GMRT Internal Technical Report

NCRA • TIFR

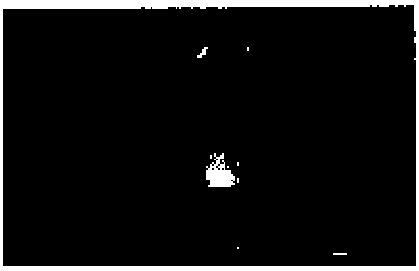
PSR_SYSTEM_TEST.C

version : 1.0 **date : 14th May 2004.**

Software tool for testing the pulsar receiver system .

written by

Mr. Irappa M. Halagali



Guided by

Mr. Kiran H. Dahimiwal
&
Prof. Yashwant Gupta

Report on psr_system_test.c program

I OVERVIEW :

This program is for checking the BINARY OUTPUT files from IA (Incoherent Array) & PA (Phased Array or Coherent Array) modes of the pulsar receiver using test data generated in the correlator or GAC (GMRT Array Combiner) system. The test data generated in the correlator system goes to pulsar receiver system through GAC system. The test data consists of patterns like sine/pulse , pat1 , random & gac1. These pattens repeats with every time samples of 256 spectral channels. The test data with patterns like sine/pulse , pat1 & random are generated in the correlator system and test data with gac1 patten in the GAC system. These binary files can be acquired from the IA & PA pulsar receivers using " devcat " program. The syntax of this "devcat" command is ;

"time devcat {Number of blocks} {das port} > {file name with path }"

The binary output file has 4096 (8 kbytes) words , in each block.

The binary files consists of 16 bit words , with a sequence of 256 such words repeating on time samples. The beginning/end of each sample is specified by a marker word. Data and marker words are written by a set of DSP node cards. Marker word appears at the end of each time sample. Last DSP node card writes data first followed by first , second , third , till last but one DSP node cards. The GPS signal , which is required for putting time stamps in the astronomical data has been coupled in these words. Presently GPS is coupled on MSB for IA and on LSB for PA systems. This GPS in the data can be enabled or disabled using SPDT switch provided in the pulsar receiver rack.

Ideal requirements of the data in the binary file to pass through this program with no errors , are as follows ;

1. The data should be synchronized with init , means data should always start with channel 0 .
2. Ideally Data from each block should be identical , except marker. Block length in this program is , the total number of words written by nodes involved (one or two for IA & 4 for PA) in a particular data acquisition including marker words written by all DSP nodes.
3. If marker comes in the 256 * n (n can be integer 1,2,3,4 .....) word , then marker can appear more than once in a block. This happens in total I (dual) polarization of IA and data from PA.
4. If marker comes in other than 256 or it's multiple words , then marker can be in any other word , but it must be only once in a block. Data from IA system with either u or l (single) polarization has marker in second word.
5. Number of spectral channels should be 256.

Using a template of one block length , this program checks the recorded data file and puts a report in the "output.data" log file . From this user can find the following :

1. Which bit(s) of the 16 bit word is (are) corrupted .
2. Which channels are corrupted .
3. Which node(s) is(are) giving corrupted words.
4. Which blocks are corrupted .

Refer Appendix A for "code sequence and flow chart" of this program .
And refer Appendix E for descriptions of the technical words used in this report .

II Main features of this program :

1. User friendly :

1. This program is made quite generalized by taking only common and minimum command line arguments from user.
2. This program can create the template file from the data itself , by comparing 'n' blocks , where 'n' is the user selectable number in the "psr_system_test.hdr" file .
3. GPS bit masking :
   This program can mask the GPS bit in the data at 1st or 9th through 16th bit of the 16 bit word. GPS in the data can be disabled , for which user should give command line or in file argument for <GPS bit> as 0.
4. This program displays "IN PROGRESS" and then displays "xxx blocks checked" while checking the binary file. xxx depends upon the value set for BLK_CHECK (Blocks Checked) in the "psr_system_test.hdr" file. This feature is helpful while checking the binary files , having long stretch of data. Here each block is of 256 words length , irrespective of data from which system and number of accumulations and polarization .
5. This program even works for different data from each node , because the template consists data from all nodes . Template file of IA system , single polarization will have only one node's data . With data from all DSP node cards identical , user can check the binary data file with any one node's data as template by giving the DSP node card number for the user argument <Node #> . DSP node cards , counting starts from Zero.

3. Compatibility :

This program writes errors found in the binary file to "output.data" file with block number and byte number. Here byte number is compatible to the value in " od -xv -Ad " of the binary file. This feature is useful to the user for debugging.

4. Easy to modify :

Those variables expected to be changed in future by the user are kept in the "psr_system_test.hdr" file. These user variables are read by the header_in subroutine in the program. These variables are as follows :

1. STRT_BLK : Start Block is the starting block for making template file , while using the program with code # 0 (without template mode).
2. NBCS : No. of Blocks to be Compared to make out the template.
3. BLK_CHECK : To display 'Number of blocks checked' while checking the binary file.
4. MAX_BLK_SIZE : Maximum block size of the input and template data.
5. MAX_NODE_SIZE : Maximum number of words in a node.
6. FPL :First Part of "input*.data" file , which is not to be printed in "output.data" file. These are user arguments required by the program.
7. DIR1 : Directory in which "input*.data" files , "psr_system_test.hdr" file and "psr_system_test.help" file exists.
8. DIR2 : Directory in which "output.data" & "mean.data" files gets created by this program , for writing the results.

5. Handling vast variety of Binary files :

This program handles about 56 types of output data files. These files are the result of various pattens and other parameters such as ,

1. Patterns : sine/pulse , pat1 , random from correlator and gac1 test pattern from GAC rack.
2. Number of Accumulations : 64 , 32 , 16 & 8.
3. Number of Polarizations : u or l (single) , total I (dual) , 4 stokes (RR , LL , RL* , LR*) and total I (total intensity).
4. Source of data : IA and PA systems.

