	IFSI CNR	Herschel Space Observatory SPIRE-DPU Virtual Machine	Ref: CNR.IFSI.2003.TR01 Issue: 1.0 Date: 27/03/2003 Page: 1 of 26
---	-------------	---	--

SPIRE-DPU Virtual Machine

ISSUE: 1.0

Prepared by: Riccardo Cerulli-Irelli



Distribution List :

K. King	SPIRE	
S.D. Sidher	SPIRE	
J.L. Auguères	SPIRE	
C. Cara	SPIRE	
R. Orfei	IFSI	
A. Di Giorgio	IFSI	
S. Molinari	IFSI	
S. Pezzuto	IFSI	
S. J. Liu	IFSI	

Document Status Sheet:

Issue	Revision	Date	Reason for Change
	Draft 1	11/06/2002	Initial issue
	Draft 2		Added ICALL and TABLE instructions New program figures with many new features.
	Draft 3	23/09/2002	Added ICPT, ICPF, TER13, TER15, TER17, EVNT, TXTBL instructions. New input file to simulate READ data words.
	Draft 4	3/02/2003	Added IRCALL, IRCPT, ICPF
1.0		27/03/2003	Changed packet definition in sect. 4.3

Reference documents

Document Reference	Title



Acronyms

CDMS	Central Data Management System
CI	Critical instruction
CNR	Consiglio Nazionale delle Ricerche
CPU	Control Processing Unit
DPU	Digital Processing Unit
FCU	Focal plane Control Unit
FIFO	First In First Out storage element
FIRST	Far InfraRed and Submillimeter Telescope
HK	HouseKeeping
HRS	High Resolution Spectrometer
HW	HardWare
DPU	Digital Processing Unit
I/F	Interface
IFSI	Istituto di Fisica dello Spazio Interplanetario
ISR	Interrupt Service Routine
LCU	Local Oscillator Control unit
LSB	Least Significant Bit(s)
LSU	Local oscillator Source Unit
MSB	Most Significant Bit(s)
mutex	Mutual Exclusive flag
NA	Not Applicable
OBS	On-Board Software
OS	Operating System
PC	Program Counter
PDU	Power Distribution Unit
RT	Real Time
S/C	Spacecraft
SPIRE	Spectral and Photometric Imaging REceiver
SS	Subsystem
SW	SoftWare
TBC	To Be Confirmed
TBD	To Be Defined
TBW	To Be Written
TC	Telecommand
TM	Telemetry
VM	Virtual Machine
WBS	Wide Band Spectrometer



Table of contents

Reference documents	2
1 Introduction.....	5
2 Reason for a Virtual Machine.....	5
3 The Virtual Machine	6
3.1 Critical instructions (CI)	6
3.2 VM structure	6
3.3 VM_Map Table	7
3.4 VM Program.....	7
3.5 VM program exec TC.....	9
3.6 VM Multitasking.....	9
3.7 VM Instructions.....	10
3.8 Instructions Format	12
4 VM Compiler/Simulator.....	13
4.1 Compiler	13
4.2 VM Simulator.....	17
4.3 Packetiser.....	18
4.4 Directory structure	19
5 Example	20



1 Introduction

This document describes the special command line interpreter of SPIRE-DPU, implemented in order to control the SS (via the LS I/F) in all the situation where the variations in time distance between commands must be less than few milliseconds. Such a command line interpreter can be seen as a kind of an elementary computer with a simple pseudo assembler commanding language that from now on we call virtual machine (VM).

The document describes also the developing SW tools associated with the VM which consists in a compiler, a simulator and a VM -program TC packet generator.

2 Reason for a Virtual Machine

The driving requirement for the VM is the time sequence constraint between SS commands during an observation. The time sequence jitter on the SS commands (LS I/F) goes from seconds down to 10us. Consider the following example:

```
Cmd1      @ T
Cmd2      @ T + t1 +-5ms = T2
Cmd3      @ T2 + t2 +- 100ms = T3
Cmd4      @ T3 + t3 +- 5us = T4
```

It is clear that, in a multi-task OS as Virtuoso, the only way to achieve the 10us and probably a 10 ms constraint is via an Interrupt Serviced Routine (with a high priority interrupt). It is also evident that once it has been decided to implement the interrupt environment, every command in the sequence should be sent via interrupt, so that all the commands will have the same (10 us) jitter in the time sequence.

The HW problem to generate the sequence of different period interrupts, is solved by using the DPU programmable 32 bit (1 MHz clock) down counter. This down counter starts decrementing its content from the last preset initial value, and generates an interrupt on zero value. Then the counter restarts again the cycle, beginning from the last preset initial value loaded before the zero count.

Now we have a mechanism which forces the execution of a routine (ISR_3) at pre-defined time intervals. Entering the routine, the relevant SS command must be sent. In order to preserve the time jitter constraint, this command must be already prepared (in a table).

After the command is sent (written in the low speed serial output I/F), we might want to change the down counter initial count for the next interrupt, the only time constraint now is to exit from the ISR before the present terminal count. This new "initial count" value will be stored in some table, let's say we store this value in the same table with the command sequence.

We can build a table as a sequence of two words: command and initial count, and perform always the same two operations inside the ISR:

- Increment the table pointer and send the command stored at the current table location
- Increment the table pointer and preset the initial count stored at the current table location

This scheme is not the most efficient in the case when a series of commands can be equally spaced in time and use the same initial count with no need to rewrite it. Moreover we have to disable/enable the LS_Task, depending on the interval time between the SS observation commands (HK are collected via LS_Task), as an example we might decide that every time the delay between two commands is greater than 10ms we want to enable LS_Task. So we have to build a table that is interpreted inside the ISR: every time an interrupt occurs a number of actions (table instructions beginning at the current pointer) is performed, the first one (time critical) being a command to SS and the following being some type of DPU internal commands.

Now we have come to a long table containing all the SS and DPU observation commands already somehow interpreted by an OBS routine (ISR_3). The first thing to note is that the commands are repeated in blocks as in a computer loop, so why not to add an DPU internal loop command to the table? Well to do so we must also define some local variable (register R[256]), then we could add other simple features like subroutine etc.

Ok we have come to a Virtual Machine implemented inside the ISR_3 routine.



3 The Virtual Machine

3.1 Critical instructions (CI)

The VM is used to send timely synchronized commands to the SS via the LS I/F, each command is transmitted when the HW down counter generates an interrupt. This SS commands are here defined as “critical” instructions (CI), each CI may be followed by a number of non CI which are executed during the same interrupt cycle. The DPU has just one LS I/F which must be used both by the VM and by the LS_Task for non time critical commands like HK request et al. In order to avoid collision on the LS I/F , a VM CI which lock the I/F has been introduced. This CI, which is effectively a mutual exclusive flag (mutex), must be executed at least 2 ms prior the use of the I/F by the VM in order to allow the termination of an HK request to a possible running LS_Task.

The last “dummy” CI is a no operation (NOP) instruction, to be used whenever a time gap must be introduced in the program. Typical use of NOP is before a READ instruction.

3.2 VM structure

The main components of the VM are:

- Program area
- VM-CPU clock
- Interpreter routine
- Local variable storage

Program area - This is a 32 bit words table (array of up to 32 Kword) containing the SS commands and local control instructions which forms a VM program. The table effectively represent the program/data memory area of the VM, with the table position (array index) acting as the program counter (PC). The table will contains a number of VM programs with associated tables of constants and subroutines. Each program is identified by the table position (array index) of the entry point.

VM-CPU clock - As mentioned the VM clock is generated by a down counter whose period is dynamically modifiable by the VM. This variable period clock, triggers an interrupt signal (IRQ3) which force the DPU CPU to execute the interpreter routine, thus executing a block of VM instructions.

Interpreter routine – The interpreter routine executes a block of VM instructions starting at the present PC up to (excluding) the next “critical” instruction CI (SS command, mutex or NOP instruction). So, for every VM - CPU clock, a block of instructions is executed, the first one (time critical) being a command to SS and the following being some type of DPU internal commands. This scheme effectively minimize the SS commands time jitter.

