KS10 FPGA Processor Manual

Revision 47

doyle (at) cox (dot) net

Table of Contents

List of Figures

List of Tables

1 Introduction

The Digital Equipment Corporation (DEC) KS10 was a low cost implementation of the popular PDP-10 mainframe computer.

-The goal of this project is to re-implement the KS10 using modern components and technology. This project will retain microcode compatibility with the DEC KS10 - this will increase the chances that this design will behave *exactly* like the DEC KS10 implementation.

The KS10 system, including the Central Processing Unit (CPU), Memory Controller, DZ11 Terminal Multiplexer, RH11 Massbuss Disk Controller, and Console Interface will be implemented in a single Field Programmable Gate Array (FPGA) instead using of boards of discrete logic.

The peripherals will be significantly different: modern peripherals like solid state Secure Digital High-Capacity (SDHC) disk drives will replace rotating magnetic media disk drives and 9-track magtape drives; Universal Serial Bus (USB) and Ethernet interfaces will be provided in addition to standard RS-232 devices.

This document is a compilation of many other documents. It is an attempt to gather all of the relevant information that is required to design this product into one place.

1.1 The DEC KS10

The DEC KS10 was implemented in 1978 using AMD am29xx TTL bit-slice device and 74LSxx SSI and MSI devices. The DEC KS10 had a 6.66 MHz clock cycle.

The DEC KS10 consisted of the following circuit boards:

- 4 board CPU set
- Console based on Intel 8080 microprocessor
- Memory Controller
- 8 Memory boards(64K x 36 with ECC)
- 2 Unibus Adapters
- Unibus-based Disk IO (RH11)
- Unibus-based TTY IO (DZ11)
- Power Supply

The CPU, Console, Memory Controller, and Unibus Adapters boards are all interconnected by the KS10 backplane bus.

1.2 The KS10 FPGA

This document describes an implementation of the DEC KS10 system using modern FPGA technology. The bulk of the logic is contained in a single FPGA. This FPGA requires support from a Console Processor and a memory device. For now, this FPGA implementation assumes a Console Processor with an 8-bit multiplexed address and data bus and it assumes a 36-bit wide Synchronous SRAM memory device. It is expected that this assumption will be revisited as the FPGA design evolves and matures.

The KS10 FPGA currently has a 20.0 MHz clock cycle.

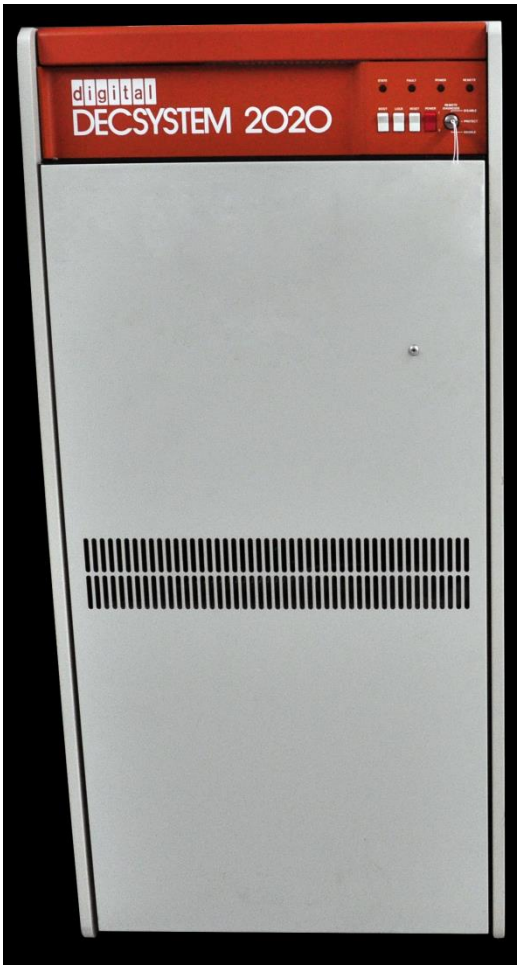


Figure 1 – DEC KS10

(photos from LCM)

Figure 2 – DEC KS10 with Covers Removed

(photos from RCIM)

2 The KS10 FPGA Architecture

The following sections describe the KS10 FPGA architecture.

2.1 The KS10 FPGA Design Hierarchy

The KS10 FPGA design hierarchy is illustrated below in Figure 3. The design hierarchy is only loosely based on the actual KS10 implementation. It is intended to describe the relationships between the Verilog modules.

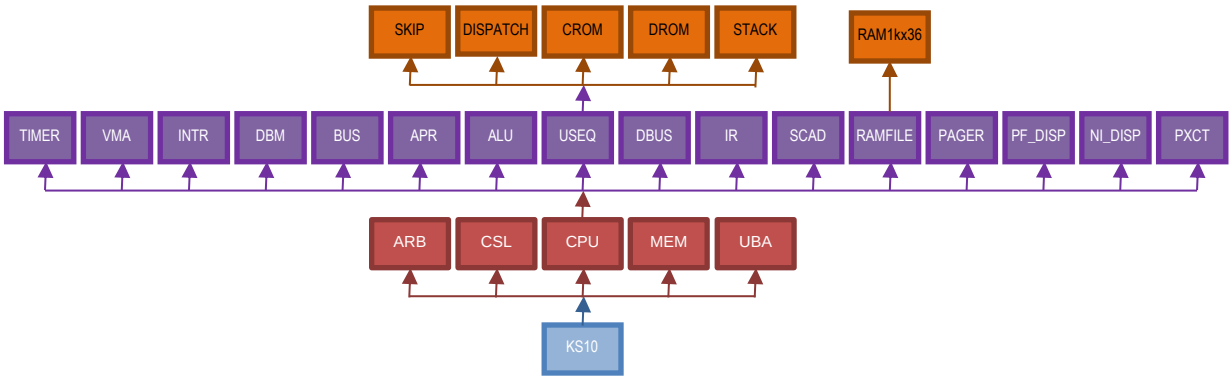


Figure 3 – KS10 FPGA Design Hierarchy

2.2 The KS10 FPGA Design Description

This section does not attempt to describe the operation of the DEC KS10. There are really excellent manuals that cover that sufficiently.

Instead, this section attempts to provide a brief description of the block and to document some of the design changes that were required to convert the original KS10 design into a design that could be implemented in the FPGA.

2.2.1 KS10 Bus Arbiter (ARB)

TBD.

2.2.2 KS10 Console Interface (CSL)

TBD.

2.2.3 KS10 Central Processing Unit (CPU)

The KS10 FPGA is organized a little differently that the DEC KS10. In many cases, the DEC KS10 was organized in a manner to minimize interconnections between circuit boards and to fill boards with circuitry.

The KS10 FPGA has neither of these constraints and attempts to organize logic based on function only.

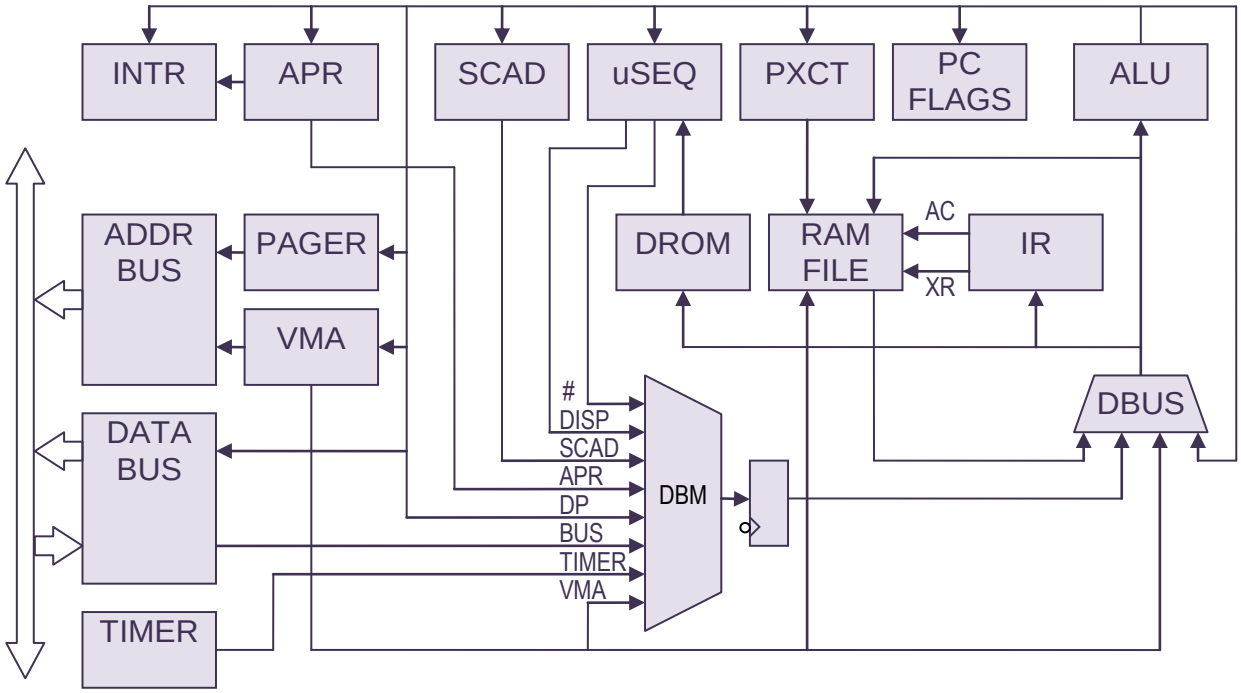


Figure 4 – KS10 FPGA CPU Block Diagram

2.2.3.1KS10 CPU Interval Timer (TIMER)

The PDP-10 Interval Timer is used by the Monitor to measure run elapsed time, run times, and time-of-day. The timer provides two basic units of time: a 10 microsecond clock where greater precision is required, and a 1 millisecond clock for normal operation.

A block diagram of the TIMER module is illustrated below in Figure 5.

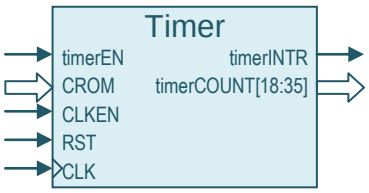


Figure 5 – Timer Interface Diagram

The KS10 implements this function by providing a 12-bit timer that is clocked at 4.1 MHz. The RDTIME instruction microcode reads the timer register contents and divides the result by 41 to support the 10 microsecond timing. The 4.1 MHz clock is divided by 4096 in the 12-bit timer which overflows every 0.999024 milliseconds. This timer overflow generates a Timer Interrupt to the CPU. You might note that the timer interrupt actually occurs 0.1 percent fast. This timing error is fixed by the microcode so that the time-of-date service is correct.

The Interval Timer Register is multiplexed into the CPU via the DBM Multiplexer. In the KS10 implementation (like the KS10), the Interval Timer Register input to the bottom half of the DBM is actually 18-bits wide: there are six bits of padding which are always zero, ten bits of timer, and the two timer LSBs which are always read as zero. This is illustrated below.

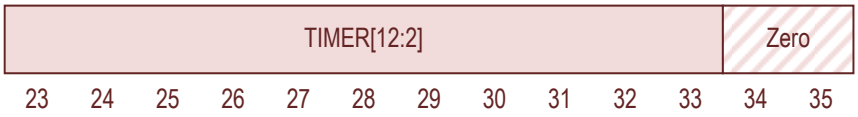


Figure 6 – Interval Timer Register

The microcode seems to read all 18-bits that are presented to the DBM and does not perform any masking. Whenever the Interval Timer Register is read, its two least significant bits are ignored so the register's contents approximately represent a count in microseconds. The upper 6 bits are just unused and unnecessary.

The time base value is a 71-bit value that is stored in the RAMFILE. The 12 LSBs of the time base corresponds to the Interval Timer Register. The Write Time Base instruction (WRTIM) can initialize the time base as a number of milliseconds but cannot alter the Interval Timer contents - the 12 LSBs are ignored.

The KS10 FPGA actually does not contain a 4.1 MHz clock source like the DEC KS10. The KS10 FPGA generates the 4.1 MHz clock enable signal using a 32-bit Fractional-N divider operating at 50 MHz. The Fractional-N divider uses an accumulator instead of a divider to create the output signal - the accumulator maintains the fractional time when the count overflows. The accumulator keeps the average output frequency correct although the output will jitter by 20 nanoseconds as the output is still synchronous to the input clock. The 32-bit accumulator was chosen so that the frequency error caused by the Fractional-N Divider implementation is less than the frequency error of the oscillator device. Gates are cheap.

As stated above, a Timer Interrupt is generated when the Interval Timer overflows which is approximately every millisecond. Once asserted, the Timer Interrupt is cleared by a microcode instruction.

A block diagram of the TIMER module is illustrated below.

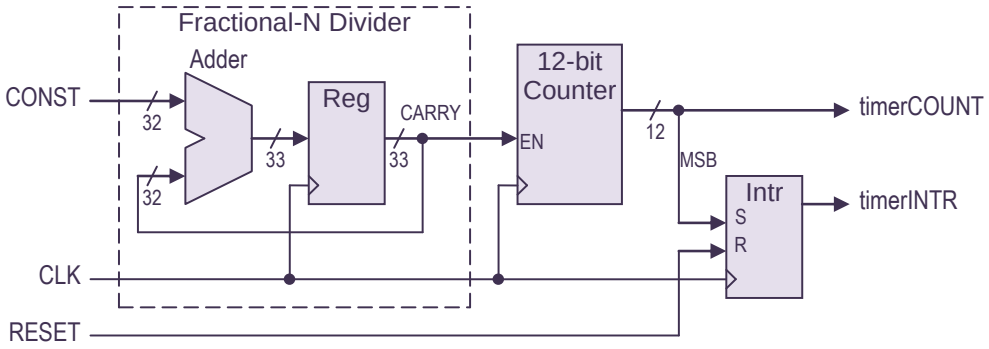


Figure 7 – Timer Block Diagram

2.2.3.2KS10 CPU Virtual Memory Address (VMA)

TBD.

2.2.3.3KS10 CPU Priority Interrupt Controller (PI)

The KS10 Priority Interrupt Controller is responsible for controlling the KS10 CPU's response to interrupt requests. The Priority Interrupt Controller maintains a notion of the Current Interrupt Priority, whether an individual interrupt is enabled and whether the Priority Interrupt Controller is enabled.

When an interrupt input is asserted, the interrupt controller determines if the new interrupt is of a higher priority than the current interrupt state. If it is, and interrupt request to the KS10 CPU is made.

The KS10 supports three interrupt sources which are enumerated below:

1. Arithmetic Processor (APR) Interrupts
2. IO Bus (UBA) Interrupts
3. Software Interrupts (Program Requests).

Each of the interrupt sources can provide 7 interrupt requests. Tb he highest interrupt request priority is interrupt 1; the lowest interrupt request priority is interrupt 7.

The Priority Interrupt Register state is stored inside the ALU Register 14 (octal). When the "specLOADPI" microcode instruction is executed, the Priority Interrupt Register state is loaded from the ALU into the Priority Interrupt hardware via the DP bus.

The format of the Priority Interrupt Register is closely resembles the operand of the Executive Mode RDPI instruction.

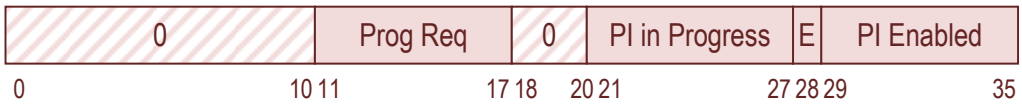


Figure 8 – Priority Interrupt Register Format

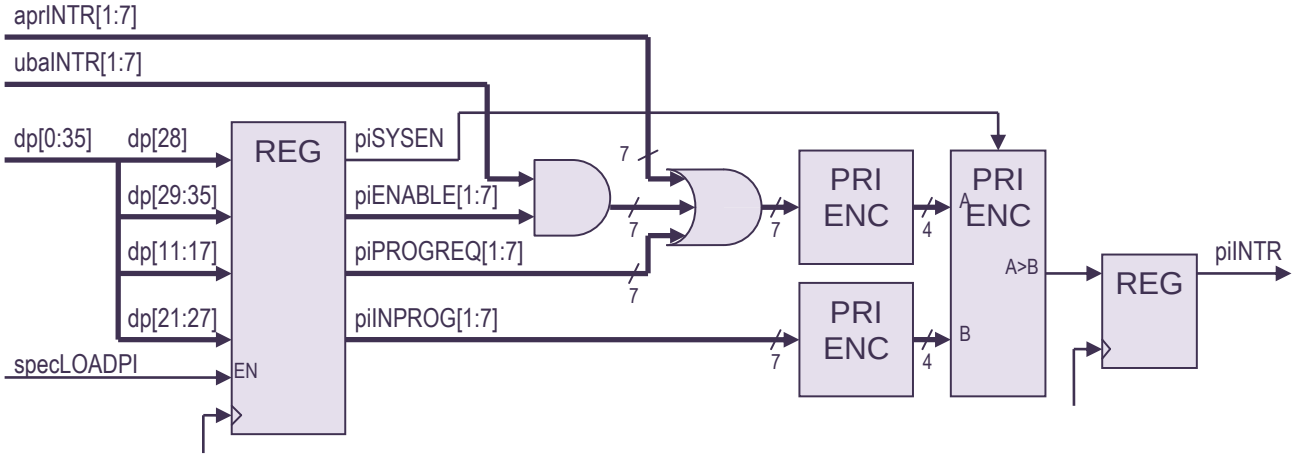


Figure 9 – Priority Interrupt Block Diagram

2.2.3.4KS10 CPU DBM Mux (DBM)

The DBM Mux is used to access the SCAD, Timer, VMA Register, Bus Interface, and the Number field of the microcode. It can perform left-half/right-half bus swap and it is also used to do hardware byte selection when the byte is exactly 7 bits.

The DBM Mux is controlled by the microcode DBM Select (DBM_SEL) field, the byte select mux is controlled by the microcode SPECIAL (DBM_SPEC) field.

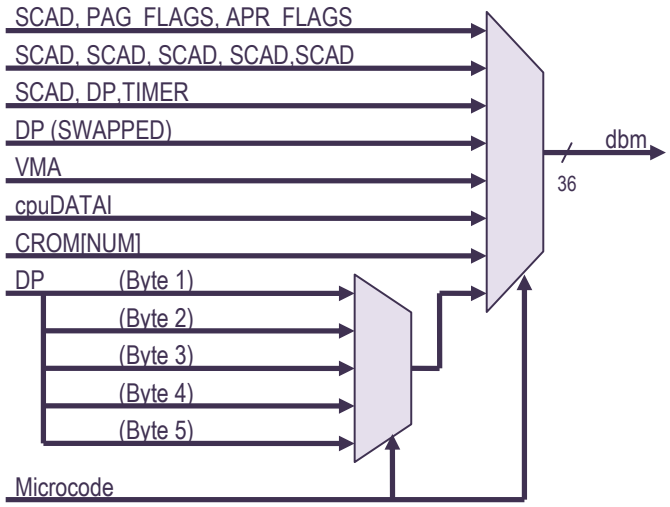


Figure 10 – DBM Block Diagram

2.2.3.5KS10 CPU Backplane Interface (BUS)

The CPU Backplane Interface is the interface between the CPU and the backplane peripherals.

The major function of the backplane is to determine the KS10 addressing mode. The KS10 understands four addressing modes. Namely:

1. Extended Address – This is a full 20-bit address.
2. Physical Address – This is a 18-bit address.
3. Paged Address – This is a 20-bit virtual address.
4. WRU Address – A "Who Are You (WRU)" cycle is part of the interrupt acknowledge bus cycle. It addresses whatever device is asserting the highest priority interrupt request.

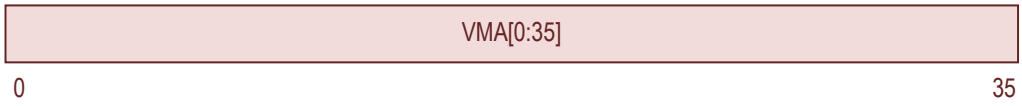


Figure 11 – Extended Address

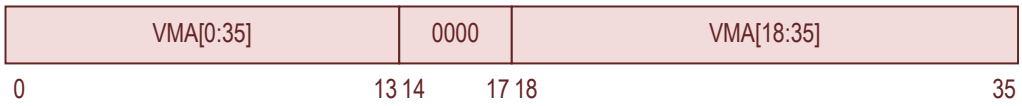


Figure 12 - Physical Address

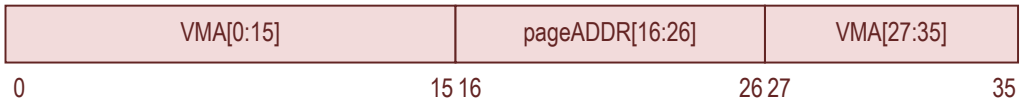


Figure 13 - Paged Address

Figure 14 - WRU Address

The Bus Address Multiplexer is simply a 36-bit wide 4-input multiplexer as illustrated below.

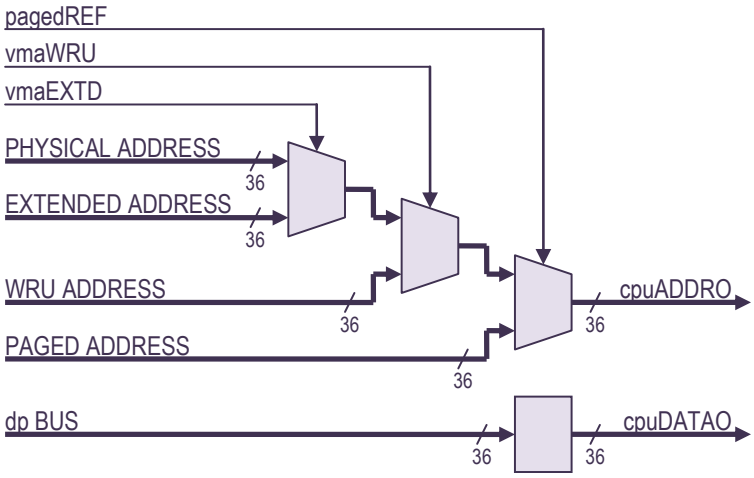


Figure 15 - Bus Interface

2.2.3.6KS10 CPU Arithmetic Processor Flags (APR)

The APR block manages the arithmetic processor flags. A block diagram of the APR module is illustrated below in Figure 16.

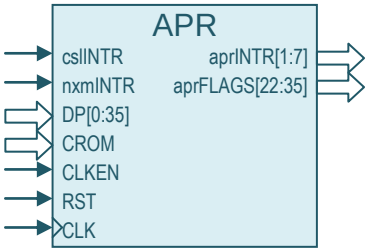


Figure 16 – APR Interface Diagram

The KS10 FPGA APR circuitry is implemented fairly closely to the DEC KS10 implementation.

The Executive Instructions that Control the APR are described in Section 15.1.1.

The aprINTR[1:7] outputs are routed to the Priority Interrupt (PI) block inputs.

The APR register bits are defined below in Table 1.

Table 1 – APR Flags	
Bit(s)	**Description**
22	Trap enable status. This bit is asserted when traps are enabled.
23	Page enable. This bit is asserted when paging enabled.
24	APR flag 24. This flag had no function on the KS10 but can be used as a software interrupt. This flag can be set or cleared by the microcode.
25	Interrupt to KS10 console. This bit is asserted when the KS10 wants to interrupt the console microcontroller. This flag can be set or cleared by the microcode.
26	Power fail interrupt. The power fail detection is not implemented so this flag is never set automatically. This flag can be set or cleared by the microcode.
27	NXM interrupt. This flag is asserted automatically when non-existent memory is accessed. This flag can be set or cleared by the microcode.
28	Uncorrectable memory error. Uncorrectable memory error (ECC) detection is not implemented so this flag is never set automatically. This flag can be set or cleared by the microcode.
29	Correctable memory error Correctable memory error (ECC) detection is not implemented so this flag is never set automatically. This flag can be set or cleared by the microcode
30	Interval timer interrupt The Interval Timer is implemented in microcode so this bit is so this bit is not set automatically. The microcode asserts this bit every 1.0 millisecond. This flag can be set or cleared by the microcode
31	Interrupt from KS10 console The Console Interrupt is implemented in microcode so this bit is not set automatically. This flag can be set or cleared by the microcode

Table 1 – APR Flags	
Bit(s)	**Description**
32	Interrupt request
33	Always set to 1
34	Always set to 1
35	Always set to 1

2.2.3.7KS10 CPU Arithmetic Logic Unit (ALU)

A block diagram of the ALU is illustrated below in Figure 17.

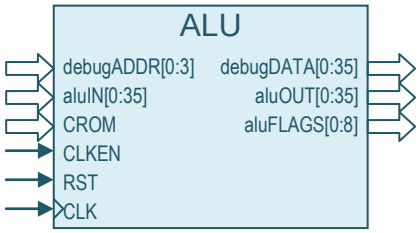


Figure 17 – ALU Interface Diagram

The DEC KS10 ALU implementation uses ten cascaded AM2901 4-bit processor slices. Some quick study showed that this did not work well with an FPGA implementation. Most FPGAs have optimized (very fast) carry logic that is provided to support counters and adders. This carry logic is much faster than the AM2902 parallel carry devices that supported the AM2901 slices in the KS10.

It turns out that the Verilog (and VHDL) synthesis tools could not infer that there was a single 40-bit carry chain from the Register Transfer Logic (RTL) description of the ten cascaded 4-bit slices. The resulting ALU was very slow.

One of the architectural features of the PDP10 ALU is the requirement to operate as a single 36-bit ALU or to operate as two independent 18-bit ALUs with no carry between the lower 18-bits and upper 18-bits. The DEC KS10 inserts an “AND Gate” in the middle of the carry string to implement this feature. This is illustrated below in Figure 18.

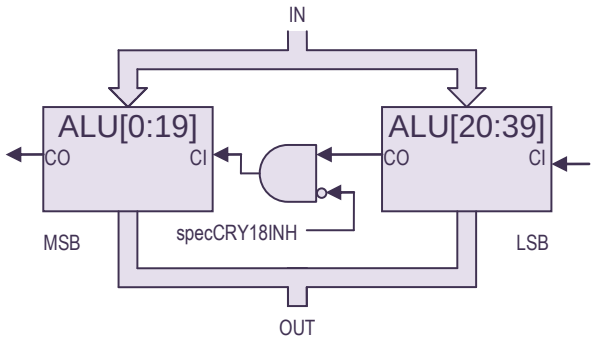


Figure 18 – DEC KS10 ALU Implementation

In the FPGA implementation, maintaining the integrity of the carry chain is very important to the speed of the ALU.

To that end, the KS10 FPGA ALU is implemented both ways: as a single 40-bit wide ALU and as two independent 20-bit ALUs. Gates are cheap. The two different types of operations occur in parallel and are selected at the module output. This is illustrated below in Figure 19.

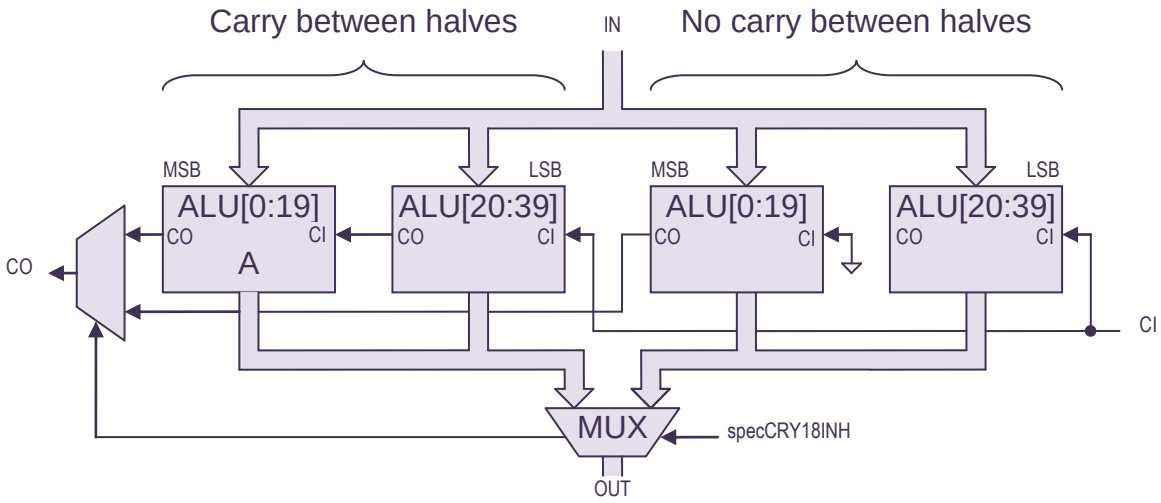


Figure 19 – KS10 FPGA ALU Implementation

Both of these ALU operations occur in parallel. Instead of the microcode enabling the carry between the two halves of the ALU, the microcode selects between the 40-bit ALU with the carry between the two halves and the 40-bit ALU with no carry between the two halves. This is illustrated in Figure 18 above.

This saves ‘cutting’ the 40-bit ALU carry chain in the middle and maintains the ALU speed.

This design change actually makes the description of the KS10 FPGA much simpler than the alternative. The operation of the 40-bit Q-shifter and the 40-bit F-shifter is much more visible than in the original DEC KS10 design.

So why is the ALU 40-bits wide instead of 36-bits?

Notice that the ALU is sign extended by two bits on the left, and zero padded by two bits on the right.

The ALU is implemented with 4-bit wide slices. This implementation gives the hardware access to the CRY2 (Carry 2) signal which is available at the interface between the first and second ALU chip. If this was implemented as a 36-bit wide ALU, this signal would be buried inside the am2901 chip and not accessible. There may be other signals also.

2.2.3.8KS10 CPU Microsequencer (USEQ)

The microsequencer is a device that sequences through the microcode. The microsequencer has various inputs that control the execution sequence but the only output from the microsequencer block is the Control ROM which provides the KS10 microcode. This microcode is used to control the various blocks of the KS10 hardware.

The interfaces to the microsequencer are captured below in Figure 20.

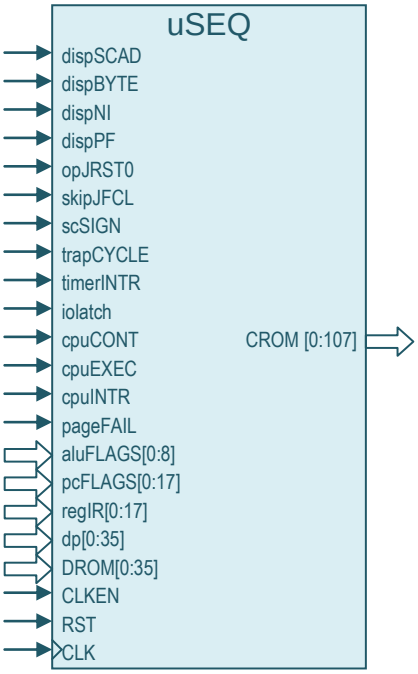


Figure 20 – Microsequencer Interface

The DEC KS10 CPU was implemented using 2048 words of 108-bit wide horizontal microcode (as opposed to vertical microcode). The microarchitecture supports 12-bit addressing of microcode (4096 words) but the DEC KS10 only implemented 2048 words microcode. The KS10 FPGA implements all 4096 words of microcode.

The microcode begins execution at address 0000. The microsequencer continuously re-executes instruction at address 0000 while the RESET signal is asserted and will only execute the next instruction after the RESET signal has been negated. This design assumes that the RESET negation is synchronized to the clock and that the RESET signal is asserted for a few clock cycles minimum to ensure that the instruction at address 0000 has been executed at least once.

The Page Fail hardware is hard coded to vector to the Page Fail handler address. The Page Fail vector hardware is a bunch of “OR Gates” which just assert all of the microcode ROM address bits - therefore the Page Fail handler is located at 3777 (octal) for the DEC KS10 and 7777 (octal) for the KS10 FPGA. In general, the hardware addressing (Reset, Page Fail, Skip, and Dispatch) must match the addresses in the microcode. Therefore the hardware cannot be modified without appropriate changes to the microcode.

The addressing the microsequencer is very minimalistic. Whereas a modern implementation might use a multiplexer to control the addressing, the KS10 uses simple “OR” gates. Therefore the SKIP, DISPATCH, and PAGE FAIL logic can only modify the addressing by setting bits – never clearing address bits.

On a normal microcode instruction, the SKIP, DISPATCH, and PAGE FAIL addresses are zero. The Control ROM supplies the next address from the microcode “Jump” field.

When a SKIP is to be performed (a primitive conditional branch-like instruction), the SKIP entity conditionally supplies an address of 0000 or 0001. This conditionally sets the LSB of the address. Of course the microcode must be designed such that the two destination addresses are appropriate.

The DISPATCH is similar to a SKIP except that an N-way branch can be executed. Again, the microcode must be designed such that all of the possible destination addresses are correct. The microcode uses a 512-way branch to quickly decode instruction opcodes, and additional N-way branches to quickly decode addressing modes. The opcode dispatch address is provided by the Dispatch ROM (DROM) – which is not part of the microsequencer illustrated below.

The PAGE FAIL always vectors the microcode to address 7777.

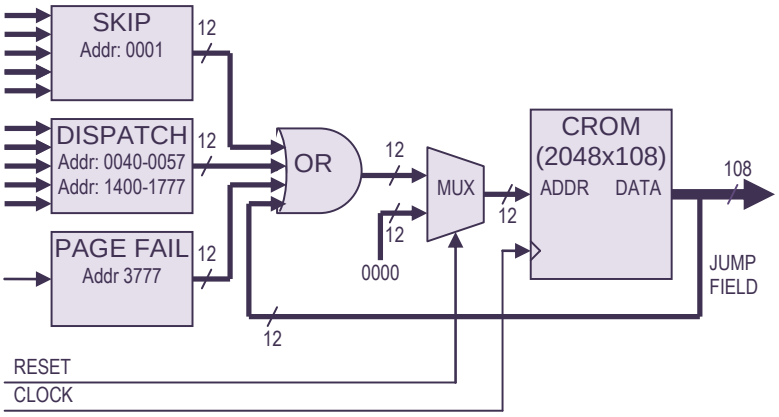


Figure 21 – Microsequencer Block Diagram

2.2.3.8.1 KS10 CPU Microsequencer Control ROM (CROM)

The Control ROM contains the executable microcode of the microsequencer.

This module of the KS10 FPGA microsequencer is implemented significantly different than the DEC KS10 circuit but performs exactly the same function.

The DEC KS10 implementation has several implementation issues which must be addressed in an FPGA design. These are:

1. The microcode is stored in a RAM which must be loaded by the console processor at power-up. This is complicated and unnecessary. The FPGA can provide ROM for this function.
2. The Control RAM is asynchronous. The FPGA provides synchronous memory.

In the KS10 FPGA, the Control RAM is replaced by a synchronous ROM which is not writeable. This simplifies the boot procedure. The Control RAM microcode is post-processed to remove unused fields and rearrange the bits (I assume to optimize the hardware design). For example, the microcode listing defines a 108-bit wide microcode word whereas the hardware is only 96-bits wide. In the KS10 FPGA, the post-processing step has been elided as it is unnecessary. The Verilog synthesis tool is smart enough to identify and remove unused (or constant) data fields in the ROM. Also, the post-processing also adds parity which is not really necessary for the FPGA.

The DEC KS10 microsequencer had a 12-bit address which could have supported 4096 words of microcode; however, only half of the memory was actually implemented in the production hardware. Unfortunately the microcode grew to be larger 2048 words. Therefore DEC shipped three version of the microcode – each matching a specific application. The various types of microcode are detailed below in Table 2.

Fortunately all three versions of microcode were derived from a single codebase using conditional compiles to remove microcode as required.

It was desirable for the KS10 FPGA to have the microcode in ROM. The additional microcode memory allowed the KS10 FPGA to support a functional superset of all of the DEC microcode.

Implementing the new version of microcode only required some minor modifications to the conditional compiles and some new build command files. A small design change was required to move the PAGE-FAIL entry point from 3777 (octal) to 7777 (octal).

When a page failure occurs, the PAGE-FAIL address is generated in the hardware by setting all of the microcode address bits to one (using 12 OR gates). Because the MSB of the address was unused and the microcode was limited to 2048 words, this created a PAGE-FAIL entry point of 3777 (octal) in the microcode. Once the full microcode memory was implemented, the correct entry point of 7777 (octal) was exposed to the microcode.

Table 2 – KS10 Microcode Variations

FILENAME	Description
KS10.MCR	Diagnostic microcode. Includes TOPS10 paging and TOPS20 paging but does not include the UBABLT instructions.
T10KI.MCR	TOPS10 microcode. Includes TOPS10 paging and UBABLT instructions but does not include TOPS20 paging.
T10KL.MCR	TOPS20 microcode. Includes TOPS20 paging and UBABLT instructions but does not include TOPS10 paging.
CRAM4K.MCR	***NEW*** Unified microcode. Includes TOPS10 paging, TOPS20 paging, and UBABLT instructions.

The Control ROM contents are extracted from the microcode listing file by an AWK script.

2.2.3.8.2 KS10 CPU Microsequencer Dispatch ROM (DROM)

The Dispatch ROM maps the instruction (from the Instruction Register) to the address of microcode in the Control ROM that decodes and executes that instruction.

The DEC KS10 implementation of the Dispatch ROM is problematic for an FPGA implementation because the DROM in the KS10 is asynchronous and FPGA memories are synchronous.

Fortunately the DROM is addressed by the Instruction Register (IR) which *is* loaded synchronously. Therefore we can absorb a copy of the OPCODE portion of IR directly into Dispatch ROM addressing.

Simply put: when we load the IR, we also simultaneously and synchronously lookup the contents of the Dispatch ROM.

The Dispatch ROM is a 512 x 36 bit synchronous ROM.

As with the DEC KS10 Control ROM, the DEC KS10 Dispatch ROM contents is post-processed to remove unused fields and rearrange the bits. In the KS10 FPGA implementation, the Dispatch ROM contents match the format of the microcode listing and the post-processing step is elided. Again, the Verilog compiler is smart enough to remove unused microcode fields from the ROM

The Dispatch ROM contents are extracted from the microcode listing file by a simple AWK script.

2.2.3.8.3 KS10 CPU Microsequencer Dispatch Logic (DISPATCH)

As stated above, the dispatch logic allows the microcode to perform an N-way branch based on a set of inputs.

The dispatch block is a large multiplexer that is controlled by the microcode that provides a 12-bit dispatch address output.

The multiplexer is broken into two halves: the upper 8-bits are controlled independently from the lower 4-bits. This segregation into halves provides many 16-way dispatches and a few larger (up to 512-way) dispatches.

The instruction opcode decode dispatch is a 512-way dispatch where the address is supplied by the dispatch ROM.

The KS10 FPGA dispatch block replicates the DEC KS10 dispatch logic with one minor exception. The DEC KS10 dispatch logic was fairly convoluted (and very difficult to understand) in order to minimize logic and limit the size of the dispatch ROM.

For example, the instruction opcode decode dispatch is constrained to be in the address range between o1400 and o1700 in the microcode. Therefore in the DEC KS10 design during the opcode decode dispatch, the logic was hardwired to generate addresses in this address range. In the KS10 FPGA, the entire dispatch address is stored in the dispatch ROM. The logic synthesis tool is ‘smart enough’ to determine that the data contents of those ROM bits is constant for all addresses, remove the data from the ROM, and replace the ROM contents with hardwired logic just like the DEC KS10 – except that the design intent is much more evident in the FPGA version.

Changes to the dispatch logic would require changes to the microcode.

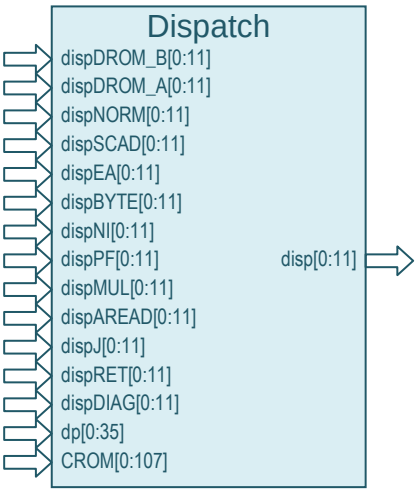


Figure 22 – Dispatch Interface Diagram

2.2.3.8.4 KS10 CPU Microsequencer Skip Logic (SKIP)

The skip logic provides a means for the microcode to perform a conditional jump operation based on a Boolean value.

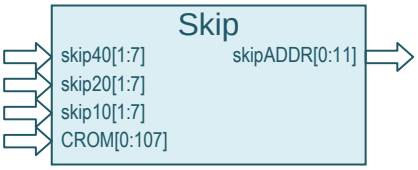


Figure 23 – Skip Interface Diagram

The skipADDR output is always either 0000 (octal) for “no skip” condition or 0001 (octal) for a “skip” condition.

Skips always occur from even addresses to odd addresses because the SKIP block can only OR the address. It is not a multiplex operation.

2.2.3.8.5 KS10 CPU Microsequencer Call/Return Stack (STACK)

This stack provides a mechanism for the microcode to ‘call’ and ‘return’ from microcode functions. The microsequencer stack is implemented quite a bit differently than the KS10 simply because the FPGA provides Dual Port RAMs.

The addrOUT 'read' port of the Dual Port RAM provides the return address. This port always points to the top-of-stack and can always be read independently.

The addrIN 'write' port of the Dual Port RAM is used to store the next 'call' address. This port always points to the address past the top-of-stack and can always be written independently.

An interface diagram of the STACK module is illustrated below in Figure 24 while a block diagram is illustrated below in Figure 25.

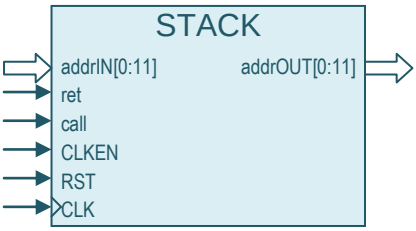


Figure 24 – Stack Interface Diagram

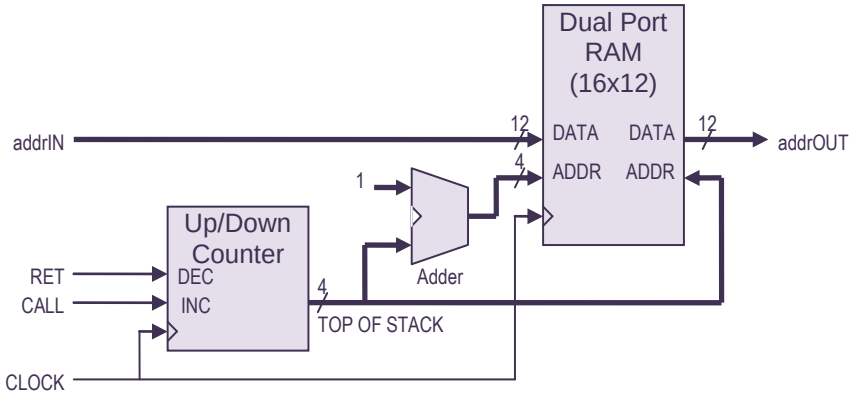


Figure 25 – Stack Block Diagram

When the ‘call’ input to the block is asserted, the 'call' address is stored, the stack pointer is incremented and the return address automatically becomes available at the new top-of-stack.

When the ‘ret’ (return) input to the block is asserted, the stack pointer is decremented. The return address is always available at the addrOUT[0:11] port.

This implementation saves all the KS10 logic to dynamically change RAM address depending if a 'call' or 'return' instruction is being processed. It also allows the stack to always update in a single clock cycle.

The ‘call’ and ‘return’ operation of the microcode is quite a bit different than modern computers. The microsequencer stores the address of the ‘call’ instruction on the stack. The ‘return’ instruction must include a dispatch offset to the address in order to return to an instruction after the ‘call’ instruction.

When a Page Fail exception occurs, the microcode vectors to the ‘Page Fail’ handler. When the Page Fail handler code has completed execution, the microsequencer returns to the microcode instruction (the read or write operation that caused the Page Fail exception) and re-executes that instruction.

2.2.3.9KS10 CPU DBUS

The DBUS module is essentially a 36-bit wide 4-to-1 multiplexer. The DBUS multiplexer selects between the FLAGS, the ALU Output (DP), the DBM Multiplexer, and the RAMFILE. On a read operation, the DBUS Multiplexer also selects between the RAMFILE (where the ACs are stored) if an AC is being read and memory if an AC is not being read.

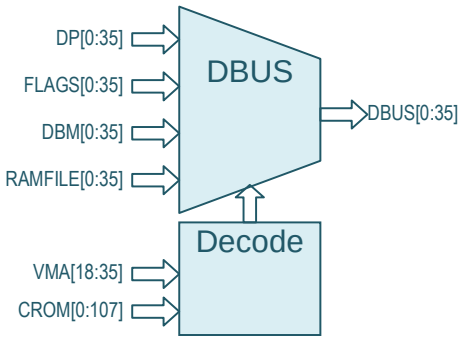


Figure 26 – DBUS Interface Diagram

When a memory request is made, the memory contents are supplied to this block via the DBM input. When a memory request is made to one of the AC registers, the forceRAMFILE bit is asserted and the contents of the RAMFILE is selected instead of the memory contents.

The Block Diagram of the DBUS Multiplexer is illustrated below in Figure 27

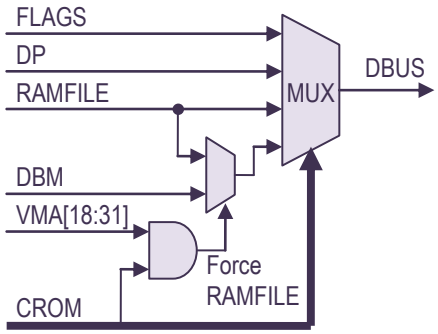


Figure 27 – DBUS Block Diagram

The forceRAMFILE signal is implemented very differently than the DEC KS10. In the KS10 FPGA implementation, the forceRAMFILE signal is asserted when the KS10 Bus input to the DBM Multiplexer is selected (i.e., during a memory read) and the VMA points to one of the ACs.

2.2.3.10 KS10 CPU Instruction Register (IR)

TBD.

2.2.3.11 KS10 CPU Step Count Adder (SCAD)

The Step Count Adder (SCAD) is a small 10-bit accumulator and register set that is used for loop counting and for floating-point exponentiation.

This block includes the Step Count (SC) register and the Floating-point Exponent (FE) register. The accumulator can be multiplexed to either register.

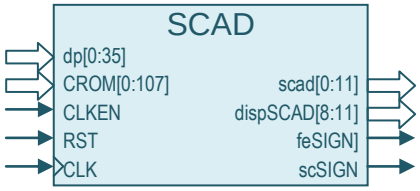


Figure 28 – SCAD Interface Diagram

The Step Count is used for generic loop control within the microcode. The Floating-point Exponent is used during floating point calculations.

