

Implementation of Software
for
Image Display and Processing

Final Report

Wesley Snyder, Mustafa Dagtekin, Rajeev Ramanath

North Carolina State University

Center for Advanced Computing and Communication

November 8, 1999

1.0	Software Extensions.....	1
1.1	System Flow Chart.....	9
1.2	Transformations	10
2.0	Transformations	11
2.1	Rotation.....	11
2.2	Translation	12
2.3	Zoom.....	12
3.0	Colormap Manipulation.....	13
3.1	Brightness and Contrast Enhancement	13
4.0	Data Flow	14
5.0	The Software.....	17
5.1	Function List	17
6.0	Software Listing.....	20

1.0 Software Extensions

Tandem (the software) communicates between a workstation and the board with the use of a device driver. In general communication to any hardware in a Windows based system is achieved by the simultaneous execution of two software components. One component is called the user mode program which is what the user actually interacts with and the other component initializes protocols between this user program and the operating system and is called a device driver.

The general architecture of a device driver and a user mode program and the communication channels is shown below in figure. 1.

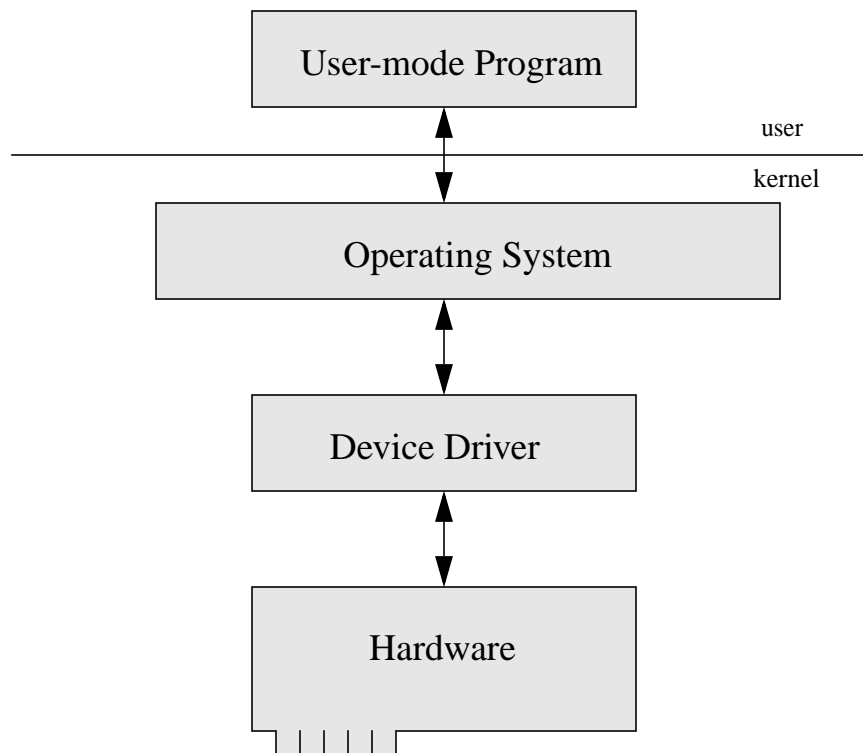


Fig. 1. General architecture of a device driver and user-mode program

In our case, the user needs to communicate only with the PCI bus after which the board takes over. What is needed hence is a device driver that reads and writes to the PCI bus of the workstation and a user program that communicates with the device driver. The PCI bus interface in our case is the PLX9050 chip installed on the PLX9050 RDK board. Hence, we need to communicate only with the PLX9050 RDK.

For the purpose of creating device drivers, Windriver¹ was used. The user mode programs were developed to provide the required interface to the user and communicate with the

device driver. For testing purposes a text-based system was developed which was later refined to a Graphical User Interface based system with the use of the Microsoft Foundation Classes (*MFC*).

The entire software system is dialog-based which makes it convenient to handle user inputs and user controls on the main display screen which would not have been the case with a Document-view architecture system.

CTandemDlg, *CTandemApp* are the primary classes that have been used as abstract data types. Other classes act as inheritances of the dialog boxes which are called when certain interactive menu options are chosen.

Figures 2 through 7 give the list of class dependencies and the class member variables and functions.

