Locosto LCD Driver Customization Guidelines

Application note

Contents

Figures

Tables

WARNING:

1 Introduction

The purpose of this App-Note is to identify and mark the changes required in the LCD Driver in TCS 3.2 in order to support the new hardware modules in the Locosto.

In the reference design LCD Controller HD66774+HD66772 and Philips panel LPH8754 is used. The scope of this document is to describe the changes required to TCS 3.2 LCD controller driver to interface with a different LCD module other than the one used in the reference design.

The document is divided into three logical parts.

1. First part describes current LCD features, current LCD driver architecture, and hardware connection between Locosto and LCD controller.
2. Second part describes about Data structures used, the API summary as well the impact on API to adapt new LCD hardware.
3. Changes identified to adapt new hardware with respect to API as well Data structures.

1.1 Current LCD Overview

The liquid crystal display (LCD) interface used in TCD 3.2 reference design (RD) connects an external color graphical controller to the digital base band (DBB) device (LOCOSTO). This 8-bit parallel interface is compliant with the 8-bit 6800-series and 8086-series parallel standard to support a large range of LCD displays available on the market.

The configuration and data information are transferred from the microprocessor unit (MPU) to the LCD interface through the TI peripheral bus (TIPB) by 16-bit words up to a rate of 52 Mwords/s. The control and data information are transferred from the LCD interface to the external LCD controller at a speed that is a fraction (1, 2, 4, or 8) of the 13-MHz GSM clock. Data are transferred in 8-bit words.

This interface targets LCD device with the following features:

- Color LCD screens (passive or active matrix)
- 8-bit 6800-series or 8086-series parallel interface
- 16-bit (RGB 656) or 24-bit (RGB 888) color
- Up to QVGA (320 x 240) format

1.2 Current LCD Main Features:

The main features of the LCD interface are:

- _ Receive and transmit capability
- _ Two interface modes: 6800 or 8086
- _ 16-bit word width between memory and the LCD interface
- _ 52 M words/s transfer rate between memory and the LCD interface
- _ Speed of data and control transfer is a fraction (1, 2, 4, or 8) of the 13-MHz GSM clock
- _ Receive and transmit interrupts
- _ FIFO buffer used for adapting transfer rate
- _ Two-port FIFO of 128 x 16-bit words
- _ FIFO used in transmission data or commands
- _ Register used in reception