The SCAD ALU can perform a load, add, subtract, bit-wise OR, increment, and decrement operations. A block diagram of the SCAD is illustrated below in Figure 29.

The SCAD can be ganged with the ALU to assist in multiplication and division.

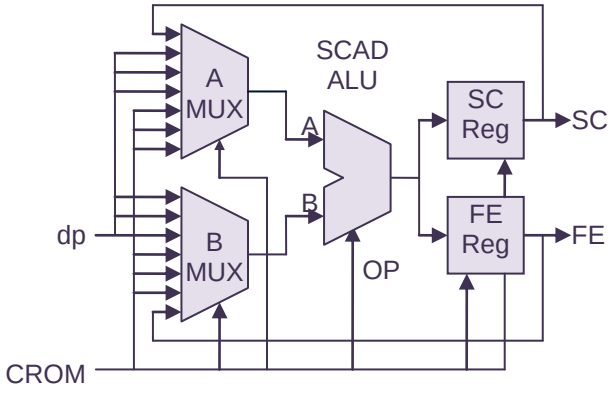


Figure 29 – SCAD Block Diagram

2.2.3.12 KS10 CPU RAMFILE

The RAMFILE contains storage for microcode which is essentially everything that is not stored in the ALU register set. The microcode does not use any of the KS10 memory.

The RAMFILE is a dedicated chunk of 1Kx36 memory. The RAMFILE is address as follows:

Table 3 – RAMFILE Addressing	
Address	**Description**
1777 1000	Cache (Not implemented)
0777- 0200	Workspace

Table 3 – RAMFILE Addressing	
Address	**Description**
0177-0160	AC Block 7
0157-0140	AC Block 6
0137-0120	AC Block 5
0117-0100	AC Block 4
0077-0060	AC Block 3
0057-0040	AC Block 2
0037-0020	AC Block 1
0017-0000	AC Block 0

In the original DEC KS10 implementation, this module was controlled only indirectly by the microcode. The relevant portions of the Control ROM microcode were multiplexed onto the DBM bus by the DBM multiplexer and the contents of the DBM bus were used to control the operation of the RAMFILE.

The KS10 FPGA is implementation different in that it is controlled directly by the Control ROM microcode. The RAMFILE control paths through the DBM multiplexer have been elided. This is faster and simpler than the alternative.

Presumably the DEC KS10 chose the original approach because of circuit board interconnection limitations.

2.2.3.13 KS10 CPU Pager (PAGER)

The KS10 CPU proper provides an 18-bit virtual address space for programs. The Pager provides an address translation mechanism that allows the KS10 CPU to access more than 256K words of memory.

The Pager adds two more bits of addressing which creates a 20-bit physical address from the 18-bit virtual address.

The Pager translates virtual addresses/page numbers to physical addresses/page numbers. There are 512 virtual pages which map to 2048 physical pages.

The address translation is illustrated below in Figure 30.

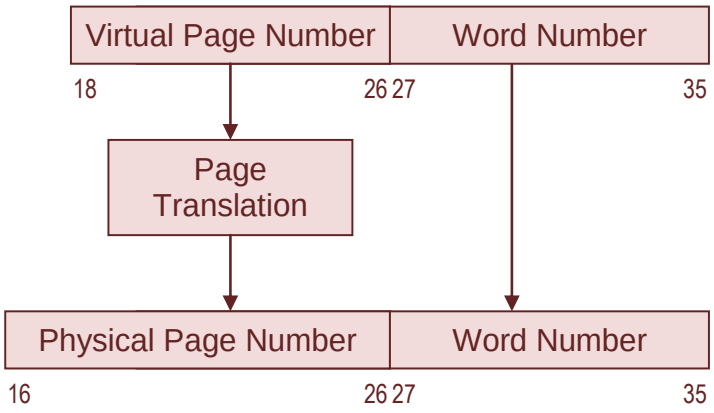


Figure 30 – Pager Address Translation

Physically, the pager sits between the CPU and the KS10 Backplane Bus on the address bus.

The DEC KS10 Page Translation Memory (Page Tables) were implemented using asynchronous memory which does not translate to an FPGA very efficiently. Since the Page Translation Memory is addressed by the VMA register and the VMA register is loaded synchronously, the FPGA implementation simply absorbs the VMA register into the Page Translation Memory addressing. In other words, when the VMA register is loaded, the VMA address is synchronously loaded into the Page Translation Memory.

The Page Table is interleaved with odd and even memories so that the memory can be swept two entries at a time.

The Page Table addressing is rearranged from that of the DEC KS10. On the DEC KS10 the two memories are interleaved by the MSB of the address. On the KS10 FPGA, the two memories are interleaved by the LSB of the address. When the interleaving is done this way, the Xilinx synthesis tool can infer a dual port memory with different aspect ratios on the two ports as follows:

- The Page Table is address by the write port as 256 x 30-bit memory. The 30-bit wide write port allows the page memory to be swept two entries at a time.
- The Page Table is address by the read port as 512 x 15-bit memory. The 15-bit wide read port allows for simple page lookups.

Referring to the block diagram below, the CPU writes data into the Page Table via the left port of dual port memory. The page translation data which consists of the virtual page number and the page flags is written to the dual port memory two entries at a time. This is consistent with the format of the User Page Tables and the Executive Page Tables. In general, the 'dp' (ALU) bus is the source of the data. The Page Table is also swept two entries at a time. This port is write-only.

On the right port of the dual port memory, the virtual address is used to address the Page Translation Memory. The output of this memory provides the Physical Page Number that is used to build the Physical Address. This port is read-only.

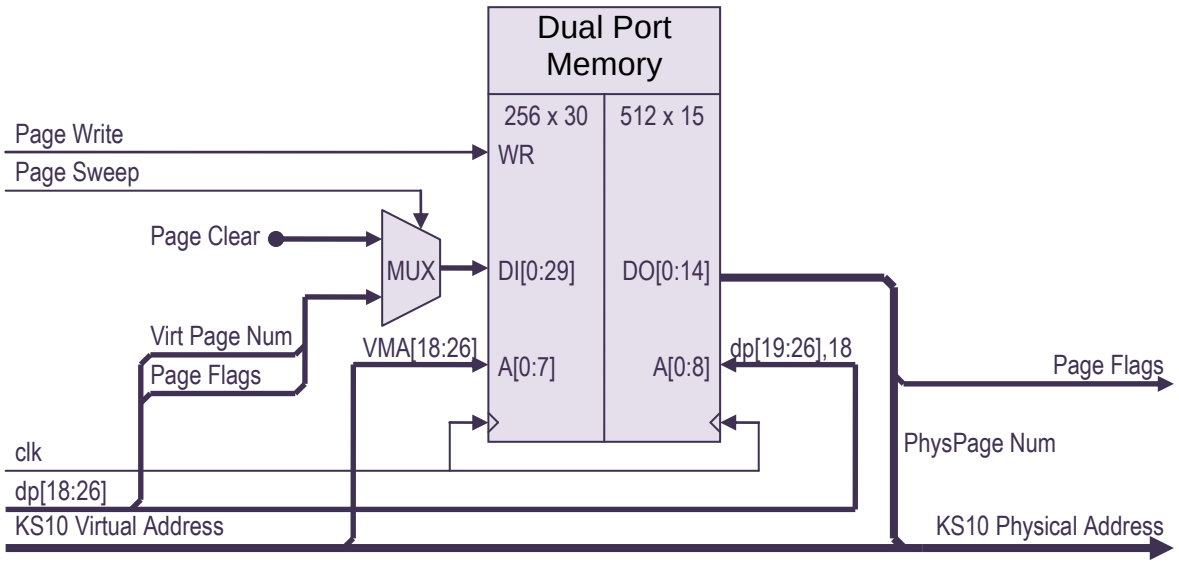


Figure 31 – Pager Block Diagram

The page flags are:

- Page Valid - this flag indicates that the page information has been initialized.
- Page Writable - this flag indicates that the page can be written.
- Page Cacheable – this flag indicates that the page is cacheable.
- Page User – this flag indicates that the page has only user-mode access privledges.

Note: All page flags are cleared when the Page Table is swept, although clearing the Page Valid flag is sufficient.

2.2.3.14 KS10 CPU Page Fault Dispatch (PF_DISP)

The title Page Fault Dispatch is a bit of a misnomer. The PAGE-FAIL entry point of the microcode (Address o3777), handles External (UBA) Interrupts, Timer Interrupts, APR Interrupts, NXM Interrupts, Uncorrectable Memory (BAD DATA) Interrupts (not implemented) and Page Failures.

Table 4 – "Page Fail" Dispatches

Dispatch (octal)	Description	Notes
00	No dispatch	
01	APR interrupt, external interrupt, timer interrupt	

03	Uncorrectable memory error	Not implemented
05	Non-existent memory error	
07	Combined Bad Data error and NXM error	Not implemented
10	Write to non-writeable page.	
11	Combined page not present and timer interrupt.	
12	Page not present	
13	EXEC / USER mismatch	

The page fault dispatch is negated when the microcode issues a MEMCLR operation.

2.2.3.15 KS10 CPU Next Instruction Dispatch (NI_DISP)

TBD.

2.2.3.16 KS10 CPU Previous Context (PXCT)

TBD.

2.2.4 KS10 Memory Controller (MEM)

TBD.

2.2.5 KS10 IO Bus Adapter (UBA)

TBD.

3 KS10 FPGA Backplane

The original DEC KS10 used a tri-state bus for the KS10 backplane. The KS10 FPGA uses a variant of this bus which is suitable for an FPGA implementation. It turns out that the operation this bus is wired into the KS10 microcode and significant design changes to this bus would require changes to the KS10 microcode.

The DEC KS10 backplane bus was implemented using a multiplexed address and data protocol. The address and control information was asserted onto the first of the bus cycles and the data was asserted onto the bus on a later bus cycle. This has been replaced by design with an independent address bus and data bus. This design change increases memory bandwidth with a slight increase in the amount of routing resources that the FPGA requires.

The KS10 FGPA backplane supports multiple *initiators* and multiple *targets*. The bus is arbitrated on a cycle-by-cycle basis. This simplifies the bus implementation.

Table 5 summarizes the operation of the Bus Arbiter.

Table 5 – Bus Arbiter Operations

Device	Initiator	Target	Priority
CPU	Yes	No	Lowest
Console	Yes	Yes	Middle
IO Bridge #1	Memory Only	Yes	Highest
IO Bridge #3	Memory Only	Yes	Highest
Memory Controller	No	Memory Only	N/A

Because most FPGAs don't support tri-state buses, the backplane is more of a logical notion. In actual implementation, the backplane is more of a multiplexer than a bus structure.

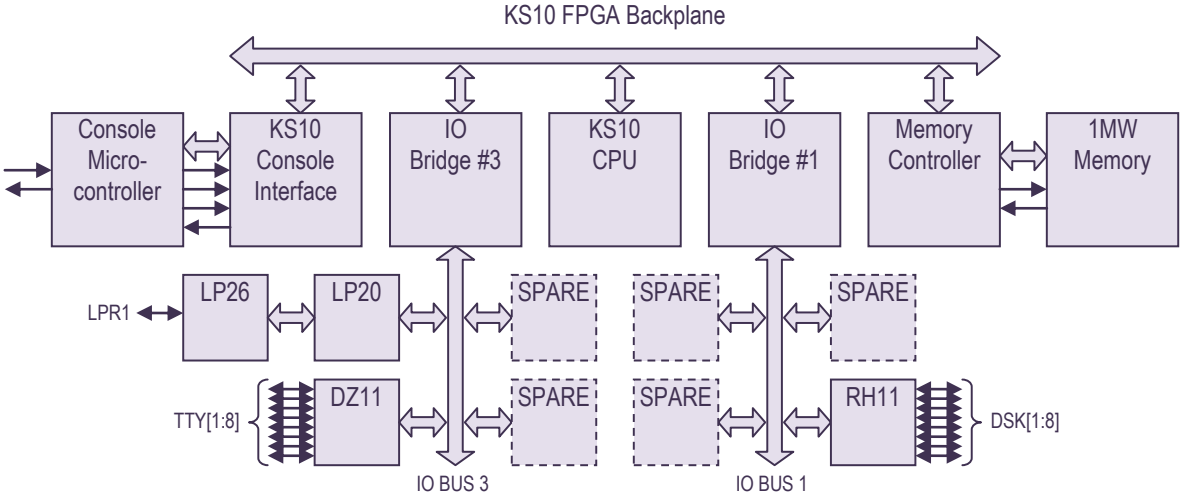


Figure 32 – KS10 FPGA Bus Architecture

3.1 KS10 FPGA Address Bus

The KS10 FPGA address bus contains control flags in the upper bits and address in the lower bits. Although the control flags aren't technically part of the address, they are often decoded with the address and this paradigm nicely simplifies the bus design.

The DEC KS10 uses a slightly different format for the bus control signals. The KS10 FPGA uses the standard VMA layout (see VMA Flags in the KS10 documentation) for the bus control signals and relies on the Verilog optimizer to remove the unused signals.

The address and control 'flags' are defined as follows:

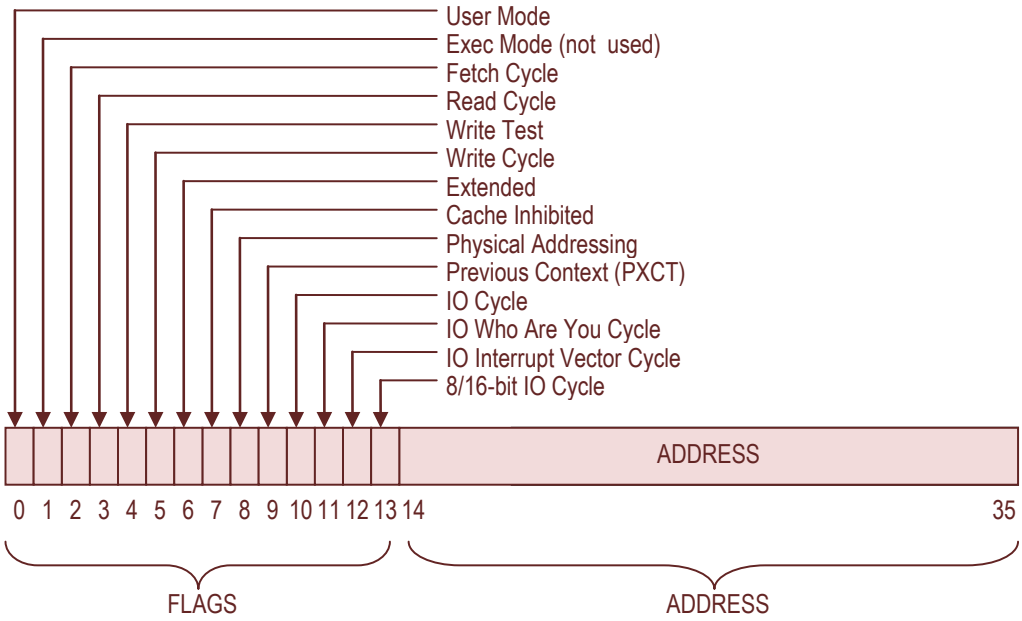


Figure 33 – KS10 FPGA Address Bus Illustration

Table 6 – Address Flag Definitions		
Bit	**Mnemonic**	**Description**
0	USER	User Mode. This signal probably isn't useful for anything outside of the CPU.
1	EXEC	Exec Mode. Not used. Always zero.
2	FETCH	Instruction Fetch.
3	READ	1 = Read Cycle (either IO or Memory)

Table 6 – Address Flag Definitions		
Bit	**Mnemonic**	**Description**
4	WRTEST	Write Test. When asserted, this will create a page fault on a virtual memory page that is not writeable. This signal will never be asserted independently of WRITE CYCLE. This signal probably isn't useful for anything outside of the CPU.
5	WRITE	Write Cycle (either IO or Memory)
6	EXTENDED	Extended Address Cycle. This signal probably isn't useful for anything outside of the CPU.
7	CACHEINH	Cache inhibit. This signal probably isn't useful for anything outside of the CPU or Cache Controller.
8	PHYSICAL	Physical Address.
9	PREVIOUS	VMA Previous Context (PXCT). This signal probably isn't useful for anything outside of the CPU.
10	IO	IO Cycle, When asserted, this indicates an IO Cycle; otherwise it is a Memory Cycle.
11	WRU	Read Device from UBA. When an interrupt is detected by the CPU, the CPU asserts the current interrupt priority onto the bits 15-17 and asserts the WRU CYCLE flag on the bus. The UBA that is asserting the interrupt request should respond with its identifier as follows: • UBA1 should assert bit 19. • UBA2 should assert bit 20. • UBA3 should assert bit 21. • UBA4 should assert bit 22. Not all UBA adapters were implemented in the DEC KS10 hardware. See note at the beginning of Section 7. If no device responds to this request, the microcode assumes that the APR has requested the interrupt.
12	VECTOR	Read Interrupt Vector from UBA. The CPU asserts a UBA device number onto bits 14-17. The addressed UBA should respond with the 36-bit interrupt vector associated with the device that is causing that interrupt.
13	IOBYTE	Unibus Byte IO operation. The LSB of the address is used to determine if the upper or lower byte is to be accessed.

3.2 KS10 FPGA Bus Cycles

Table 7 – Bus Cycles

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Memory Read	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	20-bit Memory Address																			
Memory Write	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	20-bit Memory Address																			
WRU Cycle	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	PI	Don't Care																			
Interrupt Vector Cycle	0	0	0	1	0	0	0	1	0	0	1	0	1	0	4-bit UBA		Don't Care																			
Word IO Read	0	0	0	1	0	0	0	0	1	0	1	0	0	0	4-bit UBA		18-bit IO Address																			
Word IO Write	0	0	0	0	0	1	0	0	1	0	1	0	0	0	4-bit UBA		18-bit IO Address																			
Byte IO Read	0	0	0	1	0	0	0	0	1	0	1	0	0	1	4-bit UBA		18-bit IO Address																		B	
Byte IO Write	0	0	0	0	0	1	0	0	1	0	1	0	0	1	4-bit UBA		18-bit IO Address																		B	

3.2.1 Memory Read and Write Cycles

Memory read cycles are always 36-bit bus cycles with a 20-bit address.

3.2.1.1 Memory Read Bus Cycle

Memory Read Cycles always assert READ (bit 3), negate WRITE (bit 5), and negate IO (bit 10).

If PHYSICAL (bit 8) is asserted, this address refers to a physical address (one of the ACs, for example) instead of a virtual (memory) address.

If FETCH (bit 2) is asserted, this is an instruction fetch.

Memory Read Bus Cycle

Control/Address Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Memory Read	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	20-bit Memory Address																			

3.2.1.2 Memory Write Bus Cycle

Memory Write Cycles always assert WRITE (bit 5), negate READ (bit 3), and negate IO (bit 10).

If PHYSICAL (bit 8) is asserted, this address refers to a physical address (one of the ACs, for example) instead of a virtual address (memory).

If WRTEST (bit 4) is also asserted, the PAGER will also test the address for a non-writable page access violation.

Memory Write Bus Cycle																																						
Control/Address Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Memory Write	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	20-bit Memory Address																			

3.2.2 IO Read and Write Bus Cycle

The KS10 FPGA IO Bus is implemented quite a bit different than the DEC KS10 Unibus. Whereas the DEC Unibus was limited to 16-bit registers and in some cases 18-bit NPR data (the RH11 is 18-bit), the IO Bus in the KS10 FPGA is more like an extension of the KS10 Backplane bus where the 36-bit address and a 36-bit data is arbitrated and bridged by the UBA. This greatly increases the bandwidth of the IO operations.

From a hardware point of view, IO Read operations are always 36-bit and can transfer 36-bit data, 16-bit data, or 8-bit data. The Memory Status Register (address o100000) is an example of a 36-bit IO register.

When a 16-bit or 8-bit Unibus register read, the device right justifies the 16-bit data (reads are always 16-bit) and pads the upper 20 data bits with zero. For 8-bit reads, the CPU microcode masks and shifts the 16-bit data as required to select the proper 8-bit byte.

Similarly, IO Write operations are always 36-bit and can transfer 36-bit data, 16-bit data, or 8-bit data. When a 16-bit or 8-bit Unibus register written, the device truncates the upper 20-bits of data and is left with 16-bit, right justified data. The IO BYTE bit and the LSB of the address can be used to select either or both byte-lanes for writing.

3.2.2.1Word IO Read Bus Cycle (36-bit or 16-bit)

Word IO Read Cycles always assert READ (bit 3), negate WRITE (bit 5), assert PHYSICAL (bit 8), assert IO (bit 10), and negate IO BYTE (bit 13). The hardware register size (and padding) determines whether 36-bit or 16-bit data is read.

Word IO Read Cycle																																						
Control/Address Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Word IO Read	0	0	0	1	0	0	0	0	1	0	1	0	0	0	4-bit UBA				18-bit IO Address																			

3.2.2.2Byte IO Read Bus Cycle

Word IO Read Cycles always assert READ (bit 3), negate WRITE (bit 5), assert PHYSICAL (bit 8), assert IO (bit 10), and assert IO BYTE (bit 13).

When B is negated (Address bit 35), the low byte is read.
When B is asserted (Address bit 35), the high byte is read.

Byte IO Read Cycle																																							
Control/Address Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35				
Byte IO Read	0	0	0	1	0	0	0	0	1	0	1	0	0	1	4-bit UBA				18-bit IO Address																				B		

3.2.2.3Word IO Write Bus Cycle (36-bit or 16-bit)

Word IO Write Cycles always negate READ (bit 3), assert WRITE (bit 5), assert PHYSICAL (bit 8), assert IO (bit 10), and negate IO BYTE (bit 13). The hardware register size determines whether 36-bit or 16-bit data is written.

Word IO Write Cycle																																							
Control/Address Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35				
Word IO Write	0	0	0	0	0	1	0	0	1	0	1	0	0	0	4-bit UBA				18-bit IO Address																						

3.2.2.4Byte IO Write Bus Cycle

Word IO Write Cycles always negate READ (bit 3), assert WRITE (bit 5), assert PHYSICAL (bit 8), assert IO (bit 10), and assert IO BYTE (bit 13).

When B is negated (Address bit 35), the low byte is written.
When B is asserted (Address bit 35), the high byte is written.

Byte IO Write Cycle																																							
Control/Address Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35				
Byte IO Write	0	0	0	0	0	1	0	0	1	0	1	0	0	1	4-bit UBA				18-bit IO Address																						B

3.2.3 Who Are You (WRU) Bus Cycle

The WRU Cycle is an IO READ bus cycle that to determine which UBA, if any, is currently requesting an interrupt.

WRU Cycles always assert READ (bit 3), negate WRITE (bit 5), assert PHYSICAL (bit 8), assert IO (bit 10), and assert WRU (bit 11). The address bits (ADDR[18:0]) are ignored and the requested PI Level is asserted onto PI bits (ADDR[15:17]).

WRU Bus Cycle																																							
Control/Address Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35				
WRU Cycle	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	PI			Don't Care																					

3.2.4 Interrupt Vector Bus Cycle

Vector Cycles always assert READ (bit 3), negate WRITE (bit 5), assert PHYSICAL (bit 8), assert IO (bit 10), and assert VECT (bit 12). The address bits (ADDR[18:0]) are ignored and the requested UBA number is asserted onto UBA # bits (ADDR[14:17]).

Interrupt Vector Cycle																																							
Control/Address Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35				
Interrupt Vector Cycle	0	0	0	1	0	0	0	1	0	0	1	0	1	0	UBA #				Don't Care																					

3.2.5 KS10 FPGA Interrupt Sequence

The interrupt sequence is as follows:

1. The IO Device asserts one of the Device Interrupt Request (devINTR[7:4]) signals.

2. The IO Bus Bridge "ORs" the Device Interrupt Request signals (devINTR[7:4]) from each of the IO Devices together. Next the IO Bus Bridge maps the Device Interrupt Request signals to one of the 7 CPU interrupt request priorities using the High Priority and Low Priority register settings in the IO Bus Bridge Control and Status Register. The CPU Interrupt signals are asserted on the busINTR[1:7] pins of the Backplane Bus.

   See the Priority Interrupt High (PIH) and the Priority Interrupt Low (PIL) bits of the IO Bus Bridge Control Status Register (UBACSR) for more information on how this interrupt priority mapping occurs.

3. The Backplane Bus Arbiter "ORs" together the External Priority Interrupt signals from each of the IO Bus Bridges and sends the result to the Priority Interrupt Controller inside CPU.

4. The Priority Interrupt Controller inside the CPU selects the highest priority interrupt and interrupts the CPU by asserting the "pageFAIL" signal. The CPU microcode eventually notices that the "pageFAIL" signal is asserted and dispatches to the interrupt processing microcode. There, the CPU issues a Who Are You (WRU) bus cycle with the interrupt priority that is being handled.

5. All the IO Bus Bridges receive the WRU bus cycle and each IO Bus Bridge compares the interrupt priority associated with the WRU with the interrupt priority it may be asserting. If the priority of the WRU bus cycle matches the priority of the IO Bus Bridge, the IO Bus Bridge responds with its WRU identifier response.

   a. It is possible that more than one IO Bus Bridge is asserting the same interrupt. If so, the KS10 Backplane Bus Arbiter selects the highest priority IO Bus Bridge's WRU response.
      Note: the lowest UBA number is the highest priority IO Bus Bridge.

b. It is also possible that more than one device attached to the IO Bus Bridge is asserting that same interrupt request. For the purpose of WRU bus cycle, this is irrelevant. This will be resolved later during the Interrupt Vector Cycle.

When this occurs, the microcode in the CPU remembers the UBA number associated with the interrupt. Note: the WRU identifiers are enumerated in the WIKI section above describing Bus Cycles.

6. The microcode in the CPU examines the identifier that is received, if any. If no identifier is received, the CPU handles the interrupt as an APR interrupt. If an identifier is received, the CPU performs an Interrupt Vector bus cycle which is addressed to the IO Bus Bridge that was associated with the WRU identifier that was arbitrated and received by the CPU.

7. The addressed IO Bus Bridge arbitrates the the Interrupt Vector bus cycle to the highest priority device that is asserting the highest priority interrupt; but note, the interrupt priority selection occurs before the device selection. The selected device responds with its associated Interrupt Vector and the IO Bridge forwards the Interrupt Vector back to the CPU.

   The device uses the Interrupt Vector bus cycle as an Interrupt Acknowledge indication.

   Note: as an implementation detail, the timing of the Interrupt Acknowledge with respect to the Interrupt Vector bus cycle is important:

   a. If the interrupt clears during the Interrupt Vector bus cycle, the interrupt vector will be misread by the microcode.

   b. If the interrupt clears much after the Interrupt Vector bus cycle is complete, the interrupt will be recognized twice.

   Therefore, it is recommend that IO devices acknowledge that the interrupt (clear the interrupt) immediately after the Interrupt Vector bus cycle.

8. The CPU software handles the interrupt as described by the KS10 Architecture.

3.2.6 APR / Timer Interrupt

The KS10 FPGA APR interrupt bus cycle is captured the DSKAH Diagnostic.

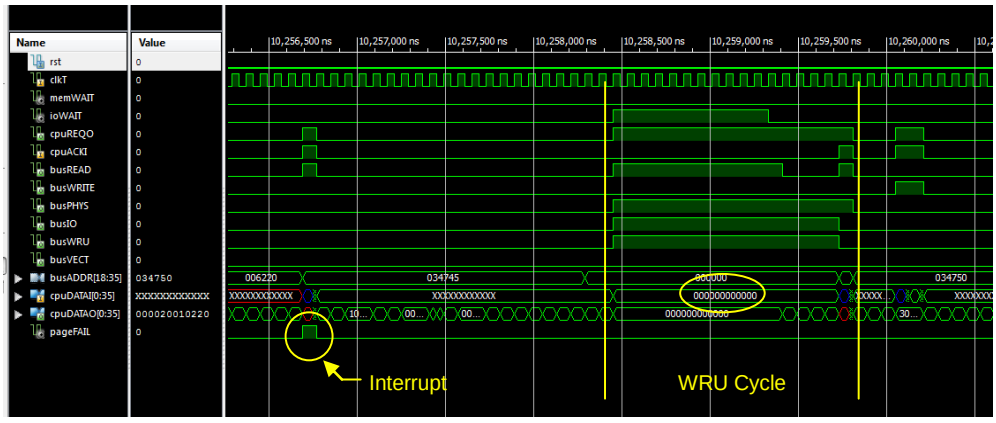


Figure 34 – APR Interrupt Bus Cycle

Note that none of the IO Bus Bridges respond to the WRU request and therefore the WRU response is “000000”. Also note that there no VECTOR cycle following the WRU request cycle.

3.2.7 KS10 FPGA External Interrupt

The KS10 FPGA external interrupt bus cycle captured from a DZ11 interrupt cycle is shown below in Figure 35.

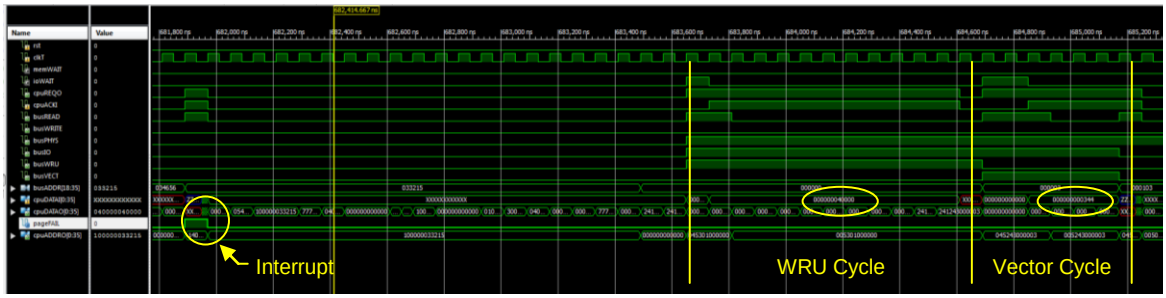


Figure 35 – External Interrupt Bus Cycle

Note that the IO Bus Bridge responds to the WRU request with “040000” which is the correct response for UBA3. Also note that the DZ11 responds to the VECTOR request with “000340” which is the correct interrupt vector for the DZ11 receiver.

4 Console Processor

Unlike modern computers, the KS10 processor can't actually bootstrap itself without support from the console microcontroller. Early PDP10 computers required the operator to key in the bootstrap program from the front panel interface using switches and lights. The DEC KS10 simplified the boot processes when it employed an Intel 8080 microprocessor and a board full of support circuitry to perform this function.

4.1 Booting the KS10 FPGA

Once the KS10 Console Processor negates the KS10 reset, The KS10 FPGA will begin to execute the microcode. This microcode will perform some initialization, perform a small Arithmetic Logic Unit (ALU) test, and enter the *halt* state.

When the KS10 FPGA enters this *halt* state, the Console will print the `KS10>` prompt.

At this point, it is necessary to interact with the console to select the boot device.

Whereas the DEC KS10 can boot from either a disk drive or a magtape attached to a Unibus Adapter, the KS10 FPGA can boot from the RP06 disk drive or any file that is accessible to the Linux system. Magtapes are not supported, yet. I have a Kennedy Magtape drive with a Pertec Interface: how hard could that be to support?

To that end many of the old DEC KS10 Commands have been deleted and some new commands have been added.

Once the boot device has been selected and the boot code has been loaded into the KS10 memory, the KS10 FPGA must be started with the Console "BT" command.

4.2 KS10 FPGA Console Commands

The KS10 FPGA Console Program implements a subset of the original KS10 Console Commands. These commands are summarized in the following sections.

Table 8 – KS10 Console Command Summary

Command	Argument	Description
BC		Boot Check. Not implemented. Check the KS10 boot path.
BT		Boot Monitor This command boots to the Monitor from the selected disk drive.
BT	1	Boot Diagnostic Monitor This command boots to the Diagnostic Monitor the from the selected disk drive.
CE	{1 | 0}	Cache Enable Enable/Disable KS10 cache.

Table 8 – KS10 Console Command Summary		
Command	**Argument**	**Description**
CH		Clock Halt Not implemented. Halt the CPU clock.
CO		Continue Continue KS10 program execution. The KS10 will exit the **HALT** state and begin program execution. The console program enters user mode.
CP	arg	Clock Pulse. Not implemented. Clock the CPU clock xx times.
CS		Clock Start Not implemented. Start the CPU clock.
DB	arg	Deposit Bus Not implemented. Deposit $(arg) onto KS10 bus. This is a console loopback selftest command.
DC	arg	Deposit CRAM Not implemented. Deposit $(arg) into CRAM memory at address previously set by EC or LC command.
DF	arg	Deposit into CRAM memory. Not implemented. Deposit $(arg) into CRAM address at address and diagnostic function previous loaded by LC and LF command.
DK	arg	Deposit into 8080 memory. Not implemented. Deposit $(arg) into 8080 memory at address previously set by LK command.
DI	arg	Deposit into IO register Deposit $(arg) into KS10 IO at address previously set by an LI or EI command. (IO can be 8-bit, 16-bit, 18-bit or 36-bit)
DM	arg	Deposit KS10 Memory. Deposit $(arg) into KS10 memory at address previously set by LA command.
DN	arg	Deposit Next Deposit $(arg) into next KS10 memory address at address previously set by LA command.

Table 8 – KS10 Console Command Summary		
Command	**Argument**	**Description**
DR		Deposit into 8080 Register. Not implemented. Deposit $(arg) into 8080 IO at address previously set by LR command.
DS		Show UBA address, RH11 Base Address, and RP06 Unit Number
DS	Param(s)	DS {UBA=n} {BASE=nnnnnnn} {UNIT=n} Set RH11 Boot Parameters The default is DS UBA=1 BASE=776700 UNIT=0 UBA must be 1. BASE must be 776700. All others are invalid. UNIT must be 0-7
EB		Examine KS10 Bus Not implemented. Prints contents of console registers. This is a console loopback selftest command.
EC		Examine CRAM register Not implemented Examine CRAM at address $(arg)
EI		Examine I/O register. Examine KS10 IO at address previously set by an LI command.
EJ		Examine CRAM address. Not implemented. Examine current CRAM address, next CRAM address, jump address, and subroutine return address.
EK		Examine 8080 memory. Not implemented Examine 8080 memory at address previously set by an LK command.
EK	arg	Examine 8080 memory. Not implemented Examine 8080 memory at address $(arg)
EM		Examine KS10 memory. Examine KS10 Memory at address previously set by an LA command.
EM	arg	Examine KS10 memory. Examine KS10 Memory at address $(arg)

Table 8 – KS10 Console Command Summary		
Command	**Argument**	**Description**
EN		Examine Next Examine contents of next KS10 Memory or I/O address. Address previously loaded by LA or LI command.
ER		Examine 8080 Register Not implemented Examine 8080 IO at address previously set by an LR command.
ER	arg	Examine 8080 Register Not implemented Examine 8080 IO at address $(arg)
EX		Execute Execute single KS10 instruction. The KS10 will exit the **HALT** state, execute a single instruction, and return to the **HALT** state.
HA		Halt Halt the KS10. The KS10 will remain in the **HALT** state until it is commanded to continue (see CO command), is single-stepped (see SI command), or it commanded to execute a single instruction (see EX command).
KL	{1 | 0}	KLINIK mode Not implemented. Enable/Disable KLINIK
LA	arg	Load Memory Address. Set KS10 FPGA memory address to $(arg). The valid address range is 0000000-3777777.
LB		Load Bootstrap to Monitor. This command boots to the Monitor the from the selected disk drive. This is the same command as BT since the microcode is not loadable.
LB	1	Load Bootstrap to Diagnostic Monitor. This command boots to the Diagnostic Monitor from the selected disk drive. This is the same command as BT 1 since the microcode is not loadable.
LC	arg	Load CRAM Address. Not implemented. Set CRAM address to $(arg)
LF	arg	Load Diagnostic Write Function. Not implemented. Set CRAM word select to $(arg)

<table>
<tr><th colspan="4">Table 8 – KS10 Console Command Summary</th></tr>
<tr><th>Command</th><th>Argument</th><th colspan="2">Description</th></tr>
<tr><td rowspan="21">LI</td><td rowspan="21">arg</td><td colspan="2">Set KS10 IO address to $(arg)
The IO address consists of a device number (0-17 octal) and a register address (0-777777 octal).
Only devices 0-4 are implemented, therefore the valid IO address range is 0000000-4777777.</td></tr>
<tr><td>IO Address</td><td>Description</td></tr>
<tr><td>0100000</td><td>Memory Status Register</td></tr>
<tr><td>0200000</td><td>Console Instruction Register</td></tr>
<tr><td>1763000-1763077</td><td>UBA1 Paging RAM</td></tr>
<tr><td>1763100</td><td>UBA1 Status Register</td></tr>
<tr><td>1763101</td><td>UBA1 Maintenance Register</td></tr>
<tr><td>1700000-1777777</td><td>UBA1 Device Registers</td></tr>
<tr><td>2763000-2763077</td><td>UBA2 Paging RAM (not implemented)</td></tr>
<tr><td>2763100</td><td>UBA2 Status Register (not implemented)</td></tr>
<tr><td>2763101</td><td>UBA2 Maintenance Register (not implemented)</td></tr>
<tr><td>2700000-2777777</td><td>UBA2 Device Registers (not implemented)</td></tr>
<tr><td>3763000-3763077</td><td>UBA3 Paging RAM</td></tr>
<tr><td>3763100</td><td>UBA3 Status Register</td></tr>
<tr><td>3763101</td><td>UBA3 Maintenance Register</td></tr>
<tr><td>3700000-3777777</td><td>UBA3 Device Registers</td></tr>
<tr><td>4763000-4763077</td><td>UBA4 Paging RAM (not implemented)</td></tr>
<tr><td>4763100</td><td>UBA4 Status Register (not implemented)</td></tr>
<tr><td>4763101</td><td>UBA4 Maintenance Register (not implemented)</td></tr>
<tr><td>4700000-4777777</td><td>UBA4 Device Registers (not implemented)</td></tr>
<tr><td>LK</td><td>arg</td><td colspan="2">Set 8080 Memory Address
Not implemented.
Set 8080 memory address to $(arg)</td></tr>
<tr><td>LR</td><td>arg</td><td colspan="2">Set 8080 IO address.
Not implemented.
Set 8080 IO address to $(arg)</td></tr>
</table>

Table 8 – KS10 Console Command Summary		
Command	**Argument**	**Description**
LT		Lamp Test Not implemented. Blink the indicator lights.
MB		Magtape Boot Not implemented. Load the monitor boot program from the tape selected last
MK	arg	Mark microcode word Not implemented Mark microcode word at CRAM address $(arg)
MM		Set KLINK state to APT Not implemented
MR		Master Reset. Momentarily reset the KS10.
MR	{on|off}	Master Reset. Set the KS10 reset state.
MS		Show Magtape Boot Parameters Not implemented. Show UBA address, RH11 Base Address, TU45 Unit Number, Slave Number, and Density
MS	arg	Set Magtape Boot Parameters Not implemented. Set UBA address, RH11 Base Address, TU45 Unit Number, Slave Number, and Density
MT		Boot to Monitor from Magtape Not implemented. This command boots to the Monitor from the selected Magtape drive.
MT	1	Boot to Diagnostics from Magtape Not implemented. This command boots to the Diagnostic Monitor from the selected Magtape drive.
PE	{1 | 0}	Parity Enable. Not implemented.
PM		Pulse Microcode Not implemented. Pulse Microcode. Issue CP and EJ

Table 8 – KS10 Console Command Summary		
Command	**Argument**	**Description**
PW		KLINIK Password Not implemented. Sets or clears KLINIK password
RC		Repeat command Not implemented. Repeat last command or last command string until any CTY key is depressed
RP		Repeat Not implemented. Repeat last command or last command string until any CTY key is depressed.
RP	arg	Repeat Not implemented. Repeat last command or last command string, $(arg) times.
SC	{1 | 0}	Soft CRAM Errors Not implemented. Enable/Disable soft CRAM errors
SD	DIR	Access Secure Digital Card Show directory of Secure Digital Card (new command)
SH		Shutdown Deposits nonzero data into KS10 memory location 30 to allow orderly shutdown of the monitor.
SI		Single Step. Executes next KS10 instruction.
SM	arg	Start Microcode Not implemented. Start microcode at address $(arg)
ST	arg	Start KS10 program at address $(arg). Console program enters user mode.
TE	{1 | 0}	Enable/Disable KS10 CPU Interval Timer
TP	{1 | 0}	Enable/Disable KS10 CPU traps.
TT		Tranfer KLINIK Not implemented. Tranfer KLINIK line to user mode
UM	arg	Unmark microcode word Not implemented. Unmark microcode word at CRAM at address $(arg)

Table 8 – KS10 Console Command Summary		
Command	**Argument**	**Description**
VD		Verify CRAM against disk. Not implemented.
VT		Verify CRAM against tape. Not implemented.
ZM		Zero Memory Deposit 0 into all KS10 memory locations.

.

5 KS10 FPGA Console Interface

5.1 KS10 FPGA Console Interface Registers

The KS10 FPGA implements four 36-bit registers and two 64-bit registers that are used to exchange information between the Console Microcontroller and the KS10 FPGA.

The Console Address register and the Console Data register are a pair of 36-bit registers that is used by the console to read and write to KS10 memory across the KS10 bus. The KS10 bus transaction is controlled by a state machine. When the transaction is completed, the state machine will update the Status Register with the results of the transaction.

The Console Instruction register is used by the console to execute a single instruction.

The FPGA Version register is a read-only register that contains a version identifier set by the FPGA firmware.

The RH11 Debug Register is a read-only register that contains status information from the RH11.

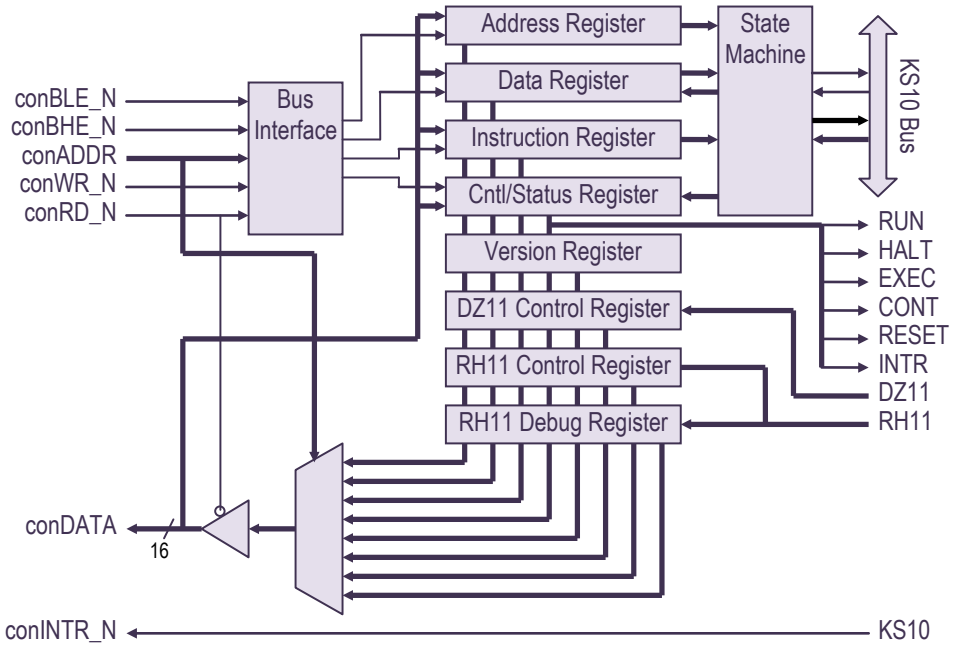


Figure 36 – KS10 Console Interface Block Diagram

5.1.1 Console Microcontroller Interface

The Console Microcontroller and KS10 are interconnected by a simple non-multiplexed 16-bit bi-directional data bus.

The TI/Stellaris microcontroller address mapping is perhaps a little confusing: the External Peripheral Interface (EPI) **A0** pin is really the microcontroller **A1** signal. The **A0** signal from the microcontroller is not available for use. Instead, the microcontroller provides two “byte lane” signals: Bus Low Enable

(**conBLE_N**) and Bus High Enable (**conBHE_N**). The **conBLE_N** signal is asserted (low) when data should be written to the FPGA from the low 8-bits of the 16-bit data bus and **conBHE_N** is asserted (low) when data should be written to the FPGA from the high 8-bits of the 16-bit data bus. In this configuration, it can be assumed that the microcontroller **A0** signal is always zero and therefore it is not necessary for the microcontroller to supply the signal. The byte lanes are not decoded for read access to the FPGA: the microcontroller simply ignores any bus signals that are not relevant to the access being performed.

The read (**conRD_N**) and write (**conWR_N**) signals control FPGA access.

Although the processor status can be obtained by polling the Console Status Register, the (**haltLED**) signal is provided to indicate when the processor is halted. This is mostly useful in the simulation environment.

Lastly, an interrupt signal (**conINTR_N**) is required to implement the KS10 FPGA/Console CTY Interface protocol.

This interface is illustrated below in Figure 37.

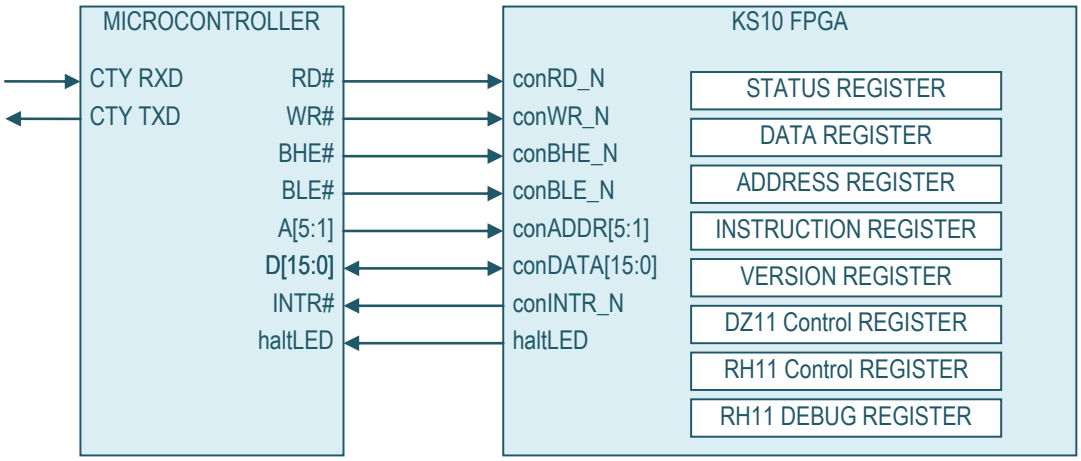


Figure 37 – Console Microcontroller and KS10 FPGA Interface

5.1.2 Console Interface Bus Design

The console microcontroller bus interface is asynchronous and must be synchronized to the KS10 clock where necessary.

The console read operation is kept asynchronous. The read hardware consists of an address decoder, a data bus multiplexer, and a bidirectional bus interface.

