National Defence Défense nationale

DEFENSIO PER SCIENTIAM

# MDR/OMNI-BAND RECONFIGURABLE TERMINAL DATA PACKET SPECIFICATION (U)

by

Robin Addison

## DEFENCE RESEARCH ESTABLISHMENT OTTAWA

TECHNICAL NOTE 98-009

Canada

August 1998
Ottawa

AQF99-04-0702

National Defence    Défense nationale

# MDR/OMNI-BAND RECONFIGURABLE TERMINAL DATA PACKET SPECIFICATION (U)

by

**Robin Addison**
*Milsatcom Group*
*Space System & Technology Section*

**DEFENCE RESEARCH ESTABLISHMENT OTTAWA**
TECHNICAL NOTE 98-009

Project
5CA11

August 1998
Ottawa

## Abstract

In 1997, Defence Research Establishment Ottawa and Communications Research Centre started research into an omni-band reconfigurable terminal. Such a terminal will enable soldiers to use a single terminal to communicate over any satellite communications or terrestrial link. Each terminal will support multiple standards and the first one to be implemented will be extremely high frequency medium data rate communications. Development begins with two hardware simulators: the payload simulator and the ground terminal simulator. Each simulator has a PC-based host to run the simulator, a digital chassis containing several digital signal processor boards to run the payload or terminal, and a radio frequency chassis to interface to the digital boards. Within the simulators, specifically on the digital signal processor boards, it is desirable to have a common data format for interchange between modules to simplify interfacing and reconfiguration.

This document contains the data packet format, data packet validation, description of the minimum implementation, expansion capabilities, coding excerpts and the listing of necessary definitions for coding. The requirements of flexibility, simplicity and future expandability drove the design of the data packets. Since the data packet design precedes the detailed system design, future expandability was a key requirement. The data packet consists of a variable length header and a variable length data area. The header includes the details of the data storage as well as information about the data source and destination.

There is sufficient versatility in the specification to allow the data packet to be used for all known intermodule data, and for new requirements. It is likely that this specification will be further refined as requirements become firmer and the design of the simulator becomes better defined.

## Résumé

En 1997, le Centre de recherches pour la défense Ottawa et le Centre de recherches des communications ont entrepris une recherche sur une station terrestre reconfigurable à omni-bande. Cette station terrestre serait capable de communiquer sur n'importe quelle bande de fréquence, qu'elle soit terrestre ou par satellite. Chaque station supportera une multitude de normes de communication et la première qui sera implantée sera celle de la bande millimétrique à taux moyen de données. La recherche commence avec la développement de deux simulateurs ***hardware***. Dans chaque simulateur, il y aura un ordinateur PC pour le commander, plusieurs cartes à traitement numériques et un châssis fréquence radio pour l'interface des cartes numériques. Entre les modules, particulièrement pour le traitement numérique, il est important d'avoir un format de données commun qui permet la reconfiguration et qui simplifie l'interface.

Dans ce rapport, le format, la validation, la réalisation minimum, les possibilitées d'accroissement, les exemples de logiciels et les définitions nécessaires pour programme des paquets de données sont fournis. La flexibilité, la simplicité et le potentiel de l'accroissement ont influencé la conception des paquets de données. La capacité de faire des changements était essentielle parce que la conception du système n'est pas encore finalisée. L'en-tête des paquets de données ainsi que la donnée elle-même est de longueur variable. L'en-tête contient les détails de l'entreposage des données et l'information de la source et de la destination.

Cette spécification est assez versatile pour permettre aux paquets de données d'être utilisées partout dans les simulateurs. Il est probable que cette spécification deviendra plus raffinée quand la définition du système sera terminée.

## Executive Summary

In 1997, Defence Research Establishment Ottawa (DREO) and Communications Research Centre (CRC) started research into an omni-band reconfigurable terminal. Such a terminal will enable soldiers to use a single terminal to communicate over any satellite communications or terrestrial link. On ships, several of these terminals, to provide redundancy and concurrent communications over different links, could replace the current multiplicity of communications installations. Each terminal will support multiple standards and the first one to be implemented will be extremely high frequency (EHF) medium data rate (MDR) communications. This also provide the technical knowledge necessary to support the Canadian Military Satellite Communications project.

In the MDR system, there are two simulators: the payload simulator and the ground terminal simulator. Each simulator has a PC-based host that provides operator control, permanent storage, and development capability. There is also a digital chassis containing several digital signal processor boards, custom general purpose boards, and specialized analog-to-digital and input/output boards. Together, these provide the functionality for ground terminal/payload control, access/resource control, synchronization control and communications processing. A radio frequency chassis includes hopping synthesizers, up and down conversion chains with associated amplifiers, mixers and filters.

Within the simulators, specifically on the DSP boards, there will be many types of data passed between boards, processors and routines. This data has many forms, some examples are data bits, encoded data bits, cryptographic key streams, synchronization estimates and analog-to-digital samples. It is desirable to have a common format that includes the various types of data. Having a common format would simplify interfacing and make it easier to reconfigure for different communications systems.

