聯陽半導體
ITE Tech. Inc.

IT8951 Programming Guide (Host I80 Command Based)

2017/09/04

Version v.2.7

■ Revision history

Revision History

Version	Date	Description	Modified by
1.0	2013/03/18	Create document	Eric Su
1.1	2013/06/05	Added new display flow for customer (User defined command based)	Eric Su
1.2	2013/08/08	Added AC characteristics for IT8951 I80 interface	Eric Su
1.3	2014/07/11	Modify some feature for Intel	Eric
1.4	2015/01/26	1. Added Appendix for SPI to I80 interface by IT8951 CX1 2. Correct the sample code of GetSystemInfo 3. Remove System Command from list	Eric Su
1.5	2015/01/27	1. Fixed some error in sample code of Appendix II	Eric Su
1.6	2015/02/02	1. Modified sample code for SPI to I80 2. LCDReadNData()	Eric Su
1.7	2015/07/31	1. Modified photo of load image	Eric Su
1.8	2015/10/1	1. Added 0x0095 command for loading 1bpp image 2. Modified typo of sample code in WriteCmd P.51	Eric Su
1.9	2015/10/25	1. Added Sample Code of host I2C/I80 interface	Eric Su
2.0	2015/11/05	1. Modified load image 1bpp method 2. Removed 0x0095 for load 1bpp command	Eric Su
2.1	2015/12/25	3. Modified Load Image 1bpp and Display 1bpp sample code	Eric Su

2.2	2016/05/04	Fixed Wait Display Engine Ready function	Eric Su
2.3	2016/06/23	1. Modify figure of SPI waveform 2. Added User define command 0x0038 for power on/off seq control 3. Added User define command 0x0039 for set VCOM of PMIC	Eric
2.4	2016/08/14	Fixed the Read 2 continued words example of SPI interface	Eric Su
2.5	2016/10/21	Modified Load Image programming flow about SPI transfer	Eric Su
2.6	2017/01/17	Fixed typo	Eric Su
2.7	2017/09/04	2048 limitation	Jim Lin

Table of Contents

■ figures

■ Tables

Chapter 1. Host Command list

- **All Command code and Parameters are Word (16bits) width**

1.1. Introduction for IT8951 Host Commands

- **0x0001 – SYS_RUN**
  - **System running command**
  - **Enable all clocks**
  - No parameter

- **0x0003 – SLEEP**
  - **Sleep command**
  - No parameter

- **0x0010 - REG_RD**
  - **Read Register command**
  - Send 1 parameter
    - **Par[0]** – Address of register (offset)
  - Read data from host bus after send command 0x0010 and register address

- **0x0011 - REG_WR**
  - **Write Register command**
  - Send 1 parameters
    - **Par[0]** – Address of register (offset)
  - Send Write data to host bus
    - **WData[0]** – Write data

- **0x0012 – MEM_BST_RD_T**
  - **Memory Burst Read Command**
  - Send 4 parameters
    - **Par[0]** – Address of Memory bit[15:0]
    - **Par[1]** – Address of Memory bit[25:16]
    - **Par[2]** - Size of Read length bit[15:0]
    - **Par[3]** - Size of Read length bit[25:16]

- Send this command for trigging and setting Burst Read parameter
- This command is sent at the beginning of Burst Read procedure

- 0x0013 – MEM_BST_RD_S
  - **Memory Burst Read Start Command**
  - No parameters
  - Send this command for starting read data from bus

- 0x0014 – MEM_BST_WR
  - **Memory Burst Write Command**
  - Send 4 parameters
    - **Par[0]** – Address of Memory bit[15:0]
    - **Par[1]** – Address of Memory bit[25:16]
    - **Par[2]** - Size of Write length bit[15:0]
    - **Par[3]** - Size of Write length bit[25:16]
  - Write data to Memory of T-Con via host bus

- 0x0015 - MEM_BST_END
  - **Burst Access End command**
    - It is must to use this command after execution of Memory Burst Read/Write command
    - Send this command after Burst R/W Transfer finish
  - no parameters

- 0x0020 – LD_IMG
  - **Load image start command**
    - **Send this command before image data transfer from host frame buffer to bus**
    - Send 1 parameter
      - **Par[0]** – Memory converter setting
        - Bit [8]
          - 0 - little Endian
          - 1 – Big Endian
        - Bit [5:4]
          - 00 – 2 bits per pixel
          - 01 – 3 bits per pixel
          - 10 – 4 bits per pixel
          - 11 – 5~8 bits per pixel
        - Bit[1:0]

- 00 – no rotate
- 01 – 90 degrees
- 10 – 180 degrees
- 11 – 270 degrees

0x0021 – LD_IMG_AREA

- **Load image area start command**
  - Send 5 parameters
    - **Par[0]** – Memory converter setting, see Par[0] of CMD- **LD_IMG**(0x0020)
    - **par[1] -** x-start position
    - **par[2]** - y-start position
    - **par[3]** - width
    - **par[4]** - height

0x0022 – LD_IMG_END

- **End Load Image Cycle**
  - It is must to use this command after execution of load image start or load image area command
  - Send this command to stop data transfer of image
- No parameter and data

Table 1 Host Command list

Command List	Code	Parameter					Data	Description
SYS_RUN	0x01							System running Command (enable all clocks, and go to idle state)
SLEEP	0x03							Sleep Command (disable all clock, and go to sleep state)
REG_RD	0x10	Addr[15:0]					rdata[15:0]	Read Register Command
REG_WR	0x11	Addr[15:0]					wdata[15:0]	Write Register Command

MEM_BST_RD_T	0x12	Addr[15:0]	Addr[25:16]	Cnt[15:0]	Cnt[25:16]		rdata[15:0]*cnt	Memory Burst Read Trigger Command (This command will trigger internal FIFO to read data from memory.)
MEM_BST_RD_S	0x13							Memory Burst Read Start Command (This is only a data read command. It will read data from internal FIFO. So, this command should be issued after MEM_BST_RD_T command)
MEM_BST_WR(1)	0x14	Addr[15:0]	Addr[25:16]	Cnt[15:0]	Cnt[25:16]		wdata[15:0]*cnt	Memory Burst Write Command
MEM_BST_END	0x15							End Memory Burst Cycle
LD_IMG(1)	0x20	ARG[15:0]					wdata[15:0]*n	Load Full Image Command (ARG[15:0] see Register 0x200) (Write Data Number Equals to full display size)
LD_IMG_AREA(1)	0x21	ARG[15:0]	start_x[10:0]	start_y[10:0]	width[11:0]	height[11:0]	wdata[15:0]*n	Load Partial Image Command (ARG[15:0] see Register 0x200) (Write Data Number Equals to partial display size according to width and height)
LD_IMG_END	0x22							End Load Image Cycle

1.2. Introduction for IT8951 User Defined Commands

In IT8951, we support user to add new extended I80 command which is called **User Defined Command**, it can be programmable in firmware of IT8951 for customization. In this version of firmware, there are 4 user defined command as following currently:

- **0x0302 – Get Device System Info**

Get device information from IT8951, Host should be send this command in initial steps to get IT8951 information.

- No parameters

IT8951 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 IT8951 Version
  - **Data[12]** ~ **Data[19]** – 16 bytes string of current LUT Version

=> Host Send I80 Command

CMD Code
0x0302

<= Read data from IT8951, Receiving Data[0], Data[1]…..in sequence

Data[0]	**Data[1]**	**Data[2]**	**Data[3]**	**Data[4]**	**Data[5]**
Panel Width	Panel Height	ImgBufAddr L	ImgBufAddr H	FWVerStr[0] FWVerStr[1]	FWVerStr[2] FWVerStr[3]
Data………		**Data[11]**	**Data[12]**	**Data……..**	**Data[19]**
FWVerStr[]…………		FWVerStr[14] FWVerStr[15]	LUTVerStr[0] LUTVerStr[1]	LUTVerStr[]….	LUTVerStr[18] LUTVerStr[19]

- **0x0034 – Display Area**

Display function for IT8951 Update panel, it would display current Image (image buffer of IT8951) on Panel when host send this command

- Send 5 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

=> Host Send I80 Command and 4 Parameters

CMD Code	**Par[0]**	**Par [1]**	**Par [2]**	**Par [3]**	**Par [4]**
0x0034	Display X	Display Y	Display W	Display H	Display mode

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

■ 0x0038 – Power on/off Sequence

The command of power sequence control for EPD Panel by Host.

In general, the power sequence will be turned on when display function executed. And power off by IT8951 once display finished, Therefore this command may not be used if host don't need to control this feature.

- ■ Send 1 parameters
  - ◆ **par[0] –**
    - ● **0 – power off sequence**
    - ● **1 – power on sequence**

=> Host Send I80 Command and 1 Parameter

CMD Code	**Par[0]**
0x0038	0 – power off 1- Power on

■ 0x0039 – Set VCOM value (PMIC platform only)

The command is used for set VCOM output value of PMIC, in general, our default output VCOM is -2.5V, if it doesn't match the value to the current connected EPD, Host can send this command to change the vcom value to meet correct display effect.

Please kindly note that this command is available for platform with PMIC only.

Set VCOM Value

- ■ Send 1 parameters
  - ◆ **par[0] –**

- **0 – Get VCom Value**
- **1 – Set VCom Value**

◆ **Par[1] –**

- **The VCom value we set**

1. **Send Command 0x0039**
2. **Send Arg[0] = 1**

   0 – Get VCOM value

   1 – Set VCOM value
3. **Send Arg [1] - VCom Value**

   e.g. -1.53 = 1530 = 0x5FA
4. **finished**

=> Host Send I80 Command and 1 Parameter to set VCOM value

CMD Code	Par[0]	Par[1]
0x0039	1 – Set VCOM	VCOM value

Get VCOM Value

■ **Send Command 0x0039**

◆ **Send Arg[0] = 0**

0 – Get VCOM value

1 – Set VCOM value

◆ **<= Get RData[0] - VCom Value**

e.g. 0x5FA = 1530 => -1.53V

◆ **finished**

=> Host Send I80 Command and 1 Parameter

CMD Code	Par[0]
0x0039	0 – Get VCOM

<= Get VCOM

RData[0]
VCOM Value

■ 0x0040 – Force Set Temperature

This command is used for force set temperature by host, in general, there should be an external thermal sensor for each IT8951 TCon board, and IT8951 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 IT8951, host can send this command to inform IT8951 to stop detection for 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**

      (IT8951 Thermal sensor detection will be disabled once set by Host)
  - **Par[1] –**
    - **The temperature value we set**

Example: Set 20 degrees to IT8951

1 **Force Set Temperature Flow – Set 20 degree to IT8951**

1.1 Send command 0x0040

1.2 Send data[0] = 0x0001 //Force Set

1.3 Send data[1]] = 0x0014; //e.g. - 20 degree

=> Host Send I80 Command, 1 Parameter and set value

CMD Code	Par[0]	Par[1]
0x0040	1 – Set Temperature	0x14

2 **Get Temperature Flow**

2.1 Send command 0x0040

2.2 Send data[0] 0x0000 //Get

2.3 Read rdata[0] //Temp 0 //Read Real Temperature if the value hasn't been set in previous.

2.4 Read rdata[1] //Temp 1 //or it should be value(20 degree) what we set in 1.3

=> Host Send I80 Command and 1 Parameter to set/get temperature value

CMD Code	Par[0]
0x0040	0 – Get Temperature

<= Host Get temperature value

rData[0]	rData[1]
0x14 (User Set Value)	Current real temperature detected by IT8951

Chapter 2. Programming Guide for Host I80 Command based

2.1. Initial flow

The power-on initial process of IT8951 has been built in SPI ROM, programmer doesn't need to set extra configuration about power-on sequence.

■ **Get Device information**

1 Send Command 0x0302

2 Receiving data of Device information

2.1 **Data[0] –** Current Panel Width

2.2 **Data[1]** – Current Panel Height

2.3 **Data[2]** – Image buffer address L (Bit[15:0])

2.4 **Data[3]** – Image buffer address H (Bit[23:16])

2.5 **Data[4] ~ Data [11]**

2.5.1 Storing the string indicates the current firmware version (16 bytes)

2.6 **Data[12] ~ Data[19]**

2.6.1 Storing the string indicates the current LUT version (16 bytes)

Please storing above information in initial flow of Host

■ **Set Default setting(Recommend)**

1 Set **I80CPCR(0x04)** to 1 for I80 command parameter packed mode

See sample code (Initial function 1 – Set Default value for initial setting) for more detailed description

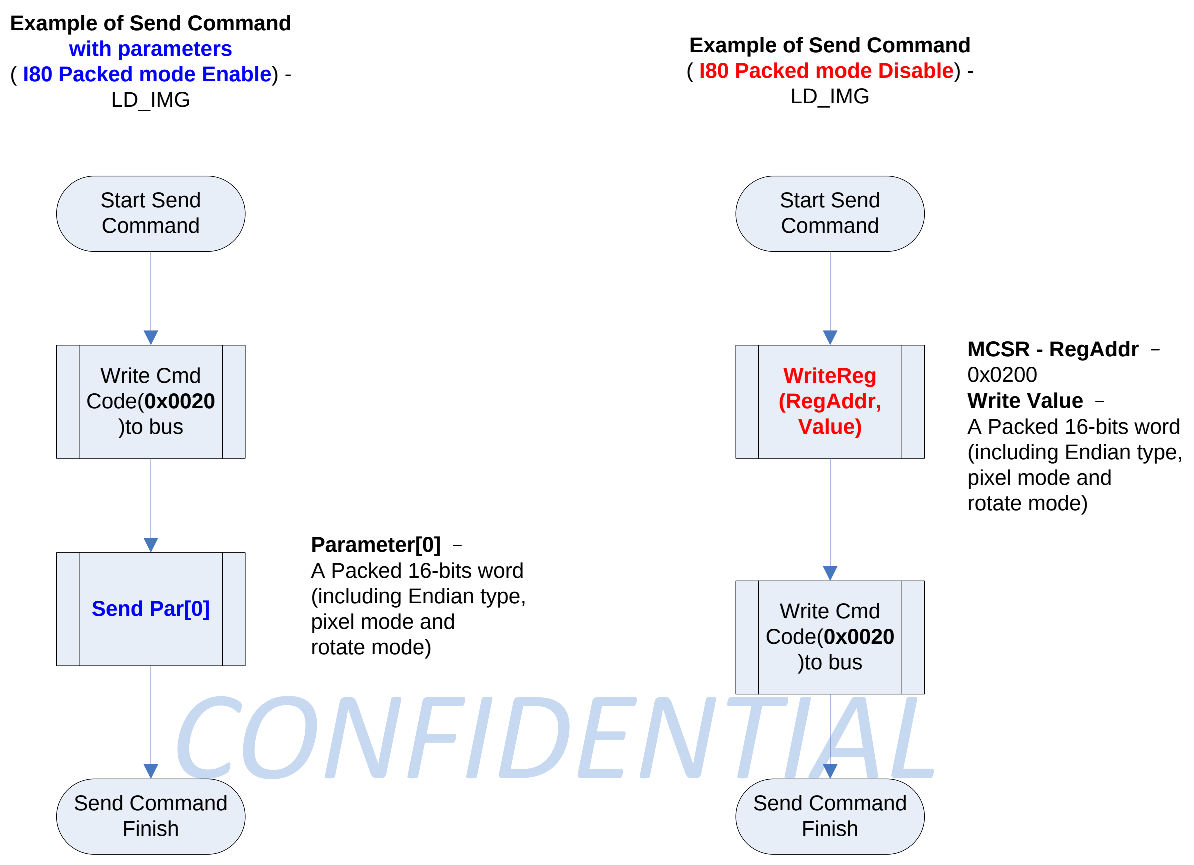
■ **About Parameter packed Enable**

◆ If parameter packed is set enable, we can send continuation of parameters to Host Data Bus after sending command code, otherwise, we need to use write register method to set for each parameter before sending command.