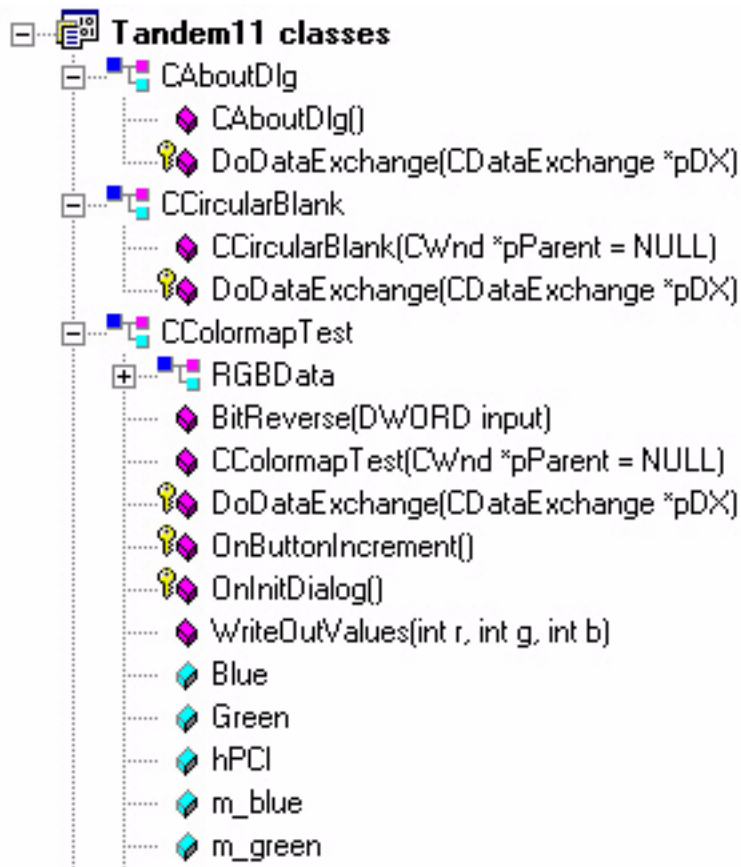


Fig. 2. Class Description of Tandem

1. Windriver is a trademark product of KRFTech Inc.

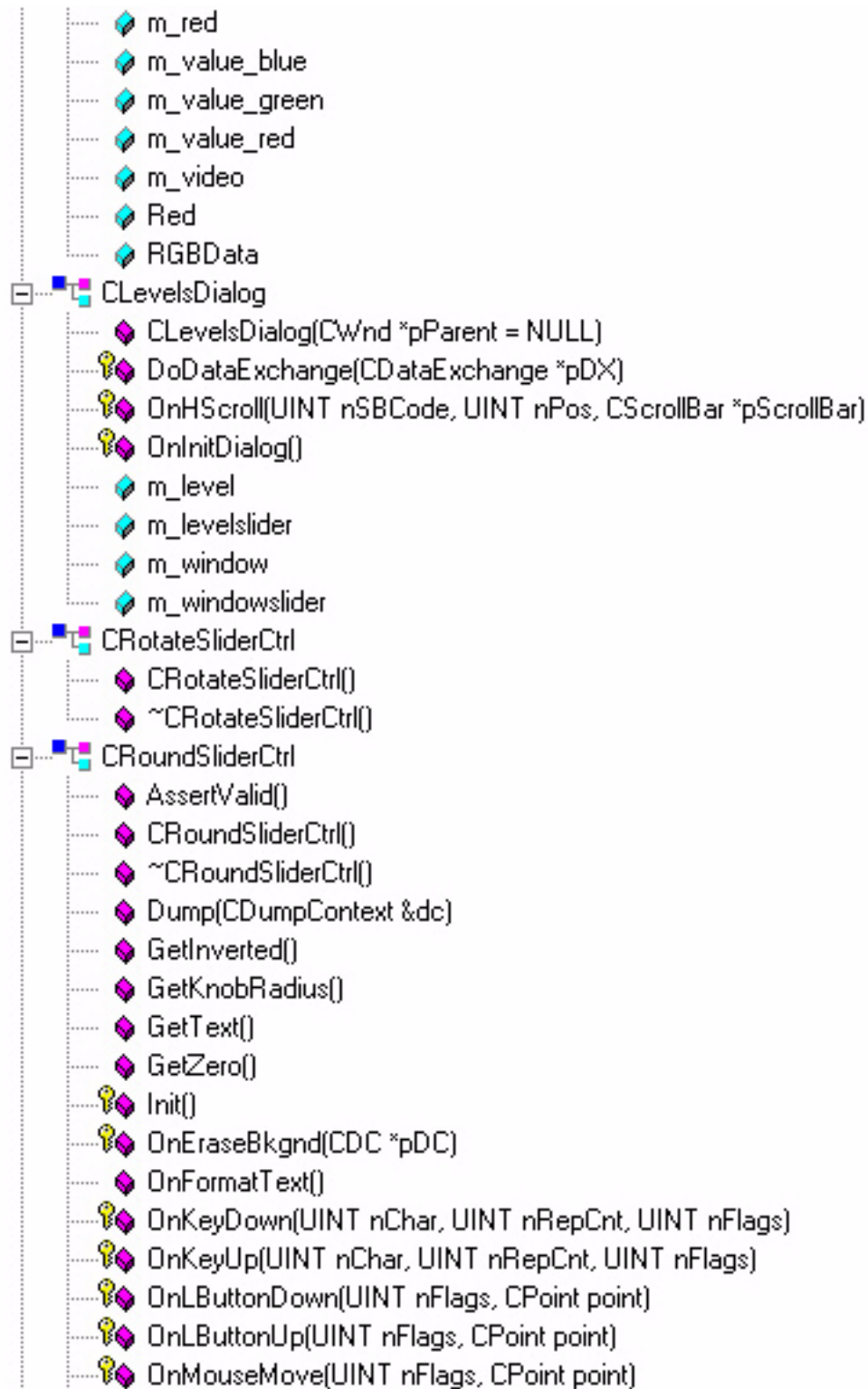


Fig. 3. Class Description of Tandem(contd.)