III DESCRIPTION :

A. Marker checking by this Program :

Marker in the data has been written by DSP node cards at the end of each time sample. But marker appears in the second word of IA system , single polarization with any number of accumulations and any pattern data. Marker in the data is of 16 bits or one word. Lower 8 bits for counter and upper 8 bits for node identification. To make the program independent of the number of nodes writing the data in IA & PA systems , upper 8 bits have been masked. This program can handle the marker and it's increment in the following way :

a. Maker in any one word of the whole block ,except 256*n channels, where n is the integer 1,2,3,4 ...... and has to increment by one in the next block or,

b. Marker in 256 * nth word with same marker values in all 256*n channels except last marker word , where it has to get incremented by one .

Refer Appendix B for more information on marker and related topic .

B. GPS masking by this Program :

GPS is required for putting time stamps in the astronomical data. However GPS is masked and ignored by this program. GPS can be enabled or disabled by the user before acquiring the data from IA & PA , using SPDT switches provided in the pulsar receiver rack. For the IA mode , MSB of the data is being used as GPS bit. And in the PA mode LSB is being used as GPS bit and additional SPDT switch facility has been given to enable GPS in the MSB and disable for the LSB.

<GPS bit> is a user argument , either from command line or through "input*.data" file. Depending upon the <GPS bit> user argument , gps_word_hex will get prepared in the program for masking the gps bit in the data words and marker words.

Since the standard/template files are with gps disabled , program masks gps bit for this template file before checking the data.

If data doesn't have GPS , then command line or through "input*.data" file user argument for <GPS bit> should be 0 .

If gps bit is 1 , marker gets incremented by 2 instead of 1.
eg. if marker is 0x0001 (0xfffc for node 0, 0xfefe for node 1, 0xfdfe for node 2 & 0xfefe for node 3 on LINUX PC and BYTEs will be swapped on Unix PC), then it has to increment to 0x0002 . Because of GPS present as low , marker data becomes 0x0003 (gps bit inverted). This has been taken care in the program .

GPS in the 2nd through 8th bit of data word cannot be taken care by the program , because of marker (LSByte) increments in random way.

GPS in the 9th through 16th bits of data word can be taken care by the program easily. Because these bits (MSByte) of the marker word gets masked , since these bits used for node identification.

So <GPS bit> user argument can be 0 , 1 or 9 through 16.

C. DATA checking by this Program :

A block of data from binary input file after initial offset gets read by the program and stored in the in_data array. This data gets ORed with gps_word_hex to mask the GPS bit. The template file created by either user or program , also gets ORed with gps_word_hex and gets stored in the std_data array. These two arrays gets compared word by word. If any error , it will be written to "output.data" file in the DIR2 (~/irappa/output/) area in the following form.

blk # : blk word # : ch # : node id : byte # : std_data : in_data :

Along with this comparison , data gets added in the mean array with the corresponding channel. And square of each word gets added in the rms array with the corresponding channel. Both mean and rms arrays are of 256 elements each corresponds to spectral channel. Once eof of the binary file reached , then all the 256 elements of both the arrays gets divided by N_ACC (Actual Number of ACCumulations of 256 word blocks , excluding 256 word blocks of initial offset data). To find rms , elements in the rms array are square rooted and kept in the same array element. Then mean , rms and mean/rms ratio of the data written to First , Second and Third columns respectively in the "mean.data" file in the area pointed by DIR2.

Program displays the following messages , after eof of the binary file and mean , rms calculations ,
if no errors in the binary file ;

No error : SUCCESSFULLY COMPARED %ld Blocks of %s Binary Input file

otherwise it displays ;

%ld Marker errors & %ld Other errors.
%ld Total error(s) in %ld Blocks of %s Binary Input file.

And also displays ;

User parameters , errors if any , written to "output.data" file in DIR2 directory.

Mean , RMS & mean/rms ratio of the data is written to "mean.data" file's First , Second & Third columns respectively in DIR2 directory.

V LIMITATIONS :

1. This program checks only synchronized data from pulsar receiver.
2. This program doesn't check the accuracy of gps signal.
3. Checks only offline data.

VI FUTURE SCOPE :

Next version of this program shall be able to check ;

1. On line validation of asynchronized test data from pulsar receiver system.
2. Accuracy of gps signal.
3. Graphical user interface by user , using Tcl/Tk.

VII USER GUIDE :

A. ONLINE HELP :

Just by typing the executable file name "psr_system_test" on command line , user will get the help. Initial part of the help is copied from the help file "psr_system_test.help" from DIR1 area. In this user will get the information : code # , template/standard file name , Initial offset , block length , marker channel and GPS bit in the word.

Complete help information is as follows :

```
Usage (with template file) :
psr_system_test  <Binary Input file>  <code #>

Usage (without template file) :
 psr_system_test   <Binary Input file>  < 0 (code #)>   <initial offset>    <block
length>  <marker channel>  <GPS bit>  <Node #>

If code # is 0 template & "input*.data" files not required.

help information from DIR1/ or  ~/irappa/input/psr_system_test.help    file is :


   Code Standard/Template file           initial   block    mark  no. of pols
   no.                                   offset    length   er ch.  in output.

   1    sine_64acc_single_IA.std         256       256      2      1(u or l)
   2    sine_32acc_single_IA.std         512       512      2      1(u or l)
   3    sine_32acc_dual_IA.std           512       512      256    1(total I)
   4    sine_16acc_single_IA.std         1024      1024     2      1(u or l)
   5    sine_16acc_dual_IA.std           1024      1024     512    1(total I)
   6    sine_8acc_dual_IA.std            2048      2048     1024   1(total I)
   7    pat1_64acc_single_IA.std         256       256      2      1(u or l)
   8    pat1_32acc_single_IA.std         512       512      2      1(u or l)
   9    pat1_32acc_dual_IA.std           512       512      256    1(total I)
   10   pat1_16acc_single_IA.std         1024      1024     2      1(u or l)
   11   pat1_16acc_dual_IA.std           1024      1024     512    1(total I)
   12   pat1_8acc_dual_IA.std            2048      2048     1024   1(total I)
   13   random_64acc_single_IA.std       256       256      2      1(u or l)
   14   random_32acc_single_IA.std       512       512      2      1(u or l)
   15   random_32acc_dual_IA.std         512       512      256    1(total I)
   16   random_16acc_single_IA.std       1024      1024     2      1(u or l)
   17   random_16acc_dual_IA.std         1024      1024     512    1(total I)
   18   random_8acc_dual_IA.std          2048      2048     1024   1(total I)
```