◆ **I80CPCR— Offset 0x04**

- I80 Command parameter control register
- 1 – Enable
- 0 – Disable(default)

figure 1 Flow chart of Send Command with parameter when Enable/Disable I80 Packed mode


Example of Send Command
with parameters
(I80 Packed mode Enable) -
LD_IMG
Start Send Command
Write Cmd Code(0x0020)to bus
Send Par[0]
Parameter[0] – A Packed 16-bits word (including Endian type, pixel mode and rotate mode)
Send Command Finish
Example of Send Command
(I80 Packed mode Disable) -
LD_IMG
Start Send Command
WriteReg (RegAddr, Value)
MCSR - RegAddr – 0x0200
Write Value – A Packed 16-bits word (including Endian type, pixel mode and rotate mode)
Write Cmd Code(0x0020)to bus
Send Command Finish

■

2.2. Register Read/Write process

■ **Read Register**

The steps to Read Register of IT8951 TCon (via I80 bus)

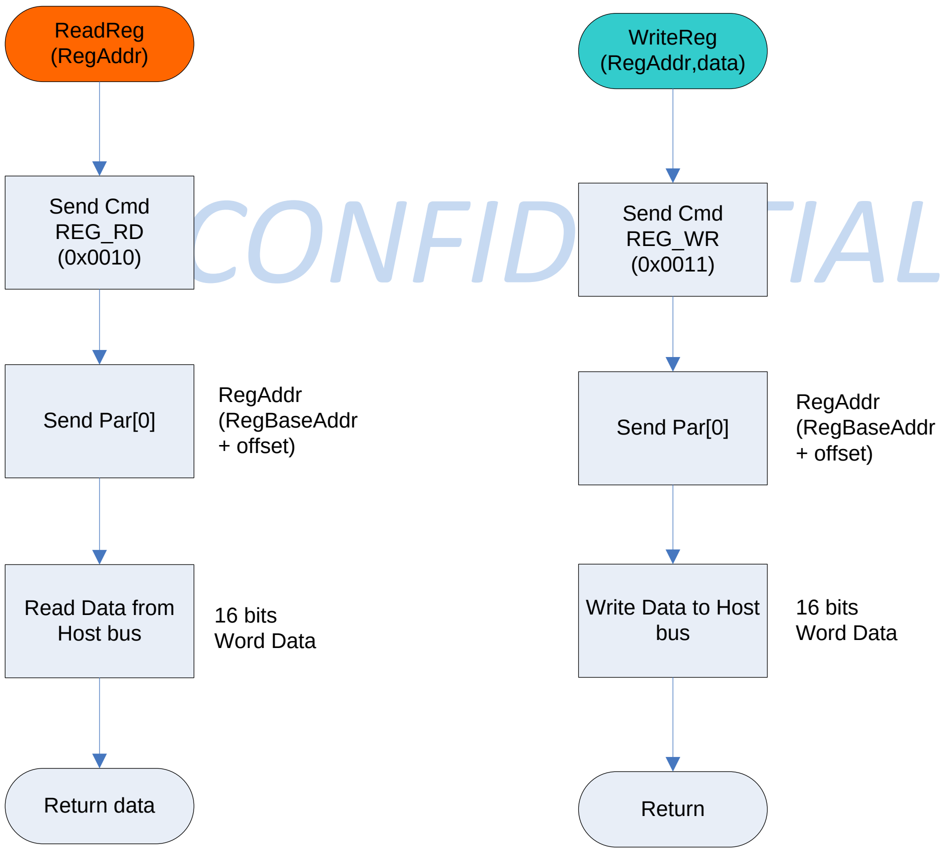
1. Send Host Command - REG_RD(0x10)
2. Send Register address [15:0]
3. Read data from Host Bus

■ **Write Register**

The steps to Write Register of IT8951 TCon (via I80 bus)

1. Send Host Command - REG_WR(0x11)
2. Send Register address [15:0]
3. Send Write Value to Host Bus

figure 2 Flow chart of Read/Write Register



Notice: The command REG_RD(0x0010) and REG_WR(0x0011) are word(16-bits) access for each command cycle in i80 bus. it have to read/write twice If the width of register is Double Word(32-bits)

For example: Read a register which is 32-bits
1.RegDataL = ReadReg(RegAddr)
2.RegDataH = ReadReg(RegAddr + 2)

2.3. Load image process

The process will load image data from Host frame buffer to Image buffer of IT8951.

The maximum width or height of image data should be less than 2048. If the data width or height is larger than 2048, you can split data into two parts or use burst write(8bpp only) to write data to image buffer directly.

1 Set Image buffer address where will be stored(target address)

1.1 The length of image buffer address is 26-bits, so that we have to send this address in twice

1.1.1 WriteReg(LISAR,ImgBufAddr[15:0])

1.1.2 WriteReg(LISAR,ImgBufAddr[25:16])

2 Send Load Image Start command

There are 2 different commands for load image process start

2.1 Load image full start command(0x20-LD_IMG)

This command starts a full frame memory load operation according the data packing setting in parameter. (see command list - LD_IMG - 0x20)

2.1.1 if **I80 parameter packed enable**

Send 1 Parameter **after** sending LD_IMG(0x20) Command

2.2 Load image area start command(0x21-LD_IMG_AREA)

This command is used for load a sub-picture in a full picture, the sub-picture can be load to assigned position of image buffer, it can be particular useful for displaying pop-up window, text box and multiple partial pictures. The partial picture area is denoted by a rectangle with a top left pos(X0,Y0) and width , height of sub-picture.

(see command list - LD_IMG_AREA - 0x21)

2.2.1 if **I80 parameter packed enable (recommend)**

Send 5 Parameters **after** sending LD_IMG_AREA(0x21) Command

2.2.1.1 Par[0] - memory converter setting

2.2.1.2 Par[1]- X-start position setting

2.2.1.3 Par[2]- Y-start position setting

2.2.1.4 Par[3] -image width setting

2.2.1.5 Par[4] - image Height setting

3 Start Data Transfer

3.1 Image data transfer via DMA if supported (see your host)

3.2 CPU write image data transfer

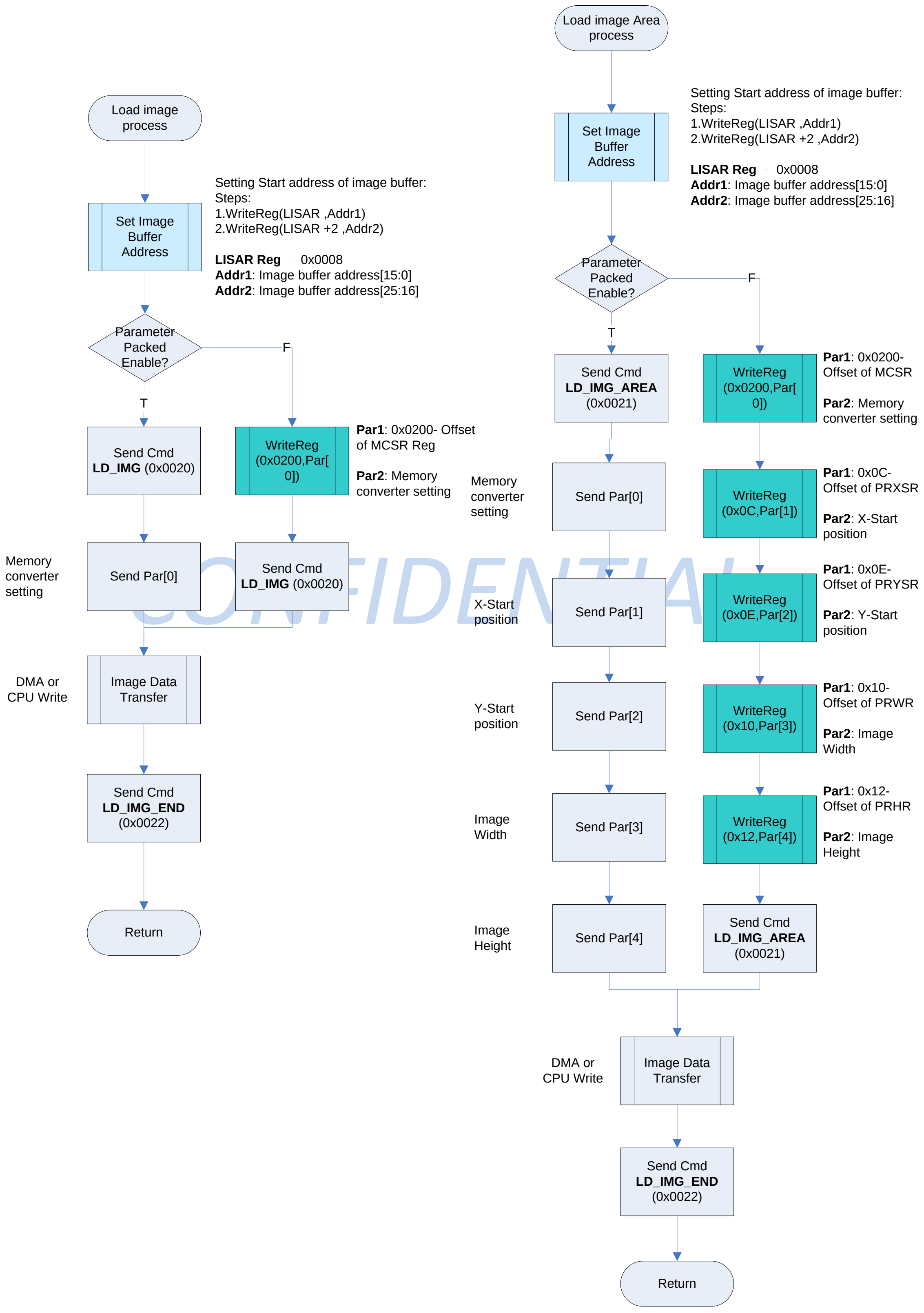
Notice: Do not use Burst Write (0x14) command while using LD_IMG or LD_IMG_AREA command

4 Send Load Image End command to finish

This is a safety measurement to return the controller into the idle state.

It is must to use this command always after execution of load commands

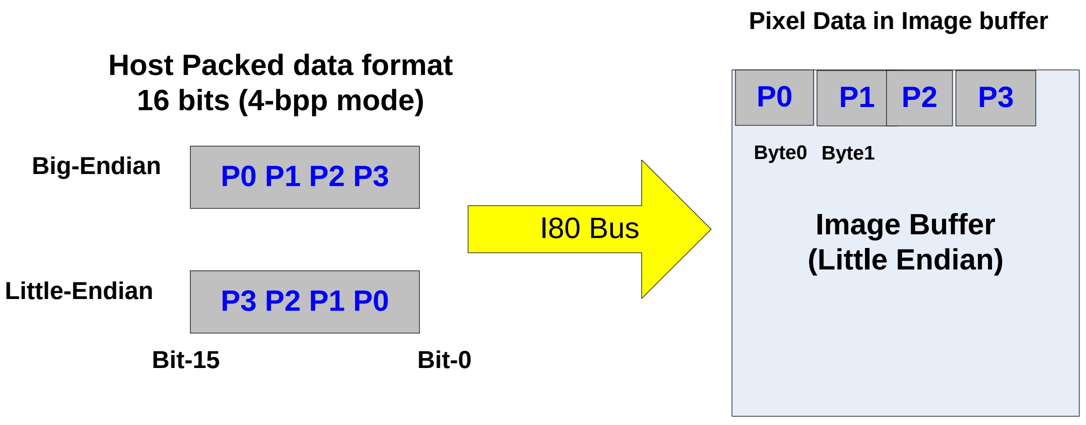
figure 3 flow chart of load image process (Host(Full/Area)PackedPixelWrite)



Notice:

- **Set Image Buffer Address**
  - In IT8951, it must be set image buffer address of IT8951 before loading image, and we just write the Image buffer address to Register LISAR(offset-0x208) to finish the setting. And we recommend to do above mentioned for each load image process to enhance flexible of programming flow.
- **Set Rotate mode**
  - In IT8951, if you want to Load image with rotation, it just set the rotate setting in Arguments Bit[1:0] of LD_IMG(0x20) command. For more detailed description, please see **LD_IMG(0x20) Command**
- **Load Image Start**
    - ◆ Bit8- Endian setting
    - ◆ Bit[5:4] – Host packed mode
    - ◆ Bit[1:0] – Rotate mode
  - In IT8951, the loaded image data can be select the Enidan format to little or big endian. As figure 4 , It can be select 2bpp, 3bpp, 4bpp and 8bpp
  - Rotate degree - 0,90,180and 270
  - Start Data Transfer

figure 4 Host packed data format for Big/Little Endian



- **Limitation of Coordinate of Load Image Area**
  - In IT8951, we recommend to follow the rules for coordinate(X,Y) of load image
    - ◆ 2 bpp mode – X Start position must be multiples of 8
    - ◆ 4 bpp mode – X Start position must be multiples of 4
    - ◆ 8 bpp mode – X Start position must be multiples of 2
  - Otherwise, programmer have to do pre-working before loading image so that it can be loaded

image to Non-multiples position (X,Y)。

- **Example – Load a 4x2 partial image to (3,0) in 4 bpp mode**

1 Check X-start position is divisible by 4?

1.1 3 Mod 4 != 0, it needs padding to (4 – (3 Mod 4)) start dummy pixels

2 Check X-end position (X + W) is divisible by 4?

2.1 7 Mod 4 != 0, it needs padding to (4 – (7 Mod 4))end dummy pixels

3 Set Load Image Area (X,Y,W,H) = (3,0,4,2) and Start Load image Area process

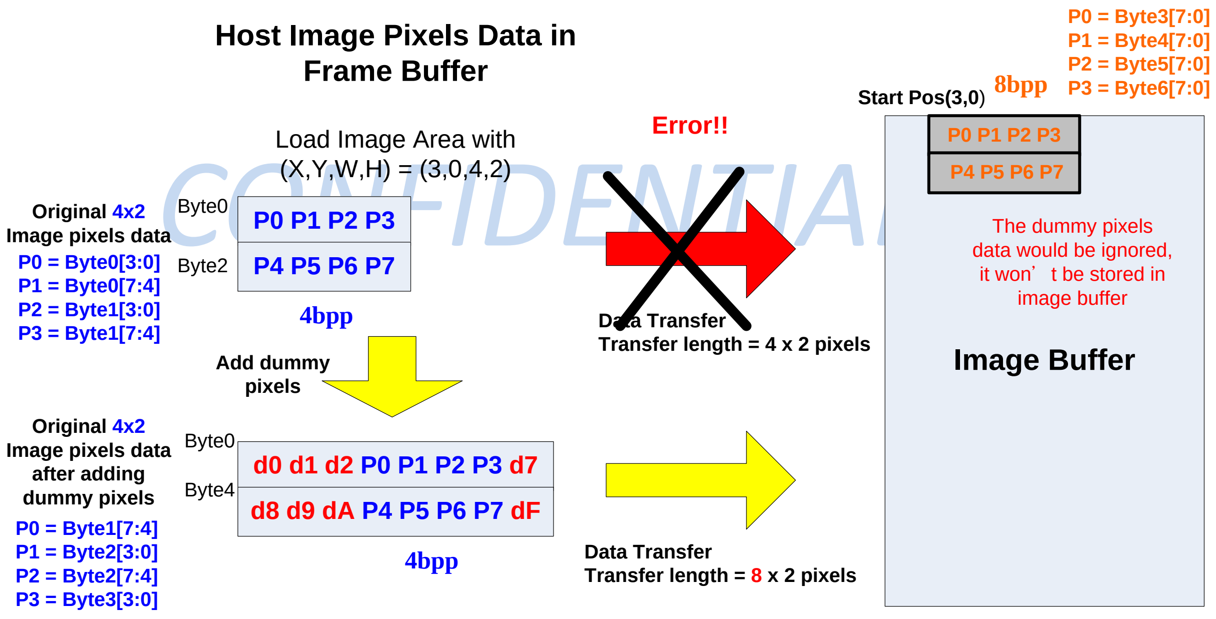
3.1 It still set (x,y,w,h) = (3,0,4,2) for load image area setting

