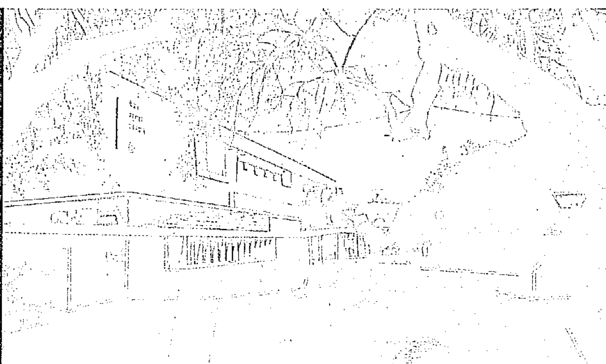
PRL-TN-2011-99

Technical Note



Real-time automated software for the operation of multiple CCD cameras and for simultaneously interfacing with the balloon-borne telemetry system

by

Parshv Shah and Duggirala Pallamraju

(Space and Atmospheric Sciences Division)

भौतिक अनुसंधान प्रयोगशाला, अहमदाबाद

Physical Research Laboratory, Ahmedabad

Contents

Real-time automated software for operation of multiple CCD cameras and for simultaneously interfacing with the balloon-borne telemetry system

Parshv Shah and Duggirala Pallamraju
(Physical Research Laboratory, Ahmedabad)
(raju@prl.res.in)

Abstract

In this report we present the technical details of the interface software which has been successfully implemented during a balloon-borne experiment for the investigations of wave dynamics in the atmosphere. This experiment was conducted from the National Balloon Facility in Hyderabad. The challenge in this work was to carryout multitasking in real-time. The tasks involved operations of two charge coupled device (CCD) cameras simultaneously, interfacing with the telemetry and telecommand systems through a digital IO board, storing of the data in the computer and sending the large data volume in a specified manner to the ground system through telemetry. This task was achieved by using four processes and interlinking through shared memory. This software is platform dependent and had been designed to operate on the Linux operating system.

Introduction

There are several scientific experiments that can not be carried out from the ground. These relate to various scientific investigations that involve observations in the X-ray, extreme ultra violet (EUV), or ultra violet (UV) wavelength regions. As it is known, photons from these wavelengths are absorbed in the upper layers of the earth's atmosphere. Therefore, for scientific investigations in these wavelength regions, one needs to use different platforms such as balloons, rockets, or satellites, depending on the scientific requirements. In the present experiment, the aim was to observe UV emissions over low geomagnetic latitudes from the atmosphere for a large duration (several hours). Since the UV radiation does not reach the ground, a balloon based platform was chosen.

For all the observational platforms mentioned above, it is extremely essential that all the technical operations such as, functioning of the detectors, data acquisition, telemetry, telecommand, turning equipments OFF for safe mode, and turning them ON when required, perform reliably and be precisely controlled by a dependable software. Further, as the equipment is located far away in an extreme environment, the hardware should be capable of withstanding harsh environmental conditions and the software should also be capable of adequately monitoring these conditions.

In our experiment, it was required that the software interface with two identical CCDs, and carryout data acquisition with different independent operations such as: changing of integration times, storing of images on the computer, sending data to ground through telemetry, and receiving telecommands. This required efficient programming with optimal usage of resources, and proper memory management so that the computer does not hang up, or crash during the actual field operations. The block diagram of the experiment is shown in figure 1. One can note that the two optical instruments (along with the data acquisition system) interface with the balloon-borne hardware. The signal that is transmitted through the telemetry at the balloon-end is received by the ground receiver and the decoded data is provided to the user. The present work discusses the development of electrical and electronic interface with the hardware at the balloon-end and on the software for accepting the data on the ground in the format followed at the balloon facility.

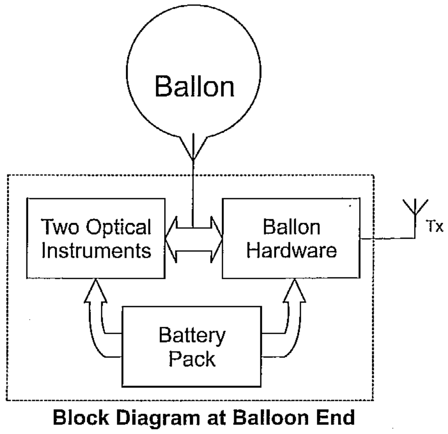


Fig. 1 Block diagram of the experiment

Description of the experiment

For this experiment, a software was developed which could execute multiple tasks in real time to cater to the scientific requirement. Further, the software had to be optimized to perform efficiently within the specifications laid down by the National Balloon Facility. Sections below describe the following requirements.

1. Scientific requirements
2. Hardware requirements at instrument end
3. Telemetry/Telecommand interface
4. Software requirements

1 Scientific requirements

For accomplishing the science objectives, it was required that the images obtained by the two CCD cameras that are connected to the two spectrographs (one measuring the UV and the other measuring the visible wavelength emissions) operate optimally in a programmed mode. Specifications of the CCD detectors used are given in table-1.

Specifications of the CCD camera	
Camera cooling	55 to 70°C (ΔT), fan-assisted air or water
Standard Camera Weight	5.5 pounds
Operating range	95% relative humidity
Operation temperature range	-40°C to 50°C
Power Requirements	12V, 72W
Standard Camera Dimensions	6.2″ x 6.2″ x 4″ (W/D/H)

Table 1: Specifications of the Finger Lakes Instrumentation (FLI) CCD detector

1.1 Flexibility of exposure time

As both the CCDs have different optical components with different transmission factors, they are expected to have different exposure (integration) times. As the exposure times are different, the CCD operations (shutter open/close) will have different phase relationships with one another. The software should be able to keep track of these differences not only for proper memory management but also for sending the images through telemetry to the ground (discussed later in this section).

1.2 Optimization of CCD temperature

As the dark noise of CCD increases with the CCD temperature, it is required that the CCD temperature is kept at the minimum. However, the CCD cooler usually employs 2 to 3-stage Peltier element, which draws large current to cool the CCD chip. This puts a significant load on the battery power, when in flight, in the actual experiment. Therefore, it is required that there should be a provision to control and change the CCD temperature to optimize the CCD performance, and at the same time conserve the battery power. This feature is required to be available as an option in case it becomes necessary to increase or turn off the CCD temperature, in the interest of battery life.