Main LCD connection with LOCOSTO

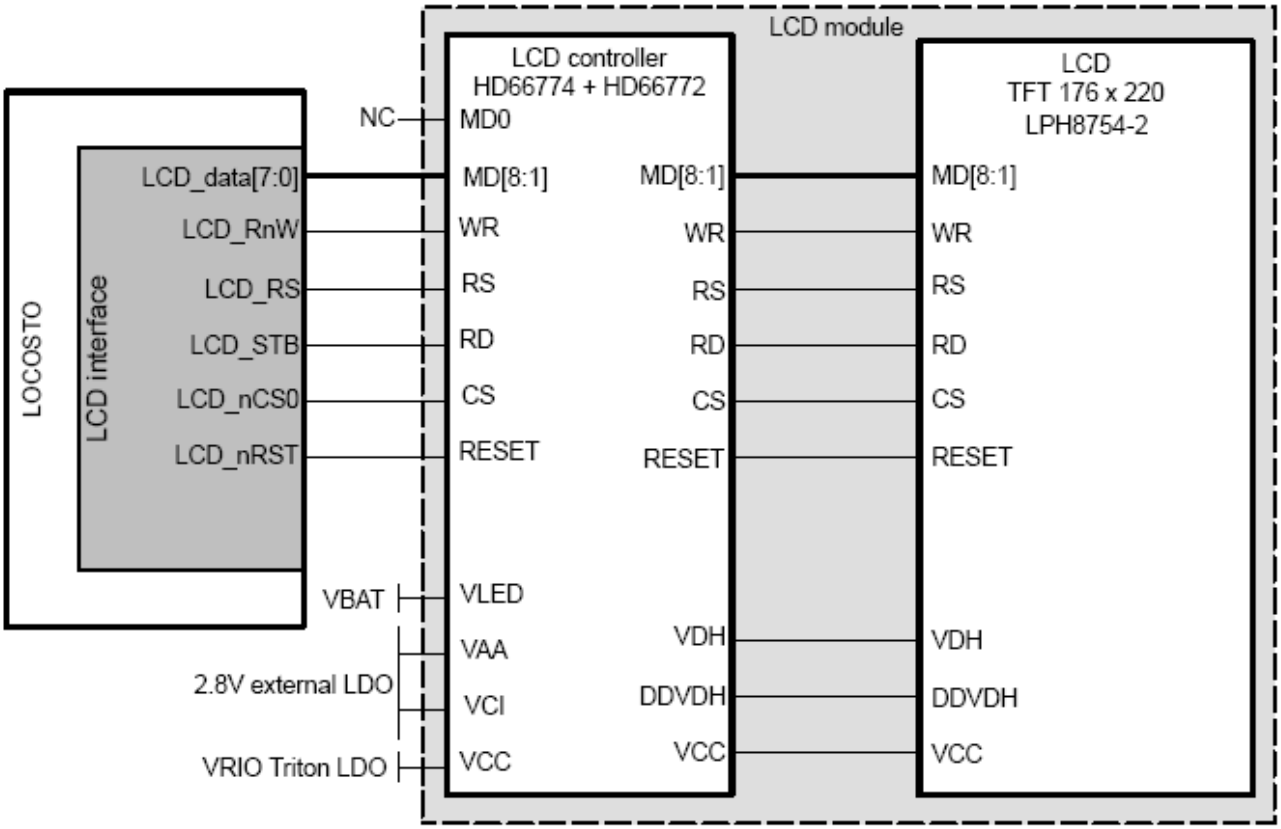

Figure 1. Lists the signals between the LOCOSTO device and LCD controller

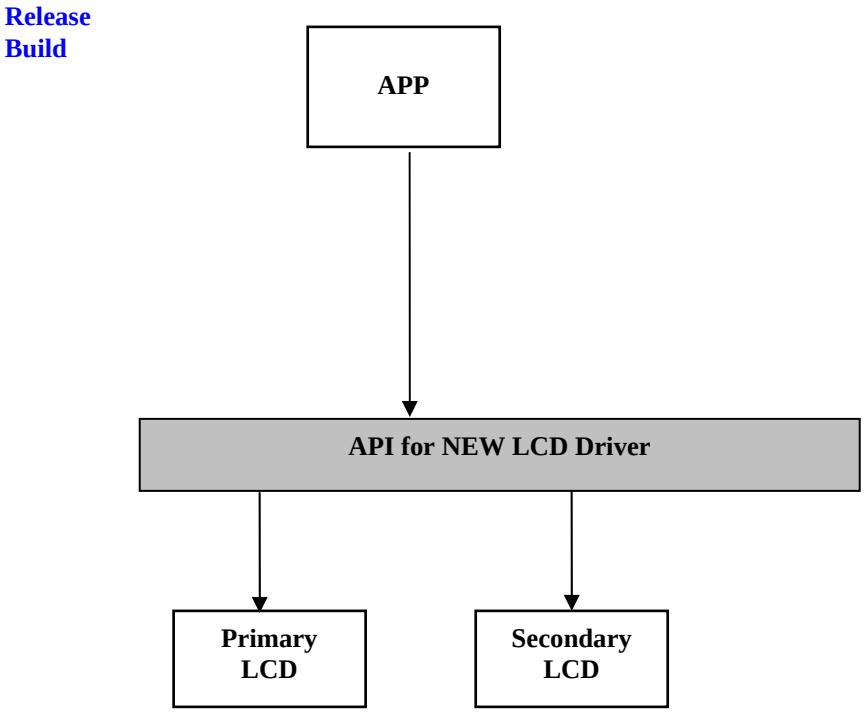

Figure 2. Figure 2: Architecture for LCD Driver

2 Locosto LCD Interface summary and data structure details

This section will focus on API summary and Data structures used.
2.1. Locosto driver API summary
2.2. Locosto LCD driver Data Structure details

2.1 Locosto LCD driver API summary :

The functions in LCD are divided into four categories.

1. Top level framework API – These are the API's which are exposed to the framework. These will be the entry-point for the driver. These API will call to one or More than one Top level API. These API's are abstracted from hardware knowledge. These API will be low impact /no impact API.