3.2 But it needs to send 8x2 data length(Not 4x2) with dummy pixels from host to IT8951

3.3 The dummy pixel would be ignored that wouldn't be stored to image buffer

3.4 In IT8951, the loaded image data shall be stored into 1 byte per pixel format, what ever your source image is 2,4 or 8-bpp format from host.

figure 5 Example of Load image area that (X,Y) is not divisible by 4



2.4. Burst Read /Write process

This process is used for Read/Write access of Memory of IT8951

- **Burst Read**

  This command starts a memory burst read operation

  1. **Send Burst Read start command (MEM_BST_RD_T – 0x12)**
  2. **Send Memory address[15:0] for read**
  3. **Send Memory address[25:16] for read**
  4. **Send Read data Count[15:0] for this burst read**
  5. **Send Read data Count[25:16] for this burst read**
  6. **Send Burst Read Fetch command (MEM_BST_RD_S – 0x13)**
  7. **Read memory data from host data bus**
  8. **Send Burst End command(MEM_BST_END – 0x15)**

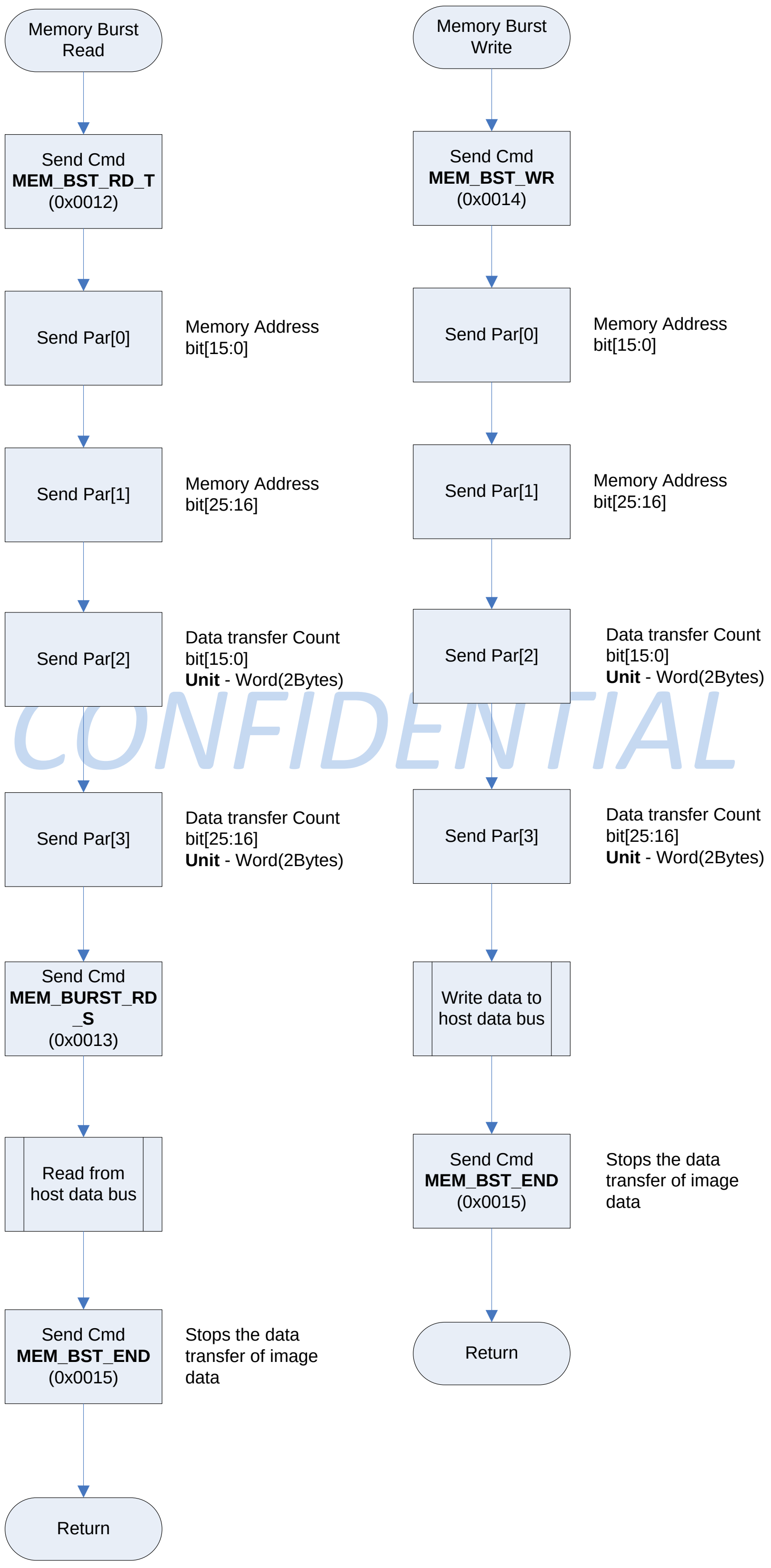
  Notice: The unit of Read data count in step 4 and 5 are Word (2-Bytes)

- **Burst Write**

  1. **Send Burst Write start command (MEM_BST_WR – 0x14)**
  2. **Send Memory address[15:0] for this burst write**
  3. **Send Memory address[25:16] for this burst write**
  4. **Send Write data Count[15:0] for this burst write**
  5. **Send Write data Count[25:16] for this burst write**
  6. **Write data to memory via host data bus**
  7. **Send Burst End command(MEM_BST_END – 0x15)**

  Notice: The unit of Write data count in step 4 and 5 are Word (2-Bytes)

figure 6 flow chart of Memory Burst Read and Write


Memory Burst Read
Send Cmd **MEM_BST_RD_T** (0x0012)
Send Par[0]
Memory Address bit[15:0]
Send Par[1]
Memory Address bit[25:16]
Send Par[2]
Data transfer Count bit[15:0] **Unit** - Word(2Bytes)
Send Par[3]
Data transfer Count bit[25:16] **Unit** - Word(2Bytes)
Send Cmd **MEM_BURST_RD_S** (0x0013)
Read from host data bus
Send Cmd **MEM_BST_END** (0x0015)
Stops the data transfer of image data
Return
Memory Burst Write
Send Cmd **MEM_BST_WR** (0x0014)
Send Par[0]
Memory Address bit[15:0]
Send Par[1]
Memory Address bit[25:16]
Send Par[2]
Data transfer Count bit[15:0] **Unit** - Word(2Bytes)
Send Par[3]
Data transfer Count bit[25:16] **Unit** - Word(2Bytes)
Write data to host data bus
Send Cmd **MEM_BST_END** (0x0015)
Stops the data transfer of image data
Return

2.5. Display Process

The display process is updated the data from image buffer to the Panel. In this version of firmware, we recommend that host can use built-in User defined command – Display Area - 0x0034 to reduce display flow.

1 Polling LUT Engine is ready ? (option)

1.1 Check LUT is ready or not?

1.1.1 Read IT8951 Register – **LUTAFSR** (0x1224) and check value == 0 ?

1.1.1.1 1 -> Busy -> go back to 1.1.1

1.1.1.2 0 -> Display finished -> go to next

1.1.2 finished

2 Enable Display 1bpp mode? No-> please skip to Step 3

2.1 Set Register UP1SR (**0x1138)** Bit[18] = 1

2.2 Set Bitmap color definition – Register BGVR (**0x1250)**

2.2.1 0 – BackGround Gray scale - Bit[15:8]

2.2.2 1 – ForeGround Gray scale Bit[7:0]

3 Send I80 Command – Display area 0x0034 to display Panel

3.1 Send I80 Command Code – 0x0034

3.2 Send Parameter[0] – Display X

3.3 Send Parameter[1] – Display Y

3.4 Send Parameter[2] – Display W

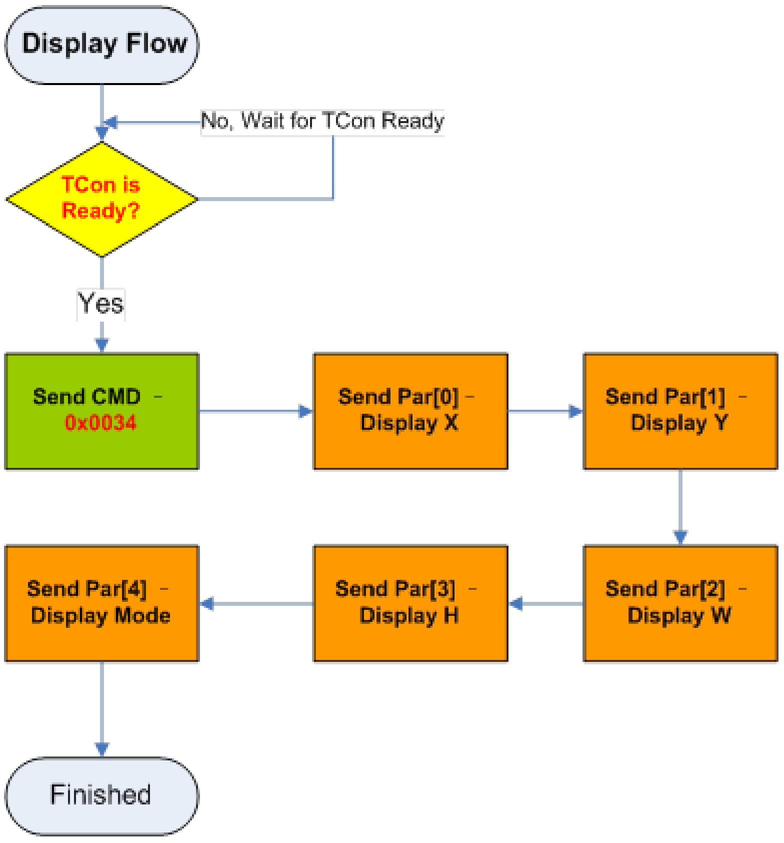
3.5 Send Parameter[3] – Display H

3.6 Send Parameter[4] – Display Mode

In this command (Display Area 0x0034), the operation of IT8951 will not check for TCon Engine ready, it needs to do checking flow by Host.

For more detailed description, please see sample code – **3.6 Display functions**

Figure 7 The display flow from Host with User defined command 0x0034 for IT8951 displaying



Chapter 3. Sample Code of Host for IT8951 Command based

3.1. Prototype Definition

//typedef for variables

```
typedef unsigned char      TByte;  //1 byte
typedef unsigned short     TWord;  //2 bytes
typedef unsigned long      TDWord; //4 bytes
```

//prototype of structure

//structure prototype 1

```
typedef struct IT8951LdImgInfo
{
    TWord  usEndianType;  //little or Big Endian
    TWord  usPixelFormat;  //bpp
    TWord  usRotate;      //Rotate mode
    TDWord ulStartFBAddr; //Start address of source Frame buffer
    TDWord ulImgBufBaseAddr;//Base address of target image buffer

}IT8951LdImgInfo;
```

//structure prototype 2

```
typedef struct IT8951AreaImgInfo
{
    TWord usX;
    TWord usY;
    TWord usWidth;
    TWord usHeight;

}IT8951AreaImgInfo;
```

//structure prototype 3

See user defined command – Get Device information (0x0302)

```
typedef struct
{
    TWord usPanelW;
    TWord usPanelH;
    TWord usImgBufAddrL;
    TWord usImgBufAddrH;
    TWord usFWVersion[8];   //16 Bytes String
    TWord usLUTVersion[8];  //16 Bytes String

}I80IT8951DevInfo;
```

