EPM-100 Driver Board
USB Programming Guide

2021/03/16

Version v.0.1

Revision history

Revision History			
Version	**Date**	**Description**	**Modified by**
0.1	2021/03/16	Create document (Corresponding to EPM-100 FW v.1.4~)	DV

Table of Contents

■ figures

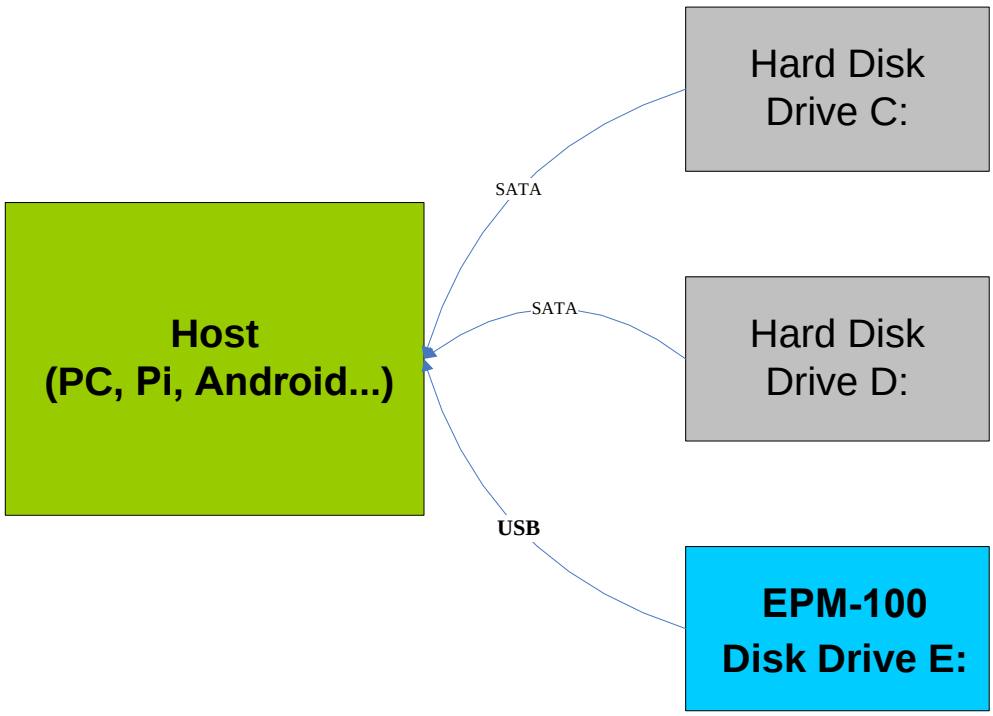
■ Tables

Chapter 1. EPM-100 USB Device

1.1. Introduction for EPM-100 programing through USB

When the EPM-100 is connected to a host device such as a PC via USB cable, it will appear as a drive. The the PC, or other equivalent device such as Raspberry Pi or Android board, can send regular CDB commands specified by SCSI transfer images or commands.

Figure 1 Block Diagram of EPM-100 Device and Host



In this document, we will outline how to program between the Host and the EPM-100 via the USB interface.

For some development environments and platforms (e.g. Android), the CDB[16] may need to be packed to a CBW wrapper as a Bulk transfer API as in the following table,

Command Block Wrapper

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11 (08h-0Bh)	*dCBWDataTransferLength*							
12 (0Ch)	*bmCBWFlags*							
13 (0Dh)	Reserved (0)				*bCBWLUN*			
14 (0Eh)	Reserved (0)				*bCBWCBLength*			
15-30 (0Fh-1Eh)	*CBWCB* CDB[16]							

In this document, we will discuss and focus on EPM-100 specified USB commands in CDB[16] only, for more detailed description about CBW, please see https://www.usb.org/sites/default/files/usbmassbulk_10.pdf

e.g. - Android Bulk API (CBW needed)

https://developer.android.com/reference/android/hardware/usb/UsbDeviceConnection

For Example:

Send CBW[15] + CDB[16] + Write(Out) Data[28]

CBW[0]	0x55	Signature 'U'
CBW[1]	0x53	'S'
CBW[2]	0x42	'B'
CBW[3]	0x43	'C'
CBW[4]	0x00	Tag (Any value)
CBW[5]	0x00	
CBW[6]	0x00	
CBW[7]	0x00	
CBW[8]	**0x1C**	**Data Transfer Length[7:0](Bytes) (IN/OUT Data, CBW exclude)**
CBW[9]	**0x00**	**Data Transfer Length [15:8]**
CBW[10]	**0x00**	**Data Transfer Length[23:16]**
CBW[11]	**0x00**	**Data Transfer Length[31:24]**

CBW[12]	0x00	Direction – Bit[7] 0x80 -Data-In, 0x00 – Data-out
CBW[13]	0x00	LUN – 0 (LUN of EPM-100 is 0)
CBW[14]	0x10	CDB Length, we use 16 here
CDB[0]		
CDB[1]		
……. ……		
CDB[14]		
CDB[15]		
WData[0]		
WData[1]		
…….. …..		
WData[27]		

1.1.1. Initial Flow of Host

As described above, the Host will detect a Mass storage device when the EPM-100 is connected to the Host. The Host will assign a Drive No. (e.g. **E:** , **F:** or…) for the EPM-100.

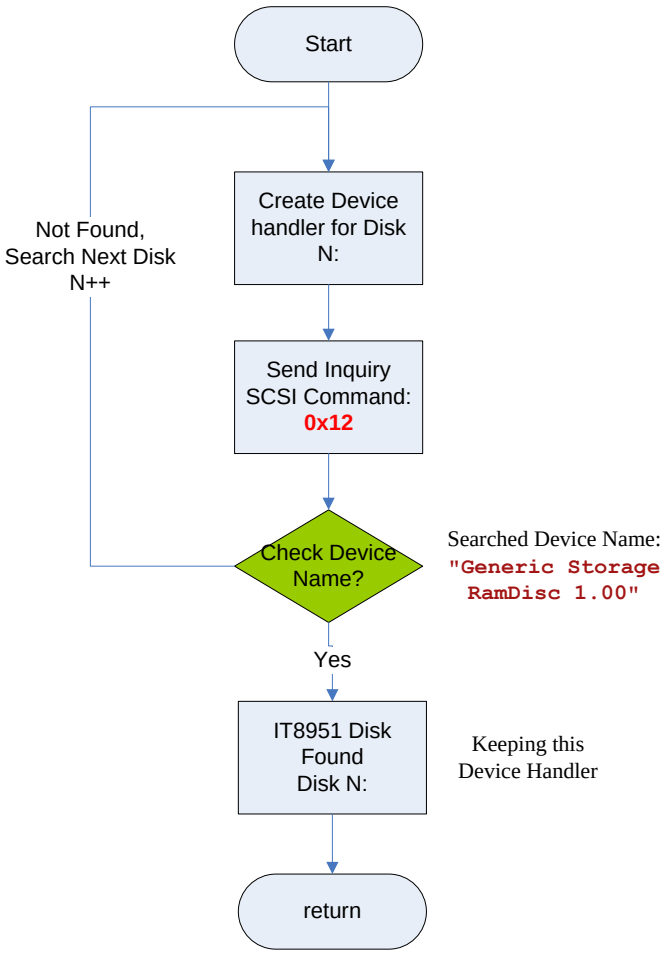
1. Searching Disk to find the Drive No of EPM-100
2. Using inquiry command to check returned device name:

2.1. Send SCSI inquiry command

2.2. Comparing string from received buffer **recvbuf[8]~recvbuf[36]**

is equal to `"Generic Storage RamDisc 1.00"`

Figure 2 Flow chart of Host Initial flow



■ SCSI Inquiry

- **Inquiry and get device name**
- **Direction: Data-In**
- **Programming flow**

1. Send CDB[16]

CDB[0]	**0x12**	Inquiry
CDB[1]	0x00	
CDB[2]	0x00	
CDB[3]	0x00	
CDB[4]	0x00	
CDB[5]	0x00	
CDB[6]	0x00	
CDB[7]	0x00	
CDB[8]	0x00	
CDB[9]	0x00	
CDB[10]	0x00	
CDB[11]	0x00	
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

2. => parameters and data

No

3. <= Read inquiry data from EPM-100

- Size : 40 bytes
- The data structure is defined as following:

Standard INQUIRY data format

bit Byte	7	6	5	4	3	2	1	0
0	Peripheral qualifier			Peripheral device type				
1	RMB	Device-type modifier						
2	ISO version		ECMA version			ANSI-approved version		
3	AENC	TrmIOP	Reserved		Response data format			
4	Additional length (n-4)							
5	Reserved							
6	Reserved							
7	RelAdr	WBus32	WBus16	Sync	Linked	Reserved	CmdQue	SftRe
8 15	(MSB) Vendor identification (LSB)							
16 31	(MSB) Product identification (LSB)							
32 35	(MSB) Product revision level (LSB)							
36 55	Vendor-specific							
56 95	Reserved							
96 n	Vendor-specific parameters Vendor-specific							

"Generic Storage RamDisc 1.00"

1.1.2. Get Device information from EPM-100

The Host needs to do an initial enquiry to get necessary information from the EPM-100:

- Image Width
- Image Height
- Image Buffer Address (Index 0)
- Numbers of waveform modes and Temperature segments
- Numbers of Image buffer index (new added feature in **v.0.3**)

For more detailed description, please see USB command list in next chapter.

Chapter 2 - USB Command list

2.3. USB Host command introduction

In this chapter, we will introduce the EPM-100 default USB command in CDB format, the host program can send the CDB with EPM-100 specified commands and data to control the EPM-100. For more detailed programming flow, please refer our sample code for Windows platform (without CBW)

2.4. New feature added in v.0.3

■ **0x80 – GET_SYS**

- **Get System information**
- **Direction: Data-In**
- **Programming flow**

4. Send CDB[16]

CDB[0]	**0xFE**	Customer command
CDB[1]	0x00	
CDB[2]	**0x38**	Signature[0] of "8951"
CDB[3]	**0x39**	Signature[1]
CDB[4]	**0x35**	Signature[2]
CDB[5]	**0x31**	Signature[3]
CDB[6]	0x80	**Get System information.**
CDB[7]	0x00	
CDB[8]	0x01	Version[8-11]: 0x00010002
CDB[9]	0x00	
CDB[10]	0x02	
CDB[11]	0x00	
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

5. => parameters and data

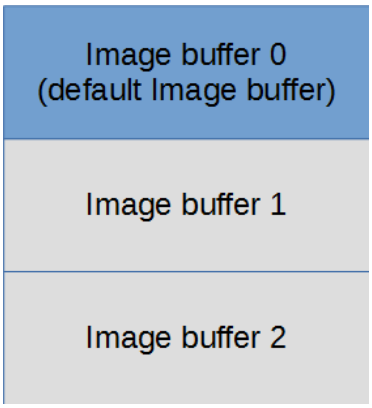
No

6. <= Read Device information data from EPM-100

- Size : 112 bytes
- The data structure is defined as following:

```
typedef struct _TRSP_SYSTEM_INFO_DATA
{
       unsigned int uiStandardCmdNo; // Standard command number2T-con
       Communication Protocol
    unsigned int uiExtendCmdNo; // Extend command number
    unsigned int uiSignature; // 31 35 39 38h (8951)
    unsigned int uiVersion; // command table version
    unsigned int uiWidth; // Panel Width
    unsigned int uiHeight; // Panel Height
    unsigned int uiUpdateBufBase; // Update Buffer Address
    unsigned int uiImageBufBase; // Image Buffer Address(index 0)
    unsigned int uiTemperatureNo; // Temperature segment number
    unsigned int uiModeNo; // Display mode number
    unsigned int uiFrameCount[8]; // Frame count for each mode(8).
    unsigned int       uiNumImgBuf;
    unsigned int       uiWbfSFIAddr;
    unsigned int     uiwaveforminfo;//low byte:A2 mode index
    unsigned int     uiMultiPanelIndex;//High two byte for Y-axis, low two
byte for X-axis
    unsigned int      uiTpXMax;
    unsigned int       uiTpYMax;
   unsigned int       uiReserved[4];
   TRspCmdInfoData     lpCmdInfoDatas[1];} TRSP_SYSTEM_INFO_DATA;
```

For example, if we get uiNumImgBuf = 3
it means there are totally 3 image buffers (includes default image buffer) that can be used currently, the host programmer can load/display an image to any one of these image buffers for some application such as preloading, previous page.., and the default image buffer (return by uiImageBufBase) would be allocated to index 0. For more detailed programming setting, please see EPM-100 USB command 0x94(DisplayArea) and 0xA2(Load Image)



- **0x81 – READ_MEM**
  - **Read Memory function**
  - **Direction: <= Data-In**
  - **Programming flow**

1. Send CDB[16]

CDB[0]	**0xFE**	Customer command
CDB[1]	0x00	
CDB[2]	**Addr[3]**	Memory Address[31:24] Big Endian for EPM-100
CDB[3]	**Addr[2]**	Memory Address [23:16]
CDB[4]	**Addr[1]**	Memory Address [16:8]
CDB[5]	**Addr[0]**	Memory Address [7:0]
CDB[6]	**0x81**	Read Memory command
CDB[7]	**Len[1]**	Length[15:8]
CDB[8]	**Len[0]**	Length[7:0]
CDB[9]	0x00	
CDB[10]	0x00	
CDB[11]	0x00	
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

2. => parameters and data

No

3. <= Read Length: N of Memory data from EPM-100

Size and Read Memory address are indicated by Host.