19	gac1_64acc_single_IA.std	256	256	2	1(u or l)
20	gac1_32acc_single_IA.std	512	512	2	1(u or l)
21	gac1_32acc_dual_IA.std	512	512	256	1(total I)
22	gac1_16acc_single_IA.std	1024	1024	2	1(u or l)
23	gac1_16acc_dual_IA.std	1024	1024	512	1(total I)
24	gac1_8acc_dual_IA.std	2048	2048	1024	1(total I)
25	sinc_32acc_full_PA.std	4096	4096	1024	4(RR,LL,RL*,LR*)
26	sine_32acc_total_PA.std	1024	1024	256	1(total I)
27	sine_16acc_full_PA.std	4096	4096	1024	4(RR,LL,RL*,LR*)
28	sinc_16acc_total_PA.std	1024	1024	256	1(total I)
29	sinc_8acc_total_PA.std	2048	2048	512	1(total I)
30	pat1_32acc_full_PA.std	4096	4096	1024	4(RR,LL,RL*,LR*)
31	pat1 _32acc_total_PA.std	1024	1024	256	1(total I)
32	pat1_16acc_full_PA.std	4096	4096	1024	4(RR,LL,RL*,LR*)
33	pat1_16acc_total_PA.std	1024	1024	256	1(total I)
34	pat1_8acc_total_PA.std	2048	2048	512	1(total I)
35	random_32acc_full_PA.std	4096	4096	1024	4 (RR,LL,RL*,LR*)
36	random _32acc_total_PA.std	1024	1024	256	1 (total I)
37	random_16acc_full_PA.std	4096	4096	1024	4(RR,LL,RL*,LR*)
38	random_16acc_total_PA.std	1024	1024	256	1(total I)
39	random_8acc_total_PA.std	2048	2048	512	1(total I)
40	gac1_32acc_full_PA.std	4096	4096	1024	4(RR,LL,RL*,LR*)
41	gac1 _32acc_total_PA.std	1024	1024	256	1(total I)
42	gac1_16acc_full_PA.std	4096	4096	1024	4(RR,LL,RL*,LR*)
43	gac1_16acc_total_PA.std	1024	1024	256	1(total I)
44	gac1_8acc_total_PA.std	2048	2048	512	1(total I)

For IA system GPS is on 16th bit.
For PA system GPS is on 1st bit.

For more info. look at ~/irappa/input/ or DIR1/psr_system_test.help1 file.

ini_offset and blk_len. 's are in terms of number of words .

User parameters , errors if any , written to "output.data" file in the DIR2 directory.

Mean ,RMS & mean/rms ratio of the data is written to "mean.data" file in First , Second & Third columns respectively in the DIR2 directory.

B. Some Warnings and things to be noted for using and while modifying this program.

Warnings :

1. Current maximum block length is 8192(8k), to increase this , change the MAX_BLK_SIZE ,which decides the array size of in_data & std_data.
2. STRT_BLK should not be more than the actual data blocks in the Binary file. Otherwise it takes zero's as template data (or the word which has been ORed or ANDed with the in_data to mask GPS) for all words of a block and gives huge errors.
3. Maximum length of parameter file , template file and output files is 80 characters including path.
   eg. /home/pulsar/irappa/input/input1.data .

Note :

1. "input*.data" files & "psr_system_test.help" file should be in the directory pointed by DIR1 in the program.
2. Template files have to be kept in a directory pointed by the first parameter in the corresponding "input*.data" file .

C. Using this program with template file :

For this usage is : psr_system_test <Binary Input file> <code #>

Code number can be 1 and above. The corresponding "input*.data" (* is the code #) file in DIR1 area will get opened by the program for getting the information such as area/path in which template file exists , initial offset , block length , marker position (channel number of the marker) , GPS bit in the data and Node number.

Initial offset and block length are in number of words. Each information should be on separate line.

eg. input3.data file : I part

```
/home/pulsar/irappa/template/sine_32acc_dual_IA.std
512
512
256
16
5
```

input1.data file : II part

standard file	: sine_32acc dual_IA.std
initial offset	: 512
block length	: 512
marker channel	: 256
GPS added at bit	: 16
Node number	: 5
pattern name	: sine
Accumulations	: 32
polarization	: Dual
system	: IA

User is free to create any input file in DIR1 area with any code number , except the code number existed. Existed code numbers can be confirmed in the "psr_system_test.help" file in DIR1 area. The first part of the newly created "input*.data" file should be added in the "psr_system_test.help" file on appropriate line , which will get displayed when user asks for help by typing the executable file.

"input*.data" file must have , complete path of the template file. Each variable should be on separate line for part I. Information in this part I should be correct and should be in the following order ;

I line : complete path and name of the template file.
II line : Initial offset (number of words).
III line : Block length (number of words).
IV line : marker channel. (word number).
V line : GPS bit.
VI line : Node number.

Part II is optional , which is required only for users. It will be written to "output.data" file as it is.

User has to create template file from the data file which has been acquired with GPS disabled. And template file has to be kept in the directory/area mentioned in the "input*.data" file. The block length of this template file should be correct.

D. Using this program without template file :

For this usage is :

psr_system_test <Binary Input file> < 0 (code #)> <initial offset> <block length> <marker channel> <GPS bit> <Node # > .

a. Initial offset :

The number of words to be skipped initially , which is the dummy data 0xffff (inversion of zero). Program will skip these many words from the starting of the file and starts checking , rest of the words in the binary file.

b. Block length :

Block length can be , the number of words written by a node including the marker written by it , irrespective of number of nodes involved in writing the data for a particular data acquisition . But in this "psr_system_test" program total number of words written by all nodes involved in a particular data acquisition including marker words , is taken as block length. So initial offset and block length's are equal.

Difference in the single polarization and multiple polarization block length and marker occurrence has been explained below.