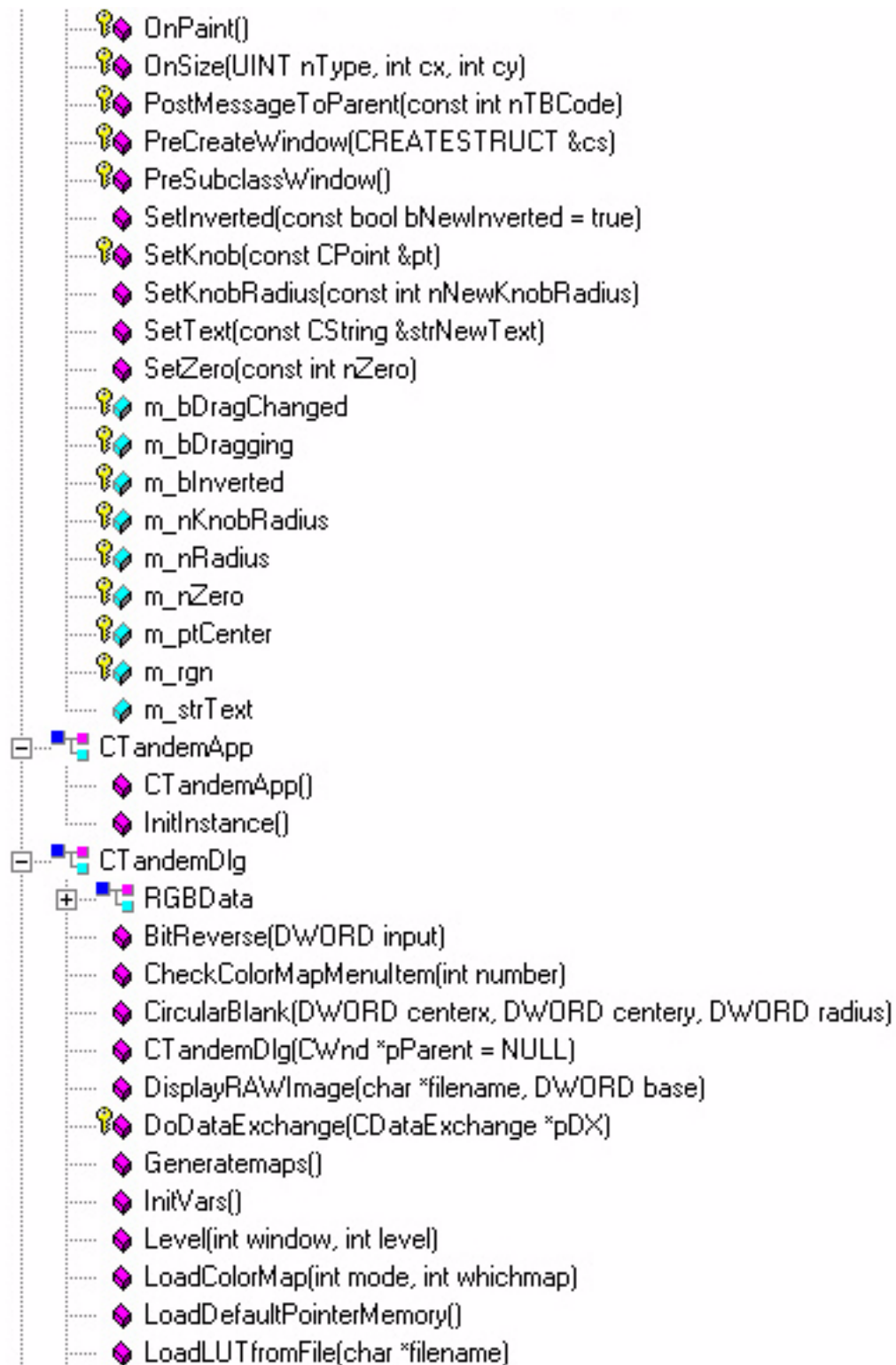


Fig. 4. Class Description of Tandem(contd.)

.....	◆ LoadVideoLUTfromFile(char *filename)
.....	◆ MultiTransform()
.....	◆ OnColormapBronson()
.....	◆ OnColormapFile()
.....	◆ OnColormapGray()
.....	◆ OnColormapHotMetal()
.....	◆ OnColormapInvertedGray()
.....	◆ OnColormapRandom()
.....	◆ OnColormapsHeatedSpectrum()
.....	◆ OnControlsCircularblinking()
.....	◆ OnControlsFrameCapture()
.....	◆ OnControlsFreezeFrame()
.....	◆ OnControlsLevels()
.....	◆ OnExit()
.....	◆ OnHScroll(UINT nSBCode, UINT nPos, CScrollBar *pScrollBar)
.....	◆ OnInitDialog()
.....	◆ OnLoadDefaultConfig()
.....	◆ OnPaint()
.....	◆ OnQueryDragIcon()
.....	◆ OnSysCommand(UINT nID, LPARAM lParam)
.....	◆ OnTestIncrementColormapEntries()
.....	◆ OnTestLoad0x2aa0x155()
.....	◆ OnTransformationsRotate()
.....	◆ OnTransformationsTranslate()
.....	◆ OnTransformationsZoom()
.....	◆ OnUploadTestPatterns()
.....	◆ OnVScroll(UINT nSBCode, UINT nPos, CScrollBar *pScrollBar)
.....	◆ Rotate(int old_angle, int new_angle)
.....	◆ Translate(int x, int y)
.....	◆ UpdateTextDisplay()
.....	◆ WritePM()
.....	◆ Zoom(int new_factor, int old_factor)
.....	◆ bronson
.....	◆ FileName
.....	◆ greymap
.....	◆ hotmetal

Fig. 5. Class Description of Tandem(contd.)