The console write operation is synchronized to the KS10 bus clock – this includes the conWR_N, the conBLE_N, and the conBHE_N signals.

A timing diagram of a console bus read/write operation is illustrated below in Figure 38.

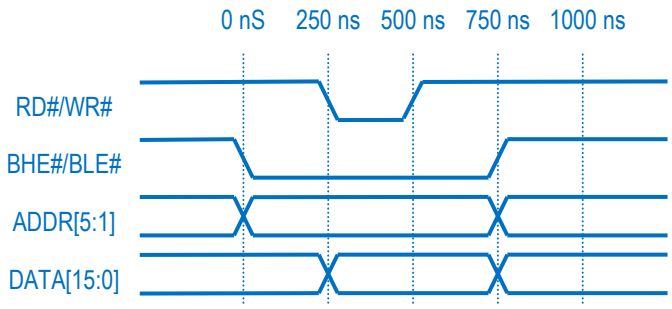


Figure 38 – Console Microcontroller Interface Read/Write Cycle Timing Diagram

5.1.3 Console Interface Register Memory Map

The Console Interface Registers are memory mapped from the point-of-view of the Console Microcontroller and are otherwise not visible to the KS10. These registers are little-endian.

A 64-bit aligned address space is reserved for each of the 36-bit register and each 36-bit register is right justified in that 64-bit address space.

The Console Interface Memory Map is summarized below in Table 9.

Table 9 – Console Interface Register Memory Map	
Address	**Description**
0x00	Console Address Register bits 28-35
0x01	Console Address Register bits 20-27
0x02	Console Address Register bits 12-19
0x03	Console Address Register bits 4-11
0x04	Console Address Register bits 0-3
0x05	Reserved
0x06	Reserved
0x07	Reserved
0x08	Console Data Register bits 28-35
0x09	Console Data Register bits 20-27
0x0a	Console Data Register bits 12-19
0x0b	Console Data Register bits 4-11
0x0c	Console Data Register bits 0-3
0x0d	Reserved
0x0e	Reserved
0x0f	Reserved
0x10	Console Instruction Register bits 28-35
0x11	Console Instruction Register bits 20-27
0x12	Console Instruction Register bits 12-19
0x13	Console Instruction Register bits 4-11
0x14	Console Instruction Register bits 0-3
0x15	Reserved
0x16	Reserved
0x17	Reserved
0x18	Console Control/Status Register bits 24-31
0x19	Console Control/Status Register bits 16-23
0x1a	Console Control/Status Register bits 8-15
0x1b	Console Control/Status Register bits 0-7

0x1c	DZ11 Control Register bits 24-31
0x1d	DZ11 Control Register bits 16-23
0x1e	DZ11 Control Register bits 8-15
0x1f	DZ11 Control Register bits 0-7
0x20	LP20 Control Register bits 24-31
0x21	LP20 Control Register bits 16-23
0x22	LP20 Control Register bits 8-15
0x23	LP20 Control Register bits 0-7
0x24	RPXX Control Register bits 24-31
0x25	RPXX Control Register bits 16-23
0x26	RPXX Control Register bits 8-15
0x27	RPXX Control Register bits 0-7
0x28	DUP11 Control Register bits 24-31
0x29	DUP11 Control Register bits 16-23
0x2a	DUP11 Control Register bits 8-15
0x2b	DUP11 Control Register bits 0-7
0x2c	Reserved
0x2d	Reserved
0x2e	Reserved
0x2f	Reserved
0x30	Reserved
0x31	Reserved
0x32	Reserved
0x33	Reserved
0x34	Reserved
0x35	Reserved
0x36	Reserved
0x37	Reserved
0x38	Reserved
0x39	Reserved
0x3a	Reserved
0x3b	Reserved

0x3c	Debug Control/Status Register bits 24-31
0x3d	Debug Control/Status Register bits 16-23
0x3e	Debug Control/Status Register bits 8-15
0x3f	Debug Control/Status Register bits 0-7
0x40	Debug Breakpoint Address Register bits 28-35
0x41	Debug Breakpoint Address Register bits 20-27
0x42	Debug Breakpoint Address Register bits 12-19
0x43	Debug Breakpoint Address Register bits 4-11
0x44	Debug Breakpoint Address Register bits 0-3
0x45	Reserved
0x46	Reserved
0x47	Reserved
0x48	Debug Breakpoint Mask Register bits 28-35
0x49	Debug Breakpoint Mask Register bits 20-27
0x4a	Debug Breakpoint Mask Register bits 12-19
0x4b	Debug Breakpoint Mask Register bits 4-11
0x4c	Debug Breakpoint Mask Register bits 0-3
0x4d	Reserved
0x4e	Reserved
0x4f	Reserved
0x50	Debug Instruction Trace Register bits 56-63
0x51	Debug Instruction Trace Register bits 48-55
0x52	Debug Instruction Trace Register bits 40-47
0x53	Debug Instruction Trace Register bits 32-39
0x54	Debug Instruction Trace Register bits 24-31
0x55	Debug Instruction Trace Register bits 16-23
0x56	Debug Instruction Trace Register bits 8-15
0x57	Debug Instruction Trace Register bits 0-7
0x58	Debug PC and IR bits 56-63
0x59	Debug PC and IR bits 48-55
0x5a	Debug PC and IR bits 40-47
0x5b	Debug PC and IR bits 32-39

0x5c	Debug PC and IR bits 24-31
0x5d	Debug PC and IR bits 16-23
0x5e	Debug PC and IR bits 8-15
0x5f	Debug PC and IR bits 0-7
0x60	Reserved
0x61	Reserved
0x62	Reserved
0x63	Reserved
0x64	Reserved
0x65	Reserved
0x66	Reserved
0x67	Reserved
0x68	Reserved
0x69	Reserved
0x6a	Reserved
0x6b	Reserved
0x6c	Reserved
0x6d	Reserved
0x6e	Reserved
0x6f	Reserved
0x70	RH11 Debug Register bits 56-63
0x71	RH11 Debug Register bits 48-55
0x72	RH11 Debug Register bits 40-47
0x73	RH11 Debug Register bits 32-39
0x74	RH11 Debug Register bits 24-31
0x75	RH11 Debug Register bits 16-23
0x76	RH11 Debug Register bits 8-15
0x77	RH11 Debug Register bits 0-7
0x78	Firmware Version Byte 0
0x79	Firmware Version Byte 1
0x7a	Firmware Version Byte 2
0x7b	Firmware Version Byte 3

0x7c	Firmware Version Byte 4
0x7d	Firmware Version Byte 5
0x7e	Firmware Version Byte 6
0x7f	Firmware Version Byte 7

The operation of these registers is enumerated in the following sections

5.1.4 Console Control/Status Register

The Console Control/Status Register is used to control the KS10 FPGA and to obtain status from the KS10 FPGA.

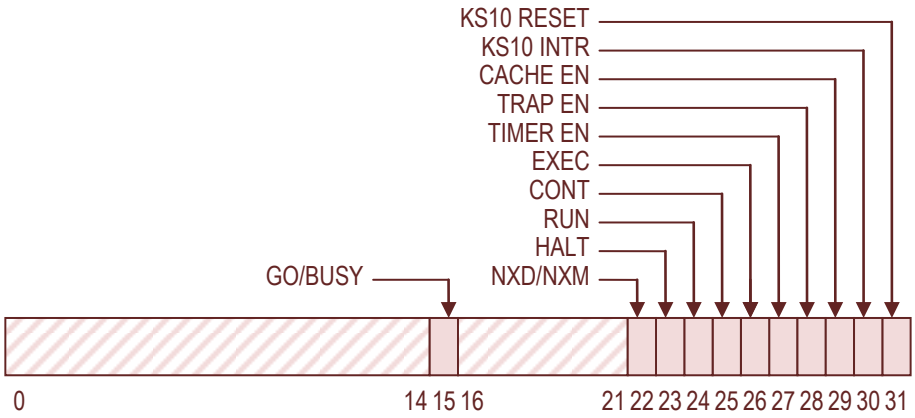


Figure 39 – Console Control/Status Register

The Console Control/Status Register bits are defined as follows:

Table 10 – Console Control/Status Register Definitions

Bit	Mnemonic	R/W	Init	Description
0-14	Reserved	R	-	Reserved. Writes ignored. Always read as zero.

Table 10 – Console Control/Status Register Definitions				
Bit	**Mnemonic**	**R/W**	**Init**	**Description**
15	GO/BUSY	R/W	0	GO. When asserted, this bit starts a state-machine that performs a memory or IO transaction. This bit remains asserted until the memory or IO transaction is completed. When the bus transaction has completed, this bit will be negated automatically. Note: The Console Address Register and Console Data Register should not be modified when this bit is asserted.
16-21	Reserved	R	-	Reserved. Writes ignored. Always read as zero.
22	NXM/NXD	R/W	0	Non-existent Memory or Non-existent Device. This bit is set if the last console-initiated bus transaction is not acknowledged by a memory or IO device. This bit is reset by writing a zero.
23	HALT	R	-	HALT Status. Writes ignored. Returns HALT status.
24	RUN	R/W	0	KS10 Run. When the RUN bit is asserted, the KS10 will begin execution. When the RUN bit is negated, the KS10 will stop execution. The RUN bit is also cleared automatically by the KS10 microcode when a HALT condition occurs. See Section 5.2.1 for a description of the operation of this bit

Table 10 – Console Control/Status Register Definitions				
Bit	**Mnemonic**	**R/W**	**Init**	**Description**
25	CONT	W	0	KS10 Continue. When the CONT bit is asserted, the KS10 will execute the next instruction and then return to the *HALT* state. The CONT bit is automatically cleared by the KS10 microcode once it is sampled. Always read as zero. Note: See Section 5.2.3 for a detailed description of the operation of this bit.
26	EXEC	W	0	KS10 Execute. When the EXEC bit is asserted and the KS10 exits the *HALT* state by either writing to the RUN (CSR[24]) or CONT (CSR[24]) bits, the KS10 will execute the instruction in the Console Instruction Register (CIR) before anything else. The EXEC bit is automatically cleared by the KS10 microcode once it is sampled. Always read as zero. Note: See Section 5.2.3 for a detailed description of the operation of this bit.
27	TIMER EN	R/W	0	Timer Enable This bit enables the KS10 one millisecond interval timer.
28	TRAP EN	R/W	0	Trap Enable This bit enables KS10 traps.
29	CACHE EN	R/W	0	Cache Enable This bit enables the KS10 cache.
30	KS10_INTR	W	0	This bit generates an interrupt from the Console to the KS10. Writing a ‘1’ generate the interrupt from the Console to the KS10; otherwise no interrupt is generated. This bit is always read as zero.

Table 10 – Console Control/Status Register Definitions				
Bit	**Mnemonic**	**R/W**	**Init**	**Description**
31	KS10_ RESET	R/W	1	When this bit is asserted, the KS10 is held in reset. This bit is asserted (and the KS10 is held in reset) at power-up. Otherwise this bit reflects the last value written to this register bit. Note: The console microcontroller must initialize the Console Instruction Register and the Console CTY interface (at least) before negating the KS10 RESET signal. This does not reset any of the peripherals.

5.1.5 Console Data Register

The Console Data Register is a 36-bit register that provides the data during a console-initiated memory or IO write transaction. The Console Data Register also receives the data that that is read during a console-initiated memory or IO read transaction.

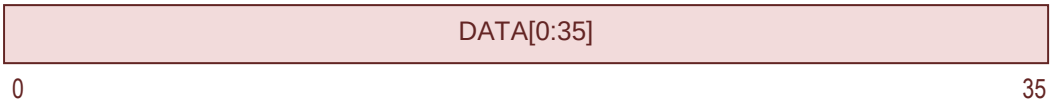


Figure 40 – Console Data Register

5.1.6 Console Address Register

The address that is used by the KS10 FPGA backplane bus is supplied by the Console Address Register which is detailed below in Figure 41.

Figure 41 – Console Address Register

The address register definition is identical to the backplane bus definition that is described in Section 3.1, i.e., the bits of the Console Address Register are directly applied, 1:1, to the address bus of the backplane.

5.1.7 Console Instruction Register

The Console Instruction Register is a 36-bit IO register located at IO Address o200000. After the KS10 Microcode initializes the KS10 micro-machine it fetches the contents of the Console Instruction Register and executes that instruction.

Normally this is a JRST instruction that jumps to the starting address of the boot loader or diagnostic code although the implementation places no constraints on the instruction that placed in this register. A JRST Instruction is opcode o254.

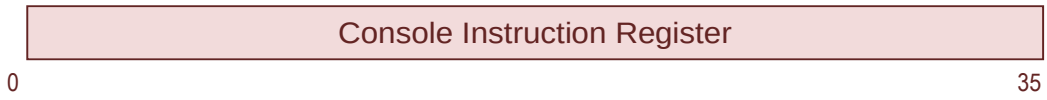


Figure 42 – Console Instruction Register

5.1.8 DZ11 Console Control Register (DZCCR)

The DZ11 Modem Control lines are not available external to the KS10 FPGA Board. The DZ11 Console Control Register is used to configure the DZ11 Modem Status register. It also controls the DZ11 loopback.

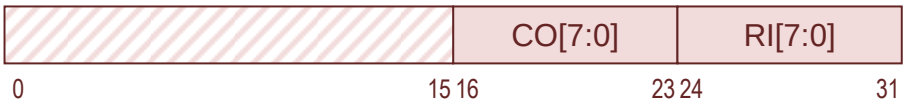


Figure 43 – DZ11 Console Control Register (DZCCR)

Table 11 – DZ11 Console Control Register (DZCCR) Definition			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0-15	Reserved	R/W	Reserved Unused bits are read/write
16-23	CO[7:0]	R/W	Carrier Sense These bits are reflected in the DZ11 Modem Status Register bits 15 through 8.
24-31	RI[7:0]	R/W	Ring Indication These bits are reflected in the DZ11 Modem Status Register bits 7 through 0.

5.1.9 LP20 Console Control Register (LPCCR)

The LP20 status is controlled by the Line Printer Console Control Register.

It provides the selection between Programmable Digital and Fixed Optical Vertical Format Units. It also provides a means to put the printer online or take it offline.

Lastly, it provides a means to set the serial port parameters of the printer interface.

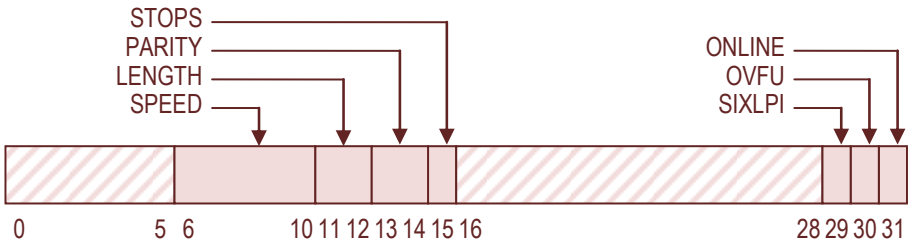


Figure 44 – LP20 - Console Control Register (LPCCR)

Table 12 – LP20 Console Control Register (LPCCR) Definition				
Bit(s)	**Mnemonic**	**R/W**	**Description**	
0-5	Reserved	-	Writes ignored. Always read as zero	
6-10	SPEED	R/W	**Baud Rate**	
			0-13	50 – 7200 baud. Not recommended. Probably will cause printer timeouts.
			14	9600 baud
			15	19200 baud
			16	38400 baud
			17	57600 baud
			18	115200 baud
			19	230400 baud
			20	460800 baud
			21	921600 baud
11-12	LENGTH	R/W	**Number of Bits**	
			0	5-bits (invalid)
			1	6-bits (invalid)
			2	7-bits
			3	8-bits

Table 12 – LP20 Console Control Register (LPCCR) Definition				
Bit(s)	**Mnemonic**	**R/W**	**Description**	
13-14	PARITY	R/W	**Parity**	
			0	No parity
			1	Odd parity
			2	Even parity
			3	Do not use
15	STOPS	R/W	**Number of Stop Bits**	
			0	1 stop bit
			1	2 stop bits
16-28	Reserved	-	Writes ignored. Always read as zero	
29	SIXLPI	R	Line Spacing When asserted, this indicates that the printer has been commanded to 6 LPI mode. Otherwise it is in 8 LPI mode. The printer defaults to 6 LPI.	
30	OVFU	R/W	Optical Vertical Format Unit (OVFU) When asserted, this sets the Vertical Format Unit type. The Vertical Format unit can either be “Optical” or “Direct Access”. See LP20 CSRB[OVFU].	
31	ONLINE	R/W	Printer On Line Writing a ‘1’ will set the printer on-line Writing a ‘0’ will set the printer off-line This bit will be read as ‘1’ if the printer is on-line; otherwise it will be read as ‘0’. Note: Various error conditions in the LP26 printer will set the printer off-line.	

5.1.10 DUP11 Console Control Register (DPCCR)

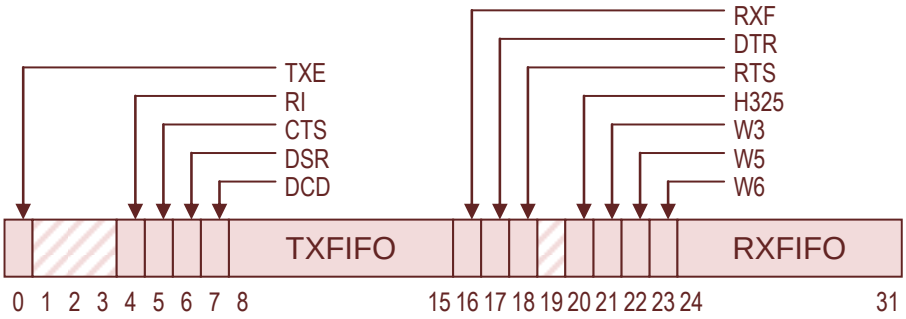


Figure 45 – DUP11 Console Control Register (DPCCR)

Table 13 – DUP11 Console Control Register (DPCCR) Definition			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0	TXE	R	Transmitter FIFO Empty This bit is asserted when the Transmitter FIFO is empty. Writes ignored.
1-3	-	R	Write ignored. Always read as zero.
4	RI	R/W	Ring Indication
5	CTS	R/W	Clear to Send
6	DSR	R/W	Data Set Ready
7	DCD	R/W	Data Carrier Detect
8-15	TXFIFO	R	Transmitter FIFO The DUP11 transmits data into this FIFO. The Console reads data from this FIFO. Writes ignored.
16	RXF	R	Receiver FIFO Full This bit is asserted when the Receiver FIFO is full. Writes ignored.
17	DTR	R	Data Terminal Ready
18	RTS	R	Request to Send
19	-	R	Writes ignored. Always read as zero.
20	H325	R/W	Install H325 Loopback

Table 13 – DUP11 Console Control Register (DPCCR) Definition			
Bit(s)	**Mnemonic**	**R/W**	**Description**
21	W3	R/W	Configuration Jumper W3 0: Jumper not installed 1: Jumper installed. See description of W3 in the description of the DUP11 Receiver Control/Status Register (RXCSR). This configuration jumper is normally installed.
22	W5	R/W	Configuration Jumper W5 0: Jumper not installed 1: Jumper installed See description of W5 in the description of the DUP11 Receiver Control/Status Register (RXCSR). This configuration jumper is normally not installed.
23	W6	R/W	Configuration Jumper W6 0: Jumper not installed 1: Jumper installed See description of W6 in the description of the DUP11 Receiver Control/Status Register (RXCSR). This configuration jumper is normally installed.
24-31	RXFIFO	W	Receiver FIFO The DUP11 reads data from this FIFO. The Console writes data to this FIFO. Always read as zero.

5.1.11 RPXX Console Control Register (RPCCR)

The RPXX Console Control Register (RPCCR) controls the status of the various RP06 disk drives.

A real DEC RP06 disk drive provides controls are not emulated in the FPGA because of the SDHC media. In this implementation, the RPXX Console Control Register provides the following equivalent functionality. These controls are:

1. Disk unit present or absent, or
2. Disk unit online or offline, or
3. Disk unit write protected or write enabled.

The RPXX Console Control Register is defined below.

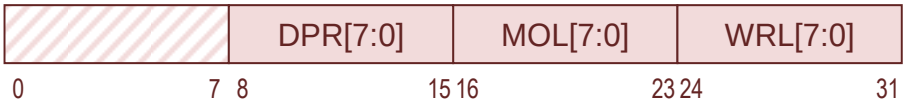


Figure 46 – RPXX Console Control Register (RPCCR)

Table 14 – RPXX Console Control Register (RPCCR) Definition			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0-7	Reserved	R/W	Reserved Unused bits are read/write.
8-15	DPR[7:0]	R/W	Drive Present This bit is reflected in RPDS[DPR]. When a drive is not present, a read of any of the registers of that device will return 0. Writes will succeed. In both reads and write, a non-existent device error (RPDS[NED]) will be asserted.
16-23	MOL[7:0]	R/W	Media On Line This bit is reflected in RPDS[MOL]. When MOL transitions to active: 1. RPDS[ATA] is asserted, and 2. RPDS[VV] is negated. When MOL transitions to inactive, RPDS[ATA] is asserted. Note: RPDS[VV] is not altered by negating MOL.
24-31	WRL[7:0]	R/W	Write Lock This bit is reflected in RPDS[WRL].

5.1.12 RH11 Debug Register

The RH11 Debug Register is a register that is not present on a DEC KS10. It is present only to facilitate debugging the KS10 FPGA RH11 Disk Controller. It also has Disk Drive blinking lights.

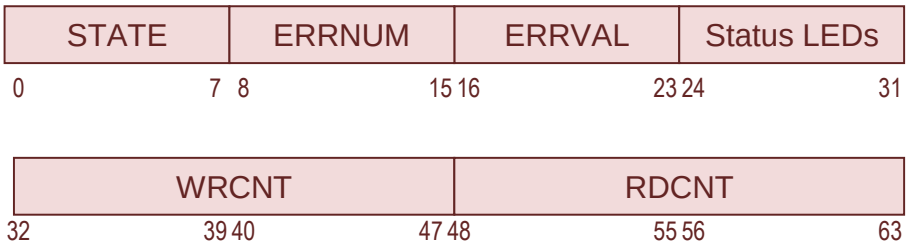


Figure 47 – RH11 Debug Register

Table 15 – RH11 Debug Register Definitions			
Bit	**Mnemonic**	**R/W**	**Description**
0-7	STATE	R	SDHC Card State Machine State
8-15	ERRNUM	R	Error Number
16-23	ERRVAL	R	Error Value
24	DISK0	R	Disk Unit 0 Activity LED
25	DISK1	R	Disk Unit 1 Activity LED
26	DISK2	R	Disk Unit 2 Activity LED
27	DISK3	R	Disk Unit 3 Activity LED
28	DISK4	R	Disk Unit 4 Activity LED
29	DISK5	R	Disk Unit 5 Activity LED
30	DISK6	R	Disk Unit 6 Activity LED
31	DISK7	R	Disk Unit 7 Activity LED
32-47	WRCNT	R	Number of Disk Writes.
48-63	RDCNT	R	Number of Disk Reads.

5.1.13 Debug Interface

The KS10 FPGA contains debug hardware that is not present on the DEC KS10. This includes a Hardware Breakpoint Facility and an Instruction Trace Buffer. These are described in the following sections.

5.1.13.1 Debug Control/Status Register (DCSR)

The Debug Control/Status Register controls the operation of the Debug Interface.

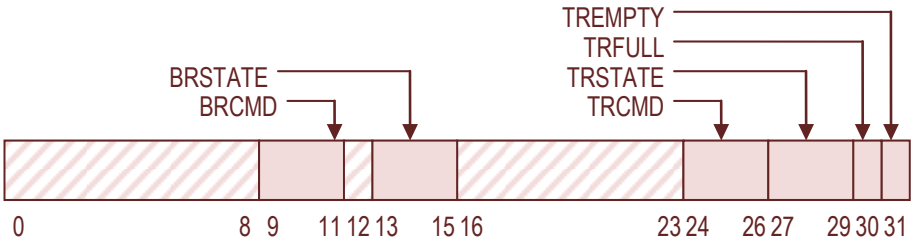


Figure 48 – Debug Control/Status Register (DCSR)

Table 16 – Debug Control/Status Register (DCSR) Definitions					
Bit	**Mnemonic**	**R/W**	**Init**	**Description**	
0-8	Reserved	R/W	-	Reserved Unused bits are read/write.	
9-11	BRCMD	R/W	0	Breakpoint Command Read/Write	
				0	Trigger Disable
				1	Trigger on Address Match
				2	Trigger on Trace Buffer Full
				3	Trigger on Address Match or Trace Buffer Full
				4-7	Reserved
12	Reserved	R/W	-	Reserved Unused bits are read/write.	
13-15	BRSTATE	R	0	Breakpoint State Writes ignored.	
				0	Breakpoint Idle
				1	Breakpoint 2(waiting for trigger)
				2-7	Reserved
16-23	-	R	0	Reserved Writes ignored. Always read as zero.	

Table 16 – Debug Control/Status Register (DCSR) Definitions					
Bit	**Mnemonic**	**R/W**	**Init**	**Description**	
24-26	TRCMD	R/W	0	Trace Command Read/Write	
				0	Reset Command This command will cause the Trace System to: 1. immediately stop trace acquisition (if it is acquiring trace data), and 2. disarm the address match, and 3. flush the trace buffer . (This sets the Trace State back to Idle)
				1	Trigger Command This command will cause the Trace System to trigger immediately and begin acquiring trace data.
				2	Trigger on Address Match Command This command arms the trace match logic.
				3	Stop Command This command will stop trace acquisition.
				4-7	Reserved
27-29	TRSTATE	R	0	Trace State Writes ignored.	
				0	Trace Idle
				1	Trace Armed In this state, the Debug Unit is waiting for and Address Match.
				2	Trace Active In this state, the Debug Unit is acquiring instruction trace data.
				3	Trace Done In this state, the Debug Unit has acquired instruction trace data.
				4-7	Reserved
30	TRFULL	R	-	Trace Buffer Full Writes ignored. This signal is asserted when the Trace Buffer is full.	

Table 16 – Debug Control/Status Register (DCSR) Definitions				
Bit	**Mnemonic**	**R/W**	**Init**	**Description**
31	TREMPTY	R	1	Trace Buffer Empty Writes ignored. This signal is asserted when the Trace Buffer is empty.

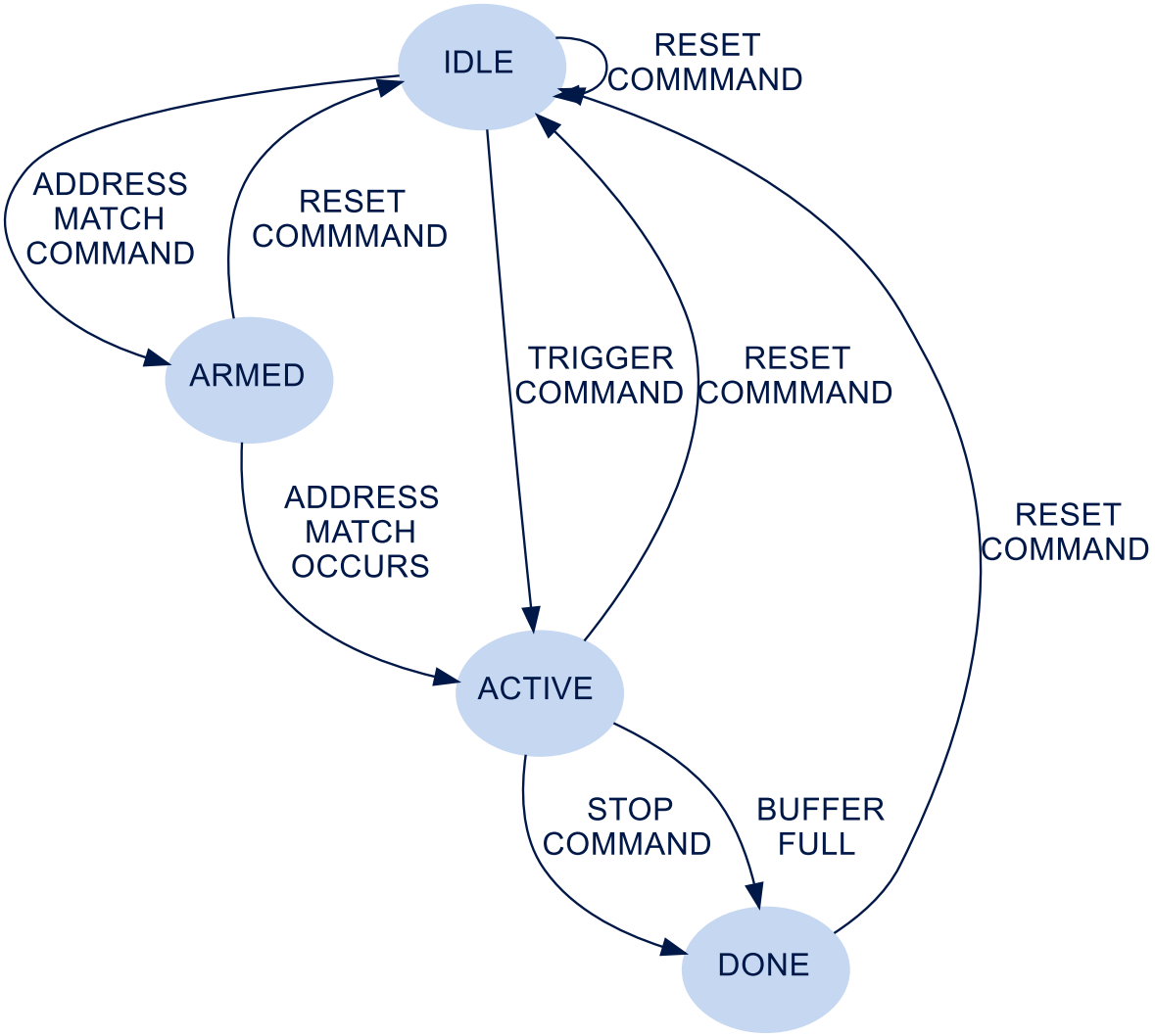


Figure 49 - Instruction Trace State Diagram

5.1.13.2 Debug Breakpoint Address Register (DBAR)

The Breakpoint Facility monitors the address on the KS10 Backplane Bus as described in Section 3.1 and halts the KS10 CPU when the “Address Match” conditions are met.

The Breakpoint Address Register is a 36-bit register that sets the breakpoint address. The Breakpoint Mask Register is another 36-bit register that controls which bits of the Breakpoint Address are used in the breakpoint comparison and which bits are ignored.

It should be noted that the Breakpoint Register examines the KS10 Backplane and therefore operates on Physical Addresses and not on Virtual Addresses. Also, the breakpoint facility will only breakpoint on addresses generated by the CPU. It will not breakpoint on DMA operations.

It goes without saying that there are a lot of bus-cycles that could be detected with the breakpoint system that are not really useful.

Figure 50 – Debug Breakpoint Address Register (DBAR)

<table>
<tr><th colspan="4">Table 17 – Debug Breakpoint Address Register (DBAR) Definitions</th></tr>
<tr><th>Bit</th><th>Mnemonic</th><th>R/W</th><th>Description</th></tr>
<tr><td rowspan="15">0-13</td><td rowspan="15">Flags</td><td rowspan="15">R/W</td><td>Flags.<table><tr><th>BIT</th><th>Description</th></tr><tr><td>0</td><td>User Mode</td></tr><tr><td>1</td><td>Not used</td></tr><tr><td>2</td><td>Fetch Cycle</td></tr><tr><td>3</td><td>Read Cycle</td></tr><tr><td>4</td><td>Write Test Cycle</td></tr><tr><td>5</td><td>Write Cycle</td></tr><tr><td>6</td><td>Extended</td></tr><tr><td>7</td><td>Cache Enabled</td></tr><tr><td>8</td><td>Physical Address</td></tr><tr><td>9</td><td>PCXT</td></tr><tr><td>10</td><td>IO Cycle</td></tr><tr><td>11</td><td>IO Who Are You Cycle</td></tr><tr><td>12</td><td>IO Interrupt Vector Cycle</td></tr><tr><td>13</td><td>IO Byte Cycle</td></tr></table></td></tr>
<tr><td>14-35</td><td>Address</td><td>R/W</td><td>Address</td></tr>
</table>

5.1.13.3 Debug Breakpoint Mask Register (DBMR)

The Breakpoint Mask Register provides a mechanism to include or ignore the various address and flag bits that are associated with the Backplane. When a bit is asserted in the Breakpoint Mask Register, the associated bit in the Breakpoint Address Register will be included in the breakpoint comparison. When the bit is negated in the Breakpoint Mask Register, the bit will be ignored in the breakpoint comparison.

Figure 51 – Breakpoint Mask Register (DBMR)

<table>
<tr><th colspan="4">Table 18 – Debug Breakpoint Mask Register Definitions (DBMR)</th></tr>
<tr><th>Bit</th><th>Mnemonic</th><th>R/W</th><th colspan="2">Description</th></tr>
<tr><td rowspan="15">0-13</td><td rowspan="15">Flags</td><td rowspan="15">R/W</td><td colspan="2">Flags.</td></tr>
<tr><th>BIT</th><th>Description</th></tr>
<tr><td>0</td><td>User Mode</td></tr>
<tr><td>1</td><td>Not used</td></tr>
<tr><td>2</td><td>Fetch Cycle</td></tr>
<tr><td>3</td><td>Read Cycle</td></tr>
<tr><td>4</td><td>Write Test Cycle</td></tr>
<tr><td>5</td><td>Write Cycle</td></tr>
<tr><td>6</td><td>Extended</td></tr>
<tr><td>7</td><td>Cache Enabled</td></tr>
<tr><td>8</td><td>Physical Address</td></tr>
<tr><td>9</td><td>PCXT</td></tr>
<tr><td>10</td><td>IO Cycle</td></tr>
<tr><td>11</td><td>IO Who Are You Cycle</td></tr>
<tr><td>12</td><td>IO Interrupt Vector Cycle</td></tr>
<tr><td>13</td><td>IO Byte Cycle</td></tr>
<tr><td>14-35</td><td>Address</td><td>R/W</td><td colspan="2">Address</td></tr>
</table>

5.1.13.4 Debug Instruction Trace Register (DITR)

The Instruction Trace Register allows certain CPU registers to be stored as program execution occurs. When the trace facility is triggered, the Program Counter (PC) and the Instruction Register (IR) is stored in the Trace Buffer whenever an instruction is executed.

The Trace Buffer is a simple FIFO. When the most significant word (bits 0-15) of the Instruction Trace Register is read, the buffer state is updated.

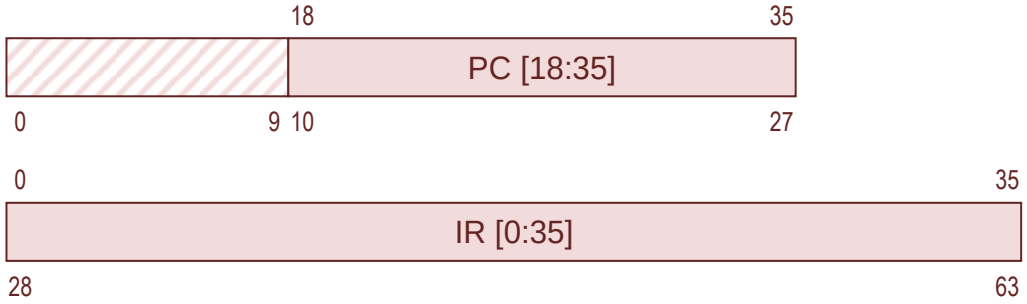


Figure 52 – Debug Instruction Trace Register

Table 19 – Debug Instruction Trace Register Definitions			
Bit	**Mnemonic**	**R/W**	**Description**
0-9	-	R	Reserved Always read as zero. Writes ignored.
10-27	PC[18:35]	R	Program Counter This value is only valid if this register is read when the Trace Buffer is not empty. If the Trace Buffer is not empty, the field contains the captured value of the Program Counter. Writes ignored.
28-63	IR[0:35]	R	Instruction Register This value is only valid if this register is read when the Trace Buffer is not empty. If the Trace Buffer is not empty, the field contains the captured value of the Instruction Register. Writes ignored.

5.1.13.5 Debug Program Counter and Instruction Register (DPCIR)

The Debug Program Counter and Instruction Register (DPCIR) allows the console microcontroller to take a snapshot of the program counter and instruction register.

This register has exactly the same format as the Debug Instruction Trace Register (DITR) detailed above.

5.1.14 Firmware Version Register

The Firmware Version Register is used for basic diagnostics and to allow the console to print the firmware revision of the FPGA.

One of the first tests that the console will perform is to attempt to read the contents of the Firmware Version Register from the FPGA. If the result is not consistent with the expected results, the console will print an error message and not attempt to boot the KS10 processor.

This test verifies that the FPGA is programmed and that there is a working bus connection between the console microcontroller and the FPGA.

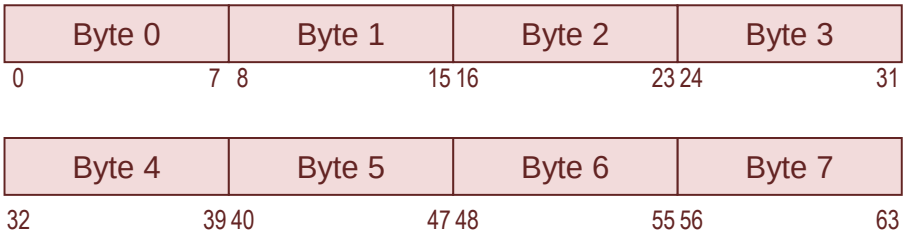


Figure 53 – Firmware Version Register

Table 20 – Console Firmware Version Register Definitions				
Byte	**Value**	**ASCII**	**R/W**	**Description**
0	0x52	'R'	R	Always 'R'
1	0x45	'E'	R	Always 'E'
2	0x56	'V'	R	Always 'V'
3	0x30	'0'	R	Major Revision MS Byte
4	0x30	'0'	R	Major Revision LS Byte
5	0x2e	'.'	R	Always '.'
6	0x30	'0'	R	Minor Revision MS Byte
7	0x34	'9'	R	Minor Revision LS Byte

Note: The combination of the values 0x52, 0x45, 0x56, and 0x2e (bytes 0, 1, 2, and 5) verifies that each of the 8 bits of the bus are asserted and negated during the test.

5.2 Controlling the KS10

This section describes the signals that are generated by the Console Microcontroller that control the operation of the KS10.

The **RUN** bit, the **CONT** bit, and the **EXEC** bit control the KS10 operation as illustrated below in Figure 54 below.

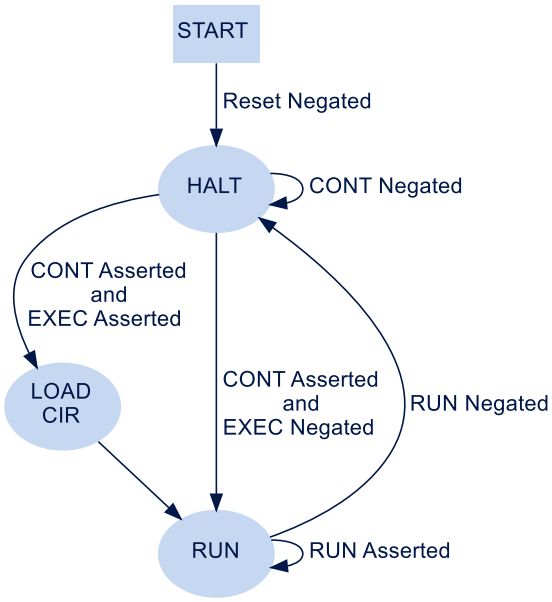


Figure 54 – KS10 Control State Diagram

When in the *HALT State*, Table 21 enumerates the types of operations that the KS10 will perform when the control bits are set in the various states.

Table 21 – Control Operation from Halt State			
CONT	**EXEC**	**RUN**	**Operation**
0	x	x	Remain in HALT State
1	0	0	Single Step instruction at current PC
1	0	1	Continue execution at current PC
1	1	0	Execute single instruction in CIR
1	1	1	Begin execution with instruction in the CIR

The operation of these bits is detailed in the following sections.

5.2.1 The RUN bit

The **RUN** bit controls whether the KS10 is in the *RUN State* or *HALT State*.

Setting the **RUN** bit will allow the KS10 to execute one or more instructions. The **RUN** bit is examined by the microcode at the end of each instruction.

When the **RUN** bit is asserted, the KS10 will execute the next instruction.

When the **RUN** bit is cleared, the KS10 will finish the current instruction and enter the *Halt State*.

5.2.2 The CONT bit

When the KS10 is in the *HALT State*, setting the **CONT** bit will cause the KS10 to exit the *HALT State* and execute at least one instruction. At the end of that instruction, the **RUN** bit is examined.

If the **RUN** bit is asserted, the KS10 will continue to execute instructions.

If the **RUN** bit is negated, the KS10 will re-enter the *HALT State* - essentially single-stepping the processor.

5.2.3 The EXEC bit

When the KS10 is in the *HALT State*, setting the **CONT** bit and **EXEC** bit will cause the KS10 to exit the *HALT State* and execute the instruction in the Console Instruction Register (CIR). At the end of that instruction, the **RUN** bit is examined.

If the **RUN** bit is asserted, the KS10 will continue to execute instructions. This technique is used by the Console at startup to cause the KS10 to begin executing the boot loader or diagnostic program at a specific address. In this case, a JRST instruction which performs a jump to the starting address of the boot loader is placed into the Console Instruction Register.

If the **RUN** bit is negated, the KS10 will re-enter the *HALT State* - essentially executing the single instruction in the Console Instruction Register (CIR).

5.3 Console Interface Protocol

The following sections describe the protocol that the Console Microcontroller should use to control the KS10 processor and should use to access KS10 memory and IO.

The Console Microcontroller will steal bus cycles from the KS10 CPU, if necessary, to perform its operations. See the description of the Bus Arbiter in Section 2.2 of this document.

The procedure that the Console Microcontroller should use to access devices across the KS10 FPGA backplane is:

1. Before modifying the Console Address Register or the Console Data Register, the Console Microcontroller should verify that “GO/BUSY” bit of the Console Control/Status Register is negated.

   See description of GO/BUSY in Table 10.

2. If the operation is a write operation, the Console Microcontroller should put the data to be written in the Console Data Register.

3. The Console Microcontroller should set the Console Address Register per Table 32.

4. Once the Console Address Register is configured, the Console Microcontroller should assert the “GO/BUSY” bit of the Console Control/Status Register.

5. The Console Microcontroller should wait until the “GO/BUSY” bit of the Console Control/Status Register is negated.

6. If the operation is a read operation, the Console Microcontroller should read the Console Data Register.

7. The Console Microcontroller should read the Console Control/Status Register NXM/NXD bit, print a message, and clear the NXM/NXD bit.

Table 6 – Console Address Register Settings

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Memory Read	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	20-bit Memory Address																			
Memory Write	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	20-bit Memory Address																			
36-bit IO Read	0	0	0	1	0	0	0	0	0	0	1	0	0	0	22-bit IO Address																					
36-bit IO Write	0	0	0	0	0	1	0	0	0	0	1	0	0	0	22-bit IO Address																					
18/16-bit IO Read	0	0	0	1	0	0	0	0	0	0	1	0	0	0	22-bit IO Address (bit 35 must be zero)																					0
18/16-bit IO Write	0	0	0	0	0	1	0	0	0	0	1	0	0	0	22-bit IO Address (bit 35 must be zero)																					0
8-bit IO Read	0	0	0	1	0	0	0	0	0	0	1	0	0	1	22-bit IO Address (asserting bit 35 reads the high byte)																					B
8-bit IO Write	0	0	0	0	0	1	0	0	0	0	1	0	0	1	22-bit IO Address (asserting bit 35 writes the high byte)																					B

5.4 The Communications Area

The Communications Area is a region of KS10 memory that is accessible by both the KS10 and the Console Processor. This memory is used to communicate between the two devices. The addressing of the memory area is summarized below in Table 22.

Table 22 – KS10/Console Communications Area

Address	Summary	Detailed Description
000030	Halt Switch **[FE_SWITCH]**	See section 5.4.1
000031	Keep Alive **[FE_KEEPA]**	See section 5.4.2
000032	CTY Input Word	See section 5.4.3
000033	CTY Output Word	See section 5.4.3
000034	KLINIK Input Word	See section 5.4.4
000035	KLINIK Output Word	See section 5.4.4
000036	RH11 Address **[FE_RHBASE]**	See section 5.4.5
000037	Unit Number **[FE_UNIT]**	See section 5.4.6
000040	Magtape Parameters **[FE_MTFMT]**	See section 5.4.7

5.4.1 Halt Switch

The KS10 stuffs the base address of the RH11 disk controller in this memory location to force a reboot of the KS10.

This register is initialized to zero at startup. The operating system may be stopped by writing a non-zero value to this register.

Table 23 – KS10 Halt Switch Word (KS10 Memory Address 000030)		
Bit(s)	**Mnemonic**	**Description**

5.4.2 Keep Alive

Table 24 – 8080 Status Word (KS10 Memory Address 000031)		
Bit(s)	**Mnemonic**	**Description**
4	KSRLD	Reload request.
5	KPACT	Keep alive active. The console will reload the KS10 if the KPALIV field does not change.
6	KLACT	KLINIK active.
7	PAREN	Memory parity error detect enabled.
8	CRMPAR	CROM parity error detect enabled.
9	DRMPAR	DROM parity error detect enabled.
10	CASHEN	Cache enabled.
11	MILSEN	1 millisecond timer enabled.
12	TRPENA	Traps enabled.
13	MFGMOD	Maintenance mode.
14-19	-	Reserved
20-27	KPALIV	Keep alive word.
28		
29		
30		

Table 24 – 8080 Status Word (KS10 Memory Address 000031)		
Bit(s)	**Mnemonic**	**Description**
31	-	Reserved
32	AUTOBT	Boot switch or power-up.
33	PWRFAL	Power fail restart (start at 70)
34	FORREL	Forced reload.
35	KEPFAL	Keep-alive failure. (XCT exec 71)

5.4.3 Console TTY (CTY) Protocol

The KS10 Processor can perform IO directly to/from the Console TTY (CTY). The KS10 processor communicates with the Console using KS10 memory buffers and a pair of interrupts.

The following sections describe this protocol.

5.4.3.1Console TTY (CTY) Input Protocol

This section describes how CTY input characters are transferred from the Console to the KS10 processor.

The CTY input protocol uses KS10 memory location 000032 to transfer the CTY character and a 1-bit flag (Valid) from the Console Processor to the KS10. The bit-definition of KS10 memory location 000032 is illustrated below.