Within this document, the data packets used between modules within the simulator are specified. The requirements of flexibility, simplicity and future expandability drove the design of the data packets. It is important that the data packets be flexible to support different data such as bits, symbols, or digitized analog samples. This data can be packed for storage efficiency or unpacked for rapid manipulation. The extraction and storing of data within the packets is simple and does not take excessive overhead. The data packet design uses the target processor (TMS 320C6201) word size and includes aids in the header to simplify extraction of the data. Since the data packet design precedes the detailed system design, future expandability is a key requirement. The design allows for different data types, sizes and packing. It also allows for different usage of the data packets that cannot be predicted at this time.

This document describes the format of the data packet and provides a section on the validation that must be performed by each module prior to using the data. Also provided is a description of the minimum implementation of data packet handling for initial development of bit handling and A/D sample handling modules. Expansion capabilities of the data packet specification are given. Finally, coding excerpts for handling of these data packets and the listing of necessary definitions for coding are given.

The data packet consists of a variable length header and a variable length data area. The header includes the details of the data storage as well as information about the data source and destination. The specification allows for future expansion in the header (format and size) and for different data formats.

There is sufficient versatility in the specification to allow the data packet to be used for all known intermodule data, and for new requirements. It is likely that this specification will be further refined as requirements become firmer and the design of the simulator becomes better defined.

## Table of Contents

**Abstract** .......... iii

**Résumé** .......... iii

**Executive Summary** .......... v

**Table of Contents** .......... vii

**List of Abbreviations** .......... ix

**1. Introduction** .......... 1

1.1 The Problem .......... 1
1.2 Requirements .......... 1
1.3 Outline .......... 2

**2. Data Packet Format** .......... 3

2.1 General .......... 3
2.2 Header .......... 3
2.3 Data Area .......... 6

**3. Essential Validation** .......... 7

**4. Minimum Implementation for Initial Development** .......... 7

4.1 General .......... 7
4.2 Bit Data Packets .......... 8
4.3 A/D Sample Data Packets .......... 8

**5. Expansion of the Data Packet Specification** .......... 8

5.1 General .......... 8
5.2 Special Fields .......... 8
5.3 Use of Symbols for Legal Values .......... 9
5.4 Inclusion of Support for Floating Point .......... 9
5.5 Data Type Field .......... 9
5.6 Header Size Field .......... 9
5.7 Header Type Field .......... 9

**6. Example Code Excerpts** .......... 9

6.1 General .......... 9
6.2 Invoking Process .......... 10
6.3 Module Initial Code .......... 10
6.4 Module Extracting Unpacked Data .......... 11
6.5 Module Extracting Arbitrarily Packed Data .......... 11

**7. DATAPACK.H Listing** .......... 11

**8. Conclusion** .......... 13

## List of Abbreviations

| | |
|---|---|
| A/D | Analog to digital |
| C6201 | Texas Instruments fixed-point TMS 320C6201 digital signal processor |
| C6701 | Texas Instruments floating-point TMS 320C6701 digital signal processor |
| CRC | Communications Research Centre |
| DATAPACK.H | Header file containing data packet definitions for 'C' programming |
| DREO | Defence Research Establishment Ottawa |
| DSP | Digital signal processor |
| EHF | Extremely high frequency |
| FSK | Frequency-shift keying modulation |
| I | In-phase |
| I/O | Input/output |
| MDR | Medium Data Rate (4.8 kb/s to 2 Mb/s) |
| PCI | A high-speed bus used in modern personal computers |
| Q | Quadrature |
| RF | Radio frequency |
| x86 | The family of personal computer processors including 486 and Pentium |
| VME | A bus used for plug-in processors |

### Hexadecimal Notation

0xFF represents FF in base 16 and is equal to 255 in base 10. This follows the 'C' convention.

### Programming Definition Notation

There are several definitions to be used by 'C' programmers with names similar to OFF_UNIQUE_WORD or HDR_NORMAL. These are definitions supplied in DATAPACK.H (section 7 contains the listing) and should be used rather than hard-coded numbers when programming in 'C'.

# 1. Introduction

In 1997, Defence Research Establishment (DREO) and Communications Research Centre (CRC) started research into an omni-band reconfigurable terminal. Such a terminal will enable soldiers to use a single terminal to communicate over any SATCOM or terrestrial link. On ships, several of these terminals, to provide redundancy and concurrent communications over different links, could replace the current multiplicity of communications installations. Each terminal will support multiple standards. The first standard to be implemented will be extremely high frequency (EHF) medium data rate (MDR) communications. This also provide the technical knowledge necessary to support the Canadian Military Satellite Communications project.

Fig. 1 shows the block diagram for the MDR system simulator. There are two main simulators within the system: the payload simulator and the ground terminal simulator. Each simulator has a PC-based host that provides operator control, permanent storage, and development capability. The host is the only portion that interacts with the user. In addition to the host, there is a digital chassis containing several digital signal processor (DSP) boards, custom general purpose boards, and specialized analog-to-digital (A/D) and input/output (I/O) boards. Together, these provide the functionality for ground terminal/payload control, access/resource control, synchronization control and communications processing. A radio frequency (RF) chassis includes hopping synthesizers, up and down conversion chains with associated amplifiers, mixers and filters. The RF chassis is based on a modular design to allow substitution for different bands.