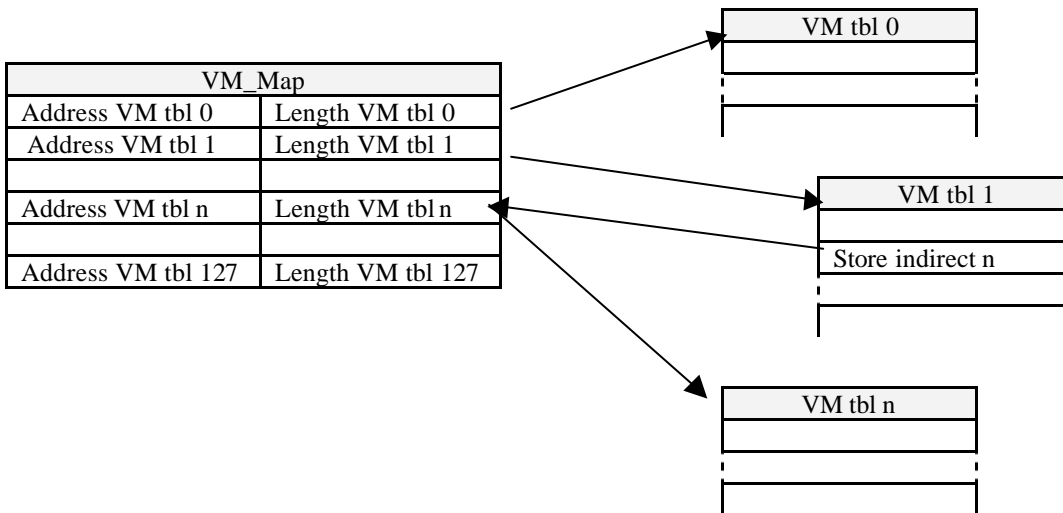
Local variable – In order to implement simple mathematical operations on SS commands, pass parameters to subroutine and keep track of “for” loops counts, a number of internal “global” registers are implemented. The 256 registers (R[0] ... R[255])¹ are statically defined inside the interpreter routine and are common to the stored VM programs.

¹ Register R[255] is also used as offset in the VM instructions ICPT and ICPF. R[254] may be used by the simulator to mimic the low speed READ port.



3.3 VM_Map Table

The DPU will contain a number (128 TBC) of VM tables (program area), each table with a maximum dimension of 32 Kwords may contain one or more VM program and may reside everywhere in the DPU data memory area. The physical address and the dimension of each VM table being stored in the 128 x 2 (TBC) **VM_Map** table. The VM program “scope” is the actual VM table, but using the “move/store indirect” or “call subroutine indirect” may also span (using the VM_Map) to the other tables.



3.4 VM Program

Each VM table has been divided in 3 sections:

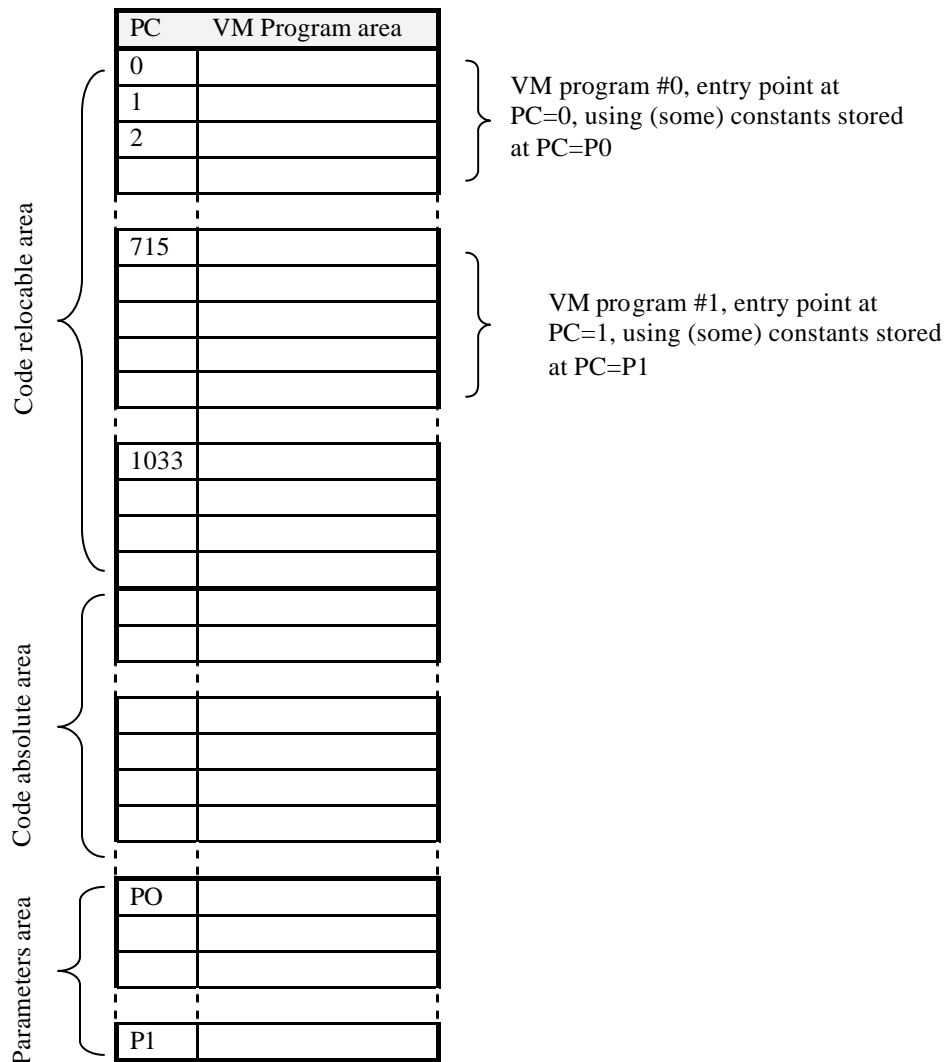
- Code relocable area
- Code absolute (library subroutine) area
- Parameters area

Code relocable area - In this area are stored the different VM programs, each program associated to an observation routine or time critical task. The programs here are completely relocable, to achieve this goal all “JUMP” instructions, with the exception of the “CALL SOUBROUTINE”, are relative.

Code absolute area - In this area are stored the “subroutine libraries” of the VM programs. As the “CALL SUBROUTINE” is implemented as a jump to an absolute address, VM instructions here coded are supposed to be relatively stables. Whenever the entry points of the library changes, the VM programs referring to the library must be updated.

It has to be noted that in this context the term absolute refer to the VM table (offset from the beginning of the table), so that each table can still be moved in the DPU memory with no modification to the VM code.

Parameters area - Each observation configuration/execution routine, store in a dedicated fixed portion of this area all the observation parameters.



A number of baseline VM programs , with functionality for the foreseen observation modes, will be stored on the DPU VM tables. These programs, stored in the VM memory area, may be modified/reloaded via TC, thus easing the need for OBS patching. The modification/addition is a simple table upload which can be performed via few TC packets to be compared to the lengthy and possibly dangerous OBS patching procedure. The compiler/simulator program described in the next chapter generates (also) the TC packet of the compiled VM program.



3.5 VM program exec TC

The VM program execution telecommand must indicate:

- I_map – index of the VM_Map table pointing to the VM table with the program
- I_prg – index in the program entry point area of the VM table with the address of the program
- N – number of run time parameters of the VM program to be stored in the first R[256] VM registers
- R[0] – first parameter
-
- R[n-1] – last parameter

3.6 VM Multitasking

In order to implement a VM multitask, SPIRE will use two types of VM: a “Real Time” VM and a “non-Real Time” VM.

The Real Time VM is the one just described which use the hardware down counter as “CPU clock”, the highest priority interrupt line (IRQ3) and direct access to the low speed interface via the lock mechanism (mutex). This VM may execute just one program at the time and is used in time critical tasks.

