1. INTRODUCTION

The SPIRE DPU communicates with the DRCU through two serial interfaces: the high speed interface, used for collecting science telemetry data from the DRCU; and the low speed interface, used both for commanding the DRCU subsystems and collecting housekeeping data. The On Board Software (OBS) must implement a task which regularly uses the low speed interface to collect housekeeping for the 'periodic housekeeping' telemetry packets that are transmitted to the ground (once per second, TBC). In addition to this task, the OBS must also provide for thermal control of the cooler and other commanding of the instrument subsystems to implement instrument operations. To do this it is necessary to have a mechanism that will allow commanding of the DRCU to be carried out in parallel with the routine housekeeping collection, while maintaining the required timing accuracy for operations.

At a recent OBS meeting, held at RAL on the 31st July and 1st August 2001, IFSI proposed such a mechanism that used the concept of a Command List, which would be interpreted by the OBS. The language used in a Command List provided facilities for ensuring that commands could be sent to the DRCU at the required times without disturbing the housekeeping task unduly.

This note provides an evaluation of that Command List concept and makes recommendations for its use with SPIRE.

2. COMMAND HANDLING BY THE OBS

In order to pass commands to the DRCU, the OBS runs a task (LS_Task), which handles the interface with the low speed serial line (issuing the command and waiting for, and reading, the response). This task takes its input from a command buffer, which contains the commands to be sent to the DRCU and issues them sequentially.

This command buffer is normally loaded regularly (once per second, TBC) by the periodic housekeeping task, which copies a set of commands into the command buffer for handling by the low speed interface task. These commands request the housekeeping information and there will be of the order of 200 commands placed in the command buffer each time.

In addition, commands received by the instrument, or control tasks being run by the OBS, may also generate requests for commands to be sent to the DRCU. These will normally be appended to the command buffer and sent when they reach the top of the buffer, but this may introduce a delay of up to ~200ms (approximately 1ms per command held in the command buffer) before the DRCU command is actually issued. A problem arises for those instrument operations, which require the DRCU commands to be issued with a higher timing accuracy than this.

Note: we can assume that, in general, OBS control tasks have such a long cycle time (of the order of a few seconds, or longer) that the possible delay in issuing commands will have no significant effect.

A hardware mechanism to address this problem has been provided in the DPU in the form of a timer, which can be arranged to interrupt the current operation of the OBS and allow a DRCU command to be issued at a known time. In order for the OBS to use this it is necessary:

a) to stop the LS_Task processing the command buffer at least 1 ms before the required command time in order to ensure that the last command has been dealt with completely before the required command is sent

b) wait until the required time, issue the command and handle the response
c) restart the LS_task to continue processing the command buffer

All of this needs to be carried out in the interrupt routine attached to the timer. Though possible, it is unlikely that two tasks will need to use this mechanism at the same time.
Question: Is it possible that two time critical commanding tasks will be running at the same time?