Figure 55 – KS10 CTY Input Word (KS10 Memory Address 000032)

Table 25 – KS10 CTY Input Word (KS10 Memory Address 000032)		
Bit(s)	**Mnemonic**	**Description**
0-26	Reserved	Ignored for reads and writes
27	VALID	Asserted by Console when character is available for the KS10 to read. Cleared by the KS10 after the character has been read.
28-35	CTY Character	ASCII Character. Note: SIMH masks the MSB to zero so the character is always 7-bits.

The procedure from transferring a character from the Console to the KS10 is a follows:

1. The Console verifies that there is no character already in the buffer to the KS10 by checking the VALID bit of location 000032. If the **VALID** bit it is still set, the console processor should wait until later.
2. The Console places a character with the **VALID** bit set in memory location 0000032.
3. The Console Processor Interrupts the KS10 by setting the **KS10_INTR** bit in the Console Control/Status Register. See Table 10.

The CTY Input Word should be initialized to zero by the Console Processor before starting the KS10.

5.4.3.2Console TTY (CTY) Output Protocol

This section describes how the CTY output characters are transferred from the KS10 processor to the Console.

The protocol uses KS10 memory location 000033 to transfer the CTY character and a 1-bit flag (Valid) from the KS10 to the Console Processor. The bit-definition of KS10 memory location 000033 is illustrated below.

Figure 56 – KS10 CTY Output Word (KS10 Memory Address 000033)

Table 26 – KS10 CTY Output Word (KS10 Memory Address 000032)		
Bit(s)	**Mnemonic**	**Description**
0-26	Reserved	Ignored for reads and writes
27	VALID	Asserted by KS10 when character is available for the Console to read. Cleared by the Console after the character has been read.
28-35	CTY Character	ASCII Character. Note: SIMH masks the MSB to zero so the character is always 7-bits.

The procedure for transferring a character from the KS10 to the Console is a follows:

1. The KS10 places a character with the **VALID** bit set in memory location 0000032.
2. The KS10 interrupts the Console Processor.
3. The Console Processor is interrupted.
4. If the **VALID** bit is asserted, the Console Processor extracts the character from location 000032 bits 28-35 and outputs the character on the CTY serial output port. If the **VALID** bit is negated, the interrupt function should not process a character from the KS10. Note: the interrupt may indicate that another character should be transferred to the KS10. See section 5.4.3.1.
5. The Console Processor zeros location 000032. This zeros the **VALID** bit.
6. The Console Processor Interrupts the KS10 by setting the **KS10_INTR** bit in the Console Control/Status Register. See Table 10.

The CTY Output Word should be initialized to zero by the Console Processor before starting the KS10.

5.4.4 KLINIK Protocol

The KILINK interface is not implemented.

5.4.4.1KLINIK Input Protocol

The KILINK interface is not implemented. The table below simply documents the bits in this interface.

Table 27 – KS10 KLINIK Input Word (KS10 Memory Address 000034)		
Bit(s)	**Mnemonic**	**Description**
0-25	Reserved	Ignored for reads and writes.
26-27	KLCHR	0: nothing 1: character available 2: KLINIK initialized 3: carrier lost
28-35	KLIICH	KLINIK character (ASCII).

The KLINIK Input Word should be initialized to zero by the Console Processor before starting the KS10.

5.4.4.2KLINIK Output Protocol

The KILINK interface is not implemented. The table below simply documents the bits in this interface.

Table 28 – KS10 KLINIK Output Word (KS10 Memory Address 000035)		
Bit(s)	**Mnemonic**	**Description**
0-25	Reserved	Ignored for reads and writes.
26	KLHUP	KLINIK hang-up request
27	VALID	KLINIK character available.
28-35	KLIOCH	KLINIK character (ASCII).

The KLINIK Output Word should be initialized to zero by the Console Processor before starting the KS10.

5.4.5 Boot RH11 Address

This address sets the RH11 Base Address.

Table 29 – KS10 RH11 Address Word (KS10 Memory Address 000036)		
Bit(s)	**Mnemonic**	**Description**
0-13	Zero	Ignored
14-36	ADDR	RH11 Base Address

5.4.6 Boot Unit Number

This selects which device (unit) on the RH11 the system will boot from.

Table 30 – KS10 Boot Unit Number Word (KS10 Memory Address 000037)

Bit(s)	Mnemonic	Description			
0-32	Zero	Must be zero			
33-35	UNIT	Unit number;			
		Bit 33	**Bit 34**	**Bit 35**	**Unit**
		0	0	0	Disk 0
		0	0	1	Disk 1
		0	1	0	Disk 2
		0	1	1	Disk 3
		1	0	0	Disk 4
		1	0	1	Disk 5
		1	1	0	Disk 6
		1	1	1	Disk 7

5.4.7 Boot Magtape Parameters

This 36-bit parameter gets copied (right justified) to a 16-bit UBA Tape Control Register.

It is unlikely that the KS10 FPGA will implement the Tape Unit. Therefore this register does nothing.

Table 31 – KS10 Boot Magtape Parameter Word (KS10 Memory Address 000040)		
Bit(s)	**Mnemonic**	**Description**
0-19	Zero	Ignored.
20	ACC	Accelerating NI.
21	FCS	Frame count status.
22	SAC	Slave address change.
23	AER	Abort on error.
24	Zero	Must be zero.
25-27	DEN	Density 011 – 800 BPI 100 – 1600 BPI
28-31	FMT	Format 0000 – Core dump 0011 – ANSI
32	EVN	Even parity
33-35	UNIT	Unit

6 KS10 Memory Controller

The KS10 FPGA Memory Controller attempts to be fully compatible with the DEC KS10 Memory Controller. Whereas the DEC KS10 Memory Controller interfaces to multiple dynamic MOS boards, the KS10 FPGA Memory Controller interfaces to a single Pipelined SSRAM device.

6.1 Memory Status Registers

The Memory Status Register is a 36-bit IO register located at IO Address o100000. The Memory Status Register in the DEC KS10 provides status information about KS10 memory status.

The KS10 FPGA does not require or support memory Error Detection and Correction (EDAC). The Memory Status Register bits are implemented as required to be compatible with a real KS10 but none of the underlying functionality is implemented.

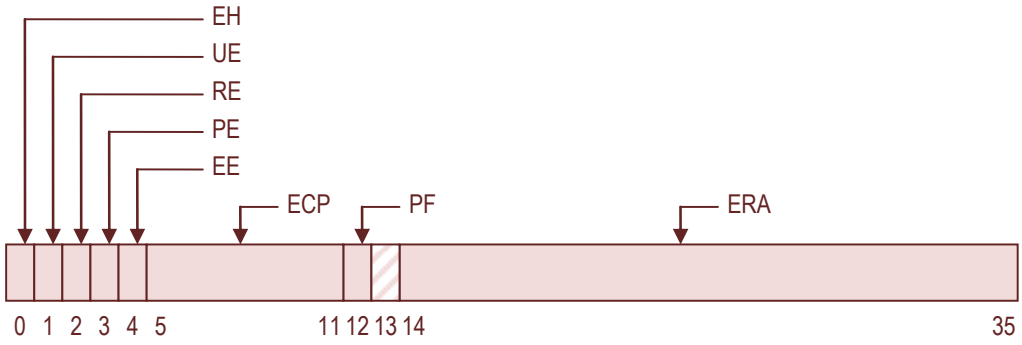


Figure 57 – Memory Status Register (Read)

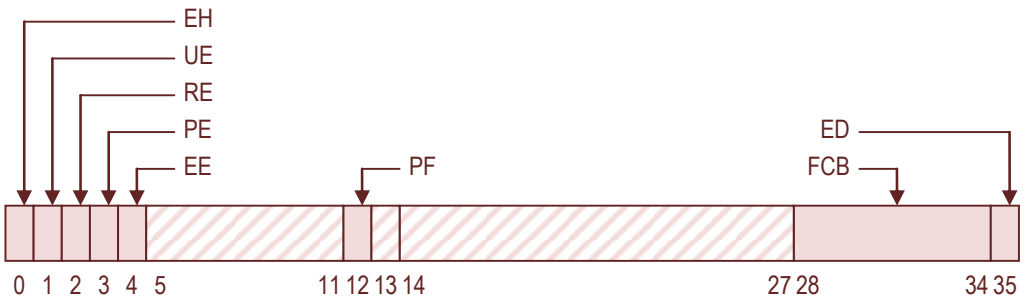


Figure 58 – Memory Status Register (Write)

Table 32 – Memory Status Register Definitions			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0	EH	R	Error Hold. Not implemented. Always read as 0. Writes are ignored.

Table 32 – Memory Status Register Definitions			
Bit(s)	**Mnemonic**	**R/W**	**Description**
1	UE	R	Uncorrectable Read Error aka BAD DATA. Not implemented. Always read as 0. Writes are ignored.
2	RE	R	Refresh Error. Not implemented. Always read as 0. Writes are ignored.
3	PE	R/W	Parity Error. Not implemented. Last bit written is read-back.
4	EE	R	ECC Enable. Not implemented. Reads back inverse value set by write to bit ECC DISABLE bit. Writes are ignored. See ECC DISABLE bit below.
5-11	ECC	R	ECC syndrome. Writes ignored. Read as zero.
12	PF	R/W	Power Fail. Not implemented. Initialized to 1 at power-up. Writing zero clears POWER FAIL.
14-35	ERA	R	Error Read Address – i.e., the address of the last ECC error. Not implemented. Always read as 0. Writes are ignored.
28-34	FCB	W	Force Check Bits. Writes are ignored. Read as part of Error Read Address.
35	ED	W	Read as part of ERROR READ ADDRESS. Writing zero sets ECC ENABLE bit. Writing one clears ECC ENABLE bit

6.2 SSRAM Memory Interface

The KS10 FPGA prototypes have supported two different memory interfaces. The

Xilinx Spartan 6 prototype implemented a 1M x 36-bit NoBL Pipelined Synchronous Static RAM (SSRAM)

while the DE10-nano prototype implemented a 2M × 18 NoBL Flow-Through Synchronous Static RAM (SSRAM).

These implementations will be described in the following sections.

6.2.1 18-bit SSRAM Memory Interface

The DE10-nano prototype elected to use a slightly more complex memory architecture in order to save FPGA pins. This interface includes a Cypress CY7C1463KV33 18-bit SSRAM that operates with a 2-word burst

The SSRAM interface operates at four times the CPU clock rate in order to perform these burst memory reads and memory writes in a single CPU clock cycle.

The burst operation is designed such that the high word of memory (DATA[0:17]) is written with the LSB of the SSRAM Address (A0) negated. The low word of memory (DATA[18:35]) is always written with the LSB of the SSRAM Address (A0) asserted. Although the SSRAM burst control pin (ADV/LD#) is wired to the FPGA, it is not used and is always negated.

The diagram below illustrates the SSRAM bus cycles that have been implemented.

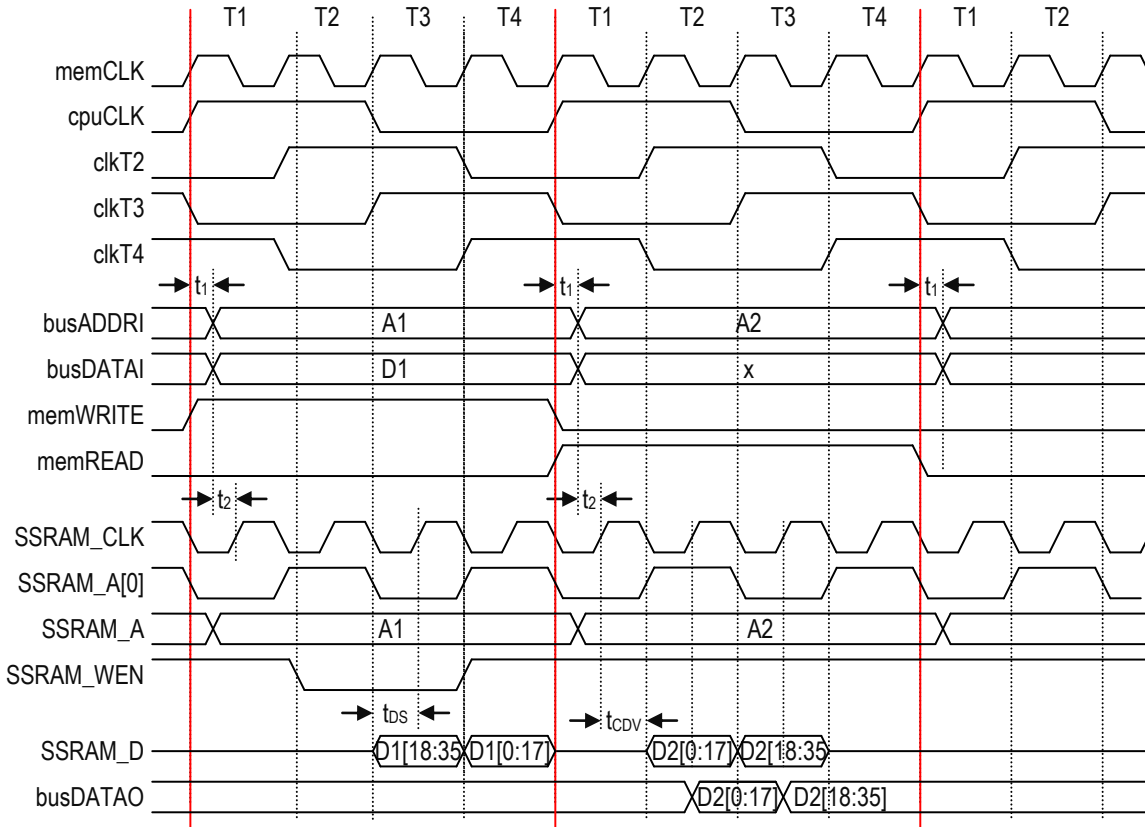


Figure 59 – 18-bit SSRAM Burst Bus Cycles

6.2.1.1 18-bit SSRAM Burst Read Cycle

In read cycles, the SSRAM Write Enable (SSRAM_WE_N) is always negated.

In T1, the FPGA asserts the address of the high-word of memory (DATA[0:17]) on the SSRAM Address Bus (A0 is negated).

In T2:

1. the FPGA asserts the address of the low-word of memory (DATA[18:35]) on the SSRAM Address Bus (A0 is asserted).
2. the SSRAM drives the high-word of data onto the SSRAM Data bus.
3. the FPGA captures the high-word of data is captured in the middle of T2.

In T3,

1. the SSRAM drives the low-word of data onto the SSRAM Data bus.
2. the FPGA captures the high-word of data is captured in the middle of T3.

6.2.1.2 18-bit SSRAM Burst Write Cycle

In T1:

1. the FPGA speculatively starts a read cycle as described above in the read cycle description.
2. the FPGA determines that this is a write cycle and sets up T2 as a write operation

In T2:

1. the FPGA asserts the address of the low-word of memory (DATA[18:35]) on the SSRAM Address Bus (A0 is asserted). This is the same as the read cycle.
2. the FPGA asserts SSRAM Write Enable (SSRAM_WE_N) for the low-word of memory.

In T3:

1. the FPGA asserts the address of the high-word of memory (DATA[0:17]) on the SSRAM Address Bus (A0 is negated). This is the same as the read cycle - but the read cycle doesn't care about address during this cycle.
2. the FPGA asserts SSRAM Write Enable (SSRAM_WE_N) for the high-word of memory.
3. the FPGA drives the low-word of memory (DATA[18:35]) on the SSRAM Data Bus

In T4:

1. the FPGA asserts the address of the low-word of memory (DATA[18:35]) on the SSRAM Address Bus (A0 is asserted). At this point, neither the read cycle or write cycle care about address.
2. the FPGA negates SSRAM Write Enable (SSRAM_WE_N)
3. the FPGA drives the low-word of memory (DATA[0:17]) on the SSRAM Data Bus

From the description, you can observe that A0 is negated in T1, is asserted in T2, is negated in T3, and is asserted in T4. In other words, A0 just alternates.

It is also worth noting that the SSRAM is read high-word first while the SSRAM is written low-word first because of the way the burst addressing works out.

6.2.2 36-bit SSRAM Memory Interface

This interface supports the older prototype board. It is not actively maintained.

The KS10 Memory Interface is design to accommodate a Cypress CY7C1460AV33 - 1M x 36 NoBL Pipelined Synchronous Static RAM (SSRAM). This memory has a 2 stage pipeline between the address signals and the memory array.

The device is capable of operating at a 166 MHz clock rate.

Because the memory device has a 2 stage pipeline, portions of the memory controller operate at four times the CPU clock rate. This creates the illusion that memory reads and memory writes complete in a single CPU clock cycle.

For writes operations the write enable signal (ssramWE_N) is asserted at the beginning (rising edge) of T2. The SSRAM device registers the address (ssramADDR[0:19]) and data (ssramDATA[0:35]) on the rising edge of the SSRAM Clock (ssramCLK). The write operation actually completes two clock cycles later at the beginning (rising edge) of T4. The address and data buses are generally unstable or contain the previous address and data during T1.

For read operations the output enable signal (ssramOE_N) is always asserted. In this mode, the SSRAM device will configure the direction of the SSRAM data bus based on the operation of the write enable signal (ssramWE_N). The data bus will be an output from the SSRAM except for two clock cycles after the write enable signal is asserted.

For reads, the address is sampled continuously but the memory device is enabled only in T4.

The memWRITE signal controls the SSRAM data bus interface direction. If the memWRITE signal is asserted, the FPGA assertes the busDATAI[0:35] signals onto the SSRAM data bus. If the memWRITE signal is negated, the FPGA tristates SSRAM data bus and the SSRAM data may be read from the SSRAM device.

The design currently supports 36-bit wide memory but could accommodate an 18-bit wide memory with a two word burst. Using18 bit wide memory would save pins and not impact performance.

The timing diagram of a read cycle is illustrated below in Figure 60.

The timing diagram of a write cycle is illustrated below in Figure 61.

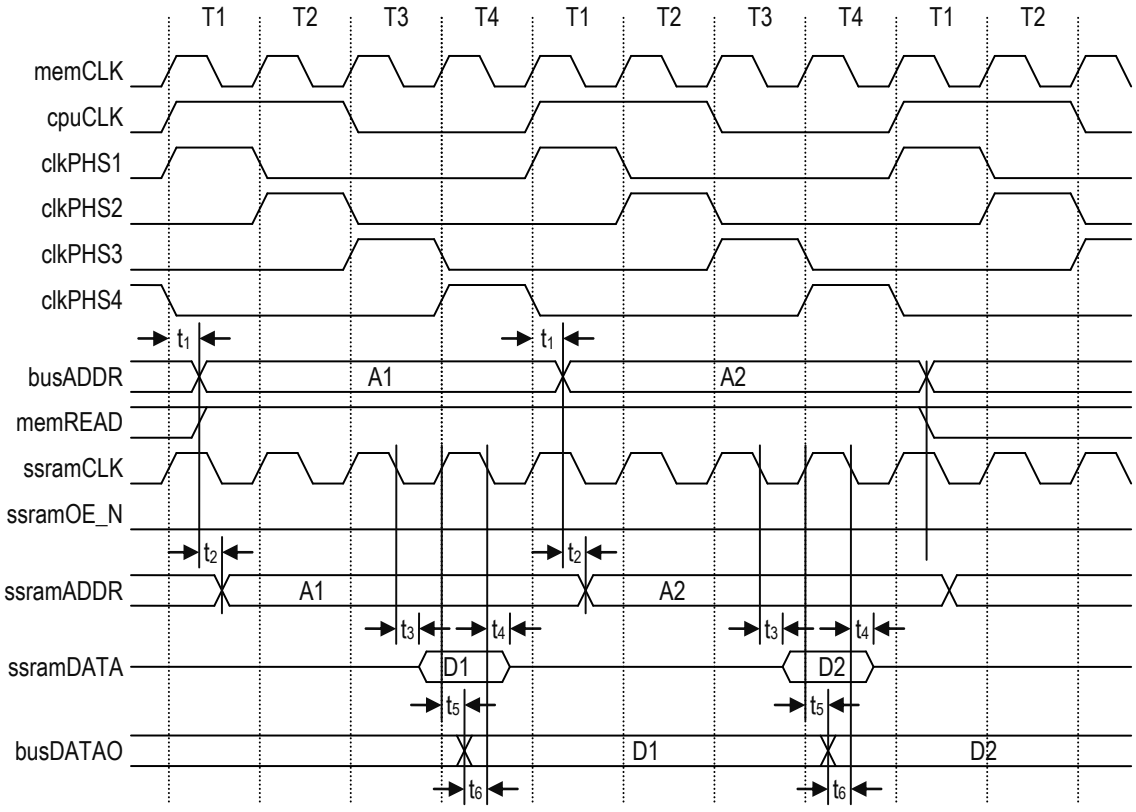


Figure 60 - SSRAM Read Timing Diagram

Table 33 – SSRAM Read Timing Parameters		
Parameter	**Value**	**Description**
t_1	TBD	Memory Controller clock to address and data valid
t_2	TBD	SSRAM address delay
t_3	TBD	SSRAM OE to data valid
t_4	TBD	SSRAM OE to data
t_5	TBD	Memory controller data delay
t_6	TBD	Memory controller data to clock setup time

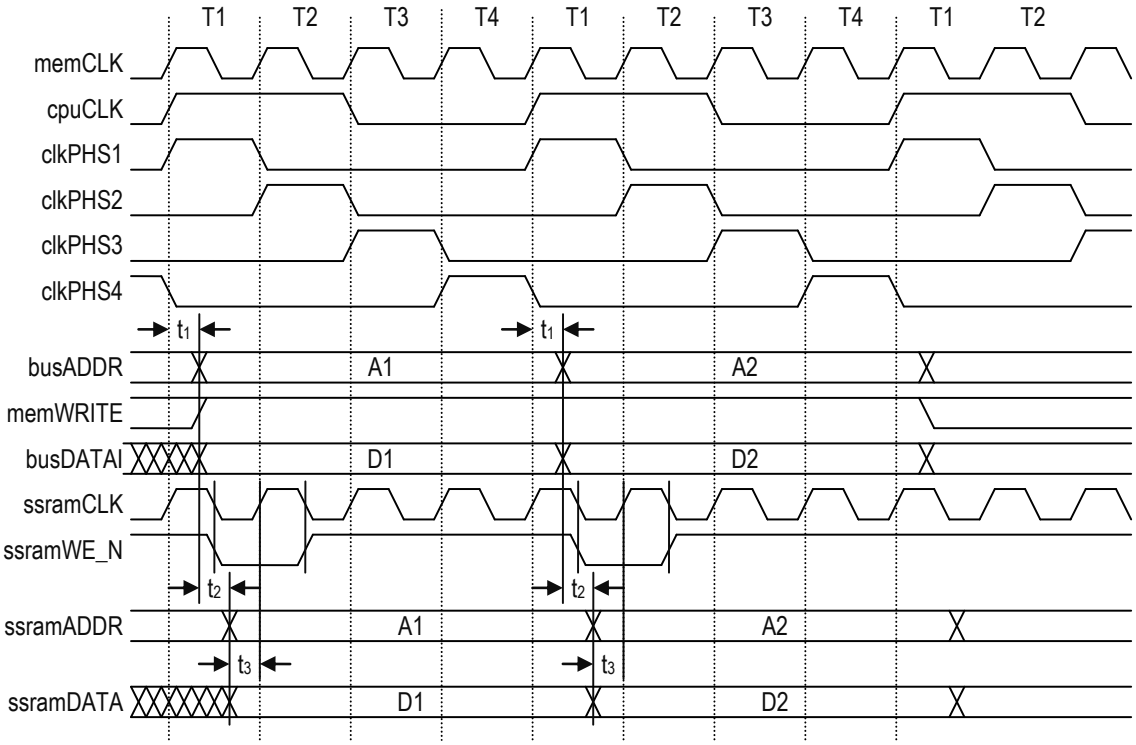


Figure 61 - SSRAM Write Timing Diagram

Table 34 – SSRAM Write Timing Parameters		
Parameter	**Value**	**Description**
t_1	TBD	Memory Controller clock to address and data valid
t_2	TBD	SSRAM address and data delay
t_3	TBD	SSRAM data valid to sample setup time

7 KS10 IO Bus (Unibus) Bridge

The DEC KS10 system architecture supports up to three IO Bridges that interface KS10 backplane to standard Unibus Devices. These Unibus Adapters are commonly referred to as UBA1, UBA3, and UBA4; although the KS10 Technical Manual only documents the possibility of UBA1 and UBA3.

Apparently there is a limitation in the KS10 backplane (wiring?) that prevents the use of UBA2, but UBA4 works in "non-standard" systems.

"TOPS-20 looks at UBA1, UBA3, and UBA4. TOPS-10 actually looks for devices on all four but of course never finds UBA2 installed."[1]

Lastly, and most importantly, the DECSYSTEM 2020 UNIBUS ADAPTER EXERCISER (DSUBA) will test UBA1, UBA3, and UBA4 per below:

```
DECSYSTEM 2020 UNIBUS ADAPTER EXERCISER  [ DSUBA ]
VERSION 0.4, SV=0.3, CPU#=2020, MCV=130, MCO=470, HO=0, KASW=003740 000000

TTY SWITCH CONTROL ? - 0,S OR Y <CR> - 0
SWITCHES = 000000 000000

MEMORY MAP =
FROM     TO          SIZE/K
00000000 03777777       1024


WHICH UNIBUS ADAPTER? (1,3,4):
```

For now, only UBA1 and UBA3 are implemented in the FPGA.

In a DEC KS10 implementation, all of the normal IO is implemented by 18-bit wide Unibus Devices. The KS10 FPGA implements register-compatible IO Bridges but does not attempt to implement the Unibus hardware or protocol inside the FPGA. The KS10 FPGA IO Bus is a synchronous, 36-bit wide, de-multiplexed address and data bus. It is in effect, an extension of the backplane bus.

7.1 IO Bus Bridge Registers

7.1.1 IO Bus Bridge Control Status Register (UBACSR)

The IO Bridge Control Status Register (UBACSR) is a 36-bit IO register located at IO Address o763100 and is compatible with the Unibus Status Register of the DEC KS10.

[1] Personal correspondence with Timothe Litt.

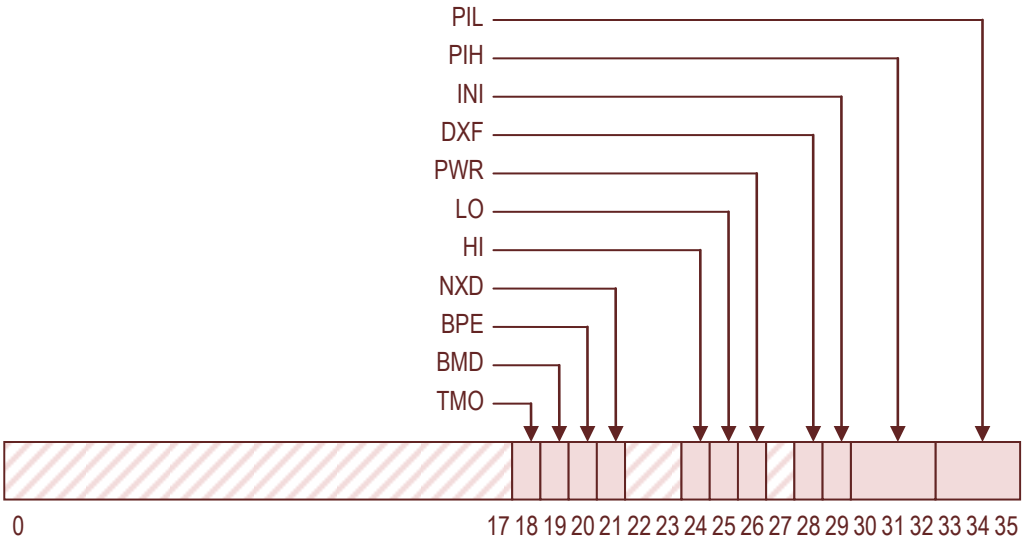


Figure 62 – IO Bridge Control Status Register (UBACSR)

Table 35 – IO Bridge Control Status Register (UBACSR) Definitions			
Bit(s)	**Mnemonic**	**R/W**	**Description**
18	TMO	R/W	Adapter timeout. This is set under the following conditions: 1. Adapter accesses memory that does not exist, or 2. Device creates an NPR access with A17 asserted, or 3. Device creates an NPR access with A1 asserted, or 4. Device creates an NPR access with A0 asserted, or 5. Device creates an NPR access with the Page Valid Flag negated, or Cleared by writing a one to TMO or by writing a one to INI.
19	BMD	R	Bad Memory Data. Not implemented. Always read as zero. Writes are ignored.
20	BPE	R	Bus Parity Error. Not implemented. Always read as zero. Writes are ignored.
21	NXD	R/W	Non-existent Device. Set when accessing an IO device attached to this IO Bridge that does not exist. Cleared by writing a one to NXD or by writing a one to INI.
22	-	-	Always read as zero
23	-	-	Always read as zero
24	HI	R	Hi Interrupt. Asserted when there is an IRQ on BR7 or BR6. Writes are ignored.

Table 35 – IO Bridge Control Status Register (UBACSR) Definitions			
Bit(s)	**Mnemonic**	**R/W**	**Description**
25	LO	R	Lo Interrupt. Asserted when there is an IRQ on BR5 or BR4. Writes are ignored.
26	PWR	R	Power Fail. Not implemented. Always read as zero. Writes are ignored.
27	-	-	Always read as zero
28	DXF	R/W	Disable Transfer. Read/Write. Not implemented. Cleared by writing a one to INI.
29	INI	R/W	Initialize. Writing 1 resets all devices on this IO Bridge. The KS10 has a one-shot that asserts this signal for 1 µS.
30-32	PIH	R/W	Priority Interrupt High PIH maps the priority of the high priority device interrupts, devINTR[7] and devINTR[6], to the CPU interrupt priority defined by the PIL register. PIH is set by writing to this register. PIH is cleared by: 1. Writing a zero to PIH, or 2. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1). Note: There are seven CPU interrupt priorities - Priority 1 through Priority 7. Setting PIL to 0 disables all high priority interrupts from this IO Bridge.
33-35	PIL	R/W	Priority Interrupt Low PIL maps the priority of the low priority device interrupts, devINTR[5] and devINTR[4], to the CPU interrupt priority defined by the PIL register. PIL is set by writing to this register. PIL is cleared by: 1. Writing a zero to PIL, or 2. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1). Note: There are seven CPU interrupt priorities - Priority 1 through Priority 7. Setting PIL to 0 disables all low priority interrupts from this IO Bridge.

7.1.2 IO Bus Bridge Maintenance Register

The IO Bridge Maintenance Register is a 36-bit IO register located at IO Address o763101 and is compatible with the Unibus Maintenance Register of the DEC KS10.

Figure 63 – IO Bridge Maintenance Register (UBAMR)

Table 36 – IO Bridge Maintenance Register (UBAMR) Definitions			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0-34	Reserved	W	Writes ignored.
35	MAINT	W	Maintenance Mode.

Some of the Maintenance Loopback feature of the IO Bridge is implemented as required to pass the DSUBA diagnostics.

7.2 IO Bus Bridge Paging

7.2.1 IO Bus Bridge Paging Memory

The IO Bus Page Translation Memory is a sequence of memory locations at IO Address o76300 – o763077 and is register-compatible with the Unibus Paging Memory of the DEC KS10.

The paging memory is a lookup table that is used to translate the IO Virtual Address to the KS10 Physical Address. For each of the 64 Virtual Pages there are a possible 2048 Physical Pages.

The format of the IO Bridge Paging Memory when written is detailed in Figure 64. The format during a read operation is detailed in Figure 65.

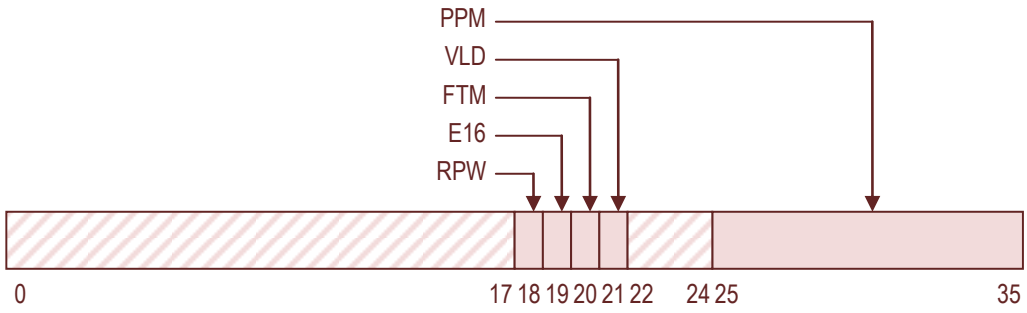


Figure 64 – IO Bridge Paging RAM Write

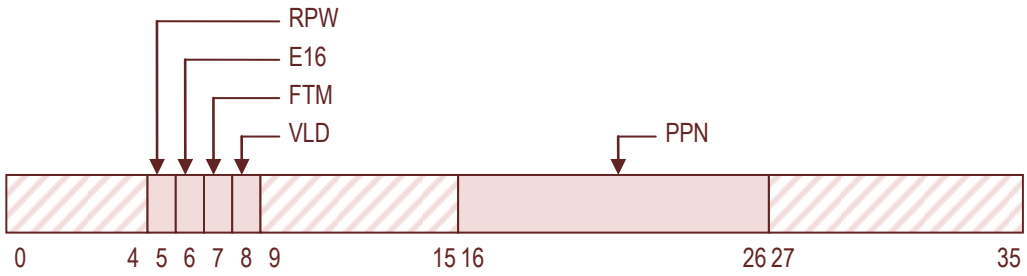


Figure 65 – IO Bridge Paging RAM Read

Table 37 – IO Bridge Paging RAM Definitions			
Bit(s)	**Mnemonic**	**R/W**	**Description**
5/18	RPW	R/W	Force Read-Pause-Write. This is also known as “Read Reverse” in some of the documents. This is implemented as required for the maintenance loopback diagnostics. This bit is ignored for IO Bus transactions which are never RPW.
6/19	E16	R/W	Enable 16-bit IO Bus Transfers and disable 18-bit IO Bus Transfers. Not implemented. IO Bus transactions are always 36-bit.
7/20	FTM	R/W	Fast Transfer Mode. In this mode, both odd and even words of Unibus data were transferred during a single KS10 memory operation. This is implemented as required for the maintenance loopback diagnostics. This bit is ignored for IO Bus transactions which are always 36-bit.
8/21	VLD	R/W	Page valid. This bit is set when the page data is loaded.
16-26 25-36	PPN	R/W	Physical Page Number.

The format of the IO Bridge Paging RAM Read is chosen so that the PPN is in the correct bit positions when performing paging in software.

7.2.2 IO Bus Page Translation

The IO Bus Paging converts IO Bus Virtual Addresses to Physical Addresses in a manner that is similar and consistent to the KS10 processor paging. The IO Bus Paging translates a 16-bit Unibus-compatible IO address to a 20-bit KS10-compatible physical memory address.

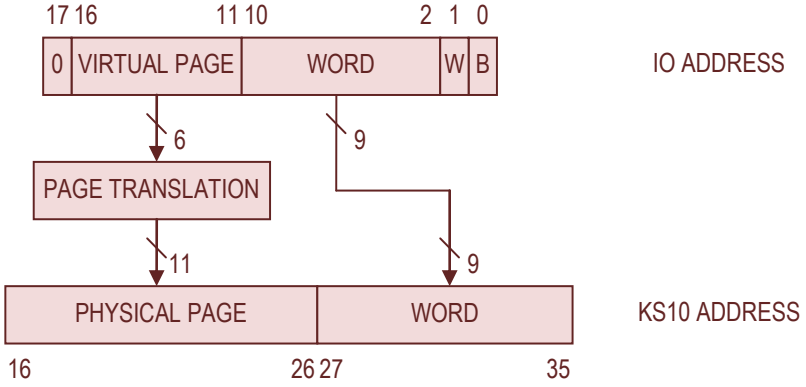


Figure 66 – IO Bus Page Translation

7.3 IO Bus Bridge Address Mapping

The UBA can provide address mapping between the 36-bit KS10 bus and 16-bit and 8-bit peripherals. This translation is illustrated below in Figure 67.

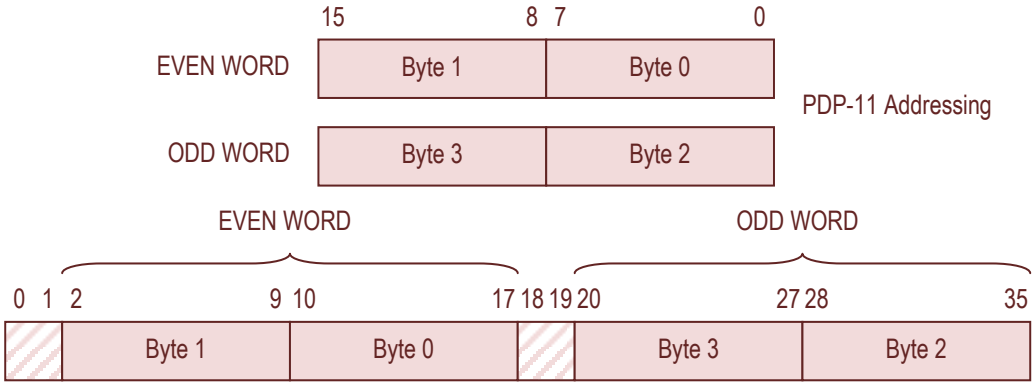


Figure 67 – IO Bus Byte and Word Translation

Table 38 – UBA Address Translation			
UBA A1	**UBA A0**	**OP**	**UBA Data Description**
0	0	Byte	Byte 0 (Even word, low byte)
0	1	Byte	Byte 1 (Even word, high byte)
1	0	Byte	Byte 2 (Odd word, low byte)
1	1	Byte	Byte 3 (Odd word, high byte)
0	0	Word	Even word

Table 38 – UBA Address Translation			
UBA A1	**UBA A0**	**OP**	**UBA Data Description**
1	0	Word	Odd word

8 DUP11 Synchronous Communications Adapter

The DUP11 Synchronous Line Interface provides a relative high speed (for the time) interface to other computers.

The DUP11 Verilog implementation is fully parameterized: the base IO address and the interrupt vector are controlled by module parameters. Two DUP11s could be instantiated and attached to the IO Bus Bridge (UBA) adapters - however only on DUP11 is currently instantiated in the code.

The configuration parameters of these devices are summarized below in Table 39.

Table 39 – DUP11 Configuration

Device	UBA	Interrupt	Interrupt Vector	Base Address
DUP11 #1	UBA3	5	RX: 000560 TX: 000564	760300
DUP11 #2	UBA3	5	RX: 000600 TX: 000604	760310

8.1 Synchronous Serial Protocols

8.1.1 DDCMP/BISYNC Mode

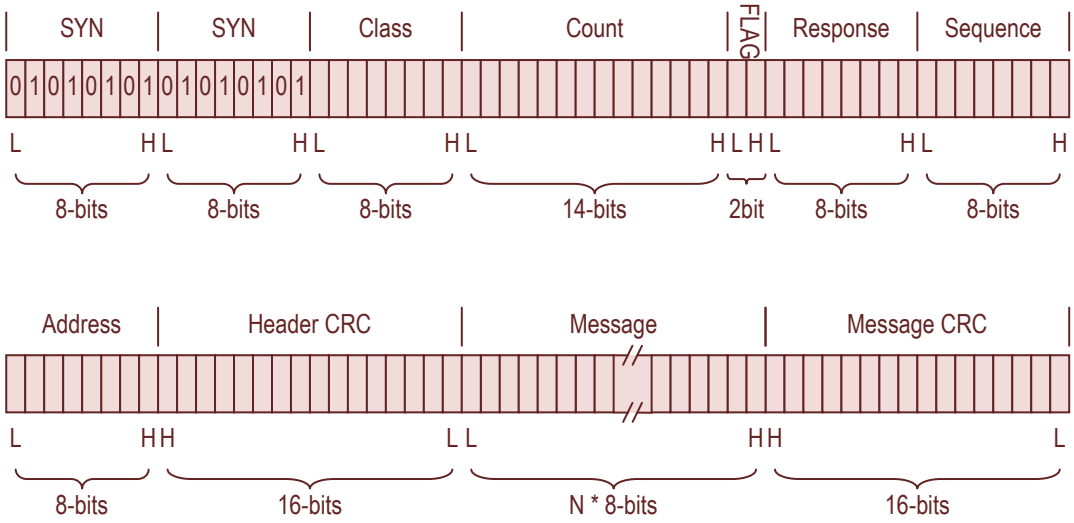


Figure 68 – DDCMP Message Format

8.1.2 SDLC/ADCCP Mode

The SDLC protocol is illustrated below.

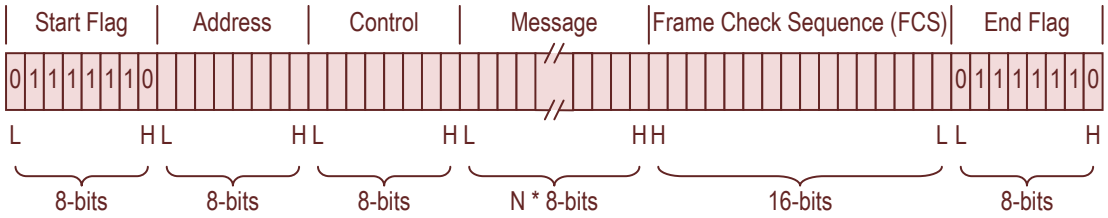


Figure 69 – SDLC Message Format

The CRC check is performed over the Address, Control, and Message fields.

Zero-bit insertion and zero-bit deletion is performed over the Address, Control, Message, and FCS fields.

8.1.3 SDLC Receiver Synchronization

8.2 DUP11 Registers

A summary of DUP11 registers is shown below.

Table 40 – DUP11 Register Summary

IO Addr (Dev 1)	IO Addr (Dev 2)	Register Name	Access	Register Description
760300	760310	RXCSR	Byte	Receiver Control/Status Register (R/W)
760302	760312	RXDBUF	Word	Receiver Data Buffer (R)
760302	760312	PARCSR	Word	Parameter Control/Status Register (W)
760304	760314	TXCSR	Byte	Transmitter Control/Status Register (R/W)
760306	760316	TXDBUF	Byte	Transmitter Data Buffer (R/W)

8.2.1 DUP11 Receiver Control/Status Register (RXCSR)

The RXCSR provides most of the control and status applicable to the receiver operation, including the modem control signals.

The RXCSR is read/write and is word-addressable and byte-addressable.

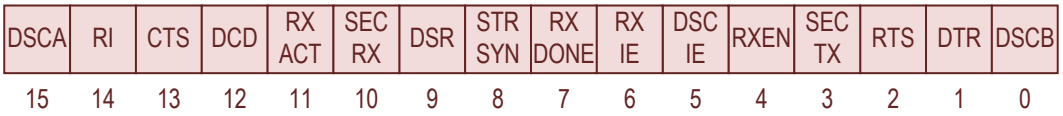


Figure 70 – DUP11 Receiver Control and Status Register (RXCSR)

Table 41 – DUP11 RX Control/Status Register (RXCSR) – IO Address 760300			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	DSCA	R	Data Set Change A This bit is set when: 1. Any transition on the Ring Indication Line (RXCSR[RI]), or 2. Any transition on the Clear to Send Line (RXCSR[CTS]), or 3. Configuration W5 is installed and any transition on the Data Carrier Detect Line (RXCSR[DCD]), or 4. Configuration W5 is installed and any transition on the Data Set Ready Line (RXCSR[DSR]), or 5. Configuration W5 is installed and any transition on the Secondary Received Data Line (RXCSR[SECRX]). This bit is cleared when: 1. *Controller Clear* (TXCSR[INIT]), or 2. *IO Bridge Clear* (UBACSR[INI] = 1), or 3. After this register, RXCSR, is read. If the Data Set Change interrupt is enabled (RXCSR[DSCIE] = 1), the assertion of this bit causes an interrupt to the receiver vector. Writes ignored. Note: Configuration W5 is normally not installed. The manual entitled DUP11Bit Synchronous Interface Maintenance Manual (EK-DUP11-MM-003) says that “A positive transition on the Ring line greater than 10 ms” will set DSCA. This statement is misleading if not incorrect. Refer to Figure 4-31 for the schematic of the Ring Indication Line. This circuit detects both rising and falling edges. The DSDUA diagnostic Test 61 will fail if the circuit only detects rising edges. The 10 millisecond debounce is not implemented.

Table 41 – DUP11 RX Control/Status Register (RXCSR) – IO Address 760300			
Bit(s)	**Mnemonic**	**R/W**	**Description**
14	RI	R	Ring Indication This bit reflects the state of the modem Ring Indication modem signal. Writes ignored.
13	CTS	R	Clear To Send This bit reflects the state of the Clear to Send modem signal. Writes ignored.
12	DCD	R	Data Carrier Detect This bit reflects the state of the Data Carrier Detect modem signal. Writes ignored.
11	RXACT	R	Receiver Active This bit is set when: 1. In SDLC or ADCCP mode (PARCSR[DECMD] = 0) and the first character of a message is received, or 2. In DDCMP or BISYNC mode (PARCSR[DECMD] = 1) and the first character after the two SYNC symbols is received. This bit is cleared when: 1. The Receive Enable (RXCSR[RXEN]) transitions to disabled, or 2. In SDLC or ADCCP Mode (PARCSR[DECMD] = 0) and an ABORT sequence is received, or 3. *Controller Clear* (TXCSR[INIT]), or 4. *IO Bridge Clear* (UBACSR[INI] = 1). Writes ignored.
10	SECRX	R	Secondary Received Data This bit would normally reflect the state of the Secondary Received Data modem signal. The secondary receive data function is not implemented. This bit is looped-back to the SECTX bit, therefore this bit reflects the state of the RXCSR[SECTX] bit. Writes ignored.
9	DSR	R	Data Set Ready This bit reflects the state of the Data Set Ready modem signal. Writes ignored.

Table 41 – DUP11 RX Control/Status Register (RXCSR) – IO Address 760300			
Bit(s)	**Mnemonic**	**R/W**	**Description**
8	STRSYN	R/W	Strip Sync Once the receiver is synchronized, any received characters that match the Sync Character (PARCSR[SYNADR]) and are contiguous with the sync sequence are not presented to the program (i.e., the receiver done (RXCSR[RXDONE] will not be set). This strips extra sync characters that may be present in the sync sequence. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1). Note: This bit is only used with the DDCMP and BISYNC protocols (PARCSR[DECMD] = 1). Setting this bit in SDLC and ADCCP mode (PARCSR[DECMD] = 0) will disable the receiver.
7	RXDONE	R	Receive Done. This bit indicates that data is available in the Receiver Buffer Register (RXDBUF[RXDBUF]). This bit is set when: 1. The receiver is active (RXCSR[RXACT] = 1) and a character is transferred from the Receiver Shift Register into the Receiver Buffer (RXDBUF[RXDBUF]), or 2. In SDLC/ADCCP mode and an Abort Sequence is received, or 3. In DDCMP/BISYNC mode, and the Strip Sync bit is not set (RXCSR[STRSYN] = 0), and a SYN character is received immediately following the SYNC character. This bit is cleared by: 1. Reading the Receiver Buffer (RXDBUF[RXDBUF]), or 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1). Receiver Done is NOT set when a the Address Character is received and the receiver is configured for Secondary Station Mode (PARCSR[SSM] = 1). Writes ignored.