The second non-Real Time VM is the same as the Real Time one but use the Virtuoso OS sleep instruction to implement the “CPU clock”, and utilise the same LS_Task used by HK and normal commanding via a higher priority queue thus avoiding the lock mechanism. This second VM can execute different programs in a multitasking-like way utilising the multitask feature of Virtuoso OS.

The VM code for the non RT VMs is the same used by the RT one. In order to maintain full compatibility, the sleep time will refer to the time interval between the next critical instruction and the followings.



3.7 VM Instructions

A preliminary set of “VM assembler” instructions follows:

Instr. code (hex)	VM asm Mnemonic	Description	Code type
(7)	CMD	Send_Command(addr, code, val)² Send command code/val to SS addr	
0	RCMD	Send_Command_Reg(addr, code, reg)² Send command code/R[reg] to SS addr	3
4	RSND	Send_Reg_Command (reg) Send command R[reg] to SS	1
1	MTX	Mutex(OnOff)³ Lock/Unlock low speed I/F port	1
2	NOP	NOP() No operation	1
8	TIM	Set_Timer(val) ⁴ Set counter value (us) for next IRQ3	1
9	RTIM	Set_Timer(R[reg]) ⁴ Set counter value (us) for next IRQ3	1
A	READ	Read_HK_Reg(reg) Store received HK in R[reg] <i>For simulation purpose, data is read from an optional file or R[254] register (see chapter 4.1)</i>	1
10	RINC	Increment_Register(reg) R[reg] = R[reg] + 1	1
11	RDEC	Decrement_Register(reg) R[reg] = R[reg] - 1	1
12	RSET	Set_Register(reg, val32) ⁵ R[reg] = val32	1
13	RADD	Add_To_Reg(reg, val32) ⁵ R[reg] = R[reg] + val32	1
14	RSUB	Sub_To_Reg(reg, val32) ⁵ R[reg] = R[reg] - val32	1
15	RMUL	Multiply_To_Reg(reg, val32) ⁵ R[reg] = R[reg] * val32	1
16	RDIV	Divide_To_Reg(reg, val32) ⁵ R[reg] = R[reg] / val32	1
18	RAND	And(reg, val32) ⁵ R[reg] = R[reg] & val32	1
19	ROR	OR(reg, val32) ⁵ R[reg] = R[reg] val32	1
1A	RSHR	Reg_Shift_Right(reg, val) R[reg] >>= val	2
1B	RSHL	Reg_Shift_Left(reg, val) R[reg] <<= val	2
20	RREQ	Reg_Equate(reg1, reg2) R[reg1] = R[reg2]	2
21	RRAD	Add_Register_To_Register(r1, r2, r3) R[r1] = R[r2] + R[r3]	4
22	RRSB	Sub_Register_To_Register(r1, r2, r3) R[r1] = R[r2] - R[r3]	4
23	RRMP	Multiply_Register_To_Register(r1, r2, r3) R[r1] = R[r2] * R[r3]	4
24	RRDV	Divide_Register_To_Register(r1, r2, r3) R[r1] = R[r2] / R[r3]	4
30	JMPR	Jmp_Relative(vmAddr) PC = PC + vmAddr	1
31	RJPR	Jmp_Relative_Reg(reg) PC = PC + R[reg]	1
32	JPNZ	Jump_NZ(reg, vmAddr) If (R[reg] != 0) PC = PC + vmAddr	2
33	RSZ	Skip_Reg_Zero(reg) If (R[reg] == 0) PC = PC + 1	1
34	RSGT	Skip_Reg_GT(reg1, reg2) If (R[reg1] > R[reg2]) PC = PC + 1	2
35	RSLT	Skip_Reg_LT(reg1, reg2) If (R[reg1] < R[reg2]) PC = PC + 1	2
40	CALL	Call_Subr(vmAddr). Up to 16 nested subroutine. <i>PC = vmAddr and remember the present PC</i>	1
41	RET	Return() Return from subroutine	1
48	WRT	Write(reg) Write R[reg] to DPU frame/HK	1
49	RMOV	Move_To_Reg(reg, [vmAddr]) R[reg] = val32[vmAddr] <i>Copy the value stored at address vmAddr to R[reg]</i>	2

² Assuming all commands can be divided only in 3 fields. If this is not the case “code” disappear.

³ May be forced in the program, but the compiler insert automatically this instruction whenever is needed based on the optimisation level.

⁴ This time is the interrupt period valid after the next instruction. The minimum interrupt period is the maximum value between the time used by the I/F to transmit a command (100 us) and the actual duration of the ISR3. For the time being let’s fix it to 1 ms. This period is the minimum period between two SS commands

⁵ These instructions are coded as two consecutive 32 bit words, the second containing the plain value of “val32”. Do not put this opcode after a “skip” (RSZ, RSGT, RSLT) instruction.



Instr. code (hex)	VM asm Mnemonic	Description	Code type
4A	RRMV	Move_To_Reg(reg,[reg1]) R[reg]=val32[R[reg1]] <i>Copy the value stored at address R[reg1] to R[reg]</i>	2
4B	RSTO	Store_From_Reg(reg,[vmAddr]) val32[vmAddr]= R[reg] <i>Copy the value stored in R[reg] at address vmAddr</i>	2
4C	RRST	Store_From_Reg(reg,[reg1]) val32[R[reg1]] = R[reg] <i>Copy the value stored in R[reg] at address R[reg1]</i>	2
50	TER13	Send_TC_ExecPkt_13() <i>Send telecommand execution packet 1,3</i>	1
51	TER15	Send_TC_ExecPkt_15(stepNo) <i>Send telecommand execution packet 1,5 with stepNo</i>	1
52	TER17	Send_TC_ExecPkt_17() <i>Send telecommand execution packet 1,7</i>	1
53	EVNT	Send_Event(reg, EventNo) <i>Send event=EventNo with parameter R[reg]</i>	2
54	TXTBL	Transmit_Table (VM_Map_Idx) <i>Signal to the OBS to transmit a TM frame. The data is stored at address VM_Map[VM_Map_Idx]. The first word is the TM frame length (set to zero by OBS when the operation is completed)</i>	1
Indirect instructions via VM_Map table			
60	ICALL	Call_Subr(VM_Map_Idx, Offset) ⁶ . Up to 16 nested subroutine. <i>Call subroutine at address specified in VM_Map[VM_MapIdx] plus Offset</i>	2
61	ICPT	Copy_To_ExtMem(VM_Map_Idx, [reg], n) <i>Copy n (<256) words from local address R[reg] to external address specified in VM_Map[VM_Map_Idx] plus offset defined by R[255]</i>	4
62	ICPF	Copy_From_ExtMem(VM_Map_Idx, [reg], n) <i>Copy n (<256) words from external address specified in VM_Map[VM_Map_Idx] plus offset defined by R[255] to local address R[reg]</i>	4
63	IRCALL	Call_Subr([reg], Offset) ⁷ . Up to 16 nested subroutine. <i>Call subroutine at address specified in VM_Map[R[reg]] plus Offset</i>	2
64	IRCPT	Copy_To_ExtMem([reg1], [reg2], n) <i>Copy n (<256) words from local address R[reg2] to external address specified in VM_Map[R[reg1]] plus offset defined by R[255]</i>	4
65	IRCPF	Copy_From_ExtMem([reg1], [reg2], n) <i>Copy n (<256) words from external address specified in VM_Map[R[reg1]] plus offset defined by R[255] to local address R[reg2]</i>	4
80	END	End End current VM program	1
Pseudo instructions			
	INC	Include source file (up to 3 nested INC)	
	EQU	Store at the current address the constant parameter	
	DEF	Set constants	
	ORG	Address of code	
	TABLE	Table(n). The parameter n<128, must be numeric. <i>This instruction forces the compilation unit to be stored in VM_Tbl=n</i>	
	_Label	Label referred by loop/jmp	
Debug instructions			
	COM	COM text string Comment printed during the simulation	
	ROUT	ROUT 0, 4, 72 Print contents of R[0], R[4], R[72] during simulation	

It has to be noted that in order to make the VM program as relocable as possible inside its VM table, all jump instructions, with the exclusion of the Call Sub and indirect instructions, are relative to the PC.

