A Remote Laboratory for Real-Time Digital Image Processing on Embedded Systems

doi:10.3991/ijoe.v5i4.1064

A. Kalantzopoulos, D. Markonis and E. Zigouris
University of Patras, Patras, Greece

***Abstract*—The purpose of this paper is to present a Remote Laboratory on embedded systems focused in real-time digital image processing. This laboratory consists of a Main Web Server and several Workstations which are designed for digital image retrieval from a CMOS Image Sensor and real-time image processing on a Digital Signal Processor development platform. The Main Web Server redirects the authorised remote users to available Workstations in order to execute and verify their image processing algorithms or test their system designs using a developed Application Programming Interface. Through user-friendly web pages users can interfere with the hardware parameters and observe the results of their solutions.**

***Index Terms*—Digital Signal Processors, e-learning, Real-time Image Processing, Remote Laboratories.**

I. INTRODUCTION

Distance learning is an area which is gaining ground rapidly on the educational process during the last decades. However, a certain limitation arose due to the fact that physical presence of the students was required on traditional hands-on laboratories during practical sessions. Remote Laboratories [1, 2] is the newest approach that came to overcome this obstacle while offering higher availability of laboratory equipment and reduction of the total cost. It provides better exploitation of students' and tutors' time as it allows the conducting of experiments by the students on their own place without time restrictions. This approach also enables the sharing of hardware between institutions for minimization of expense and encourages co-operation between scientific faculty.

This paper presents the design and implementation of a Remote Laboratory for real-time digital image processing on embedded systems based on Digital Signal Processors (DSPs). Due to the rapid growth of the scientific research on the fields of embedded systems and digital image processing, the development of such a laboratory is of great interest. The limitations on both computational speed and storage space of embedded systems prescribe for a different approach on digital image processing algorithms, much more so if real-time processing is necessary. This renders existing Remote Laboratories which concern exclusively embedded systems [3] or digital image processing, insufficient for this purpose.

The proposed Remote Laboratory, named **R**emote **D**igital **Im**age **Pr**ocessing **Lab**oratory (R-DImPr Lab), is accessed by the students through internet via a common Web Browser (Internet Explorer, Mozilla Firefox) and grants control of the laboratory equipment using a user-friendly Graphical User Interface (GUI). Through this GUI, there is the capability of carrying out laboratory exercises such as image filtering, edge detection, histogram equalization etc. During the conduction of these exercises students are asked to develop their solutions using Code Composer Studio (CCS) and upload the executable codes to the R-DImPr Lab. As an additional assistance to them with the implementation of their own real-time image processing solutions and the communication with the Workstation application, a high-level Application Programming Interface (API) was developed. The students are also able to observe both the images that were captured by the CMOS Image Sensor based embedded system and the results of the processing algorithms through the GUI.

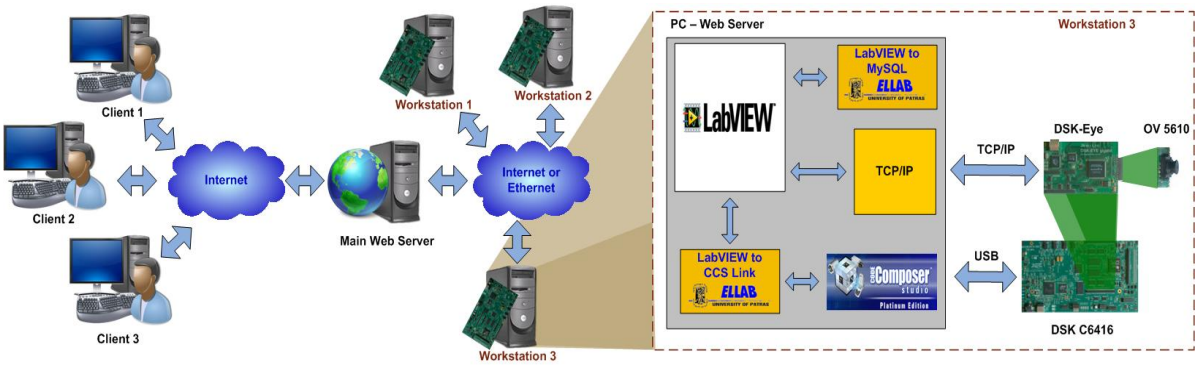

Figure 1. R-DImPr Lab's Architecture.