## 1.1 The Problem

Within the simulators, specifically on the DSP boards, there will be many types of data passed between boards, processors and routines. This data has many forms. Some examples are data bits, encoded data bits, cryptographic key streams, synchronization estimates and analog-to-digital samples. It is desirable to have a common data format that includes all the various types of data. A common format simplifies interfacing and reconfiguration for different communications systems.

This document specifies the data packets used between modules within the simulator. An example of such a packet is the data packets used between the coder and interleaver of the communications processor.

## 1.2 Requirements

The requirements of flexibility, simplicity and future expandability drove the design of the data packets. It was important that the data packets be flexible to support different data such as bits, symbols, or digitized analog samples. This data can be packed for storage efficiency or unpacked for rapid manipulation.

The process of extraction and storage of data within the packets must be simple and not take excessive overhead. Thus the data packet design uses the target digital signal processor (C6201) word size. Aids were also included, in the header, to simplify extraction of the data. Since the data packet design precedes the detailed system design, future expandability is a key requirement. The design allows for different data types, sizes and packing. It also allows for different usage of the data packets that cannot be predicted at this time.

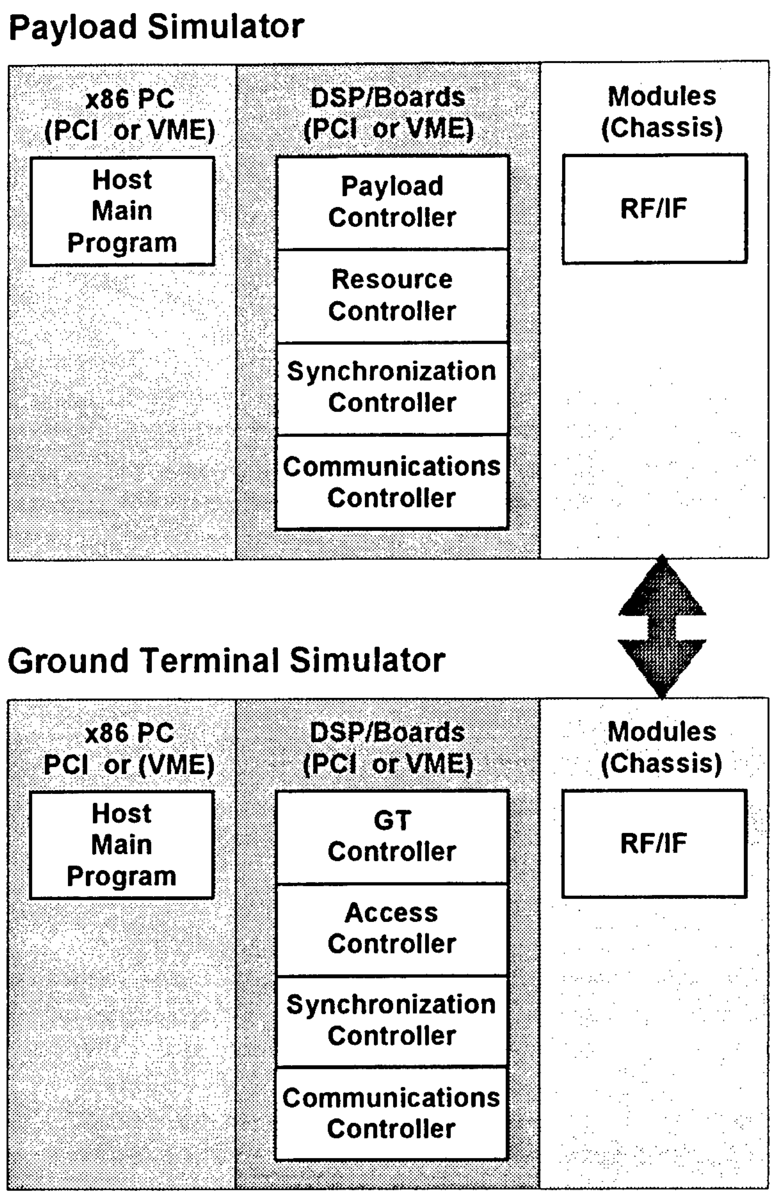

**Fig. 1. System block diagram.**

## 1.3 Outline

This document first describes the format of the data packet, then the validation procedure that must be performed by each module prior to using the data is given. This is followed by a description of the minimum implementation of data packet handling for initial development of bit handling and A/D sample handling modules. Expansion capabilities of the data packet specification are given. Example coding excerpts are given for handling of these data packets. Finally, the listing of DATAPACK.H is given which contains the necessary definitions for coding.

# 2. Data Packet Format

## 2.1 General

The data packet consists of a variable length header and a variable length data area shown in Fig. 2. The header includes the details of the data storage as well as information about the data source and destination. The specification allows for future expansion in the header (format and size) and for different data formats.