2. Top level API – These are the API's which are exposed to the application and are abstracted from hardware knowledge. These API will call the other functions in the LCD driver, so these are identified as API with low impact / no impact.

3. High level API: These are the API's which are internal to the LCD driver. These are called by Top level API, and will make request to low level API. These API either have little information regarding the hardware or doesn't have. These API's are identified as low impact /no impact API's.

4. Low level API: These are the API which requires the hardware knowledge and are tightly coupled with underneath hardware. Any change in hardware, requires change in these functions accordingly. These API's are identified as high level impact APIs

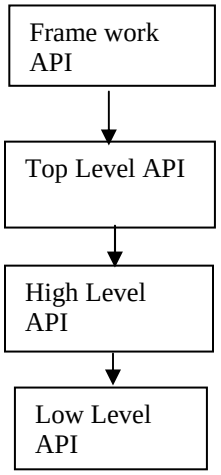

Figure 3. API structure in the LCD Driver

Definition of the terminologies used

High impact: Most part of the function needed to be rewritten with the change in the hardware.

Medium impact: Some part of the function may need to be rewritten with the change in the hardware.

Low impact: minimal changes may be required with change in the hardware. Like change in parameters that are passed to the function

No impact: No changes are required with the change in the hardware.

Sr No	**Type**	**Name**	**File Name**	**Impact**	**Remark**
1	Top Level API	lcd_initialization	lcd_manager.h	Low	If customer decides to use only one LCD then, DISPLAY_MAIN_LCD should be select by default.
2	Top Level API	lcd_display	lcd_manager.h	No	
3	Top Level API	lcd_control	lcd_manager.h	Low	For new hardware if different operations are required then add required commands in the Command union
4	High Level API	lcd_pri_if_init	lcd_interface.c	No	
5	High Level API	lcd_pri_if_display	lcd_interface.c	No	
6	High Level API	lcd_pri_if_control	lcd_interface.c	No	
7	Top Level Framework API	pei_monitor	lcd_pei.c	No	These are the framework APIs and are hardware independent. So need not to change these APIs.
8	Top Level Framework API	pei_config	lcd_pei.c	No	Hardware independent function.
9	Top Level Framework API	pei_timeout	lcd_pei.c	No	Hardware independent function.
10	Top Level Framework API	pei_signal	lcd_pei.c	No	Hardware independent function.
11	Top Level Framework API	pei_exit	lcd_pei.c	No	Hardware independent function.
12	Top Level Framework API	pei_primitive	lcd_pei.c	No	Hardware independent function.
13	Top Level Framework API	pei_run	lcd_pei.c	No	Hardware independent function.
14	Top Level Framework API	pei_init	lcd_pei.c	No	Hardware independent function.
15	Top Level Framework API	pei_create	lcd_pei.c	No	Hardware independent function.
16	Top Level Framework API	lcd_start	lcd_pei.c	No	Hardware independent function.
18	Top Level Framework API	lcd_stop	lcd_pei.c	No	Hardware independent function.
19	Top Level	lcd_kill	lcd_pei.c	No	Hardware independent function.

	Framework API				
20	Top Level Framework API	lcd_init	lcd_pei.c	No	Hardware independent function.
21	Top Level Framework API	lcd_get_info	lcd_env.c	No	Hardware independent function.
22	Top Level Framework API	lcd_set_info	lcd_env.c	No	Hardware independent function.
23	Top Level Framework API	lcd_start	lcd_env.c	No	Hardware independent function.
24	Top Level Framework API	lcd_stop	lcd_env.c	No	Hardware independent function.
25	Top Level Framework API	lcd_kill	lcd_env.c	No	Hardware independent function.
26	Top Level Framework API	lcd_init	lcd_env.c	No	Hardware independent function.
27	Low Level API	lcd_pwr_interface	lcd_pwr.c	High	Function is hardware dependent. It sets the control register of the LCD controller, function need to change with new hardware.
28	High Level API	Lcd_Vote_DeepSleepStatus	lcd_pwr.c	Low	This function updates the Sleep status and is hardware independent. Need to verify the calling sequence.
29	Low Level API	lcd_parallel_config	lcd_transport	High	This function is used to initialize the LCD .It also initializes the LCD interface and the LCD controller. This function need to be changed
30	Low Level API	f_lcd_if_poll_write	lcd_transport	High	The writing sequence may be different for different controller
31	Low Level API	f_lcd_if_dma_enable	lcd_transport	Low	This function grabs the bus and enables the DMA. Need to be modified if no other device is there on bus
32	Low Level API	f_lcd_if_dma_disable	lcd_transport	Low	This function disables the DMA capabilities and releases the bus. Need to be modified if no other device is there on bus