1.3 Modification of CCD binning

Due to the constraints on the balloon telemetry bandwidth, it is required that the size of the image be as small as possible which reduces the time of transfer of images through telemetry. However, excess binning compromises the spectral resolution of the image that is being obtained. Therefore, there needs to be a provision to change the binning of the CCD chip on a real-time basis as and when the situation demands, to enable efficient transfer of images through telemetry.

2 Hardware requirements at instrument end

Each optical instrument consists of a CCD as the detector. As mentioned earlier, there are two instruments in our experiment. Operations of the CCD, as described above, have to be controlled using a reliable computer. Further, the data transfer to the balloon-borne telemetry encoder and receiving of telecommands need to be accomplished by this computer through a compatible digital IO (DIO) board. Both these components (the single board computer and the DIO board) should be capable of functioning at low temperatures as the balloon will be positioned at stratospheric heights (~35km) with low temperature (~ -20°C) and low pressure (6-7mbar) for a large duration (~6 hours).

2.1 Single board computer

We used winsystems single board computer for this experiment (figure 2). It has the following features:

- Intel ® Celeron 1 GHz processor
- 512 kb CPU cache size and 512 MB memory
- Industry standard phoenix BIOS
- 4 USB2.0 ports, X86 compatible interrupt and DMA controllers
- AT Key board controller and PS2 mouse
- 10/100 Mbps Ethernet controller
- 2 parallel ports

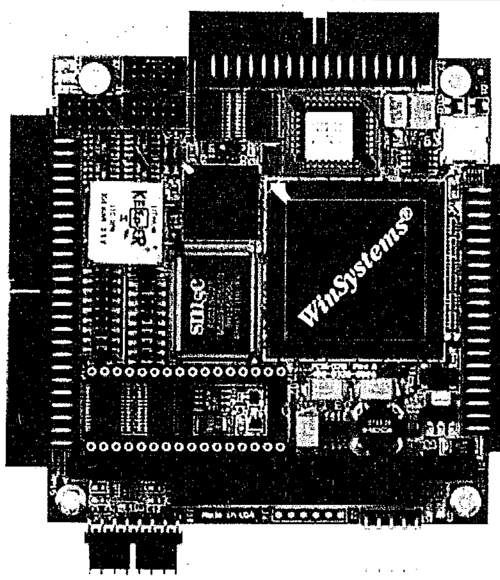


Fig. 2 : Image of the single board computer used in this experiment

- 48 bidirectional digital input/output
- Power : +12V required, 1A typical
- Industrial grade operating temperature : -40° C to +85° C
- Form Factor : PC/104-compliant and 3.60" X 3.80" (90mm x 96mm) size

2.2 Digital input/output (DIO) Board

The PCM-UIO96B is a highly versatile PC/104 input/output module (shown if figure 3). One important feature of this card is its ability to monitor 48 of the 96 lines for both rising and falling digital edge transitions, latch them and then interrupt the host processor notifying that a change-of-input status has occurred. This is an efficient way of signaling the CPU of real-time events without the burden of polling the digital I/O points. Specifications of this DIO board are shown in table 2.

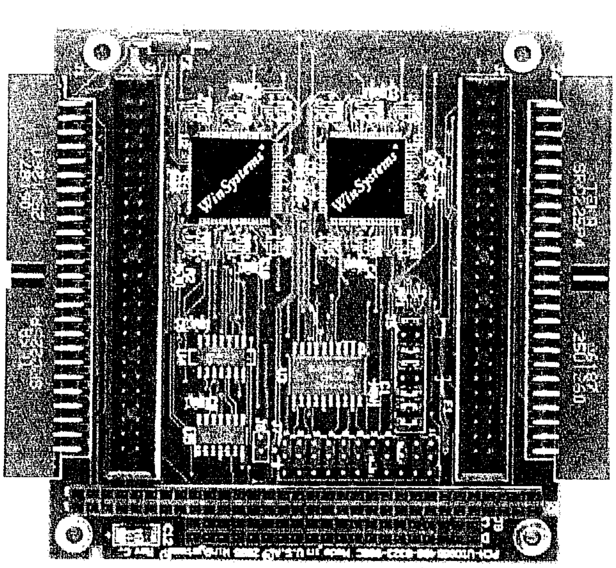


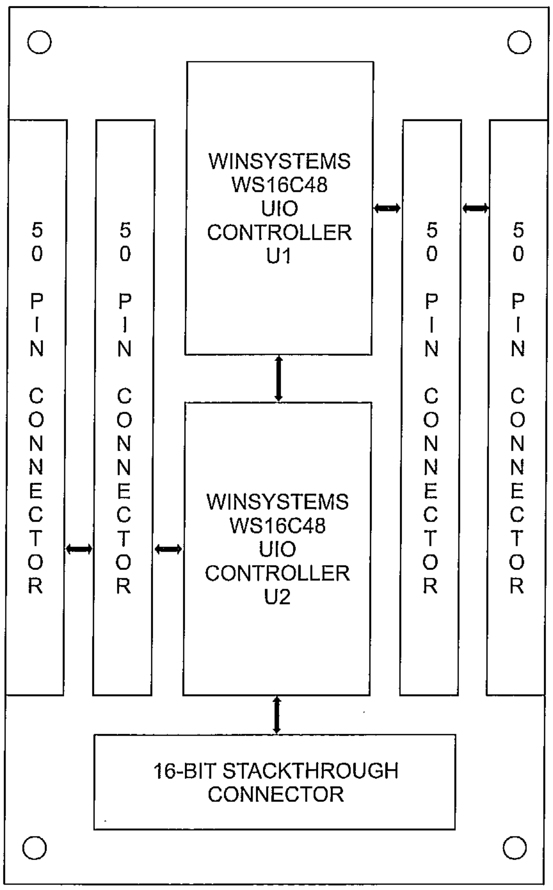
Fig. 3: Image of the Digital IO Board used in our experiment.