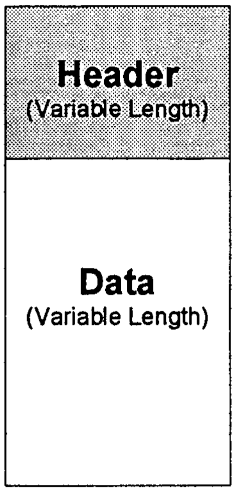

**Fig. 2. Data packet diagram.**

The intent with data packets is to minimize unnecessary data transfer or reformatting. Modules will receive pointers to input and output data packets rather than being passed the data on the stack. Ideally, the data will only be extracted once, and stored once during the processing.

## 2.2 Header

The header is variable length, with a minimum size of 14 32-bit words. The preferred size to use is 16 which allows for two unused special fields at the end of the header. If more special fields are required, the header length can be increased to the necessary value. Fig. 3 shows the header including the distinct 32-bit fields.

The header consists of 16 fields of 32 bits each. All fields in the header must be completed. The different fields of the header, along with values to be used and ".h" definitions used to access them are given the following table:

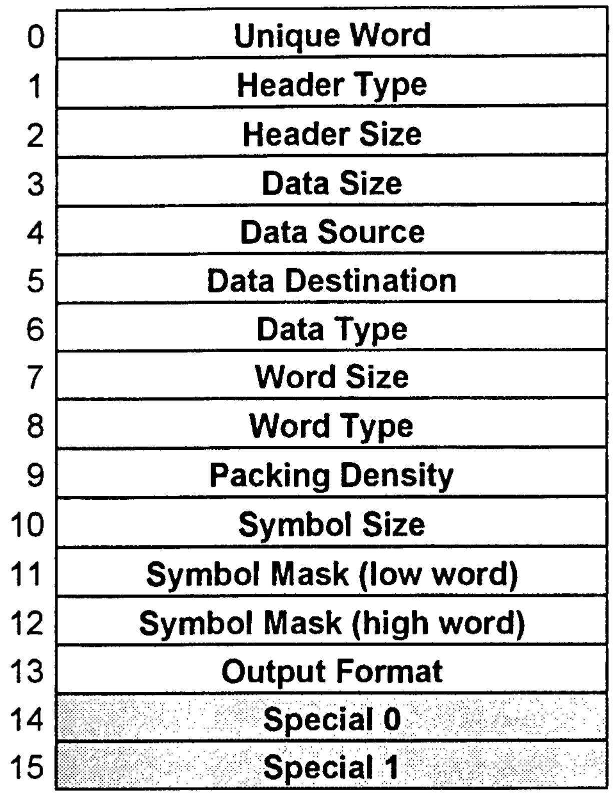

**Fig. 3. Data packet header diagram.**

**Table 1. Data packet header fields.**

| Offset | Definition | Description | Legal Values |
|---|---|---|---|
| 0 | OFF_UNIQUE_WORD | This is a unique word used to delimit and identify a data packet. | HDR_UNIQUE_WORD |
| 1 | OFF_HEADER_TYPE | For future expansion, this field allows different kinds of headers | HDR_NORMAL<br><br>(other values may be added later) |
| 2 | OFF_HEADER_SIZE | This indicates the header size in 32-bit words. Larger sizes allow more special fields. Since the data follows immediately after the header, this also is the offset to the data area. | Minimum 14, but normally 16 |
| 3 | OFF_DATA_SIZE | This is the data area size in 32-bit words. | Minimum 1 |

| Offset | Definition | Description | Legal Values |
|---|---|---|---|
| 4 | OFF_DATA_SOURCE | This is the module that is the source of the data. It can be used to verify the data packet's origin. It can also be used by a module to distinguish data packets from various sources. | MOD_ACCESS_GENERATION<br>MOD_ACCESS_PROCESSING<br>MOD_CRC_DECODER<br>MOD_CRC_ENCODER<br>MOD_DATA_INPUT<br>MOD_DATA_OUTPUT<br>MOD_DECODER<br>MOD_DEINTERLEAVER<br>MOD_ENCODER<br>MOD_DEMODULATOR<br>MOD_FRAME_EXTRACTOR<br>MOD_FRAME_FORMATTER<br>MOD_FREQUENCY_DEPERMUTE<br>MOD_FREQUENCY_PERMUTE<br>MOD_INTERLEAVER<br>MOD_MODULATOR<br>MOD_RESOURCE_ALLOCATOR<br>MOD_RESOURCE_REQUESTER<br>MOD_SYNC_PROCESSOR<br>MOD_TIME_CONTROLLER<br>MOD_TIME_DEPERMUTE<br>MOD_TIME_PERMUTE<br>MOD_TRANSEC |
| 5 | OFF_DATA_DESTINATION | This is the destination of the packet. For an input packet, this should match the executing module. For an output packet, typically the module is not known. | Same as above. Also, for output packets the following value is normally used:<br>MOD_UNKNOWN |
| 6 | OFF_DATA_TYPE | This is the type of data packet. Some modules may receive different data and this allows them to be distinguished. | TYPE_STANDARD<br><br>(others value may be added later) |
| 7 | OFF_WORD_SIZE | This is the word size used for extracting the data from the data area. (It is **not** used for header/data sizes which are always 32 bits.) | WORD_32_BITS (normal)<br>WORD_64_BITS (used only for double and 40 bit longs) |
| 8 | OFF_WORD_TYPE | This is the bit format of the data. Integer values are signed. Floating point values have sign and exponent. | WORD_UNSIGNED (normal bits)<br>WORD_INTEGER (normal samples)<br>WORD_FLOAT (floating point) |
| 9 | OFF_PACKING_DENSITY | This is the number of symbols stored per word (32 or 64 bits depending on word size). Symbols cannot cross word boundaries. | 1 (unpacked)<br>2 or more (packed)<br><br>The maximum packed value is:<br>Word Size / Symbol Size |
| 10 | OFF_SYMBOL_SIZE | The number of bits per symbol | See Table 2 |
| 11 | OFF_SYMBOL_MASK_LO | This is the mask used to extract bits from a packed word. For 32-bit word size, only the low part is used. For 64-bit word size, the high part (below) is also used. | See Table 2 |