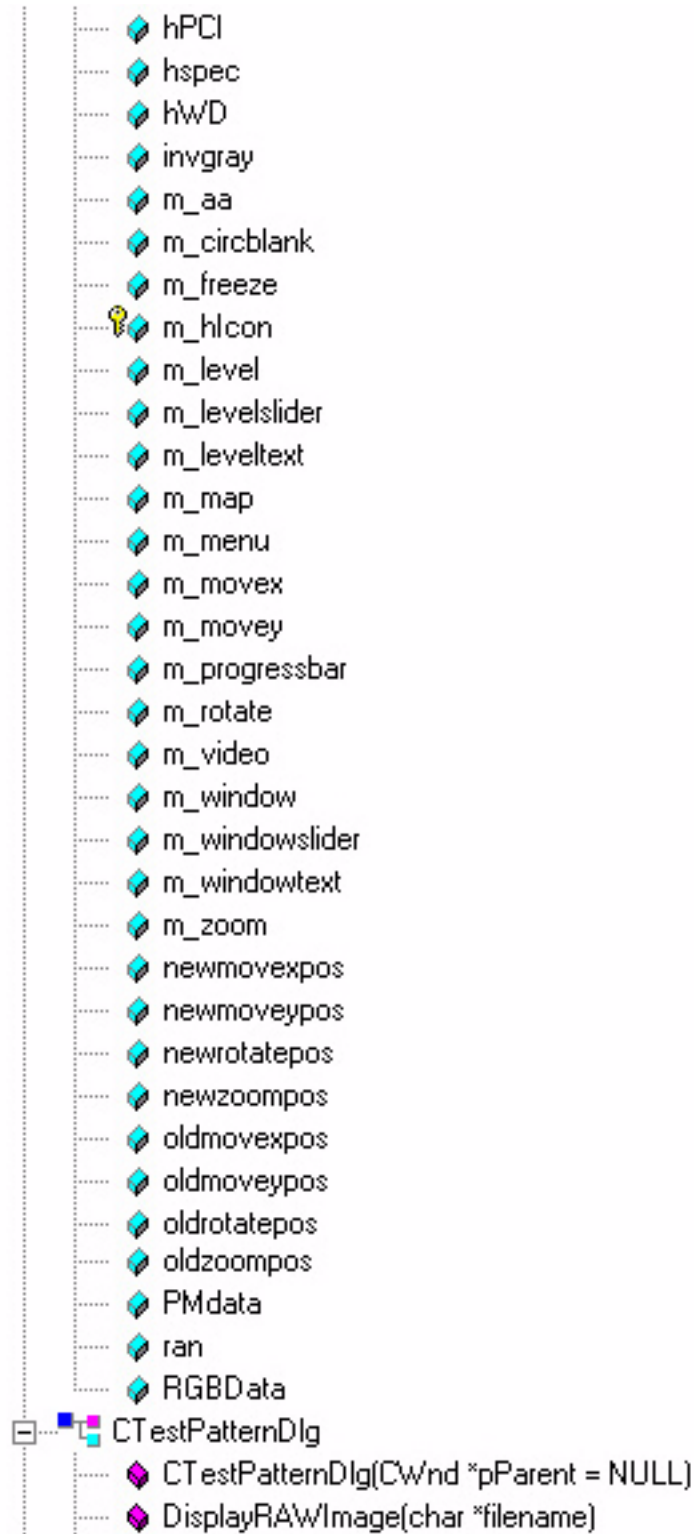


Fig. 6. Class Description of Tandem(contd.)

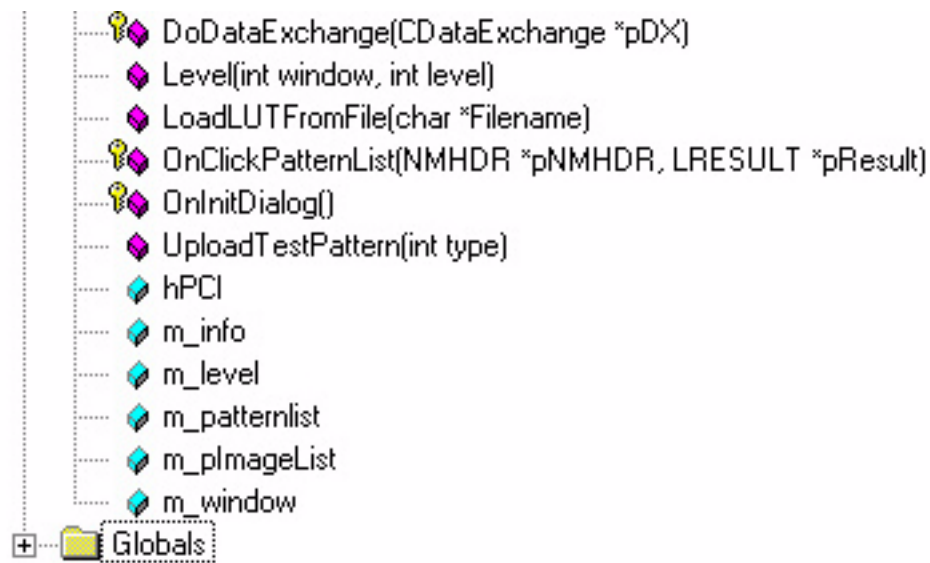


Fig. 7. Class Description of Tandem(contd.)

Most of the system designed follows a pre-determined information flow and the user needs to follow this procedure (given in the following flow-chart) to initialize the board with the required contents so as to produce desired output, meaning thereby that the address spaces need to be set, the video-mode registers loaded according as the type of output desired. Tandem incorporates all these initialization procedures during the initial loading of the software and primarily hides the sequence from the user.

The PLX 9050 RDK requires the following registers to be initialized before the system can be setup for use:

Register	Data (base 16)	Purpose
0x8	0xE000008	Initialize Local Address Space 2 range
0x1C	0x1	Initialize Local Address Space 2 base address
0x3C	0x8000001	Initialize Chip Select base address
0x30	0x828FF8	Initialize Region Descriptor for Local Address Space 2
0x50	(existing data & 0xFFFFF8)	Set USER0 Bit to get VBlank signal as input

Any Local Address Space on the card could have been used, the choice of Local Address Space 2 was completely arbitrary.