3.2. Constant for IT8951 and Panel setting

```
//Built in I80 Command Code
#define IT8951_TCON_SYS_RUN         0x0001
#define IT8951_TCON_STANDBY         0x0002
#define IT8951_TCON_SLEEP            0x0003
#define IT8951_TCON_REG_RD          0x0010
#define IT8951_TCON_REG_WR          0x0011
#define IT8951_TCON_MEM_BST_RD_T    0x0012
#define IT8951_TCON_MEM_BST_RD_S    0x0013
#define IT8951_TCON_MEM_BST_WR      0x0014
#define IT8951_TCON_MEM_BST_END     0x0015
#define IT8951_TCON_LD_IMG             0x0020
#define IT8951_TCON_LD_IMG_AREA      0x0021
#define IT8951_TCON_LD_IMG_END       0x0022
#define USDEF_I80_CMD_LD_IMG_1BPP   0x0095

//I80 User defined command code
#define USDEF_I80_CMD_DPY_AREA             0x0034
#define USDEF_I80_CMD_GET_DEV_INFO         0x0302

//Panel
#define IT8951_PANEL_WIDTH   1024 //it depends on  Get Device information
#define IT8951_PANEL_HEIGHT   758

//Rotate mode
#define IT8951_ROTATE_0     0
#define IT8951_ROTATE_90    1
#define IT8951_ROTATE_180   2
#define IT8951_ROTATE_270   3

//Pixel mode , BPP - Bit per Pixel
#define IT8951_2BPP        0
#define IT8951_3BPP        1
#define IT8951_4BPP        2
#define IT8951_8BPP        3
```

```
//Waveform Mode
#define IT8951_MODE_0            0
#define IT8951_MODE_1            1
#define IT8951_MODE_2            2

#define IT8951_MODE_3            3
#define IT8951_MODE_4            4



//Endian Type
#define IT8951_LDIMG_L_ENDIAN  0
#define IT8951_LDIMG_B_ENDIAN  1

//Auto LUT
#define IT8951_DIS_AUTO_LUT    0
#define IT8951_EN_AUTO_LUT     1

//LUT Engine Status
#define IT8951_ALL_LUTE_BUSY    0xFFFF

//-----------------------------------------------------------------------
// IT8951 TCon Registers defines
//-----------------------------------------------------------------------
//Register Base Address
#define DISPLAY_REG_BASE        0x1000    //Register RW access for I80 only

//Base Address of Basic LUT  Registers
#define LUT0EWHR    (DISPLAY_REG_BASE + 0x00)    //LUT0 Engine Width Height Reg
#define LUT0XYR     (DISPLAY_REG_BASE + 0x40)    //LUT0 XY Reg
#define LUT0BADDR   (DISPLAY_REG_BASE + 0x80)    //LUT0 Base Address Reg
#define LUT0MFN     (DISPLAY_REG_BASE + 0xC0)    //LUT0 Mode and Frame number Reg
#define LUT01AF     (DISPLAY_REG_BASE + 0x114)   //LUT0 and LUT1 Active Flag Reg

//Update Parameter Setting Register
```

```
#define UP0SR      (DISPLAY_REG_BASE + 0x134)   //Update Parameter0 Setting Reg
#define UP1SR      (DISPLAY_REG_BASE + 0x138)   //Update Parameter1 Setting Reg

#define LUT0ABFRV   (DISPLAY_REG_BASE + 0x13C)   //LUT0 Alpha blend and Fill
rectangle Value

#define UPBBADDR   (DISPLAY_REG_BASE + 0x17C)   //Update Buffer Base Address
#define LUT0IMXY   (DISPLAY_REG_BASE + 0x180)   //LUT0 Image buffer X/Y offset Reg
#define LUTAFSR    (DISPLAY_REG_BASE + 0x224)   //LUT Status Reg (status of All
LUT Engines)
#define BGVR     (DISPLAY_REG_BASE + 0x250)  //Set BG and FG Color if Bitmap mode
enable only
//-------System Registers----------------
#define SYS_REG_BASE  0x0000
//Address of System Registers
#define I80CPCR    (SYS_REG_BASE + 0x04)

//-------Memory Converter Registers----------------
#define MCSR_BASE_ADDR 0x0200
#define MCSR   (MCSR_BASE_ADDR + 0x0000)
#define LISAR   (MCSR_BASE_ADDR + 0x0008)

//-------Enable SPI to I80 interface----------------
#define EN_SPI_2_I80   //Enable SPI to I80 interface for IT8951 CX only
```

3.3. Host controller function

The following Host controller functions are just for reference only, for more detailed operation, please refer to your Host Controller for I80 protocol

//Host controller function 1 – Wait for host data Bus Ready

```
void LCDWaitForReady()
{
    TDWord ulData = HRDY;

    while(ulData == 0)
    {
        //Get status of HRDY
        ulData = HRDY;
    }
}
```

//Host controller function 2 – Write command code to host data Bus

```
void LCDWriteCmdCode(TWord usCmdCode)
{
    //wait for ready
    LCDWaitForReady();

    //write cmd code
    HOST_DATA_BUS = usCmdCode;
}
```

//Host controller function 3 – Write Data to host data Bus

```
void LCDWriteData(TWord usData)
{
    //wait for ready
    LCDWaitForReady();

    //write data
    HOST_DATA_BUS = usData;

}
```

//Host controller function 4 – Read Data from host data Bus

```
TWord LCDReadData()
{
    TWord usData;
    //wait for ready
    LCDWaitForReady();

    //read data from host data bus
    usData = HOST_DATA_BUS;
    return usData;
}
```

//Host controller function 5 – Write command to host data Bus with aruments

```
void LCDSendCmdArg(TWord usCmdCode,TWord* pArg, TWord usNumArg)
{
    TWord i;

    //Send Cmd code
    LCDWriteCmdCode(usCmdCode);
    //Send Data
    for(i=0;i<usNumArg;i++)
    {
         LCDWriteData(pArg[i]);
    }
}
```

3.4. Host Command Functions

//Host Cmd 1 – SYS_RUN

```
void IT8951SystemRun()
{
    LCDWriteCmdCode(IT8951_TCON_SYS_RUN);
}
```

//Host Cmd 2 - STANDBY

```
Not used now
```

//Host Cmd 3 - SLEEP

```
void IT8951Sleep()
{
    LCDWriteCmdCode(IT8951_TCON_SLEEP);
}
```

//Host Cmd 4 - REG_RD

```
TWord IT8951ReadReg(TWord usRegAddr)
{
    TWord usData;

    //----------I80 Mode-------------
    //Send Cmd and Register Address
    LCDWriteCmdCode(IT8951_TCON_REG_RD);
    LCDWriteData(usRegAddr);

    //Read data from Host Data bus
    usData = LCDReadData();

    return usData;

}
```

//Host Cmd 5 - REG_WR

```
void IT8951WriteReg(TWord usRegAddr,TWord usValue)
```

```
{

    //I80 Mode
    //Send Cmd , Register Address and Write Value
    LCDWriteCmdCode(IT8951_TCON_REG_WR);
    LCDWriteData(usRegAddr);
    LCDWriteData(usValue);

}
```

//Host Cmd 6 - MEM_BST_RD_T

```
void IT8951MemBurstReadTrigger(TDWord ulMemAddr , TDWord ulReadSize)
{
    TWord usArg[4];

    //Setting Arguments for Memory Burst Read
    usArg[0] = (TWord)(ulMemAddr & 0x0000FFFF);          //addr[15:0]
    usArg[1] = (TWord)( (ulMemAddr >> 16) & 0x0000FFFF ); //addr[25:16]
    usArg[2] = (TWord)(ulReadSize & 0x0000FFFF);          //Cnt[15:0]
    usArg[3] = (TWord)( (ulReadSize >> 16) & 0x0000FFFF ); //Cnt[25:16]

    //Send Cmd and Arg
    LCDSendCmdArg(IT8951_TCON_MEM_BST_RD_T , usArg , 4);

}
```

//Host Cmd 7 - MEM_BST_RD_S

```
void IT8951MemBurstReadStart()
{
    LCDWriteCmdCode(IT8951_TCON_MEM_BST_RD_S);
}
```

//Host Cmd 8 - MEM_BST_WR

```
void IT8951MemBurstWrite(TDWord ulMemAddr , TDWord ulWriteSize)
{
    TWord usArg[4];

    //Setting Arguments for Memory Burst Write
    usArg[0] = (TWord)(ulMemAddr & 0x0000FFFF);         //addr[15:0]
    usArg[1] = (TWord)( (ulMemAddr >> 16) & 0x0000FFFF ); //addr[25:16]
    usArg[2] = (TWord)(ulWriteSize & 0x0000FFFF);         //Cnt[15:0]
    usArg[3] = (TWord)( (ulWriteSize >> 16) & 0x0000FFFF ); //Cnt[25:16]

    //Send Cmd and Arg
    LCDSendCmdArg(IT8951_TCON_MEM_BST_WR , usArg , 4);

}
```

//Host Cmd 9 - MEM_BST_END

```
void IT8951MemBurstEnd(void)
{
    LCDWriteCmdCode(IT8951_TCON_MEM_BST_END);
}
```

//Host Cmd 10 - LD_IMG

```
void IT8951LoadImgStart(IT8951LdImgInfo* pstLdImgInfo)
{
    TWord usArg;
    //Setting Argument for Load image start
    usArg = (pstLdImgInfo->usEndianType << 8 )
           |(pstLdImgInfo->usPixelFormat << 4)
           |(pstLdImgInfo->usRotate);

    //Send Cmd
    LCDWriteCmdCode(IT8951_TCON_LD_IMG);
    //Send Arg
    LCDWriteData(usArg);

}
```

//Host Cmd 11 - LD_IMG_AREA

```
void IT8951LoadImgAreaStart(IT8951LdImgInfo* pstLdImgInfo ,
                            IT8951AreaImgInfo* pstAreaImgInfo)
{
    TWord usArg[5];
    //Setting Argument for Load image start
    usArg[0] = (pstLdImgInfo->usEndianType << 8 )
               |(pstLdImgInfo->usPixelFormat << 4)
              |(pstLdImgInfo->usRotate);
    usArg[1] = pstAreaImgInfo->usX;
    usArg[2] = pstAreaImgInfo->usY;
    usArg[3] = pstAreaImgInfo->usWidth;
    usArg[4] = pstAreaImgInfo->usHeight;

    //Send Cmd and Args
    LCDSendCmdArg(IT8951_TCON_LD_IMG_AREA , usArg , 5);

}
```

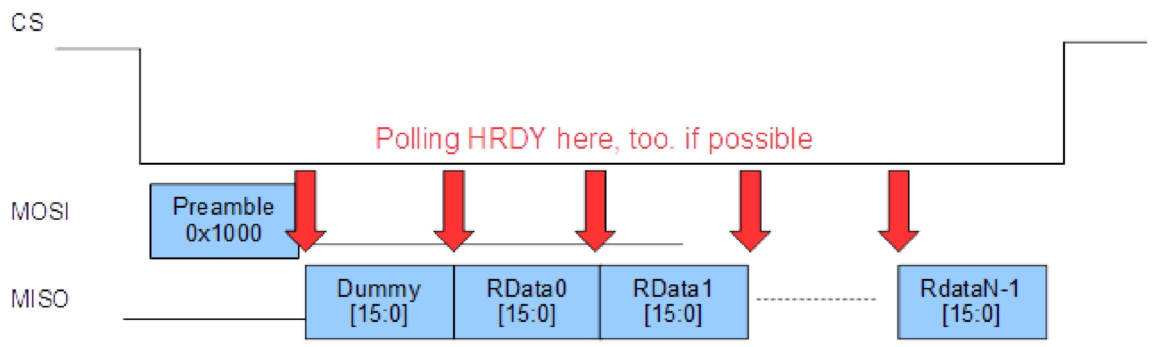
//Host Cmd 12 - LD_IMG_END

```
void IT8951LoadImgEnd(void)
{
    LCDWriteCmdCode(IT8951_TCON_LD_IMG_END);
}
```

3.5. Initial Functions

//Initial function - 1

This is an initial function to get IT8951 Device information, please try to add LCDWaitReady in LCDReadNData for SPI Transfer as following:



If we can't, please try to speed down the SPI Clock (2MHZ) during the Get Device information flow in order to get correct IT8951 Data for more safety.

```
//Global variable
I80IT8951DevInfo gstI80DevInfo;

void GetIT8951SystemInfo(void* pBuf)
{
    TWord* pusWord = (TWord*)pBuf;
    I80IT8951DevInfo* pstDevInfo;
    //Send I80 CMD
    LCDWriteCmdCode(USDEF_I80_CMD_GET_DEV_INFO);
    //Get System Info
    #ifdef EN_SPI_2_I80 //SPI interface only
    LCDReadNData(pusWord, sizeof(I80IT8951DevInfo)/2);
    #else
    for(i=0;i<sizeof(I80IT8951DevInfo)/2;i++)
    {
        pusWord[i] = LCDReadData();
    }
    #endif
    //Show Device information of IT8951
    pstDevInfo = (I80IT8951DevInfo*)pBuf;
    ShowMessage("Panel(W,H) = (%d,%d)\r\n",
               pstDevInfo->usPanelW, pstDevInfo->usPanelH );
    ShowMessage("Image Buffer Address = %X\r\n",
              pstDevInfo->usImgBufAddrL | (pstDevInfo->usImgBufAddrH << 16));

    //Show Firmware and LUT Version
    ShowMessage ("FW  Version = %s\r\n", stI80IT8951DevInfo.usFWVersion);
    ShowMessage ("LUT Version = %s\r\n", stI80IT8951DevInfo.usLUTVersion);

}
```

//Initial function 2 – Set Image buffer base address

```
void IT8951SetImgBufBaseAddr(TDWord ulImgBufAddr)
{
    TWord usWordH = (TWord)((ulImgBufAddr >> 16) & 0x0000FFFF);
    TWord usWordL = (TWord)( ulImgBufAddr & 0x0000FFFF);

    //Write LISAR Reg
    IT8951WriteReg(LISAR + 2 ,usWordH);
    IT8951WriteReg(LISAR     ,usWordL);

}
```

3.6. Display Functions

- Polling functions
- Load image function
- Display functions
    - General Display
    - Display for 1bpp mode

// Display function 1 - Wait for LUT Engine Finish

Polling Display Engine Ready by LUTNo

```
void IT8951WaitForDisplayReady()
{
    //Check IT8951 Register LUTAFSR => 0 - Free, Non-Zero - Busy
    while(IT8951ReadReg(LUTAFSR));
}
```

//Display function 2 – Load Image Area process

.

```
void IT8951HostAreaPackedPixelWrite(IT8951LdImgInfo*  pstLdImgInfo,
                                    IT8951AreaImgInfo* pstAreaImgInfo)
{
    TDWord i,j;
    //Source buffer address of Host
    TWord* pusFrameBuf = (TWord*)pstLdImgInfo->ulStartFBAddr;

    //Set Image buffer(IT8951) Base address
     IT8951SetImgBufBaseAddr(pstLdImgInfo->ulImgBufBaseAddr);

     //Send Load Image start Cmd
     IT8951LoadImgAreaStart(pstLdImgInfo , pstAreaImgInfo);

     //Host Write Data
    for(j=0;j< pstAreaImgInfo->usHeight;j++)
    {
     #ifdef __SPI_2_I80_INF__  //{__SPI_2_I80_INF__
```

```
            //Write 1 Line for each SPI transfer
            LCDWriteNData(pusFrameBuf, pstAreaImgInfo->usWidth/2);
            pusFrameBuf += pstAreaImgInfo->usWidth/2;//Change to Next line of loaded
image (supposed the Continuous image content in hsot frame buffer )

        #else

        for(i=0;i< pstAreaImgInfo->usWidth/2;i++)
        {
            //Write a Word(2-Bytes) for each time
            LCDWriteData(*pusFrameBuf);
            pusFrameBuf++;
        }
        #endif//}__SPI_2_I80_INF__
    }
     //Send Load Img End Command
     IT8951LoadImgEnd();

}
```

//Display functions 3 - Application for Display panel Area

```
Void IT8951DisplayArea(TWord usX, TWord usY, TWord usW, TWord usH, TWord usDpyMode)
{
    //Send I80 Display Command (User defined command of IT8951)
    LCDWriteCmd(USDEF_I80_CMD_DPY_AREA); //0x0034
    //Write arguments
    LCDWriteData(usX);
    LCDWriteData(usY);
    LCDWriteData(usW);
    LCDWriteData(usH);
    LCDWriteData(usDpyMode);

}
```

//Display functions 4 – Load Image Area for 1bpp mode

We still use regular load image command(**0x0021**) to send 1bpp image.