| Offset | Definition | Description | Legal Values |
|---|---|---|---|
| 12 | OFF_SYMBOL_MASK_HI | This is the high part of the mask used to extract bits from a packed word. This part is only used for 64-bit word size and then is used with the low part of the mask (above). | See Table 2 |
| 13 | OFF_OUTPUT_FORMAT | This is the output format requested by the process invoking the module for most efficient processing of the subsequent module. It should be adhered to where possible, but is not compulsory. | OUT_UNSPECIFIED (no preference)<br>OUT_UNPACKED (1 symbol / word)<br>OUT_PACKED (maximum symbols / word)<br><br>(other values may be added later) |
| 14 | OFF_SPECIAL_0 | This is a special field as yet unassigned to allow for future expansion and custom uses between modules. | SPEC_UNUSED (not used now) |
| 15 | OFF_SPECIAL_1 | This is a special field as yet unassigned to allow for future expansion and custom uses between modules. | SPEC_UNUSED (not used now) |

The next table presents the various types of data symbols to be used in the data packets along with their associated sizes and the masks used to extract them from the data area.

**Table 2. Data packet header symbol size and mask detail.**

| Symbol Size (bits) | Symbol Mask (Low word) | Symbol Mask (High word) | Use |
|---|---|---|---|
| 1 | 0x1 | 0x0 | normal bits |
| 3 | 0x7 | 0x0 | FSK symbol |
| 3 or 4 | 0x7 or 0xF | 0x0 | soft decision bits |
| 8 | 0xFF | 0x0 | byte |
| 16 | 0xFFFF | 0x0 | A/D samples |
| 32 | 0xFFFFFFFF | 0x0 | integer |
| 32 | 0xFFFFFFFF | 0x0 | float |
| 40 | 0xFFFFFFFF | 0xFF | long integer |
| 64 | 0xFFFFFFFF | 0xFFFFFFFF | double |

## 2.3 Data Area

The data area, which follows immediately after the header, consists of a sequence of symbols. If in unpacked format, there is one symbol per word. If packed, the maximum number of symbols that fit evenly in a word will be used. Packed symbols do not cross word boundaries. This can result in some wasted bits. Any unused bits must be set to 0. (Sign extension is not allowed.)

Long or double values are to be stored least significant word first (lowest address) followed by most significant word. A/D samples will be stored with the in-phase (I) sample first followed by the quadrature (Q) sample.

The size of the data area is application/module dependent but the design goal is to ensure a data packet contains one frame's worth of data. The module should not restrict the data size and should be able to handle data packets of just one symbol. Varying size of data packets and module algorithms mean that there will not be a one-to-one correspondence between input packets and output packets. All modules will return a logical TRUE if the output packet contains valid data. (This will not be part of the packet itself.)

## 3. Essential Validation

To ensure robustness, it is essential that all modules validate the data packet header prior to processing the data. Validation is to ensure that it is a data packet, that it was meant to be processed by this module, and finally that it is in a format that can be handled by this module. The fields that must be validated are:

a. Unique Word: must be HDR_UNIQUE_WORD
b. Header Type: must be a supported type
c. Header Size: must be at least 14
d. Data Destination: must match current module
e. Data Type: must be a supported type
f. Word Size: must be a supported size
g. Word Type: must be a supported type
h. Packing Density: must be a supported density
i. Symbol Size: must be a supported size
j. Output Format: must be a supported format

Validation must proceed in the order of the fields. This is especially important if different headers types are used later where some of the fields may be redefined. If any validation fails, it is important that processing not continue and that an error message be generated.

The header does not include any special error checking fields such as a checksum. This is because the overhead involved in computing end error check was too large compared to the minimal risk of an erroneous transfer of data packet. Packets will be transferred within the same DSP board or through a short shielded link, and it is unlikely that errors will be caused.