■ 0x82 – WRITE_MEM

- ■ Write Memory function
- ■ Direction: => Data-Out
- ■ Programming flow

1. Send CDB[16]

CDB[0]	**0xFE**	Customer command
CDB[1]	0x00	
CDB[2]	**Addr[3]**	Memory Address[31:24] Big Endian for EPM-100
CDB[3]	**Addr[2]**	Memory Address [23:16]
CDB[4]	**Addr[1]**	Memory Address [16:8]
CDB[5]	**Addr[0]**	Memory Address [7:0]
CDB[6]	**0x82**	Write Memory command
CDB[7]	**Len[1]**	Length[15:8]
CDB[8]	**Len[0]**	Length[7:0]
CDB[9]	0x00	
CDB[10]	0x00	
CDB[11]	0x00	
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

2. => Write N data (N= Length[15:0] in above)

WDataBuf[0]~ WDataBuf[N-1]

Size and Write Memory address are indicated by Host.

- **0x94 – DPY_AREA**
  - **Display Area function**
  - **Direction: => Data-Out**
  - **Programming flow**

1 Send CDB[16]

CDB[0]	**0xFE**	Customer command
CDB[1]	0x00	
CDB[2]	0x00	
CDB[3]	0x00	
CDB[4]	0x00	
CDB[5]	0x00	
CDB[6]	**0x94**	Display Area command
CDB[7]	0x00	
CDB[8]	0x00	
CDB[9]	0x00	
CDB[10]	0x00	
CDB[11]	0x00	
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

2 Write Arguments - 7 Double Words (28 bytes)

- New Features

In this version, we added a new feature so that a host program can select Address mode(originally) or Index mode to indicate the current loaded/displayed image buffer. For setting new Arguments, see the following:

Image buffer address content:

Bit 31	**Bit [30:0]**	
0 – Address mode	EPM-100 Image buffer Address	
Bit 31	**Bit [30:4]**	**Bit [3:0]**
1 – index mode	reserved	Index

- **If you want to use Address mode as previous:**

Bit[31] = 0 – Address mode (originally)

Bit[30:0] – the Indicated Image buffer Address for displaying

e.g. – load image to address: 0x12345678

Set Arg[0~3] to Image Buffer Addr = 0x12345678 as usual for loading image to indicated memory address.

Arg[0]	**MemAddr[3]**	**0x12**
Arg[1]	MemAddr[2]	0x34
Arg[2]	MemAddr[1]	0x56
Arg[3]	MemAddr[0]	0x78

- **If you want to use Index mode of Image buffer, please try to set image buffer Addr**

  Bit[31] = 1 – index mode

  Bit[30:4] - reserved

  Bit[3:0] – indicate index of Image buffer for displaying

e.g. – load image to index 1 of Image buffer

Set Arg[0~3] Image Buffer Addr = **0x80000001**

Arg[0]	**MemAddr[3]**	**0x80 – Enable Index mode**
Arg[1]	MemAddr[2]	0x00
Arg[2]	MemAddr[1]	0x00
Arg[3]	MemAddr[0]	0x01 – Index 1

Write Arguments - 7 Double Words (28 bytes)

Arg[0]	**MemAddr[3]**	**Image Buffer Addr [31:24] Big Endian**
Arg[1]	MemAddr[2]	Image Buffer Addr [23:16]
Arg[2]	MemAddr[1]	Image Buffer Addr [15:8]
Arg[3]	MemAddr[0]	Image Buffer Addr [7:0]
Arg[4]	**Mode[3]**	**Display Mode [31:24] Big Endian**
Arg[5]	Mode[2]	Display Mode [23:16]
Arg[6]	Mode[1]	Display Mode [15:8]
Arg[7]	Mode[0]	Display Mode [7:0]
Arg[8]	**DpyX[3]**	**Display X [31:24] Big Endian**
Arg[9]	DpyX[2]	Display X [23:16]
Arg[10]	DpyX[1]	Display X [15:8]
Arg[11]	DpyX[0]	Display X [7:0]
Arg[12]	**DpyY[3]**	**Display Y [31:24] Big Endian**
Arg[13]	DpyY[2]	Display Y [23:16]

Arg[14]	DpyY[1]	Display Y [15:8]
Arg[15]	DpyY[0]	Display Y [7:0]
Arg[16]	**DpyW[3]**	**Display Width [31:24] Big Endian**
Arg[17]	DpyW[2]	Display Width [23:16]
Arg[18]	DpyW[1]	Display Width [15:8]
Arg[19]	DpyH[0]	Display Width [7:0]
Arg[20]	**DpyH[3]**	**Display** Height **[31:24] Big Endian**
Arg[21]	DpyH[2]	Display Height [23:16]
Arg[22]	DpyH[1]	Display Height [15:8]
Arg[23]	DpyW[0]	Display Width [7:0]
Arg[24]	**EnWaitRdy[3]**	**EnWait Display Ready [31:24] Big Endian**
Arg[25]	**EnWaitRdy**[2]	**EnWaitRdy** [23:16]
Arg[26]	**EnWaitRdy**[1]	**EnWaitRdy** [15:8]
Arg[27]	**EnWaitRdy**[0]	**EnWaitRdy** [7:0]

- **0xA2 – LD_IMG_AREA**
  - **Host Load Image Area function**
  - **Direction: => Data-Out**
  - All of the pixels need to be stored in 8bpp format by EPM-100 as follows:

0x00	Gray 0 (Black)	**0x40**	Gray 4	**0x80**	Gray 8	**0xC0**	Gray 12
0x10	Gray 1	**0x50**	Gray 5	**0x90**	Gray 9	**0xD0**	Gray 13
0x20	Gray 2	**0x60**	Gray 6	**0xA0**	Gray 10	**0xE0**	Gray 14
0x30	Gray 3	**0x70**	Gray 7	**0xB0**	Gray 11	**0xF0**	Gray 15 (white)

  - **Programming flow**

1. Send CDB[16]

CDB[0]	**0xFE**	Customer command
CDB[1]	0x00	
CDB[2]	0x00	
CDB[3]	0x00	
CDB[4]	0x00	
CDB[5]	0x00	
CDB[6]	**0xA2**	Load Image area command

CDB[7]	0x00	
CDB[8]	0x00	
CDB[9]	0x00	
CDB[10]	0x00	
CDB[11]	0x00	
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

2. Write Argument - 5 Double Words (20 bytes)

Bit[31] = 0 – Address mode (originally)

Bit[30:0] – the Indicated Image buffer Address

e.g. – load image to address:0x12345678

Set Arg[0~3] to Image Buffer Addr = 0x12345678 as usual for loading image to indicated memory address.