But we need to change following setting:

1. Load Image X setting => X = **X/8**;
2. Load Image Width setting => Width = **Width/8**;

```
void IT8951Load1bppImage(TByte* p1bppImgBuf, TWord usX, TWord usY, TWord usW, TWord
usH)
{

     //Setting Load image information
    stLdImgInfo.ulStartFBAddr     = (TDWord) p1bppImgBuf;
    stLdImgInfo.usEndianType      = IT8951_LDIMG_L_ENDIAN;
    stLdImgInfo.usPixelFormat     = IT8951_8BPP;
    stLdImgInfo.usRotate          = IT8951_ROTATE_0;
    stLdImgInfo.ulImgBufBaseAddr = gulImgBufAddr;

    //Set Load Area
    stAreaImgInfo.usX = usX/8;
    stAreaImgInfo.usY = usY;
    stAreaImgInfo.usWidth = usW/8;
    stAreaImgInfo.usHeight = usH;

    Report("IT8951HostAreaPackedPixelWrite [wait]\n\r");
    //Load Image from Host to IT8951 Image Buffer
    IT8951HostAreaPackedPixelWrite(&stLdImgInfo, &stAreaImgInfo);//Display
function 2

}
```

- **1bpp Pixel Image format**

  e.g. Image Buffer – 16bits Data[0] = 0x2314

  Byte[0] = 0x14 = b00010100

  Byte[1] = 0x23 = b0010011

  1 – foreground

  0 - background

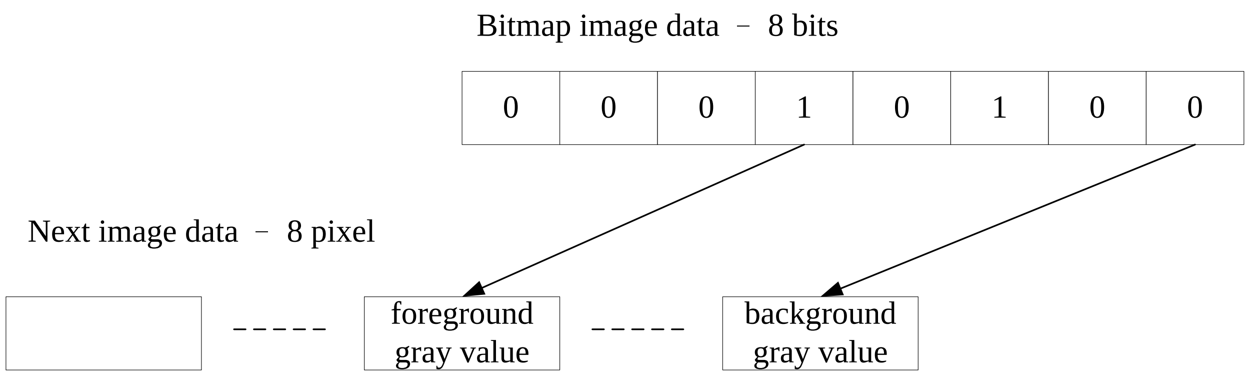
  Assume we set Register BGVR(offset: 0x1250)

  bit[7:0] for foreground = 0xF0 => G15 (white) bit[15:8] for background = 0x00 => G0 (Black)

Pixel 7~0 = {0,0,0,1,0,1,0,0} => {G0, G0, G0, G15, G0, G15, G0, G0}

Pixel 15~8 = {0,0,1,0,0,0,1,1} => {G0, G0, G15, G0, G0, G0, G15, G15}

background gray value: bit15~bit8 of 0x1250
foreground gray value: bit7~bit0 of 0x1250



//Display functions 5 - Application for Display panel Area 1bpp mode

```
void IT8951DisplayArea1bpp(TWord usX, TWord usY, TWord usW, TWord usH, TWord
usDpyMode, TByte ucBGGrayVal, TByte ucFGGrayVal)
{
    //Set Display mode to 1 bpp mode - Set 0x18001138 Bit[18](0x1800113A Bit[2])to
1
    IT8951WriteReg(UP1SR+2, IT8951ReadReg(UP1SR+2) | (1<<2));

    //Set BitMap color table 0 and 1 , => Set Register[0x18001250]:
    //Bit[7:0]: ForeGround Color(G0~G15)  for 1
    //Bit[15:8]:Background Color(G0~G15)  for 0
    IT8951WriteReg(BGVR, (ucBGGrayVal<<8) | ucFGGrayVal);

    //Display
    IT8951DisplayArea( usX, usY, usW, usH, usDpyMode);
    IT8951WaitForDisplayReady();
    //Restore to normal mode
    IT8951WriteReg(UP1SR+2, IT8951ReadReg(UP1SR+2) & ~(1<<2));
}
```

3.7. Test Functions

In host, We need to declare some global structures or variables for storing necessary information of IT8951 device to display.

```
//Global structures and variables
I80IT8951DevInfo gstI80DevInfo;
TByte* gpFrameBuf;      //Host Source Frame buffer
TDWord gulImgBufAddr;  //IT8951 Image buffer address
```

//Test function 1 –Software Initial flow for testing

```

void HostInit()
{
    //Get Device Info
    GetIT8951SystemInfo(&gstI80DevInfo)
    //Host Frame Buffer allocation
    gpFrameBuf = malloc(gstI80DevInfo.usPanelW * gstI80DevInfo.usPanelH);
    //Get Image Buffer Address of IT8951
    gulImgBufAddr = gstI80DevInfo.usImgBufAddrL
                 | (gstI80DevInfo.usImgBufAddrH << 16);
    //Set to Enable I80 Packed mode
   IT8951WriteReg(I80CPCR, 0x0001);
}
```

//Test function 2 – Example of Display Flow

```
void IT8951DisplayExample()
{
   IT8951LdImgInfo  stLdImgInfo;
   IT8951AreaImgInfo stAreaImgInfo;

   //Host Initial
   HostInit(); //Test Function 1
   //Prepare image
   //Write pixel 0xF0(White) to Frame Buffer
   //or you can create your image pattern here..
```

```
    memset(gpFrameBuf, 0xF0, gstI80DevInfo.usPanelW * gstI80DevInfo.usPanelH);

     //Check TCon is free ? Wait TCon Ready (optional)
     IT8951WaitForDisplayReady();

    //Load Image and Display
    //Setting Load image information
     stLdImgInfo.ulStartFBAddr    = (TDWord)gpFrameBuf;
     stLdImgInfo.usEndianType     = IT8951_LDIMG_L_ENDIAN;
     stLdImgInfo.usPixelFormat    = IT8951_8BPP;
     stLdImgInfo.usRotate          = IT8951_ROTATE_0;
     stLdImgInfo.ulImgBufBaseAddr = gulImgBufAddr;
     //Set Load Area
    stAreaImgInfo.usX      = 0;
    stAreaImgInfo.usY      = 0;
    stAreaImgInfo.usWidth  = gstI80DevInfo.usPanelW;
    stAreaImgInfo.usHeight = gstI80DevInfo.usPanelH;
    //Load Image from Host to IT8951 Image Buffer
    IT8951HostAreaPackedPixelWrite(&stLdImgInfo, );//Display function 2

    //Display Area - (x,y,w,h) with mode 0(initial white only) or mode 2(gray)
    IT8951DisplayArea(0,0, gstI80DevInfo.usPanelW, gstI80DevInfo.usPanelH, 0);

}
```

//Test function 3 – Example of Display 1bpp mode Flow

```
//-------------------------------------------------------------
//Test function 3 - Example of Display 1bpp Flow
//-------------------------------------------------------------
void IT8951Display1bppExample()
{
    //Host Initial
    HostInit(); //Test Function 1
    //Prepare image
    //Write pixel 0x00(Black) to Frame Buffer
    //or you can create your image pattern here..
    memset(gpFrameBuf,          0x00,          (gstI80DevInfo.usPanelW          *
gstI80DevInfo.usPanelH)/8);//Host Frame Buffer(Source)

    //Check TCon is free ? Wait TCon Ready (optional)
    IT8951WaitForDisplayReady();

    //Load Image and Display
    //Load Image from Host to IT8951 Image Buffer
    IT8951Load1bppImage(gpFrameBuf,0,0,gstI80DevInfo.usPanelW,
                        gstI80DevInfo.usPanelH);//Display function 4

    //Display Area - (x,y,w,h) with mode 0 or mode 2 for Gray Scale
    //e.g. if we want to set b0(Background color) for Black-0x00 , Set b1(Foreground)
for White-0xFF
    IT8951DisplayArea1bpp(0,0, gstI80DevInfo.usPanelW, gstI80DevInfo.usPanelH,
0, 0x00, 0xFF);

}
```

Appendix I - AC Characteristics for IT8951 I80 interface Timing

Table 2 IT8951 I80 Interface timing

Signal	Symbol	Parameter	Min	Max	Unit	Description
HD/C	t_{ast}	Address setup time (write)	0		ns	
		Address setup time (read)	5		ns	
	t_{wah}	Address hold time (write)	5		ns	
	t_{rah}	Address hold time (read)	0		ns	
HCS_L	t_{wcs}	Chip Select setup time to HWE_L falling edge	0		ns	
	t_{rcs}	Chip Select setup time to HRD_L falling edge	5		ns	
	t_{ch}	Chip Select hold time (write)	5		ns	
		Chip Select hold time (read)	10		ns	
HWE_L	t_{wl}	Pulse low duration	5		ns	
	t_{wh}	Pulse high duration	5		ns	
	t_{wc}	Write cycle for Register	8		Ts	
		Write cycle for Memory	12		Ts	
	t_{w2r}	HWE_L rising edge to HRD_L falling edge	6		Ts	
HRD_L	t_{r2w}	HRD_L rising edge to HWE_L falling edge	0		ns	
	t_{rc}	Read cycle for Registers	9		Ts	
		Read cycle for Memory	5		Ts	
	t_{rl}	Pulse low duration (for Registers)	8T + 10		Ts	
		Pulse low duration (for Memory)	4T + 10		Ts	
	t_{rh}	Pulse high duration	5		ns	
HDB[15:0]	t_{dst}	Write data setup time	7		ns	
	t_{dht}	Write data hold time	6		ns	
	t_{rodz}	Read data hold time from HRD_L rising edge	10		ns	
	t_{rrdz}	HRD_L rising edge to HDB[15:0] Hi-Z		11	ns	

	t_{codh}	Read data hold time from HCS_L rising edge		0	ns	
	t_{crdz}	HCS_L rising edge to HDB[15:0] Hi-Z		0	ns	
	t_{rdv}	HRD_L falling edge to HDB[15:0] valid for Registers		8T + 10	ns	
		HRD_L falling edge to HDB[15:0] valid for Memory (if t_{rc} not met)		4T+10	ns	
	t_{rdd}	HRD_L falling edge to HDB[15:0] driven		0	ns	
HRDY	t_{rdyd}	HCS_L falling edge to HRDY driven		0	ns	
	t_{rdyz}	HCS_L rising edge to HRDY Hi-Z		0	ns	
	t_{wrdyl}	HWE_L rising edge to HRDY low		2	Ts	
	t_{rrdyl}	HRD_L falling edge to HRDY low		2	Ts	
	t_{rrdyh}	HRD_L falling edge to HRDY high		8T + 11	Ts	
	t_{rdyh}	HRDY high to HWE_L rising edge	5		ns	

1. Ts = system clock period

Figure 8 Read Timing for Intel 80 Interface

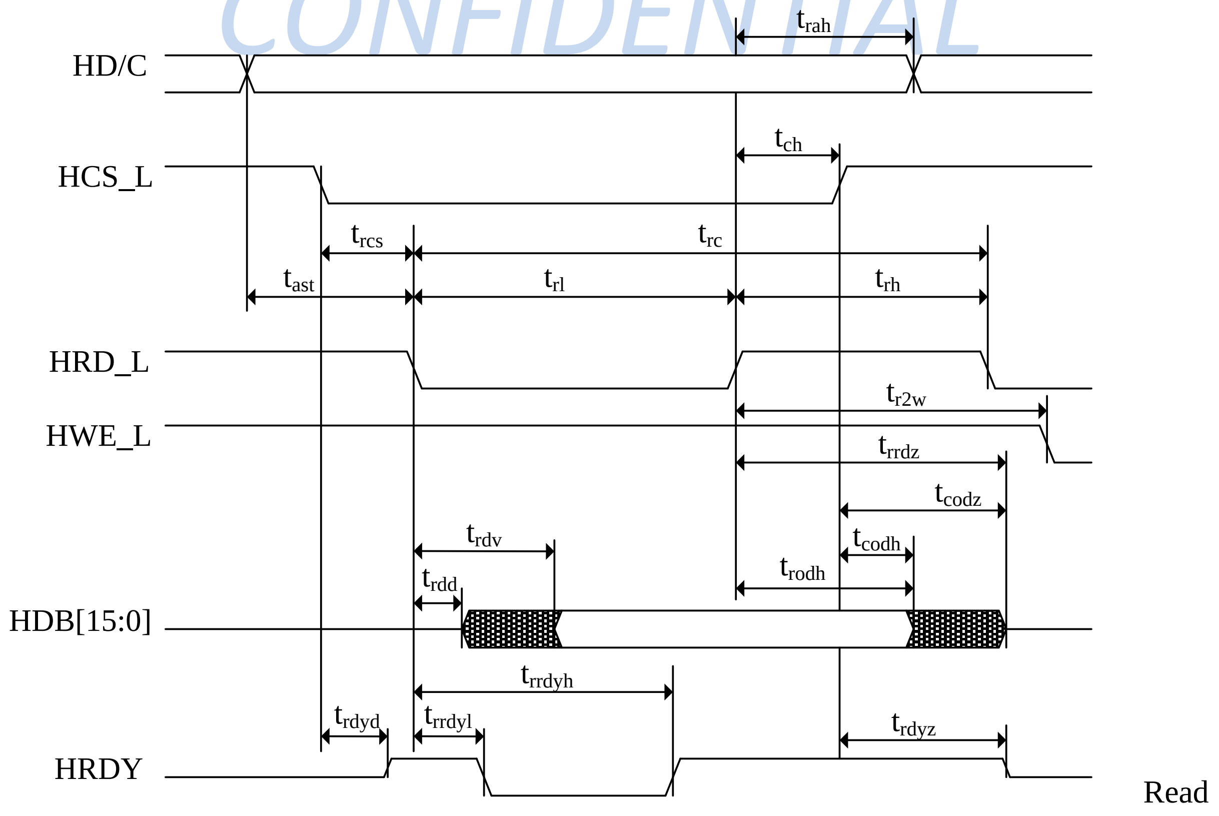
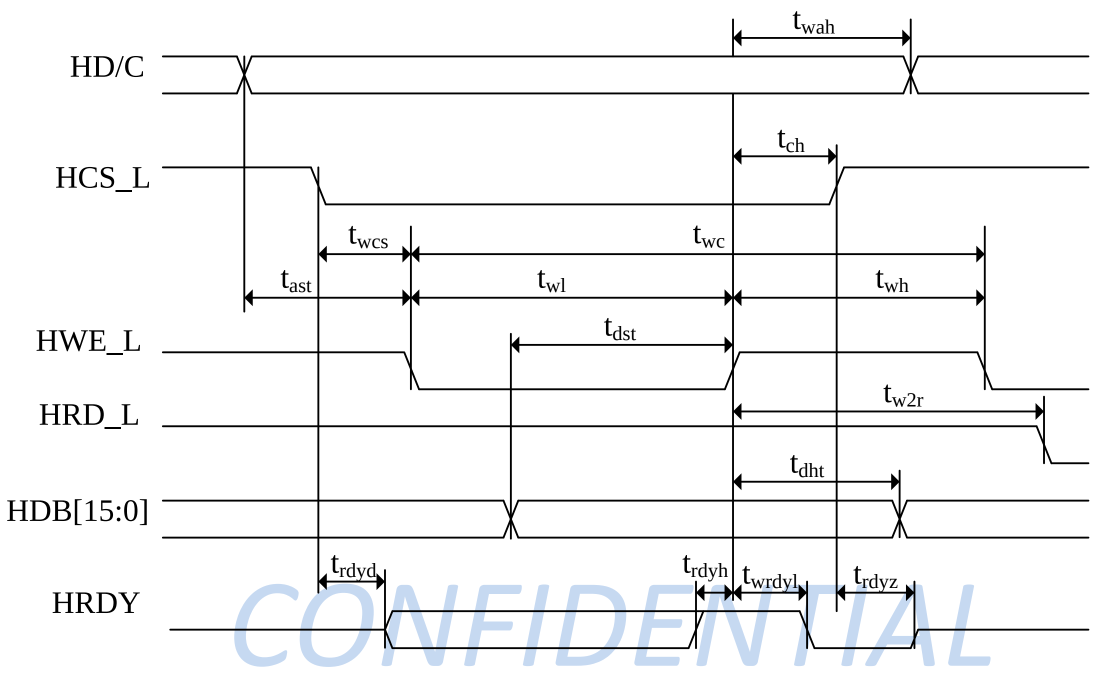