After this initialization, the Tandem can address the Video card as memory mapped space. This is followed by the initialization of the Video card to sample and handle 640 by 480 pixel images. The user is referred to the PLX 9050 RDK specification sheets for information about the data written.

The Video Card has four Video Mode Registers(VMR) on board. They are organized as shown below (each register is of length 32 bits)

Video Mode Register	Content	Size (bits)	Content
VMR1	HSON	5	16
	HSOFF	8	112
	HBOFF	7 (higher order)	160
	prefix	2	0
	junk	10	dont care
VMR2	HBOFF	2 (lower order)	160
	Line Length	11	800
	VBON	7 (higher order)	239
	prefix	2	1
VMR3	junk	10	dont care
	VBON	3 (lower order)	239
	VSON	10	241
	VSOFF	7 (higher order)	243
	prefix	2	2
VMR4	junk	10	dont care
	VSOFF	3 (lower order)	243
	Frame Size	10	261
	junk	7	dont care
	prefix	2	3
	junk	10	dont care

From the above table, the data written out to the Video Mode Registers is as follows;

VMR1	0x40E10
VMR2	0x1DEC81
VMR3	0x2E6789
VMR4	0x300829 0x2000

1.1 System Flow Chart

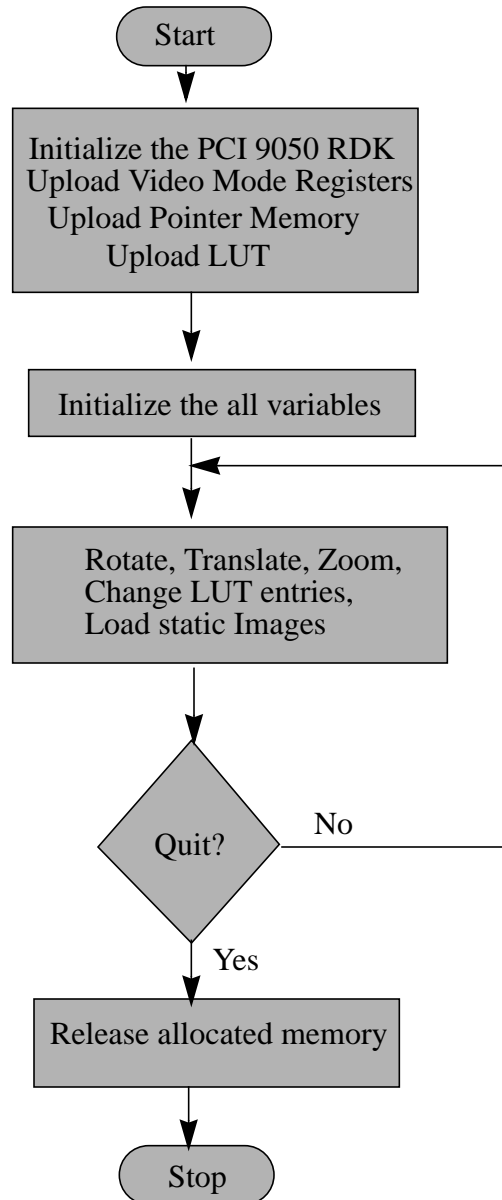


Fig. 8. Flow-chart for the initialization and use of Tandem.

The flow chart above does not include all the possible exceptions that need to be handled and simply gives an idea about the sequence of the steps involved.

Once initialized, the pointer memory and look-up tables need to be uploaded. Upon initialization, the contents of the pointer memory will be monotonically increasing. This would correspond to the contents of the Video RAM without any transformation. When a transformation is desired, the contents of the pointer memory need to be changed accordingly.

1.2 Transformations

The user-mode software applies the linear transformations desired, on the workstation and change the pointer memory contents accordingly so as to produce the indicial mapping desired.

Consider for example the following VRAM contents and PM contents.

VRAM	PM	LUT	Image Displayed
3	0	0	31
1	1	7	7
0	2	15	0
3	3	2	15
2	4	31	0
0	5		
0	6		
	7		
	8		

Fig. 9. Contents of memories - case 1

Let us consider the pixel in location (0, 0) which has a value of 3. The pointer memory contains a value of zero which means that the (0, 0) location in the image will have contents corresponding to the pixel in (0, 0) in the VRAM. The look-up table translates the value of 3 to an intensity value of 31. Therefore we get a value of 31 displayed as the result.

Now, consider the case when the image needs to be flipped about the vertical axis. We need to simply change the contents of the pointer memory. An explanation as above can be extended for the following case also. The following will be the contents of the memories.

VRAM		
3	1	0
3	2	1
2	0	0

PM		
2	1	0
5	4	3
8	7	6

LUT	
0	0
1	7
2	15
3	31

Image Displayed		
0	7	31
7	15	31
0	0	15

Fig. 10. Contents of memories - case 2

The look-up table contents will be uploaded upon initialization and changed only when a colormap change is desired.

2.0 Transformations

Let us now consider the mathematical functions that this card can perform.

2.1 Rotation

Rotation is a linear function which re-locates the pixel by rotating it about a certain axis. For a pixel at location (x, y) , which need to be re-positioned to (x', y') by the process of rotation, the equation is given by

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where, θ is the angle through which we desire the rotation to occur. This rotation is about the z axis if we consider the image to lie in the xy plane.

As an aside, notice that the above transformation does not change the magnitude of the initial vector (x, y) as the magnitude of the transformation matrix \mathbf{R} , is unity. Transformations which maintain magnitude are called orthogonal transformations.

