# Application with:

## TMS320DM6437

**Diana Ripoll Pérez**

**Prague 21-June_2009**

## Thanks to:

Thanks to my family especially to
my parents because they gave me
unconditional support during all
my studies, without they this can
not be possible.

Thanks to all my colleagues, the
people how I met at the university
and we shared many time working.

Thanks to my friends because they
supported me during this years
and for all the unforgettable
moments that they have given me.

And finally, thanks to all the
people that I met this year in
Czech Republic because they have
done that this year has been really
special.

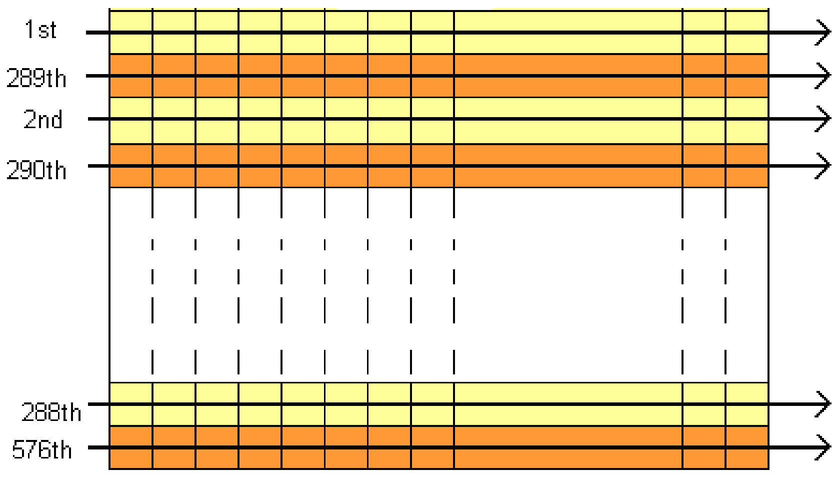
## INDEX:

## Introduction:

I have done this study at the university ČVUT in Prague during the academic year 2008/2009. I have been studying with the Erasmus program thank you to the convention of this university and my own faculty ETSEB from UPC in Barcelona. My tutor has been the professor Zahradni.

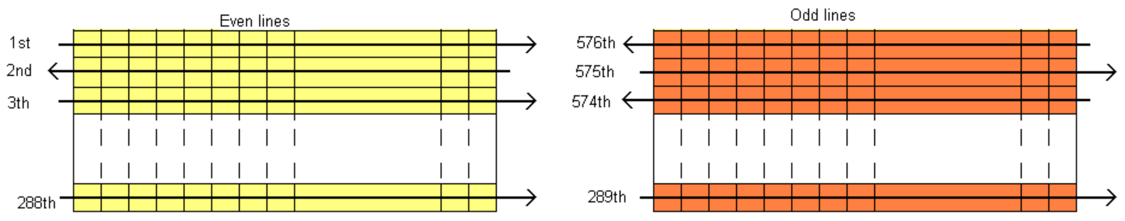
The project is related with video signal processing. Specifically, it is about how to rearrange the pixel of a digital image, in a real time, for can be connected to the mirror projector.

From a video playback device we are obtaining an analogy signal with PAL (Phase Alternating Line) encoding, this signal it can't be connected directly to the projector if we want to optimize the travel of the mirror. An image in this standard is composed for 720 rows and 576 lines and for draw it first are drown the even lines and after the odd lines. The travel for draw an image encoded on the PAL format it's from the top to the bottom and horizontally from the left to the right, this is useful for all lines, even and odd. We can see on the figure bellow this comportment.



The arrows are indicating the direction of each line while the numbers are indicating the order of each line.

If we want to optimize the travel of the mirror on the mirror projector we should rearrange the pixels because the optimum it's that the mirror doesn't have big jumps so always should draw a pixel next to the other. On the figure bellow we can see how should be:



The arrows are indicating the direction of each line while the numbers are indicating the order of each line.

For solve this problem we have done an application, this is upload to a DSP from Texas Instrument, TMS320DM6437. I this DSP the application is run it and it convert the standard signal PAL to the signal that we want to connect on the mirror projector.

The material that we use for this work is a camera, screen, computer and two DSP (one convert the signal and the other one simulates the mirror projector). And the software that we use is CCStudio v3.3 and Matlab.

On the next pages we can see with more detail the steps that we have done and some important concept for do it.

## Components works:

For do this application we use the next components:

### *Hardware:*

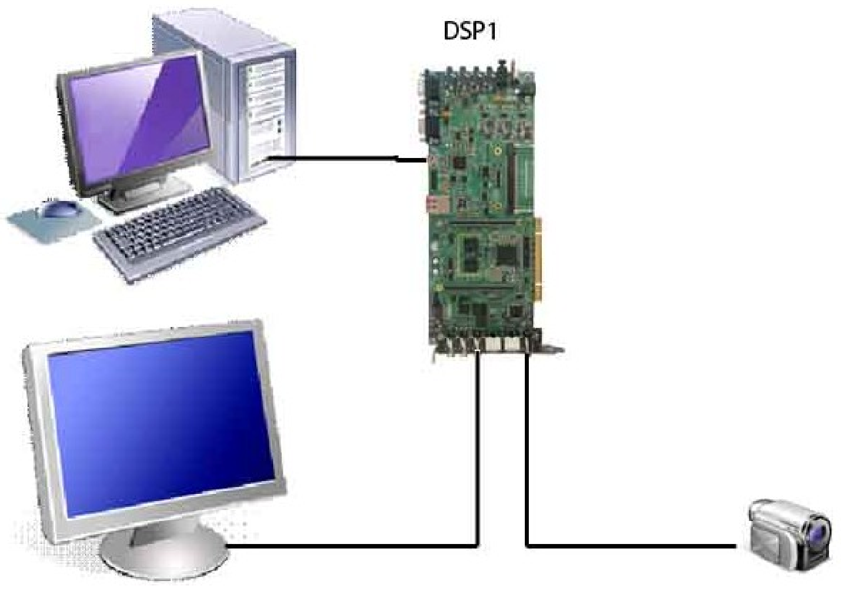
**TMS320DM6437:** It's a DSP, *Digital Signal Processor*, from the fabricant Texas Instrument. It is used for process image and voice signal but in this work we are using only for the video signal processing.
This is the most important hardware of this project and in the next pages we will explain more about it.

**Computer:** we are working directly with the computer, with the correct software we are designing the code and uploading to the DSP, after that, the board can work independently. In addition, we use the computer for obtain information and analyze the results.

**Screen:** On the screen we can observe the directly the result of the application. This it will be connect directly to the DSP.

**Camera:** the camera captures images in a real time. The codification of the output signal from this peripheral is codification PAL *(Phase Alternating Line),* and it's connect directly to the DSP, this means that on the board we have to switch correctly.

In the figure bellow we can see how the work space where the DSP is connected is:



## ***Software:***

**Code Composer Studio v3.3:** this software allows us to write, compile the code of the application of our design. Moreover, we have the options for check, and save in a file, the values of the memory or the values of different parameters of our code, that is useful for debug and find errors. In addition, we can upload to the DSP this code and run it there, on the board, the application can be run independent of the computer.

**Matlab:** Matlab is mathematical software that allows us to analyze the values that we have on the memory. We can draw it in a graphic, check concrete values, measure parameters, etc. Ultimately, with this program we can analyze how is working the application in a simple and clear way.

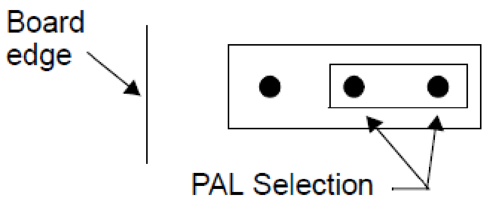
## Configuration:

The DSP have to be configured extern: the switches, and internally: programming, some parameters.

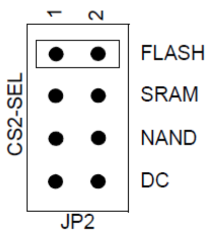
Before connect the DSP with the computer we must to position the switches and jumpers correctly, on the annex 1 we can see a schematic figure of this board where are indicates the names and the situation of the connectors and the switches.

The specifications and correct position for each jumper and switches are:

***JP1 Jumper:*** Jumper JP1 is used to select the display output format, NTSC or PAL. We are working with PAL format so the correct position is shown in the figure bellow.



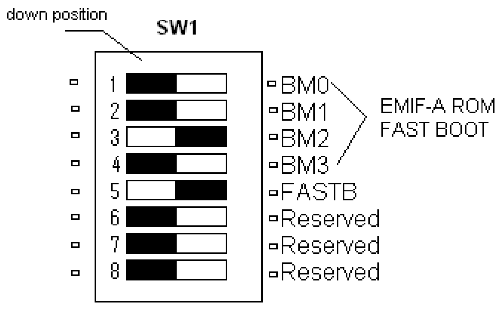
***JP2 Jumper:*** Jumper JP2 is a jumper bank used to select the routing of the CS2 signal. The position that we have worked with is as shown in the following figure

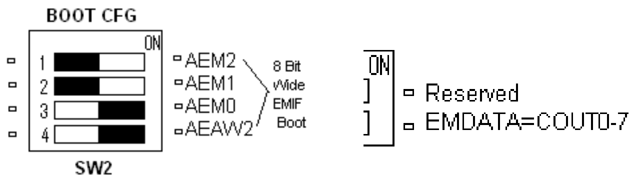


***Switches:*** The DM6437 EVM has seven switches. These switches are used to create certain actions on the board or to select certain functions on the board. The switch functions are summarized in the table.

| SW# | Function |
|---|---|
| SW1 | Bootload Mode Select |
| SW2 | DM6437 Muxing Configuration |
| SW3 | EMIF Data Select |
| SW4 | 4 position user readable |
| SW5 | Power On Reset |
| SW6 | Reset |
| SW7 | Slide Switch |

And we had work with the initial position, for each switch like is shown on the figures bellow:





***SW4, 4 Position User Readable:***

| | |
|---|---|
| 0 | →Down |
| 1 | →Down |
| 2 | →Down |
| 3 | →Up |

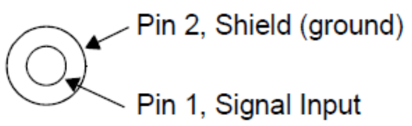
***SW5, Power On Reset Switch:*** Switch SW5 is a momentary switch that asserts power on reset to the DM6437 device.

***SW6, Reset Switch:*** Switch SW6 is a momentary switch that asserts a reset to the DM6437 processor.

***SW7, Slide Switch:*** Switch SW7 is a 2 position slide switch used by demonstration software. The switch is read via a $I^2C$ expander. Refer to the I2C section for more information.

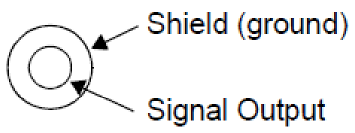
The connectors that we are using in this research are J5, J4, J16 and J501.

***J5, Video in:*** it is a n RCA jack used as a video input to the TVP5246M2 video decoder. This connector brings in a video signal to the TVP5146M2. The figure below shows this connector as viewed from the card edge.

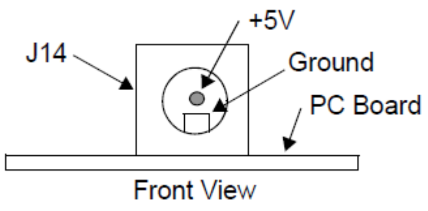


| Pin # | Signal Name |
|---|---|
| 1 | Pin 8, TVP5146M2 |
| 2 | GND |

***J4, DAC D Video Out:*** it is an RCA jack used to interface to DAC D of the DM6437 to a video device. This connector is driven directly by the VPSS back end via an opamp.



***J16, +5V Input:*** Connector J16 is the input power connector. This connector brings in +5 volts to the EVM. This is a 2.5 mm. jack.