An overview of the commanding scheme is shown in Figure 2-1

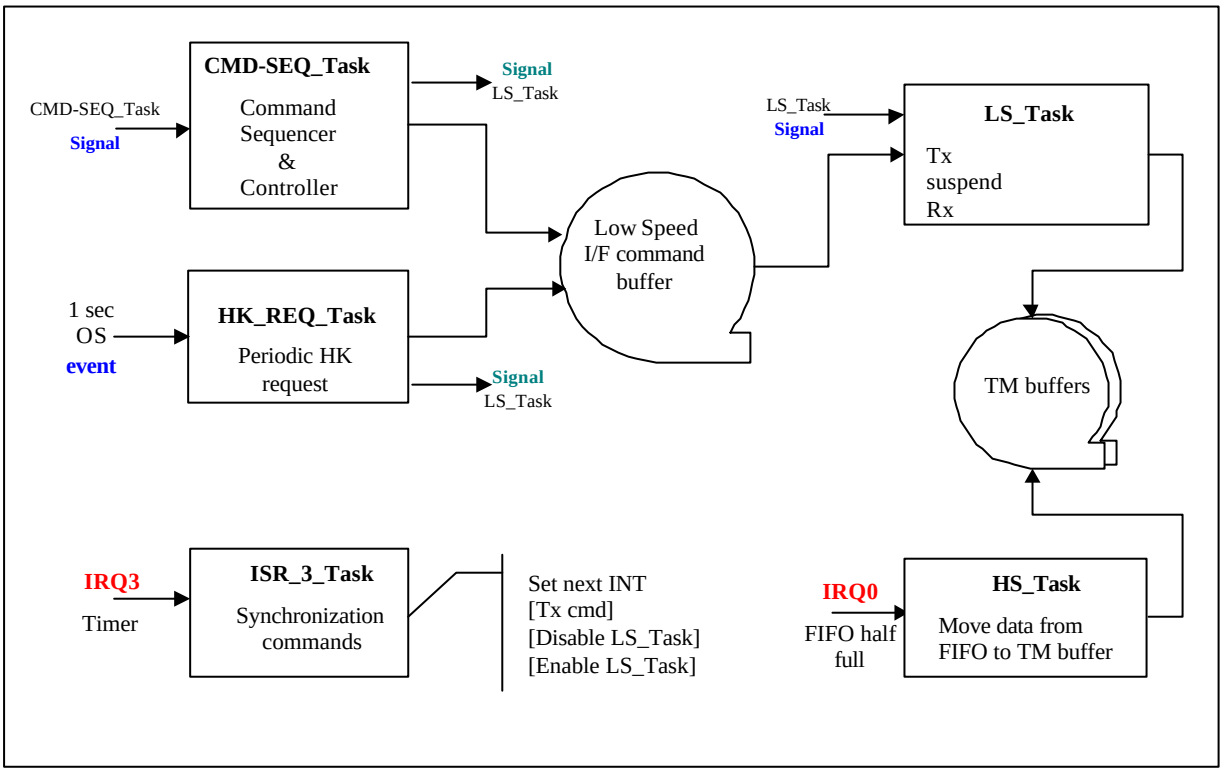

Figure 2-1 OBS Commanding Tasks

2.1 'Command List' concept

The method of using the timer interrupt to issue DRCU commands at the correct times, described above, could be used by any code being executed by the OBS, but IFSI have proposed that instrument functions that need this facility should be coded using a 'language' that is interpreted by the OBS and provides the facilities required for this function. The advantage of this is that the function can be defined/changed without a change to the OBS and it is possible for the function to be entirely encoded within the command packet sent to the instrument.

2.2 Command List Interpreter Virtual Machine

The language interpreter Virtual Machine, as described, has the following characteristics:

- A set of general purpose registers to hold data and allow calculations to be made with this data
- The response from each command issued is returned in a fixed VM register. This value is overwritten each time a command is sent when the response to the previous command is read into the register.
  Note: this removes the need to schedule a second interrupt in order to read the response to a command. (necessary because the interrupt routine cannot wait for the response as this would stop other OBS tasks from running at this time). It does, however, mean that an additional opcode is required to read the response to the last command issued in a function.

2.3 Command List Opcodes

The language interpreter Virtual Machine, provides the following Operation Codes (opcodes).
Note: some language elements have been added since the meeting, in order to provide additional functionality that has been identified (see below):

Opcode	Description
Set(reg, value)	Set a register to contain a constant value
Set(reg, reg')	Copy contents of reg' to reg
Add(reg, const)	Add a constant to reg
Add(reg, reg')	Add the contents of reg' to reg
Sub(reg, const)	Subtract a constant from reg
Sub(reg, reg')	Subtract the contents of reg' from reg
Get(reg, table, reg')	Fetch the value in a table (at the offset given by reg') into reg
Send(response?, subsystem, command, value)	Send a command to the subsystem with a parameter value *Note: Error Checking of the response is TBD*
Send(response?, subsystem, command, reg)	Send a command to the subsystem with a parameter value taken from reg *Note: Error Checking of the response is TBD*
Read(reg)	Fetch response to last command into reg
SetTimer(value)	Set the interrupt timer count to the given value *Note:This will take effect in the interrupt routine following the next interrupt routine*
SetTimer(reg)	Set the interrupt timer count from the content of reg *Note: This will take effect in the interrupt routine following the next interrupt routine*
Mutex(flag)	Flag = set, release Setting the mutex stops the LS_Task reading commands from the command buffer. The low speed interface is then guaranteed to be available after 500ms (TBC)
JumpNZ(reg, displacement)	Jump by displacement to a new position in the Command List, if reg contains non-zero value
Wait()	Return from current interrupt routine
End()	Stop timer interrupts

