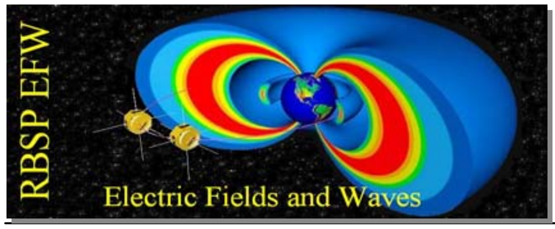


RBSP EFW

Data Controller Board (DCB) FPGA Specification & Functional Description

RBSP_EFW_DCB_001O

D. Gordon 2010.03.25
09:56:04 -07'00'

Dorothy Gordon, U.C Berkeley RBSP EFW DCB FPGA Engineer

Peter R Harvey 2010.03.23
16:06:41 -07'00'

Peter Harvey, U.C Berkeley RBSP EFW FSW

Michael M Ludlam Michael Ludlam
2010.03.22
13:54:41 -07'00'

Michael Ludlam, U.C. Berkeley RBSP EFW Systems Engineer

Signature Page

RBSP EFW

Data Controller Board (DCB) FPGA Specification & Functional Description

RBSP_EFW_DCB_001O

23 November 2009

References.

1. Interface Control Document (ICD) for the EFW Investigation, Applied Physics Laboratory (Dwg No. 7417-9083)
2. THEMIS Data Controller Board Specification and Functional Description (thm-dcb-001, Rev. A.06, 11 October 2004)
3. Zilog Z8400/Z84C00 NMOS/CMOS Z80 Specification
4. CZ80CPU Synthesizable HDL Core Specification (CZ80CPU-DES-1H18N00S00-200), Evatronix SA, April 26, 2007
5. RBSP_EFW_DCB_003E_Specification: Digital Control Board (DCB) Specification
6. RBSP_EFW_BPL_001M_Specification: IDPU Backplane Specification
7. RBSP EFW PCB Logic, Rev. B, June 23, 2008
8. RBSP_EFW_BEB_SCH_002A, Rev. A
9. Boom Electronics Board (BEB) Specification, RBSP_EFW_BEB_001F
10. AD5544/AD5554 DAC Specification, Analog Devices, 2004, Rev. A
11. NAND FLASH Memory MT29F16G08--- Specification, Rev. B, 2/07

Table of Contents

Document Revision History

Revision Number	Date	Change Summary
A	11 February 2008	Initial Draft
B	04 March 2008	Incremental changes
C	24 March 2008	Additional design details
D	18 April 2008	References to data repository removed (changed to FLASH); references to SSR removed (changed to SDRAM); addition of Baud Rate Selection for telemetry, protoboard debug hooks (Diagnostic Switch and LEDs); reduction of telemetry buffer page size to 4Kbytes, reorganization of the DFB DMA channel; additional design details and corrections.
E	10 June 2008	Added debug UART details, minor corrections Changed document name (added FPGA to title) Added Latchup Protection Description Modified FLASH Power-Switching Removed Housekeeping Mux Channel Assignment Table (now included in Reference 5).
F	06 August 2008	Added debug UART facility for S/C Interface (temporary for initial test) Added memory mapped I/F for FLASH and FLASH I/O based controls Modified DFB-DMA: as of this revision all channels can be directed to either SDRAM or SRAM.
G	09 September 2008	Removed Latchup Protection Slight modification to DFB CDI Status -- added error detect Reduced # of DFB DMA channels to 16 (APIDs 0x40 - 0x4F) Changed DFB DMA High Rate DMA Swap Option to 128Hz Added DFB DMA buffer overflow error detection S/C Telemetry I/F simplification (removed double-buffering)
H	21 October 2008	UART-DMA page size reduced to 1Kbyte Added details relating to DFB DMA and S/C DMA Subsystems
I	18 November 2008	Added details relating to the BEB and PCB Interfaces Added Board-ID and a few minor comments and modifications
J	10 December 2008	Modified BEB DAC Interface Defined ETU/FLIGHT Debug Interface Added FLASH subsystem details
K	27 January 2009	FLASH Subsystem description updated
L	27 February 2009	FLASH Subsystem Error Detection and Diagnostic Logic detailed SDRAM PowerOn Cntl restored to the DCBCtl Reg Description
M	18 May 2009	DIAG LEDs increased to 8-bit wide, DIAG SWITCH eliminated SDRAM ECC Multibit and Single Bit error counters cleared at the beginning of each scrub cycle Corresponds to Rev. C3
N	31 Aug 2009	Changed LVPS- SYNCH-CLOCK from 839KHz to 799KHz Corresponds to Rev. C4
O	23 Nov 2009	Eliminated S/C Telemetry High-speed Mode & Minor Corrections to document Corresponds to Rev. C5

1.0 Introduction

The Data Controller Board (DCB) is a component of the RBSP-EFW IDPU. It receives commands from the Spacecraft Processor and manages the Electric Field Instrument (EFI). The DCB communicates with the BEB, DFB and PCB subsystems via custom interfaces, receiving/storing data and forwarding instrument commands and register loads.

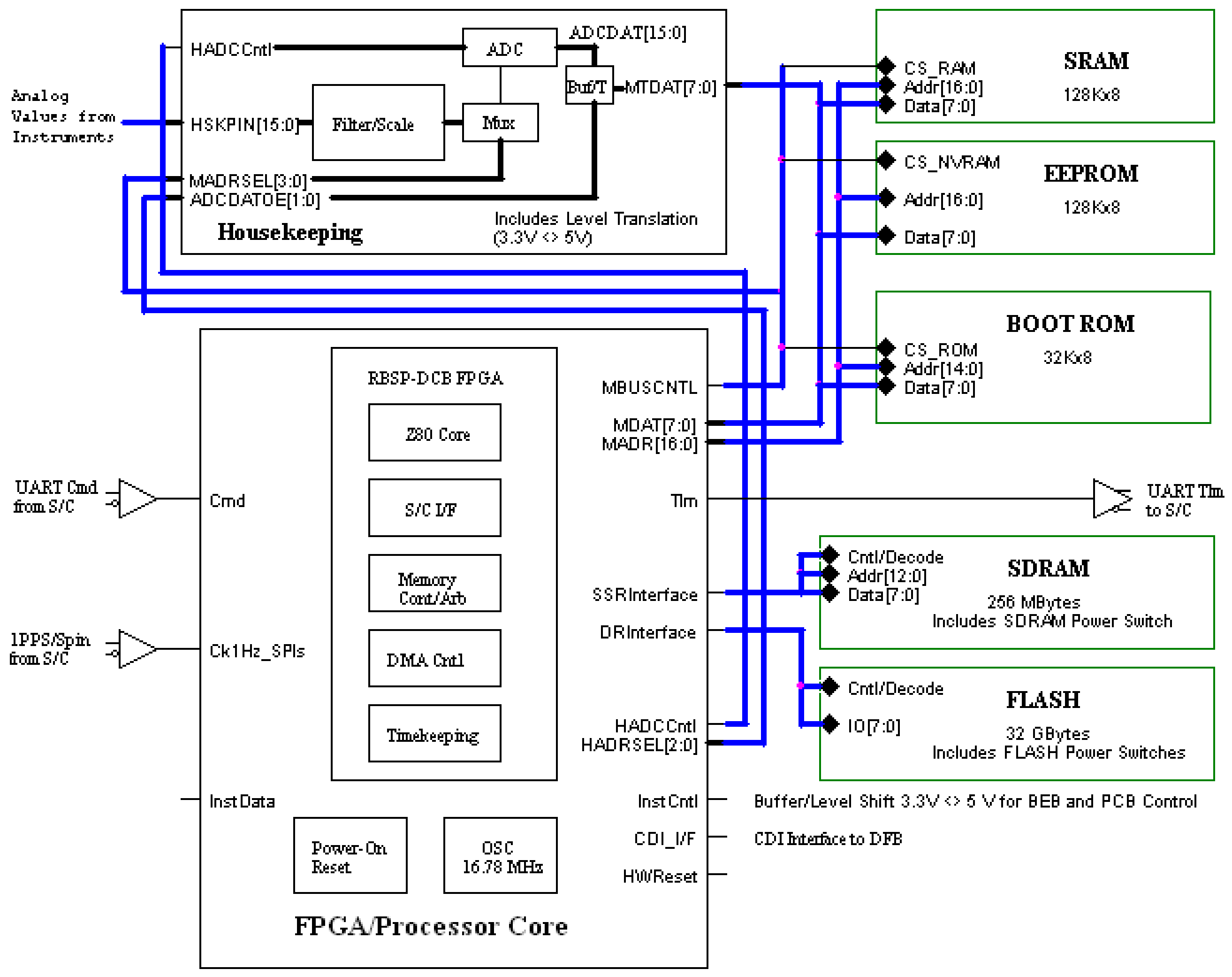


FIGURE 1. DCB: Overall Block Diagram

Figure 1 shows an overview of the main subsystems comprising the DCB. The microprocessor is a Z80 8-bit microcontroller (implemented in the FPGA via an IP Core). The flight FPGA is the Actel RTAX2000S, with the CAST Inc IP core instantiated as the CPU.

The CPU and logic subsystems are supported by an 32Kx8 Boot Rom, SEU-Immune Static RAM and EEPROM. (NOTE: the 128Kx8 SRAM shown is segmented and shared by the FPGA subsystems.) The FPGA handles the processor bus control and provides registers for accessing the various sections of memory. Also included in the FPGA are the Instrument I/Fs, the SDRAM controller, the FLASH Memory Controller, Error Detection/Correction for SDRAM and FLASH, the S/C Interface Logic, DMA and data management control, analog housekeeping control, and timing/time-tagging support.

The DCB resides on a 6U VME board, connecting to the backplane via the standard VME 96 pin connector. Power is received through the backplane connection, which is also used to communicate with the following IDPU based instrument interfaces and subsystems:

<u>The Digital Fields Board</u> (DFB) - IDPU resident interface board which talks to the EFI, SCM and MAG FGM. The Interface Board also performs processing and programmable filtering.

<u>The Boom Electronics Board</u> (BEB) - IDPU resident interface board, controls sensor biasing and modes. The BEB receives low level digital control signals and outputs analog telemetry.

<u>The Power Supply and Control Board</u> (LVPS/PCB) - IDPU resident subsystem includes the Boom Deployment Power Switching. The PCB receives low level digital control signals and outputs analog telemetry.

A harness is used to connect the DCB to the C&DH Board (S/C Processor). This C&DH Interface contains a UART based Telemetry Interface (115.2KBaud), a timing signal (1Hz Clock and "Spin Pulse"), and a UART (115.2Kbaud) based Command Interface.

2.0 Subsystem Descriptions

2.1 Clocks

The DCB receives one clock from the spacecraft, a 1 Hz tick (1PPS), correlated to the S/C time broadcast. (The 1PPS line is also used to transmit the SpinPulse. The differentiation is accomplished via pulse-width coding.)

The DCB internal timebase is set by a local oscillator (SCLK = 16.78MHz) which is used by the FPGA as the overall system clock. SCLK is further divided to generate the timebase clock used by the DFB(8.39 MHz), a 256HZ CPU Interrupt, and a 1Hz Synch (CLK1HZ) used by the DFB and DCB. A derivative of SCLK, 799KHz, 52.4%(HI)/47.6%(LO) duty cycle, provides a power converter synch clock.

2.2 Reset

The overall DCB reset, generated by the FPGA, is an OR of two sources: a power-on hardware reset (generated by an RC delay network and AC14 gate) and a watchdog reset.

A watchdog reset pulse (duration = 3 μs) is generated if the CPU does not write to the Watchdog Reset Clear register for a period of 3 seconds (3 ticks of the internally generated 1 Hz clock). An external jumper is provided, allowing for disabling the watchdog during debug. If the jumper is installed, the watchdog reset is disabled; otherwise the watchdog reset is always enabled. Watchdog Reset detect is available as status.

2.3 CPU

The CPU is an FPGA based Z80 microcontroller, running at 16.78MHz. Support circuitry in the FPGA includes bus timing and memory/arbitration/decode logic.

2.3.1 Debug Interface

The DCB debug connections allow for external diagnostic peripherals and monitoring of the CPU Bus.

The debug subsystem includes a RS232 compatible UART, mapped into Z80 I/O space and decoupled via 128 byte FIFOs in both directions. The Debug UARTs are detailed in their Register descriptions, starting on page 20.

