emWin Display Driver GUIDRV_FlexColor

Supported hardware

Controllers

The supported display controllers are listed in the description of the function GUIDRV_FlexColor_SetFunc() below.

Bits per pixel

Supported color depth is 16 bpp and 18 bpp.

Interfaces

The driver supports 8-bit, 9-bit and 16-bit indirect interface.

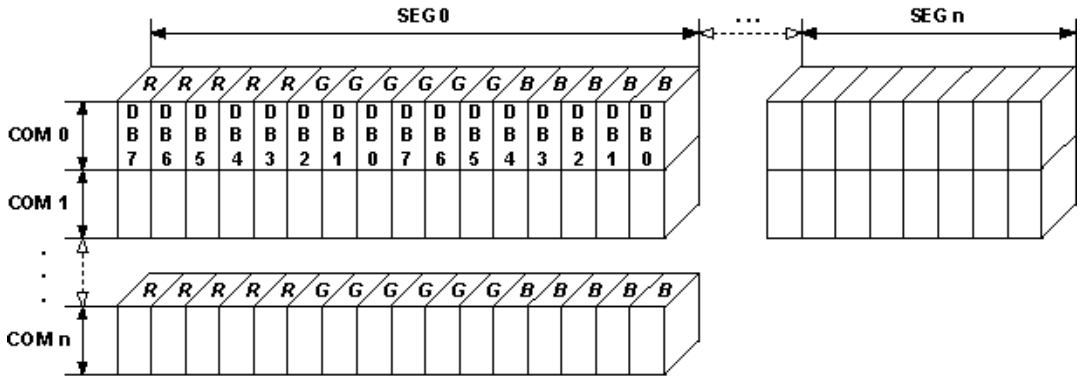
Driver selection

To be able to use this driver the following call has to be made:

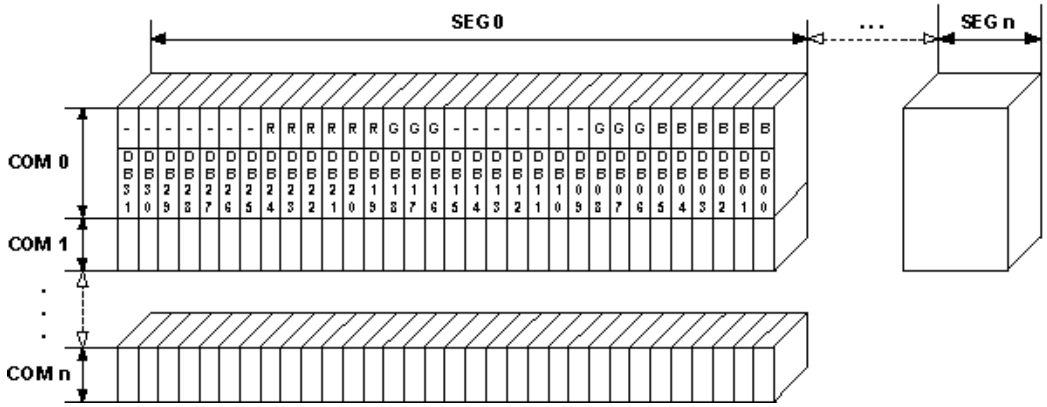
```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_FLEXCOLOR,
COLOR_CONVERSION, 0, Layer);
```

Display data RAM organization

16 bits per pixel, fixed palette = 565



18 bits per pixel, fixed palette = 666_9



RAM requirements

This display driver requires app. 500 Bytes to work. It can also be used with and without a display data cache, containing a complete copy of the content of the display data RAM. The amount of memory used by the cache is:

```
LCD_XSIZE * LCD_YSIZE * BytesPerPixel
```

BytesPerPixel is 2 for 16bpp mode and 4 for 18bpp mode. Using a cache avoids reading operations from the display controller in case of XOR drawing operations and further it speeds up string output operations.

Configuration routines

Routine	Description
GUIDRV_FlexColor_SetFunc	Configures bus, cache and hardware routines.
GUIDRV_FlexColor_Config()	Configures orientation and offset of the SEG- and COM-lines.
GUIDRV_FlexColor_SetInterface66709_B16()	Detailed interface selection.
GUIDRV_FlexColor_SetInterface66712_B16()	Detailed interface selection.
GUIDRV_FlexColor_SetInterface66712_B18()	Detailed interface selection.
GUIDRV_FlexColor_SetInterface66720_B16()	Detailed interface selection.