⁶ The "Offset" value must be resolved in the current compilation unit (VM table).

⁷ The "Offset" value must be resolved in the current compilation unit (VM table).



The table notation is:

Val 16 or 24 bit numeric constant possibly defined in a DEF statement.

Val32 32 bit numeric constant.

Reg VM internal registers index. Numeric constant between 0 and 255 possibly defined in a DEF statement.

VmAddr Signed 16 bit numeric constant indicating the relative address displacement in a Jump instruction. It may be coded as a `_label` mnemonic, in this case the relative address displacement is computed by the compiler.

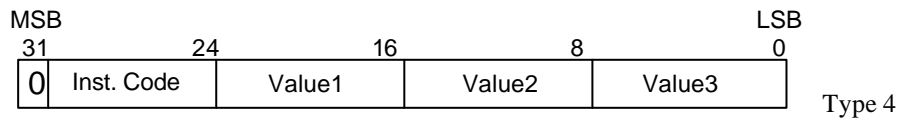
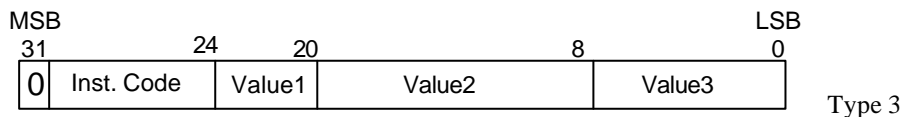
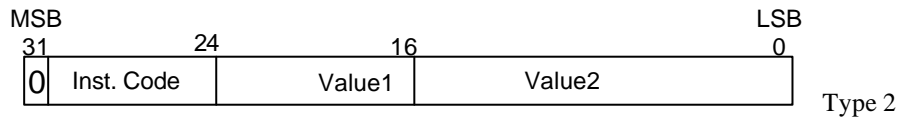
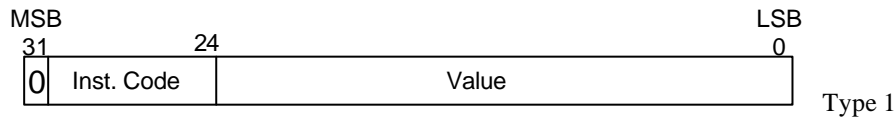
3.8 Instructions Format

The present instruction coding is as follows:

1. First (MSB) bit=1 then it is a plain command to the SS, as the first bit (start bit) is always set. Here we assume that the data content of the command can be splitted in two fields (code and value). The MSBit of addr field indicate cmd/hk request.



2. First (MSB) bit=0 then it is a coded 32 bit instruction with:



A VM assembler compiler/simulator program is provided in order to simplify the on ground coding of the observation programs.



4 VM Compiler/Simulator

4.1 Compiler

The compiler resolve all the mnemonic labels and constant in a VM program and produce the absolute VM code. The compiler optimiser try also to take care of the MTX instructions which enable/disable the low speed I/F usage by the LS_Task.

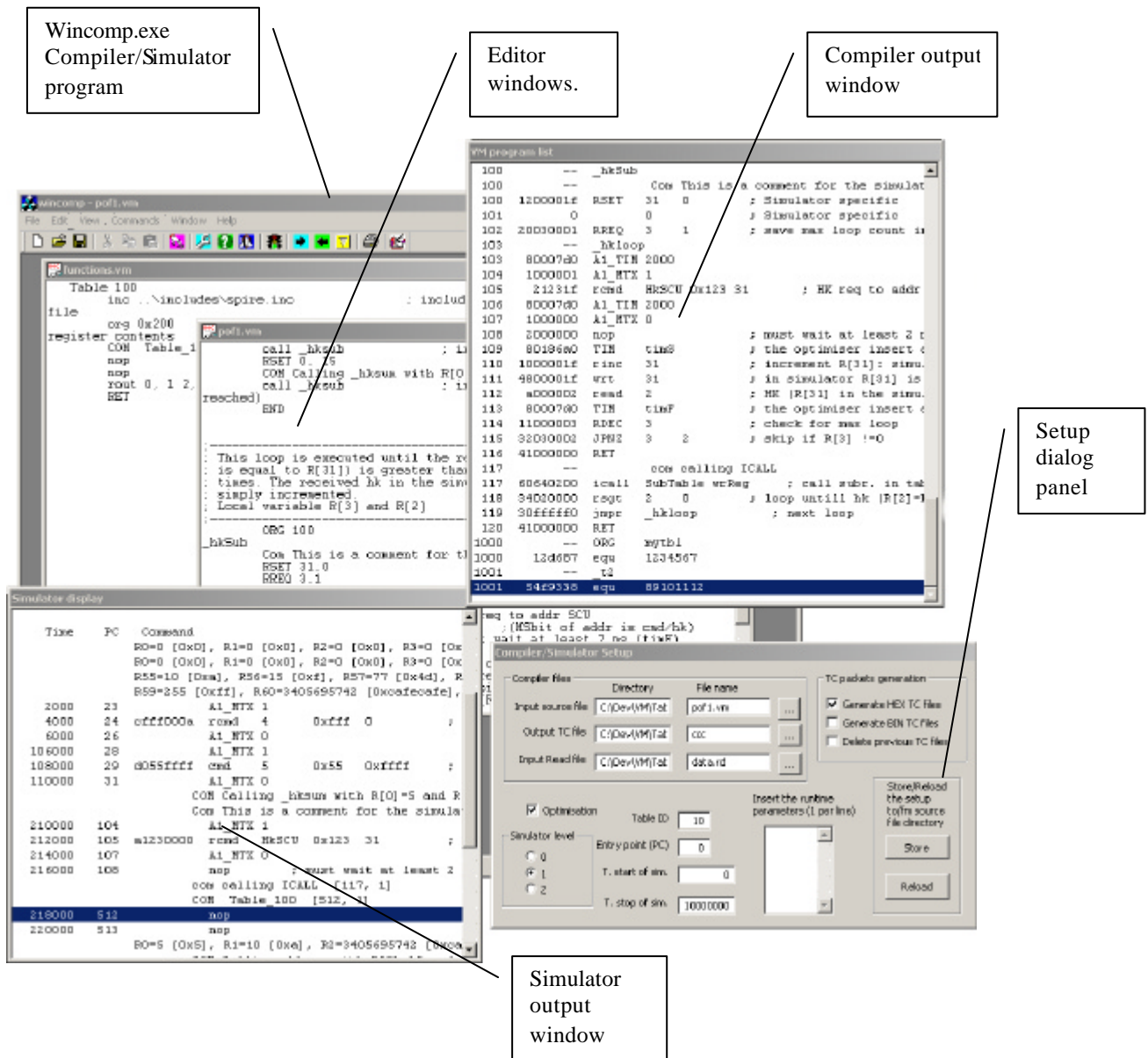
The VM program syntax is:

- Code is case insensitive.
- Hexadecimal constants are prefixed with 0x
- Comments begin with “;” and can appear also after an instruction.
- Labels begin with “_”.

The compiler/simulator program consists of a MDI simple editor, a dialog box used to set the program parameters and two list windows with the compiler and simulator output.

The program should run on every Win98, WinNT, Win2000, WinXp computer.

The figure below shows the compiler/simulator program





The following figure shows/describes few details of the setup panel and toolbar.

Compiler/Simulator Setup

Compiler files

Directory	File name
Input source file: C:\Dev\VM\Tab	pof1.vm
Output TC file: C:\Dev\VM\Tab	ccc
Input Read file: C:\Dev\VM\Tab	data.rd

TC packets generation

- Generate HEX TC files
- Generate BIN TC files
- Delete previous TC files

Optimisation

Table ID: 10

Simulator level

- 0
- 1
- 2

Entry point (PC): 0

T. start of sim.: 0

T. stop of sim.: 10000000

Insert the runtime parameters (1 per line)

Store/Reload the setup to/from source file directory

Store

Reload

wincomp - pof1.vm

File Edit View Commands Window Help

Toolbar icons: File, Edit, View, Commands, Window, Help, New, Open, Save, Print, Undo, Redo, Run, Stop, Help, Refresh, Clear, Compile, Simulate, Generate TC, Hide/Show Windows, Clear Compiler Output, Clear Simulator Output, Clear Both, Compile and Simulate.