Arg[0]	**Addr[3]**	**0x12**
Arg[1]	Addr[2]	0x34
Arg[2]	Addr[1]	0x56
Arg[3]	Addr[0]	0x78

Bit[31] = 1 – index mode

Bit[30:4] - reserved

Bit[3:0] – indicate index of Image buffer

e.g. – load image to index 1 of Image buffer

Set Arg[0~3] Image Buffer Addr = 0x80000001

Arg[0]	**Addr[3]**	**0x80 – Enable Index mode**
Arg[1]	Addr[2]	0x00
Arg[2]	Addr[1]	0x00
Arg[3]	Addr[0]	0x01 – Index 1

Arguments - 5 Double Words (20 bytes)

Arg[0]	**Addr[3]**	**Image buffer Addr [31:24] Big Endian**
Arg[1]	Addr[2]	Image buffer Addr [23:16]
Arg[2]	Addr[1]	Image buffer Addr [15:8]
Arg[3]	Addr[0]	Image buffer Addr [7:0]
Arg[4]	**DpyX[3]**	**Ld Image X [31:24] Big Endian**

Arg[5]	DpyX[2]	Ld Image X [23:16]
Arg[6]	DpyX[1]	Ld Image X [15:8]
Arg[7]	DpyX[0]	Ld Image X [7:0]
Arg[8]	**DpyY[3]**	**Ld Image Y [31:24] Big Endian**
Arg[9]	DpyY[2]	Ld Image Y [23:16]
Arg[10]	DpyY[1]	Ld Image Y [15:8]
Arg[11]	DpyY[0]	Ld Image Y [7:0]
Arg[12]	**DpyW[3]**	**Ld Image Width [31:24] Big Endian**
Arg[13]	DpyW[2]	Ld Image Width [23:16]
Arg[14]	DpyW[1]	Ld Image Width [15:8]
Arg[15]	DpyH[0]	Ld Image Width [7:0]
Arg[16]	**DpyH[3]**	**Ld Image** Height **[31:24] Big Endian**
Arg[17]	DpyH[2]	Ld Image Height [23:16]
Arg[18]	DpyH[1]	Ld Image Height [15:8]
Arg[19]	DpyW[0]	Ld Image Width [7:0]

3. Send Image (Size N = Image Width * Image Height)

The sent image will be collected by EPM-100 whatever Host sends partial or full image.

Data[0]	**Pixel[0]**	**Image Pixel – 8bpp (Raw Data)**
Data[1]	Pixel[1]	**Image Pixel – 8bpp**
Data[2]	Pixel[2]	**Image Pixel – 8bpp**
Data[] ….	Pixel[]	**Image Pixel – 8bpp** …….
Data[N-1]	Pixel[N-1]	Last sent Pixel N =Image Width x Image height

- **0xA3 – PMIC control**
  - **PMIC control for switching power on/off sequence**
  - **Direction: => Data-In**
  - **Programming flow**

4. Send CDB[16]

CDB[0]	**0xFE**	Customer command
CDB[1]	0x00	
CDB[2]	0x00	
CDB[3]	0x00	
CDB[4]	0x00	
CDB[5]	0x00	
CDB[6]	**0xA3**	PMIC Control command
CDB[7]	0x00	Set VCom Value [15:8]
CDB[8]	0x00	Set VCom Value [7:0]
CDB[9]	0x00	Do Set VCom? (0 – no, 1 – Do)
CDB[10]	0x00	Do Power on/off? (0 –no, 1- Do)
CDB[11]	0x00	Power on/off 0 – off, 1 - on
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

- **Set VCom Value**
  - CDB[9] must be set to 1
  - Unit: mV
  - E.g. 2500 => -2500 mV = -2.5V
    - CDB[7] = 0x09
    - CDB[8] = 0xC4
- **Set Power on/off sequence**
  - CDB[10] must be set to 1
  - CDB[11]:
    - 0 – power off
    - 1 – power on
- e.g. Both set Power on and Set VCom
  - CDB[7] = 0x09
  - CDB[8] = 0xC4
  - CDB[9] = 1
  - CDB[10] = 1
  - CDB[11] = 1

■ 0xA5 – FAST_WRITE_MEM

- **Fast Write Memory function**
  - New enhanced command based on 0x82 Memory Write command
- **Direction: => Data-Out**
- **Programming flow**

1. Send CDB[16]

CDB[0]	**0xFE**	Customer command
CDB[1]	0x00	
CDB[2]	**Addr[3]**	Memory Address[31:24] Big Endian for EPM-100
CDB[3]	**Addr[2]**	Memory Address [23:16]
CDB[4]	**Addr[1]**	Memory Address [16:8]
CDB[5]	**Addr[0]**	Memory Address [7:0]
CDB[6]	**0xA5**	Fast Write Memory command
CDB[7]	**Len[1]**	Length[15:8]
CDB[8]	**Len[0]**	Length[7:0]
CDB[9]	0x00	
CDB[10]	0x00	
CDB[11]	0x00	
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

2. => Write N data (N= Length[15:0] in above)

WDataBuf[0]~ WDataBuf[N-1]

Size and Write Memory address are indicated by Host.

■ 0xA6 – Scenario configuration

- **Scenario configuration**
- **Direction: <= Data-In**
- **Programmer could set CDB[2]and CD[3] to do Scenario configuration.**
  - Get Scenario Mode;

    CDB[2]=0,

    CDB[3]=0,// don't care
  - Set Scenario Mode (for example : Scenario 2)

    CDB[2]=1,

CDB[3]=2,

Then EPM-100 would return 4 bytes data(Data[0]~ Data[3]).

Please check the details in the following.