II. ARCHITECTURE

The R-DImPr Lab is based on the flexible and upgradable architecture of the R-DSP Lab [4]. The structural elements of R-DImPr Lab architecture (Fig.1) are the Main Web Server and the Workstations. The Main Web Server can be connected to one or more Workstations via Ethernet or Internet. The number of Workstations that can be used depends on the occasional needs and may be dynamically increased or decreased without influencing the operation of the R-DImPr Lab.

A. Main Web Server

The Main Web Server undertakes the reception and redirection of users to the appropriate Workstation. The user that wishes to carry out a predefined laboratory exercise or to verify the functionality of his own executable code, should follow the Login process. During this process, the user is asked to give his Username and Password. Those users, who have successfully completed the identification procedure, by accessing the Experiment Selection Web Page, are able to select one of the predefined laboratory exercises or to upload their executable files. The main Web Server redirects the authorized user to an available Workstation.

The functions of the Main Web Server were developed using the features of PHP and HTML. For the hosting of Web Pages the Apache HTTP Server is used. Furthermore, a MySQL database is installed in the Main Web Server. Inside the database, the users' information, the executable files that have been uploaded and certain statistical information are stored.

B. Workstation

Each Workstation consists of a Windows based PC, a DSK C6416 development platform by Spectrum Digital based on fixed-point TMS320C6416 DSP by Texas Instruments (TI) and a DSK-Eye Gigabit daughter card by Bitec which is used for interfacing between an Omnivision OV5610 CMOS Image Sensor and the DSK [5-8]. This equipment allows the acquisition of an image frame with resolution up to 5 Mpixels from the sensor and its storage in Bayer format in the DSK C6416 SDRAM for further processing. Moreover, data can be transferred to a PC and vice versa through a TCP/IP connection using the DSK-Eye gigabit Ethernet interface adaptor.

For the control of each Workstation and the communication between the user and the available laboratory equipment, an application using LabVIEW v8.0 has been developed. With this application, the complete control of the CCS v3.1 and DSK C6416 is achieved, using the features of the toolkit LabVIEW to CCS Link [9]. The captured by the CMOS Image Sensor frame in the predefined resolution, as well as the processed one, are transferred to the application's GUI over a TCP/IP connection. This application also grants the user with real-time control over the values of the CMOS Image Sensor control registers through the program running on the DSP. Moreover, it gives the user the capability to control the Workstation through the environment of a suitably designed Web Page which is hosted by an embedded LabVIEW Web Server. Additional assistance to the students with building the executable codes of their solutions, was given by the development of a high level API.

III. APPLICATION PROGRAMMING INTERFACE (API)

The presented, open source R-DImPr API [10] consists of both system control and image processing C functions. This constitutes this Remote Laboratory useful to a variety of courses that are focused in either the design of image processing embedded systems or the implementation of image processing algorithms. For convenience purposes the R-DImPr API is divided into two sections, the System Design API and the Image Processing API.

A. System Design API

The System Design section is a high-level API based on the DSK-Eye Gigabit API [8, 10], which enables designing and testing user-made image processing systems based on the laboratory equipment. It includes functions which initialize the system, set the system's IP, port and Gateway, control the values of the CMOS Image Sensor's registers, undertake the TCP/IP image transfer etc. A brief description of the System Design API's high-level functions is given below, as well as a sample code of one of the functions.

System_Init: Initializes the system in order to use the TCP/IP protocol for the communication between the system and the GUI. Additionally, it resets the CMOS Image Sensor parameters such as the resolution of the CMOS Image Sensor, the brightness of the captured image, the data transfer rate between the DSK-Eye Gigabit daughter card and the DSK C6416.

Set_Socket: Creates a TCP socket for the connection between the system - server and the GUI - client. To accomplish this, at first it sets the IP address and port of the server and the Gateway which are given as arguments. Then it creates the TCP socket using these parameters and returns the number of the TCP socket.