The DCB prototype has been constructed using a reprogrammable Actel FPGA (the A3P1000, a ProASIC3 FLASH based device). The prototype board (different FPGA footprint than flight) includes enough testpoints to facilitate monitoring of the Z80-Core and an option for an external discrete Z80 processor. Logic analyzer and Debug-UART connectors included on the main board.

The ETU uses a dedicated "debug connector" that feeds the following signals to an auxiliary "debug board". The debug connector includes:

Debug Connector Net Name	**Function**	**Driven by**
MEMADDR(16:0)	Memory Address Bus	DCB-FPGA
MEMDATA(7:0)	Memory Data Bus	Both (BiDIR I/O)
MBUSCYCTYPE(3:0)	Debug decode (see below)	DCB-FPGA
MISCCNTL(11:0)	Debug control/indicators (see below)	Both (Split Inputs/Outputs)
LASTROBE(2:0)	Logic Analyzer Clocks	DCB-FPGA
DBG_NMI_PBSWITCH	NMI Input to CPU (debounced in FPGA)	Debug Board
DBG_RESET_PBSWITCH	RESET Input (debounced by RC circuit)	Debug Board
RxD	Debug UART Receive Port	Debug Board
TxD	Debug UART Transmit Port	DCB-FPGA
Vcc_3.3	3.3 V. Supply	connects to DCB supply via series resistor
GND	Ground	to DCB ground

The ETU Debug configuration allows the z80 CPU to boot from either the on-board ROM or an external debug board based ROM. Four of the MISCCNTL signals are allocated for this purpose. The remaining eight MISCNTL signals are used for LEDs.

Bus Assignment	**Signal Name/Function**	**Comments**
MISCCNTL(0)	ALTBOOTREAD	
MISCCNTL(1)	ALTBOOTWRITE	
MISCCNTL(2)	ALTBOOTCS	
MISCCNTL(3)	ALTBOOTSEL	Pulled up on DCB, Ground on Debug board via jumper for debug board PROM boot select
MISCCNTL(7:4)	DBG_LED(7:4)	Debug board based LED indicators - upper nibble
MISCCNTL(11:8)	DBG_LED(3:0)	Debug board based LED indicators - lower nibble

The MBUSCYCTYPE, used to facilitate Logic Analyzer tracing is defined as follows:

Bus Assignment	**Signal Name/Function**	**Comments**
MBUSCYCTYPE(3)	MBUS RD/WR	1=> READ; 0 => WRITE
MBUSCYCTYPE(2)	MBUS DMA/CPU Cycle	1=> CPU; 0 => DMA

MBUSCYCTYPE(1:0)	MBUS CYCLE DEF	During DMA access:
		`00 => DFB`
		`01 => TLM`
		`10 => FSH`
		`11 => CMD`
		During CPU access:
		`00 => SRAM`
		`01 => PROM`
		`10 => EEPROM`
		`11 => OTHER (CPU-FLASH or ADC)`

LASTROBE(1:0) are driven by MBUS Read and Write Strobes. LASTROBE(2) is a spare.

2.3.2 Interrupts

Unlike the 8085, the Z80 has a general interrupt signal designed for connection to one of the Z80 Peripheral Interrupt controllers. The “Mode 1” configuration provides one interrupt that vectors to 038H. Hardware subsystems that can interrupt the CPU are:

Interrupt Source	**Description**
TimerTick	Latched version of the timing interrupt (256 Hz).
Telemetry DMA	Latched version of TLM UART DMA done interrupt
FLASH Memory DMA	FLASH Memory DMA

The various interrupt sources are ORed together; the CPU can determine which interrupt(s) are active by reading the Status flags. Any of the three interrupts can be enabled/disabled by software, allowing the CPU to run in polled or interrupt driven mode on a per subsystem basis.

The proto and debug boards include an NMI Interrupt pushbutton switch, debounced by the FPGA.

2.4 CPU Memory

The CPU bus is directed by the FPGA to a memory bus containing a boot ROM (32Kx8 CMOS device), an external SRAM (128Kx8), and a EEPROM (128Kx8). (The EEPROM also contains internal protection mechanisms as well as an FPGA administered “write-protect”.) At reset, the processor fetches from address 0, where boot ROM is mapped. When the ROMON bit is cleared, SRAM is mapped into the lower section of CPU memory, but ROM is still accessible at an upper location in the linear memory map (so it can be used to store canned parameters, tables, etc.).

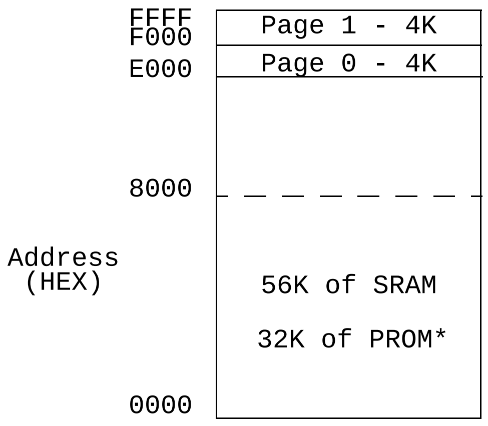
ROM and EEPROM accesses incur three wait states; SRAM accesses are zero wait states.

The Z80, with its 16-bit address bus, provides for a 64Kbyte memory space. In order to access the larger memory devices, the FPGA includes a memory paging mechanism, as described in the following section.

2.4.1 CPU Memory Map

The FPGA decodes Z80 addresses as follows:

For memory accesses (when the Z80 signal IORQ=0), the mapping is:



```
* Inst Space Decode: Lower 32K selects ROM only when ROMON = 1
  Selects SRAM only when ROMON = 0
```

Following a reset, the ROMON bit (read-writable via the Control Register) is asserted. When ROMON = 1, memory accesses to the lower 32K result in a fetch from ROM. When ROMON = 0, the entire non-paged memory area maps to the lower 56K of SRAM.

The upper 8Kbytes of memory addresses the full linear memory map via two FPGA based Page Registers, writable via the register interface. The Page Registers, address pointers spanning MemAdr[28:12], default to zero at reset. With this configuration a data access to upper CPU addresses such as 0xE000 actually result in an access to SRAM address 0. This may be useful for copying from ROM to lower SRAM at startup.

A “Low-Half Write Protect” control prohibits CPU writes to the lower 32Kbytes of CPU memory. Upon reset, writes are enabled; in order to disable, the CPU sets a Write Disable Control Register bit. The Low-Half Write Protect applies to both paged and non-paged addressing modes. DMA writes are always prohibited from the lower half of SRAM. Both CPU and DMA SRAM write violations are flagged and available to the CPU as status.

2.4.1.1 Full Linear Memory Map

Lower memory address are directly driven by the processor; upper memory address and chip selects are determined by the Page Registers.

The full linear map of external memory is shown below:

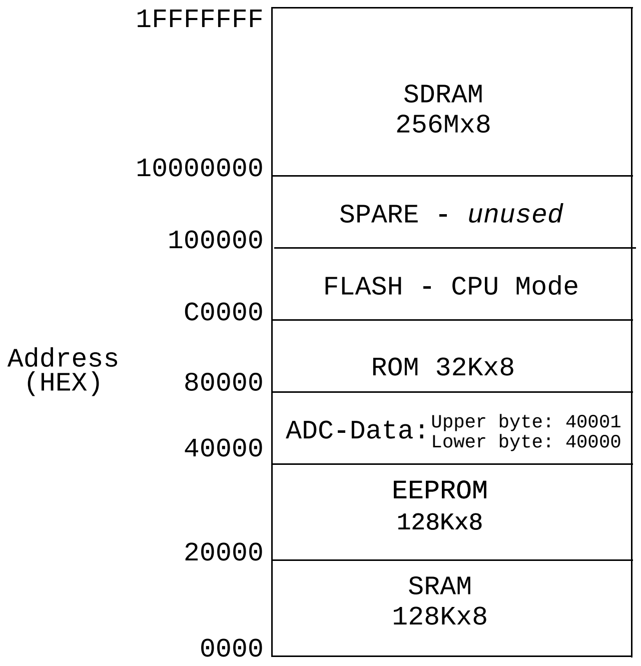


FIGURE 2. DCB Full Linear Memory Map

The Page Registers, along with the CPU address, are used to select a region in the linear memory map. The 17-bit Page Registers define pointers for MemAdr[28:12] of the Linear Memory Array. Note: bit 28 selects SDRAM.

Smaller devices (such as ADC-Data and FLASH CPU Mode) alias within their regions.

Accesses to unused areas result in a no-op bus cycle (normal bus cycle termination, but no chip selects are generated). Similarly, if SDRAM is accessed when powered off, a no-op cycle occurs.

Note: the upper quadrant SDRAM is reserved for ECC check-bits when Error Correction is enabled.

Null Accesses

If the CPU performs an access to an undefined area of the memory map, the bus control subsystem performs a 0 wait state cycle, but treats the access as a “No-op”. No chip selects are generated and reads will return indeterminate data. Accesses to SDRAM when the SDRAM has not been activated are treated as null accesses and flagged as such.

2.4.2 CPU I/O Space

Z80 I/O space, selected with IORQ=0, is consists of DCB FPGA based registers devoted to the various DCB & IDPU subsystems (i.e. DCB Control, CDI Interface, Analog Housekeeping). Register definitions appear below:

DCB-FPGA Register Definitions.

I/O Addresses allocated to the DCB based registers, are defined below:

NOTE: Unless otherwise noted, DCB-FPGA Register based counters start counting at zero.

Addr (hex)	Read Register	Write Register
10	Control (dcbCtl) - same as write value	Control (dcbCtl) - Bit 7 - Spare Bit[6:4]: Interrupt Enables (all default to 0, disabled, at reset) Bit 6 - Enable FLASH DMA int Bit 5 - Enable Tlm Int Bit 4 - Enable Timer Int Bit 3 - Memory Low Half Write disable - prohibits CPU writes to the lower half of Z80 memory space (DMA writes are always blocked from this region). Defaults to 0, writes enabled. Bit 2 - EEPROM write enable - defaults to 0, writes disabled Bit 1 - SDRAM Power: Controls 3.3V Power to SDRAM array and associated buffers. Default is 0 at power on reset -- SDRAM off. (NOT cleared by watchdog reset.) Bit 0 - ROMON - Boot ROM Disable: Controls mapping of the Boot ROM; reset (power on or watchdog) default is 1, ROM enabled
11	Page Register 0 (PgReg0Lo) - Lower Bits - same as write value	Page Register 0 (PgReg0Lo) - Lower Bits Bit[7:0] - CPU MemPage Addr[19:12]
12	Page Register 0 (PgReg0Hi) - Upper Bits - same as write value	Page Register 0 (PgReg0Hi) - Upper Bits Bit[7:0] - CPU MemPage Addr[27:20]
13	Page Register 0 (PgReg0SD) - SDSel - same as write value	Page Register 0 (PgReg0SD) - SDSel Bit[0] - CPU MemPage Addr[28]
14	Page Register 1 (PgReg1Lo) - Lower Bits - same as write value	Page Register 1 (PgReg1Lo) - Lower Bits Bit[7:0] - CPU MemPage Addr[19:12]
15	Page Register 1 (PgReg1Hi) - Upper Bits - same as write value	Page Register 1 (PgReg1Hi) - Upper Bits Bit[7:0] - CPU MemPage Addr[27:20]
16	Page Register 1 (PgReg1SD) - SDSel - same as write value	Page Register 1 (PgReg1SD) - SDSel Bit[0] - CPU MemPage Addr[28]
18	DIAG Port - (LEDReg/SwitchStat) - Bits[7:0] - SWITCH(7:0) (proto) Bits[7:0] - LEDS[7:0] (ETU/Flight)	DIAG Port - (LEDReg) - Bits[7:0] - LED(7:0) (ETU/Flight & proto)