## 4. Minimum Implementation for Initial Development

### 4.1 General

For initial development, it is not necessary that all modules support all possible combinations of data storage. For purposes of commonality and interoperability, there are two minimum implementations of data packet processing. The minimum implementation to be used depends on the type of data: bits or A/D samples. Modules must always return errors in the event that they cannot handle the data packet received.

## 4.2 Bit Data Packets

Almost all modules in the data communications processor deal with bits. The minimum implementation for these modules (such as interleavers and coders) is:

a. Header Type: HDR_NORMAL
b. Header Size (32-bit words): 16 (this allows two unused special fields)
c. Data Type: TYPE_STANDARD
d. Word Size: WORD_32_BITS
e. Word Type: WORD_UNSIGNED
f. Packing Density (symbols/word): 1 (unpacked)
g. Symbol Size (bits): 1
h. Symbol Mask: 0x1
i. Output Format: OUT_UNPACKED

## 4.3 A/D Sample Data Packets

Within the data communications processor, at least two modules deal with digitized samples: the modulator and demodulator. To handle A/D samples, modules must also support an additional implementation:

a. Header Type: HDR_NORMAL
b. Header Size (32-bit words): 16 (this allows two unused special fields)
c. Data Type: TYPE_STANDARD
d. Word Size: WORD_32_BITS
e. Word Type: WORD_INTEGER
f. Packing Density (symbols/word): 1 (unpacked)
g. Symbol Size (bits): 16
h. Symbol Mask: 0xFFFF
i. Output Format: OUT_UNPACKED

# 5. Expansion of the Data Packet Specification

## 5.1 General

It is anticipated that the definition provided here will need to be expanded as the development of the simulator progresses and requirements become better defined. The specification deliberately includes room for expansion in various areas described below in order from least to most effect on the specification.

## 5.2 Special Fields

Every data packet will normally have at least two special fields at the end (to round the header size to 16 words). At this time, there is no special use for the fields. Later, they could be used to identify an access, user, service or terminal for the access control routines. They could also be used to include time (absolute or by hop number, and frame number) for synchronization routines. If the data has some form (such as rectangular) then the special fields could be used for row and column size.

### 5.3 Use of Symbols for Legal Values

All options provided for the legal values are identified by symbols defined in DATAPACK.H. It is relatively easy to add additional options by adding more definitions. For example, if more detail if needed for the output format field, it would be easy to add a symbol for a specific type of packed data output (say 16 bits per word).

### 5.4 Inclusion of Support for Floating Point

Though the target digital signal processor, C6201, does not support floating point, the 6701 will have that capability. It may be advantageous to use floating point representation especially for packets used by the synchronization processor. The definitions provided in DATAPACK.H contains symbols to support single and double precision floating point data.

### 5.5 Data Type Field

Some modules, such as the synchronization processor, will generate or accept different types of data, possibly to or from the same module. By specifying a different data type, type data packets can be easily distinguished.

### 5.6 Header Size Field

The header size, recommended to be 16, allows for two special fields. By increasing the header size, extra fields which are needed for special data packets can be accommodated. See the previous section on special fields for their use.

### 5.7 Header Type Field

If the data packet specification is too restrictive for some unanticipated requirement, then a separate header type can be defined. This allows changing the definition of all subsequent fields. A different header type will only be used as a last resort.

## 6. Example Code Excerpts

### 6.1 General

Below are examples of code to demonstrate how to invoke a module, how to do the validation, and how to extract unpacked and packed data. This code has not been tested. The definitions can be found in DATAPACK.H which is also listed at the end of this document.

## 6.2 Invoking Process

This portion of the code is in the invoking process (for example the one that calls the interleaver). The code used here reflects the use of an interleaver module and the data packet has a header length of 16 with 32-bit data words. Note to minimize data transfer, the invoking routine passes a pointer to the data packet rather than the packet itself. The routine return data in the output packet withou modifying the input packet. Also note that the interleave state is passed to the module as well - this allows multiple uses of the interleaver without reinitializing for each use.

```
unsigned int data_packet_in[48];
unsigned int data_packet_out[48];
struct Interleaver_State data_interleaver
      ... initialize data_interleaver for startup
      ... set values in header and data area of data_packet_in

if (interleave(data_packet_in,data_packet_out,&data_interleaver)) {
      ... process data_packet_out
}
```