***J501, Embedded Mini USB Emulation Interface:*** This connector allows the user to run software development tools and emulation without an external emulator. The signals on this connector are shown in the table below.

| Pin # | Signal Name |
|---|---|
| 1 | VBUS |
| 2 | D- |
| 3 | D+ |
| 4 | ID (not used) |
| 5 | Ground |

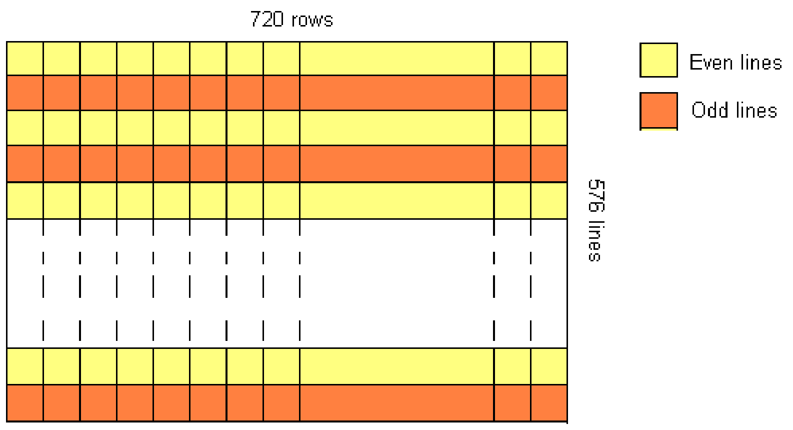
## APPLICATION:

In this work we create an application for connect a PAL signal to a projector mirror which is drawing the image optimizing the travel of the mirror. For do this, we must to know the signal PAL and how should be the signal that we send to the projector.

### *PAL:*

Acronyms PAL are reference short for Phase Alternating Line, and it is an analog television encoding system used in broadcast television systems in large parts of the world. This is an evolution of the NTSC have the same concept but it introduce an improved: part of the color information on the video signal is reversed with each line, which automatically corrects phase errors in the transmission of the signal by canceling them out.

For our study the useful information is how the image is defined in this standard: how it is sent and how many rows and lines are defined. On the figure bellow we can see how the image is a frame:



The relevant parameters for our study are as follows:

Aspect ratio: 4:3

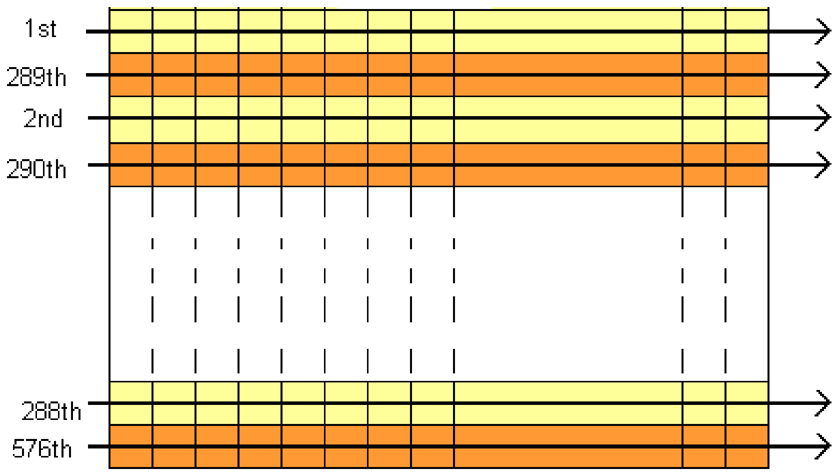
Number of lines: 625

Active lines (effective vertical resolution): 576

Active columns: 720

In this standard, each frame is divided in two half frame, one is compose for the odd lines while the other is compose for the even lines. When the picture is drawn, these frames are painted alternately, first, odd lines then, even lines and so on.

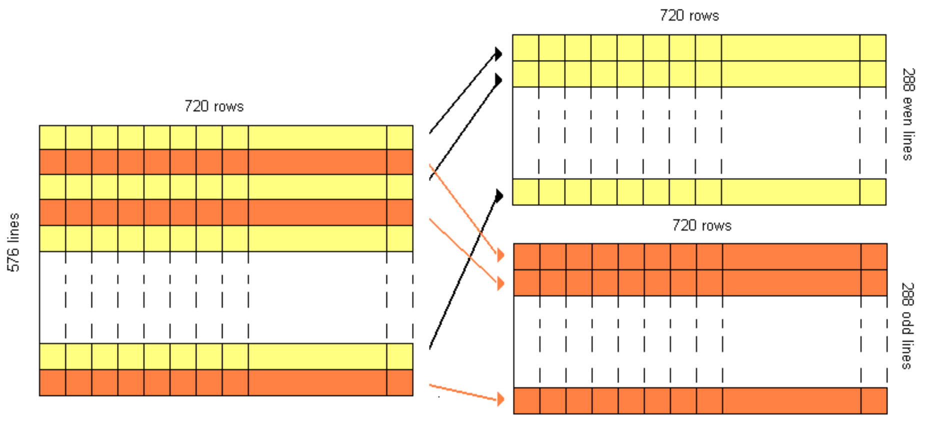
The order which each frame is drawn is: from left to right and from the top to the bottom. On the figure bellow we can see an explanation about how is working.



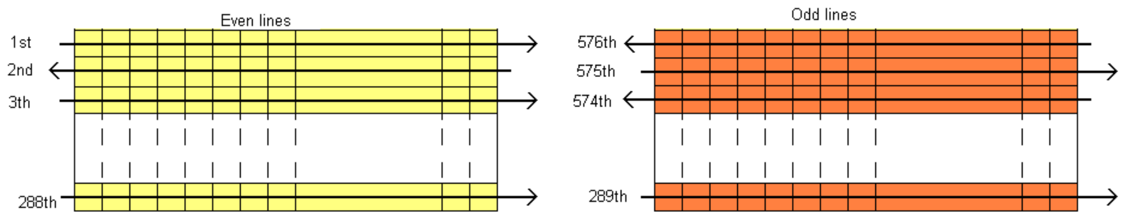
The arrows are indicating the direction of each line while the numbers are indicating the order of each line.

## *PROJECTOR:*

The projector that we use is working optimizing the travel of the mirror. The mirror starts drawing the even lines from top to bottom. Horizontally, the travel is from right to left and from left to right alternating order in each line. After drawing all the even lines, it's time for the odd lines, this time the tour is from the bottom to the top vertically and horizontally starts from right to left and continues left to right alternating the direction in each line. In this way always is drawing a pixel between the last one.
We can understand better if we are working with two half frames, this half frames are compose from even lines or odd lines. In the figure bellow we can see how this division is:



After see how is each half frame we can see how the image draw is:

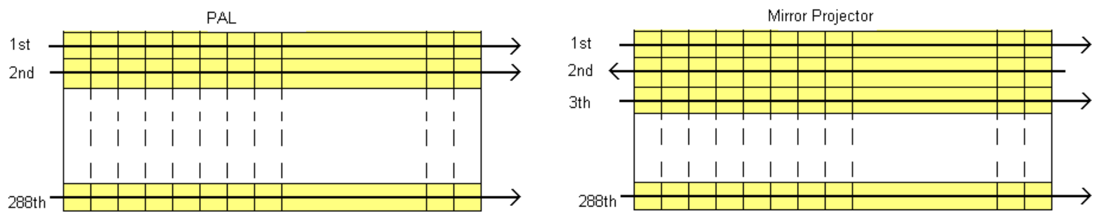


The arrows are indicating the direction of each line while the numbers are indicating the order of each line.

With this information we can think how should be the rearrange of each pixel and we can make the application.

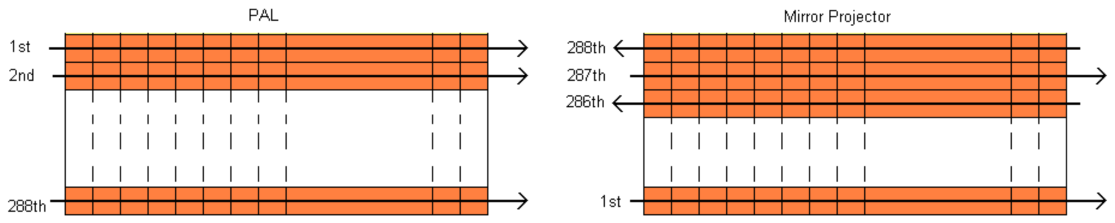
## *REARRANGE:*

Comparing the figures for each method is easy to see how the rearrange should be. For the half frame composed of even lines the way to work is:



The first lines will be copy directly while the odd lines are invert: the first pixel will be the last one an so on.

For the other half frame, the rearrange is more difficult, we should invert the order of all the lines. In addition odd lines of this frame should invert the order too how we see on the figure bellow.



## PROGRAMS:

### *Video_loopback_test.*

This file is common to all the programs and On the annex we can see the code of the file video_loopback_test, in this file we are initializing the parameters on the board, between the code it's commented which are the function of each field.

One parameter that it is relevant is SDOFT. This is controlling how save the values on

VPFE_CCDC_SDOFT=0x00000249;

With the parameter SDOFST from the CCDC we can control how it will be save the image on the memory. We assigned to this parameter the value:

This means that: FINV=0 →No inverse

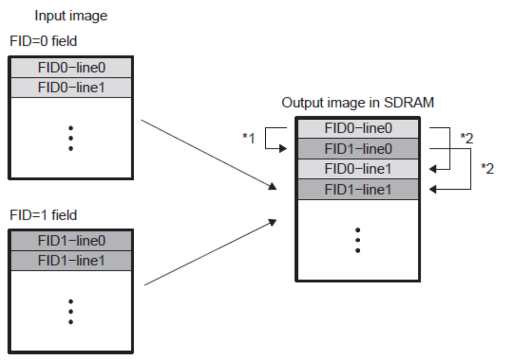
LOFST=0→1 line offset

LOFST0= LOFST1= LOFST2= LOFST3= 1h →2 lines offset

With these values, we are saving on the memory a complete frame, compose of field 0 and field 1, in the way that the picture shown:

- SDOFST.FIINV – invert interpretation of the Field ID signal
- SDOFST.LOFTS0 – offset, in lines, between even lines on even fields (field 0)
- SDOFST.LOFTS1 – offset, in lines, between odd lines on even fields (field 0)
- SDOFST.LOFTS2 – offset, in lines, between even lines on odd fields (field 1)
- SDOFST.LOFTS3 – offset, in lines, between odd lines on odd fields (field 1)

**Figure 20. Frame Image Format Conversion (de-interlaced, 2-field input)**



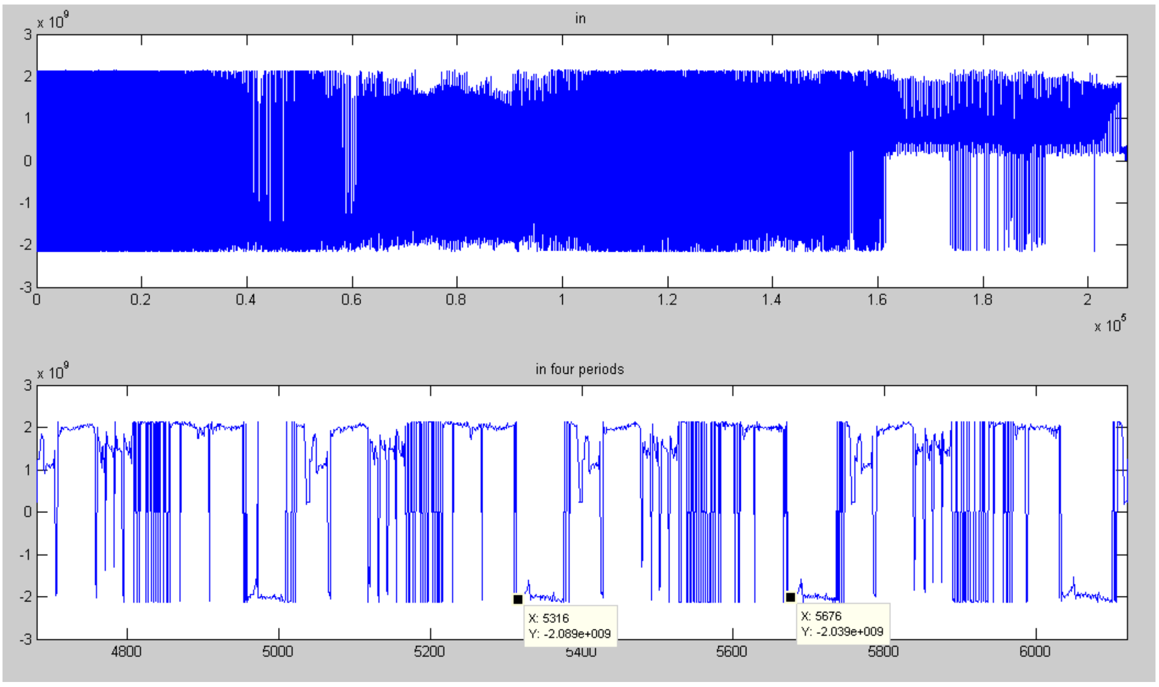
*1 – SDOFST.FOFST = 00b; +1 line for FID=1 line
*2 – SDOFST.LOFTS0=001b; +2 lines
SDOFST.LOFTS1=001b; +2 lines
SDOFST.LOFTS2=001b; +2 lines
SDOFST.LOFTS3=001b; +2 lines

For know more of each field we can consult the manual.

