esa estec

european space agency
european space research
and technology centre

WSM/SH/400
Issue 2
March 1998

Local Time Management System Validation Board

Design Description

Prepared by S. Habinc

Control, Data and Power Division (TOS-ES)
Keplerlaan 1 - Noordwijk - The Netherlands
Mail address: Postbus 299 - 2200 AG Noordwijk - The Netherlands
Tel: +31-71-565 4722 - Telex: 39098 - E-mail: sandi@ws.estec.esa.nl - Fax: +31-71-565 4295

This page is left intentionally blank.

1 INTRODUCTION

1.1 Scope

This document describes the Local Time Management System (LTMS) validation board. The LTMS itself is not described in detail in this document, since such a description can be found in the Data Sheet (RD1) with which the reader is assumed to be acquainted.

The document provides a brief background to the LTMS device followed by an introduction to the LTMS time synchronisation mechanism (RD2). Thereafter, an overview of the LTMS validation board is given, followed by a detailed description of the Central Time Management System (CTMS) device which is part of the validation board. Finally, the validation board interfaces etc. are described in the included appendices.

1.2 Background

With the advent of packet telemetry, the time correlation between the generation of onboard data and its reception on ground has become more complex compared to previous use of fixed frame protocols. Packet telemetry allows dynamic bandwidth allocation schemes, buffering etc., making it flexible and adaptable. This however leads to an asynchronous delivery system which is no longer fully deterministic in that the sampling instant is not related to a position in the transfer frame. The task of determining the time when data were sampled or generated onboard is therefore not simple anymore. Hence, new methods are required for providing the users with time information related to the telemetry data stream.

The purpose of the Local Time Management System (LTMS) component is therefore to provide time coherence throughout the spacecraft without requiring processing power from the applications using it. It provides a time stamp facility for datation of events, an alarm clock, a pulse generator, a waveform generator and a stopwatch.

The LTMS was developed by IMEC (B) under the prime contractor Dornier Satellitensysteme (D) and is manufactured by MITEL Semiconductors (S) in their radiation hard process. The foundry (former ABB Hafo) will support the LTMS as an Application Specific Standard Product (ASSP), with a complete data sheet (RD1).

The LTMS is supplied in an 84 pin ceramic quad flat package. It is manufactured in the radiation hard CMOS/SOS5 process, and is guaranteed to withstand radiation up to a total dose of 100 kRad. It is latch-up immune and has a very low Single Event Upset sensitivity thanks to the Silicon On Sapphire (SOS) technology.

Direct questions regarding price and availability to the foundry (Int+ 46 8580 24500).

The development was funded by the European Space Agency under the Basic Technology Research Programme and was managed by the Microelectronics and Technology section at the European Space Research and Technology Centre.

1.3 Reference documents

RD1 *Local Time Management System*, Data Sheet MS13196, MITEL Semiconductor
RD2 *A Local Time Management System ASSP*, S. Redant, S. Habinc, ESA Workshop on Microelectronics for Satellite Applications, Noordwijk 1996
RD3 *Time Code Standards*, Consultative Committee for Space Data Systems CCSDS 301.0-B-2, Blue Book, Issue 2, April 1990
RD4 *4-255 Data Bus Protocol Extensions*, ESA PSS-04-256
RD5 *A Versatile Backplane Bus: VMEbus*, ANSI/IEEE Std 1014-1987

1.4 Acronyms and abbreviations

ASIC	Application Specific Integrated Circuit
ASSP	Application Specific Standard Product
CCSDS	Consultative Committee for Space Data Systems
CMOS	Complementary Metal Oxide Semiconductor
CTMS	Central Time Management System
CUC	CCSDS Unsegmented Code
DPLL	Digital Phase Locked Loop
EPROM	Electrically Programmable Read Only Memory
ERC32	Embedded Real-time Computer
ESA	European Space Agency
ESTEC	European Space Research and Technology Centre
ET	Elapsed Time
FPGA	Field Programmable Gate Array
IEEE	Institute of Electrical and Electronics Engineers
LSW	Least Significant Word
LTMS	Local Time Management System
MSW	Most Significant Word
SOS	Silicon On Sapphire
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

2 BASIC LTMS TIME SYNCHRONISATION

To provide the reader with an understanding of the purpose and operation of the LTMS validation board, the synchronisation between a CTMS and LTMS is recapitulate here.

The LTMS maintains a local copy of a central elapsed time (ET) reference and is typically embedded in the host application. The ET counter is the local time reference consisting of the fine ET counter (22 bits with sub-second weights) and the coarse ET counter (32 bits, with the least significant bit having a weight of 1 second). All time information sent to or provided by the LTMS is compliant to the CCSDS CUC format (RD3). The ET counter resolution depends on the operating mode of the LTMS. As a result of this, the least significant bit of the ET counter can have the weights 2^{-22}, 2^{-21}, 2^{-20} or 2^{-19} of a second, corresponding to a resolution between approximately 240 ns to 2 μs.

The central ET reference is assumed to be managed by the CTMS. Coherence between the central and the local ET references is maintained by means of synchronisation information distributed from the CTMS. The LTMS performs regular synchronisations with respect to the central ET reference using this information. The information provided by the CTMS to the LTMS comprises messages containing time codes, synchronisation markers and phase references.

Every second a message including a time code is broadcast from the CTMS to all LTMS components in the system. Each message is followed by a synchronisation marker which acts as a ‘mark’ to establish the instant at which the delivered time code is applied. The synchronisation instances occur at one second intervals and coincide with the increments of the units of seconds bit in the ET counter. No sub-second information needs to be included in messages sent by the CTMS.

The messages can be acquired through a bit serial interface or via a parallel microprocessor interface. In the serial operation (RD4), the messages are Manchester encoded and carry the synchronisation marker and phase references. The parallel interface is compatible with the ERC32 and MA31750 microprocessors, requiring little or no additional external hardware for connecting them together.

The CTMS provides the LTMS with phase references to which the clock driving the ET counter can be adjusted. The clock driving the LTMS ET counter does not need to originate from the CTMS and may thus be generated locally. This clock may be provided from external circuitry and should then be consistent with the desired ET resolution and be acting as the phase reference. Alternatively, the LTMS may derive it from a supplied 2^{24} Hz (≈16.77 MHz) crystal or clock signal. The built-in crystal oscillator requires only a crystal, two capacitors and a resistor to operate, and can also be used for driving external components. The internally derived clock can be slaved to the incoming phase references by either using the built-in or an external Digital Phase Locked Loop (DPLL).

The LTMS can also operate in stand alone operation without a central time reference. It then maintains its own time reference and no synchronisation is required. It is however possible to initialise the LTMS from time to time via the parallel microprocessor interface. This provides the capability to use the LTMS even when no CTMS is available in the system, making it backward compatible with some earlier equipment.

3 LTMS VALIDATION BOARD DESCRIPTION

The validation board comprises a LTMS device to be evaluated and a Central Time Management System (CTMS) which can be used as a test generator, etc. The CTMS is implemented in a Field Programmable Gate Array (FPGA). The CTMS is described more in detail in section 4. The board schematic is shown in appendix B.

The board allows the LTMS and the CTMS to be interfaced through a VMEbus P2 compliant connector (RD5). The connector provides a parallel microprocessor interface with a 16 bit wide data bus and a 7 bit wide address bus, separate select signals to the LTMS and CTMS, separate ready signals from the two devices, common read and write strobes, two interrupts from the CTMS, a rest signal to the CTMS, and finally a microprocessor clock for the LTMS.

The VMEbus connector also provides all signals related to the LTMS DPLL interface, Extension interface, user facilities and serial/parallel message transmission. Finally, the board is powered via the this connector according to the VMEbus standard. The VMEbus connector is described in appendix D. The validation board is of standard Euro board size, and a top view is shown in appendix C.

The board provides six 20 pin connectors suitable for interfacing to a HP logic analyser. Most of the signals on the board can be monitored this way. Amongst others, the Elapsed Time (ET) counter of the CTMS can be observed, showing the full ET Fine time (24 bits) and the least significant byte of the ET Coarse time (8 bits). The logic analyser connectors are described in appendix E.

For nominal operation, an onboard EPROM is used for setting up the Altera FPGA. The board also provides the capability for dynamic programming of the CTMS FPGA through a 10 pin connector. The Altera Flex programming connector is described in appendix G.

The board provides a connector and drivers for full duplex serial communication with the CTMS using the RS232 protocol. The protocol is currently not implemented in the CTMS, but could be used in the future. The RS232 connector is described in appendix F.

The board supports both a crystal oscillator circuit (connected to the CTMS) and a crystal (connected to the LTMS). It can operate with both clock sources, or with only one of them.

The board provides a separate power plane for the LTMS, allowing its voltage to be varied with respect to the surrounding logic and enables power dissipation measurements. Two connections are supplied between the LTMS power plane and the rest of the board. For a simple voltage drop, the connections can be made using diodes, as shown in appendix C.

The board features all necessary pull up/down resistors, and clock and signal terminations.

Finally, the board has a power-on LED and a reset switch.

4 CENTRAL TIME MANAGEMENT SYSTEM (CTMS)

The Central Time Management System (CTMS) FPGA implements all logic necessary to setup and run the LTMS on the validation board. The board can be run with both the CTMS and LTMS active; with only the LTMS active (although configured via the CTMS); or with only the CTMS active.

The CTMS provides the following main services:

- programmable central Elapsed Time (ET) counter, partially readable by the user;
- generation and transmission of serial messages and synchronisation markers, with parity calculation and Manchester encoding (serial operation);
- transmission of Pulse and Waveform fields (serial operation);
- interrupt driven Mission Parameter transmission (serial operation);
- generation of periodic synchronisation markers (parallel and stand alone operation);
- resetting of the LTMS;
- setting of programmable LTMS pins, supporting all operating modes;
- generation of LTMS bus clock and/or Xtal1 clock;
- tristating or driving of serial message and bus clock lines.

The CTMS also provides the following debug and error introduction capabilities:

- speed up or slow down of central ET;
- preset of central ET;
- driving the LTMS time facility inputs;
- tristating or driving the LTMS time facility inputs;
- latching of central ET on LTMS time facility output events;
- parity error insertion in serial message;
- stuck-at error insertion on bus clock or serial message line.

The CTMS registers are described in appendix A. The CTMS interface signals are described in appendix I. The CTMS pin list is described in appendix J. The CTMS pinout is shown in appendix K.

The CTMS is implemented in an Altera FPGA, type EPF81188, in a plastic 208 pin quad flat package. Refer to section 4.6 for a description of the FPGA booting alternatives. The CTMS was designed using VHDL and the Altera Max2Plus design software. The CTMS VHDL source code is listed in appendix L.

The rest of this section mainly describes the CTMS, but references to the validation board design are provided for sake of clarity when necessary.

Note that all signal names used in this section are related to those used in the CTMS design. For the connection between the CTMS, LTMS and various connectors, please refer to the electrical schematic of the validation board in appendix B.

Signals with the suffix “_N” are active low. i.e. active when at logical zero. Bit 0 is the least significant.

4.1 Clocking scheme

The CTMS is assumed to operate on a 2^{24} Hz clock provided on the Clk input (with the maximum frequency being 20 MHz). The Clk clocks the programmable central Elapsed Time (ET) counter, with a fixed resolution of 2^{-24} Hz. The ET counter in the CTMS is divided in two parts: the ET Coarse counter (32 bits, with the least significant bit having a weight of 1 second) and the ET Fine counter (24 bits with sub-second weights). The CTMS ET counter matches the LTMS ET counter, but provides slightly higher resolution.

The board provides space for a clock oscillator part, and for a crystal oscillator circuitry to be interfaced with the LTMS. The clock input configurations for the CTMS and LTMS are controlled via two switches as shown in figure 1 and figure 2 respectively. The location of the two switches on the board can be found in appendix C. For the LTMS clock inputs, please refer to RD1.

The board can operate with both or either clock source, supporting a tightly coupled system for logical validation or a loosely coupled system for the validation of time coherence and system level synchronicity.

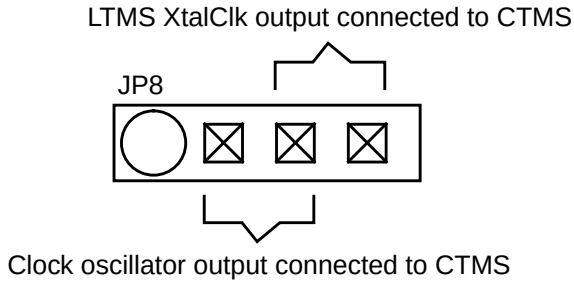


Figure 1: *CTMS clock configuration.*

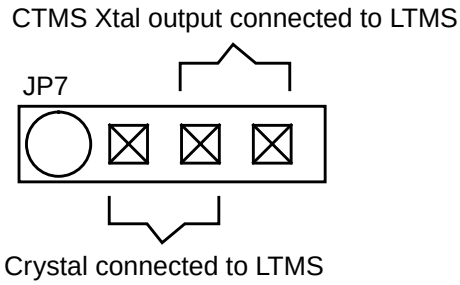


Figure 2: *LTMS clock configuration.*

The frequency of the CTMS BusClk output is controlled via the Mode register (cf. table 2). It is tristated at reset and can be enabled via the Debug register (cf. table 3). Since it is possible to tristate the BusClk clock, it can be provided via the VMEbus interface from an external source, such as a second LTMS validation board. The output can also be fixed for the purpose of error introduction via the Debug register (cf. table 3).

The CTMS Xtal output is only toggled when XtalAux is enabled (cf. table 2). It cannot be set to a fixed value as for the BusClk output, instead, time incoherence between the CTMS and the LTMS can be introduced via the ET counter (cf. section 4.5).