Callout Boxes:

- Parameters copied to VM registers beginning from R[0]**: Points to the 'Insert the runtime parameters' text box.
- The input program directory and file name may be also forced to the file in the active editor window, with the pink toolbar button.**: Points to the 'Input source file' field and the pink 'Run' icon in the toolbar.
- The setup in this panel may be saved/restored in the source file directory. Starting the program, the setup stored in the last visited directory is loaded. Ending the program, the actual setup is stored in the current source file directory.**: Points to the 'Store' and 'Reload' buttons.
- Compile the program named in the setup panel and store the "binary" output on the specified table (Table ID). The output list is appended in the compiler output window**: Points to the 'Compile' icon in the toolbar.
- Simulate the program stored in the specified table beginning at offset "Entry point (PC)". The optional "n" run time parameters are loaded in the VM internal registers R[0]... R[n-1]. The output list, defined by the "Simulator level", is appended in the simulator output window.**: Points to the 'Simulate' icon in the toolbar.
- Generate the telecommands (TC) packets in the specified directory**: Points to the 'Generate TC' icon in the toolbar.
- Hide/Shows the other windows**: Points to the 'Hide/Show Windows' icon in the toolbar.
- Clear the simulator output window**: Points to the 'Clear Simulator Output' icon in the toolbar.
- Clear the compiler output window**: Points to the 'Clear Compiler Output' icon in the toolbar.
- Clear the compiler and simulator output windows. Compile, simulate and generate the TC packets.**: Points to the 'Clear Both' icon in the toolbar.



The input files of the program are:

Filename.vm, ... Filenamex.vm: source program files.

DataFile.rd: optional file with input data to the READ instruction. If the file exists, each READ instruction found during the simulation, read a new number from that file. If the file is not present or contains less numbers than then READ instruction, the contents of register R[254] is used instead.

This is an example of formats allowed in the read data file.

```
;----- READ data file-----  
; Comments begin with # or ;  
; Number in decimal or "c-hex" format: (0xff)  
; More then a number per each line,  
; number separator are: space commas  
;-----  
0xa 15 ; My comment  
  
77, 0xcafecafe,,12  
;  
0xff ;comment
```

The output files of the program are:

In the same directory as the input source files (specified in the dialog box)

Filename.lst: This file list the compiled program. The file name is the same as the input program filename with extension ".lst".

Filename.sim: This file list the simulator output. The file name is the same as the input program filename with extension ".sim".

In the output directory specified in the dialog box:

outfilnam0.txt, outfilnam .txt outfilnamn.txt: files with the TC packets of the compiled VM program in Hex format

outfilnam0.bin, outfilnam1.bin outfilnamn.bin: files with the TC packets of the compiled VM program in binary format

The compiler optimisation level 1 check for any "unprotected" (MTX=0) CMD/RCMD instruction, and protect the command with a double TIM -MTX couple using the following criteria:

If exist a CMD/RCMD instruction while MTX=0 and TIM=oldtim

Then modify to:

```
TIM 2000      (1 ms is chosen as the minimum TIM value)  
MTX 1  
CMD/RCMD xxx (original instruction)  
TIM oldtim  
MTX 0
```



The TIM-MTX instructions inserted by the optimiser are prefixed by A1_ (TIM-MTX).

Example:

No optimisation	Optimisation level 1
MTX 0	MTX 0
TIM 30000	TIM 30000
...	
	A1_MTX 1
	A1_TIM 2000
	CMD aaa
	A1_MTX 0
CMD aaa	A1_TIM 30000
CMD ccc	A1_MTX 1
TIM 100000	A1_TIM 2000
...	CMD ccc
...	A1_MTX 0
...	A1_TIM 30000
	TIM 100000



4.2 VM Simulator

The simulator section of the compiler program, is a modified version of the OBS VM section. The simulator control any “unprotected” CMD/RCMD instruction and output (on the out list file) a timeline of the SS commands. Two VM program instructions are interpreted only by the simulator:

Comment instructions:

COM comment string
inserted in the input program, are listed by the simulator as:
COM comment string [addr,n]

Display internal register

ROUT n1 n2 ... nx
Display on the simulator out list the value of register R[n1], R[n2],... R[nx]. The following instruction
rout 0, 1 2,31
encountered at address 126 for the 6th time, generate on the simulator list:
R0=5 [0x5], R1=10 [0xa], R2=0 [0x0], R31=6 [0x6], [126, 6]

with addr = address of the next instruction
n = auto incrementing number counting # of occurrence.

The simulator output file format is controlled by the run time switch s0-2 (radio button objects on the dialog window):
s0 -> (default value) only command to SS (in hex) are listed with relative time and PC for each CMD RCMD MTX NOP instructions.

Time	PC	Command
2000	4	
4000	5	cfff000a
6000	7	
106000	9	

s1 -> as for s0 but the input text and comment for the above command is also shown. If a WRT instruction is encountered, the content of the addressed register is also shown.

Time	PC	Command
		R[0]=10
2000	4	A_MTX 1
4000	5	cfff000a rcmd 4 0xffff 0 ; command to addr 0 (MSbit of addr
6000	7	A_MTX 0
106000	9	A_MTX 1

s2 -> as for s1 but the input text and comment for all instructions is also shown.

Time	PC	Command
0	0	TIM timS ; the optimiser 01 switch insert a MXT
0	1	RSET 0 r0val ; R[0]=0xa
0	3	wrt 0
		R[0]=10
0	4	A_TIM 2000
2000	5	A_MTX 1
4000	6	cfff000a rcmd 4 0xffff 0 ; command to addr 0 (MSbit of
4000	7	A_TIM 100000
6000	8	A_MTX 0
6000	9	A_TIM 2000
106000	10	A_MTX 1



4.3 Packetiser

At the end of the compiler and simulation phase, two group of TC packet files (one file per packet) are generated. The packet file format is: big endian 16 bit words hexadecimal (*.txt) and binary (*.bin), each group of files has the order number included in the name.

Example:

```
vmTC_0.txt
vmTC_1.txt
vmTC_2.txt
vmTC_0.bin
vmTC_1.bin
vmTC_2.bin
vmTbl.hex
```

The packet structure is as follows:

```
----- PACKET HEADER (48 bits) -----
w16_0=   Packet ID
         Version Number (3)
         Type (1)
         Data field header (1)
         PID (7)
         PCAT (4)
w16_1=   Packet Sequence control
         Sequence flag (2)
         Sequence count (14)
w16_2=   Packet Length = (Number of octets in Packet Data Field) - 1
----- Packet data field -----
w16_3=   Packet Data field header
         PUS (3)
         Checksum type (1)
         ACK (4)
         Pkt Type (8)           = 8
w16_4=   Packet Data field header
         Pkt SubType (8)       = 4
         Pad (8)
w16_5=   Application data
.....
w16_n=   Application data
w16_n+1=CRC (16) of full packet

===== SPIRE APP DATA =====
w16_5=   Function ID (8 MSB)
         Activity ID (8 LSB)
w16_6=   Table ID
w16_7=   Offset from beg of table (8 MSB)
w16_8=   N. of 32 bit words data items
w16_9=   Data
.....
w16_n = CRC
```

In the same directory the file **vmTbl.txt** is generated. This hex file contains the VM program code to be included (as initial program) on the OBS at compile time.



4.4 Directory structure

The compiler executable needs always the files:

```
wincomp.exe           // executable
spiresyntax.h        // implemented instructions
```

in the same directory. In the dialog box must be specified the directory, absolute or relative to the compiler program, of the source files and TC packet files. The include files are always in the same directory as the source files, the generated output list and simulator file will be generated in the same directory with the same name of the source file and extension .lst and .sim..

