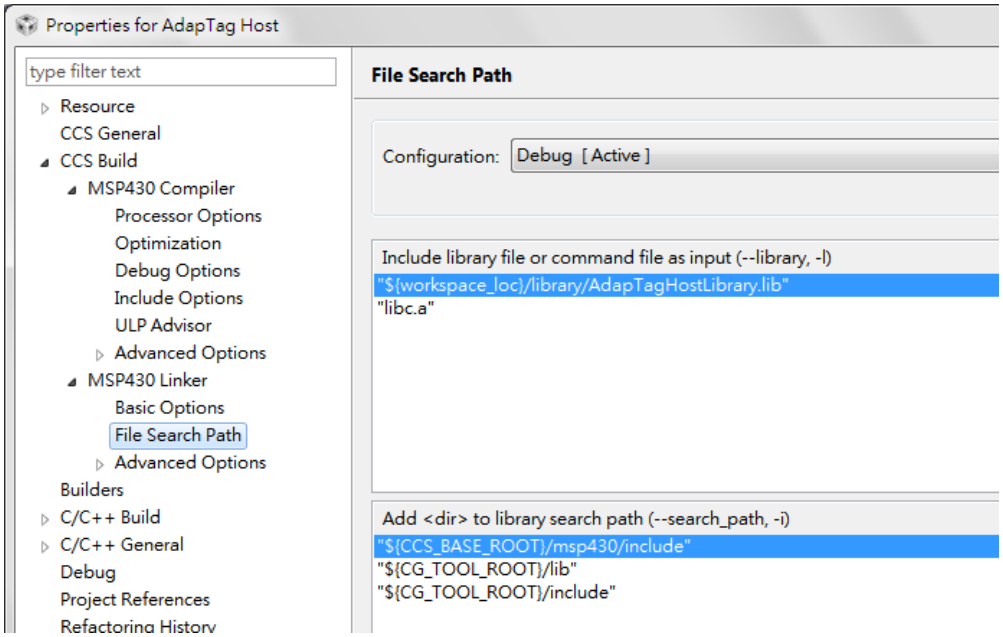


- AdapTag Slave
  (the different is the third list "Slave Example Code")

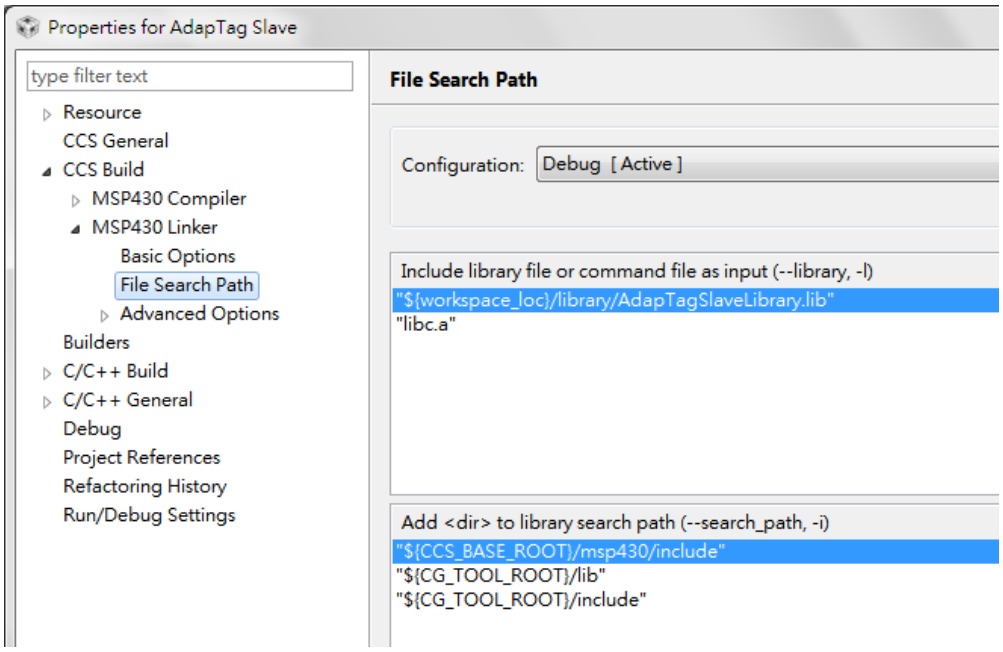


✧ File Search Path

- Switch the left side menu to "CCS Build \ MSP430 Linker \ File Search Path"
- AdapTag Host
  Add below library file or command file to the list.
  `"${workspace_loc}/library/AdapTagHostLibrary.lib"` and `"libc.a"`



- AdapTag Slave
  Add below library file or command file to the list.
  `"${workspace_loc}/library/AdapTagSlaveLibrary.lib"` and `"libc.a"`



4. REFERENCES

The following references provide additional information on the AdapTag development kit, the Texas Instruments ALPS protocol, and the kit specification in general.

[1] The Microsoft® Visual Studio® project and source code of AdapTag Manager is available for download on customer zone of PDi website

[2] The Code Composer Studio ™project and firmware source code of AdapTag host and slave is available for download on customer zone of PDi website

[3] AdapTag Development Kit Quick Start Guide. (Printed copy included in the box with the kit)

[4] AdapTag Development Kit User Manual (Doc No. 8P002-00)

[5] AdapTag Development Kit COG Driver Programming Guide (Doc No. 8P003-00) (This document)

[6] AdapTag Development Kit System Development Guide (Doc No. 8P004-00)

[7] AdapTag Development Kit Protocol and Communication Guide (Doc No. 8P005-00)

[8] Code Composer Studio ™(CCStudio) Integrated Development Environment (IDE) URL: http://www.ti.com/tool/ccstudio

[9] E-paper Display COG Driver Interface Timing (Doc No. 4P008-00 rev.02)

[10]Texas Instruments CC430F5137, 16-Bit Ultra-Low-Power MCU. http://www.ti.com/product/cc430f5137

Glossary of Acronyms

EPD	Electrophoretic Display (e-Paper Display)
EPD Panel	EPD
TCon	Timing Controller
MCU	Microcontroller Unit
SPI	Serial Peripheral Interface
COG	Chip on Glass
PDI, PDi	Pervasive Displays Incorporated
RF	Radio Frequency
ALPS	Advanced Low Power Star network protocol
CCS, CCStudio	Texas Instruments' Code Composer Studio ™ IDE development tool
RSSI	Received Signal Strength Indicator