■ **Programming flow**

1. Send CDB[16] (16 bytes totally)

CDB[0]	**0xFE**	Customer command(do not modify it)
CDB[1]	0x00	
CDB[2]	**0x00/0x01**	0 for get scenario, 1 for set scenario
CDB[3]	**Mode**	Scenario mode 0x00 – Normal (TP report enable) 0x01 – Keyboard 0x02 - Handwriting
CDB[4]	0x00	
CDB[5]	0x00	
CDB[6]	**0xA6**	Scenario configuration(do not modify it)
CDB[7]	0x00	
CDB[8]	0x00	
CDB[9]	0x00	
CDB[10]	0x00	
CDB[11]	0x00	
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

2. data (4 bytes totally)

Data[0]		0 for get scenario, 1 for set scenario
Data[1]		Scenario mode
Data[2]		Error code,0 for OK
Data[3]		reserved

■ **0xA7 – AUTO_RESET**

■ **Auto reset EPM-100 function**

◆ Host may send this command to inform EPM-100 Auto Reset after fw upgrading process through USB.

■ **Direction: => Data-Out**

- **Programming flow**

  1. Send CDB[16]

CDB[0]	**0xFE**	Customer command
CDB[1]	0x00	
CDB[2]	0x00	
CDB[3]	0x00	
CDB[4]	0x00	
CDB[5]	0x00	
CDB[6]	**0xA7**	Auto Reset command
CDB[7]	0x00	
CDB[8]	0x00	
CDB[9]	0x00	
CDB[10]	0x00	
CDB[11]	0x00	
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

No parameters

- **0xA8 – SEND_MAILBOX**
  - **Send mailbox content by this specified command**
    - The sent data of this command would be specified mailbox format of EPM-100
  - **Direction: => Data-Out**
  - **Programming flow**

    3. Send CDB[16]

CDB[0]	**0xFE**	Customer command
CDB[1]	0x00	
CDB[2]	0x00	
CDB[3]	0x00	
CDB[4]	0x00	
CDB[5]	0x00	
CDB[6]	**0xA8**	Enhanced load image area command
CDB[7]	0x00	

CDB[8]	0x00	
CDB[9]	0x00	
CDB[10]	0x00	
CDB[11]	0x00	
CDB[12]	0x00	
CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

4. => Write N data (N= Length[15:0] in above)

WDataBuf[0]~ WDataBuf[N-1]

Size and Image buffer **base** address are indicated by Host.

■ 0xA9 – READ_MAILBOX

- This command will return the read data from RData Buffer of mailbox of EPM-100
  - Regarding to the data format of mailbox, please see chapter 3
- **Direction: <= Data-In**

- **Programming flow**

5. Send CDB[16]

CDB[0]	**0xFE**	Customer command
CDB[1]	0x00	
CDB[2]	0x00	
CDB[3]	0x00	
CDB[4]	0x00	
CDB[5]	0x00	
CDB[6]	**0xA9**	
CDB[7]	**Mode**	0 – Read Data 1 – Read Status only
CDB[8]	Size[15:8]	Read Size[15:8]
CDB[9]	Size[7:0]	Read Size[7:0]
CDB[10]	0x00	
CDB[11]	0x00	
CDB[12]	0x00	

CDB[13]	0x00	
CDB[14]	0x00	
CDB[15]	0x00	

- **Set Read mode**
  - Set to CDB[7]
    - 0 – Read Data Buffer of mailbox
      - RData[0]~RData[N-1]

      <=RData (N bytes =by ReadSize)

Data[0]	**RData0 [7:0]**	**RData[] Buffer of mailbox** **Offset: 0x34**
Data[1]	**RData0 [15:8]**	
Data[2]	**RData1 [7:0]**	
Data[3]	**RData1 [15:8]**	
……		
Data[N-2]	**RData(N-2)/2 [7:0]**	
Data[N-1]	**RData(N-2)/2 [15:8]**	

    - 1 – Read Status only
      - 4 bytes

      <=Status (4 bytes totally)

Data[0]	**CmdIndex[7:0]**	**Command index set by malibox**
Data[1]	**CmdIndex[15:8]**	
Data[2]	RetStatus[7:0]	0x0001 - Set_Cmd Done 0x0003 - Function Done Others - Reserved status
Data[3]	RetStatus[15:8]	

3. Introduction to the Mailbox command

3.1. Introduction

For the EPM-100, we have added the protocol called **mailbox Command** for data exchange between a Host and the EPM-100. The mailbox command is able to program on user's demand in EPM-100 Firmware. In this version of firmware, we support the following mailbox commands.

- **How it works?**

  In EPM-100, there is a command table which is used to process the user's defined command, and it is operated by the mailbox mechanism, therefore the command, write parameters and read data should all be exchanged here.

- **Mailbox table:**

All of the arguments/Data are little endian in **16-bits** (word alignment)

Table 1 EPM-100 Mailbox structure

Offset	Byte 0	Byte 1	Direction (Host view)	Field Name
0x0000	0x57	0x89	<= Read	Command Table Signature
0x0002	CmdAddrL[7:0]	CmdAddrL[15:8]	<= Read	Cmd Address[15:0] (Base + 0x10)
0x0004	CmdAddrH[23:16]	CmdAddrL[31:24]	<= Read	Cmd Address[31:16]
0x0006	RDataAddrL[7:0]	RDataAddrL[15:8]	<= Read	Read Data Buffer Address[15:0]
0x0008	RDataAddrH [23:16]	RDataAddrH [31:24]	<= Read	Read Data Buffer Address [31:16]
0x000A	MaxArgCnt[7:0]	MaxArgCnt[15:8]	<= Read	
0x000C	Status[7:0]	Status[15:0]	<= Read	Bit[0] 0 – Busy 1 - Ready
0x000E	Reserved	Reserved	<= Read	
0x0010	**CmdCode[7:0]**	**CmdCode[15:8]**	=> Write	**User defined command code**
0x0012	**CmdIndex[7:0]**	**CmdIndex[15:8]**	=> Write	new added for handshaking Host can set the

				CmdIndex for each cmd (optional)
0x0014	WData0[7:0]	WData0[15:8]	=> Write	Write Data Buffer
0x0016	WData1[7:0]	WData1[15:8]	=> Write	
.....			=> Write	
0x0032	WData15[7:0]	WData15[15:8]	=> Write	
0x0034	RData0[7:0]	RData0[15:8]	<= Read	Read Data Buffer
0x0036	RData1[7:0]	RData1[15:8]	<= Read	
.....			<= Read	
0x00B2	RData63[7:0]	RData63[15:8]	<= Read	
0x00B4	RetCmdIndex[7:0]	RetCmdIndex[15:8]	<= Read	mappped with offset: 0x12
0x00B6	RetStatus[7:0]	RetStatus[15:8]	<= Read => Write 0xFFFF	This status is set when OR1200 **Send the event** to Master CPU only 0x0001 - Set_Cmd Done 0x0003 - Function Done Others - Reserved status

Bit1	Bit0
Function 0 – Busy 1 - Done	Set Cmd 0 – Busy 1 - Done

				Host shall write status to 0xFFFF to indicate the status has been clear.