4.2 Microprocessor interface

All CTMS registers are accessible via the microprocessor interface. Most of them are both read and writeable. It is an asynchronous interface, allowing both MA31750 and ERC32 microprocessors to accesses it. All registers are cleared to 0000h at reset, except for address 0Bh which is 0001h after a reset (cf. table 1).

During write accesses, the data and address buses are latched asynchronously on the rising Wr_N edge, while CS_N is asserted and Rd_N is de-asserted. Data is synchronously written internally to the CTMS when Wr_N has been sampled as de-asserted. To guarantee a write access, Wr_N should be asserted for at least two Clk periods. Note that CS_N must be asserted while the Wr_N strobe is asserted, a write access is not completed correctly if the CS_N strobe is de-asserted before the Wr_N is de-asserted.

During read accesses, the data bus will be asynchronously driven when Rd_N and CS_N are asserted, while Wr_N is de-asserted. Note that CS_N does not need to be de-asserted between accesses.

The LTMS and the CTMS are configured via the Mode register (cf. table 2). Most of the configuration inputs of the LTMS are controlled form this register, and at the same time configuring the operating mode of the CTMS, e.g. serial or parallel operation.

Only address bits 3 down to 0 are used for the decoding of accesses to the CTMS. Address bits 5 down to 4 are ignored. Note that address bits 5 and 6 should be held statically at logical zero to allow proper LTMS operation, since being connected to the LTMS production test pins.

4.3 Serial operation

The CTMS transmits a CTMS message to the LTMS via the SerMsg output once every second when in serial operation. The message can either be automatically generated (which is the nominal case) or be provided by the user via the microprocessor interface. Independently of whether generated automatically or provided, the CTMS message is transmitted according to the protocol specified in RD1, i.e. with a synchronisation marker attached, calculated parity bits and being Manchester encoded.

When being provided, the message should be written to Message registers 1 to 9 by the user (cf. table 1) via the microprocessor interface. This should be performed well in time before the message transmission. The CTMS output Irq0 is asserted on every half second transition, and de-asserted on every integer second transition, being suitable for interrupting a microprocessor to write the Message registers.

The definition of the CTMS Message registers can be found in RD1 and is thus the same as for the LTMS. The CTMS has to be informed when a valid message is available from the user. This is done via the Mode register (cf. table 2), where one of bits 14-12 should be set. If a Pulse and/or a Waveform field is to be transmitted, the corresponding control bits should also be set in the Mode register (bit 14 and bit 13, respectively). Note that there is no automatic correlation between these bits and the corresponding flags in the Message

registers. When the message transmission begins, the previously set bits in the Mode register will be cleared when the CTMS ET Fine counter reaches a count of 000000h.

For the three first words of the message, an internal register set is used for serial shifting. Message registers 1 to 3 are therefore never shifted and can be read back correctly both prior to and after a message transmission. However, Message registers 4 to 9 are used as serial shift registers during message transmission, and have therefore only been made writeable since their contents cannot be properly retained after being transmitted.

Prior to the integer second transition, a synchronisation marker is transmitted on the SerMsg output. At the time of the integer second transition in the CTMS ET counter, several registers are updated and the transmission of the serial message begins. The subsequent order of events depends on whether or not a new message has been provided by the user. Firstly, just at the time of the integer second transition the least significant byte of the Pending ET register pair is copied to the ETCoarse output.

In case bit 12 is was set (Initialisation), the Message register 2 and 3 contents are copied to the Pending ET registers and to the internal register set. The Message register 1 contents are also copied to the internal register set. This internal register set, together with Message registers 4 and 5 (when Mode register bit 14 is set), and/or together with Message registers 6 to 9 (when Mode register bit 13 is set), form a serial shift register from where the message is shifted out on the SerMsg output, after being properly encoded.

In case none of bits 14-12 were set, the Pending ET registers is instead incremented by one, and the result is copied to the internal register set. This occurs just after the integer second transition. The part of the internal register set forming the CTMS status word is cleared to 0000h. The contents of the internal register set are shifted out on the SerMsg output, after being properly encoded. A nominal three word message is thus formed, with the Pulse, Waveform and Initialisation flags not being set. This allows the CTMS to continuously transmit messages, without requiring any user interventions.

In case either or both bits 14 and 13 were set while bit 12 was not, the Message register 1 contents are copied to the internal register set and the Pending ET register is incremented as above. This allows sending Pulse and/or Wave fields without upsetting the ET counter.

After the ET integer second transition and the message transmission, the CTMS ETCoarse output will show the current valid ET. The Pending ET registers will hold the ET that will become valid on the next integer second transition, which can be an increment of the ETCoarse output value or an arbitrary value that was provided by the user. Message registers 4 to 9 will not retain any valid data if they were shifted during the transmission. Data not shifted will however be retained, allowing off-line programming of the registers while normal message transmission is ongoing.

During the transmission of the message, the CTMS output Irq1 is asserted when there is only one word left to be transmitted and is de-asserted eight bits later. The output is suitable for interrupting a microprocessor to write an optional Mission Parameter word to Message register 1 (cf. table 1) and to indicate the availability of such a word by setting bit 15 in the Mode register (cf. table 2). This should be done before the last word is completely transmitted, the available time being mode dependent as derived from RD1.

The Mode register bit is automatically cleared when a new optional Mission Parameter word has been accepted. The Message register 1 contents are copied to the internal register set, from where they are shifted out.

The above process can be repeated, allowing additional words to be transmitted. Message register 1 is never shifted and can therefore be correctly read back prior to and during Mission Parameter transmission.

It is also possible to provide five Mission Parameter words using Message registers 4 to 9. Bit 14 and 13 in the Mode register should then both be set prior to transmission, with the corresponding flags in Message register 1 being zero (i.e. the LTMS will recognise the transmitted data as Mission Parameters instead of Pulse or Waveform fields. For additional words, the interrupt driven process described above can be used.

4.4 Parallel operation

In parallel or stand alone operation, the Message is written directly to the LTMS from the microprocessor. The message is qualified by a synchronisation marker on the CTMS SerMsg output (effectively a pulse), indicating the ET integer second transition. The SerMsg output is asserted one Clk period before the BusClk rising edge, and is de-asserted one Clk period after the same edge. The qualified rising BusClk edge ‘marks’ the integer second transition.

The periodic generation of the synchronisation marker is performed automatically by the CTMS when in parallel operation.

In stand alone operation, the synchronisation marker is optionally required by the LTMS and can be initiated by setting bit 15 of the Mode register (cf. table 2), which should be done at least two Clk periods before the expected synchronisation marker. The CTMS Irq0 output is suitable for scheduling such operations. The register bit is automatically cleared after a synchronisation marker has been generated.

Even though the Message is not transmitted from the CTMS, the Pending ET registers and Message registers 1 to 3 can be used for maintaining the central ET counter. As for the serial operation, the CTMS ETCoarse output will show the current valid ET and the Pending ET registers will hold the ET that will become valid on the next integer second transition. The latter can be read via the microprocessor interface and written to the LTMS, thus off-loading the user from implementing the counter in software.

If the ET count needs to be modified, the user can write the new value to Message registers 2 and 3, and indicate that a the new values is available through bit 12 of the Mode register. The new value will then come into effect not on the next integer second transition, but on the one following that, i.e. in the same way as it is done for serial operation. The new value is thus copied to the Pending ET registers on the first transition, which in turn is copied to the ETCoarse output on the second transition. Bit 12 of the Mode register is automatically cleared as soon as the Pending ET registers are updated (which is performed earlier than in serial operation).

4.5 Debugging

To allow error or time skew introduction between the CTMS and the LTMS, the CTMS provides some special debugging capabilities which are all controlled through the CTMS Debug register (cf. table 3). Some capabilities are described in sections related to the nominal functionality of the CTMS and are not repeated here (cf. section 4.1).

The ET counter implemented in the CTMS can be made to speed up or to slow down through the Debug register. A speed up makes the counter to once increment by a 2^{-23} Hz step instead of a nominal 2^{-24} Hz step. A slow down makes the counter to skip one nominal increment. The ET counter cannot be reset to zero, instead, the ET Fine counter can be preset to FFE000h, which is just before the start of the (slowest) synchronisation marker transmission in serial operation. The ET Coarse counter can be reset via normal Message handling. The corresponding control bit in the Debug register is cleared when one of the above actions is performed. These combined capabilities can be used for purposely introducing skew between the CTMS and LTMS ET counters.

The SerMsg output is tristated at reset and can be enabled via the Debug register. The output can also be held at logical zero for the purpose of error introduction. Parity errors can be introduced on the SerMsg output, effectively inverting the parity bit.

Some of the dynamic LTMS inputs can be controlled by the CTMS via the Debug register. The CTMS SWStart, SWEvent and ETStrobe outputs can be driven to a logical zero or one. The outputs are tristated at reset and can be enabled via the Debug register. This allows an external unit to be connected to the LTMS inputs via the VMEbus connector.

The LTMS input PulseIn can be controlled by the CTMS via the Debug register. It can be forced to logical zero or one, or be connected to the PulseOut or ETAlarm input. The output is tristated at reset and can be enabled together with the three signals above via the Debug register.

The source for the Irq1 output is controlled via the Debug register. It can be forced to logical zero or one, connected to the ETAlarm input, or be used in serial operation for announcing that a the CTMS is ready to receive an optional Mission Parameter word.

The reset input of the LTMS is controlled via the Debug register. The CTMS output is asserted (at logical zero) during and after a reset, and should be de-asserted after the relevant LTMS configuration has been set via the Mode register.

The CTMS ET counter can be latched into the ET Latch registers (cf. table 4 and table 5) on the occurrence of events on the ETAlarm, WaveOut and PulseOut inputs, or when the CTMS ETStrobe output is asserted. The source of the latest event is also stored in the ET Latch registers. The latched values are overridden by a subsequent event, implementing a last-event register.

4.6 CTMS FPGA boot

The CTMS FPGA can either be booted from an onboard EPROM (part U3 in appendix C) or via the Altera Flex programming connector (cf. table 12) which is intended for validation board debugging and prototyping purposes. The means for selection are shown in figure 3. Refer to Altera documentation for further information on FPGA configuration procedures via the Flex connector.

For nominal operation, the selector should be set to EPROM booting and the validation board should simply be powered up, automatically configuring the CTMS.

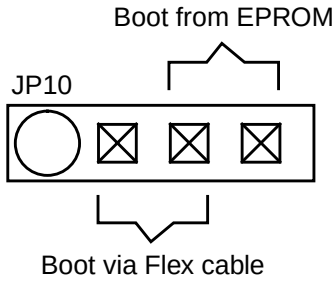


Figure 3: *CTMS FPGA boot configuration.*

APPENDIX A: CTMS REGISTERS

A.1 CTMS register address map

Register name	Address	ERC32	Read/Write	Length	Remark
CTMSMode	00h	00h	R/W	16 bit	Operating mode register
CTMSMsg1	01h	04h	R/W	16 bit	Status / Mission Parameter
CTMSMsg2	02h	08h	R/W	16 bit	ET Coarse MSW
CTMSMsg3	03h	0Ch	R/W	16 bit	ET Coarse LSW
CTMSMsg4	04h	10h	W	16 bit	Pulse MSW
CTMSMsg5	05h	14h	W	8 bit	Pulse LSW
CTMSMsg6	06h	18h	W	16 bit	Wave Coarse Length
CTMSMsg7	07h	1Ch	W	16 bit	Wave Fine Length MSW
CTMSMsg8	08h	20h	W	8 bit	Wave Fine Length LSW
CTMSMsg9	09h	24h	W	16 bit	Wave Polarity
PendingETCoarseMSW	0Ah	28h	R	16 bit	CTMS ET Coarse MSW
PendingETCoarseLSW	0Bh	2Ch	R	16 bit	CTMS ET Coarse LSW
Debug	0Ch	30h	R/W	16 bit	Debug register
ETLatch0	0Dh	34h	R	16 bit	ET Coarse & ET Fine
ETLatch1	0Eh	38h	R	16 bit	ET Fine & error indication

Table 1: *CTMS register address map*

A.2 CTMS mode register, 00h (00h ERC32)

Bit number	Description
15	Serial operation: send one Mission Parameter word
	Stand alone operation: enable single shot marker at next tick
	(reset after action for both operating modes)
14	Send Pulse field (reset after message being sent)
13	Send Waveform field (reset after message being sent)
12	ET Initialisation - send user specified ET Coarse field (reset after message being sent)
11	MProc line
10 downto 8	ClkFreq(2:0) lines
7 downto 6	GoThr(1:0) lines
5 downto 4	ETThr(1:0) lines
3	PFGMode line
2	AuXtal line
1	SerEnable line
0	CTMsg line

Table 2: *CTMS mode register, 00h (00h ERC32)*

A.3 CTMS Debug register, 0Ch (30h ERC32)

Bit number	Description		
15	Preset ET Fine time (reset after action)		
14	Increment ET Fine time by one (reset after action)		
13	Decrement ET Fine time by one (reset after action)		
12	Introduce parity errors		
11 downto 10	Select Irq1 source	00	Force zero
		01	Ready for an optional Mission Parameter word
		10	PulseOut
		11	ETAlarm
9 downto 8	Select PulseIn source	00	Force zero
		01	Force one
		10	PulseOut
		11	ETAlarm
7	Drive LTMS/CTMS lines (BusClk, SerMsg)		
6	Block SerMsg line		
5	Block BusClk line		
4	Drive user lines (ETStrobe, SWEvent, SWStart, PulseIn)		
3	SWStart line		
2	SWEvent line		
1	ETStrobe line		
0	ExRst_N line (LTMS in reset at power up)		

Table 3: *CTMS Debug register, 0Ch (30h ERC32)*

A.4 CTMS ETLatch registers, 0Dh and 0Eh (34h and 38h ERC32)