33	Low Level API	f_lcd_if_set_cs_and _data_type	lcd_transport	High	Updates LCD_CNTL_REG to set data type and read or write access
34	Low Level API	lcd_parallel_display	lcd_transport	High	Need to be Changed
35	Low Level API	r2d_dma_callback	lcd_transport	Low	Need to check the Impact

Table 1. LCD API impact summary to adapt new LCD hardware

2.2 Locosto LCD Driver Data Structure details

Types definitions and constants

2.2.1 T_LCD_SELECT

T_LCD_SELECT selects a specific LCD

Synopsis:
```
typedef enum {
        DISPLAY_MAIN_LCD,
        DISPLAY_SUB_LCD
} T_LCD_SELECT;
```

2.2.2 T_LCD_ENDIAN

T_LCD_ENDIAN selects the endianness to be used for the Pixel data

Synopsis:
```
typedef enum {
        LITTLE_ENDIAN,
        BIG_ENDIAN
} T_LCD_ENDIAN;
```

2.2.3 T_LCD_PIXFORMAT

T_LCD_PIXFORMAT selects the Pixel format for the pixel data

Synopsis:
```
typedef enum {
        RGB565,
        RGB666,
        RGB888
} T_LCD_PIXFORMAT;
```

2.2.4 T_LCD_ORIENTATION

T_LCD_ORIENTATION selects the orientation of the LCD

Synopsis:
```
typedef enum {
        HORIZONTAL,
        VERTICAL
} T_LCD_ORIENTATION;
```

2.2.5 T_LCD_REFCONTROL

T_LCD_REFCONTROL selects if LCD refresh is enabled or disabled

Synopsis:

```
typedef enum {
                REF_ENABLED,
                REF_DISABLED
        } T_LCD_REFCONTROL;
```

2.2.6 T_LCD_COMMAND

T_LCD_COMMAND Command to be issued to the LCD driver

Synopsis:

```
typedef enum {
                LCD_CLEAR,
                LCD_DISPLAYON,
                LCD_DISPLAYOFF,
                LCD_GETCONFIG,
                LCD_SETCONFIG
        } T_LCD_COMMAND;
```

2.2.7 lcd_fb_coordinates

Lcd_fb_coordinates selects the Pixel co-ordinates to be refreshed

Synopsis:

```
typedef struct {
                Uint16 start_x;
                Uint16 start_y;
                Uint16 end_x;
                Uint16 end_y;

        } lcd_fb_coordinates;
```

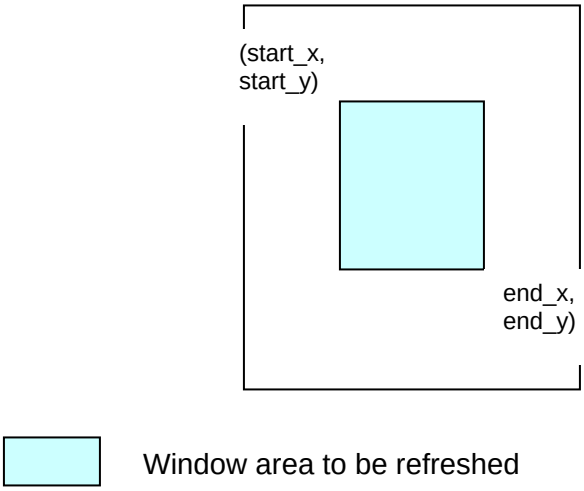

Figure 4. Figure 4: lcd_fb_coordinates

2.2.8 lcd_configParams

lcd_configParams	Parameters for LCD configuration. These are the parameters which could be configured from the application.