2.2 Translation

Translation is also a linear function which moves the vector along the co-ordinate axes

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where dx and dy are the desired shift in corresponding directions. This operator is referred to as T .

2.3 Zoom

The zoom function is also linear in behavior like R and T and scales an image accordingly.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & k \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where k , is the factor of zoom desired. If we need to zoom in, k should take values greater one and if we need it to zoom out, k should take values less than one. It is referred to as Z

The interesting and useful property with these functions being linear is that they can be composed with each other. This reduces mathematical computation complexity greatly when performing multiple transformations one after the other.

Consider a translate and a zoom being applied to an image.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & k \end{bmatrix} \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

which can be re-written as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & k \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

We could rotate this by simply applying the rotation operator before this ZT operator, giving us RZT which would reduce our computations to one set of matrix-vector multiplication.

As a part of a later version, it is intended to include warping routines which would warp the image arbitrarily. The only change that needs to be made in the existing operators is non-zero entries in the third row. In general, affine transforms can be performed by operators as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

if a_{31} and a_{32} are non-zero.

3.0 Colormap Manipulation

Since the board has a separate look-up table(LUT), the system can output images with pseudo-color based on what is stored in the LUT. Experiments with different colormap values can be conducted. As of now, Tandem has in its database, six colormaps, Gray, Inverted Gray, Hot Metal, Bronson, Heated Spectrum and Random. A option has been created for the user to load a custom made colormap too. The colormap needs to have 1024 entries. In the case of a 256 color display, each entry needs to be entered four times.

Also, we could load into the colormap, values that correspond to the logarithm of the brightness or the exponent of the brightness or the brightness raised to a certain power, generally less than one; i.e. load into the colormap, values 0 through 1024^γ . γ takes values less than one. This is called *gamma-correction*. Infact, we could load into the colormap numbers that are any function of the brightness values to help us understand the details in the image that are accenuated with different colormaps.

3.1 Brightness and Contrast Enhancement

Since the colormap can be loaded dynamically, we can use windowing and level-shifting techniques on the output. The windowing and level shifting techniques sets pixels with brightness values over / below a certain value to fixed numbers and allocates the rest of the colormap to the brightness values within the remaining window.

Consider the following layout

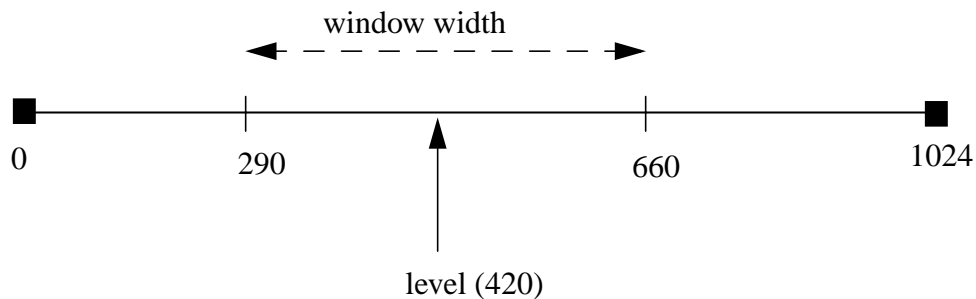


Fig. 11. Brightness and Contrast enhancement

In Fig. 5, 0 and 1024 correspond to the minimum and maximum entries in the colormap. 290 is the minimum value at which the window width has been set and $660-290 = 370$ is the window width. The level is set at 420. Here is how this is interpreted. If a pixel with a brightness of less than 290 is received by the LUT, the output of the LUT is set to zero and similarly for a value of greater than 660. In general, what this gives us is a greater brightness range if the window size is increased and the position of the level knob gives us contrast enhancement.

The above process can be written in the form of the following pseudo-code;

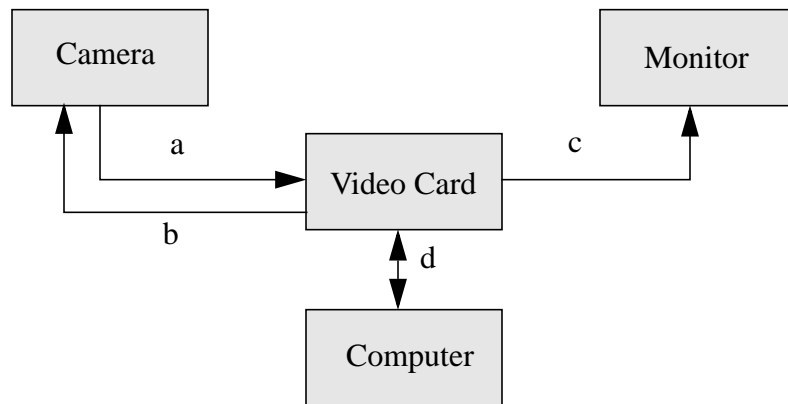
begin:

```
if input < level - (window width)/2
    output zero
if input > level + (window width)/2
    output 255 (brightest value)
else
    output the brightness level as such (after scaling it over the available window range)
```

end.

