

SOFTWARE PORTING INFORMATION

for the

CX100 / CX104 Frame Grabbers

Section 1

PORTING FROM THE FG LIBRARY TO THE CX LIBRARY

You need to read this section if you have Cortex I software that uses the FG libraries and you wish to port it to the CX100 using the CX libraries. This discussion also applies to our Windows DLL's. All libraries and DLL's are 16-bit.

The FG libraries are our previous Cortex I libraries for the C programming language:

FG_B.LIB	Borland linkable library
FG_M.LIB	Microsoft linkable library
WFG.DLL	MS-Windows Dynamic Link Library (DLL)

The CX libraries are our new CX100 libraries for the C programming language:

CX100_B.LIB	Borland linkable library
CX100_M.LIB	Microsoft linkable library
CX100_W.LIB	Watcom linkable library
WCX.DLL	MS-Windows Dynamic Link Library (DLL)

If you are planning to use the CX100 as a Cortex I there is no need to port your code, since the CX100 operates as a drop-in replacement for the Cortex I. In fact, you should continue to use the FG libraries. The CX libraries initialize the CX100 in the CX100 mode and do not allow you to switch back to the Cortex mode.

This section does not explain how to use the new features of the CX100 or the CX libraries. It simply shows you how to modify your Cortex program to make it link with the CX libraries. After you accomplish that, you can begin developing CX100 specific features.

LIBRARY INITIALIZATION

Initialization is the most significant change between the FG and CX libraries. In the FG libraries, we recommended using `find_fg()`. You may also have used `select_fg()` or `init_fg()`.

`find_fg()` and `init_fg()` DO NOT exist in the CX libraries, and `select_fg()` has some significant differences.

Delete your Cortex initialization and use the following:

```
char *err;
err = init_library();
if(err != NULL) {
    puts(err);
    exit(1);
}
reset_cx()          /* only if you want to reboot the CX100 */
```

init_library() initializes the library globals, attempts to locate all the CX100 frame grabbers, in your computer, maps each of them into a vacant memory segment if possible, and selects and activates the one with the lowest index.

init_library() is careful not to disturb the current of the CX100 ports, LUT's or any image that may be in the frame buffer.

If **init_library()** returns a pointer that is not NULL, you should print it and exit. The error string will explain the problem.

reset_cx() recycles the power on the CX100 which clears all the ports, resets the LUT's to their default values, and destroys any image that might be in the frame buffer.

If you are using the power conservation feature of the CX100, you may not want to power up all your frame grabbers at once. **init_library()** WILL power up all your frame grabbers. Use **select_fg()** to power up just one at a time. If **select_fg()** is called instead of **init_library()**, it will accomplish the same initialization of the library globals, however it will not search for any frame grabbers other than the one you specify.

```
char *err;
err = select_fg(2); /* select CX100 at port 0x240 */
if(err != NULL) {
    puts(err);
    exit(1);
}
```

If **select_fg()** is called instead of **init_library()**, it will perform the same initialization of library globals and will return a NULL pointer upon success or a pointer to an error string to explain the problem.

LIBRARY EXIT

The CX libraries contain a function called **exit_library()**. It puts your CX100 frame grabbers back to the default Cortex I mode. If you are running a combination of CX100 and Cortex I programs, you will want to use it. Older programs will not recognize a CX100 as a frame grabber because the status port is at a different location. They can't detect vertical blank. Using **exit_library()** will guarantee that older programs will be able to recognize the CX100's. If you

don't want your older programs to see the CX100's do not use `exit_library()`.

CONTROL PORT BIT NAMES

The names of the control port bits are still spelled the same but their definitions and, in some cases, their uses have changed. There is one exception. `LUT_LOAD_ENABLE` has been changed to `IN_LUT_ENABLE`. The CX100 has two LUT's, `IN_LUT_ENABLE` and `OUT_LUT_ENABLE`.

In the FG library, you could OR two port bits together in a call to the `set()` function:

```
set(VERTICAL_ACCESS | LOW_RESOLUTION);
```

With the CX library, you CANNOT OR any port bits. The above call must be written as two calls.

```
set(VERTICAL_ACCESS);  
set(LOW_RESOLUTION);
```

However, there is no real penalty in execution time. The previous `set()` function was actually a function. The new `set()` is a macro. It results in putting an `outp()` instruction in your source code. The same is true for the new `clr()`.

Cortex control ports

To set or clear control port bits in the Cortex I, you had to write a byte to one of the ports, which meant that you had to guarantee that setting one bit would not clear any other. Our solution to that problem was the creation of the `STATE` array:

```
STATE[8][2]
```

We kept the current bit settings for each port, 0 and 1, in the `STATE` array and OR'ed in any new settings before writing to the port..