Synopsis

```
typedef  struct {
 Uint16                  height     /* height of the display panel */
 Uint16                  width      /* width of the display panel */
 T_LCD_ORIENTATION       orientation /* orientation of the LCD */
 T_LCD_PIXFORMAT         pixel_format;   /* RGB format */
 T_LCD_ENDIAN            endianness              /* Endianness of the pixel data */
 T_LCD_REFCONTROL        refresh_control    /* refresh control */
                } lcd_configparams;
```

2.2.9 lcd_tuningtable

lcd_TuningTable	parameters of the tuning table. This table gives the whole list of parameters which the application can configure as well as the read-only parameters which are controlled at the driver level.

Synopsis

```
Typedef struct {
bool            partial_update; /* does it support windowing or partial update of the LCD framebuffer*/
bool            OSD; /* does it support OSD (On Screen Display */
bool            dedicated_dma; /* is there dedicated dma */
lcd_configparams *p_lcd_configparams

}lcd_tuningtable;
```

2.2.10 #define directives used

```
#define LCD_HEIGHT 220                  // Change LCD Height with new hardware
#define LCD_WIDTH 175                   // Change LCD Width with new hardware
#define LCD_PIXEL_FORMAT RGB565
#define LCD_REFRESH_PERIOD 40
```

3 Adoption of different LCD Hardware

This section will focus on changes required to adapt to new hardware:

3.1. Functions requiring modification to adapt to new hardware.

3.2. Data structure requiring modification to adapt new hardware.

3.1 Functions requiring modification to adapt to new hardware:

We are listing here the functions which have probable impact to adhere to new hardware.

3.1.1 lcd_initialization

```
T_RV_RET lcd_initialization (T_LCD_SELECT sel  //selects a specific LCD  main or
                                    Sub
                            )
```

Description

This function initialises the LCD display and the LCD controller driver. This API should be called before any other functions in this driver.

Parameters

- **sel** specifies whether this API is intended for main or sub-LCD.

Immediate Return

- **T_RV_RET**

Possible changes required

If the customer is using only one LCD then parameter is overhead. Still to adhere to the architecture we insist to keep API unchanged and select Main LCD.

3.1.2 lcd_control

```
T_RV_RET  lcd_control(T_LCD_SELECT sel, // selects a specific LCD  main or Sub
                 T_LCD_COMMAND command, // Command to be issued to the LCD driver
                 void *p_cmd_param     //  void pointer to send param if req

                            )
```

Description

This is a generic API which could be further scaled for any additional commands which might come up later. Currently added commands are listed below:

Command Description

- **LCD_GETCONFIG** Need to pass a structure pointer of type “lcd_tuningtable” to get the configuration items.
- **LCD_SETCONFIG** Need to pass a structure pointer of type “lcd_configparams” to set the configuration items.
- **LCD_DISPLAYON** Only command is sufficient. Display is switched ON.
- **LCD_DISPLAYOFF** Only command is sufficient. Display is switched OFF.
- **LCD_CLEAR** Only command is sufficient. Contents of the LCD are cleared.

Parameters

- **sel** Specifies whether this API is intended for main or sub-LCD.
- **Command** Command that can be given to the LCD driver for eg., clear, Display on, Display OFF, etc.
- **p_cmd_param** Structure to pass parameters if required for the commands.

Immediate Return

- **T_RV_RET**

Possible changes required

To add any additional command which might be required to support new hardware. The available commands should support most of the LCD's.

Example:

If LCD hardware has separate control register for ICON display
LCD_DISPLAY_ICON_ON This will enable the ICON display
LCD_DISPLAY_ICON_OFF This will disable the ICON display

3.1.3 lcd_pwr_interface

```
UINT8 lcd_pwr_interface(UINT8 command) //  Sleep, wakeup or Clock enable
```

Description

This function is used to set Clock Mask command, Sleep command, Wake up command.

Parameters

- **Command** Sleep, wakeup or Clock enable. Specifies on of these command

Immediate Return

- **UINT8**

Possible changes required

If the customer is using new hardware then depending on the new hardware need to set the required state to control register.

Example:

In LCD_ACTIVE command enable clock and come out of power saving mode. Incase of DISPLAY_OFF or CLOCK_OFF (If the hardware supports any power saving mode) enter into power saving mode.