Wait_for_Command: Sets the system - server to the listen mode and waits to receive a command word from the GUI - client. It takes as an argument the number of the TCP socket and returns the command word.

Get_Frame: Is the main interface for frame grabbing. This function returns a pointer to the memory region containing the Bayer raw image frame and suspends the calling task until a new frame is available.

Send_RGB_Image: Undertakes the transfer of an RGB image frame through the TCP socket to the GUI. This function (Fig.2) takes as an argument a pointer to the image to be transferred. Depending on the value of the pointer, either the original or the processed image is sent.

Send_GrayScale_Image: Transfers a single component of the processed image frame through the TCP socket to the GUI. If this single component is the luminance component, then a grayscale image is sent. This function is particularly useful when the result of an image processing algorithm is a grayscale or a black and white image (e.g. edge detection).

Register_Write: Changes any parameter of the CMOS Image Sensor. This is accomplished by assigning a new value to the appropriate control register whose internal address and the new value are given as arguments.

Register_Read: Reads the current value of any CMOS Image Sensor control register. The register internal address is given as an argument.

```
void Send_RGB_Image( int cs, unsigned char *rgb,
                     int offset)
{
  int y, Buff_ptr;
  Buff_ptr = 0;

  for (y=0; y<RGB_SIZE; y++)
  {
    if(Buff_ptr == BUFF_SIZE)
    {
      if(send(cs, (unsigned char*) Buffer,
         BUFF_SIZE,0 ) != BUFF_SIZE)
         return;
      Buff_ptr = 0;
  }
  Buffer[Buff_ptr++] = *(rgb+y+offset);
  Buffer[Buff_ptr++] = *(rgb+y+offset + RGB_SIZE);
  Buffer[Buff_ptr++] = *(rgb+y+offset + RGB_SIZE*2);
  }
  if(send(cs, (unsigned char*)Buffer, BUFF_SIZE,0)
     != BUFF_SIZE)
     return;
}
```

Figure 2. The Send_RGB_Image function.

B. *Image Processing API*

The Image Processing section is a high-level API [10] designed from scratch, which contains functions implementing image processing algorithms such as color space conversions, filtering, edge detection etc, specially designed for the fixed point arithmetic DSPs. A brief description of the Image Processing API's high-level functions is given below, as well as a sample code of one of the functions.

Bayer_to_RGB: Converts a Bayer raw image stored in the memory region pointed by a pointer which is given as an argument, into the RGB color space. It returns a pointer to the memory region containing the converted image.

YCbCr_to_RGB: Converts a YCbCr image stored in the memory region pointed by a pointer which is given as an argument, into the RGB color space. This function replaces the YCbCr image with the result image.

RGB_to_YCbCr: Converts an RGB image stored in the memory region pointed by a pointer which is given as an argument, into the YCbCr color space. This function (Fig.3) stores the result right after the end of the RGB image.

3x3_Filter: Applies a 3x3 mask to the luminance component of the image. This function takes as arguments a pointer which indicates the memory region containing the luminance component and a 3x3 filtering mask array. Then it replaces the luminance component with the filtering result.

```
void RGB_to_YCbCr(unsigned char *rgb)
 {
   int i,offset = RGB_SIZE*3 ;
   int b,g,r;

   for (i=offset; i<offset+RGB_SIZE; i++)
   {
     b = *(rgb+i - offset);
     g = *(rgb+i - offset +RGB_SIZE);
     r = *(rgb+i - offset +RGB_SIZE*2);
     *(rgb+i) = (b*3736 + g*19235 + r*9798)>>15;
     *(rgb+i+RGB_SIZE) = ((b*16384 - g*10856
                           -r*5528)>>15) + 128;
     *(rgb+i+RGB_SIZE*2) = ((- b*2664 - g*13720
                            +r*16384)>>15) + 128;
   }
 }
```

Figure 3. The RGB_to_YCbCr function.