Control port bits in the FG libraries were defined as a bit position and a port location:

```
#define RAM_ENABLE    0x120
```

CX100 control ports

All control port bits in the CX100 are set and cleared by writing special codes to port 6. Each bit can be set or cleared individually. There is no longer any need for the `STATE` array. Hence, it is gone. If your code refers to the `STATE` array, you will have to make some modifications. The most common reason for using the `STATE` array would be to get a quick status. Use the `status()` function.

Control port bits in the CX libraries are indexes to an array:

```
#define RAM_ENABLE    13
```

Hence, the bit names now have no intrinsic meaning. They are used to index the array **port_bits[31][2]**.

You will probably never need to address the `port_bits` array directly. It is used by `set()`, `clr()`, and `status()`. The number 31 has no special significance. There are 8 ports, 0 thru 7, each with 8 bits and we decided to only use 31 of them with the library functions.

The `RAM_ENABLE` bit definition in the `port_bits` array looks like this:

```
{ 0x0120, 0x001A }, /* RAM_ENABLE */
```

The left item matches the definition in the FG library. It is the port and bit position. The right item is the special code that is written to port 6. To set `RAM_ENABLE`, the library writes 1A+1. To clear `RAM_ENABLE`, the library writes 1A. The `set()` and `clr()` macros use the right item, and the `status()` function uses the left item.

LIBRARY GLOBALS

The CX library has many more global variables than the FG library. However, this section is only concerned with changes to the old globals

FG library

```
WORD pc_display
WORD video_seg
BYTE far *videoram
BYTE state[8][2]
WORD lastrow
WORD fgp
WORD fg
```

CX library

```
int pc_display
WORD video_seg
BYTE far *videoram
GONE
int lastrow
GONE
int fg
```

A few words about the changes. **lastrow** is normally used in an integer context. It was confusing to keep casting it all the time. It takes its values from a new array called **videolen[2]**. **fg** is an index to an array, and internally it is initialized to the value -1. The **state** array has no use anymore because the ports on the CX100 can be read as well as written, and because bits are set individually through port 6. **fgp** specified the base port address in the FG libraries. If you wanted to write to port 1, you could use **fgp+1**. In the CX libraries, the structure `FGDATA` contains the base port address in item **fgp**. If you want to write to port 1, you can use the macro `PORT1`. The value that gets substituted for it is: **Fd[fg].fgp+1**
fg is global but should never be changed externally. Change it with **select_fg(index)**, which will change it to the value of **index** if possible.

In general, where you used **fgp+n**, you can now use **PORTN** for `PORT0` through `PORT7`.

LIBRARY FUNCTIONS

Most of the library functions that were macros in the FG libraries have had their names changed to lower case. Some of the functions in the FG library that were actually functions have been changed to macros, **set()** and **clr()** are two prime examples. A number of functions from the FG library no longer exist, such as **find_fg()** and **init_fg()**. The FG functions that were carried over to the CX libraries and that still have the same names and interfaces, will operate in the same manner. **set_address()** is an exception.

The interface specification for **set_address()** in the CX libraries is similar to that of the FG libraries.

```
set_address(int addr);
```

However, the parameter **addr** has changed from a WORD to an int, and the format of the data has changed from **A000** (hex) to **000A** (hex). In the FG libraries, you would set **addr** to 0xA000 to specify RAM segment A000. In the CX libraries, you now set **addr** to 0x000A to specify RAM segment A000. The values that **set_address()** puts into the globals **video_seg** and **videoram** have not changed. If **set_address()** succeeds in setting the address to 0xA000, **video_seg** and **videoram** will contain the following:

```
video_seg: A000 (hex)
```

```
videoram: A0000000 (hex)
```

The following is a list of the FG library functions and macros that have been changed or removed in the CX libraries. Functions not listed have not changed.

FG library

```
EGA  
VGA  
AB_VIDEO_SEG  
DE_VIDEO_SEG  
GET_PAGE  
LOW_RES  
HIGH_RES  
LIVE  
COLUMN_ACCESS  
ROW_ACCESS  
GET_COLUMN(buf,column)  
GET_ROW(buf,row)  
PUT_COLUMN(buf,column)  
PUT_ROW(buf,row)  
SAVE_PORTS(BYTE tmp)  
RELOAD_PORTS(BYTE tmp)
```