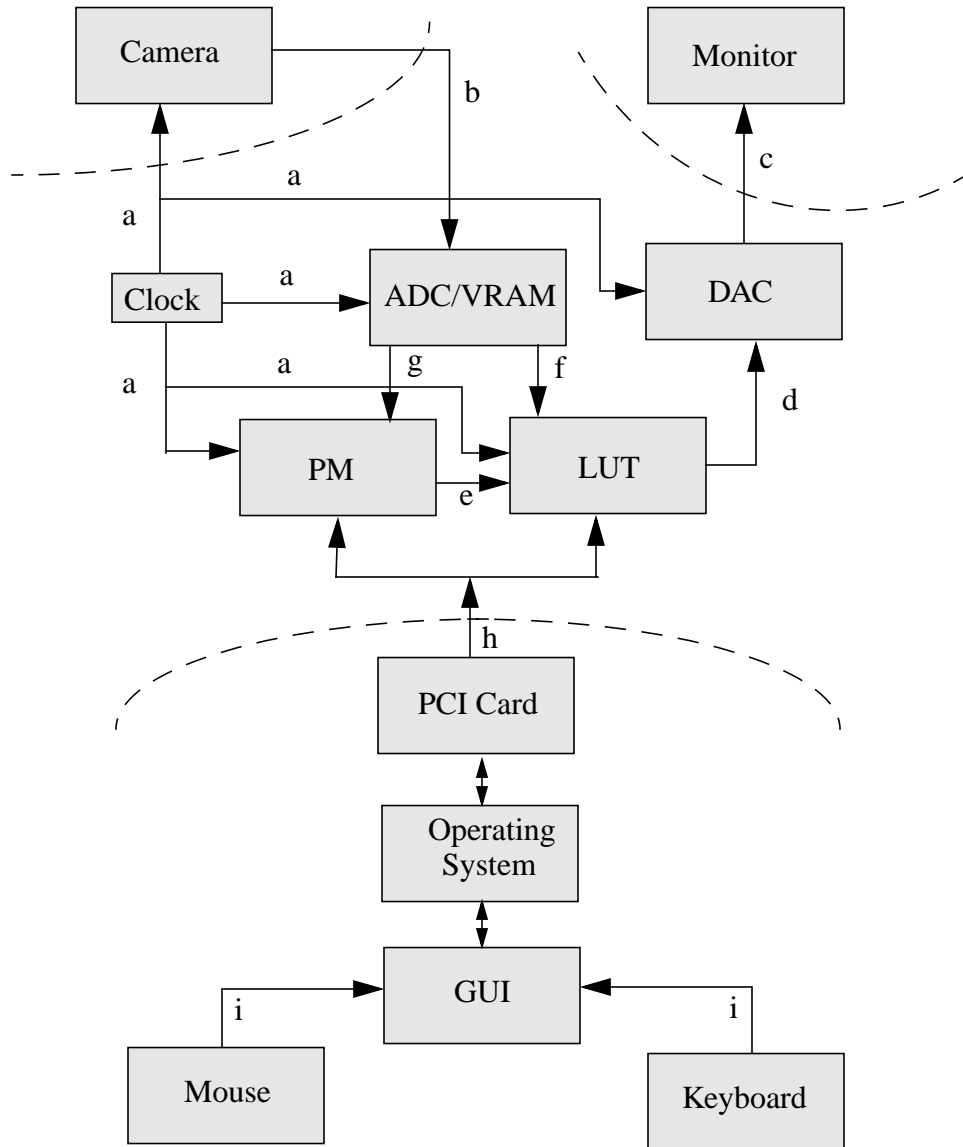
4.0 Data Flow

A data flow diagram for the system may be useful in understanding the methodology involved in getting the video board functional. Two levels of abstraction of the data flow have been given to give a bigger perspective of things.



- a. Analog Input to card*
- b. Sync to camera*
- c. Analog output to monitor*
- d. Data from PC to card, PM data, LUT, VRAM data, etc.*

Fig. 12. Data Flow Diagram - Level One



- a. Clock signal*
- b. Analog input to card.*
- c. Analog Output to monitor*
- d. Digital Data from LUT (8 bits on each, R, G, B)*
- e. Data from PM (10 bits)*
- f. Data from VRAM to LUT*
- g. VRAM to PM*
- h. Data from PC to card, PM data, LUT, VRAM data, etc.*
- i. Interrupts*

Fig. 13. Data Flow Diagram - Level Two

5.0 The Software

The GUI is designed to reduce the user's knowledge requirements. As can be seen in Fig.14, the GUI is rather intuitive. Initialization of the system is done at the initial running of the software and later on, is redone in the Configuration menu option.