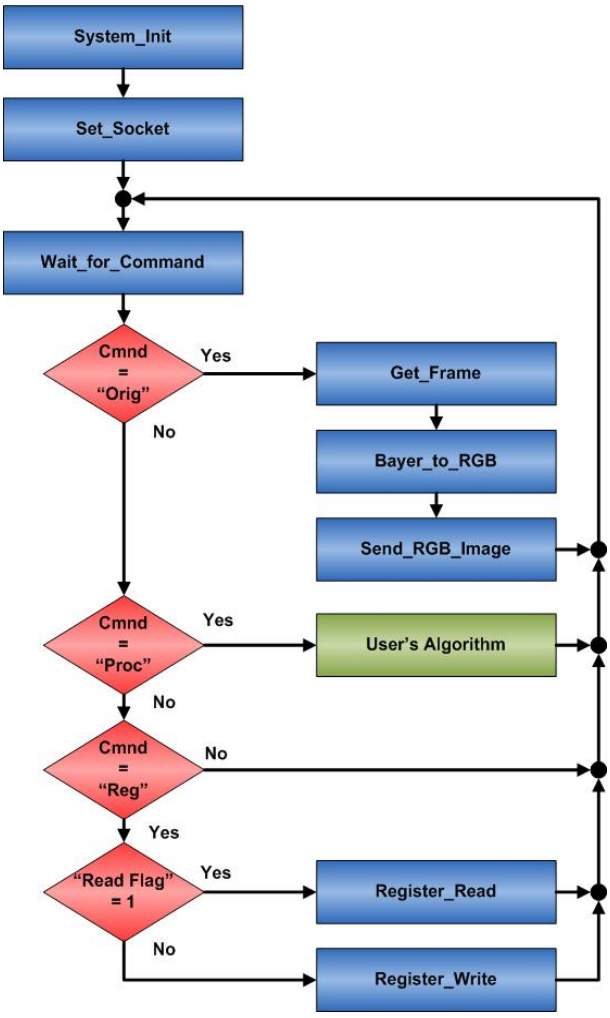

Figure 4. Block diagram of code architecture.

Sobel: Performs a sobel edge detection algorithm. It takes as an argument a pointer which indicates the memory region containing the luminance component. Then it replaces the luminance component with the algorithm result.

C. *Using theR-DImPr API*

The main purpose of the R-DImPr API is to provide students with handy tools for developing real-time image processing embedded systems. The proposed code architecture using the R-DImPr API is presented in the block diagram shown in Fig.4.

According to the proposed code architecture the user-made program at first calls the System_Init function initializing the system parameters. Then the Set_Socket function creates a TCP Socket for the communication with the Workstation's Application GUI. By using the Wait_for_Command function the program waits until a command word (Cmnd) is sent through the GUI from the user. Depending on the Cmnd, the program can perform a variety of tasks. If the value of Cmnd is "Orig" the system captures a frame from the CMOS Image Sensor in Bayer format, converts it into RGB color space and sends it to the appropriate GUI's picture indicator. In case the value of Cmnd is "Proc" the system executes the user's image processing algorithm. Otherwise, if the value of Cmnd is "Reg" the system calls either the Read_Register or the

Write_Register function depending on the value of the Read_Flag. After the end of each task the program calls the Wait_for_Command waiting for a new Cmnd.

IV. LABORATORY EXPERIMENT

In order to demonstrate the functionality of the R-DImPr Lab, an executable code that implements a Sobel edge detection algorithm using the laboratory's API was built on CCS according to the code architecture shown in Fig.4. For this purpose, the functions RGB_to_YCbCr, Sobel and Send_GrayScale_Image were added to the "User's Algorithm" section. It is worth to be mentioned that with the use of the API and the proposed code architecture, the development of complicated image processing applications on DSPs is greatly simplified.

After following the R-DImPr Lab's identification procedure, the executable code was uploaded to an available Workstation and the Workstation Web Page (Fig.5) was used to monitor the algorithm result. Through this Web Page the user, by pressing the "Get Original Image" button, can obtain the original image in predefined resolution which was captured by the CMOS Image Sensor and was converted on DSP from Bayer format to the RGB color space. The "Get Processed Image" button activates the "User's Algorithm" code section and that way the result of the real-time Sobel edge detector was displayed on the appropriate picture indicator. The original image and the algorithm result when the CMOS Image Sensor automatically handles image features such as exposure time, color channel gains and white balance correction are presented in Fig.5a. This level of use is sufficient for courses, whose subject lies mostly on the image processing part.