This card has the following features:

- Supports 96 digital I/O lines
- Each line is capable of bidirectional operation, input/output or output with read back and 2mA sink current.
- Generates an interrupt on signal change of state. It supports 48 event sense lines. It has software selectable edge polarity for each line and software enabled interrupt for each line. It has change-of-state latched for each line.
- Write-protection mask register for each 8-bit port
- Compatible with industry standard I/O racks
- Fused +5V logic supply for I/O modules
- 16-bit PC/104 interface
- +5 volt only operation
- Extended temperature range: -40°C to +85°C
- Replaces two PCM-UIO48A modules

Specifications of the DIO board	
Electrical	PC/104 Bus: 16-bit stack through Parallel Interface: 96 I/O lines, TTL compatible
Power Requirements	Vcc = +5V ±5% at 24mA (excluding rack power with no loads on the outputs)
Mechanical	Dimensions: 3.6" x 3.8" (90mm x 96mm)
Connectors	Digital I/O: J1 and J4, 50-pin right angle on 0.100" grid J2 and J3, 50-pin dual 0.100" grid Jumpers: 0.025" square posts
Environmental	Operating Temperature: -40°C to +85°C Non-condensing relative humidity: 5% to 95%

Table 2: Specifications of the DIO board.



PCM-I096B BLOCK DIAGRAM

J1 and J4

Pin	Description
1	P2-7
3	P2-6
5	P2-5
7	P2-4
9	P2-3
11	P2-2
13	P2-1
15	P2-0
17	P1-7
19	P1-6
21	P1-5
23	P1-4
25	P1-3
27	P1-2
29	P1-1
31	P1-0
33	P0-7
35	P0-6
37	P0-5
39	P0-4
41	P0-3
43	P0-2
45	P0-1
47	P0-0
49	+5V
Even	Ground

J2 and J3

Pin	Description
1	P5-7
3	P5-6
5	P5-5
7	P5-4
9	P5-3
11	P5-2
13	P5-1
15	P5-0
17	P4-7
19	P4-6
21	P4-5
23	P4-4
25	P4-3
27	P4-2
29	P4-1
31	P4-0
33	P3-7
35	P3-6
37	P3-5
39	P3-4
41	P3-3
43	P3-2
45	P3-1
47	P3-0
49	+5V
Even	Ground

Rack I/O Connector Pin-Out

Fig. 4: The block diagram of PCM-UIO96B and the I/O connector Pin-outs

This DIO board contains the following functional capabilities:

- PC-104 interface

The PCM-UIO96B is a PC/104 compatible stack through card. Unlike the popular ATX form factor which utilizes the PCI bus and is currently used for most PCs, the PC/104 form factor has no backplane, and instead allows modules to stack together like building blocks. The stacking of buses is more rugged than typical bus connections in PCs. This is a result of mounting-holes in the corner of each module, which allow the boards to be fastened to each other with standoffs. This interface makes this card more flexible and rugged so that the system can be prevented from the shocks.

- Parallel I/O Controller

This card contains WS16C48 Universal I/O controller ASIC to support various input/output and interrupt configurations. A WS16C48 supports 48 digital I/O lines addressed through 6 contiguous registers as ports P0-P5. Two WS16C48 chips are on the PCM-UIO96B (as shown in Figure 4). Each I/O line is individually programmable for input, output, or output with read back operation.

- Event Sense Operation

Each WS16C48 ASIC supports event sense lines which can generate an interrupt when an event occurs. PCM-UIO96B can sense a positive or negative transition on up to 48 lines. Transition polarity is programmable and enabled on a bit-by-bit basis. Each line's transition is latched by the event so that even short duration pulses will be recognized.

- Interrupts

The PCM-UIO96B can generate a system interrupt request which can be routed via a jumper block to IRQ channels 2 - 7, 10 - 12, 14, and 15. Both WS16C48s can generate an individual interrupt, however, the interrupt requests from both chips can also be OR'ed together.

- I/O Connectors

The signals from each WS16C48 are wired to two 50-pin connectors. There are 24 lines capable of event sense operation in each connector.

We have to set proper IO address (to communicate with the Digital IO board) and proper IRQ address (for handling Interrupt Request).

2.3 *Telemetry system*

Telemetry is a system which is used to transfer data from a remote place to the ground. Telemetry systems are used with all the remote platforms such as in satellites, rockets, balloons, to meet the communication requirements. Initially telemetry systems used to be pulse-position modulation (PPM) type which has now been replaced with pulse-code modulation (PCM). The telemetry system consists of two parts: 1) on-board (on balloon gondola), and 2) on the ground.

- On board telemetry system

A telemetry encoder gathers information from various subsystems of the payload, which can be the data from the detector electronics, data from orientation circuit, housekeeping data like temperature, pressure, position (latitude, longitude, altitude), etc. The block diagram of telemetry system is given in Figure 5.

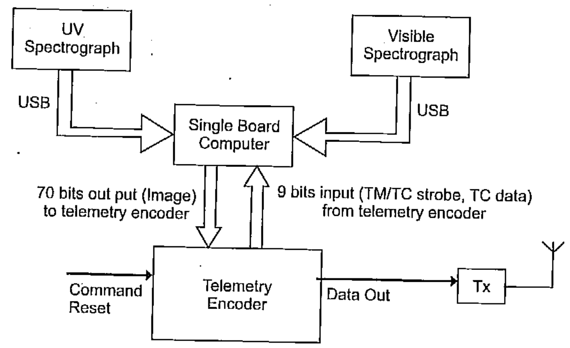


Fig. 5: Block diagram of balloon telemetry system

Information gathered is encoded as per the required telemetry format. Initially the data is sampled, then quantized, and then modulated using the PCM and finally converted into analog signal. The signal is transmitted serially over the transmitter bit-by-bit.

- Ground telemetry system

At the ground side modulated signal is received which is then demodulated, and decoded by the demodulator. Then the raw data is given to either the PC or other systems. Its block diagram is shown in Figure 6.