## *One DSP:*

The first program that we run on that DSP was a simply program for copy the input image to the output. This is using a common space on the memory, DDR2, for save the output and input information. On the file video_loopback_test we can see that the parameters VPFE_CCDC_SDR_ADDR in the function that we are initializing the VPFE, and the parameter VPBE_OSD_VIDWIN0ADR on the function for initialize the VPBE have the same value, video_buffer. These parameters save the position on memory where are the input and output information, respectively. If we are assigned the same position the result it will be perfect, without errors and losing information.

On the figure bellow we can see how the information on the memory is:



Analyzing this figure we can see that are some periods, it's easy to see that the each period is corresponding to one line of the image, so we can know how many pixel are defining a line. On the labels of the image and with a simple calculation we can know that the pixels on a line are:

$$Pixels\_lines = 5676 - 5316 = 360$$

It’s quite strange this result because we are working with signals coded according to the standard PAL. This standard defines the image like 720 pixels on a line and 576 pixels on a row.

Finally, we can see that in this case the DSP is working with the half pixels per line.

```
}
```

## Invert program

The next programs that we implement are basic applications for copy and invert the image. We use the same file where are defined some function, three of them are inverting the image in a different way and the other is for copy the image directly. We can see the code and the result in the next pages:

```
#include <stdio.h>
#include <stdlib.h>
#include "stdio.h"
#include "evmdm6437.h"

extern Int16 video_loopback_test();

#define Pixels 207360

Int32   buffer_out[Pixels];    //from 0x80000000
Int32    buffer_in[Pixels];    //from 0x800CA800

#pragma DATA_SECTION(buffer_out,".ddr2")
#pragma DATA_SECTION(buffer_in,".ddr2")

void Copy(void){
        Int32   i;
        Uint32 temp;

        i = 0;

        do{
                        temp=buffer_in[i];
                        buffer_out[i]=temp;
                        i++;
        } while (i < Pixels);
}
```

```
void Inv_1(void){
        Int32   i,j;
        Uint32 temp;

        i = 0;
        j=Pixels-1;

        do{
                        temp=buffer_in[j];
                        buffer_out[i]=temp;
                        i++;
                        j--;
        } while (i < Pixels);
}

void Inv_2(void){

        Int32   i,j;
        Uint32 temp;

        j=Pixels/576-1;
        i = 0;

        do{
                        temp=buffer_in[j];
                        buffer_out[i]=temp;
                        i++;
                        j--;

                        if(i%(Pixels/576)==0){ j=i+Pixels/576-1;}

        } while (i < Pixels);

}
```

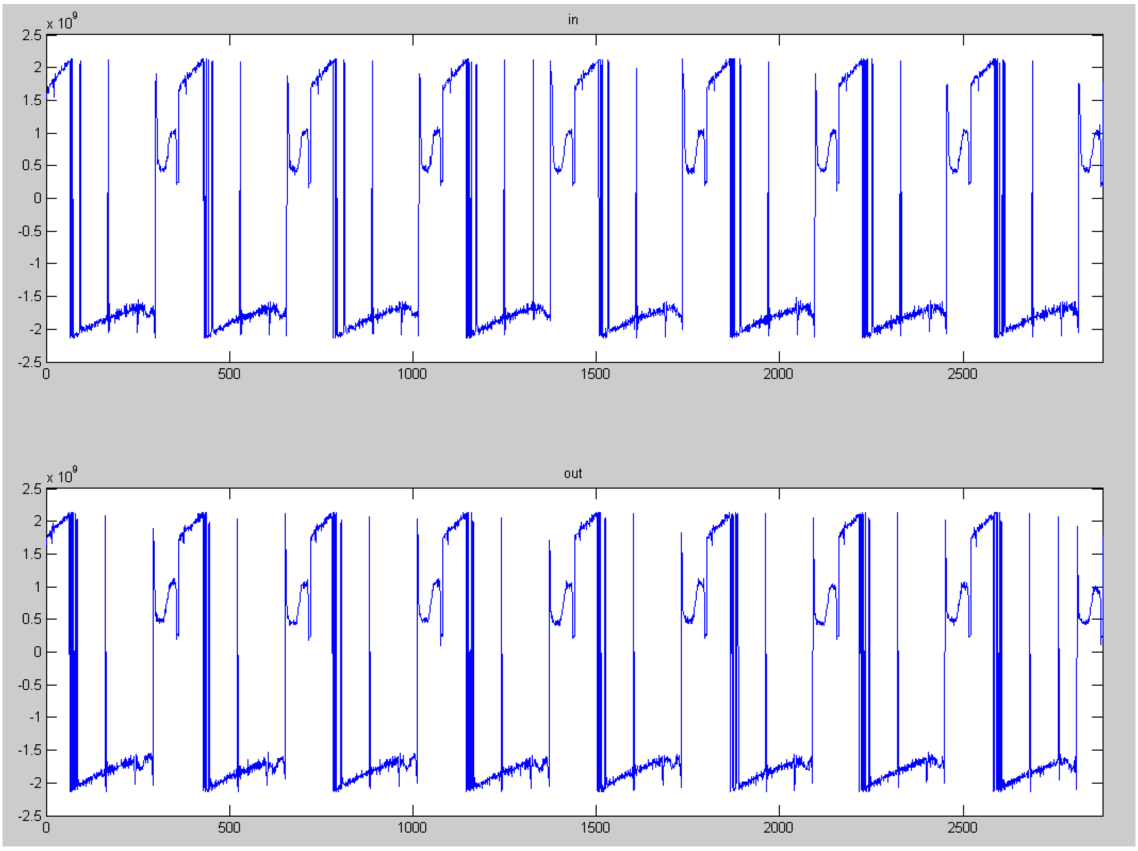
```
void Inv_3(void){

        Int32   i,j;
        Uint32 temp;

        i = 0;
        j=Pixels-Pixels/576;

        do{
                        temp=buffer_in[j];
                        buffer_out[i]=temp;
                        i++;
                        j++;
                        if(i%(Pixels/576)==0){j=Pixels-i-Pixels/576;}

        } while (i < Pixels);
}
void main( void ){

   /* Initialize BSL */
   EVMDM6437_init();

   video_loopback_test();

        while (1){

                //Copy();
                //Inv_1();
                //Inv_2();
                Inv_3();
        }

   printf( "\n***ALL Tests Passed***\n" );
   SW_BREAKPOINT;
}
```

## Copy();

This function copies the signal from the input on the output. In this program we are defining a space on memory for save the information input and output. The input information is saved on the address 0x800CA800 and the output information is saved on the address 0x80000000. In that case they are not sharing the space on the memory and it's for these that when quick movements are, we can appreciate lost information.

The algorithm for do this is simply, we are copying all the values from the buffer_in on the buffer_out in the same way. For do this, we can see on the code that we need to define a local variable. We will use this for move around both buffers and copy the values in the way that at the end both will be the same.

On the figure bellow we can see the result.



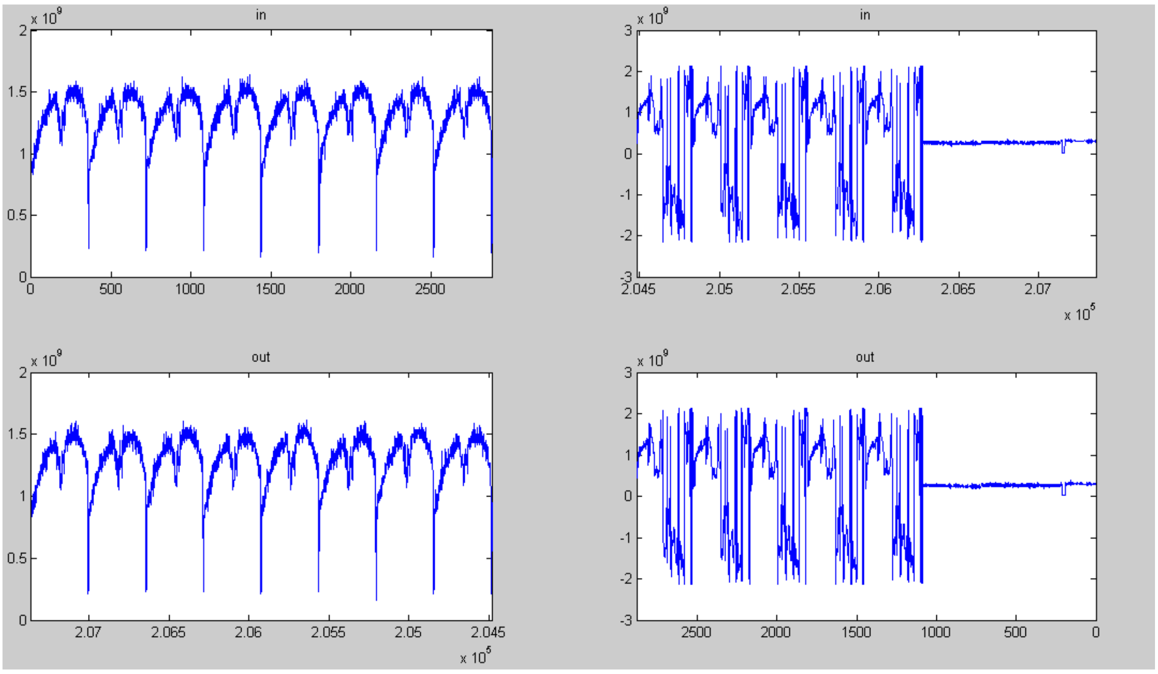
Both are equal like we expected.

## Inv_1():

This function is modifying the image repositioning the pixels. This reposition consists in invert the image having the center like invert point.

For do this applications we need to define two variables more: i, j. This ones are moving around the buffer for selected the value. The variable j is going from the last position of the output buffer to the first one, while the i is going from the first position of the input buffer to the last position. They are decreasing and increasing one position, respectively, en each loop till finally they move around all the positions.

On the figure bellow we can see the results:



The figures on the first row are corresponding to the buffer input, the first and the last values respectively. On the second row are the values of the output buffer, in this case the first figure shows the last values and the second shows the first values.
For compare the results of both buffers we invert the x axe of the output so the signal shape should be the same, and exactly we can see that are equals. Finally, we can say that this is working well.