## 6.3 Module Initial Code

Upon receipt of a data packet, the interleaver module validates the data packet and sends an error message if the format is not right or not supported (this is shown by the pseudo-code "...crash"). While validating the header, the module also does some preliminary work to prepare for data extraction.

```
int interleave(unsigned int *dp_in,unsigned int *dp_out,
             struct Interleaver_State *my_interleaver)
{
      unsigned int header_size;          // Size of header (words)
      unsigned int packing_density       // Number of symbols per word
      unsigned int *data_base;           // Pointer to start of data
      unsigned int symbol_size           // Number of bits / symbol
      unsigned int symbol_mask;          // Symbol bit mask
      unsigned int data_word_N;          // Nth symbol value
      unsigned int word_N;               // Word number of desired symbol
      unsigned int shift_N;              // Symbol location within a word

      if (dp_in[OFF_UNIQUE_WORD] != HDR_UNIQUE_WORD) ...crash
      if (dp_in[OFF_HEADER_TYPE] != HDR_NORMAL) ...crash
      header_size = dp_in[OFF_HEADER_SIZE];
      if (header_size < 14) ...crash
      if (dp_in[OFF_DATA_DESTINATION] != MOD_INTERLEAVER) ...crash
      if (dp_in[OFF_DATA_TYPE] != TYPE_STANDARD) ...crash
      if (dp_in[OFF_WORD_SIZE] != WORD_32_BITS) ...crash
      if (dp_in[OFF_WORD_TYPE] != WORD_UNSIGNED) ...crash
      packing_density = dp_in[OFF_PACKING_DENSITY];
      if (packing_density != 1) ...crash
      symbol_size = dp_in[OFF_SYMBOL_SIZE];
      if (symbol_size != 1) ...crash
      symbol_mask = dp_in[OFF_SYMBOL_MASK_LO];
      data_base = dp_in + header_size;
      if ((dp_in[OFF_OUTPUT_FORMAT] != OUT_UNPACKED) ||
          (dp_in[OFF_OUTPUT_FORMAT] != OUT_UNSPECIFIED)) ...crash
```

## 6.4 Module Extracting Unpacked Data

Within the interleaver module, when the Nth symbol is required, the following code can be used as long as the data is unpacked (packing density = 1).

```
        data_word_N = data_base[N];
```

## 6.5 Module Extracting Arbitrarily Packed Data

Within the interleaver module, when the Nth symbol is required, the following code works regardless of whether the data is packed or not. This code is less efficient than the previous code for unpacked data. Simple optimizations to this code are available if the access to the data is sequential.

```
        word_N = N / packing_density;
        shift_N = (N % packing_density) * symbol_size;
        data_word_N = (data_base[word_N] >> shift_N) & symbol_mask;
```

# 7. DATAPACK.H Listing

Below is the listing for the header file containing the symbols used in 'C' programs.

```
/*=========================================================================*/
/*                                                                         */
/*                         D A T A P A C K . H                             */
/*                                                                         */
/*      Header file with definitions necessary for the header of intermodule */
/* data packets within the DSP portion of the simulator.  This header file  */
/* should be included in all modules.                                       */
/*                                                                         */
/* Author:      Robin Addison                                               */
/* Language:    TI C for 6201, no special compile flag requirements         */
/* Hierarchy:   "Include" with all modules that pass data at compile time   */
/* Purpose:     Provide definitions to access elements of the data packet   */
/* Inputs:      NA                                                          */
/* Outputs:     NA                                                          */
/* Returns:     NA                                                          */
/*                                                                         */
/* Change History:                                                          */
/*                                                                         */
/*    Date     Version     Programmer      Comment                          */
/* ---------   -------   --------------   -------------------------------- */
/* 11 Jun 98     1.0     Robin Addison     First written                    */
/*                                                                         */
/*                                                                         */
/*                                                                         */
/*-------------------------------------------------------------------------*/

/* Version number */
#define VER_DATAPACK_H                    1.0

/* Header offsets and definitions */
#define OFF_UNIQUE_WORD                                             0
        #define  HDR_UNIQUE_WORD          0xF981006F
```

```
#define OFF_HEADER_TYPE                                                   1
        #define  HDR_ILLEGAL                0
        #define  HDR_NORMAL                 1
#define OFF_HEADER_SIZE                                                   2
#define OFF_DATA_SIZE                                                     3
#define OFF_DATA_SOURCE                                                   4
        #define  MOD_ILLEGAL                0
        #define  MOD_UNKNOWN                1
        #define  MOD_DATA_INPUT             2
        #define  MOD_DATA_OUTPUT            3
        #define  MOD_ENCODER                4
        #define  MOD_DECODER                5
        #define  MOD_INTERLEAVER            6
        #define  MOD_DEINTERLEAVER          7
        #define  MOD_FRAME_FORMATTER        8
        #define  MOD_FRAME_EXTRACTOR        9
        #define  MOD_TIME_PERMUTE           10
        #define  MOD_TIME_DEPERMUTE         11
        #define  MOD_FREQUENCY_PERMUTE      12
        #define  MOD_FREQUENCY_DEPERMUTE    13
        #define  MOD_MODULATOR              14
        #define  MOD_DEMODULATOR            15
        #define  MOD_TRANSEC                16
        #define  MOD_SYNC_PROCESSOR         17
        #define  MOD_TIME_CONTROLLER        18
        #define  MOD_CRC_ENCODER            19
        #define  MOD_CRC_DECODER            20
        #define  MOD_ACCESS_PROCESSING      21
        #define  MOD_ACCESS_GENERATION      22
        #define  MOD_RESOURCE_ALLOCATOR     23
        #define  MOD_RESOURCE_REQUESTER     24
#define OFF_DATA_DESTINATION                                              5
        /* use definitions from OFF_DATA_SOURCE */
#define OFF_DATA_TYPE                                                     6
        #define  TYPE_ILLEGAL               0
        #define  TYPE_STANDARD              1
#define OFF_WORD_SIZE                                                     7
        #define  WORD_ILLEGAL               0
        #define  WORD_32_BITS               1
        #define  WORD_64_BITS               2
#define OFF_WORD_TYPE                                                     8
        /* use WORD_ILLEGAL from OFF_WORD_SIZE */
        #define  WORD_UNSIGNED              1
        #define  WORD_INTEGER               2
        #define  WORD_FLOAT                 3
#define OFF_PACKING_DENSITY                                               9
        #define  PACK_ILLEGAL               0
#define OFF_SYMBOL_SIZE                                                   10
        #define  SIZE_ILLEGAL               0
#define OFF_SYMBOL_MASK_LO                                                11
#define OFF_SYMBOL_MASK_HI                                                12
#define OFF_OUTPUT_FORMAT                                                 13
        #define  OUT_ILLEGAL                0
        #define  OUT_UNSPECIFIED            1
        #define  OUT_UNPACKED               1000
        #define  OUT_PACKED                 2000
#define OFF_SPECIAL_0                                                     14
        #define  SPEC_UNUSED                0
#define OFF_SPECIAL_1                                                     15
        /* as above */
```