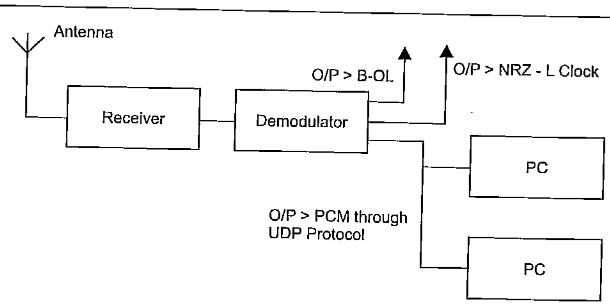


Fig. 6: Block diagram of ground telemetry system

Specifications of telemetry encoder	
Bit Rate	125 K bits/sec
o/p Voltage Level	0 to 5 Volts
Bits/Word	8 bits (7 Data + 1 Parity)
Words/Frame	32
Frames/Sub Frames	39
Data Alignment	MSB First
Power Input	Unregulated +16 to +18.5 V – 10mA -12 to -15 V – 25 mA
Bit time	8 microseconds
Word Time	64 microseconds
Frame Time	2.05 milliseconds
Sub Frame Time	78 milliseconds

Table 3: Specifications of telemetry encoder used in our experiment and as provided by the National Balloon Facility

Features of telemetry encoder were:

1. It is rated to work up to the transmission rate of 125 Kbps.
2. It has an interface port in the form of a 37-pin D-type connector to communicate with other subsystems of the payload.
3. Different housekeeping signals, both analog and digital signals, are needed to monitor the health of various subsystems on the payload from time to time. This system is capable of handling thirty two, 8-bit digital words.
4. The serially transmitted data is also simultaneously transmitted in an 8-bit parallel format with the strobe pulse. This feature enables direct interfacing of the telemetry encoder with the PC for troubleshooting and for debugging, which otherwise is not possible without a telemetry decoder.

In this experiment, we had been provided with 10 words in a single frame. (Some of the telemetry specifications are listed in Table 3). Each word contains 7 data bits and 1 parity bit. We can send 70 bits/frame. Each frame is sent at an interval of 2.05 milliseconds, and so the bit rate of telemetry is 125 K bits/second. Telemetry transmission and reception was based on S-band modulating frequency at 2.259 GHz. At the ground side, the raw data is broadcasted (within a network) through User Datagram Protocol (UDP) by ground telemetry system.

2.4 *Telecommand handling*

The commands used to control the on-board telemetry system are called telecommands. A telecommand is one which is sent to control a remote system or systems not directly connected via wires to the place from which the telecommand is given. For a Telecommand (TC) to be effective, it must be compiled into a pre-arranged format (which may follow a standard structure), modulated onto a carrier wave which is then transmitted with adequate power to the remote system. The remote system then demodulates the digital signal from the carrier wave, decodes the TC, and gives it to the appropriate hardware on the balloon gondola for necessary action. Transmission of the carrier waves can be by ultrasound, infra-red, or by other electromagnetic means. The software should be capable of receiving the telecommands sent from the ground from time-to-time and then execute the command which can be changing any of the modes of camera operations, such as: ON/OFF the telemetry systems or sub-systems, configure other balloon circuits, control balloon payload, balloon cut-off, ballast drop (to push the balloon to higher altitude), azimuth and elevation rotation of gondola. Sometimes telecommands are used to prevent unexpected race conditions.

For our experiment, we had been provided with an 8-bit data command. So, a maximum of 256 telecommands are possible. We used these telecommads to configure the camera operations, such as exposure time, separation time, binning and temperature. We also used three of these commands to turn our single board computer and cameras to ON/OFF positions.

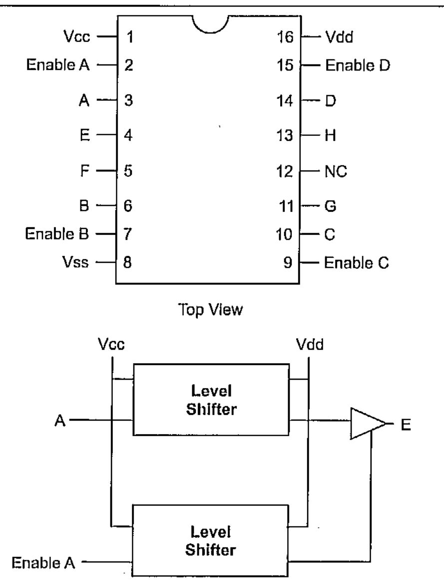
3 Telemetry/Telecommand interface

Proper communication should exist between the single board computer (along with the Digital IO board) and the balloon hardware. The DIO board operates on TTL logic, that is, it operates with a data or strobe of 5 Volts, however the balloon telemetry system operates on CMOS logic, thus, it requires a strobe of 9 Volts to register the signal. Therefore, as a part of ground interface it is required to convert 5 Volts TTL to 9 Volts CMOS when sending the data through the DIO board to the balloon telemetry and from 9 Volts to 5 Volts for receiving the telecommands sent to the balloon system from the ground. We used a voltage level shifter (CD40109B IC) which shifts the voltage level from 5 to 9 Volts and vice-versa. This board is placed as an intermediate circuitry between the DIO board and the balloon system hardware.

CD40109B IC was used in the interface as it has a good success record from the previous balloon flights. We needed level shifting for 80 input/output lines (70 bits for telemetry data, 1 bit for telemetry strobe, 8 bits for telecommand data and 1 bit for telecommand strobe). It contains four high-to-low or low-to-high level shifting circuits depending on V_{CC} and V_{DD}. If $V_{CC} < V_{DD}$ then it operates as low-to-high level shifter. If $V_{CC} > V_{DD}$ then it operates as high-to-low voltage shifter (as shown in table 4). We had used 21 ICs (20 + 1 extra) in a voltage level shifter. The IC pin diagram and the functional diagram are shown in Figure 7.

Inputs	Oputputs	
A, B, C,D	Enable A, B, C, D	E, F, G, H
0	1	0
1	1	1
X	0	Z

Table 4: Truth table for the performance of the functional diagram shown in Figure 7. Logic 0 = LOW (Vss), X = Don't Care, Z = High Impedence; Logic 1 = Vcc at inputs and V_{DD} at outputs.



Functional Diagram

Fig. 7: The IC pin and functional diagrams used for shifting of voltage levels to make our system compatible with that of the telemetry hardware.

4 Software requirements

For the present work, software is configured in two parts. One part of the software operates in the single board computer (SBC) on the balloon, which in addition to the several tasks mentioned above, separates the images into many parts to be sent to the ground system through telemetry. The other part of the software operates on the ground computer and is used to recreate the telemetry data back into images.