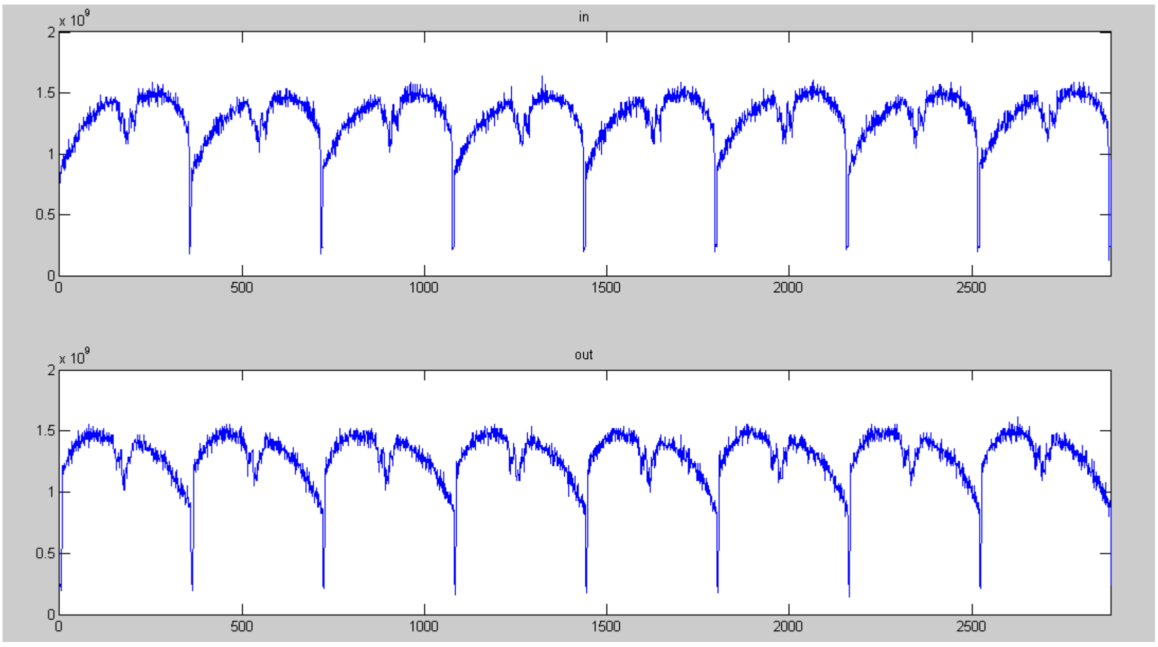
## Inv_2()

With this application we want to flip horizontally the image. We need two variables that are moving around the buffers. The variable i is always increasing while the variable j is always decreasing. The difference between this application and the previous is that we must to control the change of the line and replace the position in the buffer_in to select the correct value.

The algorithm of this program consists in change the order of the pixels in each line. So if we are copying on the buffer_out each pixel in ascendant order we must to control when we start one line for replace the position on the buffer_in to select the last value of the line. We can do this easily with one line of code:

```
if(i%(Pixels/576)==0){ j=i+Pixels/576-1;}
```

The condition always it will be true when we start a new line and in this case we are replacing the position of the buffer_out in the last position of this row. We can see the result on the figures bellow:



In this figure are showing the 2880 values of both buffers, 8 periods are shown and each period means one line. We can appreciate that the shape of one line is invert. With this and watching the screen we can say that this is working well.

## ***Inv_3()***

In this application, the last from the invert images, we want to flip vertically the image. In this way, we want that the last line of the image be the first one. Again we must to control when we are changing the line.

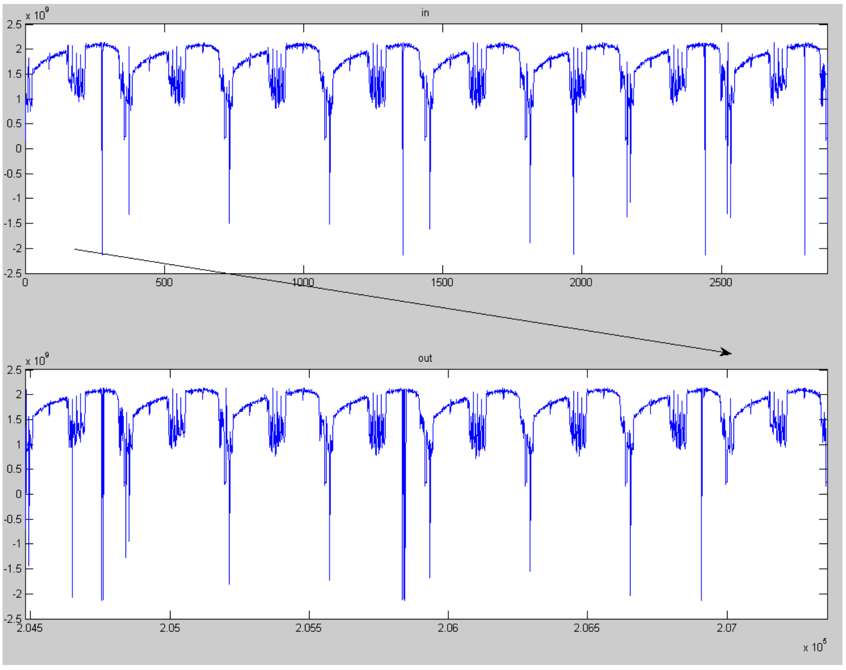
In this case, as we have done in the other cases, we define two variables. The i is indicating in which position of the buffer_out we are and the j is indicating which position we want to copy from buffer_in.
The initialization of j should be the first pixel of the last line. And each time that we are starting a new line we need to replace the position on the buffer in. The line of the code that is controlling this is:
Initialization:

```
j=Pixels-Pixels/576;
```

Control the change line:

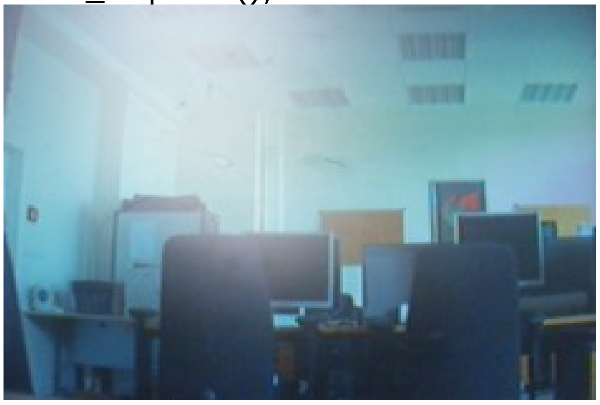
```
if(i%(Pixels/576)==0){j=Pixels-i-Pixels/576;}
```

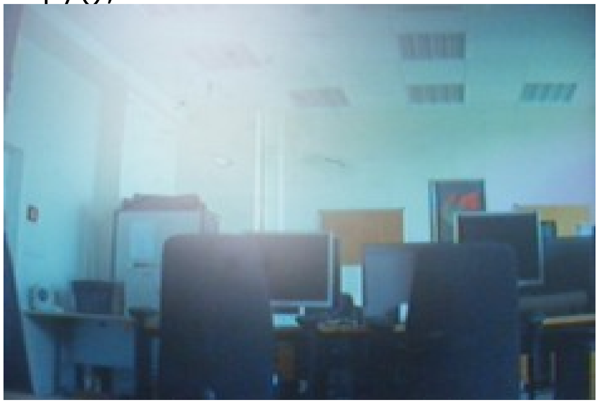
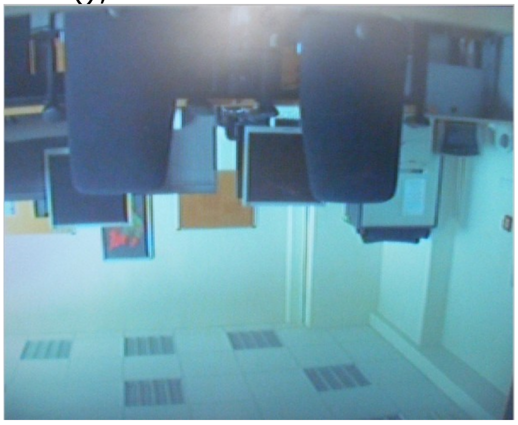
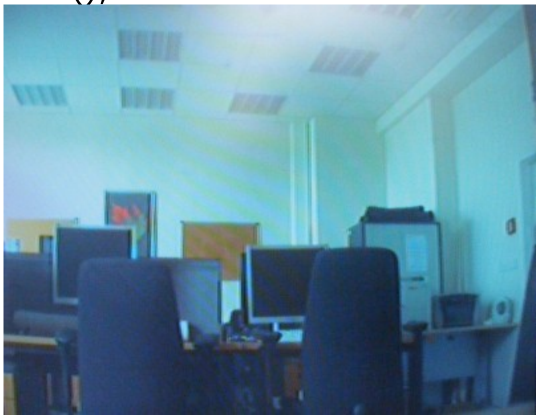
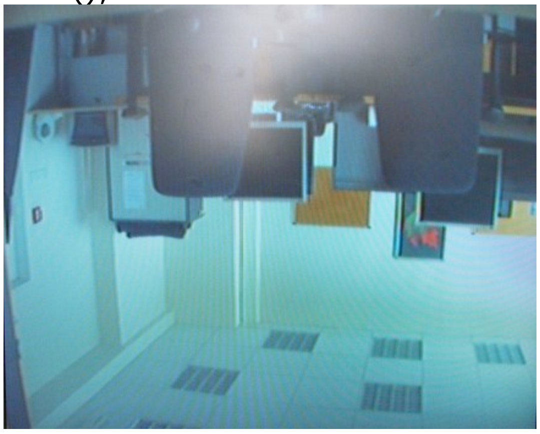
In the figures bellow we can see the result:



We can see how the shape it's the same in each line but in a new place: The first lines at the end.

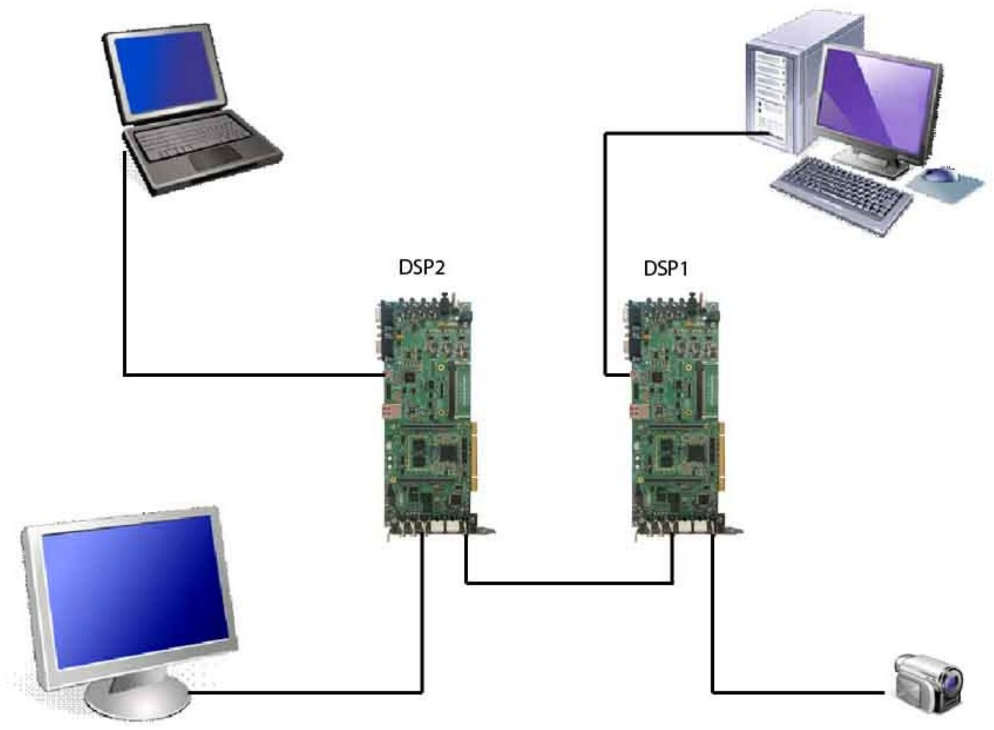
Here we can see the visual effect of each program:

| Original | Video_loopback(); |
|---|---|
| |  |

| Copy(); | Inv1(); |
|---|---|
|  |  |
| Inv2(); | Inv3(); |
|  |  |

## PROGRAMS WITH TWO DSPs

After do the simples programs in one DSP we are ready for start to work with two DSPs. In the final montage is not necessary to use the second DSPs, the reason why we use is because we do not have a mirror projector and the second DSP will behave like this and its output will recover the original image. All devices are connected as shown in the following figure:



After see how should be the application we are ready to do it.