GUIDRV_FlexColor_SetFunc()

Description

Configures bus width, cache usage and hardware routines.

Prototype

```
void GUIDRV_FlexColor_SetFunc(GUI_DEVICE   * pDevice,
                              GUI_PORT_API * pHW_API,
                              void (* pfFunc)(GUI_DEVICE *
pDevice),
                              void (* pfMode)(GUI_DEVICE *
pDevice));
```

Parameter	Description
pDevice	Pointer to the driver device structure.
pHW_API	Pointer to a GUI_PORT_API structure. See required routines below.
pfFunc	Controller selection macro. See table below.
pfMode	See table below.

Permitted values for parameter pfFunc Supported display controller	
GUIDRV_FLEXCOLOR_F66702	Set up the driver to use one of the following controllers: • Solomon SSD1284, SSD1289, SSD1298
GUIDRV_FLEXCOLOR_F66708	Set up the driver to use one of the following controllers: • FocalTech FT1509 • Ilitek ILI9320, ILI9325, ILI9328, ILI9335 • LG Electronics LGDP4531, LGDP4551 • OriseTech SPFD5408 • Renesas R61505, R61580
GUIDRV_FLEXCOLOR_F66709	Set up the driver to use one of the following controllers: • Epson S1D19122 • Himax HX8325A, HX8353, HX8357 • Ilitek ILI9163, ILI9338, ILI9340, ILI9341, ILI9342, ILI9481, ILI9486, ILI9488 • Novatek NT39122 • Orisetech SPFD54124C, SPFD5414D • Renesas R61516, R61526 • Sitronix ST7628, ST7637, ST7687, ST7735 • Solomon SSD1355
GUIDRV_FLEXCOLOR_F66712	Set up the driver to use one of the following controllers: • Himax HX8340, HX8347, HX8352
GUIDRV_FLEXCOLOR_F66714	Set up the driver to use the following controller:

	• Solomon SSD2119
GUIDRV_FLEXCOLOR_F66715	Set up the driver to use the following controller: • Himax HX8352B
GUIDRV_FLEXCOLOR_F66718	Set up the driver to use the following controller: • Syncoam SEPS525
GUIDRV_FLEXCOLOR_F66719	Set up the driver to use the following controller: • Samsung S6E63D6
GUIDRV_FLEXCOLOR_F66720	Set up the driver to use the following controller: • Solomon SSD1961, SSD1963
GUIDRV_FLEXCOLOR_F66721	Set up the driver to use the following controller: • RAIO RA8870, RA8875
GUIDRV_FLEXCOLOR_F66722	Set up the driver to use the following controller: • Solomon SSD1351
GUIDRV_FLEXCOLOR_F66772	Set up the driver to use the following controller: • Himax HX8301 • Hitachi HD66772 • Ilitek ILI9220, ILI9221 • LG Electronics LGDP4525 • Samsung S6D0117 • Sitronix ST7712

The display controllers listed in the table above are the currently known controllers compatible to the driver. Please note that the used numbers of the selection macros are compatible to some of the LCD_CONTROLLER macro of the driver GUIDRV_CompactColor_16. This makes it easy to migrate from the compile time configurable GUIDRV_CompactColor_16 to the runtime configurable GUIDRV_FlexColor.

Permitted values for parameter pfMode	
GUIDRV_FLEXCOLOR_M16C0B8	16bpp, no cache, 8 bit bus
GUIDRV_FLEXCOLOR_M16C1B8	16bpp, cache, 8 bit bus
GUIDRV_FLEXCOLOR_M16C0B16	16bpp, no cache, 16 bit bus
GUIDRV_FLEXCOLOR_M16C1B16	16bpp, cache, 16 bit bus
GUIDRV_FLEXCOLOR_M18C0B9	18bpp, no cache, 9 bit bus
GUIDRV_FLEXCOLOR_M18C1B9	18bpp, cache, 9 bit bus
GUIDRV_FLEXCOLOR_M18C0B18	18bpp, no cache, 9 bit bus
GUIDRV_FLEXCOLOR_M18C1B18	18bpp, cache, 9 bit bus

Each controller selection supports different operation modes. The table below shows the supported modes for each controller:

Selection macro	**M16C0B8**	**M16C1B8**	**M16C0B16**	**M16C1B16**	**M18C0B9**	**M18C1B9**	**M18C0B18**	**M18C1B18**
GUIDRV_FLEXCOLO	X	X	X	X	-	-	-	-