On the balloon-end, software was programmed to carry out the following operations:

a. *Configuration of the camera*

This part of software configures the CCD camera wherein various operations are carried out. They are changes in: exposure and separation times, temperature, on-chip binning of the image, size of the image, coordinates of the origin of the image and cooler ON/OFF.

b. *Image acquisition and data storage*

The software has been interfaced with the CCD hardware through the driver files of the camera. The camera then takes images as per the commands given by the software. Software takes these images and stores them in the single board computer.

c. *Transfer of data to telemetry*

The telemetry hardware provides a strobe pulse at every 2.05 ms. The software keeps 70 bits of data in the queue ready for the telemetry to transfer. On receiving the strobe pulse, these bits are sent to the balloon telemetry encoder through TM/TC interface.

d. *Telecommand reception and system configuration*

The telecommand is received at the time of strobe pulse. The software monitors the arrival of any telecommand from the peer system and reacts accordingly.

The software configured on a computer on the ground side receives raw data from the telemetry system and recreates the images serially that are sent from the balloon telemetry. Raw data contains image name, image size, and image data. Raw data are received in the form of UDP packets every second. Each UDP packet is of 1565 Bytes in size and contains 39 frames. Each frame consists of 32 words. The sketch of a UDP packet is shown in Figure 8.

Each frame contains experiment as well as housekeeping data. In each frame, out of the 32 words, ten words (21-30) are earmarked for the scientific data. Each word contains 7 data bits and 1 parity bit. So, one frame contains 70 bits (10 words x 7 data bits per word) data. The ground software application takes the whole UDP packet, divides it into several frames and extracts only the experiment data (10 words per frame) from each frame. It checks for the new file by checking "New_File" string in each frame (because at the balloon-end a new image is sent by putting a "New_File" string before it). If and when a new file arrives, the software extracts the file name and file size from the experiment data. Accordingly, it creates a file and based on the file size it writes image data in that file. Each time it extracts 70 bits from a single frame and groups them together. Then it merges all the groups into a single image, and thus, each image is recreated. On an average, for a single image recreation it takes about three minutes, which includes the time taken for telemetry. This application is written in Core Java (J2SE).

	0	21	30 31
0	H	E	
1	O	X	
2	U	P	
3	S	E	
..	E	R	
..		I	R
..	K	M	E
..	E	E	S
..	E	N	E
..	P	T	R
..	I		V
..	N		E
..	G		D
..			
..	D	D	
..	A	A	
37	T	T	
38	A	A	

Fig. 8: The UDP packet structure used at the ground to reconstruct the image.

Description of the code

The software was divided into four modules with each module indicating a single process.

i. Two processes were used for data acquisition from the 2 CCDs connected to the optical instruments and storing the data collected in to the single board computer.

ii. One process was used to work with the digital IO board. There were two threads in this single process. One thread was used to handle telemetry and other thread was used to handle the telecommands.

iii. One process was used to create (fork) the above two types of processes and monitor them.

Block diagram for these processes mentioned above is shown in Figure 9.

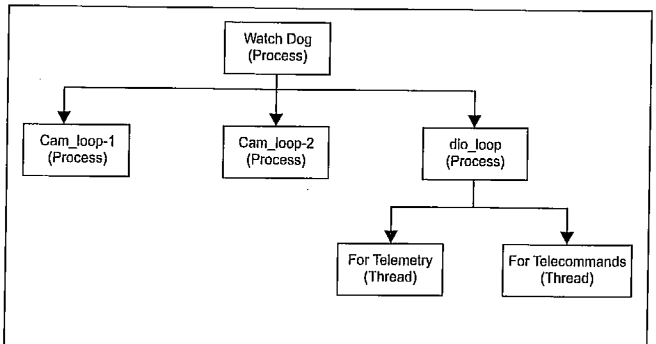


Fig. 9: Schematic of the block diagram for the system software.

In this block diagram, program starts from the watchdog process. The watchdog process creates 3 other processes. Cam_loop-1 and Cam_loop-2 processes operate two cameras and the dio_loop process operates the digital IO board. The dio_loop process creates two threads to handle the work simultaneously. They communicate with each other through shared memory. This software is divided into different files, which are only briefly described below. The actual codes used in the experiment are provided in Appendix-A.

camera.h

This is a header file which contains all important global defines, variables and data structures which will be used by all the processes.

watchdog.c

This is a C program file which represents the watchdog process. This process creates a watchdog loop and in that it creates the three processes as mentioned earlier: two processes for data acquisition from the two CCDs and one process for communicating with the digital IO board.

cam_loop.c

This file represents the cam_loop processes for the two CCDs. Therefore, this process needs to be forked twice from the watchdog process. This file checks whether any camera is connected to the single board computers through the USB. If any camera is connected to it, then it configures the CCD camera(s) using the FLI library files and drivers. It then carries out data acquisition and stores the data into the single board computer in a specific directory with that days' date as the name of the folder. If the file acquired happens to be the first one on that day (date), then a new directory with the name of that date is created and then the image file is stored in that folder. It also creates a catalog file for that day which contains various parameters related to the image, such as: filename, mode of operation, exposure time, binning, image matrix size, temperature, and power consumption of the CCD. Information from subsequent acquisitions are appended into this catalog file. After the data acquisition, it updates the shared memory by saving the last filename, which is used by telemetry as described in the next section.

dio_loop.c

This file represents the dio_loop process and is used to communicate with the DIO board. This file carries out the operations in two threads: one to handle the telemetry and the other thread to handle telecommands with the same digital IO board at the same time.

Telemetry thread opens the last saved file (filename updated by the cam_loop process as mentioned above) for both the CCD cameras alternately. It waits for an interrupt on the DIO board from the telemetry system. When it identifies a strobe on the DIO board it sends 70 bits of data to telemetry system through the DIO board. It then repeats the above step until it completes sending the whole image. Then it opens the last saved image of another camera.

Telecommand's thread awaits telecommand strobe on the DIO board. Once it receives a strobe it reads the telecommand data from the DIO board and replaces the old telecommand by this new one in the shared memory. This value is used by the cam_loop process to change the configuration of camera or the single board computer, which is implemented after the completion of data transfer that may be ongoing at that point in time.