Table 41 – DUP11 RX Control/Status Register (RXCSR) – IO Address 760300			
Bit(s)	**Mnemonic**	**R/W**	**Description**
6	RXIE	R/W	Receiver Interrupt Enable When this bit is asserted, an receiver interrupt is generated when the receiver has data (RXCSR[RXDONE] = 1). This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1).
5	DSCIE	R/W	Data Set Change Interrupt Enable When enabled causes a receiver interrupt request when the Data Set Change A (RXCSR[DSCA]) bit is set. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1).
4	RXEN	R/W	Receiver Enable When initially set, this causes the receiver to search for the synchronization sequence irrespective of mode. Once synchronization has been attained, clearing this bit will asynchronously (to the receiver clock) reset the receiver state. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1).
3	SECTX	R/W	Secondary Transmitted Data Secondary mode is selected by asserting PARCSR[SECMODE]. The secondary transmit data function is not implemented. It simply loops back to the Secondary Receiver Input. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Configuration W3 is installed and Controller Clear (TXCSR[INIT] = 1), or 3. Configuration W3 is installed and IO Bridge Clear (UBACSR[INI] = 1). Note: Configuration W3 is normally installed.

Table 41 – DUP11 RX Control/Status Register (RXCSR) – IO Address 760300			
Bit(s)	**Mnemonic**	**R/W**	**Description**
2	RTS	R/W	Request To Send This bit asserts the RTS signal to the Modem. This bit is set by writing a '1'. This bit is cleared by: 1. Writing a 0, or 2. Configuration W3 is installed and Controller Clear (TXCSR[INIT] = 1), or 3. Configuration W3 is installed and IO Bridge Clear (UBACSR[INI] = 1). Note: Configuration W3 is normally installed.
1	DTR	R/W	Data Terminal Ready This bit asserts the DTR signal to the Modem. This bit is set by writing a '1'. This bit is cleared by: 1. Writing a 0, or 2. Configuration W3 is installed and Controller Clear (TXCSR[INIT]), or 3. Configuration W3 is installed and IO Bridge Clear (UBACSR[INI] = 1). Note: Configuration W3 is normally installed.
0	DSCB	R	Data Set Change B This bit is set when: 1. Configuration W6 is installed and a transition of the Data Carrier Detect (RXCSR[DCD) signal, or 2. Configuration W6 is installed and a transition of the Data Set Ready (RXCSR[DSR]) signal, or 3. Configuration W6 is installed and a transition of the Secondary Received Data (RXCSR[SECRX]) signal. This bit is cleared when: 1. Controller *Clear* (TXCSR[INIT]), or 2. *IO Bridge Clear* (UBACSR[INI] = 1), or 3. RXCSR is read. Writes ignored. Note: Configuration W6 is normally installed.

8.2.2 DUP11 Received Data Buffer (RXDBUF)

The upper byte of this register contains the remainder of the receiver status information, including the

Receiver error flags. The lower byte provides access to the receiver buffer register which contains the received data.

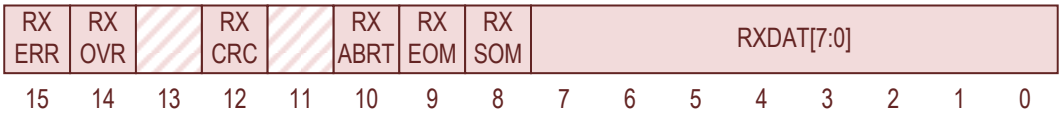


Figure 71 – DUP11 Receiver Data Buffer (RXDBUF)

Table 42 – DUP11 RX Data Buffer Register (RXDBUF) – IO Address 760302			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	RXERR	R	Receiver Error This bit is set when any of the following bits are set: 1. Receiver Overrun Error (RXDBUF[RXOVR]), or 2. In SDLC/ADCCP Mode (PARCSR[DECMD] = 0) and Receiver CRC Error (RXDBUF[RXCRC]), or 3. In SDLC/ADCCP Mode (PARCSR[DECMD] = 0) and Receiver Abort Error (RXDBUF[RXABRT] Clearing the errors above will clear this error.
14	RXOVRE	R	Receiver Overrun Error An overrun occurs data is present (RXCSR[RXDONE] = 1) but the data is not read from the Receiver Data Buffer (RXDBUF[RXDAT]) before the next character is received and stored in the Receiver Data Buffer. This bit is set when a receiver overrun occurs and data is lost. This bit is cleared by: 1. Clearing Receiver Enable (RXCSR[RXEN]), or 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1). Note: When set, this bit is only set until the next transfer from the receiver shift register to the Receiver Data Register (RXDBUF[RXDAT]).
13	-	R	Reserved. Always read as zero.

Table 42 – DUP11 RX Data Buffer Register (RXDBUF) – IO Address 760302			
Bit(s)	**Mnemonic**	**R/W**	**Description**
12	RXCRCE	R	Receiver CRC Error This bit is set when: 1. In SDLC/ADCCP mode (PARCSR[DECMD] = 0) and the receiver detects a CRC error in the message, or 2. In DDCMP/BISYNC mode (PARCSR[DECMD=1) and the internal receiver CRC register is zero, or This bit is cleared by: 1. Clearing the Receiver Enable (RXCSR[RXEN]) 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1). Note: When set, this bit is only set until the next transfer from the receiver shift register to the Receiver Data Register (RXDBUF[RXDAT]).
11	-	R	Reserved. Always read as zero.
10	RXABRT	R	Receiver Abort Error This bit is set in SDLC/ADCCP Mode (PARCSR[DECMD] = 1) when an SDLC Abort Sequence is received. This bit is cleared by: 1. Reading this register (RXDBUF), or 2. Clearing Receiver Enable (RXCSR[RXEN]), or 3. Controller Clear (TXCSR[INIT]), or 4. IO Bridge Clear (UBACSR[INI] = 1). This bit is only asserted in SDLC/ADCCP mode (PARCSR[DECMD] = 0) – it is never asserted in DDCMP/ BISYNC mode. Note: An SDLC Abort Sequence is defined as a sequence of eight contiguous ones. When an Abort Sequence is received, the receiver state is reset. When an abort is detected, RXDONE is asserted, and RXACT if negated.

Table 42 – DUP11 RX Data Buffer Register (RXDBUF) – IO Address 760302			
Bit(s)	**Mnemonic**	**R/W**	**Description**
9	RXEOM	R	Receiver End of Message This bit is set in SDLC/ADCCP mode (PARCSR[DECMD] = 1), and receiver synchronization was previously attained, and an End Flag character is received. This bit is cleared by: 1. Clearing Receiver Enable (RXCSR[RXEN]), or 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1). This bit is only asserted in SDLC/ADCCP mode (PARCSR[DECMD] = 0) – it is never asserted in DDCMP/ BISYNC mode. When the End Flag is received, the contents of the Receiver Data Register (RXDBUF[RXDAT]) are undefined. Note: When set, this bit is only set until the next transfer from the receiver shift register to the Receiver Data Register (RXDBUF[RXDAT]).
8	RXSOM	R	Receiver Start of Message This bit is asserted when: 1. In SDLC/ADCCP Mode (PARCSR[DECMD] = 0) and Primary Station Mode (PARCSR[SSM] = 0) and the first bit of the Control Character is received, or 2. In SDLC/ADCCP Mode (PARCSR[DECMD] = 0) and Secondary Station Mode (PARCSR[SSM] = 1) and the first bit of the Address Character is received. This bit is cleared by: 1. Clearing Receiver Enable (RXCSR[RXEN]), or 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1). This bit is only asserted in SDLC/ADCCP mode (PARCSR[DECMD] = 0) – it is never asserted in DDCMP/ BISYNC mode In Primary Station Mode, the Address Character is treated like part of the synchronization process – if the address is incorrect, the hardware keeps searching for a flag character followed by the correct station address. In Secondary Station Mode, the Address character is reported to the KS10.

Table 42 – DUP11 RX Data Buffer Register (RXDBUF) – IO Address 760302			
Bit(s)	**Mnemonic**	**R/W**	**Description**
7-0	RXDAT	R	Receiver Data This field is set when by data is transferred to this receiver from the receiver shift register. This field is cleared by: 1. Clearing Receiver Enable (RXCSR[RXEN]), or 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1), or

8.2.3 DUP11 Parameter Control/Status Register (PARCSR)

The high byte of this register contains the bits that control the DEC Mode, secondary address mode, and the enabling of the CRC logic.

The low byte (bits 0-7) contains the 8-bit secondary station address that is used only when the secondary mode is enabled in the SDLC protocol. In DEC Mode operation, this register contains the SYNC character.

The Parameter Control/Status Register is word accessible only and is write-only.

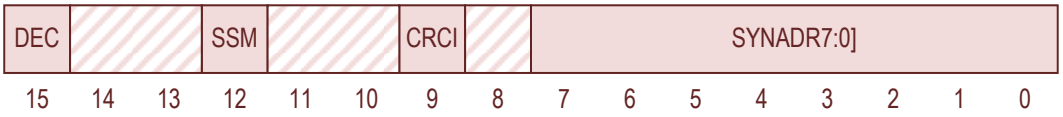


Figure 72 – DUP11 Parameter Control and Status Register (PARCSR)

Table 43 – DUP11 Param Control/Status Register (PARCSR) - IO Address 760302			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	DECMD	W	DEC Mode This controls the operation of the receiver and transmitter protocol state machines. When negated, the device supports SDLC and ADCCP protocols. When asserted the device supports DDCMP and BISYNC protocols. This bit is set by writing a '1'. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1).
14-13	-	W	Reserved. Writes ignored.
12	SSM	W	Secondary Station Mode This bit is used in SLDC mode only. It is cleared in DDCMP and BISYNC modes. If asserted, only messages with the correct secondary address are presented to the program. If negated, all characters after the last flag character are presented to the program. This bit is set by writing a '1'. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1), or
11-10	-	W	Reserved. Writes ignored.
9	CRCI	W	CRC Inhibit This bit modifies the operation of the receiver and transmitter state machine to inhibit the transmission of the CRC and the testing of the received CRC. This bit is set by writing a '1'. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT]), or 3. IO Bridge Clear (UBACSR[INI] = 1), or
8	-	W	Reserved. Writes ignored.

Table 43 – DUP11 Param Control/Status Register (PARCSR) - IO Address 760302			
Bit(s)	**Mnemonic**	**R/W**	**Description**
7-0	SYNADR	W	DDCMP Sync Character / Secondary Station Address When in DDCMP Mode (PARCSR[DECMD] = 1), this parameter is set to the SYN character and is used by the receiver to synchronize to the data stream. In SDLC/ADCCP mode and Secondary station mode, this parameter is loaded with the Secondary Station Address which immediately follows the Flag Character. If the character after Flag Character matches the contents of this field, then receiver synchronization occurs. If it mismatches, the receiver continues to search for a proper synchronization sequence and the message is dropped. These bits are set by writing a 1. These bits are cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1). Note: This parameter is not used by the transmitter.

8.2.4 DUP11 Transmitter Control/Status Register (TXCSR)

The TXCSR provides most of the control and status applicable to the transmitter operation; it also contains the bits to control the DUP11 operation during the maintenance mode.

The TXCSR is read/write and is byte addressable.

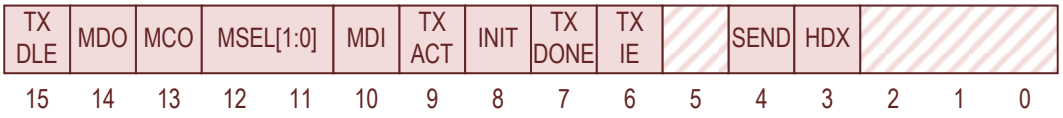


Figure 73 – DUP11 Transmitter Control and Status Register (TXCSR)

Table 44 – DUP11 TX Control/Status Register (TXCSR) - IO Address 760304			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	TXDLE	R	Data Late Error This bit indicates that the transmitter serial port has underrun and there is no data available for the next character. This bit is set when no data is available (TXCSR[TXDONE] = 1). This status is sampled in the middle of the last bit of the character being sent. This bit is cleared by: 1. Starting a new message (TXDBUF[TXSOM] transitions to asserted), or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1). When this occurs in SDLC mode, the transmitter sends an abort character and terminates the transmission. When this occurs in DDCMP mode, the transmitter goes to a “Mark Hold” state, where the transmitter sends a sequence of ones until a new message is started. Writes ignored.
14	MDO	R	Maintenance Data Out When configured for Internal Maintenance Mode (TXCSR[MSEL] = 2), this bit reflects the transmitter output. This is used by the diagnostic program to test the transmitter. Writes ignored.
13	MCO	R/W	Maintenance Clock Out When configured for Internal Maintenance Mode (TXCSR[MSEL] = 2), this bit provides the transmitter and receiver clock and allows the diagnostic software to single-step the transmitter and receiver hardware. A 0-to-1 transition of this bit causes the transmitter to transfer one bit of information to the serial line. A 1-to-0 transition of this bit causes the receiver to shift the contents of the receiver shift register and sample the serial input line. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1).

Table 44 – DUP11 TX Control/Status Register (TXCSR) - IO Address 760304				
Bit(s)	**Mnemonic**	**R/W**	**Description**	
12-11	MSEL	R/W	Maintenance Mode Select These bits are set by writing ones. These bits are cleared by: 1. Writing zeros, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1). The various modes are detailed below.	
			Maintenance Mode	
			0	User Mode (USER) This is the normal operating mode. In this mode, there is no loopback, internal or external, and the modem provides the clock signal for the DUP11 serial interface.
			1	System Test Mode (SYS) This mode performs an external loopback. A modem is not required. In this mode, the clock signal for the DUP11 serial interface which is normally provided by the modem is generated internally by the DUP11. The modem control signals and transmit/received data signals must be looped back by an external device. In a DEC KS10 system, the “H325 Turn Around Connector” provided this loopback mechanism. The description of this bit in Table 3-5 of the document entitled “DUP11 Bit Synchronous Interface User's Manual (EK-DUP11-0P-002)” is incorrect. Similarly, the description of this bit in the document entitled “DUP11 Bit Synchronous Interface Maintenance Manual (EK-DUP11-MM-003)” is incorrect. It is correctly detailed in Table 4-2 of the latter document.
			2	External Maintenance Mode (EXT) This mode performs an internal loopback. In this mode, all of the loopback, clock and data, is performed onboard the DUP11. It cannot check external interfaces. The description of this bit in Table 3-5 of the document entitled “DUP11 Bit Synchronous Interface User's Manual (EK-DUP11-0P-002)” is incorrect. Similarly, the description of this bit in the document entitled “DUP11 Bit Synchronous Interface Maintenance Manual (EK-DUP11-MM-003)” is incorrect. It is correctly detailed in Table 4-2 of the latter document.

Table 44 – DUP11 TX Control/Status Register (TXCSR) - IO Address 760304				
Bit(s)	**Mnemonic**	**R/W**	**Description**	
			3	Internal Maintenance Mode (INT) In this mode, the KS10 bit-bangs the serial clock which allows the KS10 to single-step data through both the serial transmitter and receiver interfaces. The modem control signals and transmit/received data signals must be looped back by an external device. In a DEC KS10 system, the "H325 Turn Around Connector" provided this loopback mechanism. The description of this bit in Table 3-5 of the document entitled "DUP11 Bit Synchronous Interface User's Manual (EK-DUP11-0P-002)" is incorrect. Similarly, the description of this bit in the document entitled "DUP11 Bit Synchronous Interface Maintenance Manual (EK-DUP11-MM-003)" is incorrect. It is correctly detailed in Table 4-2 of the latter document.
10	MDI	R/W	Maintenance Data In When configured for Internal Maintenance Mode (TXCSR[MSEL] = 2), this bit can be used as the receiver serial input. When this bit is set and the Maintenance Clock Out (TXCSR[MCO]) bit makes a 1-to-0 transition, a logical 1 is transferred into the receiver shift register. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1).	
9	TXACT	R	Transmitter Active This bit is set when the transmitter begins sending data on the serial output. This bit is cleared when: 1. SEND is cleared, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1). Writes ignored.	

Table 44 – DUP11 TX Control/Status Register (TXCSR) - IO Address 760304			
Bit(s)	**Mnemonic**	**R/W**	**Description**
8	INIT	W	Device Reset aka Controller Clear. Asserting this bit does the following: 1. Resets Data Set Change A (RXCSR[DSCHGA]), and 2. Resets Receiver Interrupt Enable (RXCSR[RXIE]), and 3. Resets Secondary Transmitted Data (RXCSR[SECTX]), and 4. Resets Request To Send (RXCSR[RTS]), and 5. Resets Data Terminal Ready (RXCSR[DTR]), and 6. Resets Data Set Change B (RXCSR[DSCHGB]), and 7. Resets Maintenance Clock Out (TXCSR[MCO]), and 8. Resets Maintenance Select (TXCSR[MSEL]), and 9. Resets Maintenance Data In (TXCSR[MDI]), and 10. Sets Transmitter Done (TXCSR[TXDONE]), and 11. Resets Transmitter Interrupt Enable (TXCSR[TXIE]), and 12. Resets Send (TXCSR[SEND]), and 13. Resets Transmit Abort (TXDBUF[ABRT]), and 14. Resets Transmit End of Message (TXDBUF[TXEOM]), and 15. Resets Transmit Start of Message (TXDBUF[TXSOM]), and 16. Resets Transmitter Data Buffer (TXDBUF[TXDBUF]). This bit is always read as zero.
7	TXDONE	R	Transmitter Done This bit reports the state of the transmitter. This bit is set by: 1. A character is transferred from Transmitter Data Buffer (TXDBUF[TXDBUF]) to the transmit shift register, or 2. In SDLC Mode and a FLAG character has completed sending with the SEND bit asserted, or 3. In SDLC Mode and an ABORT character has completed sending with the SEND bit asserted, or 4. Asserting Controller Clear (TXCSR[INIT], or 5. Asserting IO Bridge Clear (UBACSR[INI]). This bit is cleared by: 1. Writing to the Transmitter Data Buffer (TXDBUF[TXDBUF]) or 2. Cleared when TXACT transitions to inactive. This occurs at the end of the data transmission. Writes ignored.
6	TXIE	R/W	Transmitter Interrupt Enable This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1).

Table 44 – DUP11 TX Control/Status Register (TXCSR) - IO Address 760304			
Bit(s)	**Mnemonic**	**R/W**	**Description**
5	-	R	Reserved. Writes ignored. Always read as zero.
4	SEND	R/W	Send This bit should remain set until the TXEOM bit is loaded into the TXDBUF. If this bit is cleared at any other time, the current character is finished and the transmitter output goes to a mark hold state. If SEND is cleared while TXEOM is still asserted, the current character being transmitted is completed. Following this character, and depending on the protocol being used, any necessary CRC and/ or control characters are transmitted. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1).
3	HDX	R/W	Half-Duplex Mode Not implemented. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1).
2-0	-	R	Reserved. Writes ignored. jfcl

8.2.5 DUP11 Transmitter Data Buffer (TXDBUF)

The high byte of this register contains the transmitter control and status information, plus two status bits from the RX and TXCRC registers.

The low byte provides the transmitter data buffer that contains the information to be transmitted.

The TXDBUF is read/write and is byte addressable.

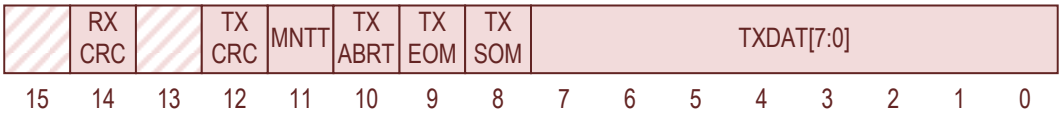


Figure 74 – DUP11 Transmitter Data Buffer (TXDBUF)

Table 45 – DUP11 TX Data Buffer (TXDBUF) - IO Address 760306			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	-	R	Reserved. Writes ignored. Always read as zero.
14	RXCRC	R	Receiver CRC Register LSB. This bit contains the LSB of the Receiver CRC registers when configured Internal Maintenance Mode (TXCSR[MSEL] = 3). In all other modes, this register is zero.
13	-	R	Reserved. Writes ignored. Always read as zero.
12	TXCRC	R	Transmitter CRC Register LSB. This bit contains the LSB of the Transmitter CRC registers when configured Internal Maintenance Mode (TXCSR[MSEL] = 3). In all other modes, this register is zero.
11	MNTT	R	Maintenance Timer This bit is used as a timing reference for the diagnostics. The diagnostic appears to use this signal as a delay reference for the modem control signals. For example, the Ring Indication Line has a 10 millisecond debounce period. The diagnostic changes the state of the line and waits, using this timer, before checking the input state.

Table 45 – DUP11 TX Data Buffer (TXDBUF) - IO Address 760306			
Bit(s)	**Mnemonic**	**R/W**	**Description**
10	TXABRT	R/W	Transmit Abort When this bit is asserted, an Abort Sequence is transmitted after the current character sent, if a character is in-process. The SEND bit should be asserted when the Abort Sequence is transmitted. TXDONE is set at the end of the Abort Sequence. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1). Note: In the DUP11 implementation, an Abort Sequence is a sequence of more 8 contiguous ones.
9	TXEOM	R/W	Transmit End of Message Asserting this bit causes the contents of the CRC register to be transmitted after the current character. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1).
8	TXSOM	R/W	Transmit Start of Message. In SDLC Mode, asserting this bit causes a Flag Character to be sent. If the TXDBUF[TXSOM] bit is still asserted at the end of the character, another Flag Character will be sent. At the end of each flag character, the CRC register is reset. The Flag Character is generated by the USRT transmitter – it does not come from the TXDBUF[TXDAT] register. In DDCMP Mode, asserting this bit causes the CRC register to be initialized at the end of each character that is transmitted. This is the only difference between Data and SYN characters as they both come from the TXDBUF[TXDAT] register. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1).

Table 45 – DUP11 TX Data Buffer (TXDBUF) - IO Address 760306			
Bit(s)	**Mnemonic**	**R/W**	**Description**
7-0	TXDAT	R/W	Transmit Data These bits are set by writing a 1. These bits are cleared by: 1. Writing a 0, or 2. Controller Clear (TXCSR[INIT] = 1), or 3. IO Bridge Clear (UBACSR[INI] = 1).

8.3 DUP11 Interrupts

8.3.1 Transmitter Interrupt

Transmitter interrupt is generated when TXIE transitions from negated to asserted when TXDONE is asserted.

8.3.2 Receiver Interrupt

Receiver interrupt generated when RXIE is enabled and RDDONE is asserted.

9 DZ11 Asynchronous Multiplexer

The DZ11 is an asynchronous multiplexer that provides an interface between 8 asynchronous serial lines and the KS10 IO Bus.

The KS10 FPGA can support multiple DZ11 devices. The configuration parameters of these devices are summarized below in Table 46.

Table 46 – DZ11 Configuration

Device	UBA	Interrupt	Interrupt Vector	Base Address
DZ11 #1	UBA3	5	RX: 000340 TX: 000344	760010
DZ11 #2	UBA3	5	RX: 000350 TX: 000354	760020
DZ11 #3	UBA3	5	RX: 000360 TX: 000364	760030
DZ11 #4	UBA3	5	RX: 000370 TX: 000374	760040

The DZ11 entity is fully parameterized for all of these configurations although the present implementation only instantiates DZ11 #1.

A block diagram of the DZ11 is illustrated below in Figure 75.

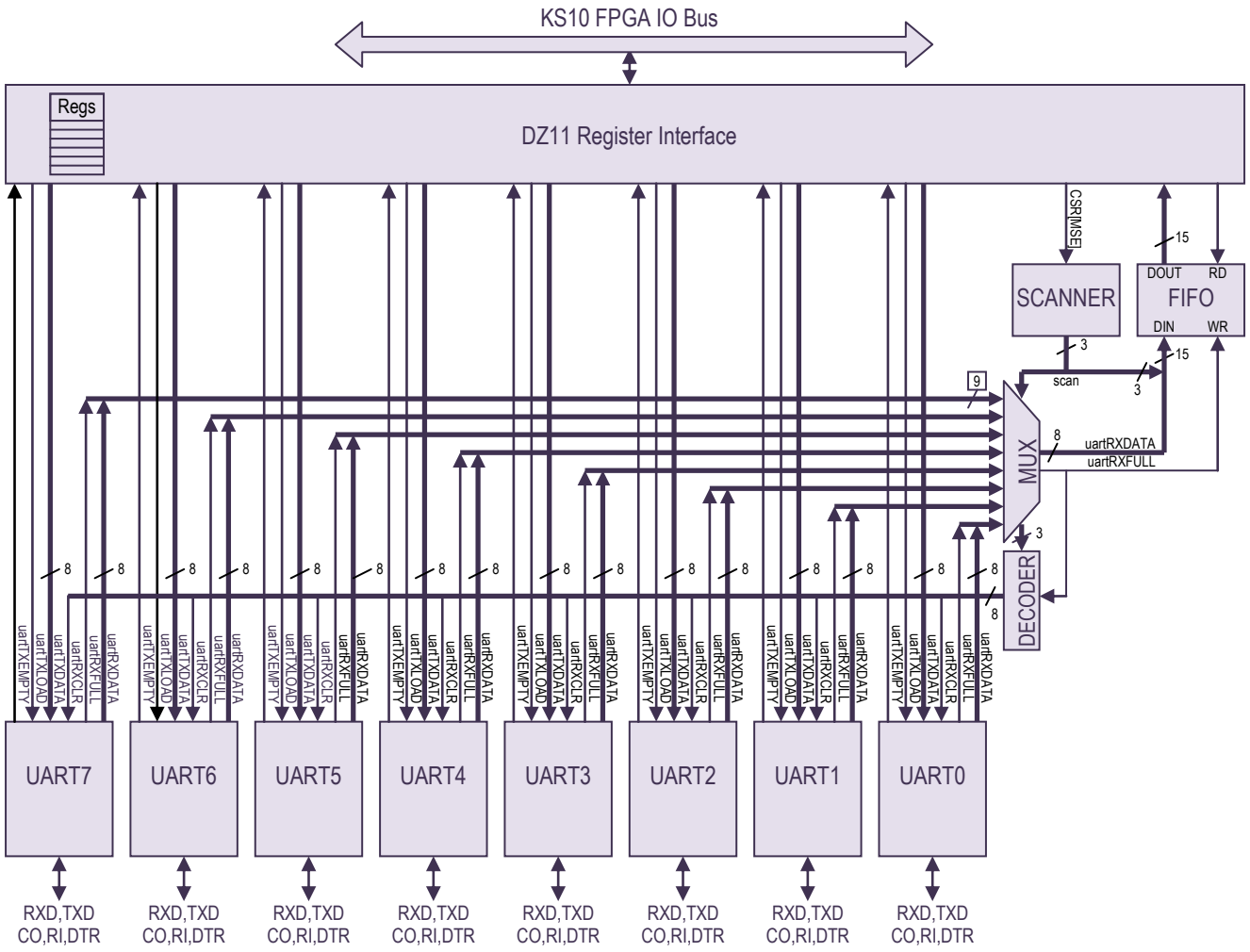


Figure 75 – DZ11 Block Diagram

9.1 DZ11 Registers

9.1.1 DZ11 Control and Status Register (CSR)

The DZ11 Control and Status Register can be accessed as bytes or words.

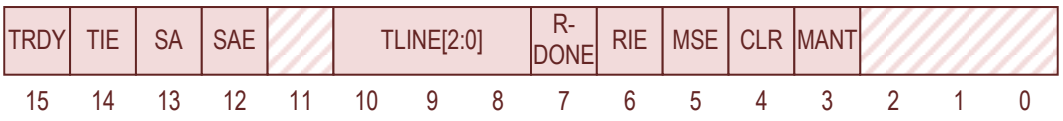


Figure 76 – DZ11 Control and Status Register (CSR)

Table 47 – DZ11 Control and Status Register (CSR) – IO Address 760010			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	TRDY	R	Transmitter Ready. This bit is asserted when a line with the LINE ENB bit asserted has an empty transmit buffer. This bit is cleared by: 1. Loading Data into the Transmitter Buffer (TDR), or 2. Negating the line (TCR[LIN]) associated with CSR[TLINE], or 3. Asserting Controller Clear (CSR[CLR]), or 4. Asserting IO Bridge Clear (UBACSR[INI]). When this bit is asserted and transmitter interrupts are enabled (CSR[TIE] = 1), a transmitter interrupt is generated.
14	TIE	R/W	Transmitter Interrupt Enable. Asserting this bit enables transmitter interrupts. This bit is set by writing a one. This bit is cleared by: 1. Writing a zero, or 2. Asserting Controller Clear (CSR[CLR]), or 3. Asserting IO Bridge Clear (UBACSR[INI]).
13	SA	R	Silo Alarm. This bit is asserted when the 16th character has been written into the receive FIFO. This bit is cleared by: 1. Reading the receiver buffer register (RBUF), or 2. Asserting Controller Clear (CSR[CLR]), or 3. Asserting IO Bridge Clear (UBACSR[INI]). When this bit is asserted and the Silo Alarm is enabled (CSR[SAE] = 1) and receiver interrupts are enabled (CSR[RIE] = 1), a receiver interrupt is generated. Note: The Silo Alarm (SA) is unrelated to the FIFO depth. The SA just independently counts characters. When the SA flag is asserted, the FIFO must be emptied because the SA flag will not be asserted again until another 16 characters are written to the FIFO. If you don't empty the FIFO, the SA will be incorrect.

Table 47 – DZ11 Control and Status Register (CSR) – IO Address 760010			
Bit(s)	**Mnemonic**	**R/W**	**Description**
12	SAE	R/W	Silo Alarm Enable. This bit enables the SILO Alarm (SA) interrupt. This bit is set by writing a one. This bit is cleared by: 1. Writing a zero, or 2. Asserting Controller Clear (CSR[CLR]), or 3. Asserting IO Bridge Clear (UBACSR[INI]).
11	Unused	R	Writes ignored. Read as zero.
10-8	TLINE[2:0]	R	Transmit Line. When a transmitter is ready (CSR[TRDY] = 1), these bits indicate which of the transmitters can accept a character to transmit.
7	RDONE	R	Receiver Done. This bit is asserted when the Receiver FIFO is not empty. When this bit is asserted and the SILO Alarm Interrupt is disabled (CSR[SAE] = 0) and receiver interrupts are enabled (CSR[RIE] = 1), a receiver interrupt is generated.
6	RIE	R/W	Receiver Interrupt Enable. This bit enables receiver interrupts. This bit is set by writing a one. This bit is cleared by: 1. Writing a zero, or 2. Asserting Controller Clear (CSR[CLR]), or 3. Asserting IO Bridge Clear (UBACSR[INI]).
5	MSE	R/W	Master Scan Enable. This bit enables the receiver, transmitter, and FIFO. This bit is set by writing a one. This bit is cleared by: 1. Writing a zero, or 2. Asserting Controller Clear (CSR[CLR]), or 3. Asserting IO Bridge Clear (UBACSR[INI]).
4	CLR	R/W	Controller Clear. When asserted, this bit clears the Receiver FIFO, all UARTS, and the CSR. This triggers a 15µS one-shot in the KS10.

Table 47 – DZ11 Control and Status Register (CSR) – IO Address 760010			
Bit(s)	**Mnemonic**	**R/W**	**Description**
3	MAINT	R/W	Maintenance Mode. When this bit is asserted, the transmitted serial data is looped back to the receiver. This bit is set by writing a one. This bit is cleared by: 1. Writing a zero, or 2. Asserting Controller Clear (CSR[CLR]), or 3. Asserting IO Bridge Clear (UBACSR[INI]).
2-0	-	R	Reserved Writes ignored. Read as zero.

9.1.2 DZ11 Receiver Buffer Register (RBUF)

The DZ11 Receiver Buffer Register is read/only and should only be accessed as a word. The RBUF register is essentially the interface to the receiver data FIFO. All of the bits in this register (except DVAL) are popped from the receiver data FIFO.

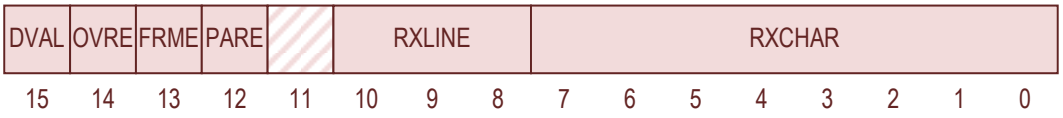


Figure 77 – DZ11 Receiver Buffer Register (RBUF)

Table 48 – DZ11 Receiver Buffer Register (RBUF) – IO Address 760012			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	DVAL	R	Data Valid. This bit is asserted when there is valid data at the FIFO output. I.e., the FIFO is not empty.
14	OVRE	R	Overrun Error. This bit is asserted when a received character has been replaced by this received character because the FIFO was full. The FIFO overwrites the last character when the FIFO is full and a character is written.
13	FRME	R	Framing Error. Asserted when any stop bits are missing.

Table 48 – DZ11 Receiver Buffer Register (RBUF) – IO Address 760012			
Bit(s)	**Mnemonic**	**R/W**	**Description**
12	PARE	R	Parity Error. Asserted when parity is enabled and the received parity is incorrect.
11	-	R	Reserved Always read as zero.
10-8	RXLINE	R	Received line. This field indicates the line number of the received character.
7-0	RXCHAR	R	Received character.

9.1.3 DZ11 Line Parameter Register (LPR)

The DZ11 Line Parameter Register is write/only and should only be accessed as a word.

A lot of the design is constrained by the COM5016 baud rate generator and the AY-5-1012 UART chip that was selected for the DZ11.

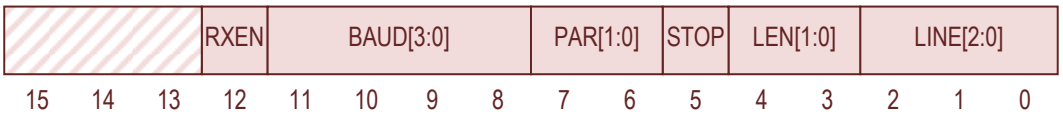


Figure 78 – DZ11 Line Parameter Register (LPR)

Table 49 – DZ11 Line Parameter Register (LPR) – IO Address 760012			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-13	Not Used	W	Unused. Writes are ignored.
12	RXEN	W	Receiver clock enable. This enables the UART receiver clock selected by LPR[LINE]. This bit is set by writing a one. This bit is cleared by: 1. Writing a zero, or 2. Asserting Controller Clear (CSR[CLR]), or 3. Asserting IO Bridge Clear (UBACSR[INI]).

Table 49 – DZ11 Line Parameter Register (LPR) – IO Address 760012				
Bit(s)	**Mnemonic**	**R/W**	**Description**	
11-8	BAUD[3:0]	W	Baud rate selection.	
			BAUD[3:0]	**BAUD RATE**
			0000	50
			0001	75
			0010	110
			0011	134.5
			0100	150
			0101	300
			0110	600
			0111	1200
			1000	1800
			1001	2000
			1010	2400
			1011	3600
			1100	4800
			1101	7200
			1110	9600
			1111	19200 Note: The DZ11 describes this value as unused even though the COM5016 baudrate generator provides for 19200 baud. The DSDZA documents this as 19200 baud also.
7-6	PAR[1:0]	W	Parity Type.	
			PAR[1:0]	**Parity**
			00	No parity
			01	Odd parity
			10	No parity
			11	Even parity
5	STOP	W	Number of stop bits.	
			STOP	**Number of Stop Bits**
			0	1 Stop bit
			1	2 Stop bits
4-3	LEN[1:0]	W	Character length.	
			LEN[1:0]	**Length**
			00	5 bits (Baudot)
			01	6 bits
			10	7 bits
			11	8 bits
2-0	LINE[2:0]	W	Line number.	

9.1.4 DZ11 Transmit Control Register (TCR)

The upper 8-bits of this register control the DTR signals for each line. The lower 8-bit enable the UART transmitters for each line.

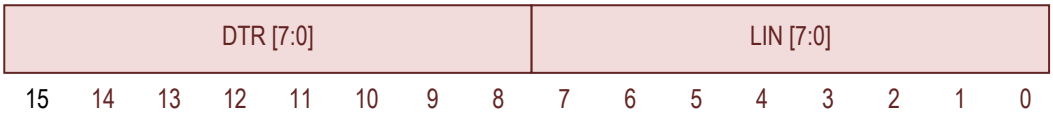


Figure 79 – DZ11 Transmit Control Register TCR)

Table 50 – DZ11 Transmit Control Register (TCR) – IO Address 760014			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-8	DTR[7:0]	R/W	Data Terminal Ready. These 8 registers control the state of the Data Terminal Ready (DTR) output signal for each of the 8 terminals. The CSR[CLR] bit does not clear the DTR signals. Note: the DTR signals are implemented but are terminated (open) at the FPGA top level. The FPGA interface does not include these pins.
7-0	LIN[7:0]	R/W	Line Enable. These 8 registers control whether or not the an empty transmitter buffer will trigger a transmitter interrupt or set the CSR[TRDY] bit.

9.1.5 DZ11 Modem Status Register (MSR)

This register accepts CO and RI inputs for each of the receiver lines.

This register is read-only and the address is shared with the Transmit Data Register. The MSR can be accessed as either bytes or words.

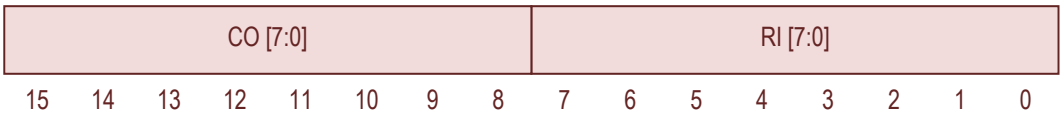


Table 51 – DZ11 Modem Status Register (TDR) – IO Address 760016			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-8	CO[7:0]	R	Carrier Detect. These bits reflect the state of the 8 Carrier Detect (CO) signals. Note: these signals are controlled by the DZ11 Console Control Register but are otherwise unused.
7-0	RI[7:0]	R	Ring Indication. These bits reflect the state of the 8 Ring Indicator (RI) signals. Note: these signals are controlled by the DZ11 Console Control Register but are otherwise unused.

9.1.6 DZ11 Transmit Data Register (TDR)

The transmitter data register receives data to be transmitted on a line.

This register is write-only and the address is shared with the Modem Status Register. The TDR can be accessed as either bytes or words.

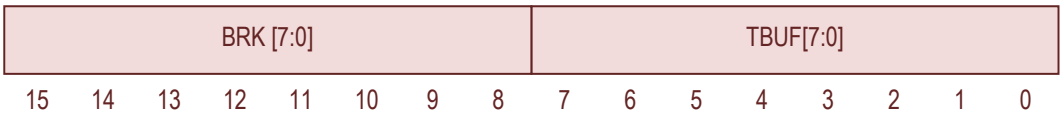


Table 52 – DZ11 Transmit Data Register (TDR) – IO Address 760016			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-8	BRK[7:0]	W	Break. Not implemented. Writes have no effect.
7-0	TBUF[7:0]	W	Transmitter Buffer. Data written to the port is written to the UART transmitter buffer that is indicated by CSR[TLINE] register bits.

9.2 DZ11 Interrupts

This section describes the DZ11 Interrupts.

9.2.1 DZ11 Transmitter Interrupt

A transmitter interrupt occurs when the transmitter interrupt is enabled (CSR[TIE] = 1) and the transmitter ready signal (CSR[TRDY]) transitions to asserted.

9.2.2 DZ11 Receiver Interupt

9.3 Hardware Description

The following description can be reverse engineered by examining the schematics.

9.3.1 The Transmitter Scanner

The transmit UARTs are scanned along with the receiver UARTS. When an empty transmitter UART is found, the CSR[TRDY] bit is asserted and the scanner contents are latched into the CSR[TLINE] bits.

The scanning resumes when the CSR[TRDY] bit is cleared.

If the transmitter scanner is latched on an input line (CSR[TRDY] asserted) and that line become disabled (TCR[LIN(I)] = 0), the TRDY bit is cleared and the scanning resumes.

9.3.2 The Baud Rate Generator

The DEC DZ11 used a Standard Microsystems COM5016 baud rate generator with a special 5.0688 MHz crystal clock source. The FPGA implementation of the DZ11 uses the standard clock source and implements the 16x baud rate clock using a Fractional-N divider. This provides similar accuracies without the special clock source.

9.3.2.1The Fractional-N Divider

The Fractional-N divider was chosen for the baud rate generator so that standard 50 MHz crystal could be used for the clock generator.

A standard divider can only divide by integers. This makes accurate baud rate timing difficult at times. The Fractional-N divider has no such limitation. The Fractional-N output is always aligned to the nearest clock cycle and timing errors do not accumulate – an accumulator maintains the fraction part of the clock signal.

The increment value for the accumulator is given by a 16x32 ROM.

The output of the baud rate generator module is a clock enable signal at 16x the selected baud rate.

A block diagram of the Fractional-N divider is illustrated below in Figure 80.

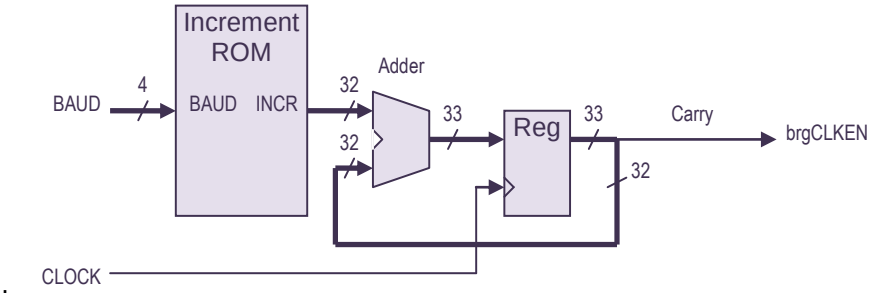


Figure 80 – Fractional-N Divider Block Diagram

The equation for the contents of the INCR ROM is given below in Equation 1.

$$INCR = 2^{32} * int\left(\frac{16 * Baud\ Rate}{Clock\ Freq} + 0.5\right)$$ Equation 1

10 KMC11 Co-Processor

The KMC11 is a communications co-processor that is used to off-load DUP11 data movement tasks from the KS10 to the KMC11.

The KS10 supports only a single KMC11; however, a single KMC11 can manage several DUP11 Synchronous Serial Interfaces.

The KMC11 configuration is illustrated below:

Table 53 – KMC11 Configuration				
Device	**UBA**	**Interrupt**	**Interrupt Vector**	**Base Address**
KMC11	UBA3	5	000540 000544	760540

10.1 KMC11 Registers

A summary of DUP11 registers is shown below.

IO Addr	Register Name	Access	Register Description
760540	CSR0 CSR1	Byte R/W	Control and Status Register #0 and #1
760542	CSR2 CSR3	Byte R/W	Control and Status Register #2 and #3
760544	CSR4 CSR5	Byte R/W	Control and Status Register #4 and #5
760546	CSR6 CSR7	Byte R/W	Control and Status Register #6 and #7

10.1.1 KMC11 Control and Status Registers (CSRs)

The KMC11 Control and Status Registers are general purpose registers used for communications between the KS10 and the KMC11. These registers are implemented using multi-port memory so that they can be accessed by the KS10 and by the KMC11.

These registers have no pre-defined purpose and can be used for anything.

These registers are shadowed by a set of Maintenance Registers: specifically the Maintenance Register, the Maintenance Address Register, and the Maintenance Instruction Register. In general, these register are used for loading and executing KMC11 microcode.

These registers are read/write and are byte addressable.

These registers are illustrated below in Figure 81.

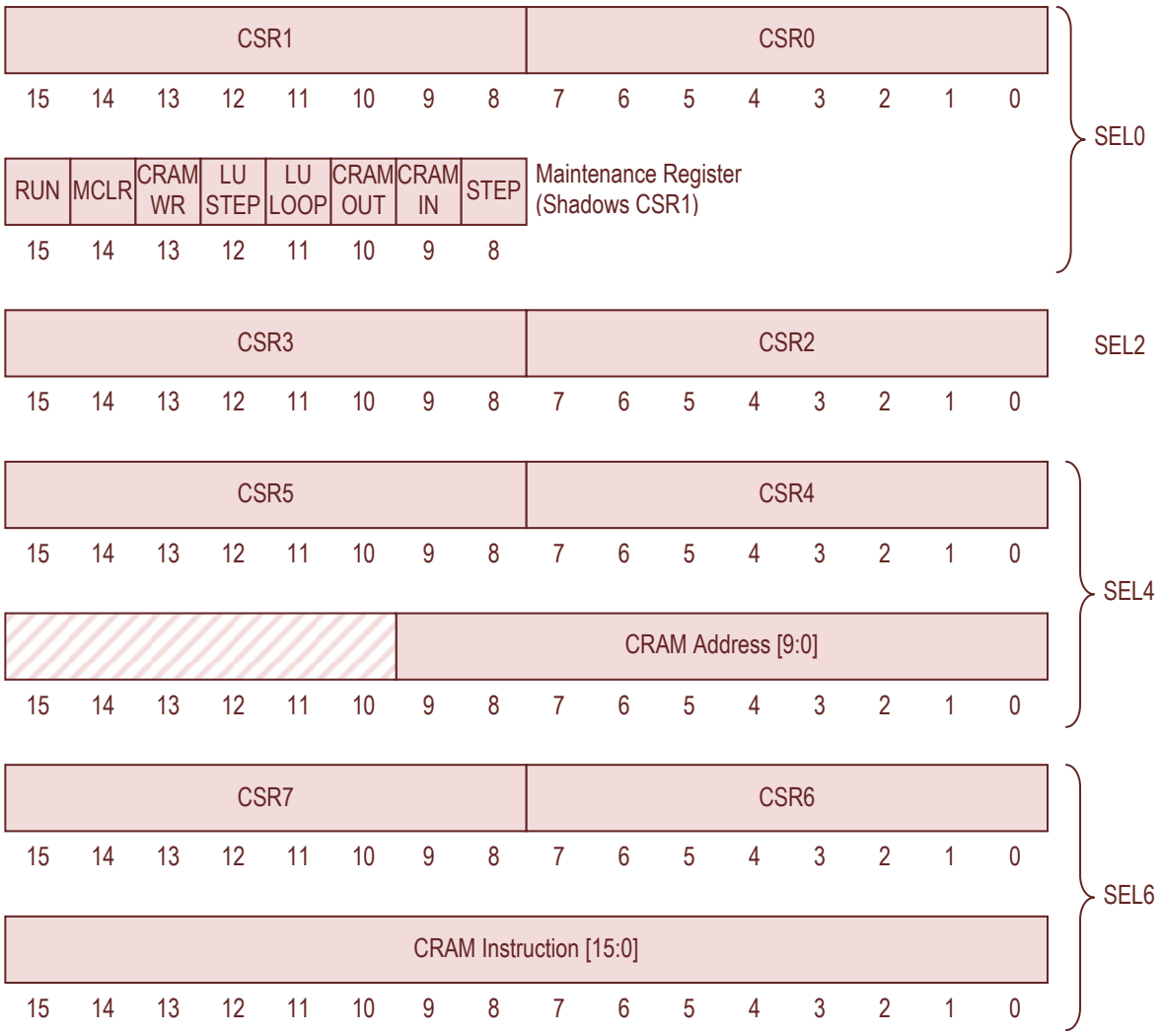


Figure 81 – KMC11 Control and Status Registers

10.1.2 KMC11 Maintenance Register

The Maintenance Register is used for loading and controlling the operation of the KMC11.