R_F66708								
GUIDRV_FLEXCOLOR_F66709	X	X	X	X	-	-	-	-
GUIDRV_FLEXCOLOR_F66712	X	X	X	X	-	-	X	X
GUIDRV_FLEXCOLOR_F66714	X	X	X	X	X	X	-	-
GUIDRV_FLEXCOLOR_F66718	X	X	X	X	X	X	-	-
GUIDRV_FLEXCOLOR_F66719	X	X	X	X	-	-	-	-
GUIDRV_FLEXCOLOR_F66720	X	X	X	X	-	-	-	-

'-' means not supported

'X' means supported

Required GUI_PORT_API routines

The required GUI_PORT_API routines depend on the used interface. In case a cache is used the routines for reading data are unnecessary for each interface:

8 bit interface

Element	Data type
pfWrite8_A0	void (*)(U8 Data)
pfWrite8_A1	void (*)(U8 Data)
pfWriteM8_A1	void (*)(U8 * pData, int NumItems)
pfReadM8_A1	void (*)(U8 * pData, int NumItems)

16 bit interface

Element	Data type
pfWrite16_A0	void (*)(U16 Data)
pfWrite16_A1	void (*)(U16 Data)
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)
pfReadM16_A1	void (*)(U16 * pData, int NumItems)

18 bit interface

Element	Data type
pfWrite32_A0	void (*)(U32 Data)
pfWrite32_A1	void (*)(U32 Data)
pfWriteM32_A1	void (*)(U32 * pData, int NumItems)

pfReadM32_A1	void (*)(U32 * pData, int NumItems)

9 bit interface

When working with a 9 bit interface and a color depth of 18 bpp the display controller uses the lines D10-D17 (8 bit) for passing instructions and D9-D17 (9 bit) for passing data. So in this case the lines D9-D17 are connected to the interface lines of the CPU. In order to be able to process pixel data as fast as possible, the driver uses two 16 bit data values per pixel (0000000R RRRRRGGG and 0000000G GGBBBBBB) in which only the first 9 bits each contain pixel data, which are passed to the hardware routines.

As mentioned above the command and parameter interface use the lines D10-D17 (8 bit) of the display controller. The driver passes 16 bit values to the hardware routines where the bits 1-8 already contain the information for the lines D10-D17 of the display controller (bits 0 and 9-15 are unused). So no shift operation is required in the hardware routines.

Element	Data type	Description
pfWrite16_A0	void (*)(U16 Data)	Register to be set (DB1-DB9)
pfWrite16_A1	void (*)(U16 Data)	Parameter to be set (DB1-DB9)
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)	Data to be written (DB0-DB9)
pfReadM16_A1	void (*)(U16 * pData, int NumItems)	Data read (DB0-DB9)

GUIDRV_FlexColor_Config()

Description

Configures orientation and offset of the SEG- and COM-lines.

Prototype

```
void GUIDRV_FlexColor_Config(GUI_DEVICE * pDevice,
CONFIG_FLEXCOLOR * pConfig);
```

Parameter	Description
pDevice	Pointer to the device to configure.
pConfig	Pointer to a CONFIG_FLEXCOLOR structure. See element list below.

Elements of CONFIG_FLEXCOLOR

Data type	Element	Description
int	FirstSEG	First segment line.
int	FirstCOM	First common line.
int	Orientation	One or more "OR" combined values of the table below.
U16	RegEntryMode	Normally the display controller uses 3 bits of one register to define the required display orientation. Normally these are the bits ID0, ID1 and AM. To be able to control the content of the other bits the RegEntryMode element can be used. The driver combines this value

		with the required orientation bits during the initialization process.
int	NumDummyReads	Defines the number of reading operations which have to be done until valid data can be retrieved.

Permitted values for parameter Orientation	
GUI_MIRROR_X	Mirroring the X-axis.
GUI_MIRROR_Y	Mirroring the Y-axis.
GUI_SWAP_XY	Swapping X- and Y-axis.

GUIDRV_FlexColor_SetInterface66709_B16()

Description

Sets the type of interface to be used.

Prototype

```
void GUIDRV_FlexColor_SetInterface66709_B16(GUI_DEVICE * pDevice,
int Type);
```

Parameter	Description
pDevice	Pointer to the device to configure.
Type	Type of the interface to be used. See possible types below.