Testing of the software

Testing is important to assess the quality of the software. Good practices throughout the development process contribute to the quality of the final product, but only testing can demonstrate that quality has been achieved and identify the problems and the risks that remain. Testing is needed to check that the software performs as expected, and its execution does not result in failures of any of the subroutines and or the computer as a whole (crashing of computer, issues related to memory overflow, etc.). Testing is essential to ensure that the software performs technically and functionally as designed, and to demonstrate that it solves the problem(s) that are intended to be solved by it. Further, testing the software also measures what risks one may be taking, as it is nearly always the case, no software is perfect.

In our case, we had followed many approaches to test our software to make it foolproof. Initially, we had divided the whole software into different modules, like cam_loop, dio_loop, and watchdog. As soon as any new module was developed, that unit was tested individually with the hardware. Then, various modules were integrated and tested to verify combined functionality after integration of each of the module. We had tested the software in all different functional scenarios like, changing camera configurations during the processes of telecommands, data acquisition, sending of data by receiving telemetry strobe, etc.

We carried out simulation tests using a signal generator to provide a strobe pulse at a frequency that the NBF would be using in the actual experiment. After the simulation for functionality in the laboratory at normal temperature, we placed the instruments (optical hardware, CCD, and the SBC) in climatic chamber and operated them continuously to test its performance in cold environment. Later, we moved the instruments into a vacuum chamber where in the temperature and pressure were set to -20°C and 7-10 mbar, respectively, that are usually expected at stratospheric heights. Balloon flight was planned for eight-hour duration. However, we had tested the system for a duration that is three times greater in both the chambers. We also tested the recovery of the system from crashes and hardware failures by abruptly turning the CCDs OFF and ON (individually and together) during the duration of data acquisition.

The software was subjected to various tools to specifically test for its efficiency in performance. For checking memory leakage(s) in the software we used mem-check sub-tool of Val-grind tool. Also, we used GDB (GNU debugger) to debug the program line by line and to read core dumps (These are files which indicate the line on which the execution of a program that stopped because of a run time error). Furthermore, we used a wireshark, which is a network analyzer tool, for the consistency in the functioning of our software. In our application at the ground side, we have to receive UDP packets from the telemetry in the actual experiment. In order to check that the code works for such an operation we set up two computers and simulated the data transfer between them through the UDP in a programmed manner.

Finally, an end-to-end test was carried out, starting from capturing images at the balloon-end to receiving raw data, and recreating images at ground, for around 24 hours of continuous operations at the NBF with all the systems in actual conditions (instrument hardware, software, telemetry system, telecommands, azimuth rotation of the balloon gondola, etc.). After the success of this test it was qualified for flight.

The balloon experiment was conducted on 8 March 2010 from Hyderabad. The software handled all the operations flawlessly and performed all the tasks efficiently in the actual experiment. Figure 10 shows a snapshot of the running application at the ground side. When ever any new image is received at ground, the application displays its name and then writes image data in it which is represented by dots, as can be seen in this figure.

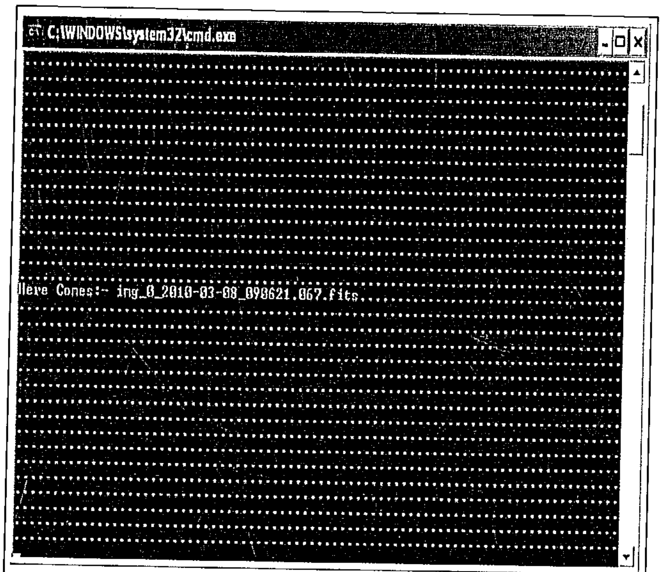


Fig. 10: Snap shot of the running application at ground side while retrieval is in progress.

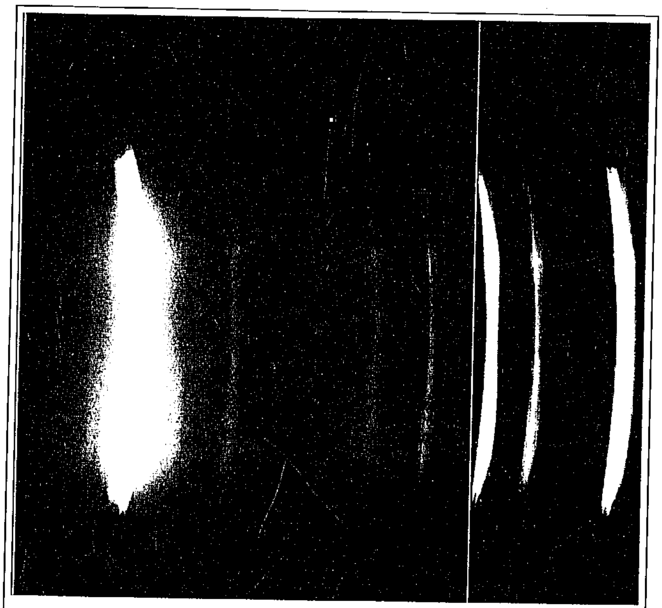


Fig. 11: Final image after reconstruction.

Figure 11 shows the image which was recreated from raw data at ground and is stored in FITS format.

Summary

In this technical report, we discuss the design and implementation of the software, which was used to operate hardware (two CCD and a Digital IO board) on a real time basis in an experiment that was flown in a balloon from Hyderabad on 8 March 2010. The software used four processes and threads to carryout this multitasking online. The software performed flawlessly contributing to the success of this experiment.