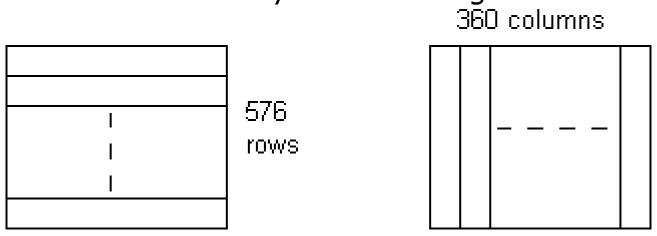
## *How should be:*

This two DSP doesn't have the same code. On the annex we can see the code of the program that we run on the first one.

## Main file of DSP1:

For try to understand better this code we hill explained for parts:

On the initial declarations we have the variable Pixels. This is used for reserve the image on memory and shifts the image for copy on the output. The image is saved on memory like an image 360 x 576 pixels like we can



see on the next figure:

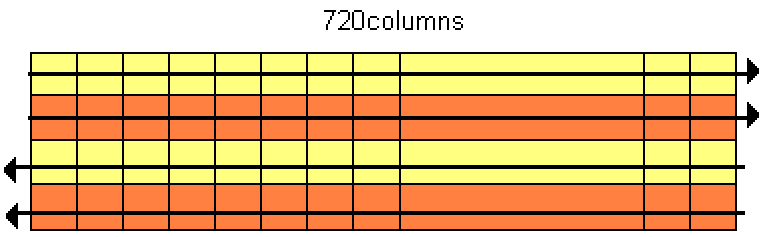
We calculate the value of this variable in the next way:

$$Pixels = row \times columns = 576 * 360 = 207360$$

After that and using this variable we define two buffers: buffer_in and buffer_out. The first one saves the input image and the second one save the image output.

On the Final function is all the rearrange process. We thought with the image like a groups of four lines because each four lines, the rearrange it's repeating. This division achieved with the sentence i% (Pixels/144), where 144=number_of_lines:4. In this way we will have the same values for the lines at the same position in each block of values.

On the first part of this function we are copying the value selected from the buffer_input to the next empty value of the buffer_output, after that we are actualizing the values of i and j for copy the next time. The variable i will be selected the next position while the variable j it will be increasing or decreasing depending of the line. On the figure bellow we can in which lines of this group of four this variable is increasing or decreasing:



The arrow is indicating the direction.

The last *if* from the code is controlling when we are finish the travel around one lines. It's easy to see that if we divided the number of pixels (Pixels) by the numbers of rows, the result is the number of pixels in a line:

$$pixels\_line = \frac{Pixels}{rows} = \frac{Pixels}{576}$$

The first line of this group it will be the same on the output that in the input so we want that when we start this line the value of i and j be the same j=i. And how we see before both values are increasing, we are copying the line exactly.

The third line should be the same that on the input but inverted. The value of j should indicate the last position of this line and should be decreasing.
$j = i + pix\_line - 1 = i + Pixels/576 - 1;$

The second line of this block belongs to the odd lines of frame. The initial value of j should indicate the first pixel of the line in the same position but counting to the end and should be increasing so the result is j=Pixels-i;

And the fourth line of the group should have the values of the line of the same position but counting to the end and should be inverted so:

$$j = Pixels - i + pix\_col - 1 = Pixels - i + Pixels / 576 - 1;$$

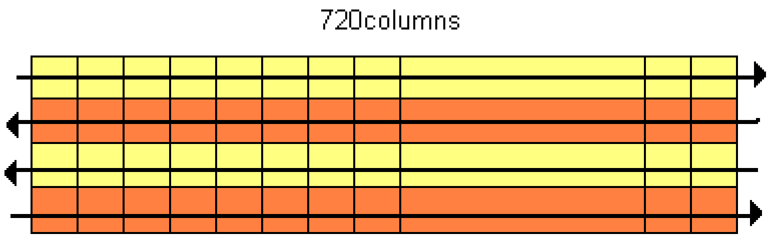
Finally we can see that with this rearrange we prepare the information of the signal to be send to the mirror projector.

## Main file of DSP2:

```
void Final(void){
    Int32       i,j;
    Uint32      temp;

    j = 0;
    i = 0;

    do{
        temp=buffer_in[j];
          buffer_out[i]=temp;
          i++;

        if(i%(Pixels/144)<(3*Pixels/576) && i%(Pixels/144)>(Pixels/576) ){  j--;}
        else{   j++;   }

        if(i%360==0){
              if((i%(Pixels/144))/(Pixels/576)==0){               j=i;     }
              else if((i%(Pixels/144))/(Pixels/576)==1){j=Pixels-i+Pixels/576-1;}
              else if((i%(Pixels/144))/(Pixels/576)==2){ j=i+Pixels/576-1;}
              else if((i%(Pixels/144))/(Pixels/576)==3){ j=Pixels-i;      }
                }
        } while (i < Pixels);
}
```

The only change respect the code that we upload to the first DSP is the Final function. Now we will explain the differences.

In the previous sections we have seen how the projector mirrors works. The rearrange each four lines it is the same so we can work with groups of four lines like in the case before. The value of i is always increasing while the value of j it's depending. On the figure bellow we can see how it is the displacement en each line for this variable:

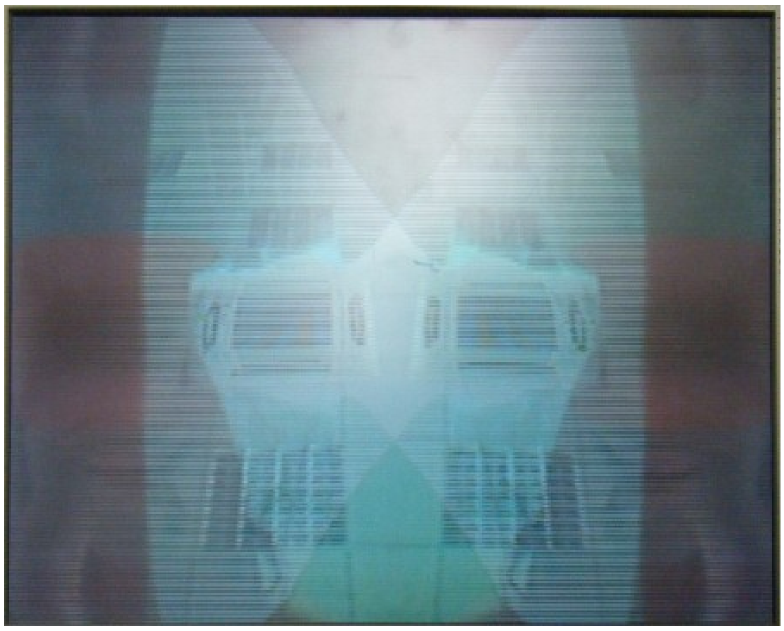


The rearrange for the even lines is the same for both DSPs, if we will process this lines two times at the end we will have the original images.
On the case of the odd lines this doesn't happens if we process the same line two times on the same way, the result it will be the image inverted, so we must to process the second line of this block like the forth line on the block in the first DSP. The result is the code that we show before.

## Results

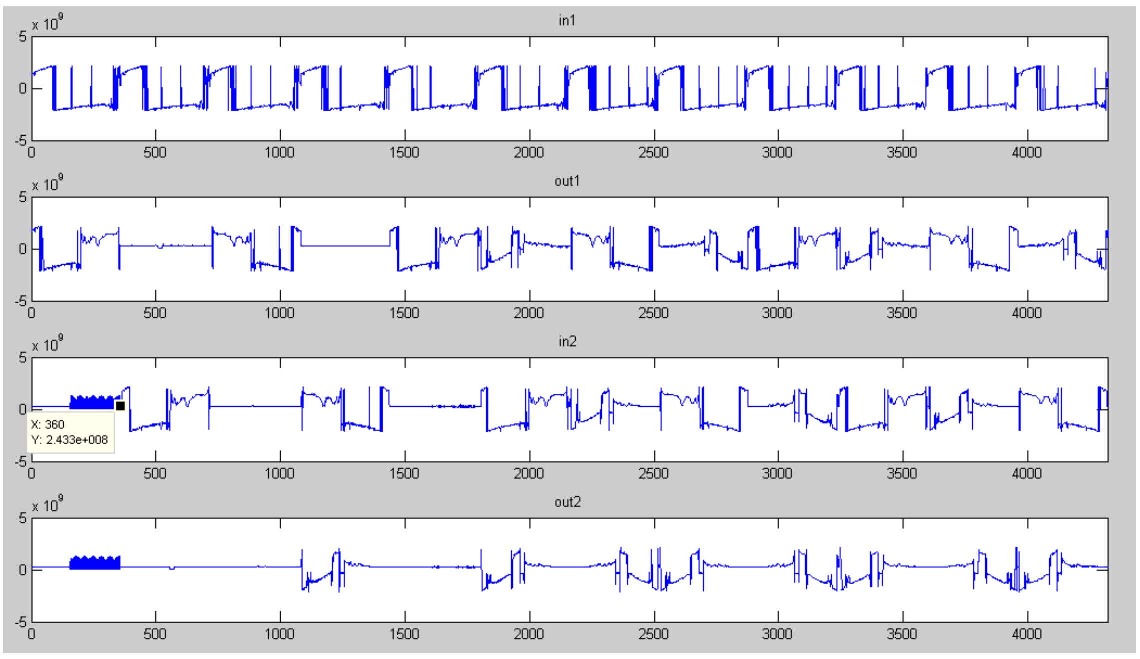
After run both programs in the DSPs we see the results on the screen we can appreciate that something is wrong, it's not working well. On the figures bellow we can see the result after each DSP from a real image.

*After the first DSP*



*After the second DSP:*

For analyse what's wrong we used the program matlab for draw the values from the buffers.



In this figure we can see for graphics. Each one is corresponding to one buffer in order, first from the DSP1: buffer_in and buffer_out, and then from the DSP2: buffer_in and buffer out.

Obviously, the first graphic and the last one are not the same, we can see it this before on the image of the screen.
The most relevant information that we can obtain is the delay from the output of the first DSP to the input of the second. This is a delay of 360 pixels, and we know that this number of pixels is the same as a line. So we have one line delay.

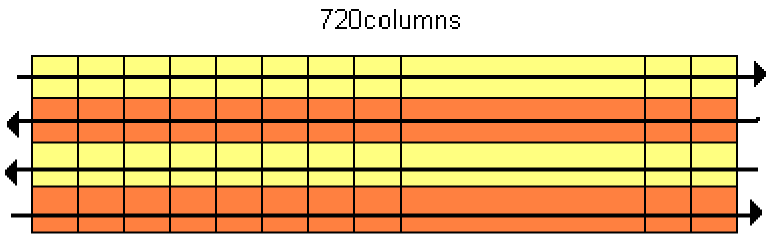
As we don't have oscilloscope we don't know where the delay is occurred so we will do two programs. If the delay is from the output of the DSP we must to change the code of the program of the first DSP while if the delay it's from the input of the second DSP we should change the code of the program from the second DSP. We will do both.

## Rearrange from DSP1:

```
void Final(void){

        Int32   i,j;
        Uint32 temp;

        j = 0;
        i = 0;

        do{
                temp=buffer_in[j];
                buffer_out[i]=temp;
                i++;


                if(i%(4*Pixels/576)<(3*Pixels/576)                  &&                  i
                %(4*Pixels/576)>(Pixels/576) )    {        j--;     }
                else{                    j++;   }

                if(i%(Pixels/576)==0){
                        if((i%(Pixels/144))/(Pixels/576)==0){j=Pixels-i;}
                        else if((i%(Pixels/144))/(Pixels/576)==1){j=i+Pixels/576-1;}
                        else if( (i%(Pixels/144))/(Pixels/576)==2){
                                                        j=Pixels+Pixels/576-i-1;}
                        else if((i%(Pixels/144))/(Pixels/576)==3){        j=i;}
                        }
        } while (i < Pixels);
}
```

The change from the other program resides on the final function. We know that we have a delay of one line, so the first line that we want to send is the second one. In this way on the second DSP this will be processed on the correct way.