Addr (hex)	Read Register	Write Register
1A	Auxiliary Status (aux_Stat) - Bit[7] - Spare Bits[6:4] - Board ID (set via external jumpers) Bit[3] - SDRAM_Active - Indicates SDRAM is ready (power on and initialization complete) Bit[2] - SDRAM Null Cycle - Indicates that a subsystem attempted and SDRAM access prior to SDRAM activation Bit[1] - CDI (DFB Cmd) Busy - CMDBUSY Bit[0] - CDI(DFB Cmd) CMDAVERRDET - Interface Error Detect Flag	*spare*
1B	Status Register (statReg) Bit[7] - Watchdog Reset Detect Bit[6] - DMA LH Mem Write Err Detect Bit[5] - CPU LH Mem Write Err Detect Bit[4] - CPU Bus Null Cycle Detect Bit[3] - Latched CLK1HZ (Sample Time) Bit[2] - Latched S/C PulErrDet Bit[1] - Latched SpinPulDet (from S/C) Bit[0] - Latched 1PPS Det (from S/C)	Pulse Register (pulReg) - Writing to this register with the appropriate bit(s) set causes the following actions: Bit[7] - Clear Latched Watchdog Reset Detect Bit[6] - *spare* Bit[5] - Clear Latched DFB Interface Error Detect Flag (CMDAVERRDET) Bit[4] - Clear Memory Error Detect (clears CPU Bus Null Cycle, CPU LH Mem Write Err, DMA LH Mem Write Err and SDRAM Null Cycle detects) Bit[3] - Clear Latched CLK1HZ Bit[2] - Clear Latched S/C PulErrDet Bit[1] - Clear Latched SpinPulDet Bit[0] - Clear Latched 1PPS Det
1C	Interrupt Status Register (intStatReg) Bit[2] - Latched TIMInt Bit[1] - Latched TLMDMAInt Bit[0] - Latched FLASHDMAInt	Interrupt Clear Register (intClrReg)- Bit[2] - Clear Latched TIMInt Bit[1] - Clear Latched TLMDMAInt Bit[0] - Clear Latched FLASHDMAInt
1D - 1E	*spare*	
1F	FPGA Version Number (fpgaVers) - Bits[7:0] *The protoboard FPGA Versions start at 0x80 (i.e. Rev. 0x84), the ETU-FLASH Revs start at 0xC0*	Watchdog Reset Clear (WATCHDOG) - a write of X5 (lower nibble = 5; upper nibble is a don’t cares) to this address “stomps” on the watchdog timer. This should be done at least every 2 seconds in order to prevent a watchdog reset.
20	Delta MET Latch Lo (dMETLo) - (MET to Sample Counter - low byte) Bit[7:0] - Internal Counter[16:9] (latched at the S/C 1PPS)	
21	Delta MET Latch Hi (dMETHi) - (MET to Sample Counter - high byte) Bit[7] - Seconds (rollover) Bit[6:0] - Internal Counter[23:17] (latched at the S/C 1PPS)	

Addr (hex)	Read Register	Write Register
22	Internal Counter Lo (ctrLo) - (Timebase Counter - low byte) Write (any data) to this address before reading the count Bit[7:0] - Internal Counter[16:9] (latched at write to this address)	(ctrLo) - Write of any data to this address latches the Timebase counter
23	Internal Counter Hi (ctrHi) - (Timebase Counter - high byte) Bit[7] - Seconds (rollover) Bit[6:0] - Internal Counter[23:17] (latched at write to ctrLo address)	
24	SpinPulse Time Lo (spTmLo) - Bit[7:0] - Internal Counter[16:9] latched at SpinPulse Detect	
25	SpinPulse Time Hi (spTmHi) - Bit[7] - Seconds (rollover) Bit[6:0] - Internal Counter[23:17] latched at SpinPulse Detect	
26	ADC Control (ADCCtl) Bit[7] - ADC Shutdown - defaults to zero (ADC in “nap mode”) at reset Bits[2:0] - AMUX address	ADC Control (ADCCtl) - Bit[7] - ADC Shutdown - defaults to zero (ADC in “nap mode”) at reset Bits[2:0] - AMXCH[2:0] - AMUX address
27		ADC Conversion Start (ADC_CONV) - Any write to this address pulses the SOC line of the ADC, causing a conversion. Data can subsequently be read back via Memory Mapped Registers (see Figure 2, “DCB Full Linear Memory Map,” on page 11)
28	Inst Cmd - DatLo - (CDIDatLo) - Bits[7:0] - CommandData (bits 7:0 of the 24 bit command word)	Inst Cmd - DatLo - (CDIDatLo) - Bits[7:0] - CommandData (bits 7:0 of the 24 bit command word)
29	Inst Cmd - DatHi - (CDIDatHi) - Bits[7:0] - CommandData (bits 15:8 of the 24 bit command word)	Inst Cmd - DatHi - (CDIDatHi) - Bits[7:0] - CommandData (bits 15:8 of the 24 bit command word)
2A	Inst Cmd - Command ID (CDIID) - Bits[7:0] - CommandData (bits 7:0 of the 24 bit command word).	Inst Cmd - Command ID (CDIID) - Bits[7:0] - CommandID (bits 23:16 of the 24 bit command word)
2B		Command Start (CDIStart) - Writing any value to this register starts the shift of the previously loaded command. (NOTE: The DFB CDI Registers (ID, Command Data and CDIStart) should not be written to when CMDBUSY bit is asserted. If this occurs, the writes are ignored and the and error flag, CMDAVERRDET, is set) (Command I/F Status is available at the AuxStat Register)
2C	PCBCmdData (PCBCmdDat) Bits[7:0] - CmdDat to be shifted to the PCB	PCBCmdData (PCBCmdDat) Bits[7:0] - CmdDat to be shifted to the PCB

Addr (hex)	Read Register	Write Register
2D	PCB Command (PCBCmdStat) Bit[1] - PCB Command Error (set if PCBCmdStart is issued illegally, in which case the command is ignored) Bit[0] - PCB Command Interface Busy	PCB Command (PCBCmd) - Bit[1] - ClrPCBCmdErr - Writing a one clears the PCB Command Error Flag Bit[0] - PCBCmdStart - Writing a one starts the shift of the data previ-ously loaded into the PCBCmdDat Register if the interface is not busy.
2E - 2F	*spare*	
30	ECC Control (eccCtl) Bit 7 - ECCSTATE -status (0-> scrubber has not yet completed initialization) Bit 6 - ScrubCSErrDet - Scrubber Chip Select Error detect - status feedback indicating that an SDRAM client has attempted to use the scrubber segment (upper quadrant) when ECC is enabled. Cleared by ECCErrClr Bits[5:4] - ECC Scrubber Upper Address - ScrubAddr[27:26] (also see Regs ECCAdr*n*) Bits[3:0] same as write values	ECC Control (eccCtl) - Bits[7:4] - *not used* Bits[3:2]: ECC Subsystem Scrub Period as follows: 0 -> 7.68µs 1 -> 250µs 2 -> 2ms 3 -> “ON-DEMAND” Default is 0 at Reset. Bit 1 - Scrub Test Mode - Defaults to zero at reset - 0-> Normal Mode, 1 -> Test Mode Bit 0 - Scrub Enable - Defaults to zero at reset, scrubbing disabled.
31	ECC SingleBit ErrCnt (SBErr) - Bits[7:0] - SBErrCnt[7:0]	ClearECCErrs (ECCErrClr) - Writing any data to this address the ScrubCSErrDet (read back at address 30) NOTE: ECCErrClr is asserted during SDRAM power-off periods.
32	ECC MultBit ErrCnt (MBErr) - Bits[7:0] - MBErrCnt[7:0]	ECCScrub Pulse (ECCScrPul) - Writing any data to this addresses causes a scrub cycle to occur when scrub period is set to “ON DEMAND”
33	ECC CheckBit Register (ECCCkBitsJam)	ECC CheckBit Register (ECCCkBitsJam) (used in testmode) Bits[7:0] - ECCCkBitsJam[7:0] Bit 7: Tag Bits[6:0]: Checkbits
34	ECC Scrubber Address (ECCAdr0) - Bits[7:0] - ScrubAddr[9:2] - Scrubber Status feedback - running address pointer (longwords)	
35	ECC Scrubber Address (ECCAdr1) - Bits[7:0] - ScrubAddr[17:10] - Scrubber Status feedback	
36	ECC Scrubber Address (ECCAdr2) - Bits[7:0] - ScrubAddr[25:18] - Scrubber Status feedback	
37	ECC Last Checkbits Read (ECCCkBitsRd) - Bits[7:0] - ECCCkBitsRd[7:0] - Scrubber Status feedback	

Addr (hex)	Read Register	Write Register
40	TLM Control Register (TLMStat) - Bit[6] - PageAddr[28] Bit[5] - Aliveness Status Flag Bit[4] - Power Down Request Flag Bit[3] - Spare Bit[2] - TLMBCErrDet Bit[1] - TLMBQErrDet Bit[0] - Enable TLM - reads back as written	TLM Control Register (TLMCtl) - Bit[6] - PageAddr[28] - MSBit of Next PageAddress Bit[5] - Aliveness Status Flag to be inserted into next packet Bit[4] - Power Down Request Flag to be inserted into next packet Bit[3] - Spare Bit[2] - Clear Error Detects (TLMBCErrDet & TLMBQErrDet) - pulse Bit[1] - Start Transmission of next queued TLM-Buffer - pulse Bit[0] - Enable TLM - defaults to 0, disabled (which holds the TLM sub-system in a reset state)
41	TLM Page Lo (TLMPAdrLo) - same as write value	TLM Page Lo (TLMPAdrLo) - PageAddress of Next TLM Tlm Buffer Bits[7:0] - TLM Page[19:12]
42	TLM Page Hi (TLMPAdrHi) - same as write value	TLM Page Hi (TLMPAdrHi) - PageAddress of Next TLM Tlm Buffer Bits[7:0] - TLM Page[27:20]
43	TLM Length Lo (TLMLenLo) - same as write value	TLM Length Lo (TLMLenLo) - Length (in Longwords) of Next TLM Tlm Buffer *(Reminder: Program to #Longwords - 1)* Bits[7:0] - TLM Length[7:0]
44	TLM Length Hi (TLMLenHi) & State Bits[6:4] - TLM State 000 -> IDLE; nonzero -> Active Bits[1:0] - TLM Page[9:8]	TLM Length Hi (TLMLenHi) - Length of next TLM Tlm Buffer Bits[1:0] - TLM Length[9:8]
45-47	*spare*	*spare*
48	TLM Tlm - CurAdr- B0 (TLM0CAdr) (Current Address Pointer) Bits[7:2] - TLMCurAddr[7:2]	NOTE: TLM Current address is a dynamic value and may change during the period required to read all four bytes. The FSW should make allowances for this possibility; when the buffer is actively being read out, the lower byte will increment approximately every 350μs at 115KBaud
49	TLM Tlm - CurAdr - B1 (TLM1CAdr) Bits[7:0] - TLMCurAddr[15:8]	
4A	TLM Tlm - CurAdr - B2 (TLM2CAdr) Bits[7:0] - TLMCurAddr[23:16]	
4**B**	TLM Tlm - CurAdr - B3 (TLM3CAdr) Bits[4:0] - TLMCurAddr[28:24]	
4C-4F	*unused*	*unused*