Here follows the directory structure utilized for the compilation of the program in figure in paragraph 4.1 (input files are underlined).

Directory of C:\VM_Comp

```
02/04/2002  12:32      <DIR>      ..
02/04/2002  12:32      <DIR>      .
26/04/2002  12:00      <DIR>      TC_SPkt
26/04/2002  11:44      <DIR>      VM_SProg
23/04/2002  14:28                4,143 spiresyntax.h
26/04/2002  11:43                598,016 wincomp.exe
          2 File(s)
```

Directory of C:\VM_Comp\VM_SProg

```
26/04/2002  11:44      <DIR>      ..
26/04/2002  11:44      <DIR>      .
25/04/2002  13:56                2,714 spire.vm
25/04/2002  13:56                1,580 spire.inc
26/04/2002  12:00                30,202 spire.lst
24/04/2002  13:04                1,418 spire.sim
          4 File(s)
```

Directory of C:\VM_Comp\TC_SPkt (if it doesn't exists, this directory is automatically generated)

```
02/05/2002  13:02      <DIR>      .
02/05/2002  13:02      <DIR>      ..
02/05/2002  13:02                721 vmTbl.hex
02/05/2002  13:02                109 vmTC_0.txt
02/05/2002  13:02                589 vmTC_1.txt
02/05/2002  13:02                97 vmTC_2.txt
02/05/2002  13:02                36 vmTC_0.bin
02/05/2002  13:02                196 vmTC_1.bin
02/05/2002  13:02                32 vmTC_2.bin
          7 File(s)          1,780 bytes
```



5 Example

As an example of the a VM program, here is the implementation of the “Total Power” measurement in HIFI. The measuring routine is based on the following algorithm:

	Control command	LS command	VM code	comment
1.	Set Mutex			
2.	hrsloop=0			
3.	WBS count =10			this is HIF_N_PERIODS
4.	Label WBS loop			
5.		Reset WBS-H Reset WBS-V	E400-0009 E800-0009	
6.	Timer = ???			HIF_T_DEL_WBS ?
7.		Start WBS H&V	FC00 003	
8.	HRS count =8			this is HIF_R_HRS
9.	Label HRS loop			
10.	hrsbuf = (hrsbuf + 1) mod 2			
11.		Select HRS-H buffer	hrsbuf=0: D700 0000 hrsbuf=1: D710 0000	
12.		Select HRS-V buffer	hrsbuf=0: DB00 0000 hrsbuf=1: DB10 0000	
13.		Start HRS H&V	FF80 0000	
14.	timer = 100 ms			this is HIF_T_ACC_HRS
15.	Reset Mutex			Time for HK-collection
16.	timer = 0.2 ms			
17.	Set Mutex			
18.		Stop HRS H&V	FF90 0000	
19.		Start Transfer HRS-H	D740 0000	
20.		Start Transfer HRS-V	DB40 0000	
21.		Reset readout buffer H	hrsbuf=0: D730 0000 hrsbuf=1: D720 0000	
22.		Reset readout buffer V	hrsbuf=0: DB30 0000 hrsbuf=1: DB20 0000	
23.	Decrement HRS count			
24.	conditional Jump to HRS loop			
25.		Stop WBS H&V	FC00 0005	
26.		Start Transfer WBS-H	E400 0006	
27.		Start Transfer WBS-V	E800 0006	
28.	Decrement WBS count			
29.	conditional Jump to WBS loop			
30.	Reset Mutex			



Here follows the VM program source of the HIFI total power measurement:

```
-----  
; Case insensitive  
; Comments begin with a ;  
; Labels begin with an _  
-----  
;  
; HIFI - Total Power  
; Ver 1.1  
-----  
                INC hifi.inc                ; include file with constant's definition for HIFI  
  
                DEF lock 1                  ; for mutex. Lock the LS I/F  
                DEF unlock 0                ; for mutex. release the LS I/F  
                DEF slow 100000             ; 100 ms timer  
                DEF fast 2000               ; 2 ms timer  
  
                ORG EntryPointTbl            ; begin of programs entry points table  
                EQU 8                       ; TotPow begin at 8  
                EQU 512                      ; next program  
                EQU 1024                     ; next program  
  
                ORG 8                       ; address of main program  
                TIM fast                    ; timer period at 2 ms  
                MTX lock                    ; lock LS I/F  
                RMOV 0 wbCnt                 ; R[0]=10. WBS loop counter  
                RSET 2 _c2                   ; in R[2] last address of table  
                RREQ 3 2                     ; in R[3] last address of table  
  
                CMD wb_h,rst_wb              ; reset WBS_H  
                CMD wb_v,rst_wb              ; reset WBS_V  
_wbLoopCMD      br,bstr_wb                  ; start WBS H&V  
                RMOV 1 hrCnt                 ; R[1]=8. HRS loop counter  
  
_hrLoop        RINC 3                      ; increment R[3]  
                RSGT 3 2                     ; Skip next instr if R[3] > R[2]  
                JMPR _intbl                  ; Skip next opcode (2 instruction code)  
                RSET 3 _c1                   ; R[3]= address of begin of table  
_intbl         RRMV 4 3                     ; move value stored at address=R[3] in R[4]  
                RCMD hr_h, 4                 ; select HRS_H buffer  
                RCMD hr_v, 4                 ; select HRS_V buffer  
                CMD br, bstr_hr              ; start HRS  
                TIM slow                     ; wait 100 ms  
                MTX unlock                    ; release SL I/F  
                TIM fast                     ; timer period at 2 ms  
                MTX lock                     ; lock SL I/F  
                CMD br, bstp_hr              ; stop HRS  
                CMD hr_h, stt_hr             ; start transfer HRS_H  
                CMD hr_v, stt_hr             ; start transfer HRS_V  
                RINC,3                       ; increment R[3]  
                RRMV 4 3                     ; move value stored at address=R[3] in R[4]  
                RCMD hr_h, 4                 ; reset HRS_H buffer  
                RCMD hr_v, 4                 ; reset HRS_V buffer  
                RDEC 1                       ; decrement HRS loop counter  
                JPNZ 1, _hrLoop              ; if R[1]>0 go to _hrLoop  
  
                CMD br, bstp_wb              ; stop WBS  
                CMD wb_h, stt_wb             ; start transfer WBS_H  
                CMD wb_v, stt_wb             ; start transfer WBS_V  
                RDEC 0                       ; decrement WBS loop counter  
                JPNZ 0, _wbLoop              ; if R[0]>0 go to _wbLoop  
                MTX unlock                    ; release SL I/F  
                END  
  
_c1            EQU sel_hrb0  
                EQU rst_hrb0  
                EQU sel_hrb1  
_c2            EQU rst_hrb1  
  
----- Parameters area -----  
; Here I store the program parameters. May be changed by TC.  
; This section can be omitted, the parameters are stored  
; by the OBS on reception of "configure/start measure" TC
```