However, for courses that focus more on the DSP system design, access to the CMOS Image Sensor Control Registers at bit level is provided. The user can monitor and remotely control the values of parameters such as color channel gains, exposure time etc. A brief description of each register function [7] for online assistance is also available. Through the "CMOS Sensor Control Register" menu the user is able to select the desirable register and by pressing the buttons "Read" and "Write", can view and change its content at bit level. The "Register Description" field provides information about each register's usage.

Simple examples of remotely adjusting the system parameters are shown in Fig.5b and Fig.5c. By changing the value of a certain control register (COMI), the CMOS Image Sensor's automatic mode was deactivated. Then the gain register (GAIN) value was increased giving improved algorithm results (Fig 5b). Further gain and exposure time increase enhanced the processed image even more (Fig 5c).

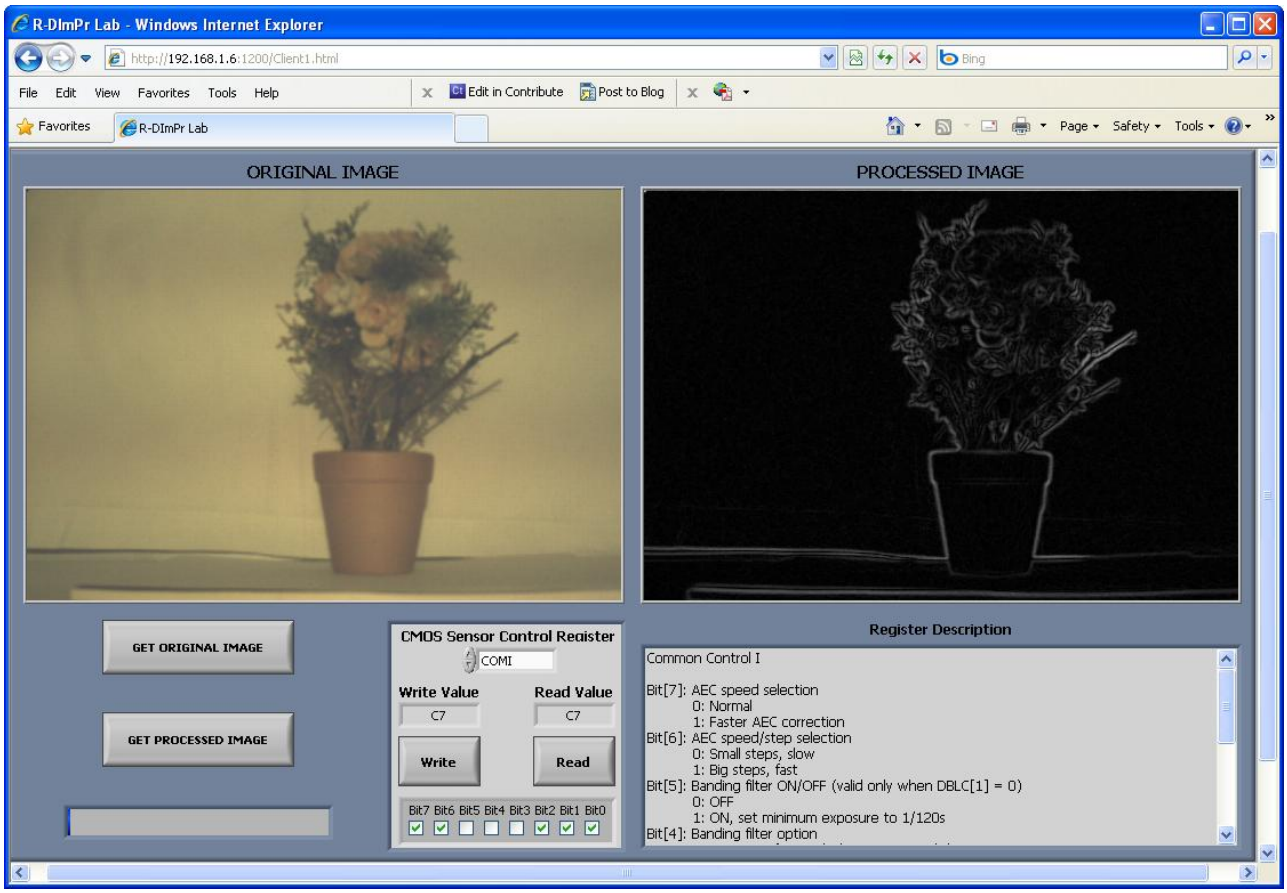

Figure 5a. Workstation's Web Page – Automatic image sensor register adjustment.

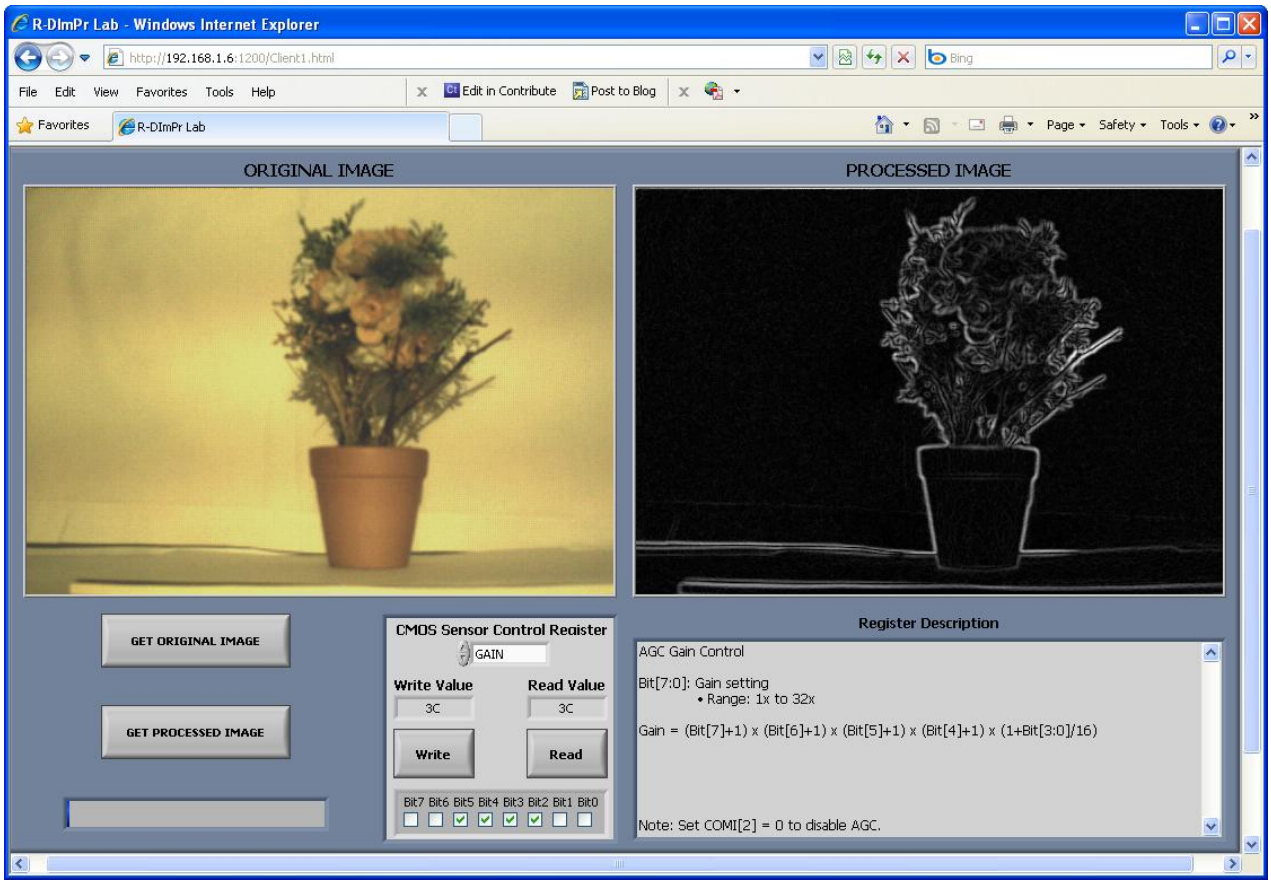

Figure 5b. Workstation's Web Page – Remotely increased image sensor gain.