Addr (hex)	Read Register	Write Register
50	BEB_DACCTL (BEB_DACCTL) - Bit[7] - BEB DAC Interface Error Detect Bit[6] - BEB DAC Interface Busy Bit[5:2] - spare Bit[1:0] - DAC Register Address	BEB_DACCTL (BEB_DACCTL) - Bit[7] - Clear BEB DAC I/F Error Detect Bit[6] - DACXMIT - starts transmission of the DAC Command String if the interface is not busy. Bit[5] - DACLOAD - creates the LDAC pulse, causing the BEB DACs to transfer data from the holding registers to the DAC output registers Bit[4:2] - spare Bit[1:0] - DAC Register Address
51	BEB AMux Ctl (BEB_AMUXCTL) - same as write value	BEB AMux Ctl (BEB_AMUXCTL) - Bit[7:6] - spares Bit[5:4] - BEB_AMUXSEL[1:0] Bit[3] - spare Bit[2:0] - BEB_AMUXADR[2:0]
52	BEB AC Test Ctl Lo (BEB_ACTestLo) - same as write value	BEB AC Test Ctl Lo (BEB_ACTestLo) - Bit[7:0] - BEB_ACTESTFREQ[7:0]
53	BEB AC Test Ctl Hi (BEB_ACTestHi) - same as write value	BEB AC Test Ctl Hi (BEB_ACTestHi) - Bit[7] - Enable ACTEST2 Bit[6] - Enable ACTEST1 Bit[5:4] - spares Bit[3:0] - BEB_ACTESTFREQ[11:8]
54	BEBDAC IC0 Lo (BEB_DACIC0REGL) - Bits[7:0] - DACIC0Data[7:0]	BEBDAC IC0 Lo (BEB_DACIC0REGL) - Bits[7:0] - DACIC0Data[7:0]
55	BEBDAC IC0 Hi (BEB_DACIC0REGH) - Bits[7:0] - DACIC0Data[15:8]	BEBDAC IC0 Hi (BEB_DACIC0REGH) - Bits[7:0] - DACIC0Data[15:8]
56	BEBDAC IC1 Lo (BEB_DACIC1REGL) - Bits[7:0] - DACIC1Data[7:0]	BEBDAC IC1 Lo (BEB_DACIC1REGL) - Bits[7:0] - DACIC1Data[7:0]
57	BEBDAC IC1 Hi (BEB_DACIC1REGH) - Bits[7:0] - DACIC1Data[15:8]	BEBDAC IC1 Hi (BEB_DACIC1REGH) - Bits[7:0] - DACIC1Data[15:8]
58	BEBDAC IC2 Lo (BEB_DACIC2REGL) - Bits[7:0] - DACIC2Data[7:0]	BEBDAC IC2 Lo (BEB_DACIC2REGL) - Bits[7:0] - DACIC2Data[7:0]
59	BEBDAC IC2 Hi (BEB_DACIC2REGH) - Bits[7:0] - DACIC2Data[15:8]	BEBDAC IC2 Hi (BEB_DACIC2REGH) - Bits[7:0] - DACIC2Data[15:8]
5A	BEBDAC IC3 Lo (BEB_DACIC3REGL) - Bits[7:0] - DACIC3Data[7:0]	BEBDAC IC3 Lo (BEB_DACIC3REGL) - Bits[7:0] - DACIC3Data[7:0]
5B	BEBDAC IC3 Hi (BEB_DACIC3REGH) - Bits[7:0] - DACIC3Data[15:8]	BEBDAC IC3 Hi (BEB_DACIC3REGH) - Bits[7:0] - DACIC3Data[15:8]
5C	BEBDAC IC4 Lo (BEB_DACIC4REGL) - Bits[7:0] - DACIC4Data[7:0]	BEBDAC IC4 Lo (BEB_DACIC4REGL) - Bits[7:0] - DACIC4Data[7:0]
5D	BEBDAC IC4 Hi (BEB_DACIC4REGH) - Bits[7:0] - DACIC4Data[15:8]	BEBDAC IC4 Hi (BEB_DACIC4REGH) - Bits[7:0] - DACIC4Data[15:8]
5E-5F	*unused*	*unused*

Addr (hex)	Read Register	Write Register
60	UART CMD DMA Ctl/Stat (UDMA_STTS) Bit[7] - CMD I/F Framing Error Detect Bit[6] - CMD I/F Parity Error Detect Bit[5] - CMD I/F DMA Buffer Overflow Error Detect Bit[4] - CMD I/F DMA Timeout Error Detect Bit[0] - Enable UART CMD DMA - defaults to 0, disabled	UART CMD DMA Ctl/Stat (UDMA_CTL) Bit[7] - Clear Latched Cmd I/F Errors - (Pulse) (Framing, Parity, Timeout and Overflow - readable at bits[7:4] Bit[4] - Arm next CMD DMA transfer (Pulse) Bit[0] - Enable UART CMD DMA - defaults to 0, disabled
61	UART CMD PageReg (UDMA_CMD_PGA) - same as write value	UART CMD PageReg (UDMA_CMD_PGA) - Bit[6:0] - SCmdPAddr[16:10]
62	UART CMD LowCurAdr (UDMA_CMD_ADRL) - Bits[7:0] = SCMDCurAddr[7:0] (pointer to next address)	*UART-DMA index address spans 1Kbyte*
63	UART CMD HiCurAdr (UDMA_CMD_ADRH) Bits[1:0] = SCMDCurAddr[9:8] (pointer to next address)	
64 - 65	*spares*	*spares*
66	DFB Ctl Register (DFB_CTL) - Bits[7:4] - DFB Register Pointer Bit[3:1] - spares Bit[0] - Enable DFB DMA - reads back as written	DFB Ctl Register (DFB_CTL) - Bits[7:4] - DFB Register Pointer - selects one of the DFB DMA channels for Read or Write Access Bit[3] - Clear Latched DFB I/F Errors Detect (Pulse) (framing and parity, read back at Addr 67) Bit[2] - Clear All DMA-Channel Error Detect Flags - Buffer Overflow Detect and Memory Timeout Error Detect (read back at Addr 78 - Addr 7B) Bit[1] - Clear DMA-Channel Buffer Swap Detect Flags - (read back at Addr 76 - Addr 77) Bit[0] - Enable DFB DMA - defaults to 0, disabled (which holds the DFB DMA subsystem in a reset state)
67	DFB Interface Errors (DFB_IFERR) - Error flags from the two DFB TLM I/Fs Bit[3:2] - DFB Parity Error Detects Bit[1:0] - DFB Framing Error Detects (NOTE: All I/F Error Detects remain set until explicitly cleared by the CPU)	*spare*
The following register group Addr 68 - Addr 6F corresponds to all the DFB DMA channels. Register access is determined by the DFB Register Pointer, a DFB Ctl Register field.		
68	DFBCH Page Lo (DFBCH_PGAL) - Bits[7:0] - DFBCH Next Page[19:12] of the DFB channel pointed to by DFB Register Pointer	DFBCH Page Lo (DFBCH_PGAL) - Next PageAddr of for DFBCH Buffer pointed to by DFB Register Pointer Bits[7:0] - DFBCH Page[19:12]

Addr (hex)	**Read Register**	**Write Register**
69	DFBCH Page Hi (DFBCH_PGAH) - Bits[7:0] - DFBCH Next Page[27:20] of the DFB channel pointed to by DFB Register Pointer	DFBCH Page Hi (DFBCH_PGAH) - Next PageAddr of for DFBCH Buffer pointed to by DFB Register Pointer Bits[7:0] - DFBCH Page[27:20]
6A	DFB Current Address B0 - pointed to by Index previously set by FSW via DFB Register Pointer (DFBCAdr0) - Bits[7:1] = DFBCurAddr[7:2]	
6B	DFB Current Address B1 (DFBCAdr1) - Bits[7:0] = DFBCurAddr[15:8]	
6C	DFB Current Address B2 (DFBCAdr2) - Bits[7:0] = DFBCurAddr[23:16]	
6D	DFB Current Address B3 (DFBCAdr3) - Bits[3:0] = DFBCurAddr[27:24]	
6E	DFB DMA Last Buffer Status Word (DFB_LBSTATWDL) Bits[7:0] - DFBLBUFSTAT[7:0]	
6F	DFB DMA Last Buffer Status Word (DFB_LBSTATWDH) Bits[7:0] - DFBLBUFSTAT[15:0]	
The PageAdr28 Selects, Buffer Swap status/enables and Buffer Termination Selects for all the DFB DMA channels are accessed via "mask" registers (70 - 79). Lower byte mask: Bit 0 corresponds to DMA Channel 40, Bit 7 to DMA Channel 47; Upper byte mask: Bit 0 corresponds to DMA Channel 48, and Bit 7 to DMA Channel 4F		
70	same as write value	DFB PageAdr28 for DFB DMA Channels 40-47 (DFB_PGA07) Bits[7:0] - DFBCHP28ADR[7:0]
71	same as write value	DFB PageAdr28 for DFB DMA Channels 48-4F (DFB_PGA8F) Bits[7:0] - DFBCHP28ADR[F:8]
72	same as write value	DFB DMA Buffer Swap Enable - Channels 40-47 (DFB_BSWAPEN07) Bits[7:0] - DFBBSWAPENB[7:0]
73	same as write value	DFB DMA Buffer Swap Enable - Channels 48-4F (DFB_BSWAPEN8F) Bits[7:0] - DFBBSWAPENB[F:8]
74	same as write value	DFB DMA Buffer Termination Option - (DFB_BTERM07) 0-> 128 Hz; 1-> 1Hz Channels 40-47 Bits[7:0] - DFBBTERM[7:0]
75	same as write value	DFB DMA Buffer Termination Option - (DFB_BTERM8F) 0-> 128 Hz; 1-> 1Hz Channels 48-4F Bits[7:0] - DFBBTERM[F:8]
76	DFB DMA Buffer Swap Status - Channels 00-07 (DFB_BSWAPSTAT07) Bits[7:0] - DFBBSWAPSTAT[7:0]	

Addr (hex)	Read Register	Write Register
77	DFB DMA Buffer Swap Status - Channels 08-0F (DFB_BSWAPSTAT8F) Bits[7:0] - DFBBSWAPSTAT[F:8]	
78	DFB DMA Buffer Overflow Error Flag - Channels 00-07 (DFB_BOFLOWERR07) Bits[7:0] - DFBBOFLOWERR[7:0]	
79	DFB DMA Buffer Overflow Error Flag - Channels 08-0F (DFB_BOFLOWERR8F) Bits[7:0] - DFBBOFLOWERR[F:8]	
7A	DFB DMA Memory Timeout Error Flag - Channels 00-07 (DFB_MEMTO07) Bits[7:0] - DFBMEMTOERR[7:0]	
7B	DFB DMA Memory Timeout Error Flag- Channels 08-0F (DFB_MEMTO8F) Bits[7:0] - DFBMEMTOERR[F:8]	
7C-8F	*spares*	*spares*
90	Debug UART Ctl (uDbgCtl) *same as write values*	Debug UART Ctl (uDbgCtl) Bit[3:2] - RateSel - defaults to 2 115.2KBaud 0 => 38400; 1 => 57600 2 => 115200; 3=> 230400 Bit[0] - Enable Debug UARTs - defaults to 0, disabled
91	Debug UARTIN Status (uDbgInStat) Bit[7] - UART-DIN FIFO FULL - status Bit[6] - UART-DIN FIFO EMPTY - status Bit[5] - spare Bit[4] - UART-DIN SHIFTIDLE - (no shift in progress) - status Bit[3] - UART-DIN Parity Error Detect (flags parity error at the serial data input stage) Bit[2] - UART-DIN Framing Error Detect (flags framing error at the serial data input stage) Bit[1] - UART-DIN FIFO Full Error Detect (flags data reject due to overflow at the serial data input stage) Bit[0] - UART-DIN FIFO Empty Error Detect (flags attempt to read when no data is available)	Debug UARTIN Status (uDbgInStat) Bit[0] - UINERRCLR - write to Bit 0 of this address clears all the Debug UARTIN Status Error detects (read back in Bits[3:0] of this register)
92	Debug UARTIN FIFO - (uDbgInFDat) Bits[7:0] - UART Data Reads the next Debug UARTIN byte from the FIFO (if read occurs when the FIFO is empty, the UART-DIN FIFO Empty Error is set, see above)	

Addr (hex)	Read Register	Write Register
93	Debug UARTIN FIFO Remaining Bytes - (uDbgInFBCnt) Bits[7:0] - Number of bytes in Debug UART In FIFO	NOTE: A 128 Byte FIFO decouples the CPU from the UART-In data reception.
94	Debug UARTOUT Status (uDbgOutStat) Bit[7] - UART-DOUT FIFO FULL - status Bit[6] - UART-DOUT FIFO EMPTY - status Bit[0] - UART-DOUT FIFO Full Error Detect (flags attempt by CPU to write to a full UART DOUT FIFO)	Debug UARTOUT Status (uDbgOutStat) Bit[0] - UOUTERRCLR - write to Bit 0 of this address clears the UART-DOUT FIFO Full Error (read back in Bits[0] of this register)
95		Debug UARTOUT FIFO - (uDbgOutFDat) Bits[7:0] - UART Data Write the next Debug UARTIN byte to the FIFO (if write occurs when the FIFO is full, the UART-DOUT FIFO Full Error is set, see above)
96	Debug UARTOUT FIFO Remaining Bytes - (uDbgOutFBCnt) Bits[7:0] - Number of bytes in Debug UART Out FIFO	NOTE: A 128 Byte FIFO decouples the CPU from the UART-Out data transmission.
97-9F		Spares
A0	FLASH Control Reg (FLASHCtl) Bit[7] - FLASHWRENB Bit[6] - FLASHMODE Bit[5] - spare Bit[4] - FLASH Active (status only) Bit[3] - FLASH_ON/OFF Bit[2:0] - FLASHPWR[2:0]	FLASH Control Reg -(FLASHCtl) Bit[7] - FLASHWRENB - sets FLASH Write Enable - drives the FLASH subsystem WRPROT_N (during power sequencing, the FLASH Write Protect signal is controlled by the hardware) 0-> write disable; 1-> write enable (NOTE: should be set prior to initiation of a DMA Transfer to the FLASH Memory, or during CPU Diagnostic mode writes; all other times, this bit should be cleared.) Bit[6] - FLASHMODE - chooses between CPU memory mapped (0)and DMA (1) modes (defaults to 1: DMA control) Bit[5:4] - spares Bit[3] - FLASH_ON/OFF - 0 -> Power Off; 1 -> module selected by FLASHPWR Powered On Bit[2:0] - FLASHPWR[2:0] - 3-8 decode applied to enable one of the eight FLASH modules when FLASH_ON/OFF = 1
A1		Spare