ORG wbCnt
EQU 10
ORG hrCnt
EQU 8

and the hifi.inc definitions file

```
-----  
; Include file with constants definitions  
; up to 3 deep nested include files  
-----  
;  
; HIFI definitions  
-----  
  
;----- VM Program area definition -----  
DEF EntryPointTbl 0 ; begin of VM programs entry points table  
DEF ParArea0 4096 ; begin of TotPower parameter area  
DEF wbCnt 4096 ; location of WBS loop for Tot Pow  
DEF hrCnt 4097 ; location of HRS loop for Tot Pow  
  
;----- Subsystems address -----  
DEF LSU, 0  
DEF FCU, 3  
DEF HR_H 5  
DEF HR_V, 6  
DEF WB_H, 9  
DEF WB_V, 0xA  
DEF LCU, 0xC  
DEF BR, 0xF ; Broadcast address  
  
; WBS definition (Val26)  
DEF BSTR_WB 3 ; Broadcast Start WBS H&V  
DEF BSTP_WB 5 ; Broadcast Stop WBS H&V  
DEF RST_WB 9 ; reset WBS  
DEF STT_WB 6 ; Start transfer WBS  
  
; HRS definition (Val26)  
DEF BSTR_HR 0x3800000 ; Broadcast start HRS H&V  
DEF BSTP_HR 0x3900000 ; Broadcast stop HRS H&V  
DEF STT_HR 0x3400000 ; Start transfer HRS  
DEF SEL_HRB1 0x3100000 ; HRS select buffer 1  
DEF SEL_HRB0 0x3000000 ; HRS select buffer 0  
DEF SEL_HRB1 0x3100000 ; HRS select buffer 1  
DEF RST_HRB0 0x3300000 ; reset readout buffer 0  
DEF RST_HRB1 0x3200000 ; reset readout buffer 1  
  
; Chopper definition (FCU)  
DEF CHOP_0 0x3105555 ; FCU Chopper pos 0  
DEF CHOP_1 0x310AAAA ; FCU Chopper pos 1  
DEF CHOP_2 0x310AAAA ; FCU Chopper pos 2  
DEF CHOP_3 0x3105555 ; FCU Chopper pos 3
```



Here is the compiler output list file (comments manually tabulated for this document):

```
VM program file:      VM_HProg\hifitotpow.vm
Compilation time:    Thu May 02 13:02:29 2002
Optimisation level= 1
Simulation level = 1
Start address (PC)= 8
```

Addr	opCode	Instruction	
0		INC hifi.inc	; include file with constant's definition for HIFI
0		DEF EntryPointTbl 0	; begin of VM programs entry points table
0		DEF ParArea0 4096	; begin of TotPower parameter area
0		DEF wbCnt 4096	; location of WBS loop for Tot Pow
0		DEF hrCnt 4097	; location of HRS loop for Tot Pow
0		DEF LSU 0	
0		DEF FCU 3	
0		DEF HR_H 5	
0		DEF HR_V 6	
0		DEF WB_H 9	
0		DEF WB_V 0xA	
0		DEF LCU 0xC	
0		DEF BR 0xF	; Broadcast address
0		DEF BSTR_WB 3	; Broadcast Start WBS H&V
0		DEF BSTP_WB 5	; Broadcast Start WBS H&V
0		DEF RST_WB 9	; reset WBS
0		DEF STT_WB 6	; Start transfer WBS
0		DEF BSTR_HR 0x3800000	; Broadcast start HRS H&V
0		DEF BSTP_HR 0x3900000	; Broadcast stop HRS H&V
0		DEF STT_HR 0x3400000	; Start transfer HRS
0		DEF SEL_HRB1 0x3100000	; HRS select buffer 1
0		DEF SEL_HRB0 0x3000000	; HRS select buffer 0
0		DEF SEL_HRB1 0x3100000	; HRS select buffer 1
0		DEF RST_HRB0 0x3300000	; reset readout buffer 0
0		DEF RST_HRB1 0x3200000	; reset readout buffer 1
0		DEF CHOP_0 0x3105555	; FCU Chopper pos 0
0		DEF CHOP_1 0x310AAAA	; FCU Chopper pos 1
0		DEF CHOP_2 0x310AAAA	; FCU Chopper pos 2
0		DEF CHOP_3 0x3105555	; FCU Chopper pos 3
0		DEF lock 1	; for mutex. Lock the LS I/F
0		DEF unlock 0	; for mutex. release the LS I/F
0		DEF slow 100000	; 100 ms timer
0		DEF fast 2000	; 2 ms timer
0		ORG EntryPointTbl	; begin of programs entry points table
0	8	EQU 8	; TotPow begin at 8
1	200	EQU 512	; next program
2	400	EQU 1024	; next program
8		ORG 8	; address of main program
8	80007d0	TIM fast	; timer period at 2 ms
9	1000001	MTX lock	; lock LS I/F
10	49001000	RMOV 0 wbCnt	; R[0]=10. WBS loop counter
11	12000002	RSET 2 _c2	; in R[2] last address of table
12	32	_c2	; in R[2] last address of table
13	20030002	RREQ 3 2	; in R[3] last address of table
14	e4000009	CMD wb_h rst_wb	; reset WBS_H
15	e8000009	CMD wb_v rst_wb	; reset WBS_V
16		_wbLoop	; start WBS H&V
16	fc000003	CMD br bstr_wb	; start WBS H&V
17	49011001	RMOV 1 hrCnt	; R[1]=8. HRS loop counter
18		_hrLoop	; increment R[3]
18	10000003	RINC 3	; increment R[3]
19	34030002	RSGT 3 2	; Skip next instr if R[3] > R[2]
20	30000003	JMPR _intbl	; Skip next opcode (2 instruction code)
21	12000003	RSET 3 _c1	; R[3]= address of begin of table
22	2f	_c1	; R[3]= address of begin of table
23		_intbl	; move value stored at address=R[3] in R[4]
23	4a040003	RRMV 4 3	; move value stored at address=R[3] in R[4]
24	500004	RCMD hr_h 4	; select HRS_H buffer
25	600004	RCMD hr_v 4	; select HRS_V buffer
26	ff800000	CMD br bstr_hr	; start HRS
27	80186a0	TIM slow	; wait 100 ms
28	1000000	MTX unlock	; release SL I/F
29	80007d0	TIM fast	; timer period at 2 ms
30	1000001	MTX lock	; lock SL I/F



```

31 ff900000 CMD br bstp_hr ; stop HRS
32 d7400000 CMD hr_h stt_hr ; start transfer HRS_H
33 db400000 CMD hr_v stt_hr ; start transfer HRS_V
34 10000003 RINC 3 ; increment R[3]
35 4a040003 RRMV 4 3 ; move value stored at address=R[3] in R[4]
36 500004 RCMD hr_h 4 ; reset HRS_H buffer
37 600004 RCMD hr_v 4 ; reset HRS_V buffer
38 11000001 RDEC 1 ; decrement HRS loop counter
39 3201ffeb JPNZ 1 _hrLoop ; if R[1]>0 go to _hrLoop
40 fc000005 CMD br bstp_wb ; stop WBS
41 e4000006 CMD wb_h stt_wb ; start transfer WBS_H
42 e8000006 CMD wb_v stt_wb ; start transfer WBS_V
43 11000000 RDEC 0 ; decrement WBS loop counter
44 3200ffe4 JPNZ 0 _wbLoop ; if R[0]>0 go to _wbLoop
45 10000000 MTX unlock ; release SL I/F
46 50000000 END
47 _c1
47 3000000 EQU sel_hrb0
48 3300000 EQU rst_hrb0
49 3100000 EQU sel_hrb1
50 _c2
50 3200000 EQU rst_hrb1
4096 ORG wbCnt
4096 a EQU 10
4097 ORG hrCnt
4097 8 EQU 8

```

Here is the simulator output list file (comments manually tabulated for this document):