Initial flow by various host interface:

SPI

Host might need to Get the EPM-100 command table address by read register 0x1230 and 0x1232

USB, I2C

No need to get mailbox address

It can access by specified built-in command

Send the Packed command and arguments

The user defined command and parameters should be packed by host

Send the user defined command package to specified memory address of EPM-100 User defined command table.

Item	16-bits Data	comment
Cmd Code	0xXXXX	Offset: 0x10
Reserved		
WData[0]		
WData[1]		
WData[2]		
……..		
WData[15]		Maximum : 16

- **How to access the mailbox?**
  - **HSPI**
    - By regular Read/Write Memory command
  - **USB**
    - By specified USB command
    - **Send Data to mailbox**
      - Please check the specified USB Command: 0xA8
      - The sent data will be wrote to WData Buffer of mailbox of EPM-100
    - **Read Data from mailbox**
      - Please check the specified USB Command: 0xA9
      - This command will return the read data from RData Buffer of

mailbox of EPM-100

- ■ **I2C**
  - ◆ By regular I2C read/write access with offset

3.2. Mailbox command list

■ **0x00E0 – Get Device System Info**

Get device information from EPM-100, Host should send this command during initial period to get EPM-100 information first.

- ■ No parameters

EPM-100 will return device information as following once host send this command.

- ■ Return 40 bytes (20 words)
  - ◆ **Data[0] –** Current Panel Width
  - ◆ **Data[1]** – Current Panel Height
  - ◆ **Data[2]** – Image buffer address L (Bit[15:0])
  - ◆ **Data[3]** – Image buffer address H (Bit[23:16])
  - ◆ **Data[4]** ~ **Data[11]** – 16 bytes string of current EPM-100 Version
  - ◆ **Data[12]** ~ **Data[19]** – 16 bytes string of current LUT Version
  - ◆ **Data[20]** – wbf buffer address L (Bit[15:0])
  - ◆ **Data[21]** – wbf buffer address H (Bit[23:16])
  - ◆ **Data[22]** – Image buffer[1] address L (Bit[15:0])
  - ◆ **Data[23]** – Image buffer[1] address H (Bit[23:16])

⇨ **Host Send User defined command package to EPM-100 Command Table**

- ■ **Size : 9 words x 2 = 18 bytes**
- ■ **Direction: Out item**

	16-bits Data	comment
Cmd Code	0x00E0	Offset: 0x10 Bit[7:0] = 0xE0 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	**None**	
WData[1]		
WData[2]		

⇨ **Host send the specified command 0xE2 to trigger the user defined command process**

- **Size: 1 bytes**
- **Direction: out**

<= Read data from EPM-100, Receiving Data[0], Data[1]…..in sequence

- **Size: 23-words x 2 = 46 bytes**
- **Direction: In**

Item	16-bits Data	comment
RData[0]	**Panel Width**	Offset: 0x34
RData[1]	**Panel Height**	
RData[2]	**ImgBufAddr L**	
RData[3]	**ImgBufAddr H**	
RData[4]	**FWVerStr[0], FWVerStr[1]**	
RData[5]	**FWVerStr[2], FWVerStr[3]**	
RData[] ……	**FWVerStr[]**	
RData[11]		
RData[12]	**LUTVerStr[0] LUTVerStr[1]**	
RData ……	**LUTVerStr[]**	
RData[19]	**LUTVerStr[18], LUTVerStr[19]**	
RData[20]	**Wbf Buffer Address L [15:0]**	
RData[21]	**Wbf Buffer Address H [31:16]**	
RData[22]	**Image Buffer1 Address L [15:0]**	2nd Image Buffer
RData[23]	**Image Buffer1 Address H [31:16]**	2nd Image Buffer

- **0x0034 – Display Area**

Display function for EPM-100 Update panel, it would display current Image (image buffer of EPM-100) on Panel when host send this command

- 7 parameters
  - **par[0] –** Display x-start position
  - **par[1]** – Display y-start position

- **par[2]** – Display width
- **par[3]** – Display height
- **par[4]** – Display mode
- **par[5]** – image buffer address for displaying [15:0]
- **par[6]** – image buffer address for displaying [25:16]

=> Host Send User defined command package to EPM-100 Command Table

Size : 9 words x 2 = 18 bytes

Item	16-bits Data	comment
Cmd Code	**0x0034**	Offset: 0x10 Bit[7:0] = 0x34 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	**Display X**	
WData[1]	**Display Y**	
WData[2]	**Display W**	
WData[3]	**Display H**	
WData[4]	**Display Mode**	
WData[5]	**Display image buffer address L**	Addr [15:0]
WData[6]	**Display image buffer address H**	Addr[25:16]

⇨ **Host send the specified command 0xE2 to trigger the user defined command process**

Notice. The polling TCon ready behavior will not be in this command.

■ **0x0038 – EPD_PWR_ON**

This command is used for switching EPM-100 DCDC power on/off for EPD, 7 parameters. EPM-100 FW will enable DCDC to generate Source, Gate and VCom power for EPD once the power on was set. And it will do power off sequence once parameters is set 0

- **par[0] –**
  - 0 – Power off
  - 1 – Power on

=> Host Send User defined command package to EPM-100 Command Table

Size : 3 words x 2 = 6 bytes

Item	16-bits Data	comment
Cmd Code	**0x0038**	Offset: 0x10 Bit[7:0] = 0x38 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	**Power on/off**	0 – Power off 1 – Power on

Note:
In FW v.1.4 – EPM-100 support Power on **VSH3**, **VSL3** and VCOM only, the VSH1~2, VSL1~2 are disable.

■ **0x003B – Update the wbf file**

Host should send this command after load the wbf file to wbf buffer of EPM-100, when EPM-100 received this command, it will be decoded and updated to current waveform (volatile)

Note: this command will not store the sent wbf file to external flash currently.

■ No parameters

=> Host Send User defined command package to EPM-100 Command Table

Size : 2 words x 2 = 4 bytes

Item	16-bits Data		comment
Cmd Code	**0x3B**	**0x00**	Offset: 0x10
Reserved	**Dummy**		

⇨ **Host send the specified command 0xE2 to trigger the user defined command process**

EPM-100 will start to parse and update the new wbf file, and the HRDY will be set to High once update done.