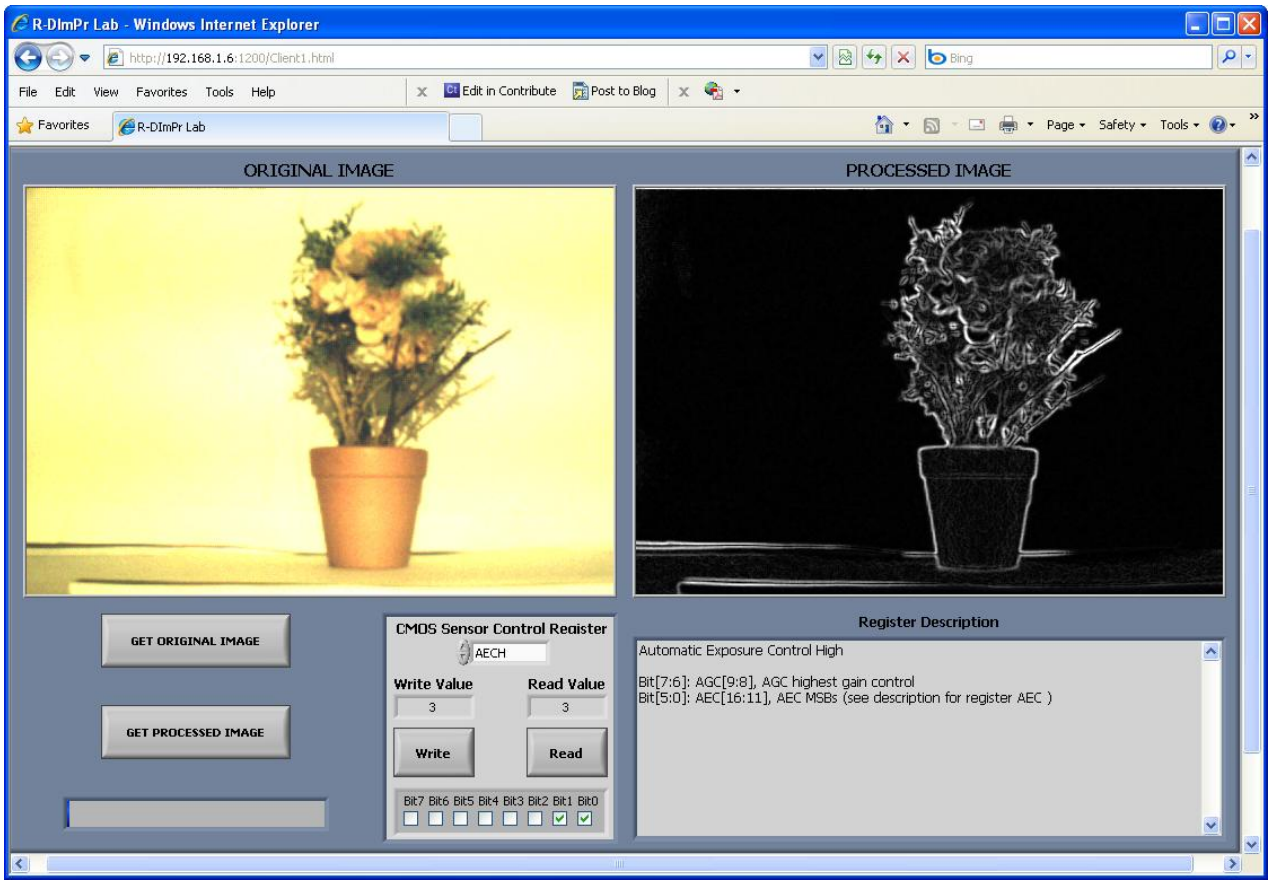

Figure 5c. Workstation's Web Page – Remotely increased image sensor gain and exposure.