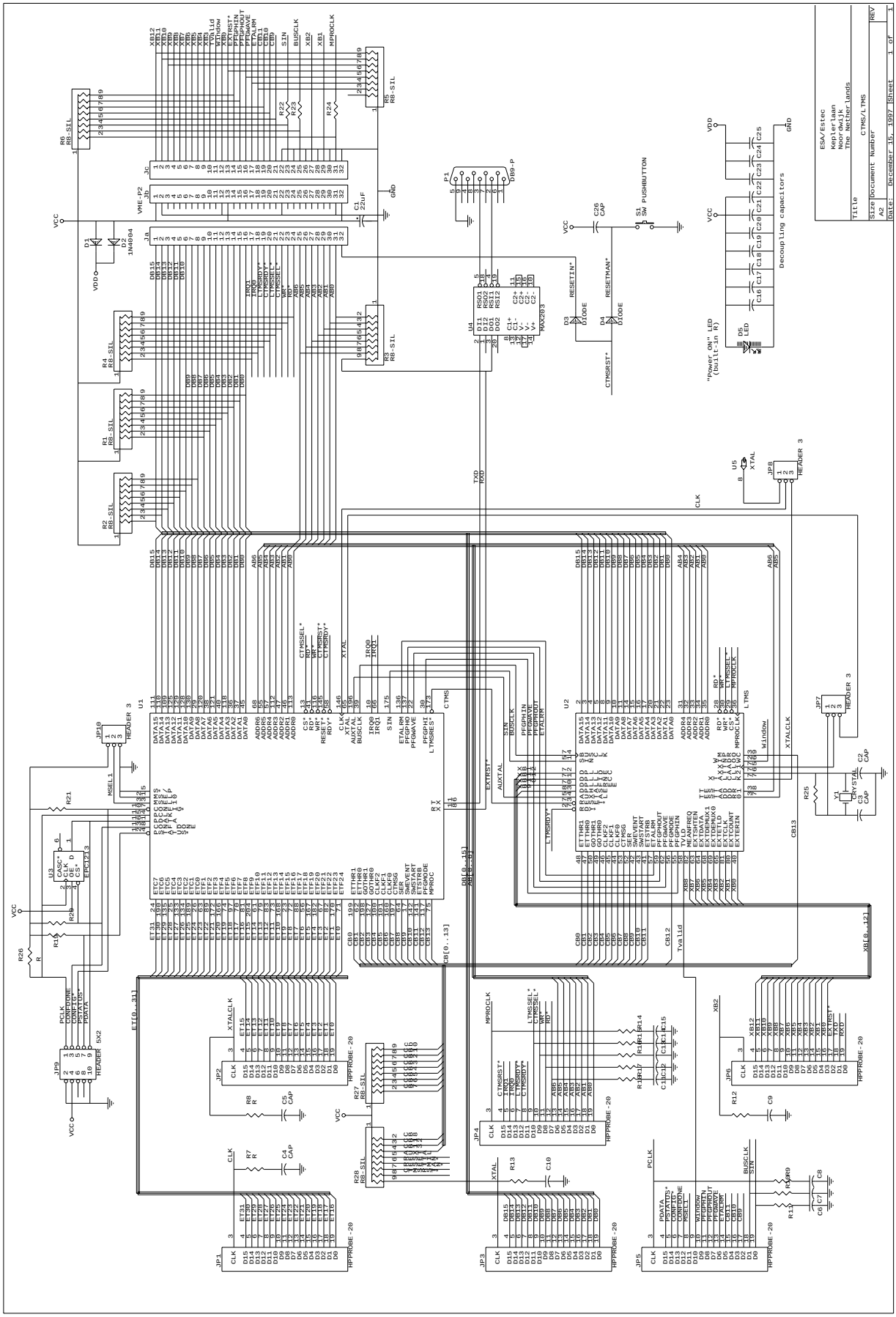
Bit number	Description
15 to 8	ET Coarse time bits 7 downto 0
7 to 0	ET Fine time bits 1 to 8

Table 4: *ETLatch0 register, ODh (34h ERC32)*

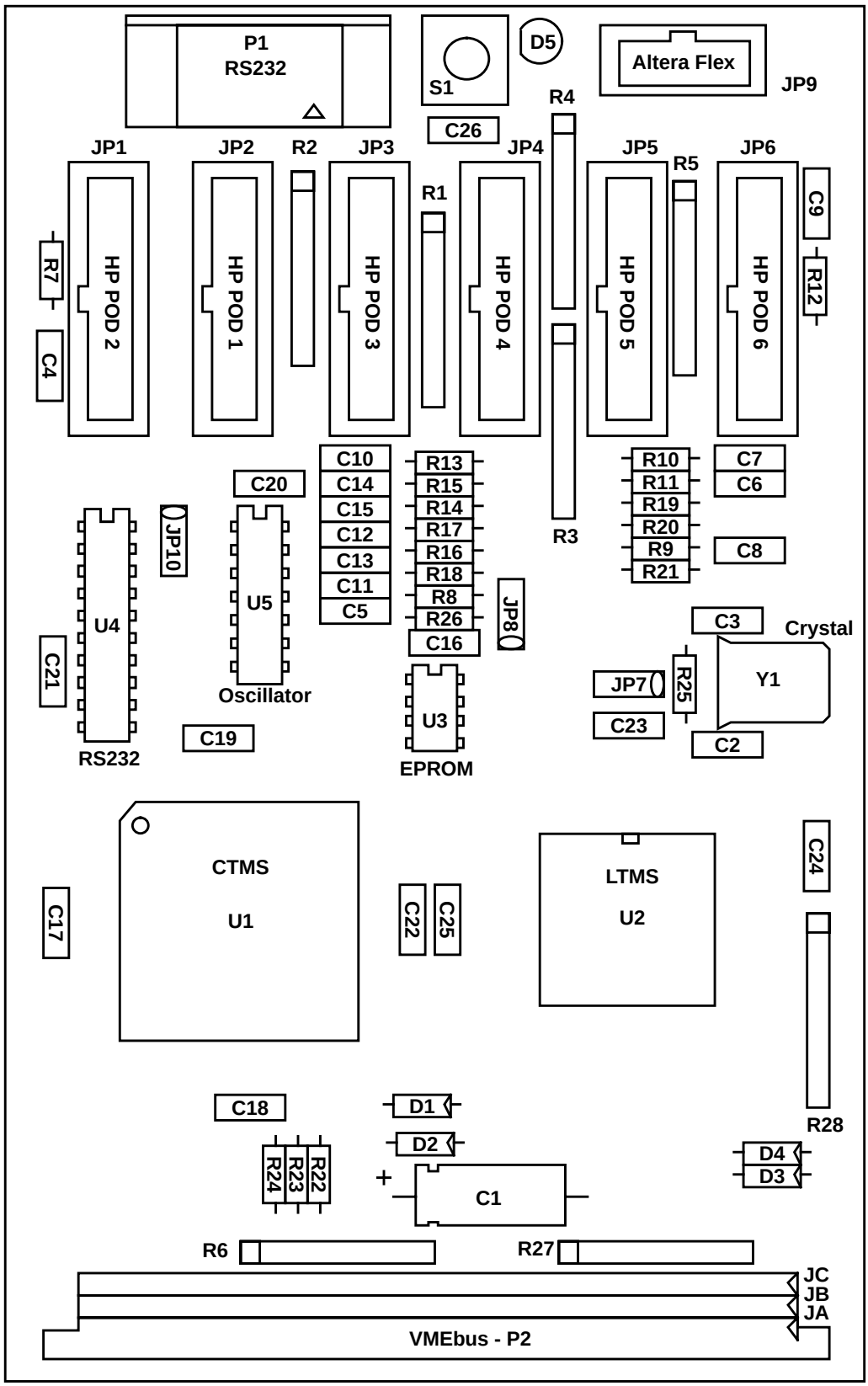
Bit number	Description		
15 downto 2	ET Fine time bits 9 to 22		
1 downto 0	Trigger source	00	ET Strobe generated from CTMS
		01	ET Alarm detected from LTMS
		10	PFG Wave out detected from LTMS
		11	PFG Pulse out detected from LTMS

Table 5: *ETLatch1 register, OEh (38h ERC32)*

APPENDIX B: LTMS VALIDATION BOARD SCHEMATIC



APPENDIX C: LTMS VALIDATION BOARD TOP VIEW



APPENDIX D: VMEBUS P2 CONNECTOR

Pin number	I/O	Type (in/out)	Name	LTMS name	Remark
P2-A:					
1	Bidir	CMOS/TTL	Data15		Pull up
2	Bidir	CMOS/TTL	Data14		Pull up
3	Bidir	CMOS/TTL	Data13		Pull up
4	Bidir	CMOS/TTL	Data12		Pull up
5	Bidir	CMOS/TTL	Data11		Pull up
6	Bidir	CMOS/TTL	Data10		Pull up
7	Bidir	CMOS/TTL	Data9		Pull up
8	Bidir	CMOS/TTL	Data8		Pull up
9	Bidir	CMOS/TTL	Data7		Pull up
10	Bidir	CMOS/TTL	Data6		Pull up
11	Bidir	CMOS/TTL	Data5		Pull up
12	Bidir	CMOS/TTL	Data4		Pull up
13	Bidir	CMOS/TTL	Data3		Pull up
14	Bidir	CMOS/TTL	Data2		Pull up
15	Bidir	CMOS/TTL	Data1		Pull up
16	Bidir	CMOS/TTL	Data0		Pull up
17	Out	CMOS/TTL	Irq1		Pull up
18	Out	CMOS/TTL	Irq0		Pull up
19	Out	CMOS	LTMS_Rdy_N	Ready_Bar	Pull up
20	Out	TTL	CTMS_Rdy_N		Pull up
21	In	CMOS	LTMS_CS_N	CS_Bar	Pull up/Terminate
22	In	TTL	CTMS_CS_N		Pull up/Terminate
23	In	CMOS	Wr_N	Wr_Bar	Pull up/Terminate
24	In	CMOS	Rd_N	Rd_Bar	Pull up/Terminate
25	In	CMOS	Addr6	TestAddr1	Pull down - Tie to zero in normal operation
26	In	CMOS	Addr5	TestAddr0	Pull down - Tie to zero in normal operation
27	In	CMOS	Addr4	Ad4	Pull down
28	In	CMOS	Addr3	Ad3	Pull down
29	In	CMOS	Addr2	Ad2	Pull down
30	In	CMOS	Addr1	Ad1	Pull down
31	In	CMOS	Addr0	Ad0	Pull down
32	In	CMOS	Reset_N	(External reset to CTMS only)	Pull up

Table 6: *VMEbus P2 connector - row A*

Pin number	I/O	Type (in/out)	Name	LTMS name	Remark
P2-B:					
1, 13, 32		PWR	VCC	Power	
2, 12, 22, 31		PWR	GND	Ground	
others			N/C	Not connected	
P2-C:					
1	In	CMOS	DPLLDec		Pull down
2	In	CMOS	DPLLInc		Pull down
3	In	CMOS	DPLLFree		Pull down
4	Out	CMOS	DPhase		
5	Out	CMOS	MeanFreq		
6	Out	CMOS	ExtShtEn		
7	Out	CMOS	ExtData		
8	Out	CMOS	ExtDemux1		
9	Out	CMOS	ExtDemux0		
10	Out	CMOS	ExtETLd		
11	Out	CMOS	TVlid		
12	Out	CMOS	Window		
13	In	CMOS	ExtErIn		Pull down
14	Out	TTL	ExRst_N	ExtRst_Bar	Pull up
15	Bidir	CMOS/TTL	PulseIn	PFGPhIn	Pull up
16	Out	CMOS	PulseOut	PFGPhOut	Pull down
17	Out	CMOS	WaveOut	PFGWave	Pull down
18	Out	CMOS	ETAlarm	ETAlrm	Pull down
19	Bidir	CMOS/TTL	ETStrobe	ETStrb	Pull up
20	Bidir	CMOS/TTL	SWStart		Pull up
21	Bidir	CMOS/TTL	SWEvent		Pull up
22		PWR	GND		
23	Bidir	CMOS/TTL	SerMsg	SIn	Pull up/Terminate
24		PWR	GND		
25	Bidir	CMOS/TTL	BusClk		Pull up/Terminate
26		PWR	GND		
27	Out	CMOS	ExtClk		
28		PWR	GND		
29	Out	CMOS	ExtCount		
30		PWR	GND		
31	In	CMOS	MProClk		Pull up/Terminate
32		PWR	GND		

Table 7: *VMEBus P2 connector - rows B and C*

APPENDIX E: HP LOGIC ANALYZER PODS

Pin number	Name	Comment	Remark
JP1:	Connect to HP logic analyser pod 2		
D15	ETCoarse7		
D14	ETCoarse6		
D13	ETCoarse5		
D12	ETCoarse4		
D11	ETCoarse3		
D10	ETCoarse2		
D9	ETCoarse1		
D8	ETCoarse0		
D7	ETFine1		
D6	ETFine2		
D5	ETFine3		
D4	ETFine4		
D3	ETFine5		
D2	ETFine6		
D1	ETFine7		
D0	ETFine8		
CLK (K)	Clk	(To CTMS)	Terminate
JP2:	Connect to HP logic analyser pod 1		
D15	ETFine9		
D14	ETFine10		
D13	ETFine11		
D12	ETFine12		
D11	ETFine13		
D10	ETFine14		
D9	ETFine15		
D8	ETFine16		
D7	ETFine17		
D6	ETFine18		
D5	ETFine19		
D4	ETFine20		
D3	ETFine21		
D2	ETFine22		
D1	ETFine23		
D0	ETFine24		
CLK (J)	XtalClk	(From LTMS)	Terminate