The Maintenance Register shadows the CSR1 register which exists in multi-port memory. Writes to that Unibus IO address go to both the Maintenance Register and to the CSR1 register in multi-port memory. That implies that the KMC11 can observe the state of Maintenance Register. The KMC11 can even modify the value of the value of CSR1 in the multi-port memory, but it cannot alter the state of the Maintenance Register.

KMC11 Maintenance Register (CSR1 Shadow)			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	RUN	W	Run When asserted, this causes the KMC11 to execute instructions. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]).
14	MCLR	W	Master Clear Asserting this bit does the following: 1. Resets all hardware logic related to the maintenance bits in CSR1, and 2. Resets all flip-flops related to the NPR register with the 3. exception of the flip-flop associated with NPR REQ, and 4. Resets all flip-flops related to the µPMISC register, and 5. Resets the ALU Z-bit, and 6. Resets the ALU C-bit, and 7. Resets the BRG, and 8. Resets the MAR, and 9. Resets the PC, and 10. Resets the Maintenance Address Register, and 11. Resets the Maintenance Instruction Register, and 12. Resets all timing signals. Asserting the Master Clear does not alter the following: 1. The contents of the Multiport Memory, or 2. The Control RAM (CRAM), or 3. The Data Memory (MEM), or 4. The scratchpad memory (SPR), or 5. The flip-flop associated with NPR REQ,
13	CRAMWR	W	CRAM Write This bit is used to write microcode into the CRAM. When CRAM Write CSR1[CRAMW] is asserted and CSR1[ROMOUT] is asserted, the contents of Maintenance Instruction Register is loaded into the CRAM at the address specified by Maintenance Address Register.
12	LUSTEP	W	Line Unit Step Not implemented
11	LULOOP	W	Line Unit Loop Not implemented.

KMC11 Maintenance Register (CSR1 Shadow)			
Bit(s)	**Mnemonic**	**R/W**	**Description**
10	CRAMOUT	W	CRAM Output When CRAM Output is negated: • writes to SEL 4 are to CSR4/CSR5, and • reads from SEL 4 are from CSR4/CSR5 When CRAM Output is asserted: • writes to SEL 4 are to the Maintenance Address Register, and • reads from SEL 4 are from the Maintenance Address Register. The register is cleared by IO Bridge Clear (UBASTAT[INI]) or by the KMC11 Master Clear(CSR1[MCLR]).
9	CRAMIN	W	CRAM Input When CRAM Input is negated: • writes to SEL 6 are to CSR6/CSR7, and • reads from SEL 6 are from CSR6/CSR7 When CRAM Output is asserted: • writes to SEL 6 are to the Maintenance Instruction Register, and • reads from SEL 6 are from the Maintenance Instruction Register Once the Maintenance Instruction Register is loaded, it can either be stored into the CRAM by asserting CRAM Write or executed by asserting STEP. The register is cleared by IO Bridge Clear (UBASTAT[INI]) or by the KMC11 Master Clear(CSR1[MCLR]).
8	STEP	W	Single Step Microprocessor When asserted, this bit steps the microprocessor through one instruction cycle. Note: The RUN bit should be negated before executing this control function. The register is cleared by IO Bridge Clear (UBASTAT[INI]) or by the KMC11 Master Clear(CSR1[MCLR]).

10.1.3 KMC11 Maintenance Address Register

The Maintenance Address Register is used to load microcode into the KMC11. Specifically, this register provides the address when microcode is written into the KMC11 Control RAM (CRAM).

From the IO Bus (Unibus) perspective, the Maintenance Address Register shares an IO address with CSR4 and CSR5. See description of the CRAM Output bit of the Maintenance Register.

KMC11 Maintenance Address Register			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-12	-	R	Reserved Writes ignored Always read as zero.
11-0	ADDR	R/W	Control RAM Address This field is read write. The register is cleared by IO Bridge Clear (UBASTAT[INI]) or by the KMC11 Master Clear(CSR1[MCLR]).

10.1.4 KMC11 Maintenance Instruction Register

The Maintenance Instruction Register is used to load microcode into the KMC11 or to execute a single microcode instruction.

From the IO Bus (Unibus) perspective, the Maintenance Instruction Register shares an IO address with CSR6 and CSR7. See description of the CRAM Input bit of the Maintenance Register.

KMC11 Maintenance Instruction Register			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-0	INST	R/W	Instruction This field is read write. The register is cleared by IO Bridge Clear (UBASTAT[INI]) or by the KMC11 Master Clear(CSR1[MCLR]).

10.2 KMC11 Microprocessor Registers

This section details the registers internal to the KMC11.

10.2.1 KMC11 MISC Register (MISC)

The MISC Register is an 8-bit register in the KMC11 Microprocessor that provides some basic IO. It is not addressable from the KS10 in any way.

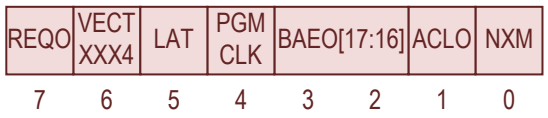


Figure 76 – KMC11 MISC Register

KMC11 MISC Register			
Bit(s)	**Mnemonic**	**R/W**	**Description**
7	IRQO	R/W	Interrupt Request Out Asserting this bit creates an interrupt request to the KS10. This bit is set by writing a 1. This bit is cleared when the interrupt is acknowledged.
6	VECTXXX4	R/W	Interrupt Vector This selects between the two interrupt vectors. When this bit is asserted, the interrupt vector will be 000544 otherwise the interrupt vector will be 000540. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]).
5	LAT	R/W	Latch This is just a bit that has no particular purpose This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]).
4	PGMCLK	R/W	Retriggerable Timer This bit triggers a 50 microsecond re-triggerable timer. When triggered, the PGMCLK pin negates and remains negated for 50 microseconds. If the PGMCLK pin is triggered while the timer is active, the timer reset to 50 microseconds again. This bit is set when: 1. The Timer expires, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]). This bit is cleared by asserting this bit when the timer is inactive.

KMC11 MISC Register			
Bit(s)	**Mnemonic**	**R/W**	**Description**
3-2	BAEO[17:16]	R/W	Base Address Extension for DMA Out - bits 17:16 These bits provide the upper two bits of the DMA address during DMA read transactions. These bits are set by writing ones. These bits is cleared by: 1. Writing zeros, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]).
1	ACLO	R/W	ACLO When asserted this forces a Power Fail condition to the KS10 CPU. This is not implemented. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]).
0	NXM	R	Non-existent Memory Although the KS10 FPGA implements all memory, a UBA Page Fail can create an NXM response. This bit is set by: 1. Writing a 1, or 2. When the KMC11 initiates a DMA request than is not acknowledged within 20 microseconds. This bit is cleared by: 1. Writing a 0, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]).

10.2.2 KMC11 NPR Control Register (NPRC)

The NPR Control Register is an 8-bit register in the KMC11 Microprocessor that provides some basic IO. It is not addressable from the KS10 in any way.

Figure 76 – KMC11 NPR Control Register

KMC11 NPR Control Register			
Bit(s)	**Mnemonic**	**R/W**	**Description**
7	BYTEXFER	R/W	Byte Transfer Asserting this bit causes the DMA to do byte transfers; otherwise the DMA will do word transfers. This bit is set by writing a one. This bit is cleared by: 1. Writing a 0, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]).
6	MAR[10]	R	MAR[10] This is used to monitor the Memory Address Register (MAR). The MAR is 10-bits but can be auto incremented. This bit indicates when the address has wrapped around to zero. Writes ignored.
5	MAR[8]	R	MAR[8] This provides access to the Memory Address Register bit 8. Writes ignored.
4	NPRO	R/W	DMA Out When NPRO is asserted and NPRQ is asserted, the KMC11 will perform a DMA write operation. When NPRO is negated and NPRQ is asserted, the KMC11 will perform a DMA read operation. This bit is set by writing a one. This bit is cleared by: 1. Writing a 0, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]).
3-2	BAEI[17:16]	R/W	Base Address Extension for DMA In - bits 17:16 These bits provide the upper two bits of the DMA address during DMA read transactions. These bits are set by writing ones. These bits is cleared by: 1. Writing zeros, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]).

KMC11 NPR Control Register			
Bit(s)	**Mnemonic**	**R/W**	**Description**
1	NLXFER	R/W	Not Last Transfer This bit allows multiple DMA transactions to occur without renegotiating for access to the KS10 Backplane Bus. This is not implemented. This bit is set by writing a 1. This bit is cleared by: 1. Writing a 0, or 2. Asserting KMC11 Master Clear(CSR1[MCLR]), or 3. Asserting IO Bridge Clear (UBASTAT[INI]).
0	NPRRQ	R/W	DMA Request Asserting this bit will start a DMA transaction. When the DMA transaction is completed, this bit will automatically clear.

11 RH11 Massbus Disk Controller

The KS10 FPGA Disk Controller design mimics the design of the actual KS10 disk subsystem with some minor exceptions.

The KS10 FPGA can support multiple RH11 devices. The configuration parameters of these devices are summarized below in Table 54.

Table 54 – RH11 Configuration

Device	UBA	Interrupt	Interrupt Vector	Base Address
RH11 #1 (RPxx)	UBA1	6	000254	776700
RH11 #2 (TU45)	UBA3	6	000224	772440

In the current implementation, the RH11 entity is fully parameterized for these configurations. At present, the IO design does not instantiate multiple RH11 devices.

The FPGA implementation retains register-set compatibility with the original KS10 disk system. The actual implementation is split into 4 major subsections:

1. RH11 compatible Disk Controller, and
1. Multiple disk drive simulators, and
2. A Disk Completion Monitor, and
3. A Secure Digital (SD) card interface.

A block diagram of the MASSBUS Disk Controller implementation is illustrated below in Figure 82.

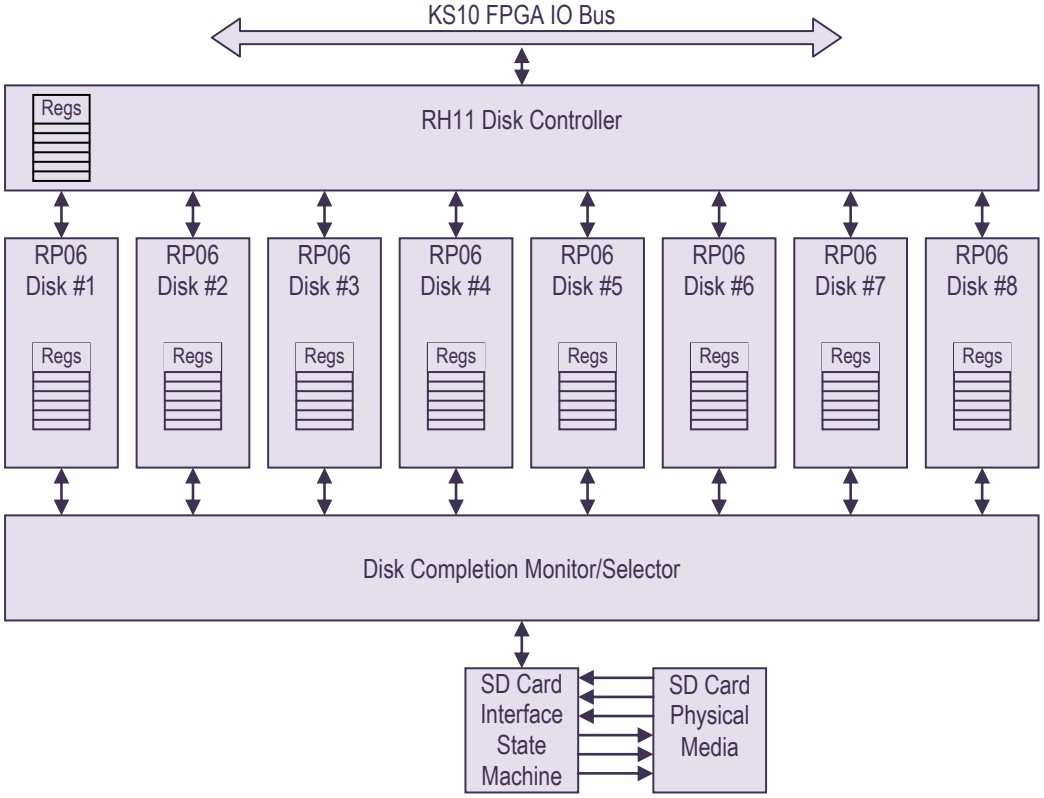


Figure 82 – KS10 FPGA Disk Subsystem Architecture

The subsections are described in the following document sections.

11.1 Definitions

11.1.1 Disk Clear Operations

The following definitions are used in the descriptions of the RH11/RP06 operation. These definitions simplify the description of the register operations.

11.1.1.1 IO Bridge Clear

The *IO Bridge Clear* signal is asserted when UBACSR[INI] is asserted

11.1.1.2 Controller Clear

The *Controller Clear* signal is asserted when RHCS2[CLR] is asserted.

11.1.1.3 Drive Clear

The *Drive Clear* signal is asserted when RHCS1 is written with RPCS1[FUN] = 04 and RPCS1[G0] = 1.

11.1.1.4 Error Clear

The *Error Clear* signal is asserted under the following conditions

1. *Write Check Data* command with GO bit set, or
2. *Write Check Data and Header* command with GO bit set, or
3. *Write Data* command with GO bit set, or
4. *Write Data and Header* command with GO bit set, or
5. *Read Data* command with GO bit set, or
6. *Read Data and Header* command with GO bit set, or
7. Write 1 to Transfer Error (RHCS1[TRE]).

11.2 RH11 Registers

The KS10 disk system exposes two different types of registers to the programmer:

1. Controller Registers that are located in the RH11 disk controller which apply to all disk drives, and
2. Device Registers that are located inside the individual disk drives and apply only to that disk drive.

In a real KS10, the Controller Registers were part of the RH11 Controller, and the Device Registers were located in the disk drives on the device side of the Massbus cabling. This hierarchy is maintained in the KS10 FPGA although the cabling is more a logical concept than a physical implementation.

The RH11 Compatible Disk Controller maintains state and reports status that is applicable to the entire disk system. For example the Controller RPCS2 register has 3 bits which select which of the 8 disks is addressed to receive commands or report status.

That controller state includes the following registers:

Table 55 – RH11 Controller Register Summary

Unibus Address	Register Name	R/W	Register Description
776700	RHCS1	R/W	Control and Status Register #1
776702	RHWC	R/W	Word Count Register
776704	RHBA	R/W	Bus Address Register
776710	RHCS2	R/W	Control and Status Register #2
776722	RHDB	R/W	Data Buffer Register

11.2.1 RH11 Control and Status #1 (RHCS1) Register

This register provides high level control and status of the disk system.

This register may be accessed as a byte or a word.

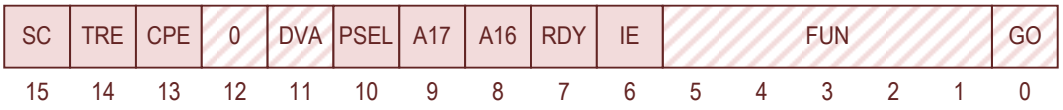


Figure 83 – RH11 Control and Status Register #1 (RHCS1)

Table 56 – RH11 Control and Status Register #1 (RHCS1) – IO Address 776700			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	SC	R	Special Conditions. This bit is set under the following conditions: 1. a Transfer Error (RHCS1[TRE]) occurs, or 2. a Massbus Control Parity Error (RHCS1[CPE]) occurs, or 3. an Attention signal from the drive (RPDS[ATA]) occurs. This bit is combinationally derived from those registers – clearing that register will clear this bit. This causes an interrupt if Ready (RHCS1[RDY]) is also asserted.
14	TRE	R/W	Transfer Error. Set when any of the following transition to active: 1. Data Late Error (RHCS2[DLT]), or 2. Write Check Error (RHCS2[WCE]), or 3. Unibus Parity Error (RHCS2[UPE]), or 4. Non-Existent Drive (RHCS2[NED]), or 5. Non-Existent Memory (RHCS2[NEM]), or 6. Program Error (RHCS2[PGE]), or 7. Missed Transfer Error (RHCS2[MXF]), or 8. Massbus Data Parity Error (RHCS2[DPE]), of 9. The addressed Composite Error (RPDS[ERR]). Note: These transitions are not detected independently. The big “OR” comes before the edge detection – not after. Cleared by writing a 1, *IO Bridge Clear, Controller Clear,* or *Error Clear.*
13	CPE	R	Control Bus Parity Error. Not implemented. Writes ignored. Always read as zero.
12	0	R	Writes ignored. Always read as zero.
11	DVA	R	Drive Available. See device.
10	PSEL	R/W	Port Select. Writes to PSEL are ignored when RHCS1[RDY] is negated. Cleared by *IO Bridge Clear* or *Controller Clear*.

Table 56 – RH11 Control and Status Register #1 (RHCS1) – IO Address 776700			
Bit(s)	**Mnemonic**	**R/W**	**Description**
9-8	A[17:16]	R/W	Address Extension Bits. See RHBA Register. Writes to these bits are ignored when RHCS1[RDY] is negated. This register is incremented as part of the RHBA register. Cleared by *IO Bridge Clear* or *Controller Clear*.
7	RDY (DONE)	R	Ready. This bit is negated when a command is executed and asserted when that command has completed. Set by *IO Bridge Clear* or *Controller Clear*.
6	IE	R/W	Interrupt enable. Writing to RHCS1[RDY] and IE simultaneously causes an immediate interrupt. Cleared by *IO Bridge Clear*, *Controller Clear*, or by an Interrupt Acknowledge bus cycle.
5-1	FUN	R/W	See device.
0	GO	R/W	See device.

11.2.2 RH11 Word Count (RHWC) Register

The program loads the RPWC Register with the two-complement number of the words to be read from or written to the disk drive.

Each time a 36-bit word is transferred, the word count is incremented by two. The disk always reads and writes full sectors. On writes, partial sectors are filled with zero.

This register may be accessed as a byte or a word.

Figure 84 – RH11 Word Count Register (RHWC)

Table 57 – RH11 Word Count Register (RHWC) – IO Address 776702			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-0	WC	R/W	Word Count The Word Count register should be zero at the end of a transfer. There is no reset mechanism.

11.2.3 RH11 Bus Address (RHBA) Register

The program loads the starting address (virtual address) of source memory (for writes) or the destination memory (for reads). The LSB is wired to zero, therefore the address is always even – supporting word addressing. See Figure 66. Each time a 36-bit word is transferred, the address is incremented by two (bit 1 is incremented). If RHCS2[BAI] is asserted, the address increment is elided.

The RH11 supports a magic-mode whereby the bus address can decrement - supporting a 'reverse write-check' and a 'reverse read' operation. The documents imply that this was never supported and this is not implemented.

This register may be accessed as a byte or a word.

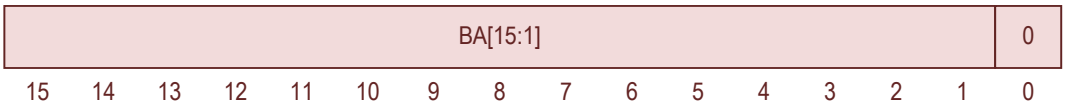


Figure 85 – RH11 Bus Address Register (RPBA)

Table 58 – RH11 Bus Address Register (RHBA) – IO Address 776704			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-1	BA	R/W	Bus Address. See also RHCS1[A17:A16] Cleared by *IO Bridge Clear* or *Controller Clear*. When the UBA Page is configured for Fast Transfer Mode, the Bus Address increments by 2 (BA register increments by 4). Otherwise the Bus Address increments by 1 (BA register increments by 2).
0	BA	R	Bus Address (Bit 0). Writes ignored. Always read as zero.

11.2.4 RH11 Control and Status #2 (RHCS2) Register

This register indicates the status of the controller.

This register may be written as a byte or a word.

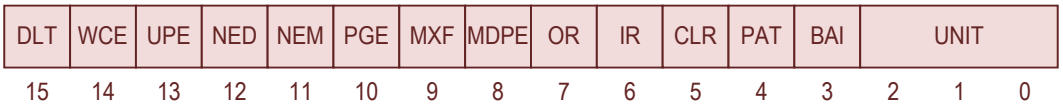


Figure 86 – RH11 Control and Status Register #2 (RHCS2)

Table 59 – RH11 Control and Status Register #2 (RHCS2) – IO Address 776710			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	DLT	R	Device Late. Set under the following conditions: 1. Set by Read Command or Write Check Command with a full FIFO. This can be simulated by storing 66 words into the FIFO via writes to the rhDB register. 2. Set by Write Command with an empty FIFO or by reading rhDB with an empty FIFO. Otherwise the DLT is not implemented as far as disk operation is concerned. Disk data bypasses the FIFO. Cleared by *IO Bridge Clear, Controller Clear*, or *Error Clear*.
14	WCE	R	Write Check Error. Set by the Write Check Command when the data read from disk does not match the data read from memory. Cleared by *IO Bridge Clear, Controller Clear*, or *Error Clear*. Note: The manual states that “The RMBA will contain the address plus two of the failing ''word in memory and the RMDB will contain the failing word from the disk.”. The RMDB operation is not implemented.
13	UPE	R/W	Unibus Parity Error. Does nothing. Set by writing 1. Cleared by *IO Bridge Clear, Controller Clear*, or *Error Clear*.
12	NED	R	Non-existent Drive. Set when reading or writing to a drive that is not present. Cleared by *IO Bridge Clear, Controller Clear*, or *Error Clear*.
11	NEM	R	Non-existent Memory. Set on non-existent memory access. Writes ignored. Cleared by *IO Bridge Clear, Controller Clear*, or *Error Clear*.
10	PGE	R	Program Error. Set by executing a command when not ready. I.e., asserting RHCS1[GO] with RHCS1[RDY] negated. Cleared by *IO Bridge Clear, Controller Clear*, or writing 1 to Transfer Error (RHCS1[TRE]).
9	MXF	R/W	Missed Transfer. Does nothing. Set by writing 1. Cleared by *IO Bridge Clear, Controller Clear*, or *Error Clear*.

Table 59 – RH11 Control and Status Register #2 (RHCS2) – IO Address 776710			
Bit(s)	**Mnemonic**	**R/W**	**Description**
8	DPE	R	Data Parity Error. Writes ignored. Always read as 0. Cleared by *IO Bridge Clear, Controller Clear*, or *Error Clear*.
7	OR	R	Output Ready. Asserted when the data SILO is not empty. Writes ignored. Cleared by *IO Bridge Clear, Controller Clear*, or *Error Clear*.
6	IR	R	Input Ready. Asserted when the data SILO is not full. Writes ignored. Cleared by *IO Bridge Clear, Controller Clear*, or *Error Clear*.
5	CLR	W	*Controller Clear*. Always read as 0.
4	PAT	R/W	Parity Test. This simulates wrong parity on the Massbus interconnection between the RH11 and the RPxx disk drives. This is tested by the DSRPA diagnostics. Cleared by *IO Bridge Clear, or Controller Clear*.
3	BAI	R/W	Bus Address Increment Inhibit. This prevents the Bus Address from incremented when words are transferred to/from the disk. Writes to BAI are ignored when RHCS1[RDY] is negated. Cleared by *IO Bridge Clear, or Controller Clear*.
2:0	UNIT	R/W	Unit Select. Cleared by IO *Bridge Clear, or Controller Clear*.

11.2.5 RH11 Data Buffer (RHDB) Register

The RHDB register interfaces to the data SILO for read and write operations. The disk unit as implemented does not use the data SILO – disk data flows directly between memory and the SD Card; however, the data SILO operation is tested as part of the DSRPA diagnostic program. Therefore the RHDB register and data SILO is implemented as required by that diagnostic. Data can be written to the SILO and read back from the SILO but has no other effect on the disk operation.

The Input Ready flag of the Control and Status #2 (RHCS2[IR]) is asserted when the data SILO is not full. The 66th write to the data SILO after reset should cause the SILO to indicate that the data SILO is full.

The Output Ready flag of the Control and Status #2 (RHCS2[OR]) is asserted when the data SILO is not empty.

RHDB is a SILO input/output so I don't see how it could be byte addressable.

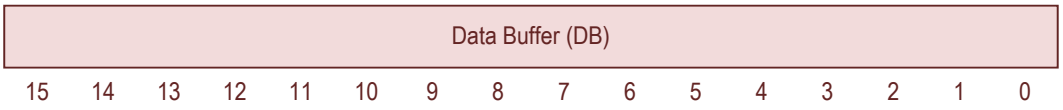


Figure 87 – RH11 Data Buffer Register (RPDB)

Table 60 – RH11 Data Buffer Register (RHDB) – IO Address 776722			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-0	DB	R/W	This is a read/write register that interfaces to the data SILO.

11.3 RH11 Interrupts

The RH11 Interrupt is a strange combination of edge-triggered and level-triggered conditions.

An interrupt flip-flop can be set under the following conditions:

1. When controller transitions to ready (RHCSR1[RDY]) with interrupts enabled (RHCSR[IE]), or
2. When the Control Register #1 (RHCS1) is written and both Interrupt Enable (IE - bit 6) and Ready (RDY - bit 7) asserted.

The interrupt flip-flop is cleared on *IO Bridge Clear, Controller Clear,* or an Interrupt Acknowledge bus cycle that addresses the RH11.

An interrupt request is generated under the following conditions:

1. The interrupt flip-flop is set, or
2. Special Conditions (RHCSR1[SC]), Ready (RHCS1[RDY]) are set.

12 RP06/07 Disk Simulator

Each of the individual disk drives maintains its own state. In this context, that device state includes everything that would normally be associated with the physical disk drive. That device state includes the following registers:

Table 61 – RPxx Device Registers

Unibus Address	RP Register Name	RM Register Name	R/W	Register Description
776700	RPCS1	RMCS1	R/W	Control and Status Register #1
776706	RPDA	RMDA	R/W	Disk Address Register
776712	RPDS	RMDS	R	Drive Status Register
776714	RPER1	RMER1	R/W	Error Register #1 Register
776716	RPAS	RMAS	R/W	Attention Summary Register
776720	RPLA	RMLA	R	Look Ahead Register
776724	RPMR	RMMR	R/W	Maintenance Register
776726	RPDT	RMDT	R	Drive Type Register
776730	RPSN	RMSN	R	Serial Number Register
776732	RPOF	RMOF	R/W	Offset Register
776734	RPDC	RMDC	R/W	Desired Cylinder Register
776736	RPCC		R	Current Cylinder
		RMHR	R	Holding Register
776740	RPER2		R/W	Error Register #2
		RMMR2	R	Maintenance Register #2
776742	RPER3		R/W	Error Register #3
		RMER2	R/W	Error Register #2
776744	RPEC1	RMEC1	R	ECC Position Register
776746	RPEC2	RMEC2	R	ECC Pattern Register

Each device maintains its own set of registers.

The disk simulator does not actually read or write data. The disk simulator strictly simulates the physical operation (timing) of a disk drive. The disk simulator can simulate rotational latency, and seek timing. The simulator maintains a notion of the current ‘head position’ and will simulate a delay that would appropriate for a disk drive as the heads are moved to different tracks or sectors based on the command

inputs. This can be accomplished with varying degrees of precision: it goes without saying that the Secure Digital (SD) disk chip has zero seek delay and zero rotational latency. This is done strictly for compatibility with the original disk systems. Some experimentation will be required to determine if this simulation fidelity is required, or not.

When the disk simulator receives a function command, the register parameters that define the disk address such as sectors, cylinders, head, are checked for validity.

Next, the disk simulator waits a period of time, as described above, before requesting exclusive access to the SD Card.

Lastly, the disk simulator calculates a 32-bit Secure Digital (SD) Linear Sector Address based on the drive address parameters (Cylinder, Head, and Sector) described above. Each of the disk simulators is allocated a sector address range on the SD card for its exclusive use.

There are some minor differences between RMxx and RPxx style disks. If at some time in the future, both types of disks need to be implemented, this is a relatively minor issue.

The MASSBUS Register addresses are summarized below in Table 62. This is provided for reference only. In this implementation, the MASSBUS registers are decoded directly from the IO Bus address.

Table 62 – Massbus Register Address Cross Reference

Reg # (decimal)	Reg # (octal)	RP Register Name	RM Register Name	Compatibility Comment
0	0	RPCS1	RMCS1	
1	1	RPDS	RMDS	
2	2	RPER1	RMER1	
3	3	RPMR	RMMR1	
4	4	RPAS	RMAS	
5	5	RPDA	RMDA	
6	6	RPDT	RMDT	
7	7	RPLA	RMLA	
8	10	RPSN	RMSN	
9	11	RPOF	RMOF	
10	12	RPDC	RMDC	
11	13	RPCC	RMHR	RMHR is read-write whereas RPCC is read-only.

Table 62 – Massbus Register Address Cross Reference				
Reg # (decimal)	**Reg # (octal)**	**RP Register Name**	**RM Register Name**	**Compatibility Comment**
12	14	RPER2	RMMR2	RPER2 and RMER2 are compatible since neither is implemented.
13	15	RPER3	RMER2	RPER3 and RMER2 are compatible since neither is implemented. The RMER2 is read-only but maybe nobody will notice.
14	16	RPEC1	RMEC1	
15	17	RPEC2	RMEC2	

12.1 RPXX Registers

12.1.1 RP Control and Status #1 (RPCS1) Register

Some of the bits in the RPCS1 Register are implemented in the RH11 Controller and some bits are implemented in the RPxx Device.

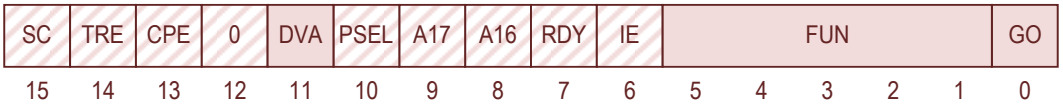


Figure 88 – RP Control and Status Register #1 (RPCS1)

Table 63 – RP Control and Status Register #1 (RPCS1) – IO Address 776700			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	SC	R	See RH controller.
14	TRE	R/W	See RH controller.
13	CPE	R	See RH controller.
12	0	R	See RH controller.
11	DVA	R	Drive available. Always read as 1
10	PSEL	R/W	See RH controller.
9	A17	R/W	See RH controller.
8	A16	R/W	See RH controller.

<table>
<tr><th colspan="4">Table 63 – RP Control and Status Register #1 (RPCS1) – IO Address 776700</th></tr>
<tr><th>Bit(s)</th><th>Mnemonic</th><th>R/W</th><th colspan="2">Description</th></tr>
<tr><td>7</td><td>RDY</td><td>R</td><td colspan="2">See RH controller.</td></tr>
<tr><td>6</td><td>IE</td><td>R/W</td><td colspan="2">See RH controller.</td></tr>
<tr><td rowspan="20">5-1</td><td rowspan="20">FUN</td><td rowspan="20">R/W</td><td colspan="2">Controller Function.
FUN is only modified by writing to this field. FUN is not cleared by IO Bridge Clear or Controller Clear.</td></tr>
<tr><td>Code (octal)</td><td>Description</td></tr>
<tr><td>00</td><td>No operation</td></tr>
<tr><td>01</td><td>Unload</td></tr>
<tr><td>02</td><td>Seek</td></tr>
<tr><td>03</td><td>Recalibrate</td></tr>
<tr><td>04</td><td>Drive Clear</td></tr>
<tr><td>05</td><td>Release</td></tr>
<tr><td>06</td><td>Offset command</td></tr>
<tr><td>07</td><td>Return to center</td></tr>
<tr><td>10</td><td>Read-in preset</td></tr>
<tr><td>11</td><td>Pack acknowledge</td></tr>
<tr><td>12-13</td><td>Illegal function(s)</td></tr>
<tr><td>14</td><td>Search command</td></tr>
<tr><td>15-23</td><td>Illegal function(s)</td></tr>
<tr><td>24</td><td>Write check data</td></tr>
<tr><td>25</td><td>Write check header and data</td></tr>
<tr><td>26-27</td><td>Illegal function(s)</td></tr>
<tr><td>30</td><td>Write data</td></tr>
<tr><td>31</td><td>Write header and data</td></tr>
<tr><td>32-33</td><td>Illegal function(s)</td></tr>
<tr><td>34</td><td>Read data</td></tr>
<tr><td>35</td><td>Read header and data</td></tr>
<tr><td>36-37</td><td>Illegal function(s)</td></tr>
</table>

Table 63 – RP Control and Status Register #1 (RPCS1) – IO Address 776700			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0	GO	R/W	Execute function specified in FUN field. Set by writing a 1 with Parity Test (RHCS2[PAT]) negated. The unit will not execute a command with incorrect parity. Other types of parity errors cannot occur by design. Cleared by writing a zero and at command completion. This field is **NOT** reset by *Controller Clear*. What about IO Bridge Clear???

12.1.2 RP Disk Address (RPDA) Register

This register addresses the sector and track of the selected unit. The disk address is incremented after the sector has been transferred to the controller.

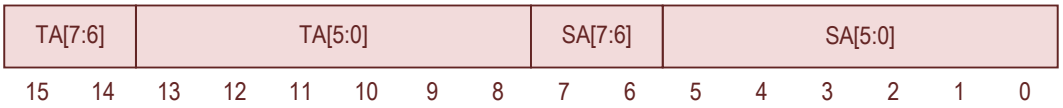


Figure 89 – RP Disk Address Register (RPDA)

Table 64 – RP Disk Address Register (RPDA) – IO Address 776706			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-8	TA[7:0]	R/W	Incremented after the last sector of the track has been transferred. Note: The track address must be valid for the type of disk. See Table 86 for the highest numbered track. Not all bits in this register field may be implemented depending on the disk parameters. Cleared by Read-in Preset command. This register is **NOT** reset by either the *IO Bridge Clear or Controller Clear*.
7-0	SA[7:0]	R/W	Sector Address. Incremented after the sector has been transferred. Note: The sector address must be valid for the type of disk. See Table 86 for the highest numbered sector. Not all bits in this register field may be implemented depending on the disk parameters. Cleared by Read-in Preset command. This register is **NOT** reset by either the *IO Bridge Clear or Controller Clear*.

12.1.3 RP Drive Status (RPDS) Register

This register reports the status of the disk drive.

Figure 90 – RP Drive Status Register (RPDS)

Table 65 – RP Drive Status Register (RPDS) – IO Address 776712			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	ATA	R	Attention Active. This bit is asserted under the following conditions: 1. The disk transitions to “on-line” or “off-line” (i.e., RPDS[MOL] changes state), or 2. Go with composite error (RPDS[ERR]) asserted, or 3. One of the following “Positioning Commands” completes: a. Unload, or b. Recalibrate, or c. Search, or d. Seek, or e. Offset, or f. Return-to-centerline. This bit is cleared under the following conditions: 1. Cleared by IO Bridge Clear, or 2. Controller Clear, or 3. Drive Clear command, or 4. Writing a ‘1’ to the bit associated with this drive in the Attention Summary (RPAS) pseudo register, or 5. Writing to RPCS1 under the following conditions: a. Go-bit (RPCS1[GO]) asserted, and b. No Parity Error (RPCS2[PAT] negated), and c. No Composite Error (RPDS[ERR] negated)
14	ERR	R	Composite Error. This bit is set if any bits in RPER1, RPER2, or RPER3 are set. This bit is combinationally derived from those registers – clearing that register will clear this bit. When this bit is set, the only command that is accepted is the “Drive Clear” command.

Table 65 – RP Drive Status Register (RPDS) – IO Address 776712			
Bit(s)	**Mnemonic**	**R/W**	**Description**
13	PIP	R	Positioning in progress. Set during Unload, Recalibrate, Seek, Offset, or Return-to-Center operation. PIP is not set during an implied seek or a mid-transfer seek. Cleared when the operation completes. Note: The RP05/RP06 Control Logic Maintenance Manual (EK-RP056-MM-01) Table 2-1 says that PIP is asserted during a search operation. This is incorrect. The RP06 will fail the DSRPA diagnostics TEST-276 if PIP is set during a search operation.
12	MOL	R	Media On-line. Asserted when an SD Card is inserted in the SD socket and has been initialized successfully. Negated when the SD Card is removed from the SD Socket.
11	WRL	R	Write Lock. Controlled by the RH11 Console Control Register
10	LST	R	Last Sector Transferred. Set when the last addressable sector has been read or written. Cleared when RPDA is written.
9	PGM	R	Programmable. Always read as zero.
8	DPR	R	Drive Present. Controlled by the RH11 Console Control Register
7	DRY	R	Drive Ready. Set at the completion of every command. Cleared at the start of every command.

Table 65 – RP Drive Status Register (RPDS) – IO Address 776712			
Bit(s)	**Mnemonic**	**R/W**	**Description**
6	VV	R	Volume Valid. This bit is set under the following conditions: 1. Issue a Pack Acknowledge command with no Composite Error (RPDS[ERR] = 0), or 2. Issue a Read-in Preset command with no Composite Error (RPDS[ERR] = 0). This bit is cleared when the device transitions from off-line (RPDS[MOL]=0) to online (RPDS[MOL]=1). This normally occurs when the SD card is inserted into the SD This is **NOT** cleared by Cleared by IO Bridge Clear, Controller Clear, or Drive Clear command. Notes: 1. Simply removing the SD Card from the SD Reader does not clear VV. Removing then reinstalling the SD Card clears VV. See DSRPA TEST-167. 2. Executing a Pack Acknowledge command or a Read-in Preset Command with a Composite Error (RPDS[ERR]) asserted does not set VV. See DSRPA TEST-170 This requires that the OPERATOR INTERVENTION tests are enabled.
5	DE1	R	Difference Equals 1. This is a signal from RP04 disk which indicates head load sequence status. Not implemented by the RP05/RP06 disk. Ignored by the diagnostics. See Note 2 on M7789/MB1. Input has pulldown resistor. Always read as zero
4	DL64	R	Difference Less Than 64. This is a signal from RP04 disk which indicates head load sequence status. Not implemented by the RP05/RP06 disk. Ignored by the diagnostics. See Note 2 on M7789/MB1. Input has pulldown resistor. Always read as zero
3	GRV	R	Go Reverse. This is a signal from RP04 disk which indicates head load sequence status. Not implemented by the RP05/RP06 disk. Ignored by the diagnostics. See Note 2 on M7789/MB1. Input has pulldown resistor. Always read as zero
2	DIGB	R	Drive to Inner Guard Buffer. This is a signal from RP04 disk which indicates head load sequence status. Not implemented by the RP05/RP06 disk. Ignored by the diagnostics. See Note 2 on M7789/MB1. Input has pulldown resistor. Always read as zero.

Table 65 – RP Drive Status Register (RPDS) – IO Address 776712			
Bit(s)	**Mnemonic**	**R/W**	**Description**
1	DF20	R	Drive Forward 20 inches/sec. This is a signal from RP04 disk which indicates head load sequence status. Not implemented by the RP05/RP06 disk. Ignored by the diagnostics. See Note 2 on M7789/MB1. Input has pulldown resistor. Always read as zero.
0	DF5	R	Drive Forward 5 inches/sec. This is a signal from RP04 disk which indicates head load sequence status. Not implemented by the RP05/RP06 disk. Ignored by the diagnostics. See Note 2 on M7789/MB1. Input has pulldown resistor. Always read as zero.

12.1.4 RP Error #1 (RPER1) Register

This register contains the error status of the addressed drive.

Figure 91 – RP Error Register #1 (RPER1)

Table 66 – RP Error Register #1 (RPER1) – IO Address 776714			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	DCK	R/W	Data check. DCK is set in Diagnostic Mode only (RPMR[DMD] asserted) when the data ECC check fails. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
14	UNS	R/W	Unsafe. Does nothing. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.

Table 66 – RP Error Register #1 (RPER1) – IO Address 776714			
Bit(s)	**Mnemonic**	**R/W**	**Description**
13	OPI	R/W	Operation Incomplete. OPI is set in Diagnostic Mode only (RPMR[DMD] asserted) when a search operation or an search associated with a read or write operation is performed and three diagnostic index pulses have been created (RPMR[DIND] asserted). Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
12	DTE	R/W	Drive Timing Error. DTE is set in Diagnostic Mode only (RPMR[DMD] asserted) when a sector pulse is created (RPMR[DIND] asserted). during a data transfer. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
11	WLE	R/W	Write Lock Error. Set by executing a write command on a write protected drive. Cleared by writing zero, *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
10	IAE	R/W	Invalid Address Error. Asserted when an invalid cylinder, sector, or track is selected and any of the following commands is executed: 1. Read, or 2. Read Header, or 3. Write, or 4. Write Header, or 5. Write Check, or 6. Write Check Header, or 7. Search, or 8. Seek Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
9	AOE	R/W	Address Overflow Error. Set when the controller requests a data transfer beyond last sector of the last cylinder of the last track on the pack. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.

Table 66 – RP Error Register #1 (RPER1) – IO Address 776714			
Bit(s)	**Mnemonic**	**R/W**	**Description**
8	HCRC	R/W	Header CRC Error. Asserted when a header CRC error is detected and Header Compare Inhibit is not enabled (rpOF[HCI] negated). Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
7	HCE	R/W	Header Compare Error. Does nothing. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
6	ECH	R/W	ECC hard failure. Does nothing. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
5	WCF	R/W	Write clock fail. Does nothing. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
4	FER	R/W	Format Error. Does nothing. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
3	PAR	R/W	Parity Error. Does nothing. Set if any write is received with Parity Test (RPCS2[PAT]) asserted. No other conditions can create a parity error as parity is not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
2	RMR	R/W	Register Modification Refused. Set by modifying any register (except RPAS or RPMR) when the unit is not ready; i.e., RPDS[DRY] is negated. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
1	ILR	R/W	Illegal register. This implementation of the RH11 cannot generate accesses to illegal registers. Does nothing. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.

Table 66 – RP Error Register #1 (RPER1) – IO Address 776714			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0	ILF	R/W	Illegal function. Set by executing an illegal function per Table 63 or by executing any function other than “Drive Clear” with Composite Error (RPDS[ERR]) asserted. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.

12.1.5 RP Attention Summary (RPAS) Register

The Attention Summary Pseudo Register allows the program to examine or modify the status of all disk drives in a single operation.

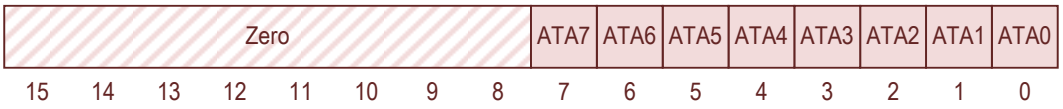


Figure 92 – RP Attention Summary Register (RPAS)

Table 67 – RP Attention Summary (RPAS) – IO Address 776716			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-8	Zero	R	Always read as zero.
7	ATA7	R/W	Attention Active. Reads value of Disk 7 RPDS[ATA]. Writing 1 clears Disk 7 RPDS[ATA].
6	ATA6	R/W	Attention Active. Reads value of Disk 6 RPDS[ATA]. Writing 1 clears Disk 6 RPDS[ATA].
5	ATA5	R/W	Attention Active. Reads value of Disk 5 RPDS[ATA]. Writing 1 clears Disk 5 RPDS[ATA].
4	ATA4	R/W	Attention Active. Reads value of Disk 4 RPDS[ATA]. Writing 1 clears Disk 4 RPDS[ATA].
3	ATA3	R/W	Attention Active. Reads value of Disk 3 RPDS[ATA]. Writing 1 clears Disk 3 RPDS[ATA].
2	ATA2	R/W	Attention Active. Reads value of Disk 2 RPDS[ATA]. Writing 1 clears Disk 2 RPDS[ATA].
1	ATA1	R/W	Attention Active. Reads value of Disk 1 RPDS[ATA]. Writing 1 clears Disk 1 RPDS[ATA].
0	ATA0	R/W	Attention Active. Reads value of Disk 0 RPDS[ATA]. Writing 1 clears Disk 0 RPDS[ATA].

12.1.6 RP Look Ahead (RPLA) Register

This register would normally report the sector “under the head”. Highly optimized software could look at the current sector and optimally access data based on the actual sector position. This is not really necessary for Secure Digital (SDHC) media as it has no rotational latency.

HOWEVER –

Some DSRPA diagnostics expect that the bit fields in the RPLA register change in a sensible manner. Also some diagnostics expect that when a search command completes, the contents of the RPLA register are consistent with the search sector in the RPDA register.

The RPXX has special Diagnostic Mode hardware that allows the sector addressing to be tested via the Maintenance Mode Register (RPMR) and the Look Ahead Register (RPLA). This is enabled when the unit is in Diagnostic Mode (RPMR[DMD] asserted).

When RPMR[FMT22] is negated (18-bit mode), there are 20 sectors per track. There are 672 bytes per sector and therefore 13440 bytes per track. Of the 672 bytes of data per sector, 576 bytes are payload and 96 bytes are pre-header, header, header gap, ECC, data gap, and tolerance gap.

When RPMR[FMT22] is asserted (16-bit mode), there are 22 sectors per track. There are 608 bytes per sector and therefore 13376 bytes per track. Of the 608 bytes of data per sector, 512 bytes are payload and 96 bytes are pre-header, header, header gap, ECC, data gap, and tolerance gap.

Notice that the number of bytes per track is fairly consistent between the two modes.

This can all be tested in Diagnostic Mode.

The sector byte counter can be reset by generating an index pulse via the Diagnostic Index Pulse bit of the Maintenance Register (RPMR[DIND]). Thereafter, the sector byte counter can be incremented by bit-banging a Diagnostic Sector Clock via the RPMR[DSCK] bit. The result can be observed via the Look Ahead Register.

The EXT field is incremented to 1 on the 127th clock pulse, incremented to 2 on the 255th clock pulse, and incremented to 3 on the 511th clock pulse. If RPMR[FMT22] is negated (18-bit mode), the EXT field is incremented back to 0 and the SECTOR field is incremented on the 672nd clock pulse. If RPMR[FMT22] is asserted (16-bit mode), the EXT field is incremented back to 0 and the SECTOR field is incremented on the 609th clock pulse.