Addr (hex)	Read Register	Write Register
A2	FLASH DMAOpCtl (FDMAOpCtl) *same as write values*	FLASH DMAOpCtl (FDMAOpCtl) Bit[7] - FLASH DMA Programming Failure Override -If a FLASH Programming failure occurs: (0 -> DMA Cycle terminates; 1-> DMA cycle continues) Bit[6] - FLASH DMA Timeout Override - If a FLASH Status timeout occurs: (0 -> DMA Cycle terminates; 1-> DMA cycle continues) Bit[5] - FLASH DMA Throttle (0 -> Full-speed; 1-> inserts additional delay of 700 ns between FDMA Requests) Bit[4] - spare Bit[3] - DMA Transfer Direction (0 -> Read Flash, 1 -> Write Flash) Bit[2] - FDMAMPage[28] -SRAM/SDRAM (to/from 0 -> SRAM; 1-> SDRAM) Bit[1] - FDMAMPage[11] - normally set to zero. If set to one, the DMA transfer starts in the middle of a 4Kbyte memory page. Bit[0] - Enable FLASH ECC
A3	FLASH DMA Start PAddr (FDMAStartPage) *same as write values*	FLASH DMA Start PAddr (FDMAStartPage) Bit[5:0] - FSH_SRT_PA[5:0]
A4	FLASH DMA End PAddr (FDMAEndPage) *same as write values*	FLASH DMA End PAddr (FDMAEndPage) Bit[5:0] - FSH_END_PA[5:0]
A5	FLASH DMAAddrBlock Lo (FDMAddrBlockLo) *same as write values*	FLASH DMAAddrBlock Lo (FDMAddrBlockLo) Bit[7:0] - FSH_BA[13:6]
A6	FLASH DMAAddrBlock Hi (FDMAddrBlockHi) *same as write values*	FLASH DMAAddrBlock Hi (FDMAddrBlockHi) Bit[6:4] - FSH_CS[2:0] Bit[3:0] - FSH_BA[17:14]
A7	FLASH DMA Status (FDMAStat) Bit[3] - FLASH Programming Failure Error Detect Bit[2] - FLASH Timeout Error Detect Bit[1] - DMA IF Error Detect Bit[0] - Transfer in Progress	FLASH DMA Pulse (FDMPulse) Bit[3] - Clear Programming Failure Error Detect Bit[2] - Clear FLASH Timeout Error Detect Bit[1] - Clear DMA IF Error Detect Bit[0] - Start DMA Transfer
A8	FLASH DMA Memory Page Lo - Start Addr (FDMAMAdrLo) - same as write value	FLASH DMA Memory Page Lo - Start Addr (FDMAMAdrLo) - Bits[7:0] - FDMAMPage[19:12]
A9	FLASH DMA Memory Page Hi - Start Addr (FDMAMAdrHi) - same as write value	FLASH DMA Memory Page Hi - Start Addr (FDMAMAdrHi) - Bits[7:0] - FDMAMPage[27:20]

Addr (hex)	Read Register	Write Register
AA	FLASH Current Page (FDMACurrPage) - current page transfer when DMA is active, last transferred page when DMA is complete Bits[5:0] -FDMAPAGE[5:0]	
AB	FLASH Latched Status (FDMALatStat) - holds last FLASH status register data read by the DMA subsystem Bits[7:0] - FSTATREG[7:0]	
AC - BF	spares - to be used for DMA Current Address and other status information	spare
C0	FLASH Correctable ErrCnt (FCorErrCnt) - Bits[7:0] -FCorErrCnt[7:0]	ClrFErrCounts (FLASHHCERRCLR) - Writing any data to this address clears the FLASH Correctable and NonCorrectable Error Counters.
C1	FLASH NonCorrectable ErrCnt (FNCorErrCnt) - Bits[7:0] -FNCorErrCnt[7:0]	
C2 - EF	spares - Feedback as status: Flash Controller State, Current Address Pointers, Latest FLASH Status Register readout and various error conditions	spares
F0	PROTO Board FLASH Power Switches - (PFLASHPower) *same as write values*	PROTO Board FLASH Power Switches - (PFLASHPower) Bit[3:0] - FLASHPWR[3:0] NOTE: should be set to either all off (0x0) or all on (0xF)
F1	PROTO Board FLASH Control/Status - (PFLASHCntl) *same as write values + two status bits* Bit[5:4] - RDY/BUSY[1:0]	PROTO Board FLASH Power Switches - (PFLASHCntl) Bit[7] - FLASHENB - Enables Drivers to Flash subsystem Bit[6] - FLASHMODE - chooses between CPU memory mapped (0)and DMA (1) modes Bit[3:0] - WRPROT[3:0] 0-> write protect; 1-> write enable
F2- FF		spares

2.5 Memory Bus Control/Arbitration

The CPU “MBUS” is home to the FPGA, SRAM, EEPROM, PROM and the ADC buffers. Accesses are arbitrated by an FPGA resident MBUS control subsystem. Normally, the processor owns the MBUS, which is used for code and data storage. The various DMA subsystems are allocated slots in between the CPU cycles. If the CPU requests access during a DMA cycle, wait states are inserted until the bus becomes available.

The MBUS Control/Arbiter allows the various subsystems to access SRAM using the a fixed priority arbitration as follows:

1. CPU   2. DFB-DMA 4. S/C (Cmd-DMA) 4. S/C (Tlm-DMA) 5. FLASH DMA

The DMA subsystems (accessing only SRAM) buffer data in units of four bytes before requesting MBUS access. FLASH DMA cycles may be "throttled" in order to control the average MBUS load.

2.6 Bulk Memory - SDRAM

The Bulk Memory, 256 Mbytes (one 256Mx8 module) of SDRAM (synchronous dynamic RAM), resides on a private memory bus, controlled by the FPGA.

SDRAM is powered by the 3.3V supply via a local switch (controlled by the CPU via the FPGA Register Interface).   There is no overcurrent monitoring for the SDRAM. (SEL LET > 56MeV/mg-cm^3) or FLASH (SEL LET > 50MeV/mg-cm^3).

Upon the application of power to the SDRAM array, the controller performs an initialization sequence after waiting (about 1/2 second) for power stabilization. Accesses prior to completion of initialization will be rejected, and indeterminate data returned to the requesting subsystem in the case of reads. If the attempted access was initiated by a DMA client, a flag (SDRAM Null Cycle) is set (available at the Aux_Stat Register). If the attempted access was by the CPU, the CPU Null Cycle Detect Flag is set. Both flags are clearable via a common pulse (Clear Memory Error Detect) register bit.

The FSW can verify that initialization has completed by checking the SDRAM_Active bit of the aux_Stat Register.

The SDRAM control circuitry, resident in the FPGA, manages refresh and access control. Most accesses to bulk memory, handled by the DMA controller, transfer DFB or FLASH data into the SDRAM or read SDRAM data out to the FLASH or Telemetry subsystems. The CPU can read/write the SDRAM array byte-wise via the Page Register facility.

The SDRAM manager allows the various subsystems to access SDRAM using the a fixed priority arbitration as follows:

1. DFB-DMA 2. S/C (Tlm DMA) 3. CPU 4. Scrubber 5. FLASH DMA

2.6.1 SDRAM Error Correction Circuitry (ECC)

The FPGA includes an error detection/correction (scrubber) subsystem, which uses the upper sections of the SDRAM to store the error correction codes. The scrubber operates on data 4 bytes at a time, generating a byte (ECC check-bits) in the ECC segment for every 32 bits in the data section. Seven of the check-bits are the ECC code; the 8th bit is a "Tag", indicating whether the scrubber has validated the 4 byte region of memory. CPU writes invalidate the checkbits (tag is cleared). The scrubber recomputes the checkbits and resets the tag when it gets back to the modified words.

The ECC scrubber is set to operate on the lower 200MBytes (yielding 49152 4K data segments) of SDRAM Module. The upper quarter of SDRAM, beginning at address (SDRAM Module Base + 0xC000000), is reserved for the checkbits. If any SDRAM client other than the scrubber attempts to read or write from the checkbit region while ECC is enabled, the ECC subsystem flags an error (ScrubCSErrDet, readable/clearable at the ECC Control Register). If an access error occurs, data writes are inhibited, and the corresponding “Tag” is invalidated. An illegal read returns indeterminate data (and illegal TLM reads do not scrub).

Following validation of the SDRAM array the scrubber reads the memory continuously. Single Bit errors are automatically corrected and counted. Multiple-bit errors are counted. The two counters (Single Bit and Multiple Bit Errors), readable via the FPGA Register Interface, are cleared by the hardware at the end of each scrub cycle. Each 8-bit counter “freezes” at the maximum count. The Scrubber Current Address is available as status, enabling the CPU to monitor the error counts periodically, and determine if SDRAM failures are address dependent. It may be possible for the Flight Software to adjust the scrub-rate based on error statistics.

ECC is also checked (and corrections made when possible) during 32-bit read accesses (scrubber, telemetry and FLASH-DMA subsystem cycles). If the data tags aren’t set, the controller writes the checkbits and tags the data. If the tags are set and a single-bit error is detected, corrected data is forwarded to the S/C Telemetry DMA subsystem and the SBErr Counter is clocked. If a multiple-bit error is detected, the MBErr Counter is clocked.

The ECC scrubbing rate can be set via the Control Register. The word-scrub rate can be varied from 7.6µs to 1ms in “free-run mode”, allowing for total array scrub time of between approximately 6.4 minutes and 14 hours.

An “On-Demand” rate is also available. Selection of the “On-Demand” rate causes the scrubber to be paused, while still retaining its state. In this mode, a scrub cycle is initiated via an FPGA-based register write. This allows for a diagnostic “backdoor”, a pause and resume mechanism, or for FSW managed slower scrub rates.

Upon startup, the scrubber begins the process of reading each value from SDRAM, and writing the corresponding checkbits into the upper quadrant of the SDRAM. When it has completed reading/verifying the lower three quadrants (assuming no CPU writes to SDRAM have occurred during this process) every checkbit tag has been set. At this time ECCSTATE is asserted, indicating that the scrubber has marked the entire array and is ready to begin checking. With ECCSTATE=1, most scrubber cycles will only read memory.

When ECCSTATE=1, if a single bit error is detected, the corrected value will be written back into the array. Multiple bit or indeterminate errors only clock the counter; no data is written back to memory.

NOTE: Since multiple bit errors are not corrected by the hardware, it is possible that they will clock the counter multiple times as the scrubber traverses memory repeatedly. Since the minimum scrub period is 6.4 minutes, it is possible to reset the counter and track errors in realtime. A stuck bit in the SDRAM array may show up as a single bit error (or possibly as a multiple bit error if it occurs in conjunction with another bit flip). Generally, a stuck bit is recognizable in the telemetry: the SBErr Counter will return a constant value over a group of scrub cycles; while for SEUs, the SBErr Counter fluctuation will be transient. (NOTE: Since the data is scrubbed as it is transferred to the spacecraft, a stuck bit may show up twice in one scrubber cycle when the packet is telemetered.) Stuck bits, generally correctable via an SDRAM power cycle, will accrue over time and be reflected in the SBErr Counter baseline per scrub cycle.

When ECC is enabled, the SDRAM subsystem blocks all non-scrubber SDRAM client accesses directed to the scrubber segment (the upper quarter). Instrument and CPU writes are ignored. S/C Telemetry and CPU reads return indeterminate data. If a scrubber segment write is attempted, the “Scrubber Chip Select ErrDet Flag” (status bit in the ECC Control Register) is set.

Note: If ECC is enabled prior to SDRAM initialization (evidenced by the assertion of SDRAM_Active (auxStat Register)), the scrubber request causes the SDRAM Null Cycle Detect flag to assert (also in the auxStat Register). This serves only as a flag that the ECC has been prematurely activated, and causes no other errors. Once SDRAM becomes active, the scrubber request is serviced and ECC continues unless disabled by the CPU. (The SDRAM Null Cycle Detect flag remains set until cleared by the CPU.)