2.3.1 Additional Opcodes

2.3.1.1 Subroutines

It will probably be useful to provide for 'subroutines' to be defined for reusable parts of functions. This will require the OBS to provide a Command List Area to contain these subroutines. Instrument commands to load and update these will be provided (through the On-board Command Procedure Service, TBC)

Call(reg, address)	Jump to Command List address. The return address is put into reg
Return(reg)	Jump to Command List address held in reg

2.3.1.2 DPU Science Data Frames

It will be necessary to be able to send the current status of the function (step number etc) in a DPU Science telemetry packet:

DPUFrame(reg, n)	Add a DPU science frame (defined by the n registers starting with reg) to the DPU telemetry packet buffer. *Time information will be automatically added at the start of the science frame by the DPU.*

2.3.1.3 Additional PUS Packet types

It will be necessary to have the capability to issue other PUS packet types:

2.3.1.3.1 Command Execution Status

These comprise the following service types :
Command Execution Start Report (1,3)
Command Execution Progress Progress Report (1,5)
Command Execution Completed Report (1,7)
Command Execution Failure Report (1,8)

CVPkt(type,reg,n)	Issue a Command Execution Status Packet of given type, with parameters given in the n registers starting with reg *It is assumed that reg n onwards contain the Telecommand Packet ID and the Packet Sequence Control words of the telecommand being executed*

2.3.1.3.2 Events

These comprise the following service types:
Event Report (5,1)
Exception Report (5,2)

Error/Alarm Report (5,4)

EventPkt(type,reg,n)	Issue an event packet of given type, with parameters given in the n registers starting with reg

3. EXAMPLE FUNCTIONS

The following typical function has been discussed:

3.1 Chopping

A possible chopping scenario is as follows:

Function:

CHOP(y0, y1, Nchops, Pchop, Nchpframes, Pchpframes, Nbdaframes, Pbdaframes, Tbda)

Where:

y0,y1	=	on/off position of BSM chop axis
Nchops	=	number of chop cycles to be completed
Pchop	=	period of the chop cycle (default = 500ms)
Nchpframes	=	number of chop science frames collected per half chop cycle
Pchpframes	=	time between collected chop science frames
Nbdaframes	=	number of BDA science frames collected per half chop cycle
Pbdaframes	=	time between collected BDA science frames
Tbda	=	time delay from half chop cycle start to first collected BDA science frame

Algorithm

Set chop number = 0
Send 'Command Execution Started' message
For Nchops cycles:
  Set chop position to y0
  Send DPU science frame giving chop number
  Increment chop number
  Wait until Tbda has passed
  Collect Nbdaframes at Pbdaframes Period
  Wait until Pchop/2 has passed
  Set chop position to y1
  Wait until Tbda has passed
  Collect Nbdaframes at Pbdaframes Period
  Wait until Pchop
Send 'Command Execution Completed' message

Implementation

Assumptions:
Assume that on receipt of the telecommand packet holding the CHOP command, the command

processor enables interrupts for 1 ms intervals and sets the start address for the Command List Interpreter to the start of the Command List.