3.1.4 lcd_parallel_config

```
ELCD_PAR_RET lcd_parallel_config( void )
```

Description

This function is used to initialize the LCD manager. It also initializes the LCD interface and the LCD controller.

Parameters

- **Void**

Immediate Return

- **ELCD_PAR_RET**

Possible changes required

LCD interface and LCD controller initialization to be put under this function. Depending on the hardware used the initialization sequence of LCD controller and LCD interface need to be taken care.

Example of Initialization sequence for I-Sample

1. Reset LCD Interface
2. Enable clock
3. Disable /Enable DMA
4. Select type of interface 8086 /6800
5. Reset Start of LCD controller
6. Reset End of LCD controller
7. Power on sequence
8. Turn on sequence.

Example of Initialization sequence for 4 level grayscale LCD module

1. Power on
2. Wait for Stabilizing power
3. Set power save
4. Oscillator on
5. Regulator register select
6. Electronic volume register select
7. LCD Bias register select
8. Gray scale select
9. power control
10. Release power state and wait for stabilizing the LCD power level.
11. End of Initialization sequence

3.1.5 f_lcd_if_poll_write

```
 void f_lcd_if_poll_write( E_LCD_IF_CS  d_cs,      // LCD Controller Chip select
                    SYS_UWORD16  *p_data,    // pointer on data buffer
                    SYS_UWORD32  d_size,     // data buffe size to write
                    E_LCD_IF_DATA_TYPE d_type//  Instruction or Data type selector
)
```

Description

This function specifies the LCD Controller write procedure in polling mode.

Parameters

- **d_cs** // LCD Controller Chip select
- ***p_data** // pointer on data buffer
- **d_size** // data buffe size to write
- **d_type** // Instruction or Data type selector

Immediate Return

- Void

Possible changes required

Depend on the hardware used the customer has to implement the write sequence with polling method.

Current write procedure in polling method

1. Acquire bus
2. Configure pin
3. Set the chip select and data type
4. While (data to write) wait
5. Exit while loop if you don't have any data to write

Example:

Sequence for writing display data for NT7506

1. Acquire bus if it is shared between peripherals.
2. Set page address
3. Set column address
4. Data write
5. Increment column and check still data to be written
6. If yes then go to step3
7. if no increment column and come out of loop

3.1.6 f_lcd_if_dma_enable

```
void f_lcd_if_dma_enable( E_LCD_IF_CS  d_cs,    // LCD Controller Chip select
                E_LCD_IF_FRAME_SZ d_min_frame_sz, //LCD Interface Minimum frame size
                E_LCD_IF_DATA_TYPE d_type )   //  Instruction / Data type selector
```

Description

The DMA capabilities are enabled using this function. In the LCD driver context DMA is used to copy the data between the frame buffer and Transmit FIFO.

Parameters

- **d_cs** // LCD Controller Chip select
- **d_min_frame_sz** //LCD Interface Minimum frame size
- **d_size** // data buffe size to write
- **d_type** // Instruction or Data type selector

Immediate Return

- **Void**

Possible changes required

The data transfer is executed by using either a DMA request or interrupt. The bus is shared between camera, LCD and nand flash. If the bus is shared between other peripherals then the sequence to grab the bus need to be modified.

The following explains the current sequence:

1. Acquire bus // Need to modify accordingly as the bus is shared.
2. Configure Pin
3. Set chip select and data or instruction type
4. Enable DMA capabilities //Enable DMA capability.

3.1.7 f_lcd_if_dma_disable

```
E_LCD_IF_RET  f_lcd_if_dma_disable(E_LCD_IF_CS d_cs,// LCD Controller Chip select
            T_LCD_IF_CALLBACK  pf_callback_sts //  Status Callback function pointer
                        )
```

Description

DMA capabilities are disabled using this function.

Parameters

- **d_cs** // LCD Controller Chip select
- **pf_callback_sts** // Status Callback function pointer

Immediate Return

- **E_LCD_IF_RET**

Possible changes required

The bus is shared between camera, LCD and nand flash. If the bus is shared between other peripherals then the sequence to release the bus need to be modified.

The following explains the current sequence:

1. Disable DMA capabilities from device // Disable the DMA capabilities.
2. Install status callback function
3. Release bus // Need to modify accordingly as the bus is shared.