2.6.1.1 Scrubber Testmode

Scrubber Testmode (invoked by setting the Scrubber Testmode bit of the ECC Control Register) allows the CPU to jam values into the upper quadrant. As additional diagnostic information, the CPU is able to read (both in normal and test modes) the checkbits corresponding to the last longword operated on by the scrubber.

When in Testmode, the only change in ECC Controller operation involves CPU Writes. Normally, a non-scrubber initiated write invalidates the tag. However, in Testmode, CPU Write accesses (always one byte) jam a “canned” byte, set by the CPU via the register I/F, into the checkbit quadrant of the SDRAM corresponding to the accessed longword. This feature, along with the “On-Demand” scrubber frequency, allows for extensive testing of scrubber functionality. The CPU can corrupt one or multiple bits of either the data or checkbit portion of memory and verify that the word is corrected (for single bit perturbations) and that the event is properly counted.

2.7 FLASH Memory

The FLASH Memory (32 GBytes) is composed of eight 4GByte Memory Modules (3D-Plus MMFN08408808S-D). Each 3D-Plus module includes eight 512Mx8 Micron FLASH Memory devices (MT29F4G08AAA). The array is broken down into eight banks (separately powered) with one module per bank.

The goal of the FPGA based logic is to streamline the intensive data-transfer operations between FLASH and the DCB-based random access memory banks. The CPU is responsible for ongoing FLASH management; the FPGA performs DMA transfers and error detection.

There are significant differences in the proto-board and ETU/Flight board FLASH capacities. The protoboard FLASH array consists of four discrete Micron devices and implements diagnostic mode only (due to the size limitation of the FPGA). Because of these differences, the protoboard has been allocated its own two FPGA registers for FLASH operation, and the ETU/Flight board and separate register set. The following description details the ETU/Flight board operation; protoboard operation incorporates the FLASH Diagnostic Mode command protocol.

Only one bank is powered-on at any one time.) The power switches are configured in the FPGA as a 3-8 decoder. FLASH_ON/OFF, a register control bit, allows the CPU to turn all the modules off. The default state following any reset (power-on and watchdog) is OFF.

The hardware asserts the Write-Protect pin during the power-cycle periods, and issues a "reset" command to all die following any power-on. A FLASH_ACTIVE status flag, returned for the powered array, indicates that the powered FLASH module is available (similar to the SDRAM_Active status bit). (This entire process requires < 5 ms.   Flash Power Switch ramp-up time is on the order of 1 ms.) Once a module is available, it can be used in DMA mode, or FLASH Diagnostic Mode (in which the CPU directly reads and writes the FLASH I/O registers using a memory mapped interface). When in DMA mode, the hardware controls the FLASH I/O and returns status via dedicated FPGA registers. In FLASH diagnostic mode, the CPU is responsible for issuing the necessary commands, monitoring status, and controlling the state of the write-protect signal. (NOTE: The default is FLASH DMA Mode. If   FLASH Diagnostic Mode is set via the register interface, the change takes effect when FLASH_ACTIVE is asserted and the DMA subsystem is inactive.)

2.7.1 FLASH Addressing

Since only one FLASH module is powered on at any one time, the upper three bits of the FLASH array address is simply the power switch setting (FSH_PWR[2:0]). The next three bits FSH_CS[2:0] determine the chip select for each 4Gx8 module. The remaining address bits, internal to the 512Mx8 flash die are:

FSH_BA[17:6] - block addr; FSH_PA[5:0] - page addr; and FSH_CA[11:0] - column addr

When setting up a FLASH DMA transfer, the CPU specifies FSH_PWR, FSH_CS, FSH_BA, and FSH_SRT_PA (start page address) and FSH_END_PA (end page address). The FLASH-DMA subsystem generates the necessary FLASH commands, and performs the transfer to/from the selected die from/to the selected region of SDRAM or SRAM. The destination memory address is set by programming the FPGA based FLASH DMA Memory Page Registers. FDMAMPage[28] determines whether the transfer is to/from SDRAM or SRAM while FDMAMPage[11] allows the transfer to start in the middle of a 4Kbyte memory sector. Since FLASH transfers can span 128Kbytes, the page address is generally not static (as for other DCB DMA clients) but increments over 4Knominal boundaries as the DMA progresses. If the maximum address is reached (for either SRAM or SDRAM), the page address will wrap.

2.7.2 FLASH Control

The FSW performs high-level flash management; the hardware provides facilities to make the repetitive tasks transparent. FLASH Memory, by its nature, must be managed. The smallest quanta is the "page" (2048 bytes + 64 byte). A "block" (128Kbytes + 4K bytes) consists of 64 pages. The user can program on a per page basis (set up sequential operations), but the minimum size for erasure is the block. Once a block is reallocated (and erased), it can be incrementally written to, either via program-page or random-data operations.

NOTE: Each FLASH device offers a "one-time programmable" area of ten full pages (which Micron says you can actually program four times). This region is separately mapped and accessed via separate "OTP" commands.

A “valid block” identification standard should be established and maintained. The industry standard identifies invalid blocks by writing a value other than 0xFF to the first spare byte (column location 0x800) on the first or second page of each bad block. Typically, factory identified bad blocks use 0x00, while user identified bad blocks use 0xF0. (Reference 11 states that the bad block indication is written into location 2048 “of the first or second page of each bad block” and “the system software should check the first spare address of the first and second page of each block” for bad block indication.) It is recommended that the hardware flag errors, but the software exclusively write the bad block indicator bytes. Since FLASH blocks might be erased erroneously, bad block indicator tables should be established and maintained (on the ground) for each DCB. Similarly, a duplicate of the Erase Count Indicator table should be maintained by the ground ops software.

The RBSP FLASH spare bit usage is:

```
0x800 - First Page & Second Page - Bad Block Indicator
  0xFF => Good Block; 0x00 => Factory Marked Bad Block; 0xF0 => User Marked Bad Block
0x801 - 0x803 - First Page - Erase Count Indicator
0x804+ - All Pages - Spares and ECC Information
```

The FLASH control is accomplished via a specialized DMA controller that transfers data in multiples of 2048 bytes (multiples of FLASH pages). The FLASH-DMA directive is:

Directive Name	**Parameters**	**Operates On**	**Comments**
PAGEXFER	n -Start Page, End Page Block Address, Chip Select	one block, up to 64 pages	Transfers data to/from CPU Memory space Up to 64 pages possible/transfer; cannot straddle the block boundary

ERASE, RESET, GET-DEVICE-ID and other simple FLASH commands can be implemented by the CPU in FLASH Diagnostic Mode (see Section 2.7.5 on page 31).

PAGEXFER operates on pages within one block only. The user can program up to 64 sequential pages, but one DMA transfer cannot straddle block boundaries. (For example, if the start page = 0x3C, only four pages can be queued.) If the user programs an illegal page combination (i.e. END PAGE < START PAGE) the FLASH DMA is terminates following the transfer of the maximum page address within a block (0x3F).

The FSW should check the bad block indicator prior to ERASE and PAGEXFER, and manage the Erase Counter (prior to erasing a block, the erase counter should be read, incremented and updated).

Bad block management is an on-going and essential process to the successful operation of the FLASH array. The FLASH Status Register should be checked following any ERASE or PAGE Write operation. If a failure has been detected, the block should be marked as bad. (Micron guarantees 4016 out of 4096 valid blocks over the lifetime of the device, given all other requirements are met. Of course, due to the radiation exposure, the RBSP FLASH arrays may degrade more unpredictably.)

2.7.2.1 DMA Subsystem Error Flags

The FLASH subsystem includes the following failure detection facilities: FLASH Interface Error Detect, FLASH Timeout Error Detect and FLASH Programming Failure Detect. Each failure detect flag is readable at the FDMAStat Register and can be cleared by writing to the FDMAPulse Register (it is up to the CPU to explicitly clear these flags).

The FLASH Interface Error Detect is set if the CPU attempts to write to a FLASH DMA Register, or start a DMA transfer during FLASH DMA Subsystem activity (when the "Transfer in Progress bit is set").

The Timeout Error is detected if the target flash device is not "Ready" within 4 ms following a command issue. This can occur following a reset command (which is issued at the start of each DMA operation), a page-read command, or a program-page termination command. If a Timeout Error is detected, the hardware halts the DMA transfer and interrupts the CPU. The "FLASH Timeout Error Detect" is set at each timeout instance. The "halt on error" function can be disabled by setting the "FLASH DMA Timeout Override" in the DMA Operation Control Register. With the timeout override bit set, the DMA subsystem continues following any status timeout. If status errors persist and the override is set, the DMA transfer will require quite a bit of time (since each status read hangs the system for 4 ms).

During a multipage DMA FLASH write, the hardware checks status following each page write. If an error is detected, the hardware halts the DMA transfer and interrupts the CPU. The "FLASH Programming Failure Error Detect Flag" is set for each page that returns bad status following a write operation. The "halt on error" function can be disabled by setting the "FLASH DMA Programming Failure Override" in the DMA Operation Control Register. With the failure override bit set, the DMA subsystem continues regardless of any error detects. The status flag, however is set following each page-program failure detect.

The FLASH Current Page the Latched Status Registers may help with diagnosing problems when errors are detected during DMA transfers.

2.7.2.2 FLASH DMA Transfer Timing

Page transfer time is expected to be limited by the SDRAM array bandwidth, as well as the FLASH page initiation times. FLASH-DMA is assigned the lowest priority into SRAM/SDRAM. Data is buffered in longword units prior to transfer to/from the SDRAM array. When the SDRAM is not busy, the transfer time for one page is:

one page DMA FLASH READ: 25µs + 690µs = 715µs (as measured on the ETU)

one page DMA FLASH WRITE: 220µs + 690µs = 910µs

The Page Read time was measured with SDRAM scrubbing on, but FLASH ECC disabled. When FLASH ECC is enabled, page read/write times will increase by about 10 - 15µs (depending on if a correction is required).

Flash Transfers to/from SDRAM require about 1 ms when the array is being accessed continuously by other high-bandwidth clients (including Scrubber, DFB and S/C Telemetry).

The basic SRAM transfer time is 670µs with CPU execution from PROM and 650µs with CPU execution from SRAM. (Add 25µs for actual Page Read time, and 220µs for actual Page Write time.)

There is an option to slow down FLASH DMA (to free up the memory busses for other clients). If the FLASH DMA Throttle is set, a 700ns delay is added to the normal delay between FLASH-DMA requests. Since FLASH is the lowest priority client, the throttle is probably not necessary for FLASH <=> SDRAM transfers. It may, however, be useful for FLASH <=> SRAM transfers in order to lessen the MBUS load and free up the bus for CPU accesses.

2.7.3 FLASH ECC

An Error Detection/Correction subsystem, providing single-bit error detection/correction and multi-bit error detection, operates (when enabled) on FLASH Pages. A portion of the upper 64 byte control region is allocated for the ECC checkbits. The page data area is divided into four 512 byte regions (ECC-SEG), each with its own Hamming ECC code. This requires three bytes of ECC code for each ECC-SEG. Since byte 2048 is used for bad block identification, and the next set of contiguous bytes are allocated for other FLASH metrics, the ECC controller begins the ECC code storage further into the array as follows:

0x830 - ECC Tag (Valid Tag = 0x42, if ECC has been performed during previous write operation)
0x831 - 0x833 - ECC Checkbits for First Quadrant
0x834 - 0x836 - ECC Checkbits for Second Quadrant
0x837 - 0x839 - ECC Checkbits for Third Quadrant
0x83A - 0x83C - ECC Checkbits for Fourth Quadrant
0x83A - 0x83C - ECC Checkbit Parity

ECC does not cover the extra bits, but the subsystem generates and checks parity on the ECC Checkbits. If the checkbit parity is bad, no correction is implemented and a non-correctable error is registered.

(When in FLASH Diagnostic mode, the FSW should either invalidate the tag, or update the ECC Checkbits when modifying data in the FLASH array.)

Following each page write, the ECC tag and checkbits are stored into the FLASH control region. Following any page transfer the checkbits are written and the tag set (even when ECC is not enabled).