1. Multiple polarization : In this modes of data acquisitions , block length is the number of words after initial offset to a next marker where it's value gets incremented. Since marker gets incremented at last marker position in a block , last marker value of a block and it's subsequent markers of next block , except last marker remains same.

2. Single polarization : In this mode of data acquisition marker appears in the second word. Since only one node is writing the data , marker comes only once in a block. Marker gets incremented in each successive blocks. So block length is the number of words after initial offset to a next marker minus 2 or the number of words after a marker to the next marker.

c. Marker channel :

This is the word number at which marker comes in the data from each DSP node card of IA & PA modes of pulsar receiver. A word after the marker is first word.

d. GPS bit :

This is the bit number of a word at which GPS is added. LSB is the first bit.
eg. For IA system GPS is coupled at MSB of the word , so <GPS bit> is 16.
This information is required to mask the particular bit. 0 for GPS off.

e. NODE number :

This is the node number , whose data to be taken as standard/template data for comparing the whole data. This facility has been given to compare the whole data with the data from any one node instead of all respective nodes .

eg. If a system has 4 nodes and user believes 2nd node's data is good , even though all nodes should give the same data . Then by providing node number 2 on the command line or through input*.data file for the <NODE #> user argument , program will take out the 2nd node's data from the template file created by user or gets created by the program. Replaces nodes 0 , 1 & 3's data in the std_data array by 2nd node's data. So whole data gets compared with the 2nd node's data.

If user argument <NODE #> is equal to or greater than the number of nodes written the data of a binary file , then template will have data from all nodes.

eg. If a binary file is written by 2 nodes. Program will create a template with node 0 and 1's data if user argument <NODE #> is either 2 or greater than 2.

Refer Appendix C for template file preparation by this program .

E. Output files from this program & interpretation :

1. "output.data" file.

Any data or marker errors in the binary file will be written by the program in the output file "output.data" , in the following form ;

```
blk # :    blk word # :    ch # :    node id :    byte # :    std_data :    in_data :
```

blk # : It is the block number in which error has occurred. This number is from the beginning of the file. Means even though data checking starts after the initial offset , it is also considered while calculating the block number.

blk word : It is the word number in a block which is corrupted. This is helpful to find out , the rate at which a particular word of a block is corrupted.

ch # : This is the word number of a node card which is corrupted. This helps to find out the rate at which a particular channel of a node is corrupted.

node id : This number represents node which has written the corrupted data. This helps to find out the frequency at which a node card writes the corrupted words.

byte # : This is the number which is compatible to od -xv -Ad of binary input file.

std_data : This is the hex word taken by the program from the std_data array or template file for comparing the data.

in_data : This is the hex word taken from binary input file which is corrupted.

So from this information user can make out which bit(s) and channel of a node is corrupted and the rate at which. And user can make out usually which blocks are more error prone. Means starting blocks or blocks after few tens or hundreds of blocks etc.
Refer appendix D for the , sample contents of the "output.data" file.

2. "mean.data" file.

"mean.data" file contains the mean , rms and mean/rms ratio of the data for 256 channels in the First , Second and Third columns respectively.
mean = 1/N [n1 + n2 + ... + nN] ,
rms = sqrt of { 1/N [n1**2 + n2**2 .... + nN**2] } , ratio = mean / rms .

If mean/rms ratio is 1.000000 , it means data is correct for that channel of all nodes and all blocks.

VIII APPENDICES :

APPENDIX A :