We work again with groups of four lines because the process it periodic each 1440 pixels. Like on the other program the value of i is always increasing while the value of j it's depending. On the figure bellow we can see how it is the displacement en each line for this variable:



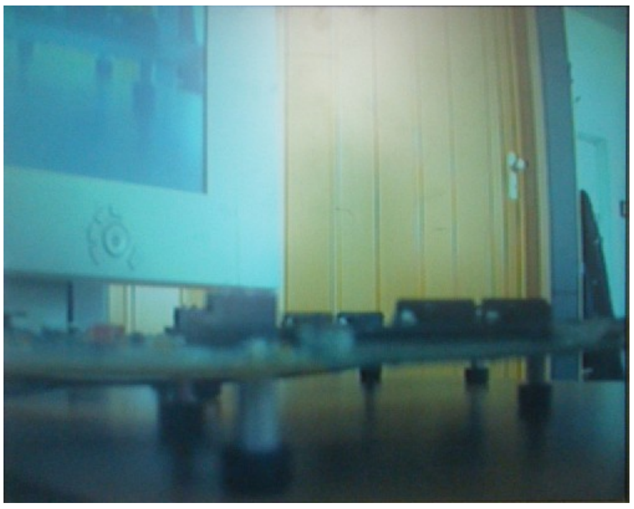
The rearrange for the even lines is the same for both DSPs, if we will process this lines two times at the end we will have the original images.

On this case when we change the line we rearrange the value of j like in the program of the first DSP in the previous case but with three lines of delay. So we are rearrange and saving in the first values of the buffer output the values of the second lines, then the third an so on. The result that we obtain we can see it on the next part.

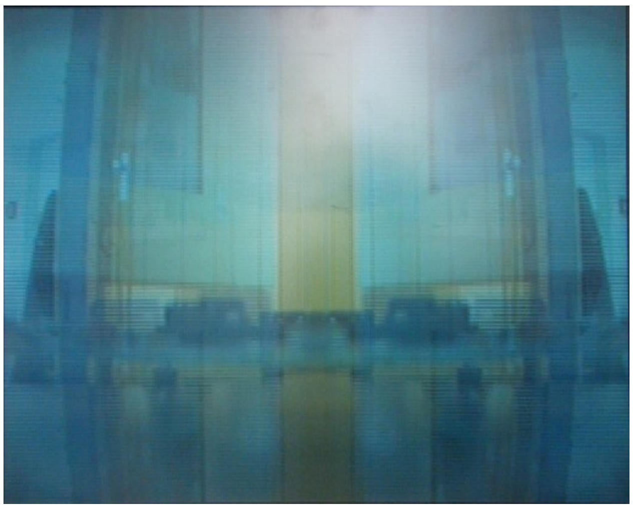
**Result:**

The visual result that we obtain is shown on the figures bellow:

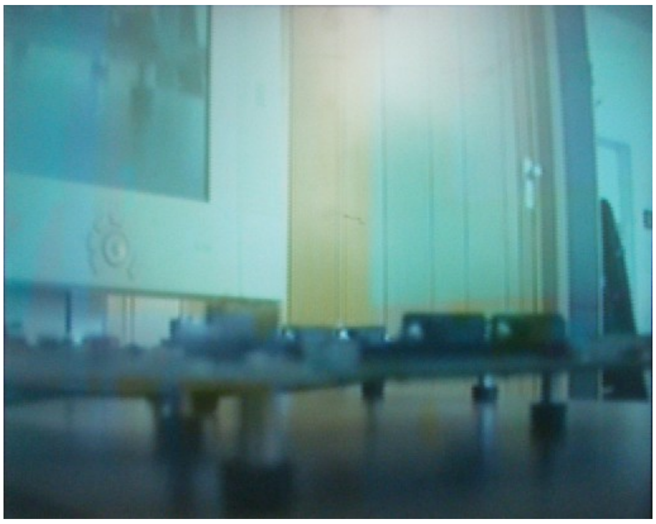
*Original image*



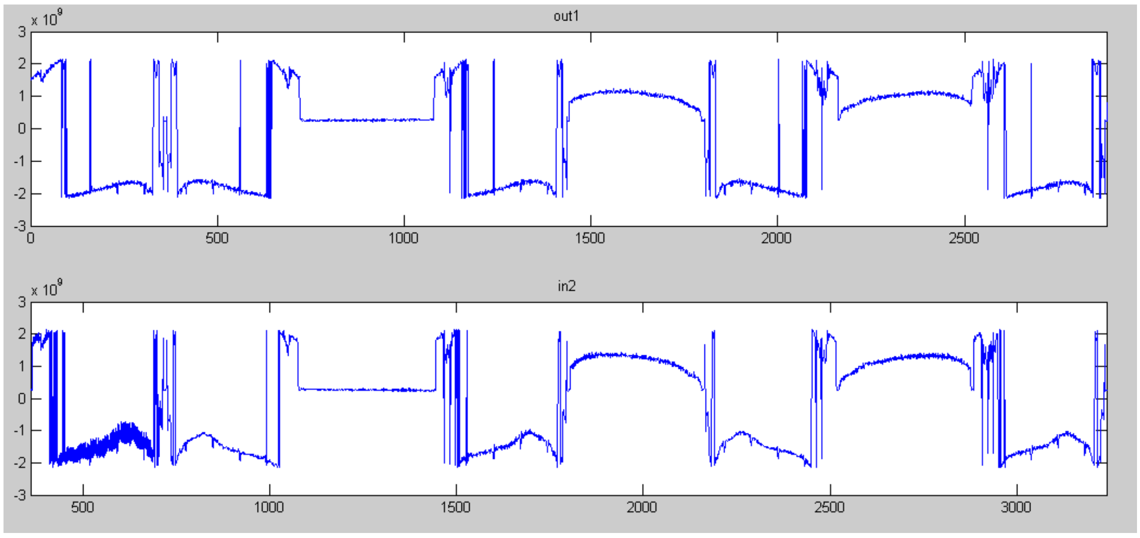
*After the First DSP*



*After the second DSP*



From matlab we will see that it is true that we are processing each line in his place:



In that figure we have the shape of the two buffers that we are interested, buffer_out from the DSP 1 and buffer in from the second DSP. We can see that the delay already exist but on the values of the second line it will be rearrange on his place on the second DSP.

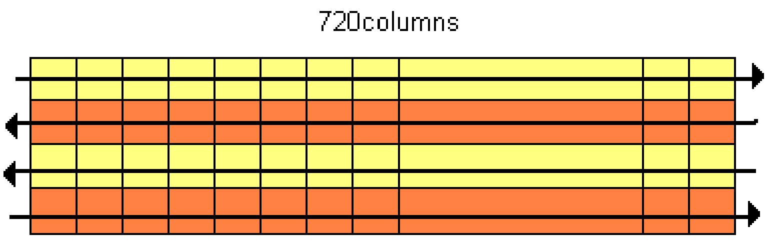
Finally the result it is ok.

## Rearrange from DSP2:

In this case we are changing the code of the Final function on the main file for process the pixels correctly taking into account this delay. The code of this function is:

```
void Final(void){
        Int32   i,j;
        Uint32 temp;

        j = 0;
        i = 0;

        do{
          temp=buffer_in[j];
          buffer_out[i]=temp;
        i++;

          if(i%(Pixels/144)<(3*Pixels/576) && i%(Pixels/144)>(Pixels/576) ){j++;}
          else{j--;     }

         if(i%360==0){
                if((i%(Pixels/144))/(Pixels/576)==0){j=Pixels-i+Pixels/576-1; }
                else if((i%(Pixels/144))/(Pixels/576)==1){j=i;}
                else if((i%(Pixels/144))/(Pixels/576)==2){j=Pixels-i;            }
                else if((i%(Pixels/144))/(Pixels/576)==3){j=i+Pixels/576-1;   }
          }
        } while (i < Pixels);
}
```

For explain the code we must to think again with groups of four lines. Like on the other program the value of i is always increasing while the value of j it's depending. On the figure bellow we can see that for the first an fourth line, the value of j is increasing and for the others is decreasing.

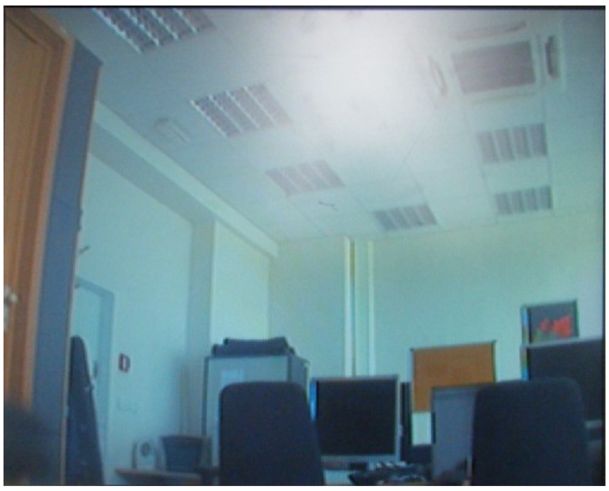


The rearrange for the even lines is the same for both DSPs, if we will process this lines two times at the end we will have the original images.

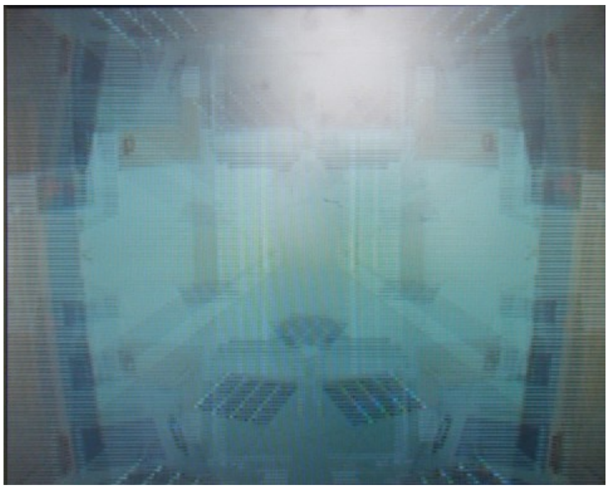
The actualization of the value of j is similar to the other programs, we can see that are the same values but rearranged to the line that corresponds now.