Figure 93 – RP Look Ahead Register (RPLA)

<table>
<tr><th colspan="6">Table 68 – RP Look Ahead (RPLA) – IO Address 776720</th></tr>
<tr><th>Bit(s)</th><th>Mnemonic</th><th>R/W</th><th colspan="3">Description</th></tr>
<tr><td>15-12</td><td>Zero</td><td>R</td><td colspan="3">Always read as zero.</td></tr>
<tr><td>11-6</td><td>LAS</td><td>R</td><td colspan="3">Look Ahead Sector.
Sector ‘under the head’.</td></tr>
<tr><td rowspan="6">5-4</td><td rowspan="6">LAE</td><td rowspan="6">R</td><td colspan="3">Look Ahead Extension</td></tr>
<tr><th>Bit 5</th><th>Bit 4</th><th>Description</th></tr>
<tr><td>0</td><td>0</td><td>First quarter</td></tr>
<tr><td>0</td><td>1</td><td>Second quarter</td></tr>
<tr><td>1</td><td>0</td><td>Third quarter</td></tr>
<tr><td>1</td><td>1</td><td>Fourth quarter</td></tr>
<tr><td>3-0</td><td>Zero</td><td>R</td><td colspan="3">Always read as zero</td></tr>
</table>

12.1.7 RP Maintenance (RPMR) Register

The maintenance register is implemented as much as is required to pass diagnostic tests. .

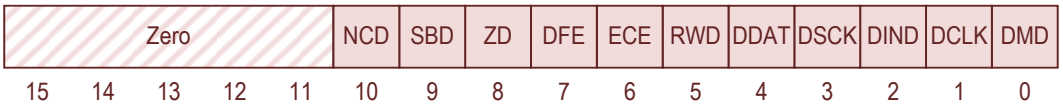


Figure 94 – RP Maintenance Register (RPMR)

Table 69 – RP Maintenance Register (RPMR) – IO Address 776724			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-11	Zero	R	Read as zero. Writes ignored.
10	NCD	R	Not implemented. Always read as zero. Writes ignored.
9	SBD	R	Sync byte detected. Read only. Writes ignored. Implemented in Diagnostic Mode only. This bit is asserted when a sync byte is detected.

Table 69 – RP Maintenance Register (RPMR) – IO Address 776724			
Bit(s)	**Mnemonic**	**R/W**	**Description**
8	ZD	R	Zero detect. Read only. Writes ignored Implemented in Diagnostic Mode only. This bit is asserted if the ECC is zero after reading the ECC field.
7	DFE	R	Data Field Envelope. Read only. Writes ignored Implemented in Diagnostic Mode only. This bit is asserted when the *Data Field* of the sector is under the disk head. This field of the disk stores 256 words of data. This corresponds to 4608 bits in 16-bit mode or 4096 bits in 16-bit mode. This bit is asserted on the bits (set by RPMR[DCLK]) of the sector as follows: 1. 496th to 4591st bits (16-bit mode) 2. 496th to 5103rd bits (18-bit mode) Note: The EK-RP056-MM-01 (Dec 1975) Maintenance Manual documents this bit in the wrong position of the RPMR. See MP-00086 Schematics (M7774/RG2) and DSRPA diagnostic.
6	ECE	R	Error Correction Envelope. Read only. Writes ignored Implemented in Diagnostic Mode only. This bit is asserted when the *ECC Field* or the sector is under the disk head. This field of the disk stores 2 16-bit words of data. This corresponds to 32 bits. This bit is asserted on the bits (set by RPMR[DCLK]) of the sector as follows: 1. 4592nd to 4623nd bits (16-bit mode) 2. 5104th to 5135th bits (18-bit mode) Note: The EK-RP056-MM-01 (Dec 1975) Maintenance Manual documents this bit in the wrong position of the RPMR. See MP-00086 Schematics (M7774/RG2) and DSRPA diagnostic.

Table 69 – RP Maintenance Register (RPMR) – IO Address 776724			
Bit(s)	**Mnemonic**	**R/W**	**Description**
5	DWRD	R	Diagnostic Write Data. Read only. Writes ignored. In Diagnostic Mode and during a Write Data Command or Write Header Command the bits that would have been written to the disk can be read serially from this bit. This includes the sector header (if applicable), data fields, ECC fields, and data gap. These bits are clocked by the falling edge of the Diagnostic Data Clock (RPMR[DCLK]).
4	DRDD	R/W	Diagnostic Read Data. In Diagnostic Mode and during a Read Command, Read Header Command, Write Check Command, or Write Check Header Command, the bits that are clocked to this port are handled by the controller as if they had been read by the disk drive. This includes the sector header (if applicable), data fields, ECC fields, and data gap. These bits are clocked by the falling edge of the Diagnostic Data Clock (RPMR[DCLK]). Reading this bit returns the last value that was written. When RPMR[DMD] is negated, this signal is held in reset (writes ignored, cleared, and read as zero).
3	DSCK	R/W	Diagnostic Sector Clock. When RPMR[DMD] is asserted, the generates a Diagnostic Sector Clock which increments the Sector Extension Counter and therefore the Sector Counter – see RPLA register. Note: this increments once per byte of data. When RPMR[DMD] is negated, this signal is held in reset (writes ignored, cleared, and read as zero).
2	DIND	R/W	Diagnostic Index Pulse. When RPMR[DMD] is asserted, the generates a Diagnostic Index Pulse which resets the Sector Counter – see RPLA register. When RPMR[DMD] is negated, this signal is held in reset (writes ignored, cleared, and read as zero).
1	DCLK	R/W	Diagnostic Data Clock. Each rising edge of the Diagnostic Data Clock clocks one bit of data as if the data was read from the disk drive. When RPMR[DMD] is negated, this signal is held in reset (writes ignored, cleared, and read as zero).

Table 69 – RP Maintenance Register (RPMR) – IO Address 776724			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0	DMD	R/W	Diagnostics mode. Enables the diagnostic bits enumerated above. Set by writing a 1 when RHCS1[GO] is negated. Cleared by writing zero, *IO Bridge Clear, Controller Clear*, or *Drive Clear* command.

12.1.8 RP Drive Type (RPDT) Register

This register indicates the type of disk drive or tape drive that is connected to the controller.

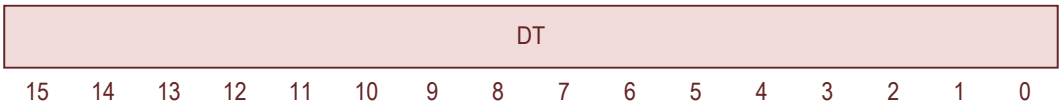


Figure 95 – RP Drive Type Register (RPDT)

Table 70 – RP Drive Type Register (RPDT) – IO Address 776726				
Bit(s)	**Mnemonic**	**R/W**	**Drive**	**Register Contents**
15-14	Zero	R	All	Always zero.
13	MOH	R	All	Always one. Indicates a 'moving head' disk drive.
12	Zero	R	All	Always zero.
11	DRQ	R	All	Always zero. Indicates a dual port disk drive.
10-8	Zero	R	All	Always zero.
7-0	DT	R	RM03	0024. RPDT reports 020024
			RP04	0020. RPDT reports 020020
			RP05	0021. RPDT reports 020021
			RP06	0022. RPDT reports 020022
			RM80	0026. RPDT reports 020026
			RM05	0027. RPDT reports 020027
			RP07	0042. RPDT reports 020042

12.1.9 RP Serial Number (RPSN) Register

The RPSN register reports the Serial Number of the disk drive. The Serial Number is hardwired to the disk drive number. These are the same values that SIMH uses.

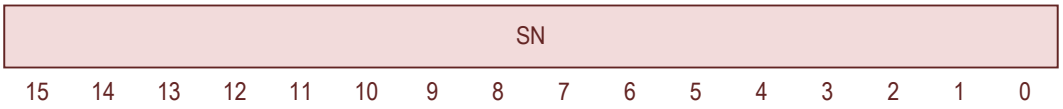


Figure 96 – RP Serial Number Register (RPSN)

Table 71 – RP Serial Number Register (RPSN) – IO Address 776730				
Bit(s)	**Mnemonic**	**R/W**	**Drive**	**Register Contents**
15-0	SN	R	0	000021
			1	000022
			2	000023
			3	000024
			4	000025
			5	000026
			6	000027
			7	000030

12.1.10 RP Offset (RPOF) Register

An RPxx drive has the ability to offset its heads off of the track centerline in either direction.

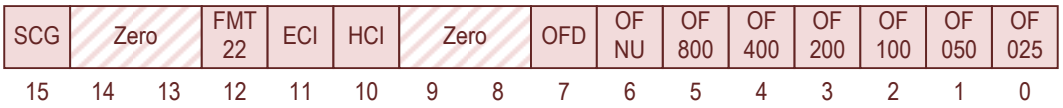


Figure 97 – RP Offset Register (RPOF)

Table 72 – RP Offset Register (RPOF) – IO Address 776732			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	SCG	R	Sign Change. Used to verify head alignment. Not implemented. Masked/ignored by diagnostics. Always read as zero. Note: The document “RJP04 Moving Head Disk Subsystem Maintenance Manual” (DEC-11-HRJPA-B-D) says bit is read/write while the schematic shows it as read-only.
14-13	Zero	R	Writes ignored. Read as zero.
12	FMT22	R/W	Format. Partially implemented. 0: 18-bit mode. 1: 16-bit mode. Does nothing except in maintenance mode as required by the diagnostics. See Section 12.1.6. The disk is always reads and writes data in an 18-bit mode. Cleared by *Read-in Preset* command.
11	ECI	R/W	Error Correction Inhibit. When asserted in Diagnostic Mode (RPMR{DMD] asserted), this bit inhibits the Error Correction when a ECC error is detected. Does nothing when not in Diagnostic Mode. Cleared by *Read-in Preset* command.
10	HCI	R/W	Header Compare Inhibit. When asserted in Diagnostic Mode (RPMR{DMD] asserted), this bit prevents reporting the header CRC errors (RPER1[HCRC]). Does nothing when not in Diagnostic Mode. Cleared by *Read-in Preset* command.
9-8	Zero	R	Writes ignored. Read as zero.
7	OFD	R/W	Head offset Direction. Does nothing. Cleared by *Master Reset* , Return-to-Center command, or by an implied Return-to-Center command which occurs before the following commands are executed: • Seek command , or • Write command, or • Write Header command.

Table 72 – RP Offset Register (RPOF) – IO Address 776732			
Bit(s)	**Mnemonic**	**R/W**	**Description**
6-0	OFS	R/W	Head offset in 25 microinch increments. Does nothing. Cleared by *Master Reset* , Return-to-Center command, or by an implied Return-to-Center command which occurs before the following commands are executed: • Seek command , or • Write command, or • Write Header command.

12.1.11 RP Desired Cylinder (RPDC) Register

The cylinder is specified in this register.

Figure 98 – RP Desired Cylinder Register (RPDC)

Table 73 – RP Desired Cylinder (RPDC) – IO Address 776734			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-10	Zero	R	Always read as zero.
9-0	DCA	R/W	Desired Cylinder. Set by writing the register. Incremented following a read/write of the last sector of the last track of the cylinder Cleared by Read-in Preset command or Recalibrate command. This register is NOT reset by either the *IO Bridge Clear, or Controller Clear*.

12.1.12 RP Current Cylinder (RPCC) Register

The RPCC register returns the Current Cylinder.

Figure 99 – RP Current Cylinder Register (RPCC)

Table 74 – RP Current Cylinder Register (RPCC) – IO Address 776736			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-10	Zero	R	Always read as zero.
9-0	CCA	R	Current Cylinder Address. The Current Cylinder Address (CCA) is updated with the contents of the Desired Cylinder Address (DCA) under the following conditions: 1. IO Bridge Clear, or 2. Controller Clear, or 3. After the following commands that cause head motion: a. Unload b. Seek c. Implied seek (Read, Write, Write Check, Search) Cleared by Recalibrate Command. Note: In Diagnostic Mode (RPMR[DMD] asserted), everything described above still occurs; however, the actual RP06 head does not move. This causes the controller and disk to become “unsynchronized”. This behavior is tested by DSRPA TEST-270. Exiting Diagnostic Mode and executing a Recalibrate function restores synchronization.

12.1.13 RP Error Status #2 (RPER2) Register

The RPER2 Register would normally report hardware status. This register is read/write but is never modified by the disk controller. This is tested by the DSRPA diagnostics.

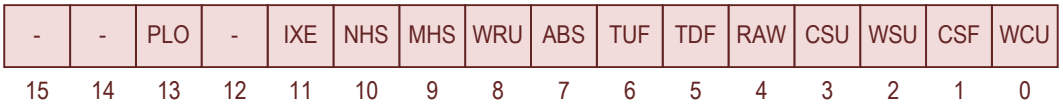


Figure 100 – RP Error Status #2 (RPER2)

Table 75 – RP Error Status Register #2 (RPER2) – IO Address 776740			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	-	R/W	Not used. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
14	-	R/W	Not used. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.

Table 75 – RP Error Status Register #2 (RPER2) – IO Address 776740			
Bit(s)	**Mnemonic**	**R/W**	**Description**
13	PLO (PLU)	R/W	PLO unsafe. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
12	-	R/W	Not used. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
11	IXE	R/W	Index error. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
10	NHS	R/W	No head select. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
9	MHS	R/W	Multiple head select. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
8	WRU	R/W	Write ready unsafe. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
7	ABS	R/W	Abnormal stop. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
6	TUF	R/W	Transitions unsafe. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
5	TDF	R/W	Transitions detected failure. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.

Table 75 – RP Error Status Register #2 (RPER2) – IO Address 776740			
Bit(s)	**Mnemonic**	**R/W**	**Description**
4	RAW	R/W	Read and write. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
3	CSU	R/W	Current switch unsafe. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
2	WSU	R/W	Write select unsafe. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
1	CSF	R/W	Current sink failure. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
0	WCU	R/W	Write current unsafe. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.

12.1.14 RP Error Status #3 (RPER3) Register

The RPER3 Register would normally report error status. This register is read/write but is never modified by the disk controller. This is tested by the DSRPA diagnostics.

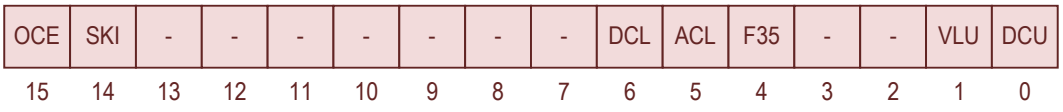


Figure 101 – RP Error Status #3 (RPER3)

Table 76 – RP Error Status Register #1 (RPER3) – IO Address 776742			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	OCE (OCYL)	R/W	Off Cylinder Error. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
14	SKI	R/W	Seek Incomplete. Does nothing except in maintenance mode as required by the diagnostics. Set by…. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
13	OPE	R/W	Unused. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*
12-7	-	R/W	Unused. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*
6	ACL	R/W	AC Low. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
5	DCL	R/W	DC Low. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
4	F35	R/W	35V Regulator Failure. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.
3-2	-	R/W	Unused. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*
1	VLU (WA0)	R/W	Velocity Unsafe. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*
0	DCU	R/W	DC Unsafe. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.

12.1.15 RP Error Position (RPEC1) Register

The ECC in the RP06 disk is a “Fire Code” with the following generator polynomial:

	$g(x) = x^{32} + x^{23} + x^{21} + x^{11} + x^2 + 1 = (x^{21} + 1)(x^{11} + x^2 + 1)$	Equation 2

The RPEC1 Register reports the error position. The ECC (Fire Code) in the RP06 is only useful with a record length of 4644 bits or less. This is a valid assumption because the maximum RP06 record length in 18-bit mode is 4608 bits – which is 128 36-bit words. The record length is smaller in 16-bit mode.

Figure 102 – RP Error Position Register (RPEC1)

Table 77 – RP Error Position Register (RPEC1) – IO Address 776744			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-13	-	R	Writes ignored. Always read as zero.
12-0	EC1	R	Not implemented. Writes ignored. Always read as zero.

12.1.16 RP Error Pattern (RPEC2) Register

The RPEC2 Register reports error correction data. The ECC (Fire Code) in the RP06 can only correct burst errors with a length of 11 bits or less.

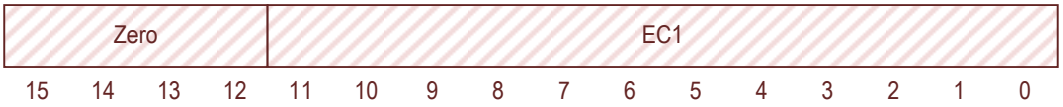


Figure 103 – RP Error Pattern Register (RPEC2)

Table 78 – RP Error Pattern Register (RPEC2) – IO Address 776746			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-12	-	R	Writes ignored. Always read as zero.
11-0	EC2	R	Not implemented. Writes ignored. Always read as zero.

12.2 RMXX Registers

12.2.1 RM Control and Status #1 (RMCS1) Register

This register is identical to the RPCSR Register.

12.2.2 RM Disk Address (RMDA) Register

This register is similar to the RPDA Register. The register fields may be different sizes depending on the disk type.

12.2.3 RM Drive Status (RMDS) Register

This register is identical to the RPDS Register.

12.2.4 RM Error #1 (RMER1) Register

This register is identical to the RPER1 Register.

12.2.5 RM Attention Summary (RMAS) Register

This register is identical to the RPAS Register.

12.2.6 RM Look Ahead (RMLA) Register

The RPLA register is similar to the RPLA register, except that the sector extension field is not implemented.

Figure 104 – RM Look Ahead Register (RMLA)

Table 79 – RM Look Ahead (RMLA) – IO Address 776720			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-12	Zero	R	Always read as zero.
11-6	LAS	R	Look Ahead Sector. Sector ‘under the head’.
5-0	Zero	R	Always read as zero

12.2.7 RM Maintenance Register #1 (RMMR1) Register

The RMMR1 has different data during reads and writes.

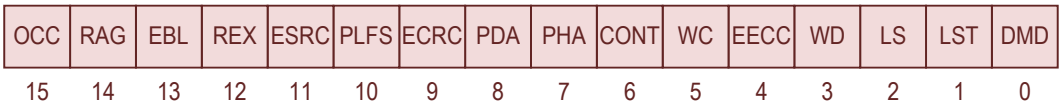


Figure 105 – RM Maintenance Register #1 (RMMR) (READ)

Table 80 – RM Maintenance Register (RMMR1) – IO Address 776724 (READ)			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	OCC	R	Occupied. Note implemented. Read as zero.
14	RAG	R	Run And Go Not implemented. Read as zero.
13	EBL	R	End of Block Not implemented. Read as zero.
12	REX	R	Massbus Exception Not implemented. Read as zero.
11	ESRC	R	Enable Search Not implemented. Read as zero.
10	PLFS	R	Looking For Sync Not implemented. Read as zero.
9	ECRC	R	Enable CRC Out Not implemented. Read as zero.
8	PDA	R	Data Area Not implemented. Read as zero.
7	PHA	R	Header Area Not implemented. Read as zero.

Table 80 – RM Maintenance Register (RMMR1) – IO Address 776724 (READ)			
Bit(s)	**Mnemonic**	**R/W**	**Description**
6	CONT	R	Continue Not implemented. Read as zero.
5	WC	R	PROM Strobe Not implemented. Read as zero..
4	EECC	R	Enable ECC Out Not implemented. Read as zero..
3	WD	R	Write Data Not implemented. Read as zero.
2	LS	R	Last Sector Not implemented. Read as zero.
1	LST	R	Last Sector / Track Not implemented. Read as zero.
0	DMD	R	Diagnostic Mode Not implemented. Read as zero.

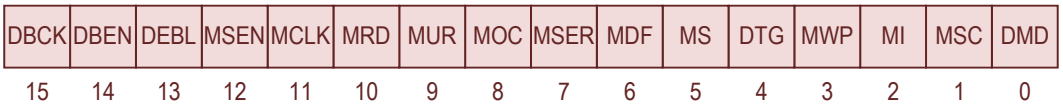


Figure 106 – RM Maintenance Register #1 (RMMR) (WRITE)

Table 81 – RM Maintenance Register (RMMR1) – IO Address 776724 (WRITE)			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	DBCK	W	Debug Clock Note implemented. Writes ignored.

Table 81 – RM Maintenance Register (RMMR1) – IO Address 776724 (WRITE)			
Bit(s)	**Mnemonic**	**R/W**	**Description**
14	DBEN	W	Debug Clock Enable Not implemented. Writes ignored.
13	DEBL	W	Diagnostic End Of Block Not implemented. Writes ignored.
12	MSEN	W	Disable Search Timeout Not implemented. Writes ignored.
11	MCLK	W	Maintenance Clock Not implemented. Writes ignored.
10	MRD	W	Maintenance Read Data Not implemented. Writes ignored.
9	MUR	W	Maintenance Unit Ready Not implemented. Writes ignored.
8	MOC	W	Maintenance On Cylinder Not implemented. Writes ignored.
7	MSER	W	Maintenance Seek Error Not implemented. Writes ignored.
6	MDF	W	Maintenance Drive Fault Not implemented. Writes ignored.
5	MS	W	Maintenance Sector Pulse Not implemented. Writes ignored.
4	DTG	W	Reserved Writes ignored.

Table 81 – RM Maintenance Register (RMMR1) – IO Address 776724 (WRITE)			
Bit(s)	**Mnemonic**	**R/W**	**Description**
3	MWP	W	Maintenance Write Protect Not implemented. Writes ignored.
2	MI	W	Maintenance Index Pulse Not implemented. Writes ignored.
1	MSC	W	Maintenance Sector Compare Not implemented. Writes ignored.
0	DMD	W	Diagnostic Mode Not implemented. Writes ignored.

12.2.8 RM Drive Type (RMDT) Register

This register is identical to the RPDT Register.

12.2.9 RM Serial Number (RMSN) Register

This register is identical to the RPSN Register.

12.2.10 RM Offset (RMOF) Register

This register is close to the RPOF register, except that the bit[6:0] is not used, always zero.

12.2.11 RM Desired Cylinder (RMDC) Register

This register is identical to the RPDC Register.

12.2.12 RM Holding Register (RMHR) Register

This register is not present in an RPxx type disk drive. This register is updated with the complement of the last value that is written to any valid RM register.

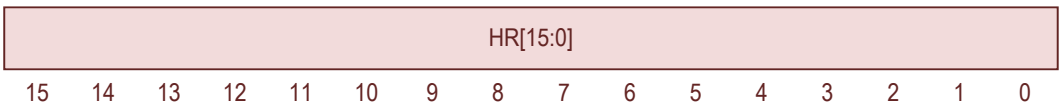


Figure 107 – RM Holding Register (RMHR)

Table 82 – RH Holding Register (RMHR) – IO Address 776736			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-10	HR	R	This returns the complement of the data that was last written to any valid RH register.

12.2.13 RM Maintenance Register #2 (RMMR2) Register

Figure 108 – RM Maintenance Register #2 (RMMR2)

Table 83 – RM Maintenance Register #2 (RMMR2) – IO Address 776740			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	RQA	R	Request A Indicates that a required has been received on Port A. Not implemented. Writes ignored. Always read as zero.
14	RQB	R	Request B Indicates that a required has been received on Port B. Not implemented. Writes ignored. Always read as zero.
13	TAG	R	Indicates the status of the control select tag lines. Not implemented. Writes ignored. Always read as zero.
12	TST	R	Control Select Not implemented. Writes ignored. Always read as zero.
11	CIC	R	Control or Cylinder Select Not implemented. Writes ignored. Always read as zero.

Table 83 – RM Maintenance Register #2 (RMMR2) – IO Address 776740			
Bit(s)	**Mnemonic**	**R/W**	**Description**
10	CIH	R	Control or Head Select Not implemented. Writes ignored. Always read as zero.
9-0	BB	R	Tag Bus Bits Not implemented. Writes ignored. Always read as zero.

12.2.14 RM Error Register #2 (RMER2) Register

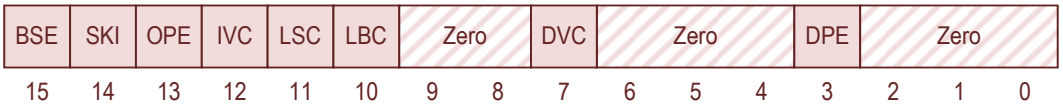


Figure 109 – RM Error Register #2 (RMER2)

Table 84 – RM Error Register #2 (RMER2) – IO Address 776742			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	BSE	R	Bad Sector Error Not implemented. Writes ignored. Always read as zero.
14	SKI	R	Seek Incomplete Not implemented. Writes ignored. Always read as zero.
13	OPE	R	Operator Plug Error Set when the logical address plug is removed from the drive. Not implemented. Writes ignored. Always read as zero.

Table 84 – RM Error Register #2 (RMER2) – IO Address 776742			
Bit(s)	**Mnemonic**	**R/W**	**Description**
12	IVC	R	Invalid Command. Set when a command is receive while Volume Valid or Drive Ready are not set. Not implemented. Writes ignored. Always read as zero.
11	LSC	R	Loss of system clock Not implemented. Writes ignored. Always read as zero.
10	LBC	R	Loss of bit clock Not implemented. Writes ignored. Always read as zero.
9-8	-	R	Reserved Writes ignored. Always read as zero.
7	DVC	R	Device Check Set by drive to indicate low AC power or head select failure. Not implemented. Writes ignored. Always read as zero.
6-4	-	R	Reserved Writes ignored. Always read as zero.
3	DPE	R	Data Parity Error Not implemented. Writes ignored. Always read as zero.
2-0	-	R	Reserved Writes ignored. Always read as zero.

12.2.15 RM Error Position (RMEC1) Register

This register is identical to the RPEC1 Register.

12.2.16 RM Error Pattern (RMEC2) Register

This register is identical to the RPEC2 Register.

12.3 Disk Functions

This section summarizes the RPXX commands and how they are implemented in the KS10 FPGA.

12.3.1 Seek Function

A seek operation moves the heads to the appropriate cylinder.

The seek operation is governed by three registers: the Desired Cylinder Register (RPDC), the Current Cylinder Register (RPCC), and a register that retains the simulated position of the disk head.

The seek operation (or an implied seek operation) is initiated when a seek command, search command, or data transfer command (read, write, or write check) is issued and the desired cylinder register is different than the current cylinder register.

The disk will not perform the seek operation if the desired cylinder is the same as the current cylinder. This is tested by DSPRA TEST-262.

The disk will not perform the seek operation if the desired cylinder is an invalid address. FIXME: Should the disk still seek if the track address or the sector address is invalid? Right now an invalid cylinder, track, or sector will prevent the disk from seeking. TODO: Check the schematic.

The RP06 advertises a track-to-track seek time of 6 milliseconds and a track 0 to track 814 seek time of 53 milliseconds.[2]

The KS10 FPGA can accurately simulate head motion using a lookup table as follows:

Table 85 - RP06 Seek Timing Simulation

Seek Distance (cylinders)	Seek Time (milliseconds)
0	N/A
1	5
2 - 3	10
4 - 7	15
8 - 15	20
16 - 31	25
32 - 63	30

[2] Memorex document 677-01/51.20-00, “677-01 DEC and 677-51 DEC Disc Storage Drives Technical Manual”, Table 1-1, pp 1-15.

64 - 127	35
128 - 255	40
256 - 511	45
512 - 813	50

The KS10 FPGA can also simulate head motion using a fixed delay for all seeks. This is faster but is less accurate than the lookup table approach.

The selection between the fast seek operation or the accurate seek operation is controlled by a conditional compile in the Verilog code. The code must be re-synthesized to change the type of seek operation.

None of the diagnostics appear to measure seek timing.

When the RPXX is in Diagnostic Mode (RPMR[DMD] asserted) the disk head does not move when a seek command is issued. If the seek command is aborted by asserting a Controller Clear Command (RHCS1[CLR]), the current cylinder register is updated with the contents of the desired cylinder register. Therefore the current cylinder and the position of the disk head can become unsynchronized. A recalibrate command will restore synchronization.

In Diagnostic Mode(RPMR[DMD] asserted) the seek operation is completed by: TBD.

An RPER3[SKI] error may only be created in Diagnostic Mode and is asserted when the current cylinder and the disk head are unsynchronized as describe above and the disk is commanded to seek off of the edge of the disk – either toward the center of the disk or toward the edge of the disk. A SKI error also causes the controller to execute an auto-recalibrate operation. The SKI error and auto-recalibration operation is tested by DSRPA TEST-270.

12.3.2 Search Function

A search operation occurs after the seek operation and after the head selection operation. The search operation finds a specific sector on the selected track.

A search operation may include an implied seek operation.

A search operation may be separate from all other operations or may be part of a data transfer operation.

The search time is related to the rotation speed of the disk. The minimum search time is zero if the disk is exactly the correct position to start reading data. The maximum search time is one complete rotation of the disk. In the case of the RP06, the disk rotates at 3600 RPM; therefore the maximum search time is 16.67 milliseconds and the average search time is 8.33 milliseconds.

The KS10 FPGA can simulate the disk rotation such that the sector under the head is constantly changing at a rate that is correct for the disk drive. The sector under the head is visible via the RPLA register. This is a very accurate simulation of disk rotation but is very slow. This level of simulation fidelity is required to pass some of the DSRPA diagnostics. For example: the DSRPA TEST-302 diagnostic watches the RPLA register and verifies that the sector under the head (RPLA register) is the same as the desired sector (RPDS register) when the seek operation completes.

The KS10 FPGA can also simulate a *seek* operation as just a short time delay. This is essentially the tactic used by SIMH. This is faster but less accurate and will fail some of the diagnostic tests.

The selection between the fast search operation or the accurate search operation is controlled by a conditional compile in the Verilog code. The code must be re-synthesized to change the type of seek operation.

Regardless of Diagnostic Mode, the search command completes on a Class B Error.

The search function can also operate in Diagnostic Mode. In Diagnostic Mode, the search operation completes when a Diagnostic Index Pulse is created via bit-banging the Diagnostic Index bit (RPMR[DIND]) of the Maintenance Register.

12.3.3 Offset Command and Return to Centerline Functions

The offset command moves the disk head off of the centerline of the track by some fraction of the track spacing ("micro"-seek) and is used to extract data from a misaligned disk pack (among other things).

The return to centerline command returns the head back to the centerline of the track.

The KS10 FPGA disk simulator only simulates the timing of these commands inasmuch as they are tested by the DSRPA diagnostics.

An implied return to centerline operation occurs before a seek command or one of the write operations. The disk will never write data to the disk in offset mode. The implied return to centerline for a seek operation is tested in DSRPA TEST-310.

12.3.4 Recalibrate Function

The recalibrate function drives the disk to cylinder 0 and clears the Current Cylinder register (RPCC). The recalibrate timing is the same as a seek from the current cylinder to cylinder 0.

12.3.5 Unload Function

On an RPxx disk, the unload function would unload the heads, spin-down the disk, off-line the disk drive, allow the operator to change the disk pack, on-line the disk, spin-up the disk, and reload the heads.

???FIXME: I'd really like to understand the application for this command. Is it used by something like the unix mount/umount command?

???FIXME: Not sure how I'm going to implement this with a single SD card.

12.3.6 Pack Acknowledge Function

Sets Volume Valid (RPDS[VV]).

12.3.7 Read-in Preset Function

This command sets the Volume Valid (RPDS[VV]) bit, clears the Sector Address Register (RPDA[SA]), clears the Track Address Register (RPDA[TA]), clears the Desired Cylinder Address Register (RPDC[DCA]), clears the 16-bit format bit (RPOF[FMT22]), clears the Header Compare Inhibit bit (RPOF[HCI]), and clears the Error Correction Inhibit bit (RPOF[ECI]).

It is used to bootstrap the device.

12.3.8 Release Function

Used by dual port operations. This command performs a drive clear function and releases the Drive for use by the other controller.

12.3.9 Data Transfer Functions

The RP06 spins at 3600 RPM (60 rotations per second).

In 18-bit mode, a track of data contains 20 sectors. Each sector contains 672 bytes of header and data. Therefore the data transfer rate of the RP06 in 18-bit mode is calculated to be 806,400 bytes per second.

In 16-bit mode, a track of data contains 22 sectors. Each sector contains 608 bytes of header and data. Therefore the data transfer rate of the RP06 in 16-bit mode is calculated to be 802,560 bytes per second.

These calculations are is consistent with the advertised data transfer rate of 806,000 bytes per second.[3]

Regardless of Diagnostic Mode, the all of the Data Transfer functions complete on a Class B Error.

The header field consists of 4 16-bit words formatted as follows:

Data is read LSB first

Figure 110 – Sector Header Word #1

[3] Memorex document 677-01/51.20-00, “677-01 DEC and 677-51 DEC Disc Storage Drives Technical Manual”, Table 1-1, pp 1-15.

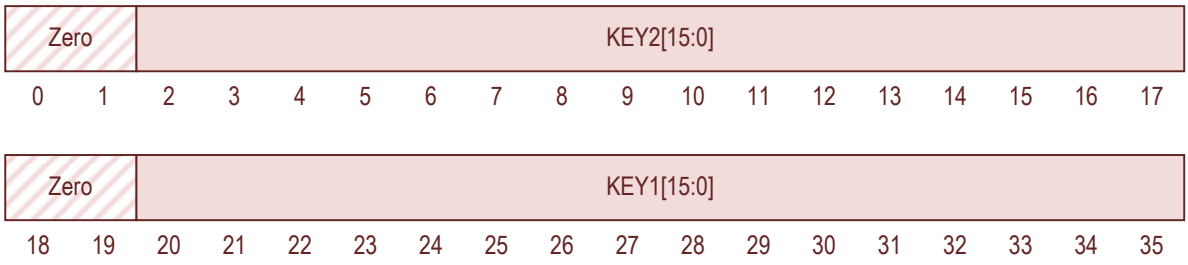


Figure 111 – Sector Header Word #2

12.3.9.1 Read header plus data

12.3.9.2 Read data

12.3.9.3 Write header plus data

12.3.9.4 Write header

12.3.9.5 Write check header plus data

12.3.9.6 Write check data

12.4 Disk Completion Monitor

This is where the KS10 FPGA design departs significantly from the classical KS10 implementation. Modern disk drives, even solid-state Secure Digital (SD) disk drives are significantly larger than the disk drives that existed when the KS10 was manufactured. To that end it is desirable for the KS10 FPGA to support 8 logical disk drives on one large chunk of physical media. To accomplish that, a mechanism for arbitrating exclusive access to the physical media must be provided.

Each of the disk simulators notifies the Disk Completion Monitor when the simulated disk delays have elapsed. When a disk is ready for access, the completion monitor will serialize exclusive access to SD card. When the SD Card access is completed, the associated disk simulator is notified. Only then will the disk simulator report that it is no longer busy and is ready for the next disk operation. The Disk Completion Monitor scans the disk simulators sequentially (round robin) and the disk simulators may have to wait for access to the SD Card.

Refer to Figure 82.

12.5 Secure Digital (SD) Disk Controller

One of the key goals of the KS10 FPGA disk system is to use the same bits-on-disk format as SIMH. That allows the KS10 FPGA to use any of the commonly available SIMH disk images without modification. It also allows the user to use SIMH to read tape images from the Internet, write the data to an SD Card from SIMH, and transfer the SD Card to the KS10 FPGA system.

The SD interface is chosen for the disk interface because:

1. The SD interface is a simple 6-wire serial interface which includes support for write-protect and for card detect. This is significantly fewer wires than a Parallel ATA (PATA) or PCMCIA interface.
2. The SD has zero rotational latency and zero seek times.
3. The SD interface is very high speed. Transfer rates up to 25 MBPS are easily supported.
4. The SD media is solid state and reliable. I don't plan on simulating head crashes.
5. The SD media is very inexpensive and is available everywhere. A $4.00 4GB SD card from Wal-Mart will provide media for 8 RP07s.
6. SD is a removable and portable media. Changing SD Media is like changing disk packs – except quicker and not as heavy. They also fit in your pocket but are easier to lose - especially the micro SD cards.

12.6 Secure Digital (SD) Capability Issues

SD cards are mostly designed to support the PC industry where 512-byte sectors are ubiquitous. There are some constraints which must be understood and addressed in order to use SD media in this design:

1. SD sectors are 512 bytes. SD Cards can only support reads and writes of the entire 512-byte sector. Partial sector reads/writes are not supported. Obviously the KS10 did not use 512-byte sectors.
2. SD cards are accessed via a 32-bit linear sector address. The SIMH/KS10 Cylinder/Head/Sector (CHS) addressing has to be mapped to a 32-bit SD Sector address.

The implications of these constraints will be analyzed in the following sections.

12.6.1 SIMH Cylinder/Head/Sector (CHS) Disk Addressing

As stated above, it would be really nice if the KS10 FPGA could use SIMH disk images without modification. To accomplish this, the disk addressing of SIMH needs to be understood.

The SIMH code calculates the disk address as follows:

```
#define GET_SC(x)        (((x) >> DA_V_SC) & DA_M_SC)
#define GET_SF(x)        (((x) >> DA_V_SF) & DA_M_SF)
#define GET_CY(x)        (((x) >> DC_V_CY) & DC_M_CY)

#define GET_DA(c,fs,d)   ((((GET_CY(c) * drv_tab[d].surf) + \
                         GET_SF(fs)) * drv_tab[d].sect) + GET_SC(fs))

where:  SF = Surface (aka Track or Head) from DA Register
        SC = Sector from DA Register
        CY = Cylinder from DC Register
```

The offset into the Disk Image is therefore:

$$File\ Offset\ (in\ bytes) = \Big(\big((C * N_H) + T\big) * N_S\Big) + S\Big) * N_w * N_{BPW} \qquad \text{Equation 3}$$

Where:

C = Requested Cylinder from the RPDC (Desired Cylinder) Register
T = Track/Surface/Head from the RPDA (Sector and Track) Register
S = Requested Sector from the RPDA (Sector and Track) Register
N_H = Number of Surfaces (or Heads or Tracks) on Disk Drive (RP06=19)
N_S = Number of Sectors per Cylinder (RP06=20)
N_W = Number of Words (36 bit) per Sector (Always 128)
N_{BPW} = Number of Bytes per 36 bit word (SIMH=8)

Note that the last two terms N_W * N_{BPW} = 1024 which is exactly two 512-byte SDHC Sectors. This is extremely fortunate because it enables the PDP10 sectors to be mapped to SDHC sectors. The reasoning for this assertion is discussed in section 12.6.3.

This can be re-written as:

$$SD\ Sector\ Address = \Big(\big((C * N_H) + T\big) * N_S\Big) + S\Big) * 2 \qquad \text{Equation 4}$$

Note, in the case of the RP06, the constant N_H is 19 and the constant N_S is 20. The multiplication by these two constants is troublesome but not impossible. The KS10 FPGA implements this algorithm using a state machine an repeated additions.

Because this Disk Simulator simulates disk motion, there are a lot of clock cycles available to perform repeated additions to implement this equation.

12.6.2 Cylinder/Head/Sector (CHS) Disk Address Increment

At the end of a transfer, the Disk Address is incremented according to the following algorithm. Note that this is consistent with the addressing described in the previous section.

```
if (sector == last_sector)
  begin
    sector <= 0;
    if (track == last_track)
      begin
        track <= 0
        cylinder <= cylinder + 1
      end
    else
      track <= track + 1
else
  sector <= sector + 1
end
```

Figure 112 – Sector Increment Algorithm

12.6.3 SIMH "Sector" Size

SIMH uses the UNIX `lseek(), read() and write()` file operations, to access the simulated disk. These operations provide no inherent limitation on read or write sizes or alignment to disk sector boundaries.

The PDP10 disk drives can read and write partial sectors – but only in a very limited sense. When a write of a partial sector is requested, the remainder of the sector is written with zeros. When a read of a partial sector is requested, only the requested data is written to memory. Whether or not the whole sector is read from the disk drive is unknown and is invisible to the system.

These PDP10 disk read and write properties can be replicated in the SD interface design.

SIMH maps a 36-bit data word into a 64-bit (8 byte) block on the disk. A hex dump of a chunk of a SIMH/PDP10 Disk Image is provided below. The bits representing the 36-bit data is highlighted in red.

```
001a040: 090f00d006000000 f9ff0b1008000000  ................
001a050: b12d00110b000000 b22d001104000000  .-.......-......
001a060: 8401800904000000 1100808904000000  ................
001a070: 004480c905000000 004080c905000000  .D.......@......
001a080: 2c970cc905000000 ff0100e908000000  ,...............
```

Figure 113 – SIMH/PDP10 Disk Image Hex Dump

Note: the disk data is in little-endian format. The first PDP10 data word shown in the hex dump above is decoded as $6D0000F09_{16}$ or 332000007411_{8}. This corresponds to a '`SKIPE 0,007411`' instruction. The second instruction is decode as $8100BFFF9_{16}$ or 402002777771_{8}. This corresponds to a '`setzm 0, 777771(2)`' instruction – or equivalently '`setzm 0, -7(2)`'.

While a SD sector is 512 bytes, a SIMH/PDP10 disk sector is actually 128 words * 8 bytes per word or 1024 bytes. Therefore a SIMH/PDP10 disk sector is exactly 2 SD sectors in length. Again this design detail can easily be incorporated into the SD interface design.

12.6.4 Disk Drive Parameters

The Tracks per Cylinder parameter is equivalent to the number of surfaces that contain usable data or the number of usable heads on the device. Any surfaces and/or heads used for servo control don't count as usable heads.

Table 86 below summarizes the disk parameters for some common disk drives.

Table 86 – Disk Parameters

Parameter	RM02/RM03	RM05	RM80	RP04/RP05	RP06	RP07
36-bit Words / Sector	128	128	128	128	128	128
Sectors / Track	30	30	30	20	20	43
Track / Cylinder	5	19	14	19	19	32
Cylinders / Pack *	823	823	559	411	815	630

Table 86 – Disk Parameters						
Parameter	**RM02/RM03**	**RM05**	**RM80**	**RP04/RP05**	**RP06**	**RP07**
PDP10 Disk Size (words)	15,801,600	60,046,080	30,051,840	19,991,040	39,641,600	110,960,640
SIMH Disk Size (bytes)	126,412,800	480,368,640	240,414,720	159,928,320	317,132,800	887,685,120

* Including FE cylinders.

The RP06 will be the first disk that is implemented. The code is designed to add other disks later.

See https://groups.google.com/forum/#!msg/alt.sys.pdp10/PmbYHKCUqmY/TRIFBkROulsJ for a discussion on RP07 support for TOPS-10 and Tops-20.

12.6.5 RPxx/RMxx Disk Addressing

These disks use Cylinder, Track, and Sector addressing. Because the disk selects the track by enabling the proper read/write head, this is roughly equivalent to the more commonly used (but later) Cylinder, Head, and Sector (CHS) addressing.

As an example, the RP06 has 5 platters. Each platter has 4 heads - which is a bit unusual in that each surface of the platter has two heads. One of the heads is a dedicated servo track therefore there are only 19 tracks available for data storage.

The RP06 has 20 sectors per track in an 18-bit mode and has 22 sectors per track in a 16-bit mode. Currently, the KS10 FPGA can only read/write data in an 18-bit mode; the 16-bit mode is only partly implemented as required for the DSRPA diagnostics.

The RP06 has 815 cylinders, 19 tracks per cylinder, and 20 sectors per track. The last 5 cylinders are dedicated to maintenance and are not used by the operating systems.

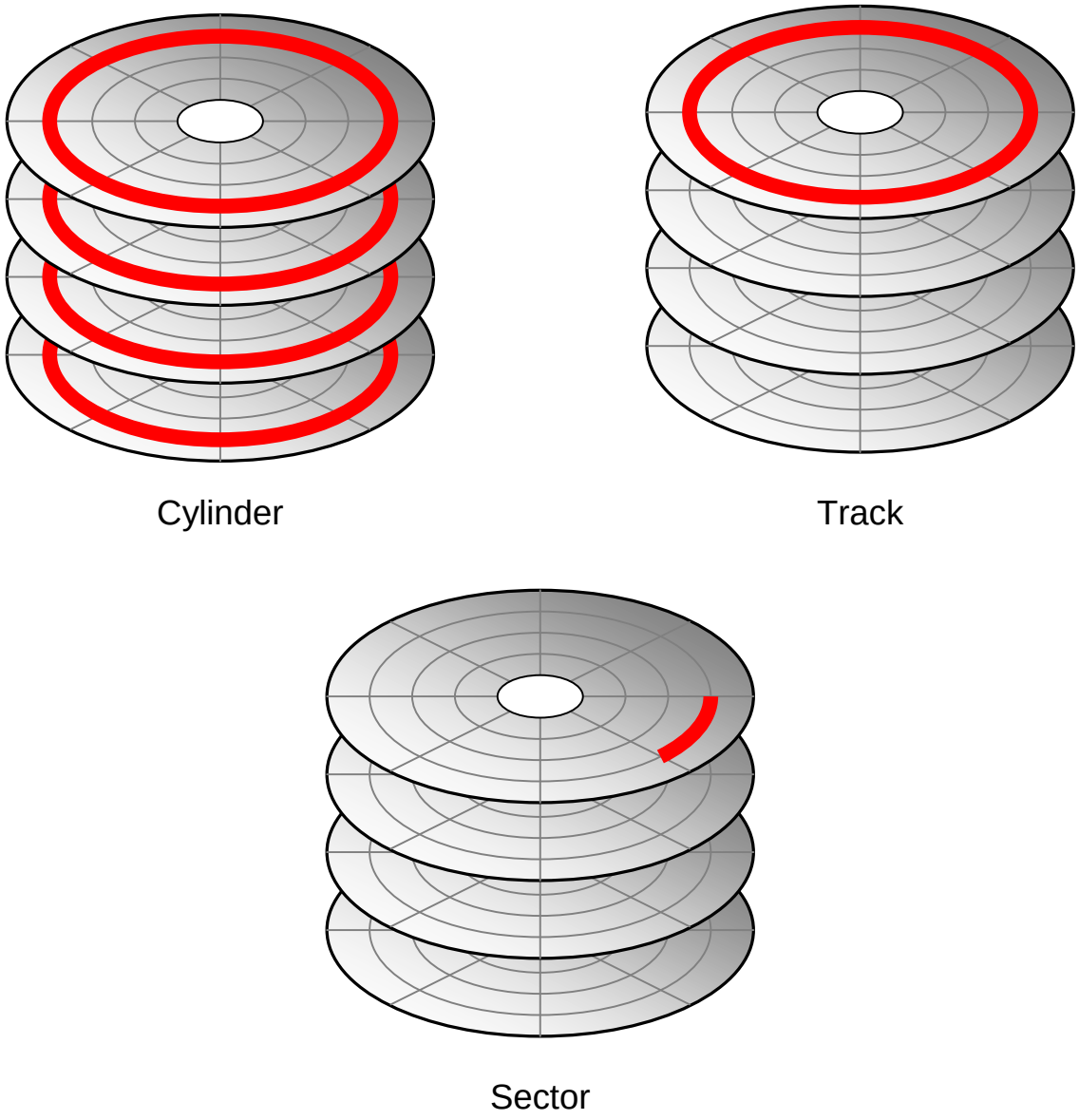


Figure 114 – Disk Cylinder, Track, and Sector

Like most modern disk drives, the SDHC card uses a linear sector address (or Logical Block Address) where the disk geometry is unknown and unimportant. The FPGA converts the RP06 CHS addressing to a linear sector address.