1. Code sequence of this program :

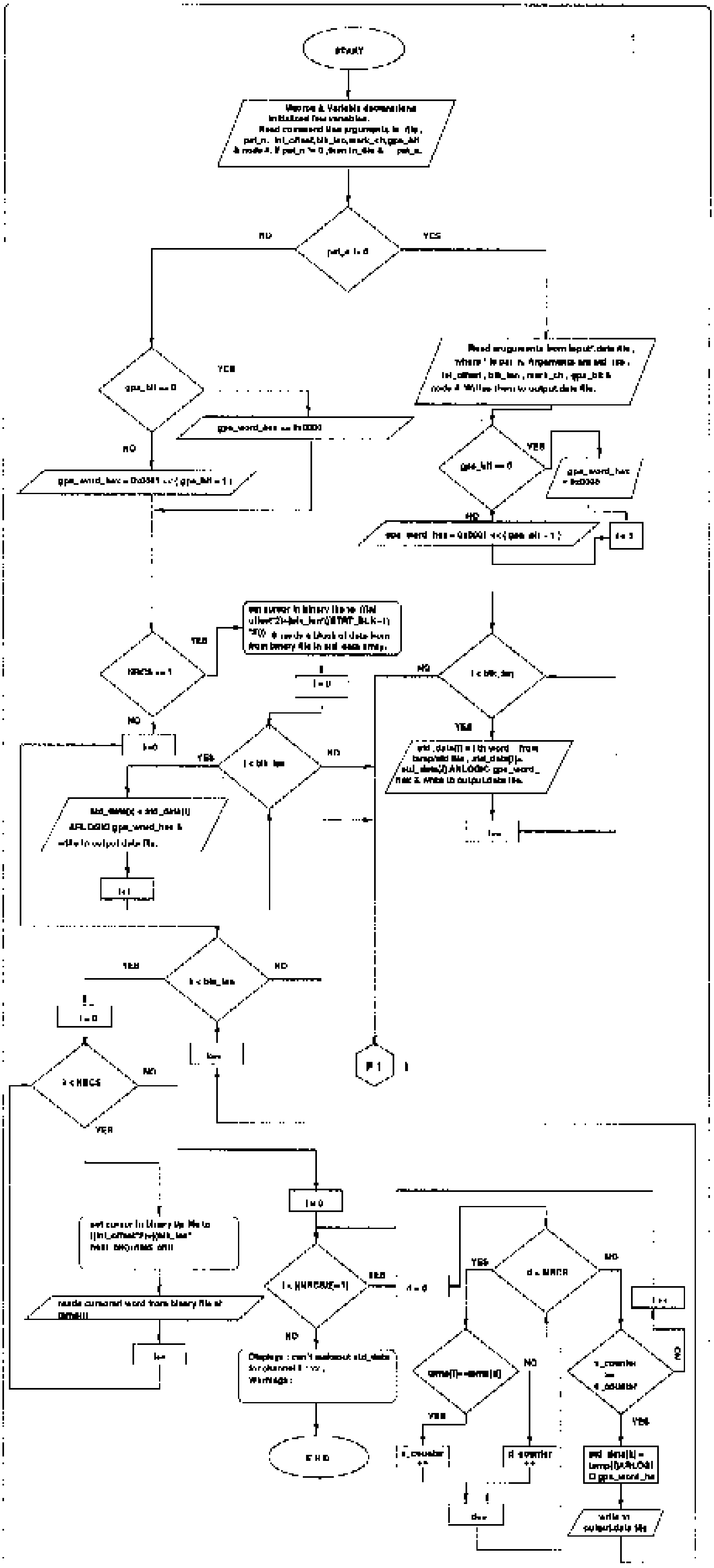
1. User variables read from "psr_system_test.hdr" file , by the header_in subroutine in the program.
2. Help for using the program.
3. Reads command line arguments.
4. Input/parameter file formed (path DIR1 & file number combined) & opens it.
5. Output file "output.data" formed (path DIR2 and file name combined) & opens it.
6. Reads user arguments from "input*.data" file , writes them to "output.data" file and closes it.
7. GPS masking word in hex , gps_word_hex is formed depending upon the <GPS bit> user argument.
8. Checks for block length , which should be less than the MAX_BLK_SIZE.
9. Reads the contents of standard/template file in std_data array , also writes it to "output.data" file and closes standard/template file.
10. Prepares the standard/template data in std_data array by comparing NBCS number of blocks from STRT_BLK (NBCS & STRT_BLK are user variables read from the "psr_system_test.hdr" file) , if this program is using in without template file mode. Each word of the template file gets ORed with the gps_word_hex to masks the GPS bit , before writing it to the std_data array. This template will be written to "output.data" file for user.
11. Modifies the std_data array , if user argument for <NODE #> points to any one node instead of all nodes.
12. Reads the data from binary file in blocks , masks the GPS bit in each word , checks all words by comparing with the corresponding template file word and writes to "output.data" file if any error. This loop continues till eof of the binary file and this loop calls the marker function marker_check to check marker word . If any marker errors , program writes them also to the "output.data" file.
13. This program displays " IN PROGRESS " and xxx blocks checked while doing the above work.
14. mean and rms values of the data also calculated by this program and writes them to "mean.data" file along with mean/rms ratio in columns.
15. After completing the comparison and mean , rms calculation work , program displays either

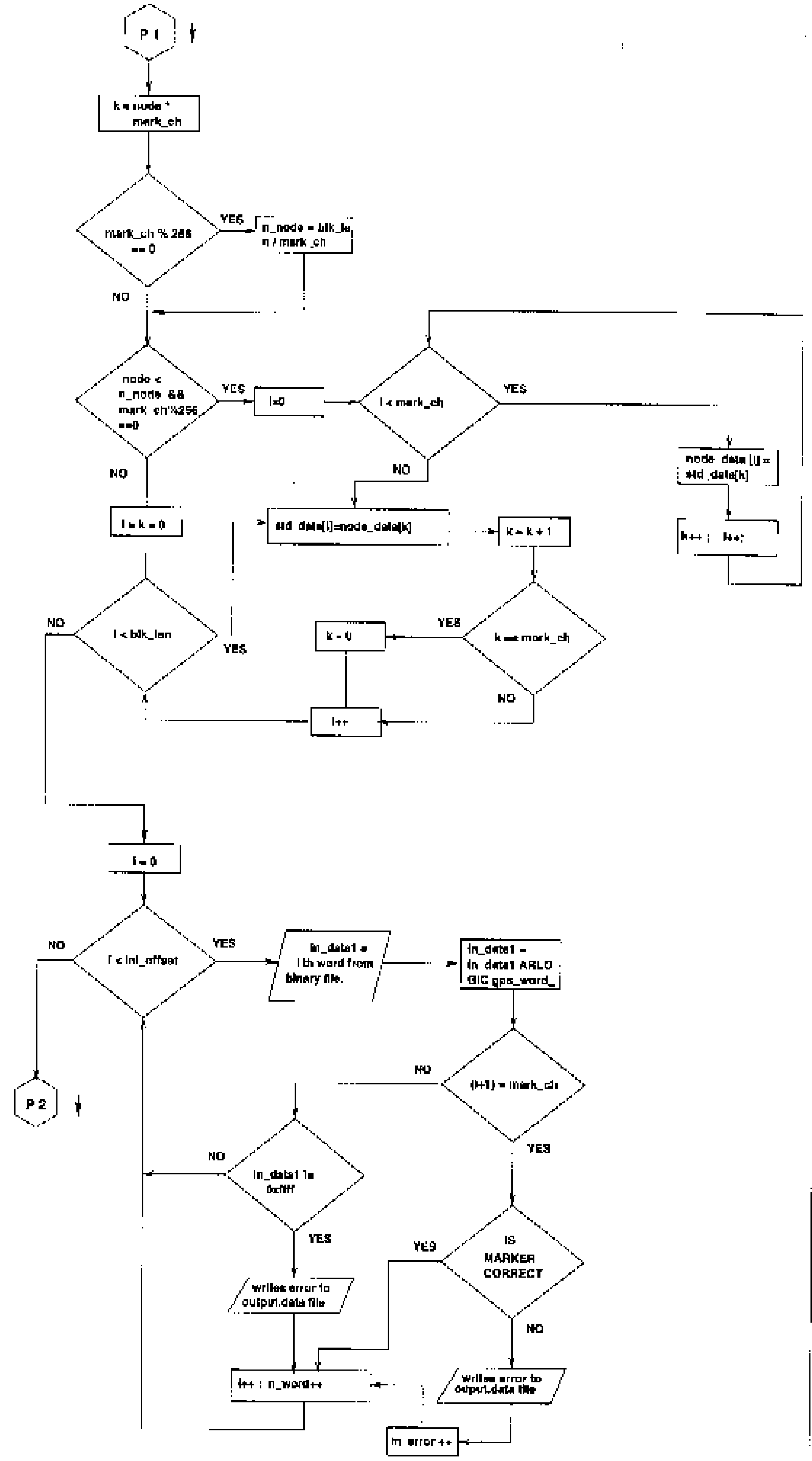
No error : SUCCESSFULLY COMPARED %ld Blocks of %s Binary Input file