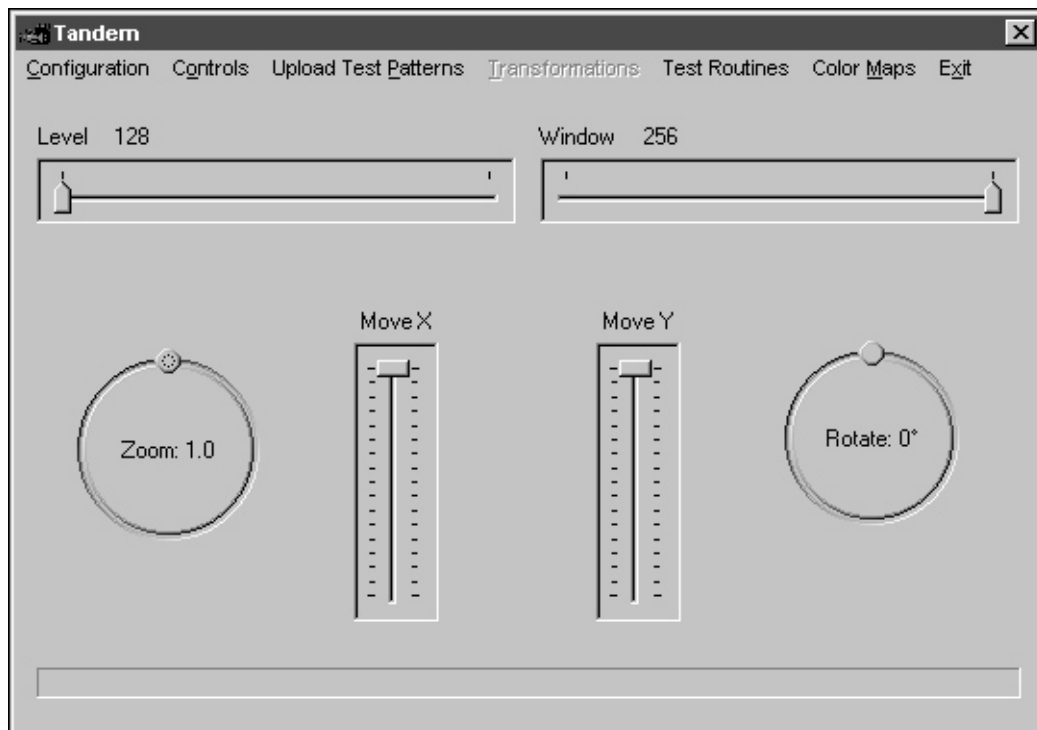


Fig. 14. Screen capture of the GUI developed

5.1 Function List

The following are the functional routines in Tandem:

Configuration->Load Default values:

To load initial values in all memories (Pointer Memory, LUT) and also initialize all controls to their base values.

Controls->Freeze Frame

Freeze the contents of the VRAM.

Controls->Frame Capture

Freeze the frame and capture contents of VRAM to a file named img.raw. The contents of this file are raw image data. The data is 8 bits per pixel and in grayscale.

Controls->Circular Blanking

To apply a circular mask onto the image displayed. Blackens out everything outside the circle. This process has been shown in the following pseudo-code

begin:

```
while i:all rows of Pointer Memory(PM)
    begin while
        while j: all columns PM
            begin while
                if  $i^2 + j^2 > \text{radius of mask desired}$ 
                    write into PM, a location that corresponds to black
                else
                    write out data corresponding to (i,j)
            end while
        end while
    end while
```

end.

Upload Test Patterns:

Stop Video mode and load static images into the VRAM.

ColorMaps->Gray:

Load the grayscale colormap.

ColorMaps->Inverted Gray:

Load the inverted grayscale colormap.

ColorMaps->Heated Spectrum:

Load the heated spectrum colormap.

ColorMaps->Hot Metal:

Load the hot metal colormap.

ColorMaps->Bronson:

Load the bronson colormap.

ColorMaps->Random:

Load a colormap generated by random numbers.

ColorMaps->Load From File:

Load a colormap previously created and stored in a file. Takes input only from files with extension "lut"

6.0 Software Listing

Using the class structure described in Figs. 2 through 7, the following source code has been organized as follows;

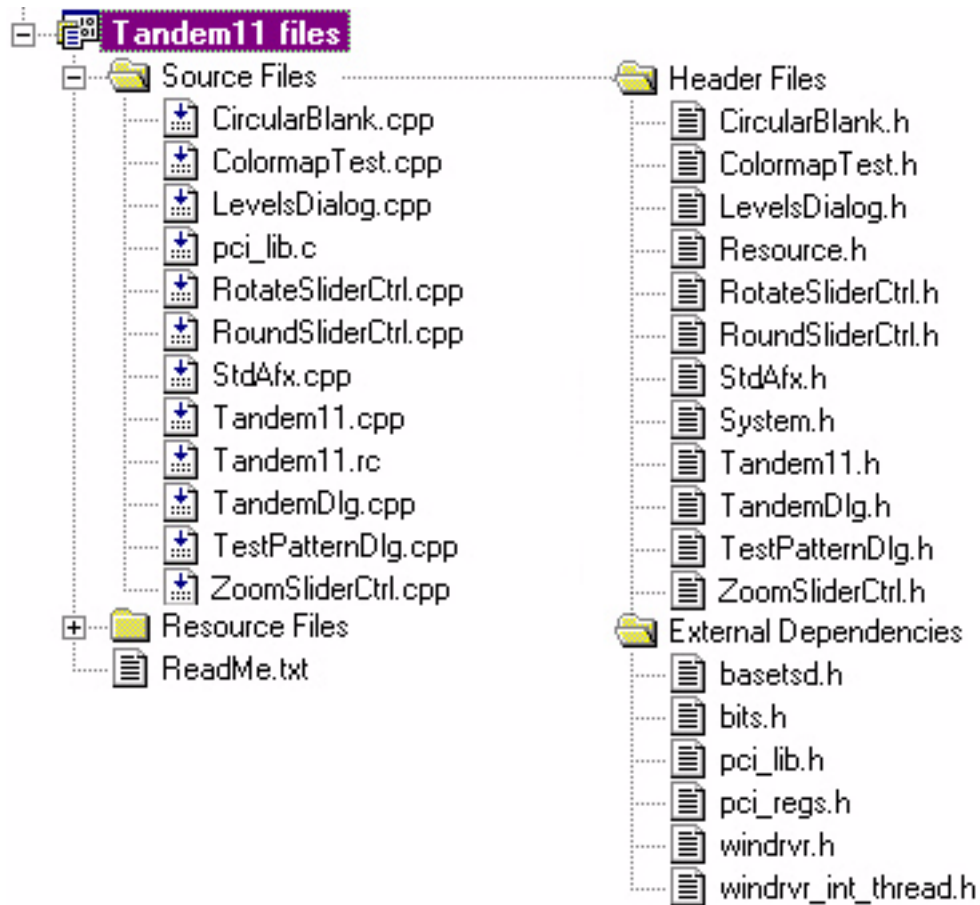


Fig. 15. Organization of files in the Tandem Workspace and in the pages that follow