Note: Host can read Register (offset: 0x0224) to check the current status of EPM-100. (HSPI only)

■ 0x0033 – Force Set Temperature

This command is used for force set temperature by host, in general, there should be an external thermal sensor for each EPM-100 TCon board, and the EPM-100 would monitor the current temperature and load mapped waveform when display, however, if host may need to control the current temperature without detecting temperature by EPM-100, host can send this command to inform EPM-100 to stop monitoring of reading thermal sensor, and the temperature will be fixed by the value sent from Host.

Force Set temperature

- Send 1 parameters
  - **par[0] –**
    - **0 – Get current set temperature**
    - **1 – Force Set current temperature**
      (EPM-100 Thermal sensor detection will be disabled once set by Host)
  - **Par[1] –**
    - **The temperature value we set**

Item	16-bits Data	comment
Cmd Code	0x0033	Offset: 0x10 Bit[7:0] = 0x33 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	**0 – Get, 1-Set**	
WData[1]	**Temperature Value**	degrees **Celsius** **-128~127** **Bit[7] is signed bit**

Example: Set 20 Celsius degrees to EPM-100

1 Force Set Temperature Flow – Set 20 degree to EPM-100

1.1 Set command 0x0033

1.2 Set WData[0] = 0x0001 //Force Set

1.3 Set WData[1]
0x0014; //e.g. 20 degree
0xFFEC //e.g. -20 degree (2's complement)

1.4 **=> Host Send package to command table of EPM-100**

Item	16-bits Data	comment
Cmd Code	**0x0033**	Offset: 0x10 Bit[7:0] = 0x33 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	**0x0001**	1 - Set
WData[1]	**0x0014 (dec:20)**	degrees **Celsius**

1.5 => Host send the specified command 0xE2 to trigger the user defined command process

2 Get Temperature Flow

2.1 Set command : 0x0033

2.2 Set WData[0] - 0x0000 //Get

2.3 => Host Send package to command table of EPM-100

Item	16-bits Data	comment
Cmd Code	**0x0033**	Offset: 0x10 Bit[7:0] = 0x33 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	**0x0000**	0 - Get
WData[1]	**Don't care**	

2.4 => Host send the specified command 0xE2 to trigger the user defined command process

2.5 Check HRDY status

0 – Busy

1 - Ready

2.6 <= Read data from EPM-100, Receiving Data[0], Data[1]…..in sequence

Item	16-bits Data	comment
RData[0]	**Real Temperature**	Offset: 0x34 if the value hasn't been set in previous
RData[1]	**Set Temperature**	it should be value(20 degree) what we set

■ 0x003C – SET_TCON_CFG

This command is used for Set/upgrade the TCon-Configure file from host to EPM-100, host should preload the TCon-configure binary to specified buffer address(e.g. image buffer) for temporary before sending this command.

Set/Upgrade the TCon-configure file

- Send 1 parameters
  - **par[0] –**
    - **0 – reserved**
    - 1 – Set the loaded TCon-configure setting
    - 2 – Store the TCon-Configure to SPI Flash ROM.(ToDo)
  - **Par[1] –** Loaded buffer address[15:0]
  - **Par[2] –** Loaded buffer address[31:16]
    - **We recommend Host can load the TConCfg.**bin to image buffer
  - **Par[3] –** TCon configure binary size[15:0] (**Unit:** byte)
  - **Par[4] –**TCon configure binary size [31:16] (**Unit:** byte)

- **Set the TCon Configure**

Item	16-bits Data	comment
Cmd Code	**0x003C**	Offset: 0x10 Bit[7:0] = 0x3C Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	0 – reserved 1 – **Set(volatile)** 2 – **Store to SPI Flash*(ToDo)**	The **1 – Set** function is only valid during run time. It will be reset once EPM-100 power off/reset.
WData[1]	**Loaded Buffer Addr L**	the loaded address of TConCfg.bin
WData[2]	**Loaded Buffer Addr H**	
WData[3]	**Size L**	TCon-Cfg.bin size (unit: Byte)
WData[4]	**Size H**	

Example: Set TCon configure to EPM-100

1 Host send the TCon-configure.bin to EPM-100

1.1 Write the TCon configure.bin to Image buffer by MemoryBurstWrite

1.1.1 Write memory Address: **Image buffer Address**(recommended)

1.1.2 Size: file size of **TCon-configure.bin**

See HW Command 0x14

2 Send Mailbox command to Start Set TCon-configure

2.1 Set command 0x003C

2.2 Set WData[0] = 0x0001 //Set

2.3 Set WData[1] = Image buffer Address[15:0]; //recommended

2.4 Set WData[2] = Image buffer Address[31:16]; //recommended

2.5 Set WData[3] = loaded TConConfigure file Size[15:0];

2.6 Set WData[4] = loaded TConConfigure file Size[31:16];

2.7 => Host Send package to command table of EPM-100

Item	16-bits Data	comment
Cmd Code	**0x003C**	Offset: 0x10 Bit[7:0] = 0x3C Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	**0x0001**	1 - Set
WData[1]	Image buffer Address[15:0]	
WData[2]	Image buffer Address[31:16]	
WData[3]	File Size[15:0]	Unit:Byte
WData[4]	File Size[31:16]	

2.8 => Host send the specified command 0xE2 to trigger the user defined command process(SPI only)

3 Wait Set TCon Cfg Done

3.1 Check HRDY

■ 0x003E – EN_REAGL_PROC

This command is used for Enable(Run) the HW Reagl Engine processing of EPM-100, the Source buffer address and Output Buffer Address should be set by parameters when enable this function.

Enable(Run) the HW Reagl Engine processing

- Send 1 parameters
  - **par[0] –**
    - 0 – reserved
    - 1 – **Enable**

    If par[0] is Enable, the following parameter must be set. Otherwise, don't care
  - **Par[1] –** Current Image buffer address[15:0]
  - **Par[2] –** Current Image buffer address[31:16]
    - REAGL Input Buffer
  - **Par[3] –** Output buffer address[15:0]
  - **Par[4] –** Output buffer address[31:16]
    - REAGL Output Buffer

Note: the Previous Image Buffer is fixed to Update buffer of EPM-100, therefore we don't need to set here.