## 8. Conclusion

The specification for a variable length data packet (including variable length header and data areas) was presented. There is sufficient versatility in the specification to allow the data packet to be used for all known intermodule data, and for new requirements .

All packets must have their header validated before use and the essential validations were described. A minimum implementation necessary for all modules was provided for initial development. The expansion capabilities of the specification were presented. Finally, coding examples including symbol definition file listing were shown.

It is likely that this specification will be further refined as requirements become firmer and the design of the simulator becomes better defined.

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

## DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

| | |
|---|---|
| 1. **ORIGINATOR** (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)<br>Defence Research Establishment Ottawa<br>Ottawa, Ontario<br>K1A 0Z4 | 2. **SECURITY CLASSIFICATION** (overall security classification of the document including special warning terms if applicable)<br>UNCLASSIFIED |

3. **TITLE** (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

MDR/Omni-band Reconfigurable Terminal: Data Packet Specification (U)

4. **AUTHORS** (Last name, first name, middle initial)

Addison, Robin D.

| | | |
|---|---|---|
| 5. **DATE OF PUBLICATION** (month and year of publication of document)<br>August 1998 | 6a. **NO. OF PAGES** (total containing information. Include Annexes, Appendices, etc.)<br>23 | 6b. **NO. OF REFS** (total cited in document)<br>0 |

7. **DESCRIPTIVE NOTES** (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.

DREO Technical Note

8. **SPONSORING ACTIVITY** (the name of the department project office or laboratory sponsoring the research and development. Include the address.

SST, Defence Research Establishment Ottawa
Ottawa, Ontario, K1A 0Z4

| | |
|---|---|
| 9a. **PROJECT OR GRANT NO.** (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)<br>5ca11 | 9b. **CONTRACT NO.** (if appropriate, the applicable number under which the document was written) |
| 10a. **ORIGINATOR'S DOCUMENT NUMBER** (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)<br>DREO TECHNICAL NOTE 98-009 | 10b. **OTHER DOCUMENT NOS.** (Any other numbers which may be assigned this document either by the originator or by the sponsor) |

11. **DOCUMENT AVAILABILITY** (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. **DOCUMENT ANNOUNCEMENT** (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). however, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

Unlimited Announcement

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

13. **ABSTRACT** (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

In 1997, Defence Research Establishment Ottawa and Communications Research Centre started research into an omni-band reconfigurable terminal. Such a terminal will enable soldiers to use a single terminal to communicate over any satellite communications or terrestrial link. Each terminal will support multiple standards and the first one to be implemented will be extremely high frequency medium data rate communications. Development begins with two hardware simulators: the payload simulator and the ground terminal simulator. Each simulator has a PC-based host to run the simulator, a digital chassis containing several digital signal processor boards to run the payload or terminal, and a radio frequency chassis to interface to the digital boards. Within the simulators, specifically on the digital signal processor boards, it is desirable to have a common data format for interchange between modules to simplify interfacing and reconfiguration.

This document contains the data packet format, data packet validation, description of the minimum implementation, expansion capabilities, coding excerpts and, the listing of necessary definitions for coding. The requirements of flexibility, simplicity and future expandability drove the design of the data packets. Since the data packet design precedes the detailed system design, future expandability was a key requirement. The data packet consists of a variable length header and a variable length data area. The header includes the details of the data storage as well as information about the data source and destination.

There is sufficient versatility in the specification to allow the data packet to be used for all known intermodule data, and for new requirements. It is likely that this specification will be further refined as requirements become firmer and the design of the simulator becomes better defined.

14. **KEYWORDS, DESCRIPTORS or IDENTIFIERS** (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

satellite communications
data packet
MDR