**Result:**
On the figures bellow we can see which the visual effect in each part is:

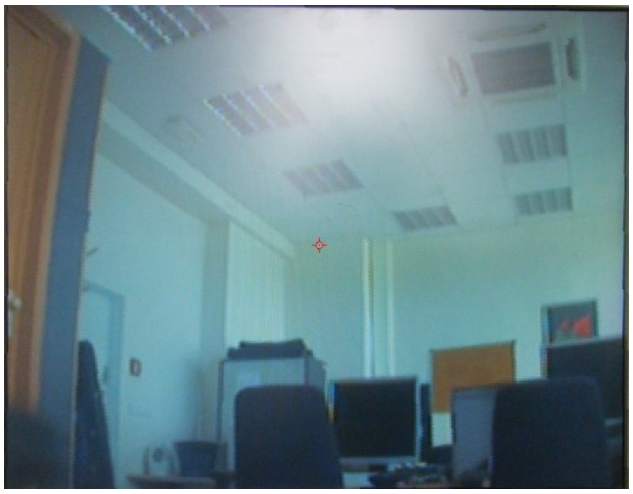
Original:



After first DSP

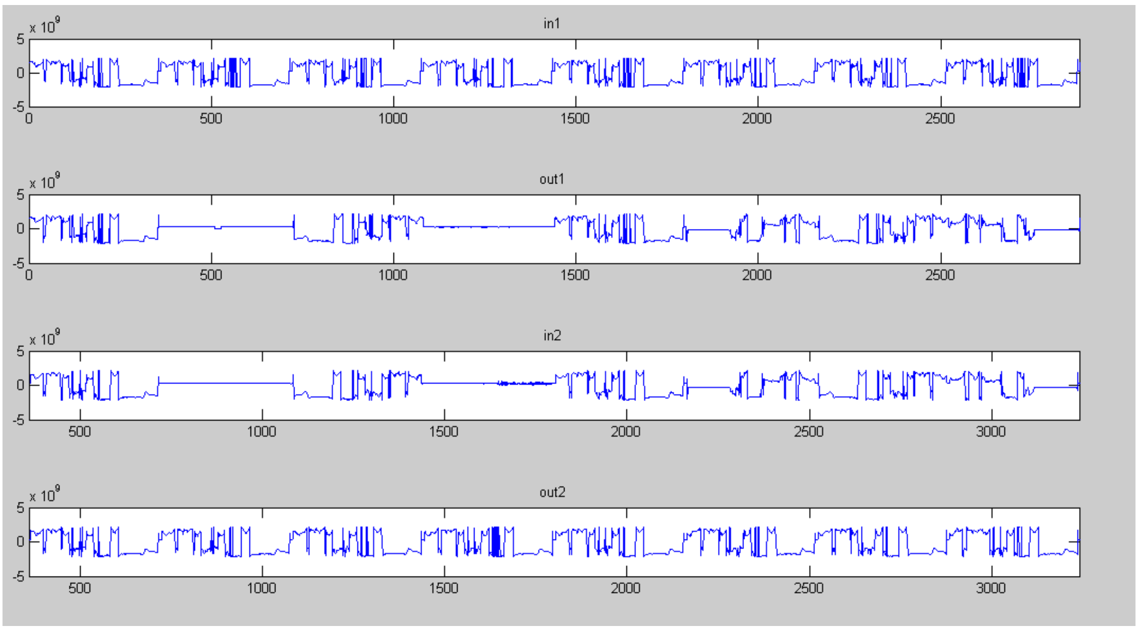


After 2nd DSP



We can observe that the last image it’s exactly the same image that the input.

With the help of matlab we can see better what's happening. On the figure bellow we are representing the shape of the first values of the buffer in, and from the buffer out we are showing the corresponding values, with the delay.



In this figure we can observe that the input image and the output image are the same, the same shape is indicating the same image. And from the buffer out of the first DSP to the buffer in of the second one we have this delay of one line, but with our code the process is correct.

## Problems:

Before to dispose the software matlab for analyze the values on the buffer, we only could check that the memory was of the size that we defined but we don't really can appreciate what was happening. For this, we spent many time trying to solve a problem that before we thought that was a problem of synchronizing of the boards or delay. After, we realized that the lines were not saving on the memory like we expect: 720 pixels per line. It was saving the half number of pixels per line: 360. From that moment this was the most important problem that we have.

This apparently is not affecting to the image but the reality is that we are losing much information. If we had had an input image where each column is a color, we could really appreciate what happen with these rows and what can we do, but we don't dispose this tester. First off all we thought of designing on the first DSP and place it in the second, but this is not possible as they exit has the same problem.

In that moment we start to revise all the values of the initialization fields and in every one the pixels per line was defined like 720 pixels.
The parameter PIX_LINES on the VPFE, which is defining the number of pixels per line and the half lines per frame is initializing correctly but this is disabled if the VD and HD are defined like an inputs, so we change the first bit of the parameter SYN_MODE for change the direction to Output. Unfortunately, this was not working.

After that we study the block diagrams that are on the manual of this VPFE from Texas Instrument for try to find if it is some bit blocking this value, but we didn't find the solution.

We thought that maybe when the board it's processing an image without upload an application, the definition of the line was the correct, but the reality is that the DSP has a program on the flash memory that is running per defect when we connect the board to the alimentation and unfortunately this program define the line like 360 pixels again.

The next step that we have done, and after make many tests, was check all the values of the field if they have been uploaded correctly to the memory. For this we find which address corresponds to each parameter and then see that is correctly uploaded. All off them where save to the memory correctly, only where some changes on some reserved bits.

The other problem that we have is the delay. Without an oscilloscope for see how the analogical signal is we can know where exactly the problem is and how to solve. Our solution was to change the code for adapt to this circumstance for the application run well.

## Conclusions:

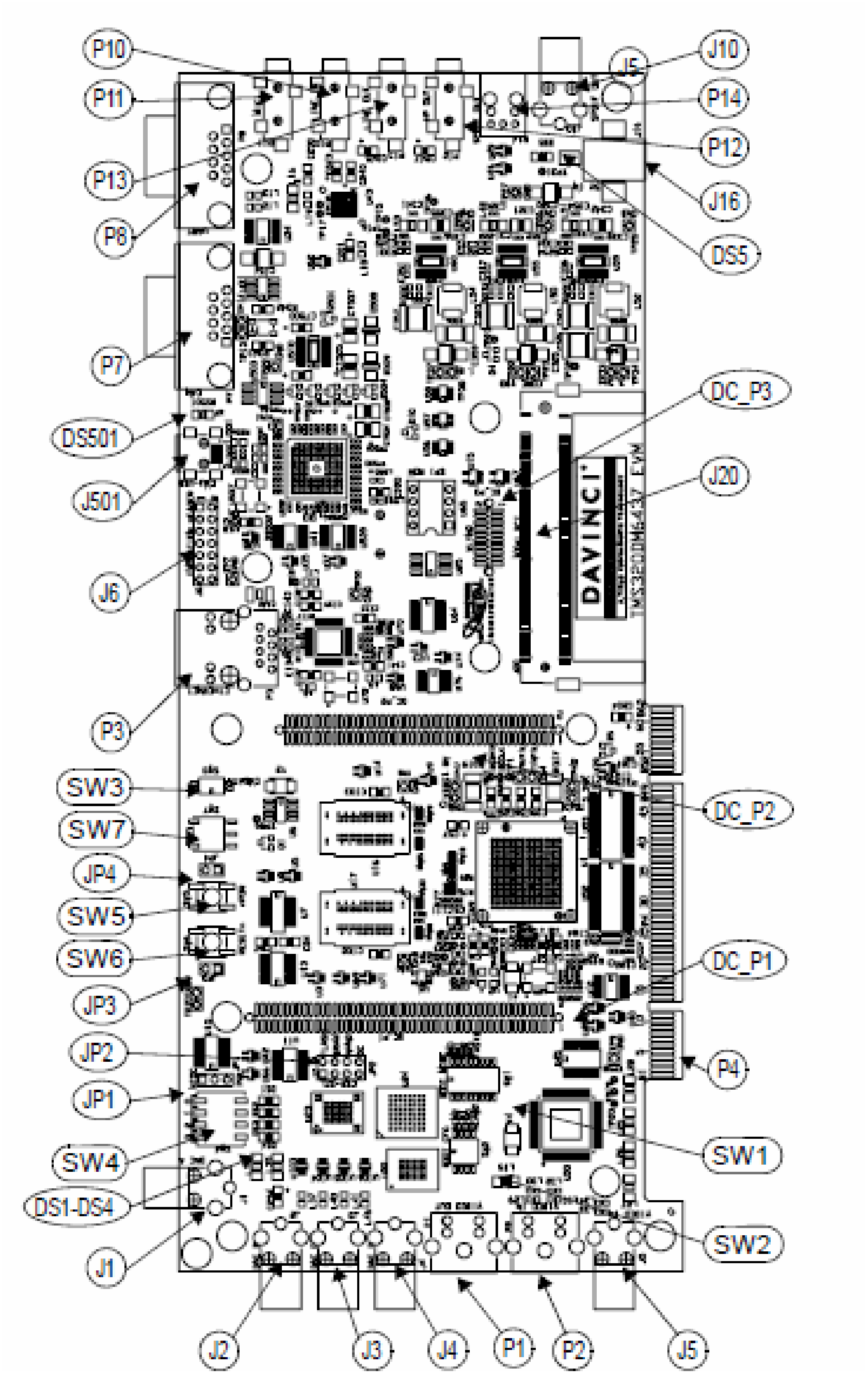
This work is to create an application to connect a PAL signal in a projector. Like we saw on the report, there are some problems yet. The first, and most important, is the definition of pixels per line following, the delay between the two boards.

The next step to continue this study should be to analyze the analogy signals that on the input and output analogy each of the DSP. This can be possible with an oscilloscope. In this way we can obtain more information, about how is the analogy signal reconstruct and thus would be easier to find a solution.

Finally, despite these problems, we managed to reconstruct the original image and reproduced on a screen. This is after being processed twice, once by placing the pixels and the second repositioned.