Acknowledgement

Major programs such as the balloon experiment require the support and cooperation of several individuals, groups, and institutions. We acknowledge Mr. D. Subhedar and Dr. Y. B. Acharya for useful discussions. We are thankful to Mr. R. Narayanan and Mr. R. P. Singh for helping with some units of the hardware. The temperature and vacuum tests were carried out in the EnTF and ESSG groups at the Space Application Center, (SAC), Ahmedabad. We thank Shri. D. R. Patel, General Manager, and Shri. J. J. Mistry, Manager, and the staff of these two divisions for their help in carrying out the tests. We thank Mr. Christopher Mendillo of Boston University, for technical support. The help and support received by Prof. R.K. Manchanda, Shri S. Sreenivasan and the staff members at the TIFR National Ballon facility is duly acknowledged. We thank Prof. J.N. Goswami, Director, PRL, for his constant support and encouragement to this program.

Reference

www.linuxforums.com

www.winsystems.com

www.flicamera.com

www.wikipedia.com

Appendix A

camera.h

This file contains important global variables and structures.

#defineSHARED_KEY	9999	/*shared key for shared memory between all processes*/
#define MAX_IM_SIZE	2500000	/*Maximum Image size*/
#define NUMBER_OF_CAMERAS	2	/*Cameras*/
#define CCD_X_SIZE	1064	/*Total number of pixels in X axis*/
#define CCD_Y_SIZE	1027	/* Total number of pixels in Y axis*/
#define CCD_X0_OFFSET	8	/*Starting offset of usable areaX*/
#define CCD_Y0_OFFSET	0	/*Offset to usable area Y/*
#define FOLDER	"%Y-%m-%d/"	/*Year-month-date wise folder name*/
#define IMAGE	"%Y-%m-%d_%H%M%S.%j"	/* Your-month-date hour, min, sec, dat wise image name */
#define CATALOG	"%Y-%m-%d.%j.cat"	/*Year-month-date,day wise catalog file name*/
#define IMAGE_TYPE	"fits"	/*Type of the image, either raw, fits or ascii*/
#define WATPER	5	/* Watchdog period in seconds */
#define SNDPER	1	/* Sendpic period for dio_loop on seconds */
#define WATCH_TIMEOUT	300	/* Number of WATPER for timeout */
#define TELE COMMAND_SIZE	8	/* Size of Telecommands in bits */
#define DEF_EXPTIME_0	60000	/*Default Exposure time for cam */
#define DEF_EXPTIME_1	20000	/*Default Exposure time for cam 1*/
#define DEF_SEPERATION_0	0	/*Default Separation time for cam 0*/
#define DEF_SEPERATION_1	60000	/*Default separation time for cam 1*/
#define DEF_X_BIN_0	1	//X binning size for cam 0
#define DEF_X_BIN_1	1	//X binning size for cam 1
#define DEF_Y_BIN_0	2	// Y binning size for cam 1
#define DEF_Y_BIN_1	8	//Y binning size for cam 1
#define DEF_X_ORIGIN_0	0	//Pixel X ORIGIN of image for cam 0
#define DEF_X_ORIGIN_1	30	//Pixel X ORIGIN of image for cam 1
#define DEF_Y_ORIGIN_0	200	//Pixel Y ORIGIN of image for cam 0
#define DEF_Y_ORIGIN_1	0	// Pixel Y ORIGIN of image for cam 1
#define DEF_X_SIZE_0	1056	// Pixel X size of image for cam
#define DEF_X_SIZE_1	970	// Pixel X size of image for cam 1
#define DEF_Y_SIZE_0	700	// Pixel Y size of image for cam 0
#define DEF_Y_SIZE_1	950	// Pixel Y size of image for cam 1
#define DEF_TEMP_0	-33	//CCD temperature for cam 0
#define DEF_TEMP_1	-33	//CCD temperature for cam 1
#define DEF_MODE_0	"A"	//(char) Camera 0 startup mode
#define DEF_MODE_1	"a"	//(char) Camera 1 startup mode
#define DEF_OUTFILE	"/home/balloon/flight/current/src/dummy.raw"	// (string) Dummy image to send
#define DEF_OUTFILE_NAME	"dumm.raw"	// (string) Dummy image to send
#define COOLER_OFF_TEMPERATURE 36		/*Cooler OFF temperature for both cameras*/
#define COOLER_ON_TEMPERATURE -33		/*Cooler ON temperature for both cameras*/

```
#define CHIP_1_BITS  36  //number of bits to write to
                         first dio chip <=36*/

#define CHIP_2_BITS  34  //number of bits to write to
                         second dio chip <=34
```

struct sm_t

This structure is used as a shared memory between all four processes. So if any changes happen in this structure value it will be notified to all process that there is some change in shared memory.

```
typedef struct {

volatile char mode[2];

volatile unsigned

int camalive[2];          /* To check cam_loop process is alive or
                          not for both camera*/

volatile unsigned         /*To check dio_loop process is alive or
int sndalive;             not */

volatile unsigned         /*if this variable is set then all process
int killall;              will be killed and program will be
                          terminated*/

volatile char             /* It contains file name with file path
lastsave                  which is stored last by cam_loop
[2][BUFF_SIZE];           process for both camera individually/*

volatile char             /*Its is like last save but it contains only
lastsave filename         file file path*/
[2] [FILE_
NAME_SIZE];

volatile char             /* It contains name of catalog file*/
catname
[BUFF_SIZE];

volatile unsigned         /*It contains type of elecommands for
char telecommand;         camera configuration*/

}sm_t;
```

struct exp_t

This structure is used configure CCD cameras.

```
typedef struct {

char mode;        //Set mode of cameras

int camnum;       //Camera number

int exptime;      //Set exposure time of cameras

int septime;      //Set separation time of cameras

int x_bin;        //Set X binning of cameras

int y_bin;        //Set Y binning of cameras

int x_origin;     //Set X origin of cameras

int y_origin;     //Set Y origin of cameras

int x_size;       //Set x size of cameras

int y_size;       //Set y size of cameras

int temp;         //Set temperature of cameras

}exp_t;
```

struct cam_t

This is used as a camera structures. Each camera connected to this software is identified by this structure. So, individual structure will be used for different cameras.

```
typedef struct {

flidomain_t       //Domain of camera like serial, parallel,
domain;           USB or inet

char *dname;      //domain name

char *name;       //name of the device

}cam_t;
```

struct Remaining_data

This structure is used to keep record of how many bits are remaining in a single byte to transmit after transmitting each frame.

```
typedef struct{

unsigned long     /*current location in file from where 70
index;            bits data should be started to
                  transmit*/

unsigned          /* Remaining bits of a current byte
remaining_bits;   which will be transmitted in next 70
                  bits lot*/

unsigned long     //File size
filesize;         {Remaining_Data;
```

watchdog.c

This file contains following functions:

Main

Name	int main(int argc,char **argv)
Description	It shares memory for watchdog process and create watchdog loop.
Argument	NA
Return Value	As this program is in infinite loop it will not return any value. But if there will be any problem then it will return 1 to operating system.