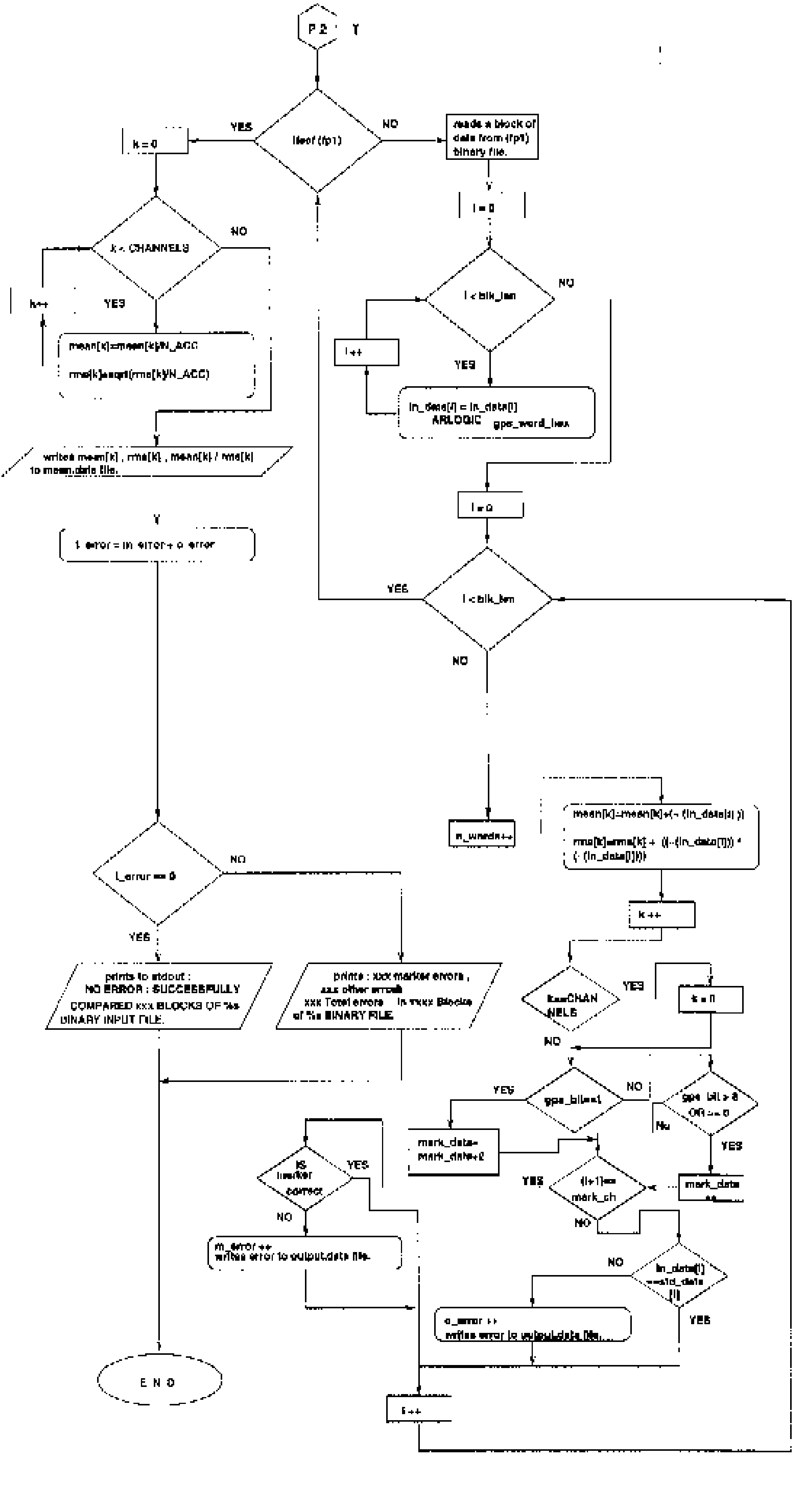
or ;

%ld Marker errors & %ld Other errors.
%ld Total error(s) in %ld Blocks of %s Binary Input file.

2. FLOW CHART OF THIS PROGRAM.


START
Macros & Variable declarations
Initialized few variables.
Read command line arguments, in_file, pat_n, int_offset, blk_len, mark_ch, gps_bit & node #. If pat_n != 0, then in_file & pat_n.
NO
pat_n != 0
YES
Read arguments from input data file, where * is pat_n. Arguments are std_file, int_offset, blk_len, mark_ch, gps_bit & node #. Writes them to output data file.
gps_bit == 0
YES
gps_word_hex = 0x0000
NO
gps_word_hex = 0x0001 << (gps_bit - 1)
gps_bit == 0
YES
gps_word_hex = 0x0000
NO
gps_word_hex = 0x0001 << (gps_bit - 1)
i = 0
set cursor in binary file to ((int_offset*2)+(blk_len*(START_BLK-1)*2)) & reads a block of data from binary file in std data array.
YES
NRCS == 1
NO
i = 0
i < blk_len
YES
std_data[i] = std_data[i] & LOGIC gps_word_hex & write to output data file.
i++
i = 0
i < blk_len
NO
YES
std_data[i] = i th word from temp/std file, std_data[i] = std_data[i] & LOGIC gps_word_hex & write to output data file.
i++
NO
i < blk_len
YES
NO
k++
i = 0
k < NRCS
NO
YES
P 1
set cursor in binary file to ((int_offset*2)+((blk_len*k)*2))
reads command word from binary file at temp[k]
k++
i = 0
i < (NRCS/2+1)
YES
d = 0
NO
Displays : can't makeout std_data for channel # ...
Warnings :
END
YES
d < NRCS
NO
temp[i] == temp[d]
YES
NO
s_counter ++
d_counter ++
d++
i++
s_counter >= d_counter
NO
YES
std_data[i] = temp[i] & LOGIC gps_word_hex
write in output data file