Figure 9 Write Timing for Intel 80 Interface



Appendix II – Programming guide for SPI to I80 interface

(IT8951 CX/DX only)

In IT8951 CX/DX version, we added new feature that Host can send I80 command through SPI interface, Programmer should modify the following basic functions and Implement SPI driver which depends on your host

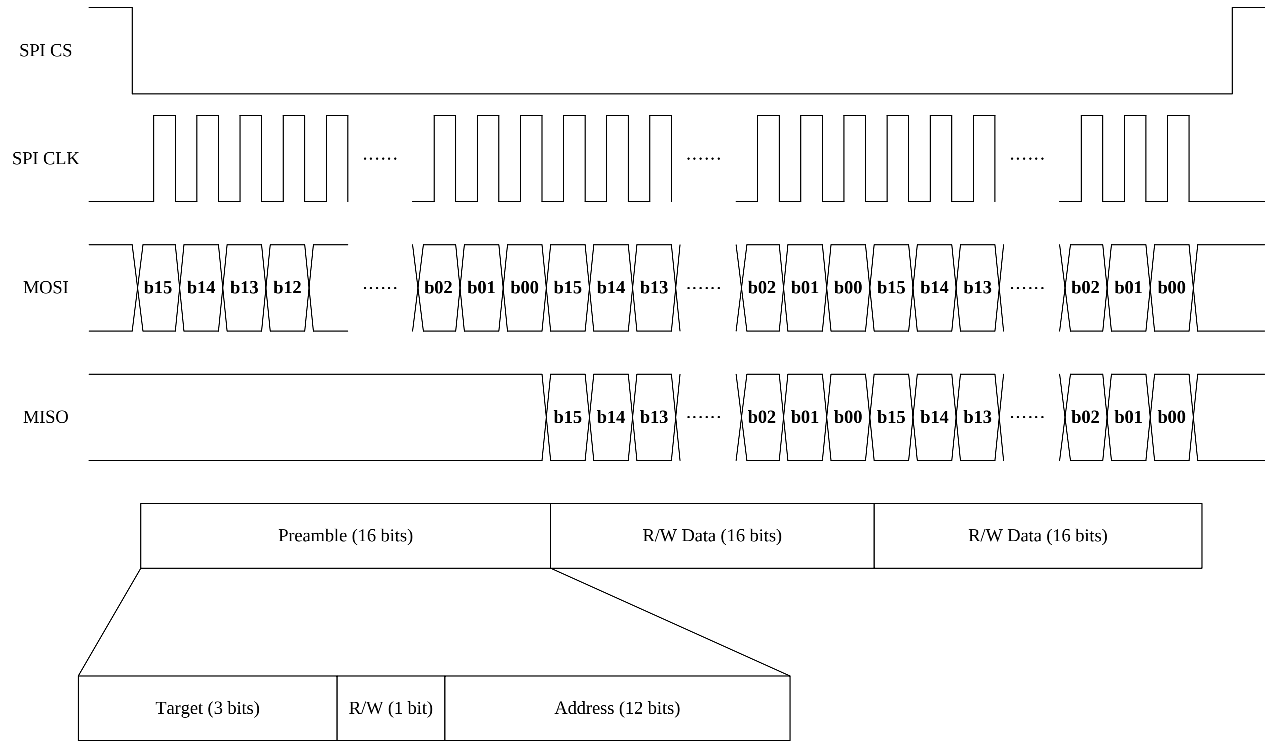
1. LCDWriteCmdCode()
2. LCDWriteData()
3. LCDWriteNData()
4. LCDReadData()
5. LCDReadNData()

Please kindly note that all of the commands/data sent via SPI interface will be converted to I80 format when IT8951 received. Therefore host programmer still have to regard some properties of I80 protocol as following:

1. Wait I80 Bus ready(HRDY) before sending command/data

   We recommend that Host may need 1 GPI pin connected to HRDY pin of IT8951.
2. 16 bits Bus width for each Write/Read transfer

SPI Format introduction:



- 16 bits base (I80 Spec)
- Send Preamble first.
- R/W bit in preamble will determine the data direction (Read or Write).
- Only the **first** 16-bits data will be preamble in each SPI cycle (SPI_CS low Period).
- Big Endian format for each Word(2 bytes) transfer
    - SPI Send Byte[0] = Word[15:8]
    - SPI Send Byte[1] = Word[7:0]

Specification of IT8951 SPI:

Suggested SPI Clock rate : 12MHZ (Max: 24MHZ)

SPI Mode : Mode 0 only (CPOL = 0, CPHA = 0)

Clock Low in idle.

Preparing Write data after falling edge, and Capture data after rising edge

Preamble for SPI to I80

Base on the SPI format, set different preamble value will convert to different I80 (M68) cycle. There are three type of cycle in I80 (M68), write command code, write data and read data. If we want to change the type of I80 (M68) cycle, you should finish this SPI cycle first, and then start another SPI cycle.

Preamble Value	I80 (M68) Cycle Type
0x6000	Write Command Code
0x0000	Write Data
0x1000	Read Data

Example

e.g.1 - Write IT8951 Register Address[0x1100] = 0x0506

1 **Send Command 0x0011 (WRITE_REG)**

1.1 Send preamble 0x6000 (command type)

1.2 Send Command 0x0011

CS to L -> MOSI Data {0x60,0x00, 0x00,0x11} -> CS to H

2 **Send Register Address 0x1100**

2.1 Send preamble 0x0000 (Data type)

2.2 Send WData 0x1100

CS to L -> MOSI Data {0x00,0x00, 0x11,0x00} -> CS to H

3 Send Write Data 0x0506

3.1 Send preamble 0x0000 (Write Data type)

3.2 Send WData 0x0506

CS to L -> MOSI Data {0x00,0x00, 0x05,0x06} -> CS to H

e.g.2 - Read IT8951 Register Address[0x1100]

1 Send Command 0x0010 (READ_REG)

1.1 Send preamble 0x6000 (command type)

1.2 Send Command 0x0010

CS to L -> MOSI Data {0x60,0x00, 0x00,0x10} -> CS to H

2 Send Register Address 0x1100

2.1 Send preamble 0x0000 (Write Data type)

2.2 Send WData 0x1100

CS to L -> MOSI Data {0x00,0x00, 0x11,0x00} -> CS to H

3 Read Data (Suppose <= Read Data = 0x0506)

3.1 Send preamble 0x1000 (Read Data type)

3.2 Read Dummy 0xXXXX

3.3 Read RData 0x0506

CS to L -> MOSI Data {0x10,0x00} -> MISO Data{0xXX,0xXX, 0x05,0x06} -> CS to H

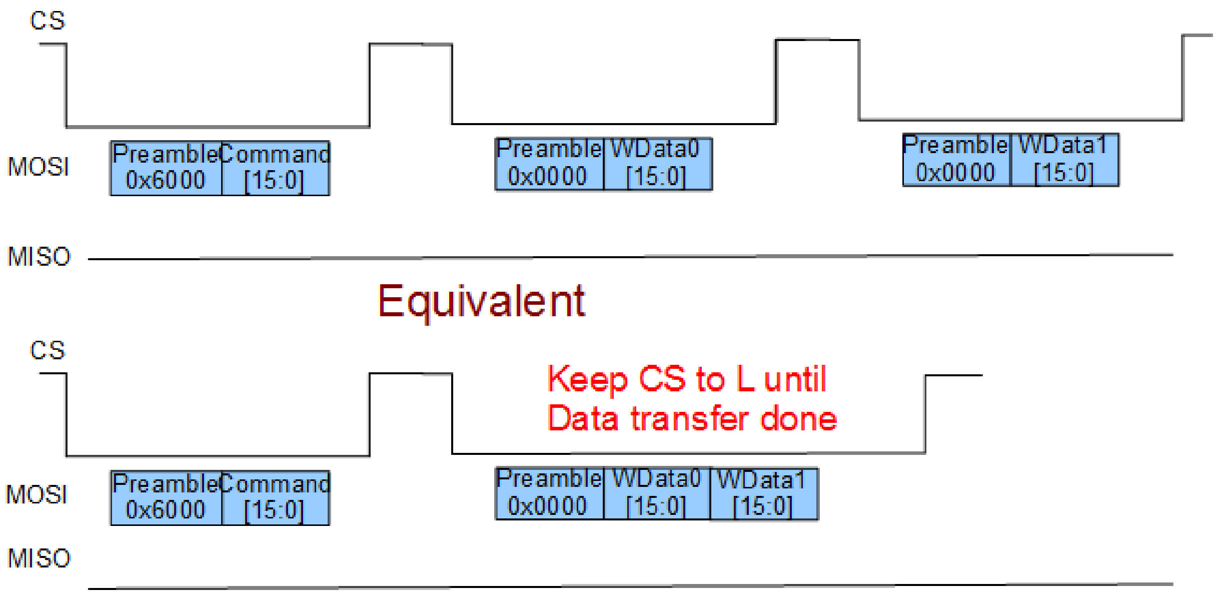
PS. In the read behavior, it needs a dummy read for each CS. The first 16-bits data are not valid data which should be ignored.

Burst Data transfer

In IT8951, it is available for Burst read/write data transfer between Host and IT8951 through interface

For example:

Write 2 words data

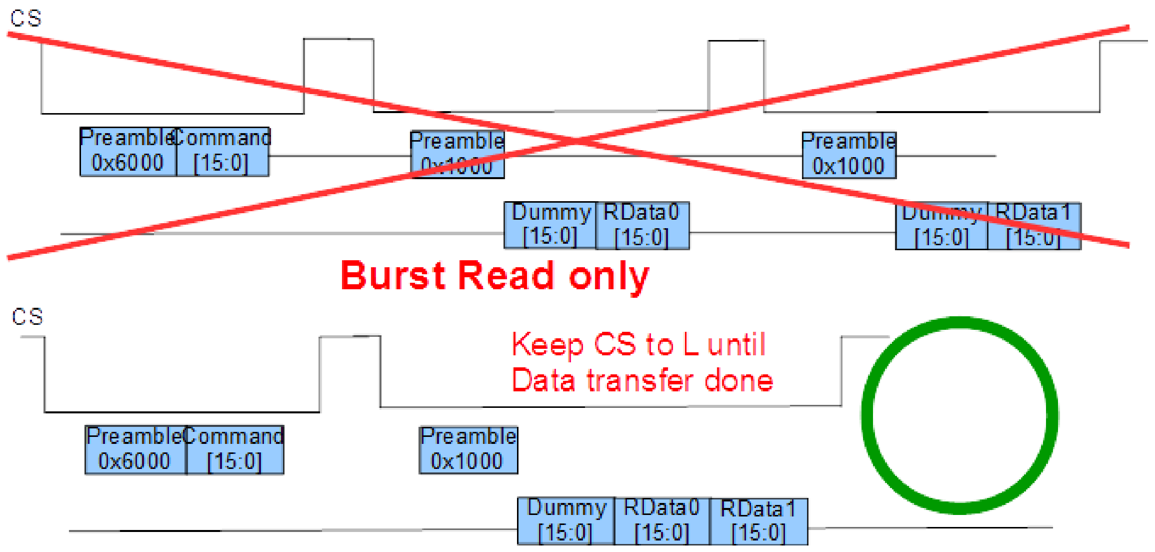


Both of above transfer type are available and equivalent in IT8951 SPI format.

Read 2 words data

However, the multiple access transfer by Single read data is not valid under IT8951 SPI specification.

Therefore, Host SPI controller must use burst read transfer only if read data >= 2;



SPI to I80 programming flow and Sample Code

If your Host MCU is little Endian type, please define the Macro as following:

```
#define __HOST_LITTLE_ENDIAN__  //Big or Little Endian for your Host platform

#ifdef __HOST_LITTLE_ENDIAN__
    //Little Endian => its needs to convert for SPI to I80
    #define  MY_WORD_SWAP(x)  ( ((x & 0xff00)>>8) | ((x & 0x00ff)<<8) )
#else
    //Big Endian => No need to convert
    #define  MY_WORD_SWAP(x)  (x)
#endif
```

Regarding to pseudo code about SPI Read/Write API Functions

They are only for reference, for more detailed description, please refer the SPI controller of your platform.