Following each page read, if ECC is enabled, the ECC tag and checkbits are read out and ECC-SEG verification performed. If any errors are detected, the FLASH ECC subsystem determines whether the errors are correctable and rewrites the corrected values to DMA Memory. NOTE: The hardware does not correct the FLASH data (so as not to expend additional write cycles). The FSW may decide to correct FLASH (which can be done by initiating DMA in the reverse direction, writing back the corrected segment of memory to the corrupted FLASH segment).

ECC-Errors are logged for either type of error (single bit correctable or noncorrectable) and the appropriate counter clocked (counter limit of 256 freezes at maximum count). The CPU is informed, via status readback, whether ECC correction/detection has occurred during the previous DMA transfer. These counters can be cleared by the FSW via the register interface.

2.7.4 FLASH Interrupt

When enabled, a FLASH Subsystem Interrupt is generated upon completion of a DMA Transfer. If the DMA transfer terminates early, due to an error condition, the FLASH Interrupt is generated, and the relevant error flag is set. If FLASH interrupts are disabled, the interrupt flag is still available as status. The FLASH Interrupt must be explicitly reset by the CPU.

2.7.5 FLASH Diagnostic Mode

There is a provision to drive the FLASH memory directly from the CPU using a memory-mapped interface. To use this mode, the FLASH array must be enabled, and the FLASHMODE control bit set to zero, selecting CPU-Mode.

In FLASH diagnostic mode, the CPU can address the FLASH array at base address 0xC0000. For the protoboard configuration: the two chip selects in a module are selected with address bit 0. The four modules are selected with address bits[2:1]. CLE is selected with address bit 4; and ALE is selected with address bit 5. For example: command data to chip2, cs0 would use address 0xC0012; address information to chip 1, CS1 would address 0xC0023.

For the ETU/Flight configuration: the eight chip selects in a module are selected with address bits [2:0]. CLE is selected with address bit 4; and ALE is selected with address bit 5. For example: command data to CS5 would use address 0xC0015; address information to CS2 would address 0xC0022; data to CS1 would use 0xC0001. (As in DMA mode, the module is selected via FLASHPWR[2:0].)

In order to avoid ECC errors when the system is switched back into DMA mode, the FSW should either invalidate the ECC Tag or update the ECC checkbits as needed when modifications are made to the data portion of the array.

2.8 Timekeeping

The system clock is divided by two to generate the 8.39MHz (CLK8M) timebase clock.

CLK8M, the common timebase used by the DCB and DFB, is used for timekeeping and general communications. CLK8M is used by the FPGA for:

- The CDI clock, timing instrument command generation, message intake. It provides the DFB system clock.
- The Timebase Counter clock (used for internal DCB measurements and time-stamping).

The DCB clock is free-running and unrelated to the spacecraft clock. Spacecraft time is communicated to the DCB via the 1PPS signal and the time-command, transmitted once/second.

2.8.1 Timebase Counter

A 24-bit timebase counter (Sample Time), resident in the FPGA, is cleared by a hard reset and clocked by SYSCLK. The DCB generates an internal CLK1HZ, based on the Sample Time Counter rollover.

The Internal Counter (Sample Time) is available (CtrLo and CtrHi Registers) as status (first write to CtrLo to latch the counter data).

The internal (Sample Time) counter is used to generate a latched Timing Tick interrupt (at 256Hz). The latched TIMINT must be explicitly cleared by the CPU upon detection.

The CLK1HZ occurrence is broadcast to the DFB and latched as status. CLK1HZ Detect stays asserted until cleared by the CPU.

NOTE: Since CLK1HZ and CLK8M (DFBCLK) are both generated by SCLK, the DFB must sample CLK1HZ at the falling edge CLK8M (as shown in Reference 6).

The timebase counter is used to generate the “subsecond timing ticks” (at 64 Hz) used by the DFB DMA subsystem.

2.8.2 S/C Timing Signals (1PPS and Spin Pulse)

The DCB receives the 1PPS and the Spin Pulse multiplexed onto one control line. A deglitching circuit removes pulses of less than four clock periods (about 240 ns) before starting to time the incoming pulse. The S/C ICD (see Reference 1) states that the 1PPS pulse is nominally 80 µs and the Spin Pulse 40 µs. A valid (passing the deglitching stage) pulse is timed and sorted as follows:

```
width between 30.5 us to 53 us => SPINPULDET set
width between 69 us  to 91 us => 1PPSDET set
any other width => SCPULERRDET set
```

Both the 1PPSDET and SPINPULSEDET cause the timebase count to be latched as well as the status flags set. On the trailing edge of the detected pulse (transition from low-high at the connector), the upper 16 bits of Sample Time are loaded into the “Delta MET Latch” (for the 1PPSDET) or the “Spin Pulse Time Latch” (for the SPINPULSEDET).

After being read, the 1PPS, SPINPULDET and SCPULERRDET flags should be explicitly cleared by the CPU.

2.9 Housekeeping Subsystem

There is an analog-to-digital converter on the DCB board, along with one analog multiplexor (eight inputs). The mux channel is selected via the ADC Control (ADCCtl) Register. Typically the CPU sets the mux channel to the next channel to be sampled, waits until the switch and low-pass filter have settled, and then starts a conversion. A conversion is performed by writing (any data) to the ADC Conversion Start Register. The LTC1604 maximum conversion time is 2.8μs.

In order to optimize for low-power, the ADC should be kept in shutdown (nap mode, the default at reset) until the CPU is ready to perform a conversion. The ADC requires 200ns to “wake-up” following the deassertion of Shutdown (ADCCtl Register). Following conversion, the ADC should be shutdown (returned to “Nap” Mode) and the next mux channel pre-selected.

The CPU should allow 500μs for the voltage to settle at the output of the analog mux following a mux channel switch prior to performing a conversion.

See Reference 5 for the analog mux channel assignment.

2.10 PCB Interface

The PCB requires three digital control signals: PCB_CMD, PCB_CLK and PCB_STB. The FPGA logic generates the necessary control signals/logic and the signals are passed to the back-plane via logic translators/buffers.

The flight software sees the interface as an 8 bit data register(PCBCmdDat). Following a write to the data register, the CPU starts the transmission by writing to the PCBCmd register. A status bit (`PCBCmdStat`) informs the CPU of the interface status (busy/ready). If the CPU attempts to write the PCBCmdDat Register or issues a PCBCmdStart while the interface is busy, an error is flag set.

PCB_CLK, which runs only during command transmission, is 1.048MHz. Since the DCB uses a non-inverting buffer for the clock, while the PCB uses inverting buffers, data changes on the rising edge and is stable on the falling edge (as seen on the backplane), allowing the PCB 54ACS164 shift register to forward data on the inverted clock’s rising edge.

Data (PCB_CMD) is transmitted MSB first, true logic. PCB_STB (active high, lasting one PCB_CLK period) is issued following the transmission of the last data bit. Reference 7 details the command decoding; Reference 6 summarizes command timing requirements.

The FPGA also provides the PCB Power Converter clock: 799KHz, 52.4%(HI)/47.6%(LO) duty cycle.

2.11 BEB Interface

The BEB interface requires analog mux control, AC test stimulus signals and a facility to load the BEB resident DACs. (Analog Mux and DAC Register mapping are detailed in Reference 8 and Reference 9.) BEB signals, buffered via level translators on the DCB, are forwarded to the back-plane without inversion. The BEB board uses inverting buffers and therefore may receive the signals in an inverted state. In order to insure that the DCB-FPGA delivers signals to the BEB in their active true state, the BEB inversion for each set of signals is detailed below:

2.11.1 BEB Analog Mux Controls

BEB_AMUXENB[2:0] - no inversion (active high logic)

BEB_AMUXADR[2:0] - inverted

The DCB-FPGA therefore inverts the AMUXADR prior to its transmission.

Since no more than one enable should be active, the analog mux enable is implemented via a 2-3 decoder:

BEB_AMUXSEL	**BEB_AMUXENB[2:0]**
00	000
01	001
10	010
11	100

A guardband is inserted by the logic when the enable configuration is changed.

2.11.2 BEB AC Test Stimulus

BEB_ACTEST[2:1] - inverted

The inversion on the ACTEST signals is only relevant when the ACTEST is inactive. In this case the ACTEST signals are held to a logic high on the backplane (logic low at the output of the BEB inverting buffer).

Two independently gated signals are either inactive or a square wave ranging from 128Hz to 512KHz. (One global frequency select is used to drive both AC Test signals.) Independent on/off controls gate the individual AC test signals. Their activation/deactivation is synched to the next instance of 1Hz clock (sample time).

The frequency is programmed via a 12-bit field in the BEB AC Test Ctl Registers. The AC Test frequency is expressed by the following formula:

$$f\ [\mathrm{Hz}] = 524288/N$$

The value (N - 1) should be written to the AC Test Frequency field. Each count of the AC Test Frequency increases the test period by ~1.9µs (2^{-19}s):   0x000 selects 524 Hz, 1 selects 262 KHz, 0x800 selects 256Hz, 0xFFF selects 128Hz.

2.11.3 BEB DAC Controls

BEB_DACLDAC, BEB_DACCMD, BEB_DACCLK - all inverted

The DCB is responsible for loading and commanding the BEB DACs in compliance with the AD5544 serial protocol (see Reference 10). There are five AD5544 quad-DACs resident on the BEB configured in a “daisy chain”, each with four 16-bit registers. BEB_DACCLK, 1.048MHz, is active only during DAC loading. (Registers are shifted out in numerical order: 0 first, 4 last. See Reference 9 for a mapping of registers to the BEB-DAC Definitions.)

The CPU should write to five 16-bit FPGA based registers, one 2-bit DAC address field and issue a command “DACXMIT” that starts the data shift process (90-bits) using DAC-CLK and DAC-CS.

The shift process takes approximately 90 µs. An “Interface Busy” status flag is provided to the CPU via the register interface. If the CPU issues a BEB-DAC command or attempts to modify any of the BEB-DAC registers while the interface is busy, an error flag is set.

The CPU should perform the DACXMIT operation 4x to load all 20 DAC registers. Following the DAC register writes, the CPU should issue a command “DACLOAD” to generate the final strobe that transfers the data from the DAC holding to output registers. (NOTE: issuing DACXMIT and DACLOAD simultaneously causes the shift, but not the load.)

2.12 DFB Interface

The DFB communicates via the serial format defined in the RBSP EFW IDPU Backplane Specification (see Reference 6) for the physical layer and timing characteristics.

2.12.1 DFB Command Interface

Command transmission complies with the protocol defined in (see Reference 6): 1 start bit followed by the 24 bit command, followed by 1 parity bit (odd parity) followed by at least one stop bit.

The CPU can initiate commands by a series of register writes: CDIDATLO, CDIDATHI, CDIID and CDISTART to FPGA based registers. The CDISTART register write initiates transmission of the command defined by the DATA and ID registers.

The CMDBUSY status indicator is asserted when a command is shifting out. The CPU should not write to any of the Command Interface Registers when CMDBUSY asserted. CDISTART register writes are ignored when CMDBUSY is set.

If the CPU writes to any of the CDI registers when a command is shifting, the CMDAVERRDET error flag is set and remains set until explicitly cleared by the CPU (Pulse Register). Both CMDBUSY and CMDAVERRDET are available as status (AuxStat Register).

In addition to CLK8M (DFBCLK), the DCB provides DFB1HZ (the 1Hz Synch Pulse) to the DFB as per Reference 6.

2.12.2 DFB Telemetry Interface

DFB Telemetry consists of a series of message words, with instrument specific content. The DFB drives two identical and independent TLM-lines, delineating channels via “Data-IDs”. The lower-level message interface is described in Reference 6. Incoming DFB data is arbitrated between the two telemetry interface based on arrival order prior to being queued for DMA. Therefore, the DFB should never transmit the same Data-ID on the two TLM-lines simultaneously as this will cause an data sequence ambiguity. The same Data-ID, however, can appear on separate TLM-lines as long the transmissions are time-offset.

2.12.2.1 DFB Data

DFB data arrives in units of 24-bits. The first 8-bits are "Data-ID" and the remaining 16-bits are data. The Data-ID, which determines DMA channel, points the DFB-DMA Controller to the appropriate address register.

Each DFB serial telemetry interface is serviced by a dedicated reception subsystem. Once two words with the same APID have been received and are ready to be written into memory, the message subsystem requests access to the appropriate memory controller. All data is transferred via DMA (see below).