Table 8: *HP pods 2 and 1*

Pin number	Name	Comment	Remark
JP3:	Connect to HP logic analyser pod 3		
D15	Data15		Pull up
D14	Data14		Pull up
D13	Data13		Pull up
D12	Data12		Pull up
D11	Data11		Pull up
D10	Data10		Pull up
D9	Data9		Pull up
D8	Data8		Pull up
D7	Data7		Pull up
D6	Data6		Pull up
D5	Data5		Pull up
D4	Data4		Pull up
D3	Data3		Pull up
D2	Data2		Pull up
D1	Data1		Pull up
D0	Data0		Pull up
CLK (L)	Xtal	(From CTMS to LTMS)	Pull up/Terminate
JP4:	Connect to HP logic analyser pod 4		
D15	CTMS_Reset_N	(Push button)	Pull up
D14	Irq1		Pull up
D13	Irq0		Pull up
D12	LTMS_Rdy_N	Ready_Bar	Pull up
D11	CTMS_Rdy_N		Pull up
D10	LTMS_CS_N	CS_Bar	Pull up/Terminate
D9	CTMS_CS_N		Pull up/Terminate
D8	Wr_N	Wr_Bar	Pull up/Terminate
D7	Rd_N	Rd_Bar	Pull up/Terminate
D6	Addr6	TestAddr1	Pull down - Tie to zero in normal operation
D5	Addr5	TestAddr0	Pull down - Tie to zero in normal operation
D4	Addr4	Ad4	Pull down
D3	Addr3	Ad3	Pull down
D2	Addr2	Ad2	Pull down
D1	Addr1	Ad1	Pull down
D0	Addr0	Ad0	Pull down
CLK (M)	MProClk	(To LTMS from VMEbus)	Pull up/Terminate

Table 9: *HP pods 3 and 4*

Pin number	Name	Comment	Remark
JP5:	Connect to HP logic analyser pod 5		
D15	DATA0	(Altera)	
D14	nSTATUS	(Altera)	Pull up 1 kOhm
D13	nCONFIG	(Altera)	Pull up 1 kOhm
D12	CONF_DONE	(Altera)	Pull up 1 kOhm
D11	MSEL1	(Altera)	
D10	TVlid		
D9	Window		
D8	PulseIn	PFGPhIn	Pull up
D7	PulseOut	PFGPhOut	Pull down
D6	WaveOut	PFGWave	Pull down
D5	ETAlarm	ETAlrm	Pull down
D4	ETStrobe	ETStrb	Pull up
D3	SWStart		Pull up
D2	SWEvent		Pull up
D1	BusClk		Pull up/Terminate
D0	SerMsg	SIn	Pull up/Terminate
CLK (N)	DCLK	(Altera)	Pull up 1 kOhm/Terminate
JP6:	Connect to HP logic analyser pod 6		
D15	DPLLDec		Pull down
D14	DPLLInc		Pull down
D13	DPLLFree		Pull down
D12	DPhase		
D11	MeanFreq		
D10	ExtShtEn		
D9	ExtData		
D8	ExtDemux1		
D7	ExtDemux0		
D6	ExtETLd		
D5	ExtClk		
D4	ExtCount		
D3	ExtErIn		Pull down
D2	ExRst_N	ExtRst_Bar	Pull up
D1	Tx		
D0	Rx		Pull down
CLK (P)	ExtClk		Terminate

Table 10: *HP pods 5 and 6*

APPENDIX F: RS232 D-SUB 9 CONNECTOR

Pin number	I/O	Name	Remark
2	In	Rx-Rs232	(From Max203)
3	Out	Tx-Rs232	(To Max203)
5		GND	

Table 11: *RS232 D-SUB 9 connector*

APPENDIX G: FLEX PROGRAMMING CONNECTOR

Pin number	I/O	Name	Remark
1	In	DCLK	
2		GND	
3	Bidir	CONF_DONE	
4		VCC	
5	In	nCONFIG	
6		N/C	not connected
7	Bidir	nSTATUS	
8		N/C	not connected
9	In	DATA0	
10		GND	

Table 12: *FLEX programming connector*

APPENDIX H: LTMS VALIDATION BOARD BILL OF MATERIALS

Quantity	Reference	Part	Value
1	C1	Capacitor	22 µF
2	C2, C3	Capacitor	27 pF
12	C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15	Capacitor	33 pF
10	C16, C17, C18, C19, C20, C21, C22, C23, C24, C25	Capacitor	0.1 uF
1	C26	Capacitor	2.7 nF
2	D1, D2	Diode	1N4004
2	D3, D4	Diode	1N914
1	D5	LED	HLMP3600
6	JP1, JP2, JP3, JP4, JP5, JP6	Header 10X2	20 pin
3	JP7, JP8, JP10	Header 3X1	3 pin
1	JP9	Header 5X2	10 pin
1	P2-JA	VMEbus - A	DIN 41612 96 pin solder plug
	P2-JB	VMEbus - B	
	P2-JC	VMEbus - C	
1	P1	DB9-P	
8	R1, R2, R3, R4, R5, R6, R27, R28	R8-SIL	4.7 kOhm
2	R7, R12	R	339 Ohm
4	R19, R20, R21, R26	R	1 kOhm
3	R22, R23, R24	R	120 Ohm
10	R8, R9, R10, R11, R13, R14, R15, R16, R17, R18	R	0 Ohm
1	R25	R	1 MOhm
1	S1	Push-button	
1	U1	EPF81188	CTMS
1	U2	MS13196	LTMS
1	U3	EPC1213	Serial PROM
1	U4	MAX203	RS232
1	U5	Oscillator	2^{24} Hz
1	Y1	Crystal	2^{24} Hz

Table 13: *Bill of Materials*

APPENDIX I: CTMS INTERFACE DESCRIPTION

Signal name	I/O	Description	Remark
Clk	in	System clock 2**24 Hz	Terminate
Reset_N	in	Asynchronous reset	Pull up
Serial interface			
Rx	in	RS232 Receiving	Pull down
Tx	out	RS232 Transmitting	
uP interface			
CS_N	in	Chip select	Pull up/Terminate
Wr_N	in	Write strobe	Pull up/Terminate
Rd_N	in	Read strobe/direction	Pull up/Terminate
Rdy_N	out	Ready signal	Pull up
Addr(6:0)	in	Address bus	Pull down x 7
Data(15:0)	inout	Data bus	Pull up x 16
Irq0	out	Half second interrupt	Pull up
Irq1	out	LTMS interrupt	Pull up
CTMS/LTMS interface and LTMS mode settings			
BusClk	out	Bus clock	Pull up/Terminate
SerMsg	out	Message and synchronisation	Pull up/Terminate
SerEnable	out	Serial/Parallel operation	Pull up
AuXtal	out	Xtal available	Pull up
ClkFreq(2:0)	out	Bus clock frequency	Pull up x 3
PFGMode	out	PFG mode	Pull up
CTMsg	out	CTMS message mode	Pull up
ETThr(1:0)	out	ET threshold	Pull up x 2
GoThr(1:0)	out	Go threshold	Pull up x 2
MProc	out	MA31750/ERC32	Pull up
LTMS interface			
ExRst_N	out	External reset	Pull up
Xtal	out	Auxiliary clock	Pull up/Terminate
ETStrobe	out	Elapsed time strobe	Pull up
ETAlarm	in	Elapsed time alarm	Pull down
SWEvent	out	Stop-watch event	
SWStart	out	Stop-watch start	
PulseIn	out	PFG wave phase in	
PulseOut	in	PFG pulse out	
WaveOut	in	PFG wave out	
CTMS ET Counter output			
ETCoarse(7:0)	out	ET Coarse	
ETFine(1:24)	out	ET Fine	

Table 14: *CTMS interface*

APPENDIX J: CTMS PIN LIST

Number	Name	Number	Name	Number	Name	Number	Name	Number	Name
10	Irq0	45	Data0	73	ETFine13	117	PFGMode	160	ClkFreq0
13	CS_N	46	Addr1	74	ETFine5	118	Data3	166	ETFine4
16	Tx	47	Addr2	75	MProc	120	Data7	167	ETFine19
17	SWEvent	55	Addr5	79	ETFine11	121	Data5	168	ETFine14
18	Rx	56	ETFine18	81	ETFine8	125	Data12	170	ETFine23
22	WaveOut	57	Addr4	82	ETFine22	127	GoThr0	172	ETFine3
23	ETFine15	58	Rdy_N	83	ETFine12	128	Data10	173	ExRst_N
24	ETCoarse7	61	ETFine10	88	ETFine17	129	Data11	175	SerMsg
25	ETCoarse4	62	ETThr0	89	ETFine2	130	Data9	182	ETFine20
26	ETCoarse0	63	ETFine1	97	ETFine6	133	ETCoarse3	189	ETCoarse1
29	Data8	64	SerEnable	100	ClkFreq2	134	ETCoarse2	190	ETCoarse6
30	PulseIn	65	Xtal	101	ClkFreq1	135	ETCoarse5	196	AuXtal
36	Data2	66	Irq1	109	Data13	136	ETAlarm	197	CTMsg
37	Data1	67	ETFine21	110	Data14	137	PulseOut	198	GoThr1
38	Data6	68	Addr6	111	Data15	141	ETStrobe	199	ETThr1
39	BusClk	70	ETFine7	112	Addr3	142	SWStart	204	ETFine9
40	Data4	71	ETFine24	113	Addr0	145	Reset_N		
41	Rd_N	72	ETFine16	116	Wr_N	146	Clk		

Table 15: *CTMS signals in pin number order*

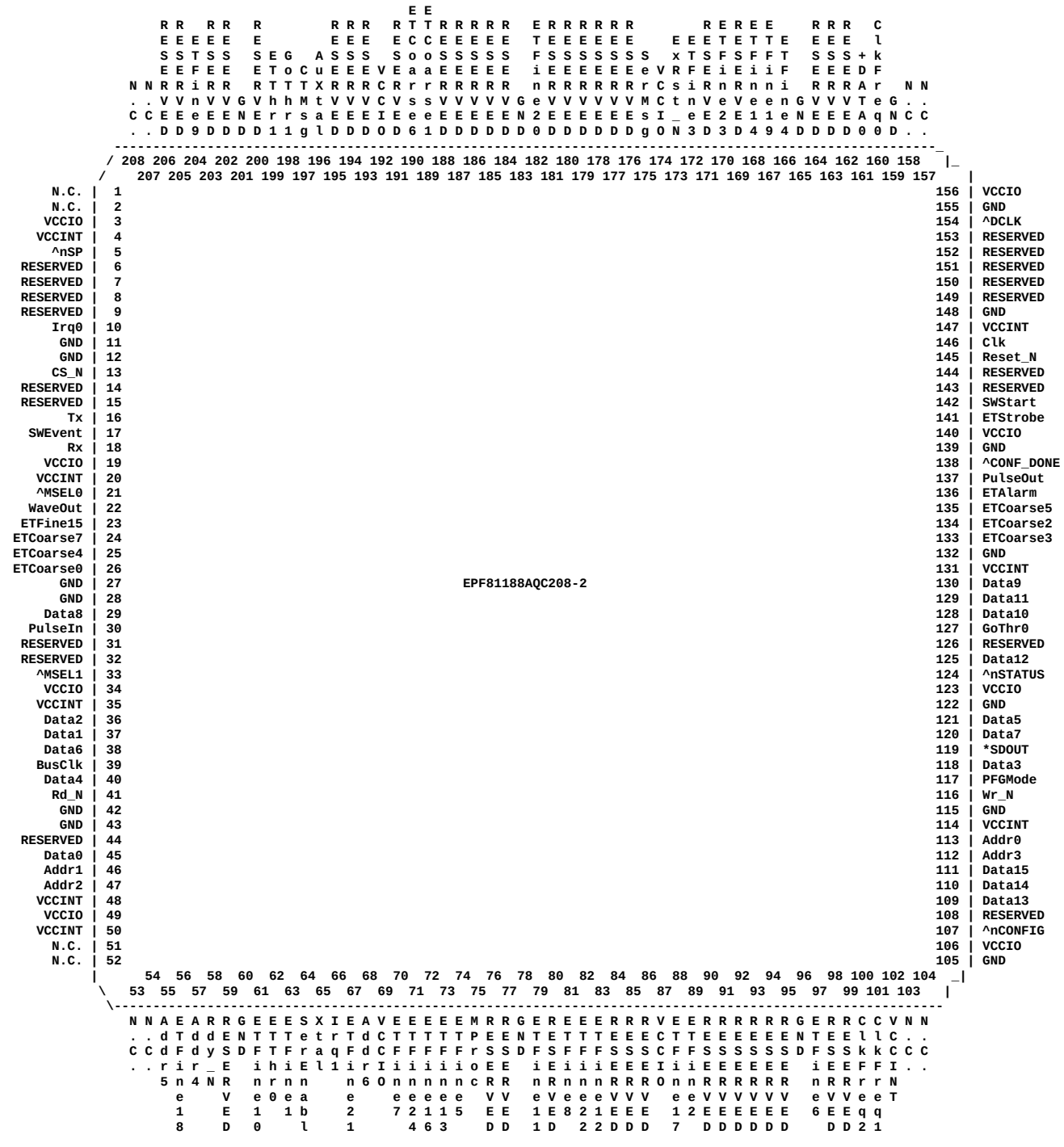
Number	Name	Name	Number	Name	Number	Name	Number	Name	Number
Addr0	113	Data3	118	ETCoarse4	25	ETFine15	23	MProc	75
Addr1	46	Data4	40	ETCoarse5	135	ETFine16	72	PFGMode	117
Addr2	47	Data5	121	ETCoarse6	190	ETFine17	88	PulseIn	30
Addr3	112	Data6	38	ETCoarse7	24	ETFine18	56	PulseOut	137
Addr4	57	Data7	120	ETFine1	63	ETFine19	167	Rd_N	41
Addr5	55	Data8	29	ETFine2	89	ETFine20	182	Rdy_N	58
Addr6	68	Data9	130	ETFine3	172	ETFine21	67	Reset_N	145
AuXtal	196	Data10	128	ETFine4	166	ETFine22	82	Rx	18
BusClk	39	Data11	129	ETFine5	74	ETFine23	170	SWEvent	17
CS_N	13	Data12	125	ETFine6	97	ETFine24	71	SWStart	142
CTMsg	197	Data13	109	ETFine7	70	ETStrobe	141	SerEnable	64
Clk	146	Data14	110	ETFine8	81	ETThr0	62	SerMsg	175
ClkFreq0	160	Data15	111	ETFine9	204	ETThr1	199	Tx	16
ClkFreq1	101	ETAlarm	136	ETFine10	61	ExRst_N	173	WaveOut	22
ClkFreq2	100	ETCoarse0	26	ETFine11	79	GoThr0	127	Wr_N	116
Data0	45	ETCoarse1	189	ETFine12	83	GoThr1	198	Xtal	65
Data1	37	ETCoarse2	134	ETFine13	73	Irq0	10		
Data2	36	ETCoarse3	133	ETFine14	168	Irq1	66		

Table 16: *CTMS signal list in name order*

Name	Number	Name	Number	Name	Number	Name	Number
Config_Done	138	DData	161	MSel1	33	nSP	5
DClk	154	MSel0	21	nConfig	107	nStatus	124

Table 17: *CTMS configuration signal list in name order*

APPENDIX K: CTMS PIN OUT (TOP VIEW)



N.C. = Not Connected.
VCCINT = Dedicated power pin, which MUST be connected to VCC (5.0 volts).
VCCIO = Dedicated power pin, which MUST be connected to VCC (5.0 or 3.3 volts).
GND = Dedicated ground pin or unused dedicated input, which MUST be connected to GND.
RESERVED = Unused I/O pin, which MUST be left unconnected.