- **SPIWrite(TByte* pWBuf, TDWord Size, TByte CS)**
  - **pWBuf** – pointer of Write buffer
  - **Size** – Size of SPI Data transfer (unit: Byte)
  - **CS** –
    - 0 – CS L, it means the CS will still keep low after current data transfer
    - 1 – CS H, it means the CS will be High after current data transfer
- **SPIRead(TByte* pRBuf, TDWord Size, TByte CS)**
  - **pRBuf** – pointer of Read buffer
  - **Size** – Size of SPI Data transfer (unit: Byte)
  - **CS** –
    - 0 – CS L, it means the CS will be low (or keep low)before transferring
    - 1 – CS H, it means the CS will be High after transferring
- **In the IT8951,**

1 LCDWriteCmdCode(usCmd)

1.1 Send Preamble 0x6000 for Write command code

1.1.1 Wait for I80 Bus Ready? (IT8951 HRDY)

1.1.2 Set CS to Low

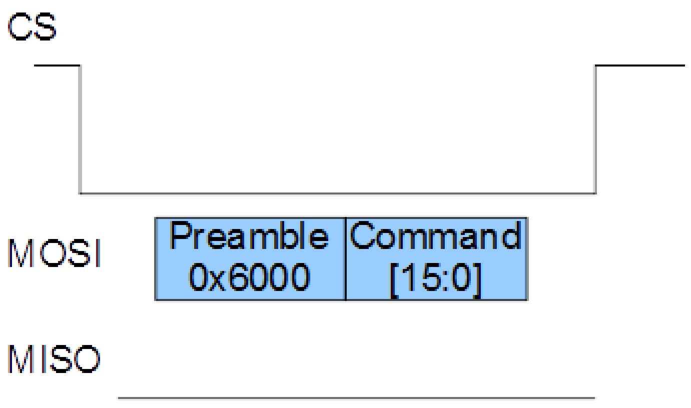
1.1.3 Send SPI Data[2] = { 0x60, 0x00 };

1.2 Send I80 Command code

1.2.1 Wait for I80 Bus Ready?

1.2.2 Send SPI Data[2] = {usCmd[15:8], usCmd[7:0]};

1.2.3 CS to High



```
void LCDWriteCmdCode (TWord wCmd)
{
    WORD wPreamble = 0;

    //Set Preamble for Write Command
    wPreamble = 0x6000;

    //Send Preamble
    wPreamble    = MY_WORD_SWAP(wPreamble);
    LCDWaitForReady();
    SPIWrite((TByte*)&wPreamble, 1*2, CS_L);

    //Send Command
    wCmd         = MY_WORD_SWAP(wCmd);
    LCDWaitForReady();
    SPIWrite((TByte*)&wCmd, 1*2, CS_H);
}
```

2 LCDWriteData(usData)

2.1 Send Preamble 0x0000 for Writing command code

2.1.1 Wait for I80 Bus Ready? (IT8951 HRDY)

2.1.2 CS to Low

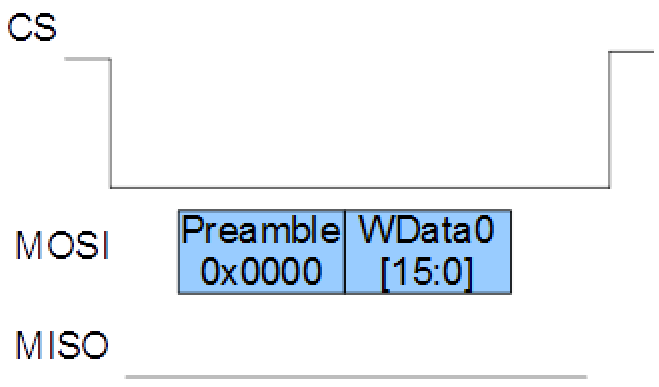
2.1.3 Send SPI Data[2] = { 0x00, 0x00 };

2.2 Send 1 Word Data

2.2.1 Wait for I80 Bus Ready?

2.2.2 Send SPI Data[2] = {usData[15:8], usData[7:0]};

2.2.3 CS to High



```
void LCDWriteData(TWord usData)
{
    WORD wPreamble  = 0;

    //set type
    wPreamble = 0x0000;

    //Send Preamble
    wPreamble = MY_WORD_SWAP(wPreamble);
    LCDWaitForReady();
    SPIWrite((TByte*)&wPreamble, 1*2, CS_L);
    //Send Data
    usData = MY_WORD_SWAP(usData);
    LCDWaitForReady();
    SPIWrite((TByte*)&usData, 1*2, CS_H);
}
```

3 LCDWriteNData(TWord* pwBuf, TDWord ulDataCnt)

This function is suitable for burst write data only. We don’t recommend

3.1 We defined Max burst Transfer Len is 2048 bytes(1024 Words) for each time

3.1.1 You have to divide into Size/1024 times to transfer if the transfer size is over 1024 words

3.1.2 or you have to polling ready for each word

3.1.3 the size of IT8951 FIFO is 2k bytes, therefore we can continued read or send data for 2kbytes(1k word)

3.2 Send Preamble 0x0000 for Write command code

In this case, the preamble 0x0000 can send once only if sending burst data.

3.2.1 Wait for I80 Bus Ready? (IT8951 HRDY)

3.2.2 Set CS to Low

3.2.3 Send SPI Data[2] = { 0x00, 0x00 };

3.3 Send N Words Data

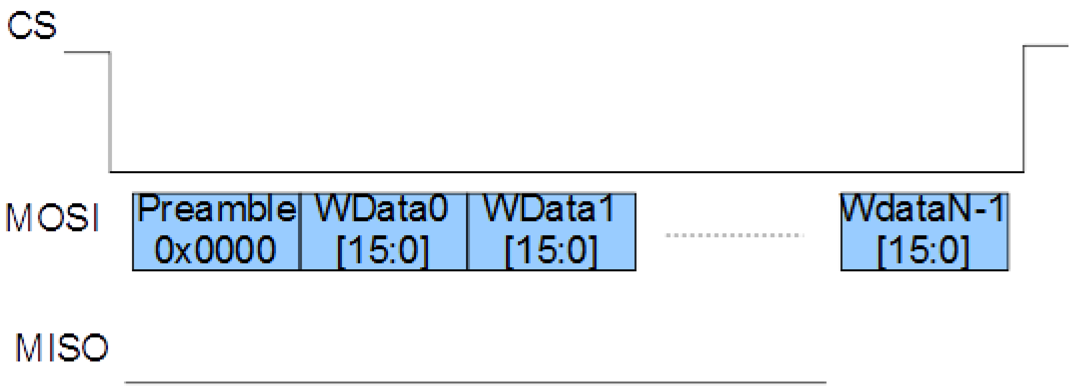
3.3.1 Wait for I80 Bus Ready?

3.3.2 Needs to Convert Data Endian? If not, go to 3.7

3.3.3 Send SPI Data for N*2 bytes = {usData[15:8], usData[7:0],…………

usDataN-1[15:8], usDataN-1[7:0]] };

3.3.4 Finished -> Set CS to High



```
void LCDWriteNData(TWord* pwBuf, TDWord ulSizeWordCnt)
{
    WORD wPreamble  = 0;
    TDWord i;
    //set type
    wPreamble = 0x0000;
    //Send Preamble
    wPreamble = MY_WORD_SWAP(wPreamble);
    LCDWaitForReady();
    SPIWrite((TByte*)&wPreamble, 1*2, CS_L);

    #ifdef __HOST_LITTLE_ENDIAN__
    //Convert Little to Big Endian for each Word
    for(i=0;i<ulSizeWordCnt;i++)
    {
        pwBuf[i] = MY_WORD_SWAP(pwBuf[i]);
    }
    #endif

    //Send Data
    LCDWaitForReady();
    SPIWrite((TByte*)pwBuf, ulSizeWordCnt*2, CS_H);
}
```

4 Word LCDReadData()

4.1 Send Preamble 0x1000 for Write command code

4.1.1 Wait for I80 Bus Ready? (IT8951 HRDY)

4.1.2 Set CS to Low

4.1.3 Send SPI Data[2] = { 0x10, 0x00 };

4.2 Read 1 Dummy Word first

4.2.1 Wait for I80 Bus Ready?

4.2.2 Get SPI Read Data[2] = {usDummy[15:8], usDummy[7:0]};

4.2.3 Read 1 Dummy Word (2bytes)

4.3 Get 1 Word Read Data

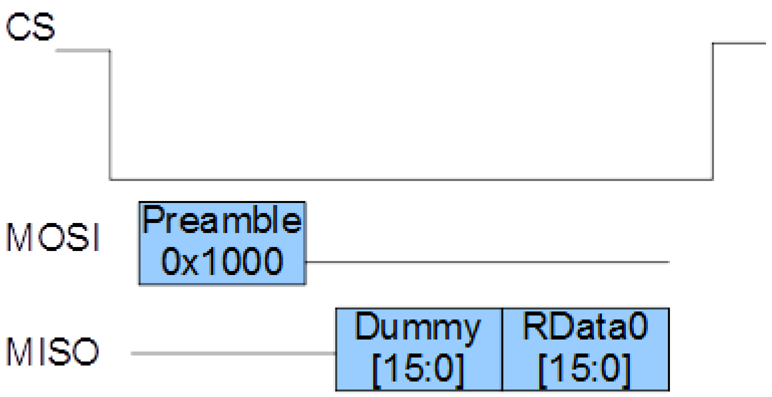
4.3.1 Wait for I80 Bus Ready?

4.3.2 Get SPI Read Data[2] = {usData[15:8], usData[7:0]};

4.3.3 CS to High

4.4 Return usData

4.4.1 Convert to Little Endian? (it depends on your host Endian type)



```
TWord LCDReadData()
{
    TWord wPreamble = 0;
   TWord wRData;
    TWord wDummy;

    //set type and direction
    wPreamble = 0x1000;

    //Send Preamble before reading data
    wPreamble = MY_WORD_SWAP(wPreamble);
    LCDWaitForReady();
    SPIWrite((TByte*)&wPreamble, 1*2, CS_L);

    //Read Dummy (under IT8951 SPI to I80 spec)
    LCDWaitForReady();
    SPIRead((TByte*)&wDummy, 1*2, CS_L ); //CS Keep L

    //Read Data
    LCDWaitForReady();
    SPIRead((TByte*)&wRData, 1*2, CS_H);

    wRData = MY_WORD_SWAP(wRData);
    return wRData;
}
```

5 Word LCDReadNData(TWord* pwBuf, TDWord ulWordDataCnt)

5.1 We define Max burst Transfer Len is 2048 bytes(1024 Words)

5.1.1 You have to divide into Size/1024 times to transfer if the transfer size is over 1024 words

5.1.2 or you have to polling ready for each word

5.1.3 the size of IT8951 FIFO is 2k bytes, therefore we can continued read or send data for 2kbytes(1k word)

5.2 Send Preamble 0x1000 for Write command code

5.2.1 Wait for I80 Bus Ready? (IT8951 HRDY)

5.2.2 Set CS to Low

5.2.3 Send SPI Data[2] = { 0x10, 0x00 };

5.3 Read 1 Dummy Word first

5.3.1 Wait for I80 Bus Ready?

5.3.2 Get SPI Read Data[2] = {usDummy[15:8], usDummy[7:0]};

5.3.3 Read 1 Dummy Word (2bytes)

5.4 Get N Words Read Data

5.4.1 Wait for I80 Bus Ready?

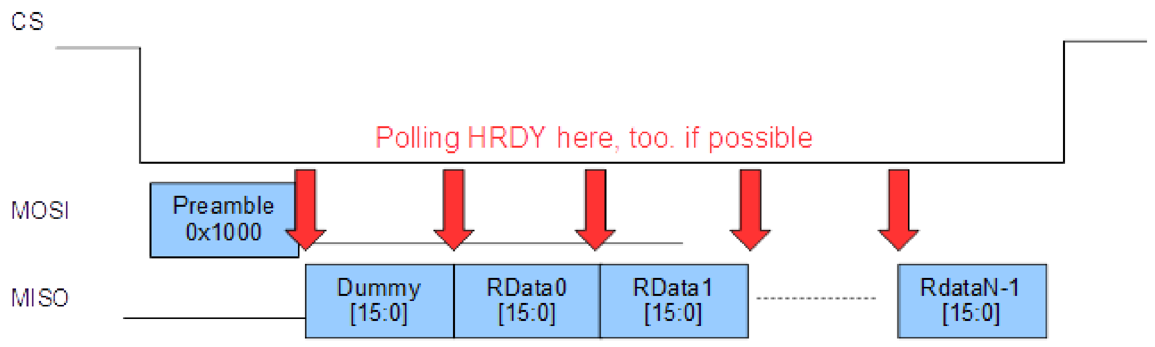
5.4.2 Get SPI Read Data[N] = {usData0[15:8], usData0[7:0],……, usDataN-1[15:8], usDataN-1[7:0]};

5.4.3 CS to High

5.5 Convert Endian from Big to Little all ReadData (Little Endian only)

5.5.1 Convert to Little Endian? (it depends on your host Endian type)

5.5.2 **Convert Endian from Big to Little all ReadData**



```
TWord LCDReadNData(TWord* pwBuf, TDWord ulSizeWordCnt)
{
    TWord wPreamble = 0;
   TWord wRData;
    TWord wDummy;
    TDWord i;

    //set type and direction
    wPreamble = 0x1000;

    //Send Preamble before reading data
    wPreamble = MY_WORD_SWAP(wPreamble);
    LCDWaitForReady();
    SPIWrite((TByte*)&wPreamble, 1*2, CS_L);

    //Read Dummy (under IT8951 SPI to I80 spec)
    LCDWaitForReady();
    SPIRead((TByte*)&wDummy, 1*2, CS_L );//CS Keep L

    //Read N Data
    //in this case, we recommend host programmer can polling LCDWaitForReady for
    //every 2 bytes(1 word)if possible to your SPI controller
    LCDWaitForReady();
    SPIRead((TByte*)pwBuf, ulSizeWordCnt *2, CS_H);

    //Convert Endian (depends on your host)
    for(i=0;i< ulSizeWordCnt ; i++)
    {
        pwBuf[i] = MY_WORD_SWAP(pwBuf[i]);
    }
}
```

Appendix III – Programming guide for I2C to I80 interface

(IT8951 CX/DX only)

In IT8951 CX1 version, we added new feature that Host can send I80 command through I2C interface, Programmer should modify the following basic functions and Implement I2C driver which depends on your host

1. LCDWriteCmdCode()

2. LCDWriteData()

3. LCDWriteNData()

4. LCDReadData()

5. LCDReadNData()

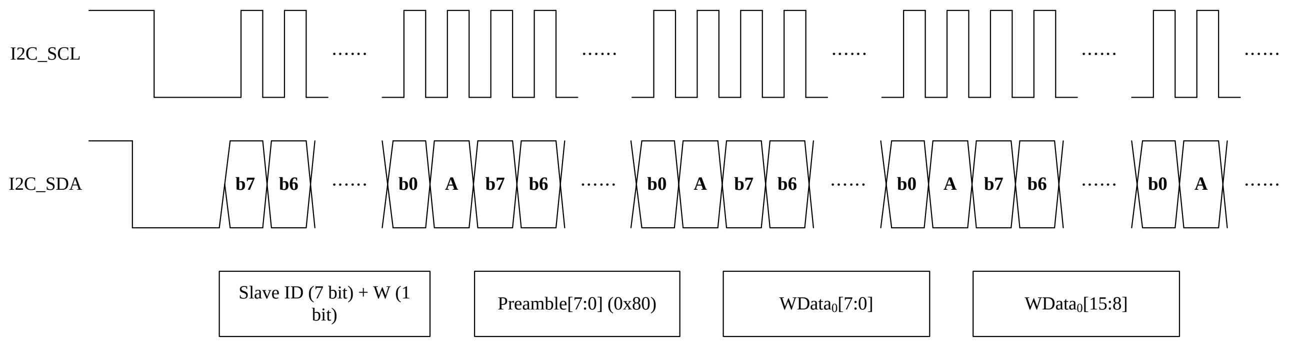
Please kindly note that all of the commands/data sent via I2C interface will be converted to I80 format when IT8951 received. Therefore the host programmer still has to regard some properties of I80 protocol as following:

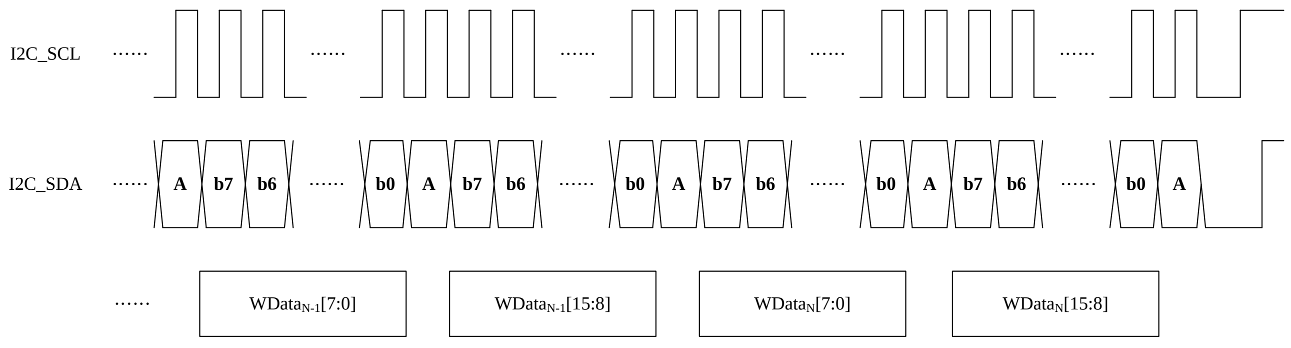
1. Wait I80 Bus ready before sending command/data

   We suggest that Host may need 1 GPI pin to connect to HRDY pin of IT8951.