■ **Example :Enable the REAGL Function**

Item	16-bits Data	comment
Cmd Code	**0x003E**	Offset: 0x10 Bit[7:0] = 0x3E Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	0 - reserved 1 – **Enable**	When enable the Reagl Function, the WData[1]~WData[4] must be set for REAGL convert.
WData[1]	**Current Buffer Addr L**	The Current Image buffer(Input) address for Reagl [15:0]
WData[2]	**Current Buffer Addr H**	The Source(Input) buffer address for Reagl [31:16]

WData[3]	**Output Buffer Addr L**	The Output processed buffer address for Reagl [15:0]
WData[4]	**Output Buffer Addr H**	The Output processed buffer address for Reagl [31:16]

■ 0x0040 – READ_RL_REG

This command is used for read register of the Reagl Engine for 32-bits access

Enable(Run) the HW Reagl Engine processing

- Send 1 parameters
  - **par[0] –**
    - 16-bits Register Address
    - The address is only valid for **0x2200~0x222F**

Item	16-bits Data	comment
Cmd Code	**0x0040**	Offset: 0x10 Bit[7:0] = 0x40 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	Register Address[15:0]	The 16-bits Register Address is limited to 0x2200~0x222F

- **<= Read 2 words for Get 32-bits Register value**
  - **RData[0] – 32-bits** Register Value[15:0]
  - **RData[0] – 32-bits** Register Value[31:16]

Item	16-bits Data	comment
RData[0]	**Read Register Value[15:0]**	
RData[1]	**Read Register Value[31:16]**	

Example:

Read the Register **RSTFR** (Address: 0x2204)

Item	16-bits Data	comment
Cmd Code	**0x0040**	Offset: 0x10 Bit[7:0] = 0x40 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	0x2204	The 16-bits Register Address is limited to 0x2200~0x222F

<= **Get the 32-bits Read Value** = 0x00030000

Item	16-bits Data	comment
RData[0]	0x0000	
RData[1]	0x0003	

■ 0x0041 – WRITE_RL_REG

This command is used for write register of the Reagl Engine for 32-bits access

Enable(Run) the HW Reagl Engine processing

- Send 3 parameters
  - **par[0] –**16-bits Register Address
    - The address is only valid for **0x2200~0x222F**
  - **par[1] – Write Value L**
    - Write Register Value[15:0]
  - **par[2] – Write Value H**
    - Write Register Value[31:16]

Item	16-bits Data	comment
Cmd Code	**0x0041**	Offset: 0x10 Bit[7:0] = 0x41 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	Register Address[15:0]	The 16-bits Register Address is limited to 0x2200~0x222F
WData[1]	**Write Value L**	Write Register Value[15:0]
WData[2]	**Write Value H**	Write Register Value[31:16]

Example:

Write 0x00300020 to Register CIBAR (Address: 0x2210)

Item	16-bits Data	comment
Cmd Code	**0x0041**	Offset: 0x10 Bit[7:0] = 0x41 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	0x2210	The 16-bits Register Address is limited to 0x2200~0x222F
WData[1]	**0x0020**	Write Register Value[15:0]
WData[2]	**0x0030**	Write Register Value[31:16]

■ 0x0042 – SET_EPD_PWR_SETTING

This command is used for set the power setting of EPD which included the High Voltage (e.g. Source H, Source L) and VCOM, please kindly note that both the VGL and VGH are not adjustable by FW.

Set the EPD voltage

- Send 8 parameters
  - **par[0] –**
    - 0 – reserved
    - 1 – **Set**
  - **Par[1] –** VSH1 Voltage (mV)
  - **Par[2] –** VSL1 Voltage (mV)
  - **Par[3] –** VSH2 Voltage (mV)
  - **Par[4] –** VSL2 Voltage (mV)
  - **Par[5] –** VSH3 Voltage (mV)
  - **Par[6] –** VSL3 Voltage (mV)
  - **Par[7] –** VCOM Voltage (mV)

Note: If you don't want to set all of Voltages, please only set for target members, the others member are set to **0** (mV).
EPM-100 will not update voltage setting if set voltage is 0.

- **Example :Set the All Source and VCOM**
  - VSH1 = +**15V**
    - Set 15000 = 0x3A98
  - VSL1 = **-15V**
    - Set 15000 = 0x3A98
  - VSH2 = +3.8V
    - Set 3800 = 0xED8
  - VSL2 = -3.6V
    - Set 3600 = 0xE10
  - VSH3 = +6.4 V
    - Set 6400 = 0x1900
  - VSL3 = -6.2 V
    - Set 6200 = 0x1838
  - VCOM = -2.49V
    - Set 2490 = 0x9BA

Item	16-bits Data	comment
Cmd Code	**0x0042**	Offset: 0x10 Bit[7:0] = 0x42 Bit[15:8] = 0x00
Reserved	**Dummy**	
WData[0]	0x0001	1 - Set
WData[1]	0x3A98	VSH1 = **+15V**
WData[2]	0x3A98	VSL1 = **-15V**
WData[3]	0x0ED8	VSH2 = +3.8V
WData[4]	0x0E10	VSL2 = -3.6V
WData[5]	0x1900	VSH3 = +6.4V
WData[6]	0x1838	VSL3 = -6.2V
WData[7]	0x9BA	VCOM = -2.49V

4. Programming Flow

4.1. USB Sample Code on Windows

Platform: Windows library

```
#include "winbase.h"
#include "Ntddscsi.h"
#include "winioctl.h"
```

https://msdn.microsoft.com/zh-tw/library/windows/desktop/aa363216(v=vs.85).aspx

```
#define SPT_BUF_SIZE  (60*1024)
```

Note: the Maximum transfer size is **60K** bytes for EPM-100 USB
If the total transfer size would be over it, please divide them to multiple 60K transactions to complete the whole data transfer.

1. **Initial Flow – Get Device information**

   In EPM-100, host can Get device information by following 2 methods

   - By built-in command `DWORD ITEDisplayAreaAPI`
     - Command: 0x80
     - Return 112 bytes data
     - See function:

```
DWORD T1000GetSystemInfoAPI()
```

   - By MailBox command
     - MailBox command code = 0x00E0
     - See function:

```
DWORD ITEMBGetSystemInfoAPI()
```

2. **Prepare Image**

   - **Please prepare Image pixels data in following format**
     - All of the pixels need to be stored in 8bpp format by EPM-100 as follows:

0x00	Gray 0 (Black)	**0x40**	Gray 4	**0x80**	Gray 8	**0xC0**	Gray 12
0x10	Gray 1	**0x50**	Gray 5	**0x90**	Gray 9	**0xD0**	Gray 13
0x20	Gray 2	**0x60**	Gray 6	**0xA0**	Gray 10	**0xE0**	Gray 14
0x30	Gray 3	**0x70**	Gray 7	**0xB0**	Gray 11	**0xF0**	Gray 15 (white)

3. **Send Image**
   - **By Built-in Command – 0xA5 or 0xA2**
   - **Please see function**

```
DWORD ITELoadImage();
```

4. **Display**
   - By Built-in Command
     - 0x94
   - By MailBox Command (EPM-100 only)
     - 0x0034
   - **Please see function in sample code:**

```
DWORD ITEDisplayAreaAPI()
```

END