^ = Dedicated configuration pin.
+ = Reserved configuration pin, which is tri-stated during user mode.
***** = Reserved configuration pin, which drives out in user mode.
PDn = Power Down pin.
@ = Special-purpose pin.

APPENDIX L: CTMS VHDL SOURCE CODE

```
--=============================================================================--
-- Design unit  : CTMS (Entity and architecture declarations)
--
-- File         : ctms.vhd
--
-- Purpose      : Models the Central Time Management System (CTMS) to be used
--                with the Local Time Management System (LTMS) ASIC from
--                MITEL Semiconductor AB (S).
--
-- Protocol     : 4-255 Data Bus Protocol Extensions, ESA PSS-04-256
--
-- Note         : The code is synthesisable with Altera Max2Plus.
--
-- Errors:      : None known
--
-- Library      : CTMS_Lib
--
-- Dependencies : IEEE.Std_Logic_1164 and IEEE.Std_Logic_Arith
--
-- Author       : Sandi Habinc, European Space Agency, sandi@ws.estec.esa.nl
--
-- Simulator    : ModelTechnology v4.6e, on Pentium 200 MMX, Windows95
--
-- Synthesis    : Altera Max2Plus v8.1,  on Pentium 200 MMX, Windows95
--
-- Copyright    : European Space Agency (ESA) 1998. No part may be reproduced
--                in any form without prior written permission of ESA.
-------------------------------------------------------------------------------
-- Revision list
-- Version Author Date       Changes
--
-- 0.0a    SH     13 Jan 98  Programmed on 26 January 1998
-- 0.0b    SH     29 Jan 98  Changes:  mission parameter length increased by one
--                                     etstrobe instead of swevent to etlatch
--                                     variable default removed
--                                     datactmsmode added to sensitivity list
--                                     re-synch etalarm, waveout and pulseout
--                                     preset pending-et to equal var-et-coarse
--                                     re-modelled MsgShift delay (maxplus bug)
--                                     re-modelled all synchronous delays
--                                     moved first parity latching to msgclk
--                                     moved ctmsmsg1 clear to msgstart
--                                     re-ordered multiple signal assigments in
--                                     a process, with if-statements
--
-- 0.0c    SH      1 Feb 98  Changes:  comments update for mission parameters
--                                     higher priority on missionload for msg1
--                                     longer missionload pulse
--                                     earlier missionblock
--                                     made reg1, 2 and 3 readable
--                                     parity xored with parity ctrl flag
--                                     pulsed sermsg one clk earlier in parallel
--                                     reset ctmsmode lsb earlier for parallel
--
```

```
-- 0.0d    SH      5 Feb 98  Changed message registers structure to allow a
--                           message to be completely entered before being
--                           before being shifted. Three serial shift registers
--                           have been introduced. The ET counters do not use
--                           variables any more, to save resources. Mission
--                           parameter insertion is now made via msg reg 1.
--                           Msg reg 1 to 3 can be read back after a being sent.
--                           CTMS mode flags are cleared on ET transition now.
--                           General signal renaming and house keeping.
--
-- 0.0e    SH     16 Mar 98  Debug register bit organisation changed
--                           ETAlarm added as alternative PulseIn source, tri-
--                           state control merged via bit 4
--                           Mode register bit organisation changed
--                           New Message sending method changed:
--                           If pulse, wave or init flag set in mode register
--                           the ctms status ctmsmsg1 will be sent, If pulse
--                           flag set pulse field will be sent, If wave flag
--                           is set wave field will be sent, If init flag is set
--                           ctmsmsg2-3 will be sent and ctms ET will be updated
--                           If init flag is not set but pulse or wave is then
--                           ET will be incremented and set as when default
--                           ctms message is being sent.
--------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Arith.all;

entity CTMS is
   port(
      -- System interface
      Clk:       in    Std_ULogic;                    -- System clock 2**24 Hz
      Reset_N:   in    Std_ULogic;                    -- Asynchronous reset

      -- Serial interface
      Rx:        in    Std_ULogic;                    -- RS232 Receiving
      Tx:        out   Std_ULogic;                    -- RS232 Transmitting

      -- uP interface
      CS_N:      in    Std_ULogic;                    -- Chip select
      Wr_N:      in    Std_ULogic;                    -- Write strobe
      Rd_N:      in    Std_ULogic;                    -- Read strobe/direction
      Rdy_N:     out   Std_ULogic;                    -- Ready signal
      Addr:      in    Std_Logic_Vector( 6 downto 0); -- Address bus
      Data:      inout Std_Logic_Vector(15 downto 0); -- Data bus
      Irq0:      out   Std_ULogic;                    -- Half second interrupt
      Irq1:      out   Std_ULogic;                    -- LTMS interrupt

      -- Nominal CTMS/LTMS interface
      BusClk:    out   Std_ULogic;                    -- Bus clock
      SerMsg:    out   Std_ULogic;                    -- Serial/synchronisation

      -- Nominal CTMS/LTMS mode interface
      SerEnable: out   Std_ULogic;                    -- Serial/Parallel
      AuXtal:    out   Std_ULogic;                    -- Xtal available
      ClkFreq:   out   Std_Logic_Vector(2 downto 0);  -- Bus clock freq.
      PFGMode:   out   Std_ULogic;                    -- PFG waveform gen. mode
```

```
      CTMsg:     out   Std_ULogic;                    -- CTMS message mode
      ETThr:     out   Std_Logic_Vector(1 downto 0);  -- Elapsed time threshold
      GoThr:     out   Std_Logic_Vector(1 downto 0);  -- Go threshold selector
      MProc:     out   Std_ULogic;                    -- MA31750/ERC32

      -- Debug: LTMS master interface
      ExRst_N:   out   Std_ULogic;                    -- External reset
      Xtal:      out   Std_ULogic;                    -- Aux clock

      -- Debug: LTMS user interface
      ETStrobe:  out   Std_ULogic;                    -- Elapsed time strobe
      ETAlarm:   in    Std_ULogic;                    -- Elapsed time alarm
      SWEvent:   out   Std_ULogic;                    -- Stop-watch event
      SWStart:   out   Std_ULogic;                    -- Stop-watch start
      PulseIn:   out   Std_ULogic;                    -- PFG wave gen. phase in
      PulseOut:  in    Std_ULogic;                    -- PFG pulse gen. out
      WaveOut:   in    Std_ULogic;                    -- PFG wave gen. out

      -- Debug: ET Counter output
      ETCoarse:  out   Std_Logic_Vector(7 downto 0);  -- ET Coarse
      ETFine:    out   Std_Logic_Vector(1 to    24)); -- ET Fine
end CTMS;

--============================== Architecture ================================--

architecture RTL of CTMS is

   ------------------------------------------------------------------------------
   -- Signal declarations related to the nominal operation of the CTMS
   ------------------------------------------------------------------------------
   signal  Reset_Int_N:      Std_ULogic;                    -- synch to Clk
   signal  Reset_First_N:    Std_ULogic;                    -- synch to Clk

   -- For microprocessor interface
   signal  CS_Int_N:         Std_ULogic;                    -- synch to Clk
   signal  Wr_Int_N:         Std_ULogic;                    -- synch to Clk
   signal  Rd_Int_N:         Std_ULogic;                    -- synch to Clk
   signal  CS_First_N:       Std_ULogic;                    -- synch to Clk
   signal  Wr_First_N:       Std_ULogic;                    -- synch to Clk
   signal  Rd_First_N:       Std_ULogic;                    -- synch to Clk

   signal  ReadData:         Std_Logic_Vector(15 downto 0); -- Read: data
   signal  WriteStrobe:      Std_ULogic;                    -- Write register
   signal  WritePending:     Std_ULogic;                    -- Pending write
   signal  WriteAddr:        Std_Logic_Vector( 3 downto 0); -- Write: address
   signal  WriteData:        Std_Logic_Vector(15 downto 0); -- Write: data
   signal  LatchAddr:        Std_Logic_Vector( 3 downto 0); -- Address bus

   -- For BusClk division in the ET counter
   subtype BusETIndexT is    Natural range 10 to 23;        -- ET cntr index
   signal  BusETIndex:       BusETIndexT;                   -- ET cntr index
   signal  BusClk_Int:       Std_ULogic;                    -- Internal BusClk

   -- CTMS message and synchronisation signals
   signal  SerUncoded:       Std_ULogic;                    -- Uncoded message
   signal  SerEncoded:       Std_ULogic;                    -- Serial message
```

```
signal  ManchesterClk:    Std_ULogic;                    -- Manchester clock
signal  MsgClk:           Std_ULogic;                    -- Message clock
signal  MsgSynch:         Std_ULogic;                    -- Start syn xfer
signal  MsgShift:         Std_ULogic;                    -- Shift msg
signal  MsgShiftDel0:     Std_ULogic;                    -- Delay on period
signal  MsgShiftDel1:     Std_ULogic;                    -- Delay on period
signal  MsgPulseOn:       Std_ULogic;                    -- Pulse field
signal  MsgWaveOn:        Std_ULogic;                    -- Pulse field
signal  Parity:           Std_ULogic;                    -- Even parity
signal  ParityLatch:      Std_ULogic;                    -- LatchedParity
subtype CntrT          is Natural range 0 to 255;        -- Msg counter type
signal  CntrMax:          CntrT;                         -- Msg counter len

signal  MissionReady:     Std_ULogic;                    -- Ready to receive
signal  MissionLoad:      Std_ULogic;                    -- Load reg2 to reg1

signal  PendingETCoarse:  Unsigned(31 downto 0);         -- ET Coarse Pending
signal  ETCoarse_Int:     Unsigned( 7 downto 0);         -- ET Coarse Time
signal  ETFine_Int:       Std_Logic_Vector(1 to 24);     -- ET Fine Time
signal  ETFine_Rev:       Unsigned(23 downto 0);         -- ET Fine reversed

signal  ETFineAlmostThree:Std_ULogic;                    -- ET Fine = max-3
signal  ETFineAlmostTwo:  Std_ULogic;                    -- ET Fine = max-2
signal  ETFineAlmostOne:  Std_ULogic;                    -- ET Fine = max-1
signal  ETFineAllOne:     Std_ULogic;                    -- ET Fine = max
signal  ETFineAllZero:    Std_ULogic;                    -- ET Fine = zero
signal  ETFineAlmostZero: Std_ULogic;                    -- ET Fine = zero-1

signal  Marker:           Std_ULogic;                    -- Parallel synch

-- Register enable signals
signal  RegCTMSMode:      Std_ULogic; -- 00H             -- Operating mode
signal  RegCTMSMsg1:      Std_ULogic; -- 01H             -- Status/Mission
signal  RegCTMSMsg2:      Std_ULogic; -- 02H             -- ET Coarse MSW
signal  RegCTMSMsg3:      Std_ULogic; -- 03H             -- ET Coarse LSW
signal  RegCTMSMsg4:      Std_ULogic; -- 04H             -- Pulse MSW
signal  RegCTMSMsg5:      Std_ULogic; -- 05H             -- Pulse LSW
signal  RegCTMSMsg6:      Std_ULogic; -- 06H             -- Wave Coarse Len
signal  RegCTMSMsg7:      Std_ULogic; -- 07H             -- Wave Fine Len MSW
signal  RegCTMSMsg8:      Std_ULogic; -- 08H             -- Wave Fine Len LSW
signal  RegCTMSMsg9:      Std_ULogic; -- 09H             -- Wave Polarity
signal  RegETCoarseMSW:   Std_ULogic; -- 0AH             -- CMTS ETCoarse MSW
signal  RegETCoarseLSW:   Std_ULogic; -- 0BH             -- CTMS ETCoarse LSW

-- Register output data
signal  DataCTMSMode:     Std_Logic_Vector(15 downto 0); -- Operating mode
signal  DataCTMSMsg1:     Std_Logic_Vector(15 downto 0); -- Status/Mission
signal  DataCTMSMsg2:     Std_Logic_Vector(15 downto 0); -- ET Coarse MSW
signal  DataCTMSMsg3:     Std_Logic_Vector(15 downto 0); -- ET Coarse LSW
signal  DataCTMSMsg4:     Std_Logic_Vector(15 downto 0); -- Pulse MSW
signal  DataCTMSMsg5:     Std_Logic_Vector(15 downto 8); -- Pulse LSW
signal  DataCTMSMsg6:     Std_Logic_Vector(15 downto 0); -- Wave Coarse
signal  DataCTMSMsg7:     Std_Logic_Vector(15 downto 0); -- Wave Fine MSW
signal  DataCTMSMsg8:     Std_Logic_Vector(15 downto 8); -- Wave Fine LSW
signal  DataCTMSMsg9:     Std_Logic_Vector(15 downto 0); -- Wave Polarity

-- Message shift registers
```

```