P 1
k = node *
mark_ch
mark_ch % 256
== 0
YES
n_node = blk_le
n / mark_ch
NO
node <
n_node &&
mark_ch%256
==0
YES
i=0
i < mark_ch
YES
node_data [i] =
std_data[k]
k++ ; i++;
NO
std_data[i]=node_data[k]
k = k + 1
NO
i = k = 0
NO
i < blk_len
YES
k = 0
YES
k == mark_ch
NO
i++
i = 0
NO
i < ini_offset
YES
in_data1 =
i th word from
binary file.
in_data1 =
in_data1 ARLO
GIC gps_word_
P 2
NO
(i+1) = mark_ch
YES
NO
in_data1 is
0xffff
YES
IS
MARKER
CORRECT
YES
NO
writes error to
output.data file
i++ ; n_word++
writes error to
output.data file
in_error ++


P 2
!feof (fp1)
YES
NO
k = 0
reads a block of data from (fp1) binary file.
I = 0
k < CHANNELS
NO
YES
k++
mean[k]=mean[k]/N_ACC
rms[k]=sqrt(rms[k]/N_ACC)
I < blk_len
NO
YES
I ++
in_data[i] = in_data[i]
ARLOGIC gps_word_hex
writes mean[k] , rms[k] , mean[k] / rms[k]
to mean.data file.
I = 0
t_error = m_error + o_error
I < blk_len
YES
NO
n_words++
mean[k]=mean[k]+(~(in_data[i]))
rms[k]=rms[k] + ((~(in_data[i])) *
(~(in_data[i])))
k ++
t_error == 0
NO
YES
prints to stdout :
NO ERROR : SUCCESSFULLY
COMPARED xxx BLOCKS OF %s
BINARY INPUT FILE.
prints : xxx marker errors ,
xxx other errors
xxx Total errors In xxxx Blocks
of %s BINARY FILE.
k==CHANNELS
YES
k = 0
NO
gps_bit==1
YES
NO
gps_bit > 8
OR == 0
No
YES
mark_data=
mark_data+2
IS
marker
correct
YES
NO
(i+1)==
mark_ch
YES
NO
mark_data
++
m_error ++
writes error to output.data file.
in_data[i]
==std_data
[i]
NO
YES
o_error ++
writes error to output.data file.
E N D
i ++

APPENDIX B :

1. Details of other parameters :

Parameters written below are in the following order ; initial offset , Marker position , marker initial value for node 0 , marker initial value for node 1 , marker initial value for node 2 , marker initial value for node 3 , marker reset value for node 0 , marker reset value for node 1 , marker reset value for node 2 , marker reset value for node 3 , block length and marker channel/position.

Sys.	Ac'ns	Pols.	Parameters
IA	64	Single	: 256 , 256 , 257 , 0 , 0 , 0 , 256 , 0 , 0 , 0 , 256 , 2
	32	Single	: 512 , 512 , 257 , 0 , 0 , 0 , 256 , 0 , 0 , 0 , 512 , 2
		Dual	: 512 , 256 , 1 , 258 , 0 , 0 , 0 , 256 , 0 , 0 , 512 , 256
	16	Single	: 1024 , 1024 , 257 , 0 , 0 , 0 , 256 , 0 , 0 , 0 , 1024 , 2
		Dual	: 1024 ,512, 1 , 258 , 0 , 0 , 0 , 256 , 0 , 0 , 1024 , 512
	8	Dual	: 2048, 1024 , 1 , 258 , 0 , 0 , 0 , 256 , 0 , 0 , 2048 , 1024
PA	32	T.Int'ty	: 1024,256, 1 ,257,513,770, 0 ,256,512,768,1024,256
		Full pol.	: 4096 ,1024 , 1 ,257 ,513 ,770 , 0 ,256 ,512,768,4096,1024
	16	T.Int'ty	: 1024,256, 1 ,257,513,770, 0 ,256,512,768,1024,256
		Full pol.	: 4096 ,1024 , 1 ,257 ,513 ,770 , 0 ,256 ,512,768,4096,1024
	8	T.Int'ty	: 2048,512, 1 ,257,513,770 , 0 ,256,512,768,2048,512

Presently node 1 writes for both lower and upper polarizations in single pol. acquisitions of IA. So marker data has same node id. But this doesn't need any modifications in the program even after changing the node 0 for lower polarization and node 1 for upper polarization , because upper 8 bits gets masked before checking the marker . Since only corresponding node will give the marker instead of node 1 for both polarizations , initial and reset values of marker will change as ;

eg. for single pol. 32 accumulations of IA system .

Lower polarization 512 , 512 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 512 , 2
Upper polarization 512 , 512 , 0 , 257 , 0 , 0 , 0 , 256 , 0 , 0, 512 , 2

2. Data rate in the correlator and pulsar receiver systems.

1. Sampling rate	: 32 Mhz.
2. Dly-Dpc write section	: 32 Mhz.
3. Dly-Dpc read section	: 32.25 Mhz.
4. FFT	: 32.25 Mhz.
5. FFT init.	: 16 u sec.
{ 1 / 32.25 MHz = 31.0078 n sec. * 516 = 16 u sec. Where , 512 clock cycles for 512 point FFT and 4 over head cycles for FFT operation in the ASIC }	
6. for 64 accumulations of IA	: 16 u sec. * 64 = 1024 u sec.
7. for 32 accumulations of IA	: 16 u sec. * 32 = 512 u sec.
8. for 16 accumulations of IA	: 16 u sec. * 16 = 256 u sec.
9. for 8 accumulations of IA	: 16 u sec. * 8 = 128 u sec.
10. for 32 accumulations of PA	: 16 u sec. * 32 * 2 = 1024 u sec.
11. for 16 accumulations of PA	: 16 u sec. * 16 * 2 = 512 u sec.