Assume the BSM is already powered on and ready to chop

Each block of opcodes are executes during one timer interrupt

Delta Time	Commands	Comments
1ms	Set(R00, y0) Set(R01, y1) Set(R02, Nchops) Set(R03, Pchop) Set(R04, Nchpframes) Set(R05, Pchpframes) Set(R06, Nbdaframes) Set(R07, Pbdaframes) Set(R08, Tbda) Set(R56, 0) Set(R57, 0) Sub(R03, R08) Sub(R03, 2) Sub(R08,1) CVPck(Execution Start,R56) Mutex(set) Wait()	Copy arguments to registers Initialise registers Chop Cycle Chop Flag R03 = Pchop-Tbda-2 R08 = Tbda-1
1ms	Send(Y, MCU, Set_chop_frame_period, R05) Wait()	Set Chop Science Frame period
1ms	Send(Y, DCU, Set_bda_frame_period, R07) Wait()	Set BDA Science Frame period
1ms	Send(Y, MCU, Move_chop, R00) Issue(R56, 2) Add(R56, 1) Set(R57, 1) SetTimer(R08ms) Wait()	Move BSM to y0 Create Science Frame Chop flag = 1
1ms	Send(Y, MCU, Start_multiple_chop_frames, R04) SetTimer(1ms) Mutex(release) Wait()	Start Chop Sampling
Tbda-1 ms	SetTimer(R03ms) Mutex(Set) Wait()	

1ms	Send(Y, DCU, Start_multiple_frames, R06) SetTimer(1ms) Mutex(release) Wait()	Start Detector Sampling
Pchop-Tbda-2ms	Mutex(Set) Wait()	
1ms	Send(Y, MCU, Move_chop, R01) Issue(R56, 2) Set(R57,0) SetTimer(R08ms) Wait()	Move to y1 Chop flag = 0
1ms	Send(Y, MCU, Start_multiple_chop_frames, R04) SetTimer(1ms) Mutex(release) Wait()	Start Chop Sampling
Tbda-1 ms	SetTimer(R03ms) Mutex(Set) Wait()	
1ms	Send(DCU, Start_multiple_frames, R06) SetTimer(1ms) Mutex(release) Wait()	Start Detector Sampling
Pchop-Tbda-2 ms	Mutex(set) Decr(R02,1) JumpNZ(R02, -37) Wait()	Repeat for Nchops cycles
1ms	Send(Y, MCU, Move_chop, R00) Mutex(release) CVpkt(Execution Completed,R56) End()	Move BSM to y0

ANALYSIS OF IMPLEMENTATION

It is apparent that the length of the Command List for the CHOP function (64 opcodes) may well be larger than can be contained in one telecommand packet (250 bytes). The length may be reduced by more efficient language design: the proposed implementation uses 8 bytes for each language statement. These could be encoded in 4 bytes (Note: all commands to the DRCU have the most significant bit set to 1). However, it should be noted that other functions could be much more complicated than CHOP and hence would be longer.

The command interpreter could copy all parameters received with the command into a fixed set of VM registers before executing the Command List. This would save many commands used to set up registers before execution begins.

The use of subroutines may reduce the length of Command Lists. For example, most of the CHOP function is used within the POINTSRC and FIELDMAP commands. A block of memory would be set aside to contain Command Lists and these could be called from any function. This does mean that we need to provide instrument commands to Create and Update Command List subroutines. We would use the On-Board Control Procedure service to provide these (TBC).

This implementation includes no error checking. This will increase the size of the Command List and require additional opcode(s) to issue event packets.

4. SUMMARY

- The proposed method of sending issuing DRCU commands is adequate for implementation of the instrument functionality and timing for observing modes. However, there are constraints on other tasks that may be running at the same time (only one function may be run at a time and control tasks must have a long cycle time). **The acceptability of this needs to be studied.**

- The Command List concept provides flexibility for defining commands, which allows their redefinition without patching the OBS. **It is recommended that this method be adopted.**

- With additional opcodes, the proposed Command List language can provide the functionality to implement observing modes of the instrument. **A proposal for the full language definition should be made as soon as possible and studied to ensure a) that is provides all the functionality needed and b) that it can be used in the CUS.**

- The language needs to be made more efficient. **It is recommended that the Command List language be designed to be more efficient in size than the proposed implementation (maximum 4 bytes per opcode).**

- It is not possible to always include the full Command List in a telecommand (due to the limit on telecommand length), therefore subroutines will have to be stored on board and called from the Command List. **An additional commanding service (On-Board Control Procedures, TBC) should be implemented in the OBS.**