   signal  ShiftRegister1:   Std_Logic_Vector(15 downto 0); -- Shift register
   signal  ShiftRegister2:   Std_Logic_Vector(15 downto 0); -- Shift register
   signal  ShiftRegister3:   Std_Logic_Vector(15 downto 0); -- Shift register

   ------------------------------------------------------------------------------
   -- Signals related to the debugging/check-out interface to the LTMS
   ------------------------------------------------------------------------------
   -- Register enable signals
   signal  RegDebugger:      Std_ULogic; -- 0CH            -- Control
   signal  RegETLatchMSW:    Std_ULogic; -- 0DH            -- Coarse+Fine
   signal  RegETLatchLSW:    Std_ULogic; -- 0EH            -- Fine+Identifier

   -- Register output data
   signal  DataDebugger:     Std_Logic_Vector(15 downto 0); -- Control/Status
   signal  DataETLatchMSW:   Std_Logic_Vector(15 downto 0); -- Coarse+Fine
   signal  DataETLatchLSW:   Std_Logic_Vector(15 downto 0); -- Fine+Identifier

   -- Latching of ET at external/internal events
   signal  ETAlarm_Edge:     Std_ULogic;                    -- ET Alarm detect
   signal  PulseOut_Edge:    Std_ULogic;                    -- PulseOut detect
   signal  WaveOut_Edge:     Std_ULogic;                    -- WaveOut detect
   signal  ETAlarm_First:    Std_ULogic;                    -- synch to Clk
   signal  PulseOut_First:   Std_ULogic;                    -- synch to Clk
   signal  WaveOut_First:    Std_ULogic;                    -- synch to Clk
   signal  ETAlarm_Second:   Std_ULogic;                    -- synch to Clk
   signal  PulseOut_Second:  Std_ULogic;                    -- synch to Clk
   signal  WaveOut_Second:   Std_ULogic;                    -- synch to Clk
   signal  LastStrobe:       Std_ULogic;                    -- synch to Clk

begin
   ------------------------------------------------------------------------------
   -- This process synchronises the asynchronous reset input
   ------------------------------------------------------------------------------
   ResetSynchroniser: process(Clk)
   begin
      if Clk='1' and Clk'Event then
         Reset_Int_N   <= Reset_First_N;
         Reset_First_N <= Reset_N;
      end if;
   end process ResetSynchroniser;

------------------------------------------------------------------------------
-- The code below is related to the microprocessor interface and the register
-- file to which it is associated.
------------------------------------------------------------------------------
   ------------------------------------------------------------------------------
   -- This process synchronises the asynchronous microprocessor interface inputs
   ------------------------------------------------------------------------------
   Synchroniser: process(Clk, Reset_Int_N)
   begin
      if Reset_Int_N='0' then
         CS_Int_N   <= '1';
         CS_First_N <= '1';
         Wr_Int_N   <= '1';
         Wr_First_N <= '1';
         Rd_Int_N   <= '1';
         Rd_First_N <= '1';
```

```
   elsif Clk='1' and Clk'Event then
      CS_Int_N   <= CS_First_N;
      CS_First_N <= CS_N;
      Wr_Int_N   <= Wr_First_N;
      Wr_First_N <= Wr_N;
      Rd_Int_N   <= Rd_First_N;
      Rd_First_N <= Rd_N;
   end if;
end process Synchroniser;

-------------------------------------------------------------------------------
-- This process generates internal signals during microprocessor accesses
-------------------------------------------------------------------------------
uPInterface: process(Clk, Reset_Int_N)
   variable Pending: Boolean;
begin
   if Reset_Int_N='0' then
      Pending      := False;
      WriteStrobe  <= '0';
      WritePending <= '0';
      Rdy_N        <= '1';
   elsif Clk='1' and Clk'Event then
      if CS_Int_N='0' and Rd_Int_N='0' and Wr_Int_N='1' then
         -- read access
         Pending      := False;
         WriteStrobe  <= '0';
         WritePending <= '0';
         Rdy_N        <= '0';
      elsif CS_Int_N='0' and Rd_Int_N='1' and Wr_Int_N='0' then
         -- write access
         Pending      := True;
         WriteStrobe  <= '0';
         WritePending <= '1';
         Rdy_N        <= '0';
      elsif CS_Int_N='1' and Rd_Int_N='1' and Wr_Int_N='1' and Pending then
         -- pending write access
         Pending      := False;
         WriteStrobe  <= '1';
         WritePending <= '1';
         Rdy_N        <= '1';
      elsif CS_Int_N='0' and Rd_Int_N='1' and Wr_Int_N='1' and Pending then
         -- pending write access
         Pending      := False;
         WriteStrobe  <= '1';
         WritePending <= '1';
         Rdy_N        <= '1';
      elsif CS_Int_N='1' and Rd_Int_N='1' and Wr_Int_N='1' and
            not Pending then
         -- no access
         Pending      := False;
         WriteStrobe  <= '0';
         WritePending <= '0';
         Rdy_N        <= '1';
      else
         -- might be just after write access
         WriteStrobe  <= '0';
         Rdy_N        <= '1';
```

```
      end if;
   end if;
end process uPInterface;

LatchAddr <= WriteAddr when WritePending='1' else
             Addr(3 downto 0);

-------------------------------------------------------------------------------
-- This process latches data & address at the end of a microprocessor access
-------------------------------------------------------------------------------
WriteLatch: process(Wr_N)
begin
   if Wr_N='1' and Wr_n'Event then
      if CS_N='0' and Rd_N='1' then
         WriteData <= Data;
         WriteAddr <= Addr(3 downto 0);
      end if;
   end if;
end process WriteLatch;

-------------------------------------------------------------------------------
-- This process enable the uP output data bus
-------------------------------------------------------------------------------
ReadLatch: process(Wr_N, CS_N, Rd_N, ReadData)
begin
   if CS_N='0' and Rd_N='0' and Wr_N='1' then
      Data <= ReadData;
   else
      Data <= (others => 'Z');
   end if;
end process ReadLatch;

-------------------------------------------------------------------------------
-- This process selects the source for the internal read bus
-------------------------------------------------------------------------------
ReadDataMutiplexer: process(
   RegCTMSMode,    DataCTMSMode,
   RegCTMSMsg1,    DataCTMSMsg1,
   RegCTMSMsg2,    DataCTMSMsg2,
   RegCTMSMsg3,    DataCTMSMsg3,
   RegDebugger,    DataDebugger,
   RegETLatchMSW,  DataETLatchMSW,
   RegETLatchLSW,  DataETLatchLSW,
   RegETCoarseMSW, RegETCoarseLSW, PendingETCoarse)
begin
   if    RegCTMSMode='1' then
      ReadData <= DataCTMSMode;
   elsif RegETCoarseMSW='1' then
      ReadData <= Conv_Std_Logic_Vector(PendingETCoarse(31 downto 16), 16);
   elsif RegETCoarseLSW='1' then
      ReadData <= Conv_Std_Logic_Vector(PendingETCoarse(15 downto  0), 16);
   elsif RegCTMSMsg1='1' then
      ReadData <= DataCTMSMsg1;
   elsif RegCTMSMsg2='1' then
      ReadData <= DataCTMSMsg2;
   elsif RegCTMSMsg3='1' then
      ReadData <= DataCTMSMsg3;
```

```
   elsif RegDebugger='1' then                          -- Related to debugging
      ReadData <= DataDebugger;
   elsif RegETLatchMSW='1' then
      ReadData <= DataETLatchMSW;
   elsif RegETLatchLSW='1' then
      ReadData <= DataETLatchLSW;
   else
      ReadData <= (others => '0');
   end if;
end process ReadDataMutiplexer;

-------------------------------------------------------------------------------
-- This process generates the internal register select signals
-------------------------------------------------------------------------------
AddressDecoder: process(LatchAddr)
begin
   RegCTMSMode       <= '0';
   RegCTMSMsg1       <= '0';
   RegCTMSMsg2       <= '0';
   RegCTMSMsg3       <= '0';
   RegCTMSMsg4       <= '0';
   RegCTMSMsg5       <= '0';
   RegCTMSMsg6       <= '0';
   RegCTMSMsg7       <= '0';
   RegCTMSMsg8       <= '0';
   RegCTMSMsg9       <= '0';
   RegETCoarseMSW    <= '0';
   RegETCoarseLSW    <= '0';
   if LatchAddr    = "0000" then
      RegCTMSMode    <= '1';
   elsif LatchAddr = "0001" then
      RegCTMSMsg1    <= '1';
   elsif LatchAddr = "0010" then
      RegCTMSMsg2    <= '1';
   elsif LatchAddr = "0011" then
      RegCTMSMsg3    <= '1';
   elsif LatchAddr = "0100" then
      RegCTMSMsg4    <= '1';
   elsif LatchAddr = "0101" then
      RegCTMSMsg5    <= '1';
   elsif LatchAddr = "0110" then
      RegCTMSMsg6    <= '1';
   elsif LatchAddr = "0111" then
      RegCTMSMsg7    <= '1';
   elsif LatchAddr = "1000" then
      RegCTMSMsg8    <= '1';
   elsif LatchAddr = "1001" then
      RegCTMSMsg9    <= '1';
   elsif LatchAddr = "1010" then
      RegETCoarseMSW <= '1';
   elsif LatchAddr = "1011" then
      RegETCoarseLSW <= '1';
   end if;
end process AddressDecoder;

-------------------------------------------------------------------------------
```

```
-- This part is related to the CTMS registers, ET counter and CTMS messaging.
-------------------------------------------------------------------------------
   -----------------------------------------------------------------------------
   -- This process holds the internal registers
   -----------------------------------------------------------------------------
   RegisterFile: process(Clk, Reset_Int_N)
   begin
      if Reset_Int_N='0' then
         DataCTMSMode <= (others => '0');
         DataCTMSMsg1 <= (others => '0');
         DataCTMSMsg2 <= (others => '0');
         DataCTMSMsg3 <= (others => '0');
         DataCTMSMsg4 <= (others => '0');
         DataCTMSMsg5 <= (others => '0');
         DataCTMSMsg6 <= (others => '0');
         DataCTMSMsg7 <= (others => '0');
         DataCTMSMsg8 <= (others => '0');
         DataCTMSMsg9 <= (others => '0');
         MsgPulseOn   <= '0';
         MsgWaveOn    <= '0';
      elsif Clk='1' and Clk'Event then

         -- Registers 1 to 3 are never shifted
         if WriteStrobe='1' and RegCTMSMsg1='1' then
            -- Write access to registers
               DataCTMSMsg1 <= WriteData;
         end if;
         if WriteStrobe='1' and RegCTMSMsg2='1' then
            -- Write access to registers
               DataCTMSMsg2 <= WriteData;
         end if;
         if WriteStrobe='1' and RegCTMSMsg3='1' then
            -- Write access to registers
               DataCTMSMsg3 <= WriteData;
         end if;

         -- Registers 4 to 5 are shifted when a Pulse field is enabled
         if WriteStrobe='1' and RegCTMSMsg4='1' then
            -- Write access to registers
               DataCTMSMsg4 <= WriteData;
         elsif MsgShift='1' and MsgPulseOn='1' then
            -- Normal shift operation during transmission
            DataCTMSMsg4 <= DataCTMSMsg4(14 downto 0) & DataCTMSMsg5(15);
         end if;
         if WriteStrobe='1' and RegCTMSMsg5='1' then
            -- Write access to registers
               DataCTMSMsg5 <= WriteData(15 downto 8);
         elsif MsgShift='1' and MsgPulseOn='1' then
            -- Normal shift operation during transmission
            DataCTMSMsg5(15 downto 9) <= DataCTMSMsg5(14 downto 8);
            if MsgWaveOn='1' then                    -- Wave field connected
               DataCTMSMsg5(8) <= DataCTMSMsg6(15);
            else
               DataCTMSMsg5(8) <= '0';
            end if;
         end if;
```

```
      -- Registers 6 to 9 are shifted when a Waveform field is enabled
      if WriteStrobe='1' and RegCTMSMsg6='1' then
         -- Write access to registers
            DataCTMSMsg6 <= WriteData;
      elsif MsgShift='1' and MsgWaveOn='1' then
         -- Normal shift operation during transmission
         DataCTMSMsg6 <= DataCTMSMsg6(14 downto 0) & DataCTMSMsg7(15);
      end if;
      if WriteStrobe='1' and RegCTMSMsg7='1' then
         -- Write access to registers
            DataCTMSMsg7 <= WriteData;
      elsif MsgShift='1' and MsgWaveOn='1' then
         -- Normal shift operation during transmission
         DataCTMSMsg7 <= DataCTMSMsg7(14 downto 0) & DataCTMSMsg8(15);
      end if;
      if WriteStrobe='1' and RegCTMSMsg8='1' then
         -- Write access to registers
            DataCTMSMsg8 <= WriteData(15 downto 8);
      elsif MsgShift='1' and MsgWaveOn='1' then
         -- Normal shift operation during transmission
         DataCTMSMsg8 <= DataCTMSMsg8(14 downto 8) & DataCTMSMsg9(15);
      end if;
      if WriteStrobe='1' and RegCTMSMsg9='1' then
         -- Write access to registers
            DataCTMSMsg9 <= WriteData;
      elsif MsgShift='1' and MsgWaveOn='1' then
         -- Normal shift operation during transmission
         DataCTMSMsg9 <= DataCTMSMsg9(14 downto 0) & '0';

      end if;

      -- CTMS mode register, automatic clear of some bits
      if WriteStrobe='1' and RegCTMSMode='1' then
         -- Write access to registers
            DataCTMSMode <= WriteData;
      else
         if ETFineAllZero='1' then
            MsgPulseOn <= DataCTMSMode(14);
            MsgWaveOn  <= DataCTMSMode(13);
            -- Reset after accepting a new message in serial mode or
            -- updating pending ET in parallel mode