## Annexes:

### *Schematic figure of TMS320DM6437:*



## *Code video_loopback_test*

```
/*
 *  Video Loopback Test
 *
 */

#include "stdio.h"
#include "evmdm6437_dip.h"
#include "tvp5146.h"

#define NTSC            1
#define PAL             0

#define LOOPBACK        0

#define SVIDEO_OUT      1
#define COMPOSITE_OUT   0

#define mem_buffer_out 0x80000000     //Start address for the buffer_out
#define mem_buffer_in 0x800CA800      //Start address for the buffer_in

/* ----------------------------------------------------------------------   *
 *                                                                          *
 *  vpfe_init( ntsc/pal )                                                   *
 *                                                                          *
 * ----------------------------------------------------------------------   */
static void vpfe_init( Uint32 ntsc_pal_mode )
{
   Uint32 video_buffer = mem_buffer_in;
   Uint32 width;
   Uint32 height;

   //Definition of the image in number of pixels
   if ( ntsc_pal_mode == NTSC ){
      width   = 720;
```

```
     height  = 480;
  }   else   {
     width   = 720; //Number of pixels on a line
     height  = 576; //Number of lines
  }

  VPFE_CCDC_PCR = 0x00000001;             //CCD controller enabled
  VPFE_CCDC_SYN_MODE  = 0x00032F85;       // interlaced, with VD polarity as
                                          //negative
  VPFE_CCDC_HD_VD_WID = 0;
  VPFE_CCDC_PIX_LINES = 0x02CF020D;       //Defined the pixels per line and
                                          the //half lines per frame

  /*
   *  sph = 1, nph = 1440, according to page 32-33 of the CCDC spec
   *  for BT.656 mode, this setting captures only the 720x480 of the
   *  active NTSV video window
   */
  VPFE_CCDC_HORZ_INFO = width << 1;       //  Horizontal  pixels  that  are
                                          output //to  the SDRAM
  VPFE_CCDC_HSIZE_OFF = width << 1;   // Offset for each output line in SDRAM
  VPFE_CCDC_VERT_START = 0;           // Vertical start line
  VPFE_CCDC_VERT_LINES = height >> 1;     //  Vertical  lines  that  are  output
                                          to //the SDRAM
  VPFE_CCDC_CULLING   = 0xFFFF00FF;   // Disable culling

  /*
   *  Interleave the two fields
   */
  VPFE_CCDC_SDOFST    = 0x00000249;       //How the values are save on the
                                          //memory
  VPFE_CCDC_SDR_ADDR  = video_buffer;     //Start  address  save  the  input
values
  VPFE_CCDC_CLAMP     = 0;                //Clamp disabled
  VPFE_CCDC_DCSUB     = 0;
  VPFE_CCDC_COLPTN    = 0xEE44EE44;       //Color pattern
  VPFE_CCDC_BLKCMP    = 0;            //Compensation of black, no activated
  VPFE_CCDC_FPC       = 0;            //Disable the replace of the pixels defined
```

```
                                          //on the next line
   VPFE_CCDC_FPC_ADDR  = 0;               //Replace pixel x with the average of x-1
                                          //and x+1
   VPFE_CCDC_VDINT     = 0;
   VPFE_CCDC_ALAW      = 0;                       //Apply gamma(A-law) disabled
   VPFE_CCDC_REC656IF  = 0x00000003;        //Enable   de   REC656   interface
                                                  and //FVH error correction

   /*
    *  Input format is Cb:Y:Cr:Y, w/ Y in odd-pixel position
    */
   VPFE_CCDC_CCDCFG    = 0x00000800;        //Location of Y signal odd pixel
   VPFE_CCDC_FMTCFG    = 0;
   VPFE_CCDC_FMT_HORZ  = 0x000002D0;        //Number of pixels on horizontal
                                                  //direction=720
   VPFE_CCDC_FMT_VERT  = 0x00000240;        //Number of pixels on vertical
                                                  //direction=576
   VPFE_CCDC_FMT_ADDR0 = 0;
   VPFE_CCDC_FMT_ADDR1 = 0;
   VPFE_CCDC_FMT_ADDR2 = 0;
   VPFE_CCDC_FMT_ADDR3 = 0;
   VPFE_CCDC_FMT_ADDR4 = 0;
   VPFE_CCDC_FMT_ADDR5 = 0;
   VPFE_CCDC_FMT_ADDR6 = 0;
   VPFE_CCDC_FMT_ADDR7 = 0;
   VPFE_CCDC_PRGEVEN_0 = 0;
   VPFE_CCDC_PRGEVEN_1 = 0;
   VPFE_CCDC_PRGODD_0  = 0;
   VPFE_CCDC_PRGODD_1  = 0;
   VPFE_CCDC_VP_OUT    = 0x041A2D00;
   VPFE_RESZ_PCR=0;                       //Disabled resizer module
}
```

```
/* ----------------------------------------------------------------------   *
 *                                                                          *
 *  vpbe_init( colorbars/loopback, ntsc/pal, svideo/composite )            *
 *                                                                          *
 * ----------------------------------------------------------------------   */
static void vpbe_init( Uint32 colorbar_loopback_mode, Uint32 ntsc_pal_mode,
Uint32 output_mode )
{
   Uint32 video_buffer = mem_buffer_out;
   Uint32 basep_x;
   Uint32 basep_y;
   Uint32 width;
   Uint32 height;

   if ( ntsc_pal_mode == NTSC )
   {
      basep_x = 122;
      basep_y = 18;
      width   = 720;
      height  = 480;
   }
   else
   {
      basep_x = 132;
      basep_y = 22;
      width   = 720;
      height  = 576;
   }

   /*
    * Setup VPBE
    */
   VPSS_CLK_CTRL= 0x00000018;   //Enable DAC and VENC clock, both at 27 MHz
   VPBE_PCR          = 0;                   // No clock div, clock enable

   /*
    * Setup OSD
 */
VPBE_OSD_MODE       = 0x000000fc;   // Background color blue using clut in
                                    // ROM0
VPBE_OSD_OSDWIN0MD  = 0; // Disable both OSD windows and cursor window
VPBE_OSD_OSDWIN1MD  = 0;
VPBE_OSD_RECTCUR    = 0;                    //Cursor displayed disabled
VPBE_OSD_VIDWIN0OFST = width>>4 ;          // line with in pixels/16
VPBE_OSD_VIDWIN0ADR = video_buffer;        //SDRAM   source   address   Video
                                            //Window 0
VPBE_OSD_BASEPX     = basep_x;      //Sets the maximum image resolution
VPBE_OSD_BASEPY     = basep_y;
VPBE_OSD_VIDWIN0XP  = 0;                    //Start position
VPBE_OSD_VIDWIN0YP  = 0;
VPBE_OSD_VIDWIN0XL  = width;                //Video window width, horizontal
                                            //display in pixels
VPBE_OSD_VIDWIN0YL  = height >> 1;          //Vertical display height
VPBE_OSD_MISCCTL    = 0;                    //Miscellaneous options
```

```
 VPBE_OSD_VIDWINMD   = 0x00000003;          // Disable vwindow 1 and enable
                                            // vwindow 0. Frame mode with no
                                            //up-scaling

   /*
    *  Setup VENC
    */

   if ( ntsc_pal_mode == NTSC )
      VPBE_VENC_VMOD  = 0x00000003;   // Standard NTSC interlaced output
   else
      VPBE_VENC_VMOD  = 0x00000043;   // Standard PAL interlaced output

   VPBE_VENC_VIDCTL     = 0x030;
   VPBE_VENC_VDPRO      = 0x00;                        //CBMD bit
                                                       //VDPRO es per als colors
   VPBE_VENC_DACTST     = 0;
   VPBE_VENC_DACSEL     = 0x00004210;

   /*
    *  Choose Output mode
    */

   if ( output_mode == COMPOSITE_OUT )
      VPBE_VENC_DACSEL = 0x00000000;
   else if ( output_mode == SVIDEO_OUT )
      VPBE_VENC_DACSEL = 0x00004210;
}
```

```
/* ---------------------------------------------------------------------   *
 *                                                                          *
 *  video_loopback_test( )                                                  *
 *                                                                          *
 * ---------------------------------------------------------------------   */

Int16 video_loopback_test( ){
   Int16 ntsc_pal_mode;
   Int16 output_mode;

   EVMDM6437_DIP_init( );
   {
      /* Check Video Settings */
      ntsc_pal_mode = EVMDM6437_DIP_get( JP1_JUMPER );    // NTSC or PAL
      output_mode   = EVMDM6437_DIP_get( SW7_SWITCH ); //SVideo/Composite

      if ( ntsc_pal_mode == NTSC )       {

         if ( output_mode == COMPOSITE_OUT )
            printf( "    Video Loopback test: [NTSC][COMPOSITE]\n" );
         else if ( output_mode == SVIDEO_OUT )
            printf( "    Video Loopback test: [NTSC][S-VIDEO]\n" );
         else
            return -1;
      }
      else if ( ntsc_pal_mode == PAL )
      {
         if ( output_mode == COMPOSITE_OUT )
            printf( "    Video Loopback test:  [PAL][COMPOSITE]\n" );
         else if ( output_mode == SVIDEO_OUT )
            printf( "    Video Loopback test:  [PAL][S-VIDEO]\n" );
         else
            return -1;
      }
      else
         return -2;

      /* Setup Front-End */
```

```
        tvp5146_init( ntsc_pal_mode, output_mode );
        vpfe_init( ntsc_pal_mode );

        /* Setup Back-End */
        vpbe_init( LOOPBACK, ntsc_pal_mode, output_mode );
    }  ;
    return 0;
}
```

## *Code tvp5146:*

```
/*
 *  Copyright 2006 by Spectrum Digital Incorporated.
 *  All rights reserved. Property of Spectrum Digital Incorporated.
 */

/*
 *  TVP5146 Video Decoder
 */

#include "tvp5146.h"

Uint8 rom_version;
Uint8 chipid_msb;
Uint8 chipid_lsb;

/* ------------------------------------------------------------------------   *
 *                                                                          *
 *  tvp5146_rset                                                              *
 *                                                                            *
 *      Set codec register regnum to value regval                             *
 *                                                                            *
 * ------------------------------------------------------------------------   */
void tvp5146_rset( Uint8 regnum, Uint8 regval )
{
    Uint8 cmd[2];
    cmd[0] = regnum;    // 8-bit Register Address
    cmd[1] = regval;    // 8-bit Register Data

    EVMDM6437_I2C_write( TVP5146_I2C_ADDR, cmd, 2 );
}

/* ------------------------------------------------------------------------   *
 *                                                                            *
 *  tvp5146_rget                                                              *
 *                                                                            *
 *     Return value of codec register regnum                          *
 *                                                                    *
 * ----------------------------------------------------------------   */
Uint8 tvp5146_rget( Uint8 regnum )
{
   Uint8 cmd[2];

   cmd[0] = regnum;    // 8-bit Register Address
   cmd[1] = 0;         // 8-bit Register Data

   EVMDM6437_I2C_write( TVP5146_I2C_ADDR, cmd, 1 );
   EVMDM6437_I2C_read ( TVP5146_I2C_ADDR, cmd, 1 );

   return cmd[0];
}
```

```
/* ----------------------------------------------------------------------   *
 *                                                                          *
 *  tvp5146_init( )                                                         *
 *                                                                          *
 *     Initialize the TVP5146                                               *
 *     NTSC/PAL/SECAM 4x10-Bit Digital Video Decoder              *
 *    With Macrovision Detection, YPbPR/RBG Inputs                          *
 * ----------------------------------------------------------------------   */

void tvp5146_init( Uint32 ntsc_pal_mode, Uint32 input_mode )
{
   rom_version = tvp5146_rget( 0x70 );
   chipid_msb  = tvp5146_rget( 0x80 );
   chipid_lsb  = tvp5146_rget( 0x81 );

   _waitusec( 1000 );                 // wait 1 msec

   if ( rom_version < 8 )
   {
      tvp5146_rset( 0xE8, 0x02 );     // Initalize TVP5146, must do after power on
      tvp5146_rset( 0xE9, 0x00 );     // Skip if using TVP5146-M2
      tvp5146_rset( 0xEA, 0x80 );
      tvp5146_rset( 0xE0, 0x01 );
      tvp5146_rset( 0xE8, 0x60 );
      tvp5146_rset( 0xE9, 0x00 );
      tvp5146_rset( 0xEA, 0xB0 );
      tvp5146_rset( 0xE0, 0x01 );
      tvp5146_rset( 0xE0, 0x00 );

      _waitusec( 1000 );              // wait 1 msec
   }

   switch( input_mode )
   {
      case SVIDEO_IN:
         tvp5146_rset( 0x00, 0x46 ); // Input Video: S-video: VI_2_C(Y) VI_1_C(C)
         break;
```

```
        case COMPONENT_IN:
            tvp5146_rset( 0x00, 0x05 ); // Input Video: CVBS : VI_2_B
            break;
    }

    switch( ntsc_pal_mode )
    {
        case NTSC:
            tvp5146_rset( 0x02, 0x01 );
            break;
        case PAL:
            tvp5146_rset( 0x02, 0x02 );
            break;
    }

    tvp5146_rset( 0x34, 0x11 );         // Enabling clock & Y/CB/CR input format

    _waitusec( 1000 );                  // wait 1 msec
}
```

## ***Code first DSP, how should be:***

```
#include <stdio.h>
#include <stdlib.h>
#include "stdio.h"
#include "evmdm6437.h"

extern Int16 video_loopback_test();

//Definitions
#define Pixels 207360        //207360 number of pixels that we save on memory

Int32   buffer_out[Pixels];     //here is the output information
Int32    buffer_in[Pixels];     //here is the input information


//Charge the values from the initial address that is defined on video_loopback_test
file

#pragma DATA_SECTION(buffer_out,".ddr2")     //buffer_out from 0x80000000
#pragma DATA_SECTION(buffer_in,".ddr2")      //buffer_in from 0x800CA800

//Rearange function
void Final(void){
        //local variables
        Int32   i,j;
        Uint32 temp;

        j = 0;  //used for select a value from the buffer_in
        i = 0;  //used for select a value from the buffer_out

        do{
                //Copy the value on the position j of the buffer_in on the position i
                from //the buffer_out

                temp=buffer_in[j];
                buffer_out[i]=temp;
```

```
i++;              //always are saving the next value on the buffer_out

//With j variable we select which value from the buffer in we want to
//copy on the buffer_out.

//j is this variable which is moving for rearrange the pixels.
if(i%(Pixels/144)<(Pixels/288)  ){ j++;   }
else{                      j--;     }

if(i%(Pixels/576)==0){
         //1st line of the group of four
         if((i%(Pixels/144))/(Pixels/576)==0){     j=i;}

         //2nd line of the group of four

         else if((i%(Pixels/144))/(Pixels/576)==1){j=Pixels-i;}
```

```
                              //3rd line of the group of four
                              else if((i%(Pixels/144))/(Pixels/576)==2){j=i+Pixels/576-1;}

                              //4th line of the group of four
                         else if((i%(Pixels/144))/(Pixels/576)==3){j=Pixels+Pixels/576-i-1;}
                        }
              } while (i < Pixels);
}

void main( void ){
          /* Initialize BSL */
          EVMDM6437_init();
          //function from the video_loopback_test file
          video_loopback_test();



          while (1){
                    Final();
          }

    printf( "\n***ALL Tests Passed***\n" );
    SW_BREAKPOINT;
}
```