In PA calculations, number of integrations is multiplied by 2 , since voltage multipliers in iFFT subsystem does the integration.

Data comes from the Correlator in synchronization with the FFt init.

IA : For 32 accumulations , data gets accumulated to 32 FFt cycles and then each node writes it's marker value in the last channel. Time taken for this is 512 u seconds. In 64 accumulations one node can write. In 16 accumulations , data gets accumulated to 16 FFt cycles and written out ,then again data gets accumulated for another 16 FFt cycles and written out along with marker. So block length gets doubled. Here point to be noted is writing of marker by DSP node cards is fixed , ie 512 u seconds. So for 8 accumulations, block length becomes 4 times of the 32 accumulations .

Initial zero es will be of one block length. This block length depends upon the number of node cards writing the data . In 2 node IA system , each node card writes , 256 words for 32 accumulations , 512 words for 16 accumulations , 1024 words for 8 accumulations and so on.

PA : Since voltage multipliers in iFFT subsystem(which is used for computing the stokes) does 2 integrations on the data before it reaches the DSP system . Actual integrations is double of the integration done by the DSP bin. So 16 integrations works like 32 integrations in PA.

In full polar mode , each node card writes 4 stokes information and puts marker. So blk_len becomes 4 times of the total intensity mode of the same number of accumulations . In total intensity mode only one stoke information has been written by each node cards.

APPENDIX C :

1. Template file preparation by this program when NBCS is 1.

For making out the standard/Template file ,program goes to First word of a block. The first block is decided by STRT_BLK (Start Block) user variable from the "psr_system_test.hdr" file. For this block counting starts after the initial offset. If NBCS (Number of Blocks to be Compared to make out the template data file) is set to 1 by the user in the "psr_system_test.hdr" file , then STRT_BLK will be taken as template data in the std_data array. The size of this std_data array and in_data array is decided by MAX_BLK_SIZE (Maximum Block Size). Presently MAX_BLK_SIZE is 8192 in the "psr_system_test.hdr" file. This MAX_BLK_SIZE must be greater than the block length of the binary file. Otherwise program will report many errors.

2. Template file preparation by this program when NBCS is greater than 1.

If NBCS is more than 1 , then first word of all NBCS blocks will be taken in a array and they will get compared with each other. The word , which repeats more times , means any one word which has more similar words than the dissimilar words if any , will be taken as the standard/Template word for first word and will get stored in the std_data[0]. Otherwise program will give the warning "can't make out the standard data for channel xx , try by either changing the STRT_BLK or NBCS or 'marker position' and displays all words and program gets terminated. If dissimilar data is marker (all hex words are one less than the previous hex word or each word is incrementing by one if you invert all words) , then it means , user has given wrong marker position in the command line or through the "input*.data" file. Otherwise changing the STRT_BLK or NBCS may help to make out the standard/Template file.

If first word got the template data , then program takes the second words of all NBCS blocks in a array and they will get compared with each other. The word which repeats more times will get stored in the std_data[1]. This process will get continued till last word of block. For marker word , program puts ffff as the word in the corresponding element of std_data array , because marker checking is done with the known logic. If program successfully gets the std_data for all words of a block , then it starts comparison work.

APPENDIX D :

Contents of the "output.data" file.

```
blk # : 2        blk word : 1020   ch # : 1020   node id : 0   byte # : 4088
std_data  : ffbf        in_data : febf
blk # : 2        blk word : 1021   ch # : 1021   node id : 0   byte # : 4090
std_data  : ffaf        in_data : fe3f
blk # : 2        blk word : 1022   ch # : 1022   node id : 0   byte # : 4092
std_data  : ff8f        in_data : fd7f
blk # : 2        blk word : 1023   ch # : 1023   node id : 0   byte # : 4094
std_data  : ff5f        in_data : fc3f
blk # : 3        blk word : 0      ch # : 0      node id : 0   byte # : 4096
std data  : ff0f        in_data : fed1
blk # : 3        blk word : 1      ch # : 1      node id : 0   byte # : 4098
mark_data : 41          in_data : 40
blk # : 3        blk word : 2      ch # : 2      node id : 0   byte # : 4100
std_data  : feaf        in_data : ffff
blk # : 3        blk word : 3      ch # : 3      node id : 0   byte # : 4102
std_data  : ffff        in_data : fabf
blk # : 3        blk word : 4      ch # : 4      node id : 0   byte # : 4104
std_data  : feaf        in_data : fc3f
blk # : 3        blk word : 5      ch # : 5      node id : 0   byte # : 4106
std_data  : ff0f        in_data : fd7f
```

APPENDIX E :

Initial offset : The number of words to be skipped initially. This is the dummy data which is 0xffff (inversion of zero).

Block length : Block length is the number of words written by node cards involved in a particular data acquisition once each including marker words written by node cards.

Marker channel : This is the word number which is marker word.

GPS bit : This is the n th bit at which GPS is added in the data.

NODE number : This is the node number whose data to be taken as standard/template data for comparing the whole data.

blk # : This number represents the block number in a binary file.

blk word : It is the word number in a block

ch # : This is the word number of a node card.

node id : This is the DSP node card number. Node number starts from zero.

n_words : Number of words checked by this program.

byte # : ((n_words*2)-2).

std_data : This is the reference hex word from the template file for comparing the corresponding word of a block from the binary file.

in_data : This is the hex word taken from the binary file.

Marker position : Marker word number from the beginning of a block.

Marker initial value : Each node's first marker value in the synchronized binary data file is respective node card's initial value.

Marker reset value : It is the number at which each node's marker value gets rests to starting value.

IX REFERENCES :

1. Report on GPS by Mr. Kiran H. Dahimiwal.

2. Analysis software tools for marker for IA & PA by Mr. Kiran H. Dahimiwal.

X ACKNOWLEDGEMENT :

I am very much thankful to my colleagues and friends for their encouragement and support in producing this work and report. Viz ,

Shri S. Suresh Kumar
Mr. Jitendra P. Kodilkar
Mr. Charudatta P. Kanade
Mr. Manish M. Galande
Mr. Ramachandra M. Dabade