            DataCTMSMode(14 downto 12) <= "000";

         end if;
         if Marker='1' or MissionLoad='1' then
            -- Reset after Marker being generated in parallel mode or
            --       after Mission Parameter being loaded serial mode
            DataCTMSMode(15)           <= '0';
         end if;
      end if;

   end if;
end process RegisterFile;

-------------------------------------------------------------------------------
-- This process holds the message shift registers.
-- When no message is provided, it is assumed that the serial shift register
```

```
-- is cleared so that parity and first bit are correct.
-------------------------------------------------------------------------------
ShiftRegister: process(Clk, Reset_Int_N)
begin
   if Reset_Int_N='0' then
      ShiftRegister1 <= (others => '0');
      ShiftRegister2 <= (others => '0');
      ShiftRegister3 <= (others => '0');
   elsif Clk='1' and Clk'Event then

      if    ETFineAlmostThree='1' and
            (DataCTMSMode(14)='1'  or            -- Wave field enabled
             DataCTMSMode(13)='1'  or            -- Pulse field enabled
             DataCTMSMode(12)='1') then          -- Initialisation enabled
         -- Load register with user message
         ShiftRegister1 <= DataCTMSMsg1;
      elsif MissionLoad='1' then
         -- Load shift register with mission parameter word
         ShiftRegister1 <= DataCTMSMsg1;
      elsif MsgShift='1' then
         -- Normal shift operation during transmission
         ShiftRegister1 <= ShiftRegister1(14 downto 0) & ShiftRegister2(15);
      end if;

      if    ETFineAllZero='1' and DataCTMSMode(12)='1' then
         -- Load register with user message since initialisation requested
         ShiftRegister2 <= DataCTMSMsg2;
      elsif ETFineAllZero='1' and DataCTMSMode(12)='0' then
         -- Load register with Pending ET
         ShiftRegister2 <=
            Conv_Std_Logic_Vector(PendingETCoarse(31 downto 16), 16);
      elsif MsgShift='1' then
         -- Normal shift operation during transmission
         ShiftRegister2 <= ShiftRegister2(14 downto 0) & ShiftRegister3(15);
      end if;

      if    ETFineAllZero='1' and DataCTMSMode(12)='1' then
         -- Load register with user message since initialisation requested
         ShiftRegister3 <= DataCTMSMsg3;
      elsif ETFineAllZero='1' and DataCTMSMode(12)='0' then
         -- Load register with Pending ET
         ShiftRegister3 <=
            Conv_Std_Logic_Vector(PendingETCoarse(15 downto  0), 16);
      elsif MsgShift='1' then
         -- Normal shift operation during transmission
         ShiftRegister3(15 downto 1) <= ShiftRegister3(14 downto 0);
         if    MsgPulseOn='1' then                -- Pulse field enabled
            ShiftRegister3(0) <= DataCTMSMsg4(15);
         elsif MsgWaveOn='1' then                 -- Wave field connected
            ShiftRegister3(0) <= DataCTMSMsg6(15);
         else
            ShiftRegister3(0) <= '0';
         end if;
      end if;

   end if;
end process ShiftRegister;
```

```
-- Even parity calculated over the MSB of the shift register
Parity <= ShiftRegister1(15) xor ShiftRegister1(14) xor
          ShiftRegister1(13) xor ShiftRegister1(12) xor
          ShiftRegister1(11) xor ShiftRegister1(10) xor
          ShiftRegister1(9)  xor ShiftRegister1(8);

--------------------------------------------------------------------------
-- This process sets the CTMS/LTMS operating mode:
--------------------------------------------------------------------------
-- Bit  15:       Serial mode:      send one Mission Parameter word
--                Stand alone mode: enable single shot marker at next tick
--                                        (reset after action for both modes)
-- Bit  14:       Send Pulse field        (reset after message being sent)
-- Bit  13:       Send Waveform field     (reset after message being sent)
-- Bit  12:       Initialisation enabled  (reset after message being sent)
-- Bit  11:       MProc         line
-- Bit  10 to 8:  ClkFreq(2:0)  lines
-- Bit   7 to 6:  GoThr(1:0)    lines
-- Bit   5 to 4:  ETThr(1:0)    lines
-- Bit   3:       PFGMode       line
-- Bit   2:       AuXtal        line
-- Bit   1:       SerEnable     line
-- Bit   0:       CTMsg         line
--------------------------------------------------------------------------
-- Also, bits 0 to 2 and 10 to 8 control the BusClk frequency
-- Debugging feature: bit 2 enables Xtal output
--------------------------------------------------------------------------
CTMSMode: process(DataCTMSMode, BusETIndex)
begin
   CTMsg     <= DataCTMSMode(0);
   SerEnable <= DataCTMSMode(1);
   AuXtal    <= DataCTMSMode(2);
   PFGMode   <= DataCTMSMode(3);
   ETThr     <= DataCTMSMode( 5 downto 4);
   GoThr     <= DataCTMSMode( 7 downto 6);
   ClkFreq   <= DataCTMSMode(10 downto 8);
   MProc     <= DataCTMSMode(11);
   -- ClkBus setup
   if DataCTMSMode(2 downto 0)="101" and DataCTMSMode(10)='1' then
      -- ParAux Mil-1553 mode
      if    DataCTMSMode(9 downto 8)="11" then
         BusETIndex <= 13;
      elsif DataCTMSMode(9 downto 8)="10" then
         BusETIndex <= 12;
      elsif DataCTMSMode(9 downto 8)="01" then
         BusETIndex <= 11;
      else
         BusETIndex <= 10;
      end if;
   else
      -- Other modes. Note that the clock is blocked elsewhere in SerAux mode
      if    DataCTMSMode(9 downto 8)="11" then
         BusETIndex <= 23;
      elsif DataCTMSMode(9 downto 8)="10" then
         BusETIndex <= 22;
      elsif DataCTMSMode(9 downto 8)="01" then
```

```
         BusETIndex <= 21;
      else
         BusETIndex <= 20;
      end if;
   end if;
end process CTMSMode;

BusClk <= 'Z' when DataDebugger(7)='0'            else -- Tristate
          '0' when DataCTMSMode(2 downto 0)="111" else -- SerAux mode
          '0' when DataDebugger(5)='1'            else -- Debug: block
          not BusClk_Int;                              -- ET aligned hi edge

Xtal   <= Clk when DataCTMSMode(2)='1' else            -- AuXtal enabled
          '0';

-------------------------------------------------------------------------------
-- This process generates the clock and start pulse for the serial message
-------------------------------------------------------------------------------
ETFineDecoder: process(ETFine_Int, DataCTMSMode)
 constant AlmostThree: Std_Logic_Vector(1 to 24):="111111111111111111111100";
 constant AlmostTwo:   Std_Logic_Vector(1 to 24):="111111111111111111111101";
 constant AlmostOne:   Std_Logic_Vector(1 to 24):="111111111111111111111110";
 constant AllOne:      Std_Logic_Vector(1 to 24):="111111111111111111111111";
 constant AllZero:     Std_Logic_Vector(1 to 24):="000000000000000000000000";
 constant AlmostZero:  Std_Logic_Vector(1 to 24):="000000000000000000000001";
begin
   -- flag active when first synch pattern bit is due
   if ETFine_Int( 1 to 11) = Std_Logic_Vector'("11111111111") and
      ETFine_Int(18 to 24) = Std_Logic_Vector'("1111111") and
      ((ETFine_Int(12 to 17) = Std_Logic_Vector'("111001") and
        DataCTMSMode(9 downto 8)="11")      or
       (ETFine_Int(12 to 17) = Std_Logic_Vector'("110011") and
        DataCTMSMode(9 downto 8)="10")      or
       (ETFine_Int(12 to 17) = Std_Logic_Vector'("100111") and
        DataCTMSMode(9 downto 8)="01")      or
       (ETFine_Int(12 to 17) = Std_Logic_Vector'("001111") and
        DataCTMSMode(9 downto 8)="00"))     then
      MsgSynch <= '1';
   else
      MsgSynch <= '0';
   end if;
   -- flag active when the CTMS msg register data bits are to be transmitted
   if (ETFine_Int(17 to 24) =    Std_Logic_Vector'("11111110") and
       DataCTMSMode(9 downto 8)="11")           or
      (ETFine_Int(16 to 24) =   Std_Logic_Vector'("111111110") and
       DataCTMSMode(9 downto 8)="10")           or
      (ETFine_Int(15 to 24) =  Std_Logic_Vector'("1111111110") and
       DataCTMSMode(9 downto 8)="01")           or
      (ETFine_Int(14 to 24) = Std_Logic_Vector'("11111111110") and
       DataCTMSMode(9 downto 8)="00")           then
      MsgClk <= '1';
   else
      MsgClk <= '0';
   end if;
   -- flag active when manchester bit is to be transmitted
   if (ETFine_Int(18 to 24) =    Std_Logic_Vector'("1111111") and
       DataCTMSMode(9 downto 8)="11")          or
```

```
      (ETFine_Int(17 to 24) =   Std_Logic_Vector'("11111111") and
       DataCTMSMode(9 downto 8)="10")          or
      (ETFine_Int(16 to 24) =  Std_Logic_Vector'("111111111") and
       DataCTMSMode(9 downto 8)="01")          or
      (ETFine_Int(15 to 24) = Std_Logic_Vector'("1111111111") and
       DataCTMSMode(9 downto 8)="00")          then
      ManchesterClk <= '1';
   else
      ManchesterClk <= '0';
   end if;
   -- decodes all-ones-3 on ET Fine time counter
   if ETFine_Int(1 to 24) = AlmostThree then
      ETFineAlmostThree <= '1';
   else
      ETFineAlmostThree <= '0';
   end if;
   -- decodes all-ones-2 on ET Fine time counter
   if ETFine_Int(1 to 24) = AlmostTwo then
      ETFineAlmostTwo <= '1';
   else
      ETFineAlmostTwo <= '0';
   end if;
   -- decodes all-ones-1 on ET Fine time counter
   if ETFine_Int(1 to 24) = AlmostOne then
      ETFineAlmostOne <= '1';
   else
      ETFineAlmostOne <= '0';
   end if;
   -- decodes all-ones on ET Fine time counter
   if ETFine_Int(1 to 24) = AllOne then
      ETFineAllOne <= '1';
   else
      ETFineAllOne <= '0';
   end if;
   -- decodes all-zero on ET Fine time counter
   if ETFine_Int(1 to 24) = AllZero then
      ETFineAllZero <= '1';
   else
      ETFineAllZero <= '0';
   end if;
   -- decodes all-zero+1 on ET Fine time counter
   if ETFine_Int(1 to 24) = AlmostZero then
      ETFineAlmostZero <= '1';
   else
      ETFineAlmostZero <= '0';
   end if;

end process ETFineDecoder;

-------------------------------------------------------------------------
-- This sets the CTMS nominal message length (without Mission Parsmeters)
-------------------------------------------------------------------------
CntrMax <= 144 when MsgPulseOn='1' and MsgWaveOn='1' else
           117 when MsgWaveOn='1'                    else
            81 when MsgPulseOn='1'                   else
            54;
```

```
-------------------------------------------------------------------------
-- This process generates the CTMS message
-------------------------------------------------------------------------
CTMSMessage: process(Clk, Reset_Int_N)
   variable Cntr:         CntrT;
   variable Running:      Boolean;
begin
   if Reset_Int_N='0' then
      SerUncoded    <= '0';
      MsgShift      <= '0';
      MsgShiftDel1  <= '0';
      MsgShiftDel0  <= '0';
      Cntr          := 0;
      Running       := False;
      ParityLatch   <= '0';
      MissionReady  <= '0';
      MissionLoad   <= '0';
   elsif Clk='1' and Clk'Event then
      MsgShift     <= MsgShiftDel1;                -- delay one period
      MsgShiftDel1 <= MsgShiftDel0;                -- delay one period
      if DataCTMSMode(1 downto 0)="11" then        -- Serial mode
         if ETFineAlmostTwo='1' then
            -- CTMS message to be sent
            Running      := True;
            Cntr         :=  0;
            -- first parity
            ParityLatch  <= Parity;
         elsif MsgClk='1' and Running then
            -- Bit counter
            Cntr    := Cntr+1;
            -- message being sent
            if Cntr > CntrMax then
               -- message transmission ended
               Running       := False;
               SerUncoded    <= '0';
               MsgShiftDel0  <= '0';
               MissionLoad   <= '0';
            elsif Cntr=9   or Cntr=18  or Cntr=27  or Cntr=36  or
                  Cntr=45  or Cntr=54  or Cntr=63  or Cntr=72  or
                  Cntr=81  or Cntr=90  or Cntr=99  or Cntr=108 or
                  Cntr=117 or Cntr=126 or Cntr=135 or Cntr=144 then
               -- parity bit to be sent
               SerUncoded <= ParityLatch xor
                             DataDebugger(12);      -- Debug: error
               -- save parity for the current MSB in shift register
               ParityLatch  <= Parity;
               if Cntr=CntrMax-18 then
                  -- ready to receive a mission parameter word in reg1
                  MissionReady <= '1';
               else
                  MissionReady <= '0';
               end if;
               MsgShiftDel0  <= '0';
               MissionLoad   <= '0';
            else
               -- send data bit
               if Cntr=CntrMax-1 and DataCTMSMode(15)='1' then
```

```
                -- mission parameter word available, add two octets and two
                -- parity bits to the bit counter
                Cntr         := CntrMax-18-1;
                -- load CTMS msg reg1 into shift reg1 for further shifting
                MissionLoad  <= '1';
                MsgShiftDel0 <= '0';
             else
                -- ready to shift CTMS msg/and shift registers
                MsgShiftDel0 <= '1';