3.1.8 f_lcd_if_set_cs_and_data_type

```
void f_lcd_if_set_cs_and_data_type(E_LCD_IF_CS d_cs, // LCD Controller Chip select
          E_LCD_IF_DATA_TYPE d_type,            // Instruction Data type selector
       E_LCD_IF_DATA_ACCESS d_access            // Data access
                                  )
```

Description

Updates the CNTL_REG and LCD_CNTL_REG to change LCD Controller addressing.
Updates LCD_CNTL_REG to set data type and read or write access

Parameters

- **d_cs** // LCD Controller Chip select
- **d_type**, // Instruction Data type selector
- **d_access** // Data access

Immediate Return

- **Void**

Possible changes required

The following things need to be write to the new hardware used

Current procedure to Chip select ad set data type

1. Update CNTL_REG with new chip select // Changes required with new hardware
2. Set clock enable bit in CNTL_REG // Changes required with new hardware
3. Set the defied type Instruction or data // Changes required with new hardware
4. Set access type // Changes required with new hardware

3.1.9 lcd_parallel_display

```
ELCD_PAR_RET lcd_parallel_display(UINT16 *ImageDataPtrPAR
                                                        )
```

Description

This function is responsible to copy the data to the display buffer.

Parameters

- **d_ ImageDataPtrPAR** // Pointer to a impage data buffer

Immediate Return

- **LCD_PAR_RET_OK**
- **LCD_PAR_RET_FAIL**

Possible changes required

Depending on the hardware used customer has to write sequence to copy the data to display buffer. If customer is using DMA mode, then the number of bytes to be transferred (depends on the resolution of LCD panel) should be recalculated.

The following explains the current sequence:

Copying data to display buffer can be done in either way.

1. By using polling method
2. By using DMA
   A) By using polling method
      1. Lock mutex
      2. Set ram address

3. Wait in for till copy is complete
4. In for loop check for little endian or big endian

B) By using DMA method
   1. *Calculate bytes to transfer*
   2. Calculate number of frames
   3. Call DMA for data transfer
   4. Wait till we get event from DMA callback

3.2 Data Structure requiring modification to adapt to new hardware:

3.2.1 T_LCD_SELECT

T_LCD_SELECT	selects a specific LCD

Synopsis
```
typedef enum {
                    DISPLAY_MAIN_LCD,
                    DISPLAY_SUB_LCD
} T_LCD_SELECT;
```

If customer is using only one LCD then by default DISPLAY_MAIN_LCD should be selected.

3.2.2 T_LCD_PIXFORMAT

T_LCD_PIXFORMAT	selects the Pixel format for the pixel data

Synopsis
```
typedef enum {
                    RGB565,
                    RGB666,
                    RGB888
} T_LCD_PIXFORMAT;
```

If customer decides to use any other format then customer need to add pixel format to T_LCD_PIXFORMAT. The pixel format input to the frame buffer should be same as the pixel format supported by LCD panel. Any kind of pixel format conversion does not take place in the LCD driver.

3.2.3 T_LCD_COMMAND

T_LCD_COMMAND	Command to be issued to the LCD driver

Synopsis
```
typedef enum {
                    LCD_CLEAR,
                    LCD_DISPLAYON,
                    LCD_DISPLAYOFF,
                    LCD_GETCONFIG,
                    LCD_SETCONFIG
} T_LCD_COMMAND;
```

Customer need to add the commands required to support new hardware.

Example:
If LCD hardware has separate control register for ICON display

LCD_DISPLAY_ICON_ON //This will enable the ICON display
LCD_DISPLAY_ICON_OFF //This will disable the ICON display

3.2.4 # defines directives need to change with hardware

```
#define LCD_HEIGHT 220                  // Change LCD Height with new hardware
#define LCD_WIDTH 175                   // Change LCD Width with new hardware
#define LCD_PIXEL_FORMAT RGB565         // The pixel format input to the frame buffer should be same as
                                        //   the pixel format supported by LCD panel.
```

References

1. *Locosto_BSP_API.doc*
2. *locsto_bum_LCD.pdf*
3. Data Sheet NT7506

Appendix A. Listing Figures and Tables