Permitted values for parameter Type	
GUIDRV_FLEXCOLOR_IF_TYPE_I	3 cycles and data conversion required.(default)
GUIDRV_FLEXCOLOR_IF_TYPE_II	2 cycles and no conversion required.

Additional information

The difference between the interfaces affects only reading back pixels. Whereas TYPE_I extracts the index value by assembling it from the second and third word received from the controller, TYPE_II uses the second word as it is. The right interface depends on the used controller.

TYPE_I

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	G5	G4	G3	G2	G1	G0	-	-	R4	R3	R2	R1	R0	-	-	-
3rd	-	-	-	-	-	-	-	-	B4	B3	B2	B1	B0	-	-	-

In dependence of controller settings red and blue could be swapped.

TYPE_II

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	B4	B3	B2	B1	B0	G5	G4	G3	G2	G1	G0	R4	R3	R2	R1	R0

In dependence of controller settings red and blue could be swapped.

GUIDRV_FlexColor_SetInterface66712_B16()

Description

Sets the type of interface to be used.

Prototype

```
void GUIDRV_FlexColor_SetInterface66712_B16(GUI_DEVICE * pDevice,
int Type);
```

Parameter	Description
pDevice	Pointer to the device to configure.
Type	Type of the interface to be used. See possible types below.

Permitted values for parameter Type	
GUIDRV_FLEXCOLOR_IF_TYPE_I	(default)
GUIDRV_FLEXCOLOR_IF_TYPE_II	

Additional information

The difference between the interfaces affects only reading back pixels. The right interface depends on the used controller.

TYPE_I

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	-	-	-	-	-	-	-	-	B4	B3	B2	B1	B0	-	-	-
3rd	-	-	-	-	-	-	-	-	G5	G4	G3	G2	G1	G0	-	-
4th	-	-	-	-	-	-	-	-	R4	R3	R2	R1	R0	-	-	-

In dependence of controller settings red and blue could be swapped.

TYPE_II

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	-	-	-	-	-	-	-	-	G5	G4	G3	G2	G1	G0	-	-
3rd	-	-	-	-	-	-	-	-	B4	B3	B2	B1	B0	-	-	-
4th	-	-	-	-	-	-	-	-	R4	R3	R2	R1	R0	-	-	-

In dependence of controller settings red and blue could be swapped.

GUIDRV_FlexColor_SetInterface66712_B18()

Description

Sets the type of interface to be used.

Prototype

```
void GUIDRV_FlexColor_SetInterface66712_B18(GUI_DEVICE * pDevice,
int Type);
```

Parameter	Description
pDevice	Pointer to the device to configure.
Type	Type of the interface to be used. See possible types below.
Permitted values for parameter Type	
GUIDRV_FLEXCOLOR_IF_TYPE_I	Uses lines DB0 to DB7 for register access. (default)
GUIDRV_FLEXCOLOR_IF_TYPE_II	Uses lines DB1 to DB8 for register access.

Additional information

The difference between the interfaces affects the register transfer to the controller. Normally there are 2 kinds of possible interfaces available when working with the 18 bit bus interface. TYPE_I uses the lines D0 to D7 for register access whereas TYPE_II uses the lines D1 to D8.

GUIDRV_FlexColor_SetInterface66720_B16()

Description

Sets the type of interface to be used.

Prototype

```
void GUIDRV_FlexColor_SetInterface66720_B16(GUI_DEVICE * pDevice,
int Type);
```

Parameter	Description
pDevice	Pointer to the device to configure.
Type	Type of the interface to be used. See possible types below.
Permitted values for parameter Type	
GUIDRV_FLEXCOLOR_IF_TYPE_I	3 cycles and data conversion required. (default)
GUIDRV_FLEXCOLOR_IF_TYPE_II 2	cycles and no conversion required.

Additional information

The difference between the interfaces affects only reading back pixels. Whereas TYPE_I extracts the index value by assembling it from the second and third word received from the controller, TYPE_II uses the second word as it is. The right interface depends on the used controller.

TYPE_I

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	G5	G4	G3	G2	G1	G0	-	-	R4	R3	R2	R1	R0	-	-	-
3rd	-	-	-	-	-	-	-	-	B4	B3	B2	B1	B0	-	-	-

In dependence of controller settings red and blue could be swapped.

TYPE_II

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	B4	B3	B2	B1	B0	G5	G4	G3	G2	G1	G0	R4	R3	R2	R1	R0

In dependence of controller settings red and blue could be swapped.