                MissionLoad  <= '0';
             end if;
             -- output serial shift register MSB
             SerUncoded   <= ShiftRegister1(15);
          end if;
       else
          MsgShiftDel0  <= '0';
          MissionLoad   <= '0';
       end if;
    else
       MsgShiftDel0  <= '0';
       MissionLoad   <= '0';
    end if;
 end if;
end process CTMSMessage;

-------------------------------------------------------------------------------
-- This process sends the synch pattern & Manchester encodes the CTMS message
-------------------------------------------------------------------------------
ManchesterEncoder: process(Clk, Reset_Int_N)
   subtype  CntrT is Natural range 0 to 7;
   variable Cntr:    CntrT;
   variable Odd:     Boolean;
begin
   if Reset_Int_N='0' then
      SerEncoded <= '0';
      Cntr       := 0;
      Odd        := False;
   elsif Clk='1' and Clk'Event then
      if MsgSynch='1' then
         -- ready to transmit synchronisation pattern, bit 0
         SerEncoded    <= '0';
         Cntr          := 1;
         Odd           := False;
      elsif ManchesterClk='1' then
         if Cntr=1 or Cntr=2 then
            Cntr       := Cntr+1;
            SerEncoded <= '0';
         elsif Cntr=3 or Cntr=4 then
            Cntr       := Cntr+1;
            SerEncoded <= '1';
         elsif Cntr=5 then
            Cntr       := 0;
            SerEncoded <= '1';
         else
            -- encode and send msg bits
            Cntr       := 0;
            if Odd then
```

```
            SerEncoded <= not SerUnCoded;
         else
            SerEncoded <=     SerUnCoded;
         end if;
         Odd        := not Odd;
         -- will continue to modulate zeros on the line after msg
      end if;
   end if;
end if;
end process ManchesterEncoder;

SerMsg <= 'Z'        when DataDebugger(7)='0' else  -- tristate
          '0'        when DataDebugger(6)='1' else  -- Debug: block output
          Marker     when DataCTMSMode(1)='0' else  -- parallel mode
          SerEncoded;                               -- serial mode

-------------------------------------------------------------------------------
-- This process generates sychronisation pulses in parallel/stand alone mode
-------------------------------------------------------------------------------
ParallelMarker: process(Clk, Reset_Int_N)
begin
   if Reset_Int_N='0' then
      Marker <= '0';
   elsif Clk='1' and Clk'Event then
      if (ETFineAlmostOne='1' or ETFineAllOne='1') and
          -- Stand alone and single shot enabled
          ((DataCTMSMode(0)='0' and DataCTMSMode(15)='1') or
          -- Parallel mode
          (DataCTMSMode(1 downto 0)="01")) then
         -- generate marker on SerMsg, to be high at rising BusClk edge
         Marker <= '1';
      else
         Marker <= '0';
      end if;
   end if;
end process ParallelMarker;

-------------------------------------------------------------------------------
-- This process handles the ET Coarse time:
--   loaded with each new CTMS message
--   or incremented if no new CTMS message
-- PendingETCoarse holds the pending ET and ETCoarse_Int holds the
-- LSB of the current ET (nominally 1 sec less).
-------------------------------------------------------------------------------
ETCoarseCounter: process(Clk, Reset_Int_N)
begin
   if Reset_Int_N='0' then
      PendingETCoarse <= (0 => '1', others => '0'); -- Start at one
      ETCoarse_Int    <= (          others => '0'); -- Start at zero
   elsif Clk='1' and Clk'Event then
      if ETFineAllOne='1' then                      -- Increment point
         ETCoarse_Int <= PendingETCoarse(7 downto 0);
         if DataCTMSMode(12)='1' then               -- Initialisation message
            -- load pending ET with CTMS msg 2 and 3 registers
            for i in 31 downto 16 loop
               PendingETCoarse(i) <= DataCTMSMsg2(i-16);
            end loop;
```

```
            for i in 15 downto 0 loop
               PendingETCoarse(i) <= DataCTMSMsg3(i);
            end loop;
         else                                        -- Increment coarse ET
            PendingETCoarse       <= PendingETCoarse + 1;
         end if;
      end if;
   end if;
end process ETCoarseCounter;

ETCoarse <= Conv_Std_Logic_Vector(ETCoarse_Int(7 downto 0), 8);

-------------------------------------------------------------------------------
-- This process implements ET Fine counter, BusClk and half second interrupt
-------------------------------------------------------------------------------
ETFineCounter: process(Clk, Reset_Int_N)
   constant OneFine:    Unsigned(23 downto 0) := (0 => '1', others => '0');
   constant TwoFine:    Unsigned(23 downto 0) := (1 => '1', others => '0');
   constant AllOneFine: Unsigned(23 downto 0) := (          others => '1');
begin
   if Reset_Int_N='0' then
      ETFine_Rev <= (others => '0');
   elsif Clk='1' and Clk'Event then
      if DataDebugger(15)='1' then                 -- Debug: preset ET Fine
         ETFine_Rev <= "111111111110000000000000";
      elsif DataDebugger(14)='1' then              -- Debug: add one
         ETFine_Rev <= ETFine_Rev + TwoFine;
      elsif DataDebugger(13)='1' then              -- Debug: sub one
         ETFine_Rev <= ETFine_Rev;
      else                                         -- normal increment
         ETFine_Rev <= ETFine_Rev + OneFine;
      end if;
   end if;
end process ETFineCounter;

-- Index change
ETFine_Int( 1)  <= ETFine_Rev(23);
ETFine_Int( 2)  <= ETFine_Rev(22);
ETFine_Int( 3)  <= ETFine_Rev(21);
ETFine_Int( 4)  <= ETFine_Rev(20);
ETFine_Int( 5)  <= ETFine_Rev(19);
ETFine_Int( 6)  <= ETFine_Rev(18);
ETFine_Int( 7)  <= ETFine_Rev(17);
ETFine_Int( 8)  <= ETFine_Rev(16);
ETFine_Int( 9)  <= ETFine_Rev(15);
ETFine_Int(10)  <= ETFine_Rev(14);
ETFine_Int(11)  <= ETFine_Rev(13);
ETFine_Int(12)  <= ETFine_Rev(12);
ETFine_Int(13)  <= ETFine_Rev(11);
ETFine_Int(14)  <= ETFine_Rev(10);
ETFine_Int(15)  <= ETFine_Rev( 9);
ETFine_Int(16)  <= ETFine_Rev( 8);
ETFine_Int(17)  <= ETFine_Rev( 7);
ETFine_Int(18)  <= ETFine_Rev( 6);
ETFine_Int(19)  <= ETFine_Rev( 5);
ETFine_Int(20)  <= ETFine_Rev( 4);
ETFine_Int(21)  <= ETFine_Rev( 3);
```

```
   ETFine_Int(22)  <= ETFine_Rev( 2);
   ETFine_Int(23)  <= ETFine_Rev( 1);
   ETFine_Int(24)  <= ETFine_Rev( 0);

   ETFine     <= ETFine_Int;                          -- external signal
   BusClk_Int <= ETFine_Int(BusETIndex);              -- derived frequency
   Irq0       <= ETFine_Int(1);                       -- half second interrupt

-------------------------------------------------------------------------------
-- The code here below is related to the debugging/check-out functions related
-- to the direct operation of the LTMS inputs and ET counter adjustments, etc.
-------------------------------------------------------------------------------
   -----------------------------------------------------------------------------
   -- This process synchronises asynchronous inputs
   -----------------------------------------------------------------------------
   DebugSynchroniser: process(Clk, Reset_Int_N)
   begin
      if Reset_Int_N='0' then
         ETAlarm_Edge    <= '0';
         ETAlarm_First   <= '0';
         ETAlarm_Second  <= '0';
         PulseOut_Edge   <= '0';
         PulseOut_First  <= '0';
         PulseOut_Second <= '0';
         WaveOut_Edge    <= '0';
         WaveOut_First   <= '0';
         WaveOut_Second  <= '0';
      elsif Clk='1' and Clk'Event then
         ETAlarm_Edge    <= ETAlarm_First  and not ETAlarm_Second;
         ETAlarm_Second  <= ETAlarm_First;
         ETAlarm_First   <= ETAlarm;
         PulseOut_Edge   <= PulseOut_First and not PulseOut_Second;
         PulseOut_Second <= PulseOut_First;
         PulseOut_First  <= PulseOut;
         WaveOut_Edge    <= WaveOut_First  and not WaveOut_Second;
         WaveOut_Second  <= WaveOut_First;
         WaveOut_First   <= WaveOut;
      end if;
   end process DebugSynchroniser;

   -----------------------------------------------------------------------------
   -- This process generates the internal debug register select signals
   -----------------------------------------------------------------------------
   DebugAddressDecoder: process(LatchAddr)
   begin
      RegDebugger      <= '0';
      RegETLatchMSW    <= '0';
      RegETLatchLSW    <= '0';
      if    LatchAddr = "1100" then
         RegDebugger   <= '1';
      elsif LatchAddr = "1101" then
         RegETLatchMSW <= '1';
      elsif LatchAddr = "1110" then
         RegETLatchLSW <= '1';
      end if;
   end process DebugAddressDecoder;
```

```
-------------------------------------------------------------------------------
-- This process holds the Debugger and ET latch registers.
-- This process holds the ET counter latch, triggered by various events.
-- The 2 LSBs should be interpreted as:
--   00  ET Strobe generated from CTMS
--   01  ET Alarm detected from LTMS
--   10  PFG Wave out detected from LTMS
--   11  FPG Pulse out detected from LTMS
-------------------------------------------------------------------------------
DebugRegisterFile: process(Clk, Reset_Int_N)
begin
   if Reset_Int_N='0' then
      DataDebugger   <= (others => '0');
      DataETLatchMSW <= (others => '0');
      DataETLatchLSW <= (others => '0');
      LastStrobe     <= '0';
   elsif Clk='1' and Clk'Event then
      if (DataDebugger(1)='1' and LastStrobe='0') or
          ETAlarm_Edge='1'  or
          PulseOut_Edge='1' or
          WaveOut_Edge='1'  then
         -- latch ET Coarse LSB and ET Fine counter value
         DataETLatchMSW <= Conv_Std_Logic_Vector(ETCoarse_Int, 8) &
                           ETFine_Int(1 to 8);
         DataETLatchLSW <= ETFine_Int(9 to 22) &
                           (WaveOut_Edge or PulseOut_Edge) &
                           (ETAlarm_Edge or PulseOut_Edge);
      end if;
      -- write access to register
      if WriteStrobe='1' and RegDebugger='1' then
         DataDebugger <= WriteData;
      else
         -- clear ET adjustment flags after one period
         DataDebugger(15 downto 13) <= "000";
      end if;
      -- edge detection
      LastStrobe <= DataDebugger(1);
   end if;
end process DebugRegisterFile;

-------------------------------------------------------------------------------
-- This process implements the output functions of the debug register
-------------------------------------------------------------------------------
-- Bit  15:       Preset    ET Fine Time        (reset after action)
-- Bit  14:       Increment ET Fine Time by one (reset after action)
-- Bit  13:       Decrement ET Fine time by one (reset after action)
-- Bit  12:       Introduce parity errors
-- Bit  11 to 10: Select Irq1 source:    00 Force zero
--                                       01 Ready for Mission Parameter Word
--                                       10 PulseOut
--                                       11 ETAlarm
-- Bit   9 to  8: Select PulseIn source: 00 Force zero
--                                       01 Force one
--                                       10 PulseOut
--                                       11 ETAlarm
-- Bit   7:       Drive LTMS/CTMS lines (BusClk, SerMsg)
-- Bit   6:       Block SerMsg line
```

```
   -- Bit   5:       Block BusClk line
   -- Bit   4:       Drive (ETStrobe, SWEvent, SWStart, PulseIn)
   -- Bit   3:       SWStart line
   -- Bit   2:       SWEvent line
   -- Bit   1:       ETStrobe line
   -- Bit   0:       ExRst_N line (LTMS in reset at power up)
   ------------------------------------------------------------------------
   DebugOutput: process(DataDebugger, PulseOut, ETAlarm, MissionReady)
   begin
      ExRst_N     <= DataDebugger(0);
      if DataDebugger(4)='0' then
         ETStrobe <= 'Z';
         SWEvent  <= 'Z';
         SWStart  <= 'Z';
         PulseIn  <= 'Z';
      else
         ETStrobe <= DataDebugger(1);
         SWEvent  <= DataDebugger(2);
         SWStart  <= DataDebugger(3);
         if    DataDebugger(9 downto 8)="00" then
            PulseIn  <= '0';
         elsif DataDebugger(9 downto 8)="01" then
            PulseIn  <= '1';
         elsif DataDebugger(9 downto 8)="10" then
            PulseIn  <= PulseOut;
         else
            PulseIn  <= ETAlarm;
         end if;
      end if;
      if    DataDebugger(11 downto 10)="00" then
         Irq1     <= '0';
      elsif DataDebugger(11 downto 10)="01" then
         Irq1     <= MissionReady;
      elsif DataDebugger(11 downto 10)="10" then
         Irq1     <= PulseOut;
      else
         Irq1     <= ETAlarm;
      end if;
   end process DebugOutput;

   ------------------------------------------------------------------------
   -- This assignment is a dummy, using reserved pins for future usage
   ------------------------------------------------------------------------
   Tx <= Rx and Addr(6) and Addr(5) and Addr(4);       -- Dummy!

end RTL; --====================== End of Decoder(RTL) ======================--
```