Watchdog_loop

Name	void watchdog_loop(void)
Description	It creates different processes and keep watch on it. If any of the process will be idle for some specific time then it will kill that process and recreate it.
Argument	void
Return Value	void

Open_shared_memory

Name	sm_t * open_shared_memory (key_t key)
Description	It creates shared memory based on given shared key. This function will be used by all (as external function) process for sharing memory.
Argument	key_t key -> Its is a shared key
Return Value	It returns pointer to the shared memory.

cam_loop.c

This file contains these many functions.

init_cameras

Name	int init_cameras(void)
Description	It finds the cameras by calling find cams function and initialize them.
Arguments	NA
Return Value	Number of Cameras detected.

find_cams

Name	int find cams (flidomain_t domain, cam_t **camf)
Description	It Finds the Cameras connected to the computer.
Arguments	flidomain_t domain-> Domain of flicamera cam_t **camf->Address of Different Camera structures
Return Value	Number of Cameras

Close_cameras

Name	int close_cameras(void)
Description	It closes cameras.
Arguments	int numcams->Number of cameras that are to be closed.
Return Value	NA

Cam_loop

Name	void cam_loop(int i)
Description	It configure the cameras with default configurations and then call the expSequence Function.
Arguments	int i->Camera Number.
Return Value	NA

exp Sequence

Name	void exp Sequence (exp_t exp)
Description	It configures the cameras, allocates image buffer, capture the image and store it in to some File by calling do_exposure function, de-allocate the Image Buffer.
Arguments	exp_t exp-> Structure contains all parameters of camera.
Return Value	NA

Config_camera

Name: int config_camera(exp_t exp)

Description: It configures the camera according to the parameters contain in its argument.

Arguments: exp_t exp-> Structure contains all parameters

Return Value: 0 on successfully file storing else -1

do_exposure

Name: int do_exposure(exp_t exp)

Description: It captures the image(using FLI library and stores in to the disk(either in fits or raw or ascii format), name it with date and time. and also stores its parameters to catalog File.

Arguments: exp_t exp-> Structure contains all parameters

Return Value: 0 on successfully file storing else -1

writefits

Name: int writefits(char *filename, int width, int height, void *data)

Description: It writes image as .fits file.

Arguments:
char *filename-> name of the file
int width-> Number of Pixels in each row
int height-> Number of Pixels in each column
void *data-> Buffer that contains the image and to be stored int to the file

Return Value: 0 on successfully file storing else -1

writeascii

Name: int writeascii(char *filename, int width, intheight,u_int16_t*data)

Description: It writes image as ascii format.

Arguments:
char *filename-> name of the file
int width-> Number of Pixels in each row
int height-> Number of Pixels in each column
u_int16_t *data-> Buffer that contains the image and stored in to the file

Return Value: 0 on successfully file storing else -1

writeraw

Name: writeraw (char *filename, char mode, int width, int height, u_int16_t *data)

Description: It writes image as .raw format.

Arguments:
char *filename-> name of the file
char mode-> Mode of the camera (a or A)
int width-> Number of Pixels in each row
int height-> Number of Pixels in each column
u_int16_t *data-> Buffer that contains the image and stored in to the file

Return Value: 0 on successfully file storing else -1

writecat

Name: int writecat(char *catname,char* imgname,exp_t *exp)

Description: It makes cat file and stores image name and its parameters like
->Image Name,
->Camera Mode,
->Camrea Number,
->X_Bining,
->Y_Bining,
->Start_X,
->Start_Y,
->End_X,
->End_Y,
->Current Tenperature of Camera,
->Flag,
->Power Acquired by the Cooler of respective Camera
->Configuration File Used at the time of Image capturing sequentially.

Arguments:
char *catname-> name of the file
char *imagename-> name of the image
exp_t *exp-> Structure contains all above parameters

Return Value: 0 on successfully file storing else -1

printCameraDiagonastic

Name: void printCameraDiagonastic(int cam_i)

Description: It printsthe Camera Diagnostics like,
->Model
->Hardware Revision
->Firmware Revision
-<Pixel Size
->Array Area
->Visible Area
->Temperature

Arguments: int cam_i-> Camera Number

Return Value: NA

dio_loop.c

This file contains this many functions.

Dio_loop

Name	dio_Loop
Description	It sends data to telemetary through DIO Board
Arguments	N.A
return value	N.A

thread_function

Name	thread_function
Description	This is the telemetry thread. It creates telecommands thread. It waits for strobe on DIO board. Then it checks strobe. If strobe is telecommand strobe then it signals to telecommad thread (to change telecommand variable) and wait for interrupt. If strobe is telemetry strobe then it sends data to telemetry by calling writedata function.
Arguments	N.A
return value	N.A

thread_function_for_tele_commands

Name	thread_function_for_telecommands
Description	This thread Function waits for the signal given from thread_function (Telemetry thread). When it receives signal, it reads telecommand data from the DIO board andupdate telecommand variable in the shared memory.
Arguments	N.A
return value	N.A

writedata

Name	unsigned Power(unsigned se,unsigned Pow)
Description	write 70bits of data to the dio board
Arguments	unsigned char* data:- Data to be sent
Remaining_Data remaining	structure that contains information like number of bytes which are to be sent, remaining bits to be sent of a articular last byte and the total size of the file
return value	structure that contains information like number of bytes which are already sent, remaining bits to be sent of a articular last byte and the total size of the file.