CX library

```
fgEGA  
fgVGA  
GONE, not needed  
GONE, not needed  
get_page  
low_res  
high_res  
live  
GONE, see set(VERTICAL_ACCESS)  
GONE, see clr(VERTICAL_ACCESS)  
get_column(buf,column)  
get_row(buf,row)  
put_column(buf,column)  
put_row(buf,row)  
SAVE_PORTS(int tmp)  
RELOAD_PORTS(int tmp)  
defined as int tmp[2]
```

blank(void)	blank
blankf(void)	GONE, use blank
dec_page(void)	dec_page
dec_pagef(void)	GONE, use dec_page
display(void)	display
displayf(void)	GONE, use display
find_fg(WORD index)	GONE, see init_library()
have_video(void)	status(HAVE_VIDEO)
high_res(void)	high_res
inc_page(void)	inc_page
int inc_pagef(void)	GONE, use inc_page
int init_fg(void)	GONE, see reset_cx()
live(void)	live
low_res(void)	low_res
lut_load(int *buf)	see the following functions:
	load_input_lut()
	load_input_lut_buf()
	load_output_lut()
	load_output_lut_buf()
	load_overlay_lut()
	load_rgb_lut()
	load_rgb_lut_buf()
see_ram(void)	GONE, use display
seg_switch(void)	GONE, use set_address()
segment_vacant(WORD wAddr)	segment_vacant(int addr)
int select_fg(WORD index,WORD addr)	char *select_fg(int index)
set_address(WORD addr)	set_address(int addr)
set_pagef(int page)	GONE, use set_page()
swap_page_bits(void)	GONE, not needed
unblank(void)	unblank
unblankf(void)	GONE, use unblank

Section 2

PORTING SOFTWARE THAT DOES NOT USE OUR LIBRARY

You need to read this section if you have written software for the Cortex I that does not use our library and you wish to port it to the CX100.

CONTROL PORTS

The control ports in the CX100 are read/write. In the Cortex, they were read-only which meant

you had to keep internal copies of them. You can still do that, however, it is not necessary. You can still write to the control ports in the same manner with the CX100--by writing a byte to each port. You can also take advantage of the port 6 facility that allows you to write a special code to set and clear each bit in the control ports. See the hardware manual for the exact codes.

The port 6 special code for the RAM ENABLE bit is 1A. If you write 1A+1 to port 6, you will set the RAM ENABLE bit. If you write 1A to port 6 you will clear the RAM ENABLE bit.

STATUS PORT

The status port in the CX100 is **port 6**. In the Cortex, it was **port 0**. Change your status port to **BASE PORT +6**.

CX100 MODES

The CX100 can operate in either the Cortex mode or the CX100 mode. The switch is made through port 3. Writing a **46 (hex) to port 3** puts the frame grabber in the Cortex mode. Writing a **47 (hex) to port 3** puts the frame grabber in the CX100 mode. In both cases, you need to wait 30 ms for the change to complete. During that time the port contents are unreliable.

INITIALIZATION

Initialization of the CX100 is substantially different than with the Cortex. The CX100 can operate in two modes and can have its power turned off. An initialization procedure must recognize the board in either of its two modes or guarantee to put it in CX100 mode regardless of its current state.

There are several ways to initialize a CX100 depending on your needs.

1. You can simply reset it and put it in CX100 mode if you don't care about the image that may be in the frame buffer. Video and overlay RAM are left in a random state after a reset.
2. If you know which port it is on, you can force it to CX100 mode by making sure the power is on, clearing ports 0-2, and setting it to CX100 mode. This method will preserve video and overlay RAM.
3. If you need to preserve the state of the ports as well as overlay and video RAM, then you need to determine which mode it is in before clearing any ports bits or setting the mode.

The initialization procedure outlined below is the one described in item 3 above. You can remove some of the steps to achieve the type of initialization described in items 1 and 2.

1. **Read port 6 and wait at least 30 ms.**

Reading port 6 turns on the power if it was off. When the power is turned on, the CX100

resets itself and defaults to the Cortex mode.

2. **Wait for a vertical blank on port 6.**

If the board is in CX100 mode, the vertical blank status bit will be on port 6. If the vertical blank bit on port 6 does not change, then there is no board at that base port address or it is in Cortex mode. Otherwise the board is already in CX100 mode. You need to clear the IN_LUT_ENABLE and OUT_LUT_ENABLE bits to guarantee that the board is in a known state. At this point the board is initialized.

3. **Wait for a vertical blank on port 0** if you did not find one on port 6.

Port 0 is the status port on a Cortex. If the vertical blank bit does not change on port 0, then there is no board at this base port address. Otherwise the board is in Cortex mode. Write a **47 (hex)** to port 3 to put the board in CX100 mode and wait at least 30 ms. At this point the board is initialized.

The pseudo code would be as follows:

```
read port 6
wait
if no vertical blank on port 6
    if no vertical blank on port 0
        there is no frame grabber
    else
        clear port 1 to clear the input lut load bit
        put the frame grabber in CX100 mode
else
    clear input lut load bit
    clear output lut load bit
set the address
```

SETTING THE ADDRESS

You set the address on the CX100 by writing a **DX (hex)**, where X is 8, 9, A, B, C, D, E, or F to port 3. The board will map into any available PC memory segment from 8000 (hex) to F000 (hex).