V. CONCLUSIONS

In this paper the architecture, development and operation of a Remote Laboratory for real-time digital image processing on embedded systems based on DSPs was presented. This laboratory has an upgradable and flexible architecture, is accessed through a common Web browser and it is applicable to courses of different academic degrees rendering it a powerful tool for the educational process. In order to assist users with the development of their, compatible to the Workstation's GUI executable code, a high-level API was also introduced. The proposed code architecture for the use of the API greatly simplifies the implementation and verification of image processing applications, while it provides users with system design functions, too. Such an example of a real-time image processing system using the API was given to demonstrate the functionality of the laboratory. Future plans include the expansion of the API in order to give to the users more programming options, both in application complexity and variety. Moreover, the design of a toolkit for the communication between the workstation and GUIs created by users will enhance the operation of R-DImPr Lab and fully exploit the API's potential.

REFERENCES

[1] C. Gravier, J.Fayolle, B. Bayard, M.Ates & J.Lardon, "State of Art About Remote Laboratories Paradigms – Foundations of Mutations", *International Journal of Online Engineering (iJOE)*, vol.4, issue 1, pp. 19-25, February 2008.

[2] Dr. Gokhan Gercek & Dr. Naveed Saleem, "Transforming Traditional Labs into Virtual Computing Labs for Distance Education", *International Journal of Online Engineering (iJOE)*, vol.4, issue 1, pp. 46-51, February 2008.

[3] D. Hercog, B. Gergic, S. Uran and K. Jezernik, "A DSP-Based Remote Control Laboratory", *IEEE Transactions On Industrial Electronics*, vol. 54, No. 6, pp. 3057-3068, December 2007. (doi:10.1109/TIE.2007.907009)

[4] A. Kalantzopoulos, D. Karageorgopoulos & E. Zigouris, "A LabVIEW Based Remote DSP Laboratory", *International Journal of Online Engineering (iJOE)*, vol. 4, special issue 1: REV 2008, pp. 36-44, July 2008.

[5] Spectrum Digital, *TMS320C6416 DSK*, Technical Reference, Rev.B, November 2003.

[6] Texas Instruments, *TMS320C6414,TMS320C6415, TMS320C6416 Fixed-Point Digital Signal Processors*, SPRS 146, Rev. N, May 2005.

[7] Omnivision, *OV5610 Color CMOS QSXGA (5.17 MPixels) Camera Chip with Omnipixel Technology*, Datasheet Version 1.6, March 2005.

[8] Bitec, *DSKeye gigabit*, Users Manual, Version 1.0, 2007.

[9] E. Zigouris, A. Kalantzopoulos & E. Vassalos, "LabVIEW to CCS Link for Automating Digital Signal & Image Processing Applications", *8th International Symposium on Signals, Circuits & Systems, ISSCS 2007*, pp. 445-448, Iasi, Romania, 12-13 July 2007.

[10] D. Markonis and E. Zigouris, *Design and Implementation of an API for Digital Image Processing Systems Based on DSPs*, Internal Report, Electronics Lab, Electronics and Computers Div., Physics Dept., Patras University, 2009.

AUTHORS

A. Kalantzopoulos is with the Electronics Laboratory, Electronics and Computers Div., Department of Physics, University of Patras, Rio Patras, GR-26500 (e-mail: kalan@upatras.gr).

D. Markonis is with the Electronics Laboratory, Electronics and Computers Div., Department of Physics, University of Patras, Rio Patras, GR-26500 (e-mail: dnmarkon@upatras.gr).

E. Zigouris is with the Electronics Laboratory, Electronics and Computers Div., Department of Physics, University of Patras, Rio Patras, GR-26500 (e-mail: ez@physics.upatras.gr).

Submitted, January, 17, 2009. Published as resubmitted by the authors on May, 16, 2009.