Begin simulation from t1= 0 up to t2= 1000000

```

Time PC Command
2000 9 MTX lock ; lock LS I/F
4000 14 e4000009 CMD wb_h rst_wb ; reset WBS_H
6000 15 e8000009 CMD wb_v rst_wb ; reset WBS_V
8000 16 fc000003 CMD br bstr_wb ; start WBS H&V
10000 24 d7000000 RCMD hr_h 4 ; select HRS_H buffer
12000 25 db000000 RCMD hr_v 4 ; select HRS_V buffer
14000 26 ff800000 CMD br bstr_hr ; start HRS
16000 28 MTX unlock ; release SL I/F
116000 30 MTX lock ; lock SL I/F
118000 31 ff900000 CMD br bstp_hr ; stop HRS
120000 32 d7400000 CMD hr_h stt_hr ; start transfer HRS_H
122000 33 db400000 CMD hr_v stt_hr ; start transfer HRS_V
124000 36 d7300000 RCMD hr_h 4 ; reset HRS_H buffer
126000 37 db300000 RCMD hr_v 4 ; reset HRS_V buffer
128000 24 d7100000 RCMD hr_h 4 ; select HRS_H buffer
130000 25 db100000 RCMD hr_v 4 ; select HRS_V buffer
132000 26 ff800000 CMD br bstr_hr ; start HRS
134000 28 MTX unlock ; release SL I/F
234000 30 MTX lock ; lock SL I/F
236000 31 ff900000 CMD br bstp_hr ; stop HRS
238000 32 d7400000 CMD hr_h stt_hr ; start transfer HRS_H
240000 33 db400000 CMD hr_v stt_hr ; start transfer HRS_V
242000 36 d7200000 RCMD hr_h 4 ; reset HRS_H buffer
244000 37 db200000 RCMD hr_v 4 ; reset HRS_V buffer
246000 24 d7000000 RCMD hr_h 4 ; select HRS_H buffer
248000 25 db000000 RCMD hr_v 4 ; select HRS_V buffer
250000 26 ff800000 CMD br bstr_hr ; start HRS
252000 28 MTX unlock ; release SL I/F
352000 30 MTX lock ; lock SL I/F
354000 31 ff900000 CMD br bstp_hr ; stop HRS
356000 32 d7400000 CMD hr_h stt_hr ; start transfer HRS_H
358000 33 db400000 CMD hr_v stt_hr ; start transfer HRS_V
360000 36 d7300000 RCMD hr_h 4 ; reset HRS_H buffer
362000 37 db300000 RCMD hr_v 4 ; reset HRS_V buffer
364000 24 d7100000 RCMD hr_h 4 ; select HRS_H buffer
366000 25 db100000 RCMD hr_v 4 ; select HRS_V buffer
368000 26 ff800000 CMD br bstr_hr ; start HRS
370000 28 MTX unlock ; release SL I/F
470000 30 MTX lock ; lock SL I/F
472000 31 ff900000 CMD br bstp_hr ; stop HRS
474000 32 d7400000 CMD hr_h stt_hr ; start transfer HRS_H
476000 33 db400000 CMD hr_v stt_hr ; start transfer HRS_V
478000 36 d7200000 RCMD hr_h 4 ; reset HRS_H buffer

```




```
480000 37 db200000 RCMD hr_v 4 ; reset HRS_V buffer
482000 24 d7000000 RCMD hr_h 4 ; select HRS_H buffer
484000 25 db000000 RCMD hr_v 4 ; select HRS_V buffer
486000 26 ff800000 CMD br bstr_hr ; start HRS
488000 28 MTX unlock ; release SL I/F
588000 30 MTX lock ; lock SL I/F
590000 31 ff900000 CMD br bstp_hr ; stop HRS
592000 32 d7400000 CMD hr_h stt_hr ; start transfer HRS_H
594000 33 db400000 CMD hr_v stt_hr ; start transfer HRS_V
596000 36 d7300000 RCMD hr_h 4 ; reset HRS_H buffer
598000 37 db300000 RCMD hr_v 4 ; reset HRS_V buffer
600000 24 d7100000 RCMD hr_h 4 ; select HRS_H buffer
602000 25 db100000 RCMD hr_v 4 ; select HRS_V buffer
604000 26 ff800000 CMD br bstr_hr ; start HRS
606000 28 MTX unlock ; release SL I/F
706000 30 MTX lock ; lock SL I/F
708000 31 ff900000 CMD br bstp_hr ; stop HRS
710000 32 d7400000 CMD hr_h stt_hr ; start transfer HRS_H
712000 33 db400000 CMD hr_v stt_hr ; start transfer HRS_V
714000 36 d7200000 RCMD hr_h 4 ; reset HRS_H buffer
716000 37 db200000 RCMD hr_v 4 ; reset HRS_V buffer
718000 24 d7000000 RCMD hr_h 4 ; select HRS_H buffer
720000 25 db000000 RCMD hr_v 4 ; select HRS_V buffer
722000 26 ff800000 CMD br bstr_hr ; start HRS
724000 28 MTX unlock ; release SL I/F
824000 30 MTX lock ; lock SL I/F
826000 31 ff900000 CMD br bstp_hr ; stop HRS
828000 32 d7400000 CMD hr_h stt_hr ; start transfer HRS_H
830000 33 db400000 CMD hr_v stt_hr ; start transfer HRS_V
832000 36 d7300000 RCMD hr_h 4 ; reset HRS_H buffer
834000 37 db300000 RCMD hr_v 4 ; reset HRS_V buffer
836000 24 d7100000 RCMD hr_h 4 ; select HRS_H buffer
838000 25 db100000 RCMD hr_v 4 ; select HRS_V buffer
840000 26 ff800000 CMD br bstr_hr ; start HRS
842000 28 MTX unlock ; release SL I/F
942000 30 MTX lock ; lock SL I/F
944000 31 ff900000 CMD br bstp_hr ; stop HRS
946000 32 d7400000 CMD hr_h stt_hr ; start transfer HRS_H
948000 33 db400000 CMD hr_v stt_hr ; start transfer HRS_V
950000 36 d7200000 RCMD hr_h 4 ; reset HRS_H buffer
952000 37 db200000 RCMD hr_v 4 ; reset HRS_V buffer
954000 40 fc000005 CMD br bstp_wb ; stop WBS
956000 41 e4000006 CMD wb_h stt_wb ; start transfer WBS_H
958000 42 e8000006 CMD wb_v stt_wb ; start transfer WBS_V
960000 16 fc000003 CMD br bstr_wb ; start WBS H&V
962000 24 d7000000 RCMD hr_h 4 ; select HRS_H buffer
964000 25 db000000 RCMD hr_v 4 ; select HRS_V buffer
966000 26 ff800000 CMD br bstr_hr ; start HRS
968000 28 MTX unlock ; release SL I/F
1068000 30 MTX lock ; lock SL I/F
```

Simulation: total No of errors: 0
Exeeded max time. Normal end of execution



Here follows the three TC packets to upload the program.

vmTC_0.txt	vmTC_1.txt			vmTC_2.txt
1c00	1c00	3403	0003	1c00
c000	c000	0002	0050	c000
001d	00bd	3000	0004	0019
0008	0008	0003	0060	0008
0400	0400	1200	0004	0400
0510	0510	0003	1100	0510
0000	0000	0000	0001	0000
0000	0000	002f	3201	0000
0000	0000	4a04	ffeb	0000
0303	032b	0003	fc00	0302
0000	0008	0050	0005	1000
0000	0800	0004	e400	0000
0008	07d0	0060	0006	000a
0000	0100	0004	e800	0000
0200	0001	ff80	0006	0008
0000	4900	0000	1100	d2bd
0400	1000	0801	0000	
e1fb	1200	86a0	3200	
	0002	0100	ffe4	
	0000	0000	0100	
	0032	0800	0000	
	2003	07d0	5000	
	0002	0100	0000	
	e400	0001	0300	
	0009	ff90	0000	
	e800	0000	0330	
	0009	d740	0000	
	fc00	0000	0310	
	0003	db40	0000	
	4901	0000	0320	
	1001	1000	0000	
	1000	0003	d06b	
	0003	4a04		

The following table shows the VM program code to be stored on the ICU OBS.

vmTbl.txt		
----- block	0x34030002,	0x00600004,
start address 0	0x30000003,	0x11000001,
0x00000008,	0x12000003,	0x3201ffeb,
0x00000200,	0x0000002f,	0xfc000005,
0x00000400,	0x4a040003,	0xe4000006,
----- block	0x00500004,	0xe8000006,
start address 8	0x00600004,	0x11000000,
0x080007d0,	0xff800000,	0x3200ffe4,
0x01000001,	0x080186a0,	0x01000000,
0x49001000,	0x01000000,	0x50000000,
0x12000002,	0x080007d0,	0x03000000,
0x00000032,	0x01000001,	0x03300000,
0x20030002,	0xff900000,	0x03100000,
0xe4000009,	0xd7400000,	0x03200000,
0xe8000009,	0xdb400000,	----- block
0xfc000003,	0x10000003,	start address 4096
0x49011001,	0x4a040003,	0x0000000a,
0x10000003,	0x00500004,	0x00000008,