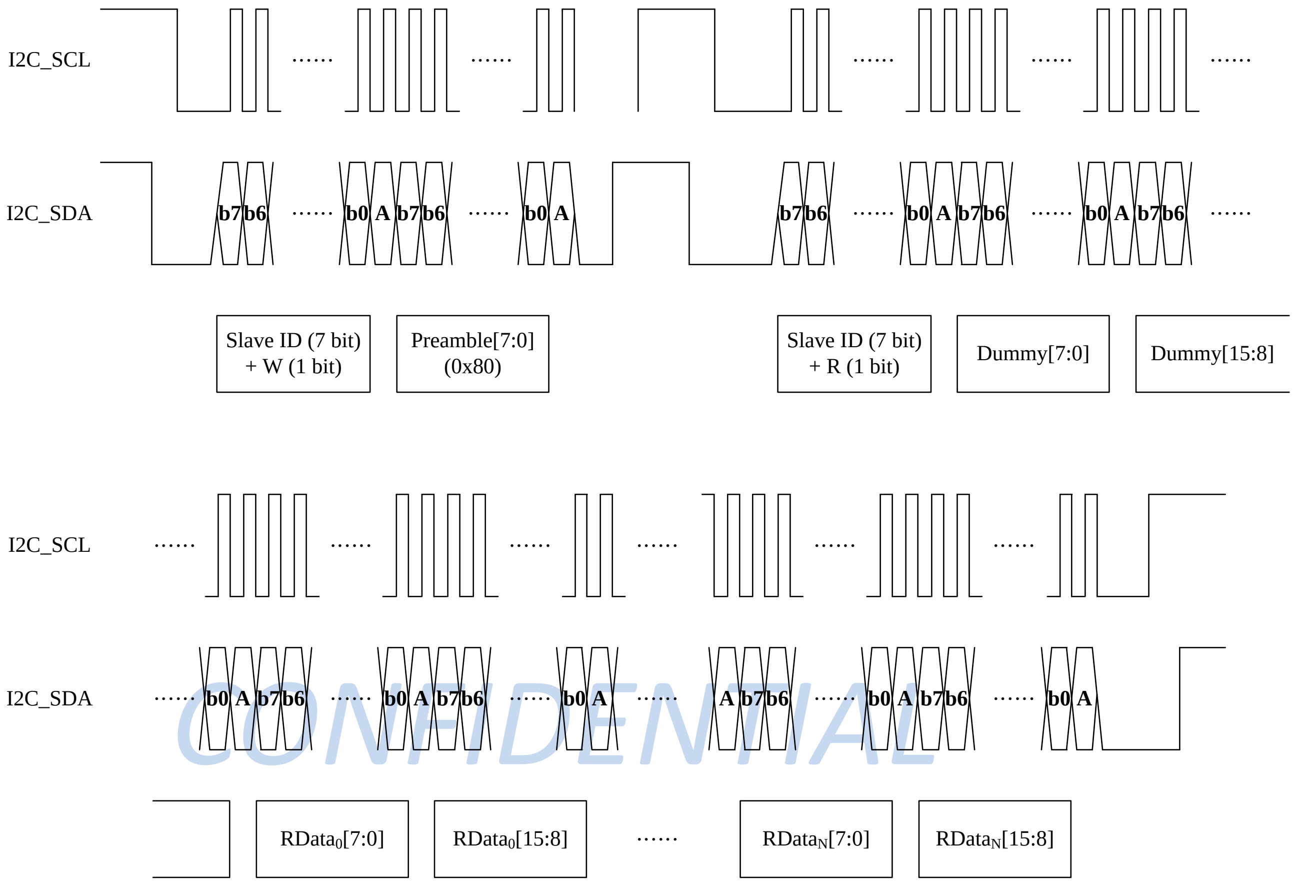
2. 16 bits Bus width for each Write/Read transfer

I2C Write Format





I2C Read Format


I2C_SCL
I2C_SDA
b7 b6 …… b0 A b7 b6 …… b0 A
b7 b6 …… b0 A b7 b6 …… b0 A b7 b6 ……
Slave ID (7 bit) + W (1 bit)
Preamble[7:0] (0x80)
Slave ID (7 bit) + R (1 bit)
Dummy[7:0]
Dummy[15:8]
I2C_SCL
I2C_SDA
RData0[7:0]
RData0[15:8]
RDataN[7:0]
RDataN[15:8]

1. R/W Data are16 bits base
2. After Slave ID + R/W, send Preamble (8 bit) first.
3. Only the first 8 bits data will be preamble in each I2C cycle (Until I2C Stop).
4. Preamble for I2C/I80 definition

Preamble Value	I80 (M68) Cycle Type
8‘h00	Command
8’h80	Data

5. IT8951 HW platform Configure Setting

Slave ID	TEST CFG{2,1,0}
7‘h46	3’b110
7’h35	3’b111

```
//--------------------------------------------------------------------
//                          I2C Inteface
//--------------------------------------------------------------------
#define I2C_SLAVE_ID    0x46  //7-bits

#define I80_I2C_CMD_TYPE_CMD     0x00 //Preamble of Command
#define I80_I2C_CMD_TYPE_DATA    0x80 //Preamble of Data
```

```
//-------------------------------------------------------
//Pseudo Code - Basic i2c Write function
//-------------------------------------------------------
void i2c_master_tx(TByte ucSlaveID, TByte Premable, TByte ulSize, TByte* pWBuf)
{
    int i;

    //0. Start
    //1. Send: (SlaveID << 1)| 0 for Write
    i2c_send_byte(ucSlaveID << 1 | 0);

    //2. Send Premable
    i2c_send_byte(Premable);

    //3. Send WBuf Data for ulSize bytes
    for(i=0;i<ulSize;i++)
    {
        i2c_send_byte(pWBuf[i]);
    }

    //4. Stop
}
```

```
//-------------------------------------------------------
//Pseudo Code - Basic i2c Read function
//-------------------------------------------------------
void i2c_master_rx(TByte ucSlaveID, TByte Premable, TByte ulSize, TByte* pRBuf)
```

```
{
    int i;
    TByte ucDummy[2];

    //0. Start
    //1. Send: (SlaveID << 1)| with Write
    i2c_send_byte(ucSlaveID << 1 | 0);

    //2. Send Premable
    i2c_send_byte(Premable);

    //3. Send: (SlaveID << 1) with Read
    i2c_send_byte(ucSlaveID << 1 | 1);

    //4. Read Dummy (1 Word = 2-bytes) and ignored
    ucDummy[0] = i2c_recv_byte();
    ucDummy[1] = i2c_recv_byte();

    //5. Recieve Data and store to Read Buffer
    for(i=0;i<ulSize;i++)
    {
        pRBuf[i] = i2c_recv_byte();
    }

    //6. Stop
}
```

1 LCDWriteCmdCode(usCmd)

1.1 Wait for I80 Bus Ready? (IT8951 HRDY)

1.2 Send I2C Start bit

1.3 Send I2C Slave ID with Write

1.3.1 Send ((0x46 << 1) | 0x00)

1.4 Send Preamble 0x00 for Write command code

1.4.1 Send I2C Data[1] = {0x00 };

1.5 Send I80 Command code

1.5.1 Send I2C Data[2] = {usCmd[7:0], usCmd[15:8]};

1.6 Send I2C Stop bit

```
//---------------------------------------------------------------------
//  Write Command code
//---------------------------------------------------------------------
void LCDWriteCmdCode (TWord wCmd)
{

    LCDWaitForReady();

    wCmd = SWAP_16(wCmd);//I2C to I80 => Little Endian format
    //Send Preamble and Command
   i2c_master_tx(I2C_SLAVE_ID, I80_I2C_CMD_TYPE_CMD, 2, (TByte*)&wCmd);

}
```

2 LCDWriteData(usData)

2.1 Wait for I80 Bus Ready? (IT8951 HRDY)

2.2 Send I2C Start bit

2.3 Send I2C Slave ID with Write

2.3.1 Send Byte = ((0x46 << 1) | 0x00)

2.4 Send Preamble 0x80 for Write data

2.4.1 Send Byte = { 0x80 };

2.5 Send usData

2.5.1 Send I2C Byte[0] = usData[7:0]

2.5.2 Send I2C Byte[1] = usData[15:8]

2.6 Send I2C Stop bit

```
//-----------------------------------------------------------------------
//  Write 1 Word Data(2-bytes)
//-----------------------------------------------------------------------
void LCDWriteData(TWord usData)
{
    LCDWaitForReady();

    usData = SWAP_16(usData);//I2C to I80 => Little Endian format
    //Send Preamble and Data
   i2c_master_tx(I2C_SLAVE_ID, I80_I2C_CMD_TYPE_DATA, 2, (TByte*)&usData);

}
```

3 LCDWriteNData(Wbuffer, N)

3.1 Wait for I80 Bus Ready? (IT8951 HRDY)

3.2 Send I2C Start bit

3.3 Send I2C Slave ID with Write

3.3.1 Send ((0x46 << 1) | 0x00)

3.4 Send Preamble 0x80 for Write data

3.4.1 Send I2C Byte = { 0x80 };

3.5 Send WBuf[] for N Words (Nx2 Bytes)

3.5.1 Send I2C WBuf[0]

=> Byte[0] = WBuf0[7:0]

=> Byte[1] = WBuf0 [15:8]};

3.5.2 Send I2C WBuf[1]

3.5.3 Send I2C WBuf[2];

3.5.4 ………………………

3.5.5 Send I2C WBuf[N-1]

3.6 Send I2C Stop bit

```
//-----------------------------------------------------------------------
//  Burst Write Data
```

```
//----------------------------------------------------------------------
void LCDWriteNData(TWord* pwBuf, TDWord ulSizeWordCnt)
{

    TDWord i;

    #ifdef __HOST_LITTLE_ENDIAN__
    //Convert Little to Big Endian for each Word
    for(i=0;i<ulSizeWordCnt;i++)
    {
        pwBuf[i] = MY_WORD_SWAP(pwBuf[i]);
    }
    #endif

    //Send Data
    LCDWaitForReady();
    //Send Preamble and Data
   i2c_master_tx(I2C_SLAVE_ID, I80_I2C_CMD_TYPE_DATA, ulSizeWordCnt*2,
(TByte*)pwBuf);

}
```

4 LCDReadData()

4.1 Wait for I80 Bus Ready? (IT8951 HRDY)

4.2 Send I2C Start bit

4.3 Send I2C Slave ID with Write

4.3.1 Send ((0x46 << 1) | 0x0)

4.4 Send Preamble 0x80 for Write data

4.4.1 Send I2C Data[1] = {0x80 };

4.5 Send I2C Slave ID with Read

4.5.1 Send ((0x46 << 1) | 0x1)

4.6 Read 1 usData (2 bytes)

4.6.1 usData[7:0] <= I2C Read Byte[0]

4.6.2 usData[15:8] <= I2C Read Byte[1]

4.7 Send I2C Stop bit

```
//------------------------------------------------------------------
// Read 1 Word Data
//------------------------------------------------------------------
TWord LCDReadData()
{
    TWord wRData;
     TWord wDummy;


     LCDWaitForReady();

     //Read 1 16-bits Data
     i2c_master_rx(I2C_SLAVE_ID, I80_I2C_CMD_TYPE_DATA, 2, (TByte*)&wRData);

    wRData = MY_WORD_SWAP(wRData);//Endian Convert if need

     return wRData;
}
```

5 LCDReadData()

5.1 Wait for I80 Bus Ready? (IT8951 HRDY)

5.2 Send I2C Start bit

5.3 Send I2C Slave ID with Write

5.3.1 Send ((0x46 << 1) | 0x0)

5.4 Send Preamble 0x80 for Write data

5.4.1 Send I2C Data[1] = {0x80 };

5.5 Send I2C Slave ID with Read

5.5.1 Send ((0x46 << 1) | 0x1)

5.6 Read N usData (2 bytes)

5.6.1 Read usData0

5.6.1.1 usData[7:0] <= I2C Read Byte[0]

5.6.1.2 usData[15:8] <= I2C Read Byte[1]

5.6.2 Read usData1

5.6.3 Read usData2

5.6.4 ………………

5.6.5 Read usDataN-1

5.7 Send I2C Stop bit

```
//--------------------------------------------------------------------
//  Read Burst N words Data
//--------------------------------------------------------------------
TWord LCDReadNData(TWord* pwBuf, TDWord ulSizeWordCnt)
{

    TDWord i;

    LCDWaitForReady();

    //Read 1 16-bits Data
    i2c_master_rx(I2C_SLAVE_ID, I80_I2C_CMD_TYPE_DATA, ulSizeWordCnt*2,
(TByte*)pwBuf);

    //Convert Endian (depends on your host)
    for(i=0;i< ulSizeWordCnt ; i++)
   {
      pwBuf[i] = MY_WORD_SWAP(pwBuf[i]);
   }
}
```