DFB Data reception logic flags parity and framing errors, but DMAs the data even when an incoming word registers an interface error. The controller takes no action (i.e. no attempt is made to reframe) in the event of an interface error. Parity and framing error detects are latched, and remain set until explicitly cleared by the FSW. (If the interface has truly lost framing, it is unlikely that data will be written into memory because the received DataID has probably been corrupted. The interface will regain framing when two consecutive data words are separated by at least 27 bits.)

2.12.3 DFB DMA

DFB DMA operates with 4096 byte page boundaries, intakes data from a serial interface and writes the data into a preprogrammed location in DCB memory (SDRAM or SRAM).

A total of 16 channels are allocated for the DFB; each can be configured to write either into SRAM or SDRAM. All channels are double-buffered; however, the selection between SRAM and SDRAM (PageAddr28) is not double-buffered.

Each channel is allocated a BufSwapEnb control bit as well as a buffer termination control field, separately configurable per channel as follows:

Timing Option	Buffer Termination
0	128 Hz
1	1 Hz

When enabled, the buffer swap occurs at the next occurrence of the selected Buffer Termination Pulse.

DFB DMA is double-buffered and terminated by a timing signal rather than a length or end-address. If the buffer fills before the selected timing pulse arrives, the data wraps, storing data in a circular manner until the DMA is terminated. The termination timing strobe is selectable for each DFB DMA channel. Each channel is allocated an Buffer Swap Enable bit, controlled by the FSW. When the enable bit is set, the buffer swap occurs at the next termination timing tick.

DFB DMA uses a Page Register (Address[27:12]) which selects a 4K segment. All channels can be located in SDRAM or SRAM, but this option is not double-buffered. Buffer location (in SDRAM or SRAM) and termination options are fixed, and should be chosen prior to startup.

The “Next Page Register” is initialized prior to startup and should be updated by the CPU while each current buffer is active. When the current buffer terminates, the “Next Page Register” is loaded into the “Current Page Register” and the index is set to a constant value representing the Packet Header Length, pointing the subsystem to the beginning of the next DMA segment.

DFB registers are accessed via a 4-bit pointer. A write to the DFB Control Register latches in the pointer value. Following the pointer write, the channel’s next address can be written and DMA current address can be read.

The Packet Header Length is a constant, set at 16 bytes.The CPU should write the header; each instrument DMA transfer will be written into memory immediately following the header.

If the buffer terminates when a 16-bit word is pending (since DMA is written in longword units), the DMA subsystem pads the last byte with 16-zeros, and completes the final transfer prior to swapping buffers. NOTE: Due to the possibility of a “stack-up” of 16 pending memory transfers, the channel buffer swap may lag (by up to 20 μs) the buffer termination strobe. The “Buffer Swap Status” (see below) reflects the delay; therefore, the CPU should poll these flags to assure proper synchronization. (NOTE: A potential race condition exists if the CPU clears the buffer swap register within this 20 μs window.)

Should the DMA address overflow (buffer fills prior to swap), the DMA subsystem alerts the CPU via an overflow error flag (one per DMA channel), and inhibits further writes until the next buffer swap occurs. The CPU should check/clear the overflow error flag when setting up the next buffer swap.

DMA status information can be read in two ways:

1. 16-bit flag Overview Registers that reflect the status of all DMA channels:

`DMA Buffer Swap Status, DMA Buffer Overflow Error Flag and DMA Memory Timeout Error Flag`

These three registers allow for a quick global look at all the DFB DMA channels. Swap status shows which of the 16-buffers have swapped since the last Clear-Pulse (issued by the CPU). The buffer overflow and memory timeout flags indicate error conditions. The overflow flag might indicate a configuration error; the timeout flag should not occur during normal operation and might indicate a hardware problem.

All of the Overview Registers are set by the hardware and remain set until explicitly cleared by the CPU (via a DFB Control Register write).

2. Last Buffer Status Word Register:

The last buffer status word displays the following information relating to the last buffer swap:

Bit[15]	**Bit[14]**	**Bit[13]**	**Bit[12]**	**Bit[11:2]**	**Bit[1:0]**
BufSwap	Timeout	Overflow	Odd	Index	Always = 0

Last Buffer Status is read on a per channel basis, by setting the pointer to the target DMA channel, and reading back the two 8-bit Last Buffer Status registers.

The Last Buffer Status Word flags (Timeout, Overflow and Odd), unlike the Overview Registers, do not stick; they truly reflect the last buffer status. (BufSwap, however, does remain set until pulsed clear by the CPU.)

Index points to the next longword that the channel would have written when the buffer closed. Odd indicates if the channel logic padded the last word (i.e. the buffer closure required the writing of the last 16-bit word). This allows the processor to determine how many DFB words were written to the previous buffer as follows:

```
 #DFB Words in Buffer = ((Last Buffer Index - 4) X 2) - Odd
```

An exception applies if an overflow has occurred. In this case, the last buffer index points to the last word written rather than the next word, and the number of DFB Words = 2040. (Words beyond the 2040 capacity are discarded during an overflow condition.)

If the timeout flag is set, the data buffer is very likely corrupted (missing words).

2.13 Spacecraft Interface

CMD and 1PPS_SPINPULSE are received from the Spacecraft; TLM is sent to Spacecraft from the DCB. Signals are buffered in accordance with Reference 1.

2.13.1 UARTs (CMD and TLM)

The UART is a standard 8-bit bidirectional UART operating at 115.2Kbaud (as described in Reference 1) using one start bit, 8 data bits followed by one parity bit. Parity is defined as “odd” in the following manner: the eight data bits plus one parity bit are forced by the parity bit to incorporate an odd number of ones. The parity bit is followed by one stop bit.

S/C data is further organized into Instrument Transfer Frames (ITFs) for both commands and telemetry. The DCB DMAs the S/C CMD data into a predetermined location in the CPU Memory, along with the ITF information. TLM data is transferred in “ITF Buffer Units”, with the hardware inserting the ITF Header and Checksum.

Detection circuitry filters/discards pulses of less than 240ns. This “glitch” filter operates on the entire command data stream prior to the start-bit/data-bit recovery stage. (It is also applied to the 1PPS/SpinPulse reception. See Section 2.8.2, “S/C Timing Signals (1PPS and Spin Pulse),” on page 32 for details.)

The CMD Interface records/latches two types interface errors: Parity and Framing. A framing error may be caused by either a runt start bit (most likely noise induced) or a missing stop bit. In the case of a parity error or missing stop bit, the received byte is DMAed into memory and the error condition(s) latched until the CPU explicitly clears them via writing to the UART CMD/TLM DMA Control register.

2.13.2 S/C Command DMA Interface

The Command DMA is contains a simple transfer protocol of writing/reading data from the UART to/from the DCB-SRAM. The maximum command buffer size is 1024 bytes.

There is a 50ms guard band guaranteed between the end of all command transmission and the next 1PPS assertion. The FSW must perform the buffer swap (set up the next CMD-DMA buffer) within this period.

NOTE: RBSP data is delivered encapsulated in “Instrument Transfer Frames” ITFs. The S/C Command DMA subsystem does not parse the Command Data ITF. ITF headers and checksums are simply transferred to memory along with the packet data. Checksum verification (along with verification of ITF length versus DMA data received, and checking the parity or framing error flags) should be performed by the CPU prior to validation of each ITF.

There are two types of errors registered by the CMD-DMA modules: Timeout and Buffer Overflow. Both errors are readable as status, along with the CMD UART errors; all are reset by a common clear pulse controlled by the FSW.

The timeout error, which occurs only in the case of a hardware failure, asserts if the CMD byte is not written into SRAM before the next byte begins shifting in. The assertion of the Timeout error is an indication of a corrupt command buffer.

The Buffer Overflow error asserts if the number command bytes received exceeds the 1024 byte allocation. It might reflect that the FSW has not rearmed CMD-DMA, or that the S/C has sent more commands than expected (or a lower level hardware problem). Upon overflow, no further commands are written into memory.

2.13.3 S/C Telemetry DMA Interface

The TLM interface is used to transfer CCSDS-formatted instrument data telemetry packets from the IDPU Memory to the spacecraft for transmission to the ground. Packetization is in conformance with CCSDS Packet Telemetry recommendations, and includes primary and secondary headers followed by a stream of data. Data is transmitted to the TLM UART in wrappers, called Instrument Transfer Frames (ITF). Each DMA unit (one or more packets) is encased in an ITF as described in Reference 1, as follows:

Byte Position	**Data**	**Comments**
0	0xFE (Sync Pattern)	32 bit sync pattern defined by Reference 1
1	0xFA (Sync Pattern)	
2	0x30 (Sync Pattern)	
3	0xC8 (Sync Pattern)	

Byte Position	Data	Comments
4	Bit 7: Aliveness Status Bit 6: Power Down Request Bit 5: Spare (set to 0) Bits[4:0]: MessageLength[12:8]	Aliveness Status and Power Down Request Flags are set by the FSW via the Register Interface. Message Length is derived from the programmed DMA Length as follows: Message Length = (NumLongwords x 4) + 4 *(where NumLongwords = TLMLen + 1, representing the number of longwords transmitted per ITF)*
5	Bits[7:0]: MessageLength[7:0]	
6	Bits[7:0]: FirstPacketHdrIdx[15:8]	always zero for RBSP EFW
7	Bits[7:0]: FirstPacketHdrIdxs[7:0]	
8 - N	Packet Data	Data DMAed from Memory
N + 1	Bits[7:0]: Checksum[15:8]	XOR of all ITF data following byte 3
N + 2	Bits[7:0]: Checksum[7:0]	

The CPU is responsible for setting up the DMA transfers and generation of the CCSDS headers. The FPGA transmits one or more variable sized CCSDS packets via the S/C TLM Interface, inserting the ITF header and trailer information as needed. The CPU is responsible for determining if there is enough time to transmit the next queued DMA unit (next ITF) prior to the next S/C 1PPS clock. (A 4Kbyte packet transmission requires approximately 0.35 seconds.) The CPU controls two of the ITF Flags (Aliveness Status and Power Down Request) via the Register Interface.

The Packet Data segment is defined by the CPU via a Page Register and an Index register. The Page Register, a constant value during each DMAed-Frame, spans bits 28:12 of the linear memory map. The Index register, cleared at the beginning of each DMAed buffer, counts longword addresses as DMA progresses.

In accordance with Reference 1, the ITF fields are transmitted most significant byte first. Packet data is transferred with no byte reordering (exactly as it is read out of memory: Byte 0, Byte 1, etc.)

TLM Address and Length registers are single buffered. The CPU must wait until the current transfer has completed before setting up the next transfer. The CPU must set the following registers for each TLM DMA transfer:

<u>TLM Buffer Length</u> - 10 bits specify the buffer length in longwords. TlmBuffer Length is used to compute the “Message Length” field of the ITF, compute transmission allowance (to meet the S/C guardband requirements) and to terminate the DMA. The maximum buffer length is 1024 longwords or 4096 bytes. The spacecraft ICD states that the message section of the ITF be <= 8192 bytes including the 2 byte checksum, inherently met by the buffer length restriction.

<u>TLM-Page</u> - 17 bits (sets Memory Page[28:12] for TLM-Buffer DMA). - indicates Start Page. NOTE: Buffer start is always aligned to a 4K page boundary (Adr[11:0] are cleared upon initialization of a TLM DMA buffer).

TLM buffers must always consist of an integral number of longwords.

When starting from a reset state, the CPU must first of all enable the interface (by asserting enable TLM). After setting-up the initial TLM registers, the CPU asserts the “Start Transmission” bit causing a buffer to be queued.

Once transmission has been started, it can be stopped by either:

- waiting until the current buffer transmission has shifted-out.
- disabling the interface - this is the most drastic way of shutting down TLM-DMA, and will immediately stop transmission - possibly halting mid-buffer (resulting in a partial ITF and packet).

The CPU can check the ongoing TLM-DMA Status by reading the Activity Status and CurrentAddress (including page and index sections).

TLM-Buffers should only be queued by the FSW during the Telemetry subsystem IDLE state.

The following error types is reported by the TLM DMA Interface:

BCERR - The 1PPS signal arrived during the transmission of an ITF. This might indicate a problem with the timing interface (i.e. fast 1PPS) or an improper DMA setup.

BQERR - A Buffer Queue error occurs if the FSW arms another buffer when there is still a pending transmission. In these cases the last “start buffer” is ignored, except for setting the BQERR status bit.

Both errors are fed back to the FSW via register bits and can be cleared via the TLM “Clear Error Detects” in the TLM Control Register.