12.6.6 SD Disk Organization

For an RP06, the number of Sectors per Track is 20, the number of Tracks per Cylinder (Heads) is 19, the number of Cylinders is 815, and each sector requires two SD Sectors, then each RP06 disk will require 619,400 SD Sectors (about 317 MB) of storage.

For an RP07, the number of Sectors per Track is 43, the number of Tracks per Cylinder (Heads) is 32, the number of Cylinders is 630, and each sector requires two SD Sectors, then each RP07 disk will require 1,733,760 SD Sectors (about 847 MB) of storage.

If we align the start of each of the 8 disk drives to a 1 GB boundary, then any selection of disk drives can be accommodated by a single 8GB SD Card. This definition allows the sector address of the start of the disk to be constant regardless of the size or type of the disk being emulated.

This is illustrated below in Figure 115.

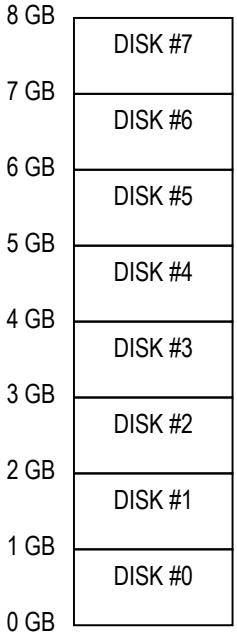


Figure 115 – SD Card Storage Allocation

13 LP20 Printer Controller

The LP20 Line Printer System is a Unibus-based hard-copy line printer system. In this implementation, the LP20 interfaces the KS10 FPGA IO Bus to a simulated LP26 Line Printer. This design is fully register compatible with the DEC LP20 and passes all relevant diagnostics. The interface to an external printer is via a simple full-duplex RS232 connection. Handshaking uses the XON/XOFF protocol.

13.1 LP20 Registers

This section provides programming and implementation details of the LP20 registers.

A summary of LP20 registers is shown below.

Table 87 - LP20 Register Summary

IO Addr (Dev 1)	IO Addr (Dev 2)	Register Name	Access	Read/ Write	Register Description
775400	775400	CSRA	Byte	R/W	Control/Status A Register
775402	775422	CSRB	Byte	R/W	Control/Status B Register
775404	775424	BAR	Word	R/W	Bus Address Register
775406	775426	BCTR	Word	R/W	Byte Count Register
775410	775430	PCTR	Word	R/W	Page Count Register
775412	775432	RAMD	Word	R/W	RAM Data Register
775414	775434	CBUF	Byte	R/W	Character Buffer Register
775415	775435	CCTR	Byte	R/W	Column Counter Register
775416	775436	PDAT	Byte	R	Printer Data Register
775417	775437	CKSM	Byte	R	Checksum Register

13.1.1 Control/Status A Register (CSRA)

The Control/Status A Register provide general printer control and status and is both byte and word addressable.

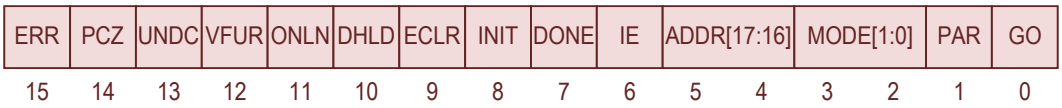


Figure 116 – Control/Status A Register (CSRA)

Table 88 – Control/Status A Register (CSRA) – IO Address 775400			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	ERR	R	Composite Error This bit is set when any of the following transition to active 1. Memory Parity Error (CSRB[MPE]), or 2. RAM Parity Error (CSRB[RPE]), or 3. Line Printer Parity Error (CSRB[LPE]), or 4. Unibus Time-out Error (CSRB[MTE]), or 5. Demand time-out Error (CSRB[DTE]), or 6. Printer Offline (CSRB[OFFL]), or 7. DAVFU Not Ready (CSRB[DVOF]), or 8. Go Error (CSRB[GOE]). This bit is cleared by: 1. Issuing and *Error Clear* (CSRA[ECLR] = 1), or 2. Issuing a *Controller Clear* (CSRA[INIT] = 1), or 3. Issuing an *IO* Bridge *Clear* (UBACSR[INI] = 1). Writes ignored.
14	PCZ	R	Page Counter Zero. This bit is set when the Page Counter is decremented to zero. This bit is cleared by: 1. Writing to the Page Count Register, or 2. Issuing a *Controller Clear* (CSRA[INIT] = 1), or 3. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1). Writes ignored
13	UNDC	R	Undefined Character This bit is set when: 1. Mode is Load RAM Mode (CSRA[MODE] = 3), and 2. DMA read occurs, and 3. One or more of the following conditions exists: a. The output of the Translation RAM indicates Interrupt asserted (RAMD[INT] = 1) and Translate negated (RAMD[TRANS] = 0), or b. The output of the Translation RAM indicates Interrupt asserted (RAMD[INT] = 1) and Delimiter Hold asserted (CSRA[DHLD] = 1) and Translate asserted (RAMD[TRANS] = 1). This bit is cleared by issuing a Go Command (CSRA[GO] = 1). Writes ignored.

Table 88 – Control/Status A Register (CSRA) – IO Address 775400			
Bit(s)	**Mnemonic**	**R/W**	**Description**
12	VFURDY	R	Direct Access Vertical Format Unit (DAVFU) Ready This bit is asserted when the DAVFU is loaded properly and is ready to use. The VFU is physically located in the printer. This bit is set when: 1. The printer is configured to use an Optical Vertical Format Unit (CSRB[OVFU] = 1), or 2. The printer is configured for a DAVFU and a START character, at least one data word (two bytes of DMA), and a STOP character is written to the DAVFU. This bit is cleared when: 1. The LP20 is configured to load the DAVFU (CSRA[MODE] = 2) and a START character is written to the DAVFU with no corresponding STOP character, or 2. The LP20 is configured to load the DAVFU (CSRA[MODE] = 2) and a START character is written to the DAVFU followed immediately by a STOP character (no data), or 3. The LP20 is configured to load the DAVFU (CSRA[MODE] = 2) and a STOP character is written to the DAVFU which is not preceded by a START character. 4. The LP20 is configured to load the DAVFU (CSRA[MODE] = 2) and a START character is written to the DAVFU and the DMA completes without transmitting a STOP character, or 5. The printer's VFU overruns the end of the DAVFU or OVFU tape. 6. The printer is re-configured from an Optical Vertical Format Unit to a DAVFU. The selection between an Optical Vertical Format Unit (OVFU) and Direct Access Vertical Format Unit (DAVFU) is controlled by the OVFU Field in the LP20 Console Control Register (LPCCR) (LPCCR{OVFU]). Therefor the console microcontroller can select between the two types of printers. The DAVFU is physically located in the printer; therefore, resetting the LP20 does not reset the DAVFU. Writes ignored.

Table 88 – Control/Status A Register (CSRA) – IO Address 775400			
Bit(s)	**Mnemonic**	**R/W**	**Description**
11	ONLINE	R	Online This bit indicates the printer online/offline status. This bit is set when the Console Microcontroller manually commands the printer to be online by asserting the LPCCR[SETONLN] bit in the LP20 Console Control Register. This bit is cleared when: 1. The Console Microcontroller manually commands the printer to be offline by asserting the LPCCR[SETOFFLN] bit in the LP20 Console Control Register, or 2. Any error programming the DAVFU. Specifically: a. Writing more than 144 words (288 bytes) to the DAVFU, or b. Writing an odd number of bytes to the DAVFU. c. The printer's VFU overruns the end of the OVFU tape or the end of the DAVFU. Writes ignored.
10	DHLD	R/W	Delimiter Hold This bit is set when the last received character was a Delimiter. This bit is set under the following conditions: 1. Writing a one to it, or 2. Set during a DMA read cycle as follows: a. DMA read cycle, and b. The Mode is Print Mode (CSRA[MODE] = 0) or the Mode is Test Mode (CSRA[MODE] = 1), and c. A Composite Error condition is not present (CSRA[ERR] = 0) , and d. The Delimiter Bit in the Translation RAM is set (RAMD[DEL] = 1). This bit is cleared when: 1. Writing a zero to it, or 2. Issuing a *Controller Clear* (CSRA[INIT] = 1), or 3. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1).

Table 88 – Control/Status A Register (CSRA) – IO Address 775400			
Bit(s)	**Mnemonic**	**R/W**	**Description**
9	ECLR	W	Error Clear Asserting this bit clears the following status bits: 1. Composite Error (CSRA[ECLR]), and 2. Go Bit (CSRA[GO]) 3. Memory Parity Error (CSRB[MPE]) 4. RAM Parity Error (CSRB[RPE]) 5. IO Bus Timeout Error (CSRB[MSYN]) 6. Go Error (CSRB[GOE]) This bit is always read as zero.
8	INIT	W	Controller Clear. Asserting this bit does the following: 1. Resets the Base Address Register (BAR) to zero, 2. Resets the Column Counter Register (CCTR) to zero, 3. Resets the Byte Counter Register (BCTR) to zero, 4. Reset the Page Counter Register (PCTR) to zero, 5. Clears the *Delimiter Hold* (CSRA[DHLD]), 6. Clears the *Interrupt Enable* (CSRA[IE]) 7. Clears the *Mode* (CSRA[MODE[3:2]]) 8. Clears the *Parity Test* (CSRA[PAR]) 9. Clears the *Go Bit* (CSRA[GO]) 10. Sets the *Done Status* (CSRB[DONE]) 11. Clears the *Test Mode* (CSRB[TEST[10:8]]) Setting this bit does not alter the Checksum Register (LPCKSM), the Character Buffer Register (LPCBUF), the Translation RAM, or the Translation RAM Address. This bit is always read as zero.
7	DONE	R	Done This bit indicates the status of the DMA controller. This bit is set when: 1. Byte Counter is incremented to zero, or 2. Issuing a *Controller Clear* (CSRA[INIT] = 1), or 3. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1). This bit is cleared when the Byte Counter is written. Writes are ignored.

Table 88 – Control/Status A Register (CSRA) – IO Address 775400				
Bit(s)	**Mnemonic**	**R/W**	**Description**	
6	IE	R/W	Interrupt Enable These bits are set by writing to this register. These bits are cleared by: 1. Writing to this register, or 2. Issuing a *Controller Clear* (CSRA[INIT] = 1), or 3. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1). When this bit is asserted, the following conditions will cause an interrupt: 1. Composite Error (CSRA[ERR] = 1), or 2. Page Zero (CSRA[PCZ] = 1), or 3. Undefined Characters (CSRA[UNDC] = 1), or 4. The DMA completes (CSRA[DONE] = 1). 5. DAVFU Ready (CSRA[DVON]) changes state, or 6. On-line (CSRA[ONLINE]) changes state.	
5-4	ADDR	R/W	Bus Address Extension [17:16] These bits are set by writing to this register. These bits are cleared by: 1. Writing to this register, or 2. Issuing a *Controller Clear* (CSRA[INIT] = 1), or 3. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1).	
3-2	MODE	R/W	Mode These bits are set by writing to this register. These bits are cleared by: 1. Writing to this register, or 2. Issuing a *Controller Clear* (CSRA[INIT] = 1), or 3. Issuing an IO *Bridge Clear* (UBACSR[INI] = 1).	
			0	Print Mode This mode allows data to be printed via DMA.
			1	Test Mode This mode allows data to be printed via DMA. Except it is never printed.
			2	Load DAVFU Mode This mode allows the DAVFU to be loaded via DMA.
			3	Load RAM Mode This mode allows the translation RAM to be loaded via DMA.

Table 88 – Control/Status A Register (CSRA) – IO Address 775400			
Bit(s)	**Mnemonic**	**R/W**	**Description**
1	PAR	R/W	Parity Test Enable When asserted, this bit enables the following parity errors to be reported as errors, otherwise they are ignored. 1. Translation RAM Parity Errors, and 2. Line Printer Parity Errors, and 3. Memory Parity Errors. This bit is set by writing one to this register. This bit is cleared by: 1. Writing zero to this register, or 2. Issuing a *Controller Clear* (CSRA[INIT] = 1), or 3. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1).
0	GO	R/W	Go. This bit starts a DMA transfer from KS10 memory to the LP20. This bit is set and DMA will start when: 1. Writing one to this register, and 2. Composite Error (CSRA[ERR]) is not asserted. This bit is cleared and DMA will stop when: 1. Writing zero to this register, or 2. The Byte Counter increments to zero (CSRA[DONE] = 1), or 3. The Page Counter decrements to zero (CSRA[PGZ] = 1),or 4. An Undefined Character (CSRA[UNDC]). If an Composite Error (CSRA[ERR]) condition exists at the time this command is issued: 1. the DMA operation will not start, and 2. the GO Error indication (CSRB[GOE]) will be indicated. Note: CSRA[GO] must be asserted after all of the other bits in CSRA are set to the desired state. Changing the register contents simultaneously with setting the GO bit can result in undefined behavior.

13.1.2 Control/Status B Register (CSRB)

The Control/Status B Register is byte addressable.

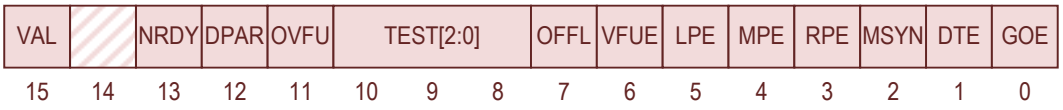


Figure 117 – Control/Status B Register (CSRB)

Table 89 – Control/Status B Register (CSRB) – IO Address 775402			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	VAL	R	Valid Data This bit toggles with each character that is sent to the printer. This bit is held clear when DMA is not active (CSRA[GO] = 0). This bit toggles when DMA is active and a character is sent to the printer.
14	-	R	Reserved. Always read as zero.
13	NRDY	R	Printer Not Ready According to the LP26 manual, this bit is NEGATED when: 1. Power and DC voltage are up, and 2. All interlocks are closed, and 3. Paper has been loaded, and 4. No printer faults are present, and 5. The alarm indicator is off. This is not implemented. The printer is always ready and this bit is always read as zero. Writes are ignored.
12	DPAR	R	LPT Data Parity This bit reflects the printer data parity as sent to the printer. The printer expects odd parity; therefore, if the data has an even number of bits set to ‘1’, the parity (reflected in the CSRB[DPAR] bit) will be ‘1’. The parity is inverted in LPT Parity Test Mode (CSRB[TEST] == 5). This is used to test the parity. Writes are ignored.
11	OVFU	R	Optical vertical format unit This OVFU bit is asserted when an Optical Vertical Format Unit is installed. This bit reflects the LPCCR[OVFU] configuration parameter of the LP20 Console Control Register (LPCCR).

Table 89 – Control/Status B Register (CSRB) – IO Address 775402				
Bit(s)	**Mnemonic**	**R/W**	**Description**	
10-8	TEST	R/W	Test Mode	
			0	Normal operation
			1	DEM Time Test
			2	MSYN Time Test
			3	RAM Parity Test
			4	Memory Parity Test
			5	LPT Parity Test
			6	Page Counter Test
			7	Not used. Does nothing.
7	OFFLINE	R	Off-line This bit reflects the negation of the Online status (CSRA[ONLINE]). Refer there for a description of this bit.	
6	VFUE	R	DAVFU Error This is the negation of the Direct Access Vertical Format Unit (DAVFU) Ready (CSRA[VFURDY]). Refer there for a description of this bit.	
5	LPE	R	Line Printer Parity Error This bit would normally indicate parity error when transferring data to the line printer. Line printer parity is not implemented in the KS10 FPGA however this is implemented as required to pass the diagnostic tests. This bit is set when: 6. The controller is in LPT Parity Test Mode (CSRB[TEST] = 5), and 7. DMA Data is written to the printer, and 8. Parity Test is enabled (CSRA[PAR] = 1) This bit is cleared when: 1. Parity Test is disabled (CSRA[PAR] = 0) or 2. The controller is no longer in LPT Parity Test Mode (CSRB[TEST] !=5) and DMA Data is written to the printer Writes are ignored. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1), *Controller Clear* (CSRA[INIT] = 1), or *Error Clear* (CSRA[ECLR] = 1) does not clear LPE.	

Table 89 – Control/Status B Register (CSRB) – IO Address 775402			
Bit(s)	**Mnemonic**	**R/W**	**Description**
4	MPE	R	Memory Parity Error This bit would normally indicate a memory parity error during a DMA operation. Memory parity is not implemented in the KS10 FPGA but this is implemented as required to pass the diagnostic tests. This bit is set when: 1. Parity Tests are enabled (CSRA[PAR] = 1), and 2. The Mode is set to *Load RAM Mode* (CSRA[MODE] = 3), and 3. The Selected Test is Memory Parity Test (CSRB[TEST] = 4), and 4. A DMA cycle is issued (CSRA[GO] = 1) which accesses memory. This bit is cleared when 1. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1), or 2. Issuing a *Controller* Clear (CSRA[INIT] = 1), or 3. Issuing an *Error Clear* (CSRA[ECLR] = 1).
3	RPE	R	RAM Parity Error This bit indicates a parity failure of the Translation RAM. Note: The parity stored in the Translation RAM is inverted by setting the Test Mode to RAM Parity (CSRB[TEST] = 3). Similarly, the parity read from the Translation RAM is inverted by setting the Test Mode to RAM Parity (CSRB[TEST] = 3). Errors do not occur when the data is written and read with the same state of the RAM Parity Test Mode. This bit is set by: 1. Parity Test are enabled (CSRA[PAR] = 1) , and 2. Not in Load DAVFU Mode (CSRA[MODE] != 2), and 3. Enabling Parity Test (CSRA[PAR] = 1) , and 4. Reading RAM data which was stored with inverted parity. This occurs when the data is written with the RAM Test mode in one state and is read with the RAM Test mode in the other state. This bit is cleared by: 1. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1), or 2. Issuing a Controller *Clear* (CSRA[INIT] = 1), or 3. Issuing an Error Clear (CSRA[ECLR] = 1). Writes are ignored.

Table 89 – Control/Status B Register (CSRB) – IO Address 775402			
Bit(s)	**Mnemonic**	**R/W**	**Description**
2	MSYN	R	IO Bus Time-out Error This bit is set if the LP20 issues a bus request that is not acknowledged. This bit is set by: 1. Enabling Parity Test (CSRA[PAR] = 1) , and 2. Enabling Test Mode (CSRA[MODE] = 2), and 3. Setting the Test to MSYN Time Test (CSRB[TEST] = 2), and 4. Issuing a DMA operation (CSRA[GO] = 1). This bit is cleared by: 1. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1), or 2. Issuing a Controller *Clear* (CSRA[INIT] = 1), or 3. Issuing an Error Clear (CSRA[ECLR] = 1). Writes are ignored.
1	DTE	R	Demand Time-out Error This bit would normally indicate a handshaking timeout issue between the LP20 and the printer. This interface is not implemented in the KS10 FPGA. This bit is set by: 1. Enabling Test Mode (CSRA[MODE] = 2), and 2. Setting the Test to Demand Time Test (CSRB[TEST] = 1) This bit is cleared by: 1. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1), or 2. Issuing a Controller *Clear* (CSRA[INIT] = 1), or 3. Issuing an Error Clear (CSRA[ECLR] = 1). Writes are ignored.

Table 89 – Control/Status B Register (CSRB) – IO Address 775402			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0	GOE	R	Go Error This bit is set when an Composite Error (CSRA[ERR]) is present and a GO Command (CSRA[GO]) is issued. This bit is cleared by: 1. Writing to this register, or 2. Issuing an *IO Bridge Clear* (UBACSR[INI] = 1), or 3. Issuing a *Controller Clear* (CSRA[INIT] = 1) The Go Error Indication is read-only.
		W	Page Decrement This is unrelated to the Go Error status described above: when the LP20 is in Page Counter Test Mode ((CSRA[MODE] = 1) and (CSRB[TEST] = 6)), asserting this bit will decrement the Page Counter in the Page Count Register. This magic test mode is write-only.

13.1.3 Bus Address Register (BAR)

The Bus Address Register is only word addressable.

The Bus Address Register contains the virtual address of the DMA data. This virtual address is translated to a physical address by the IO Bus Bridge (UBA).

The Bus Address Register is written under the following conditions:

1. Cleared by issuing a Controller Clear (CSRA[INIT] = 1), or
2. Cleared by issuing an IO Bridge Clear (UBACSR[INI] = 1), or
3. Modified by a program write to the BAR register, or
4. Incremented in every DMA mode after each DMA transaction.

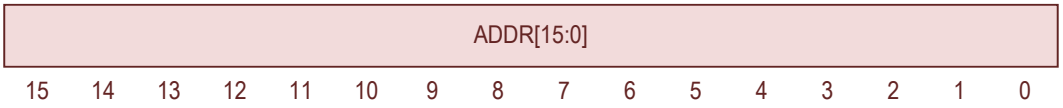


Figure 118 – Bus Address Register (BAR)

Table 90 – Bus Address Register (BAR) – IO Address 775404			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15:0	ADDR	R/W	Bus Address

13.1.4 Byte Count Register (BCTR)

The Byte Count Register is word addressable.

The Byte Count Register is use to control the length of a DMA operation. The twos-complement of the number of bytes to load is written into the BCTR. Each DMA cycle increments the BCTR. When the BCTR increments to zero, the DMA operation is complete.

Figure 119 – Byte Count Register (BCTR)

Table 91 – Byte Count Register (BCTR) – IO Address 775406			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15:12	-	R	Reserved
11:0	COUNT	R/W	Byte Counter

13.1.5 Page Count Register (PCTR)

The Page Count Register is word addressable.

The Page Count is loaded with a prescribed number of pages – presumably the number of pages in a box of fan-fold paper. As each page is printed, the PCTR is decremented. When the PCTR is decremented to zero, the Page Count Zero (CSRA[PCZ]) bit is asserted and an interrupt is generated. Again, presumably to wake up the operator and change the paper.

Figure 120 – Page Count Register (PCTR)

Table 92 – Page Count Register (PCTR) – IO Address 775410			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15:12	-	R	Reserved
11:0	COUNT	R/W	Page Counter

13.1.6 RAM Data Register (RAMD)

The RAM Data Register is word addressable.

The RAM Data Register is written under the following conditions:

1. Program write to the RAMD register, or
2. DMA read in Load RAM Mode (CSRA[MODE] = 3)

The RAM Address Register is modified under the following conditions:

1. Cleared by issuing a Go Command (CSRA[GO] = 1) in Load RAM Mode (CSRA[MODE] = 3)
2. Cleared by issuing a Controller Clear (CSRA[INIT] = 1), or
3. Cleared by issuing an IO Bridge Clear (UBACSR[INI] = 1), or
4. Modified by a write to the CBUF register, or
5. Modified by a DMA write in Print Mode (CSRA[MODE] = 0), or
6. Modified by a DMA write in Test Mode (CSRA[MODE] = 1), or
7. Incremented in Load RAM Mode (CSRA[MODE] = 3) after each DMA transaction.

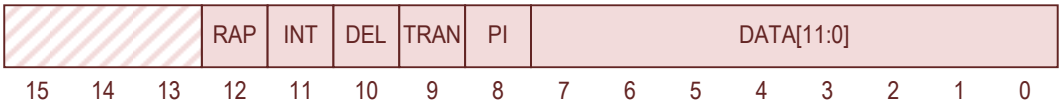


Figure 121 – RAM Data Register (RAMD)

Table 93 – RAM Data Register (RAMD) – IO Address 775412			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15:13	-	R	Reserved
12	RAP	R	RAM Parity
11	INT	R/W	Interrupt Bit. When asserted, causes the controller to generate an interrupt to the processor instead of generating a data strobe to the line printer.
10	DEL	R/W	Delimiter Bit When asserted causes the current and next printer data characters to be taken from the Translation RAM (RAMD) instead of the character buffer (CBUF).
9	TRAN	R/W	Translate Bit When asserted causes the character in RAM be sent to the printer otherwise the character in the character buffer is sent to the printer.

Table 93 – RAM Data Register (RAMD) – IO Address 775412			
Bit(s)	**Mnemonic**	**R/W**	**Description**
8	PI	R/W	Paper Instruction Bit When asserted causes the printer to interpret the character as a carriage control character rather than data to be printed.
7:0	DATA	R/W	RAM Data

13.1.7 Column Counter Register (CCTR) / Character Buffer Register (CBUF)

The Column Counter Register and Character Buffer Register are byte addressable.

The column counter is normally used by the DEC LP20 to implement tab characters. When a tab character is found, the LP20 replaces the tab character with a sequence of 1 to 8 spaces. This is not implemented in the KS10 FPGA.

The DEC LP20 also causes the printer to wrap text to the next line after column 132. This is properly implemented.

The Character Buffer Register is written under the following conditions:

1. Cleared by issuing a Controller Clear (CSRA[INIT] = 1), or
2. Cleared by issuing an IO Bridge Clear (UBACSR[INI] = 1), or
3. Modified by a program write to the CBUF register, or
4. Modified by a DMA write in Print Mode (CSRA[MODE] = 0), or
5. Modified by a DMA write in Test Mode (CSRA[MODE] = 1), or
6. Modified by a DMA write in Load DVFU Mode (CSRA[MODE] = 2).

It is not altered when loading Translation RAM via DMA.

Figure 122 – Column Counter Register (CCTR) / Character Buffer Register (CBUF)

Table 94 – CCTR and CBUF Register Column Counter Register (CCTR) – IO Address 775414 Character Buffer Register (CBUF) – IO Address 775415			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15:8	CCTR	R/W	Column Counter
7:0	CBUF	R/W	Character Buffer

13.1.8 Checksum Register (CKSM) / Printer Data Register (PDAT)

The Checksum Register and Printer Data Register are byte addressable.

The Checksum Register is used to verify the integrity of DMA transfers. The Checksum Register is set to zero when the DMA operation starts and the register accumulates the data bytes during the DMA operation. When the DMA operation has completed, the checksum is reported in this register.

Figure 123 – Printer Data Register (PDAT) / Checksum Register (CKSM)

The last character written to the printer either from DMA or via writes to the Character Buffer Register is captured in the Printer Data Register.

Table 95 – PDAT and CKSM Registers
Printer Data Register (PDAT) – IO Address 775416
Checksum Register (CKSM) – IO Address 775417

Bit(s)	Mnemonic	R/W	Description
15:8	CKSM	R	Checksum
7:0	PDAT	R	Printer Data

13.2 LP20 Interrupts

13.3 LP20 Modes

The LP20 has four major modes: Print Mode, Test Mode, Load DAVFU Mode, and Load RAM Mode. These modes are described in the following sections.

13.3.1 Print Mode

The *Print Mode* is the normal mode of operation.

13.3.2 Test Mode

In *Test Mode*, the print characters are not printed and the printer handshake lines are looped back to the LP20. This mode behaves as-if the printer output was routed to the bit-bucket – even if a printer is not present. Optionally other hardware tests are enabled for hardware testing.

These optional tests are described in the following sections.

13.3.2.1 Normal Test Mode

In *Normal Test Mode*, no additional test modes are enabled. As stated above, nothing is output to the printer.

13.3.2.2 Demand Timeout Test Mode

The *Demand Timeout Test Mode* disables the handshake acknowledge from the printer. This causes the printer interface to timeout and assert a Demand Timeout Error (CSRB[DTE]).

In the KS10 FPGA, the Demand Timeout Error is asserted while the unit is in *Demand Timeout Test Mode.*

13.3.2.3 SSYN Timeout Test Mode

The *SSYN Timeout Test Mode* disables the DMA acknowledge from the memory subsystem. This will cause the DMA controller to timeout and assert a IO Bus Time-out Error (CSRB[MSYN]).

13.3.2.4 RAM Parity Test Mode

In *RAM Parity Test Mode*, the parity written to the RAM and the parity read from RAM are inverted.

A RAM parity error will occur if the RAM is written with the *RAM Parity Test Mode* in one state and the RAM is read with the *RAM Parity Test Mode* in other state.

13.3.2.5 Memory Parity Test Mode

Memory parity is not implemented. In *Memory Parity Test Mode*, every DMA access of memory will generate a Memory Parity Error (CSRB[MPE]).

13.3.2.6 Line Printer Parity Test Mode

Line printer parity is not implemented. In *Line Printer Parity Test Mode*, every printed character will generate a Line Printer Parity Error (CSRB[LPE]).

13.3.2.7 Page Counter Test Mode

The *Page Counter Test Mode* allows the Page Counter hardware to be tested and is enabled by setting the controller into Test Mode (set CSRA[MODE[2:0]] = 2) and selecting the Page Counter Test Mode (CSRB[TEST[2:0]]) = 6).

In this mode, Page Counter is incremented every time a ‘1’ written to the Go Error bit in the CSRB (CRSB[GOE] = 1).

13.3.3 Load DAVFU Mode

The *Load DAVFU Mode* allows the DAVFU to be updated via DMA.

13.3.4 Load RAM Mode

The *Load RAM Mode* allows the Translation RAM to be updated via DMA.

14 LP26 Line Printer

The LP26 was a DEC badged Dataproducts B600 printer. Quoting the service manual, the “B-series printers are general purpose, continuous steel band, solid font impact printers designed as output devices for use with electronic information processing systems such as data communications/data entry terminals and dedicated minicomputer-based systems where reliable operation in a medium duty cycle environment is required. They are designed to provide a throughput of 300 or 600 lines per minute in a typical printing application, using a 64-character ASCII character set.”

14.1 Vertical Format Units

The LP20 supports interfaces to a number of DEC Printers – most of these were made by Dataproducts Corp. These are:

Printer	Dataproducts Type	VFU Type	Comment
LP05			
LP07			
LP10		OVFU	
LP14			
LP26			
LA180			

There are probably more, but these are the ones that a specifically mentioned in the LP20 documentation.

14.1.1 Tape Controlled Vertical Format Unit (TCVFU)

The Tape Controlled Vertical Format Unit used a Punched Tape to set the vertical formatting. The TCVFU was re-programmed by swapping the tape. In the LP26, the vertical formatting data was from read from tape and processed digitally.

14.1.2 Direct Access Vertical Format Unit (DAVFU)

The DAVFU was fully write-only software programmable. Like the TCVFU, the DAVFU supported 144 lines and 12 channels.

This section documents the Dataproducts Direct Access Vertical Format Unit (DAVFU) as implemented in the KS10 FPGA.

14.1.2.1 DAVFU Loading

Data is loaded into the DAVFU started with a Start Load Code, a sequence of data bytes, followed by a Stop Load Code. The DAVFU programming works on 12-bit words, so two bytes of data are required to create a single DAVFU word.

Data is loaded into the DAVFU, low byte first followed by the high byte. Only the six LSBs from each word are used.

As the DAVFU is loaded with data, the DAVFU keeps track of the number of lines on the page. Any attempt to overrun the page size will result in the printer going OFFLINE and indicating that the DAVFU is no longer ready (CSRA[DAVFU] = 0). This is tested by the printer diagnostics DSLPA Test 114.

14.1.2.1.1 DAVFU Start Load Codes

The DVFU Start Load Code of 154 (octal) with the PI line asserted initiates the DVFU memory load routine using 6 LPI as the line spacing, regardless of the current printer line spacing.

The DVFU Start Load Code of 155 (octal) with the PI line asserted initiates the DVFU memory load routine using 8 LPI as the line spacing, regardless of the current printer line spacing.

The DVFU Start Load Code of 156 (octal) with the PI line asserted initiates the DVFU memory load routine using the current printer line spacing as the DVFU line spacing.

Note: When the DVFU is being loaded, the data bits transferred from the LP20 to the LP26 printer are munged as follows:

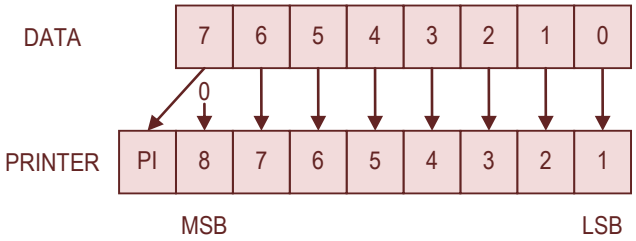


Figure 124 – Printer Data Munging

Note: Not only are the bits munged, the bits are renumbered from [7:0] to [8:1] at the printer interface.

Because the MSB of the data is used to set the PI bit, the apparent start codes as seen by the KS10 software are 354, 355, and 356 and these are the values you will see in the Diagnostics and in the Monitor software. Refer to Table 96 and Table 97 below for DVFU commands.

The KS10 FPGA doesn't really implement the notion of LPI. All Start Codes are treated identically.

14.1.2.1.2 DAVFU Stop Load Codes

The DVFU Stop Load Code of 157 (octal) with the PI line asserted terminates the DVFU memory load routine. Per all the reasons describe in the previous section, the apparent Stop Code is 357 (octal).

14.1.2.2 DAVFU Use

The DAVFU can generate absolute page motion (go directly to a position on the paper) or relative page motion (slew some lines down on the page).

These are described in the following sections.

14.1.2.2.1 Absolute Motion (Channel Codes)

In this mode, the DAVFU moves directly to the next line in the form having the specified channel number.

This occurs when the RAMD[PI] bit is asserted, and data bit-5 of the data (SLEW) is not asserted.

Under these conditions, the four least significant bits of the data specify the DVFU channel number. The DVFU moves the paper downward, on line at a time, until the associated channel is found. The channel number must be between 1 and 12 else and error condition exists.

The channel numbers are decoded as follows:

Table 96 – Channel Commands

Octal	PI	Data Bits								Channel
		8	7	6	5	4	3	2	1	
000	1	x	x	x	0	0	0	0	0	1
001	1	x	x	x	0	0	0	0	1	2
002	1	x	x	x	0	0	0	1	0	3
003	1	x	x	x	0	0	0	1	1	4
004	1	x	x	x	0	0	1	0	0	5
005	1	x	x	x	0	0	1	0	1	6
006	1	x	x	x	0	0	1	1	0	7
007	1	x	x	x	0	0	1	1	1	8
010	1	x	x	x	0	1	0	0	0	9
011	1	x	x	x	0	1	0	0	1	10
012	1	x	x	x	0	1	0	1	0	11
013	1	x	x	x	0	1	0	1	1	12
014	1	x	x	x	0	1	1	0	0	13*
015	1	x	x	x	0	1	1	0	1	14*
016	1	x	x	x	0	1	1	1	0	15*
017	1	x	x	x	0	1	1	1	1	16*

*Note: This setting is invalid. Using this setting will result in the Printer going OFFLINE and indicating DAVFU not ready (CSRA[VFURDY] = 0)

Channel commands move perform the vertical motion before printing the buffer.

14.1.2.2.2 Relative Motion (Slew)

Another method of moving paper using the PI line results in vertical slews of a specified number of lines within the form, relative to the current print line.

This occurs when the RAMD[PI] bit is asserted, and data bit-5 of the data (SLEW) is asserted.

Under these conditions, the four least significant bits of the data specify the number of lines to slew relative to the current line on the paper. This relative motion will occur even the DAVFU is un-programmed. The channel numbers are decoded as follows:

Table 97 – Slew Commands										
Code (Octal)	PI	Data Bits								Lines Slewed
		8	7	6	5	4	3	2	1	
000	1	x	x	x	1	0	0	0	0	0 (CR) *
001	1	x	x	x	1	0	0	0	1	1
002	1	x	x	x	1	0	0	1	0	2
003	1	x	x	x	1	0	0	1	1	3
004	1	x	x	x	1	0	1	0	0	4
005	1	x	x	x	1	0	1	0	1	5
006	1	x	x	x	1	0	1	1	0	6
007	1	x	x	x	1	0	1	1	1	7
010	1	x	x	x	1	1	0	0	0	8
011	1	x	x	x	1	1	0	0	1	9
012	1	x	x	x	1	1	0	1	0	10
013	1	x	x	x	1	1	0	1	1	11
014	1	x	x	x	1	1	1	0	0	12
015	1	x	x	x	1	1	1	0	1	13
016	1	x	x	x	1	1	1	1	0	14
017	1	x	x	x	1	1	1	1	1	15

*Note: Treated as a Carriage Return.

14.1.2.3 Error Conditions

ANSI compliant printers support up to 143 lines-per-page. Attempting to load more than 143 lines will result in the printer going off-line and asserting a not ready DAVFU status (CSRA[DAVFU] = 0) indication. This length is not arbitrary but is consistent with the ANSI standard and is tested by the DSLPA Diagnostic Test 112.

The DAVFU memory can be cleared (and the DAVFU made “Not Ready” by any of the following methods:

1. Sending two Start Load Codes, or
2. Sending a Start Load Code with no Stop Load Code, or
3. Sending a Start Load Code followed immediately by a Stop Load Code with no data.

15 Executive Mode and IO Instructions

This section of this document describes the operation of the Executive Mode and I/O instructions on the KS10 platform.

OPCODE Assignment Map								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
700	APR0	APR1	APR2	-	-	-	-	-
710	TIOE	TION	RDI0	WRIO	BSIO	BCIO	-	-
720	TIOEB	TIONB	RDIOB	WRIOB	BSIOB	BCIOB	-	-
730	-	-	-	-	-	-	-	-
740	-	-	-	-	-	-	-	-
750	-	-	-	-	-	-	-	-
760	-	-	-	-	-	-	-	-
770	-	-	-	-	-	-	-	-

AC Field Assignments			
AC	**700**	**701**	**702**
00	APRID (BLKI APR)	UUO	RDSPB
01	UUO	RDUBR (DATI PAG)	RDCSB
02	UUO	CLRPT	RDPUR
03	UUO	WRUBR (DATO PAG)	RDCSTM
04	WRAPR (CONO APR)	WREBR (CONO PAG)	RDTIM
05	RDAPR (CONI APR)	RDEBR (CONI PAG)	RDINT
06	UUO	UUO	RDHSB
07	UUO	UUO	UUO
10	UUO	UUO	WRSPB
11	UUO	UUO	WRCSB
12	UUO	UUO	WRPUR
13	UUO	UUO	WRCSTM
14	WRPI (CONO PI)	UUO	WRTIM
15	RDPI (CONI PI)	UUO	WRINT
16	UUO	UUO	WRHSB

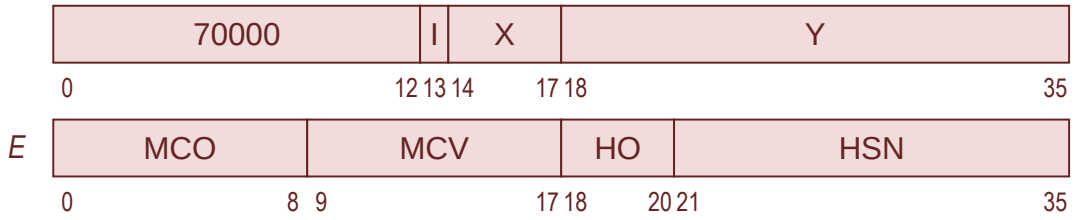
17	UUO	UUO	UUO

15.1 Executive Mode Instructions

15.1.1 Arithmetic Processor Interface (APR) Instructions

15.1.1.1 APR Identification (APRID/BLKI APR)

This instruction returns the microcode version number and the CPU serial number.



The APRID fields are controlled solely by the microcode. The following table describes the result of executing the APRID instruction.

Figure 125 – APRID (BLKI APR) Instruction

Table 98 – APRID (BLKI APR) Bit Definitions

Bit(s)	Mnemonic	Description						
		Bit	Mnemonic	Microcode Option	KS10 Value	T10KI Value	T10KL Value	CRAM4K Value
0-8	MCO	0	INHCST	Inhibit CST update is available	0	0	1	1
		1	NOCST	No CST at all	0	0	0	0
		2	NONSTD	Non Standard Microcode	0	0	0	0
		3	UBABLT	UBABLT instructions available	0	1	1	1
		4	KI Paging	KI Paging is present	1	1	0	1
		5	KL Paging	KL Paging is present	1	0	1	1
		6	Spare	Undefined	0	0	0	0
		7	Spare	Undefined	0	0	0	0
		8	Spare	Undefined	0	0	0	0
		Summary			030 octal	060 octal	450 octal	470 octal

Table 98 – APRID (BLKI APR) Bit Definitions						
Bit(s)	**Mnemonic**	**Description**				
9-17	MCV	Microcode Version	130 octal	130 octal	130 octal	130 octal
18-20	HO	Hardware Options	0	0	0	0
21-35	HSN	HW Serial Number (CPU#)	4097 decimal	4097 decimal	4097 decimal	4097 decimal

15.1.1.2 Write APR (WRAPR/CONO APR)

This instruction controls the Arithmetic Processor (APR) or CPU.

This immediate instruction decodes its effective address to control the processor. The effective address bits are used as follows:

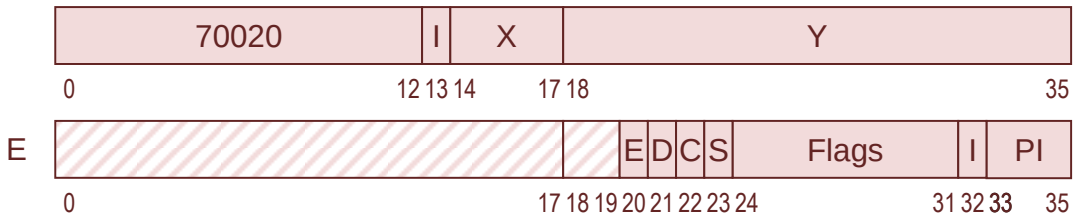


Figure 126 – WRAPR (CONO APR) Instruction

Table 99 – WRAPR (CONO APR) Bit Definitions		
Bit(s)	**Description**	
0-17	Ignored	
18-19	Ignored	
20	Enable selected flags	
21	Disable selected flags	
22	Clear selected flags	
23	Set selected flags	
24	Flags	Not implemented.
25		Interrupt to console.

Table 99 – WRAPR (CONO APR) Bit Definitions		
Bit(s)	**Description**	
26		Implemented. (Was power failure on KS10)
27		Implemented. (Was non-existent memory on KS10)
28		Implemented. (Was uncorrectable memory error on KS10)
29		Implemented. (Was correctable memory error on KS10)
30		Interval Timer
31		Interrupt from console
32	Generate interrupt request	
33-35	Interrupt priority	

All interrupt channels are implemented as described above. Only the interrupt to console, interval timer, and the interrupt from console create interrupts generated by external events. All of the others may be used by software.

Note: The result of setting both bits 20 and 21 or 22 and 23 is indeterminate.

15.1.1.3 Read APR (RDAPR/CONI APR) conditions

This instruction stores the Arithmetic Processor (APR) status in the word addressed by E. The status is as follows:

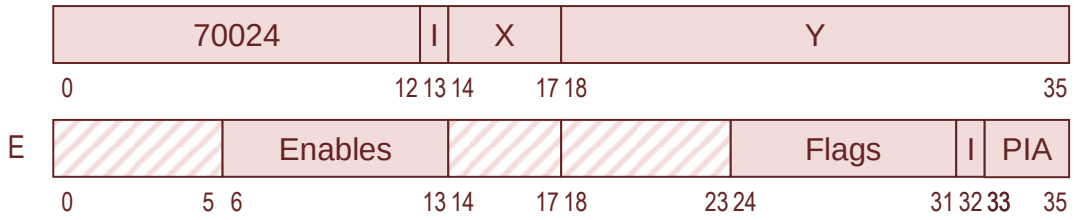


Figure 127 – RDAPR (CONI APR) Instruction

Table 100 – RDAPR (CONI APR) Bit Definitions		
Bit(s)	**Description**	
0-5	Read as zero	
6	Enables	Not implemented.
7		Interrupt to console. Always read as zero.

Table 100 – RDAPR (CONI APR) Bit Definitions		
Bit(s)	**Description**	
8		Implemented. (Was power Failure on KS10)
9		Implemented. (Was non-existent memory on KS10)
10		Implemented. (Was uncorrectable memory error on KS10)
11		Implemented. (Was correctable memory error on KS10)
12		Interval timer
13		Interrupt from console
14-17	Read as zero	
18-23	Read as zero	
24	Flags	Not implemented.
25		Interrupt to console. Always read as zero.
26		Implemented. (Was power Failure on KS10)
27		Implemented. (Was non-existent memory on KS10)
28		Implemented. (Was uncorrectable memory error on KS10)
29		Implemented. (Was correctable memory error on KS10)
30		Interval timer
31		Interrupt from console
32	Some flag (bit 24 to bit 31) is currently requesting an interrupt.	
33-35	Interrupt priority	

All interrupt channels are implemented as described above. Only the interval timer and the console create interrupts generated by external events. All of the others may be used by software.

15.1.2 Priority Interrupt Controller (PI) Instructions

15.1.2.1 Write Priority Interrupt (WRPI/CONO PI)

This instruction configures the Priority Interrupt Controller according to E:

The action of the processor is not defined when both 25 and 26 or 22 and 24 are set in the same instruction

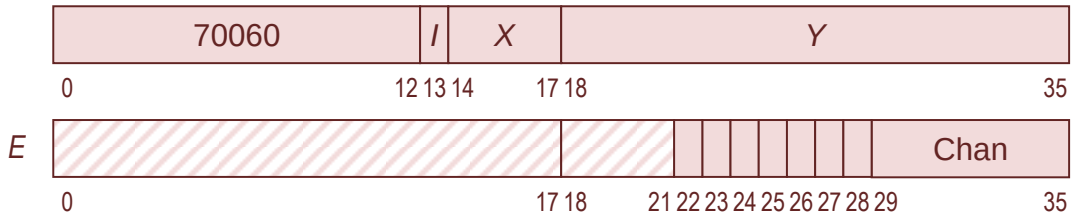


Figure 128 – WRPI (CONO PI) Instruction

Table 101 – WRPI (CONO PI) Bit Definitions		
Bit(s)	**Description**	
0-17	Ignored	
18-21	Ignored	
22	Clear interrupt on selected channel	
23	Clear PI system	
24	Initiate interrupt on selected channel	
25	Enable selected channel (bits 29-35)	
26	Disable selected channel (bits 29-35)	
27	Turn off the PI system	
28	Turn on the PI system	
29	Select	Channel 1
30		Channel 2
31		Channel 3
32		Channel 4
33		Channel 5
34		Channel 6
35		Channel 7

15.1.2.2 Read Priority Interrupt (RDPI/CONI PI)

This instruction stores the PI status in the word addressed by E.

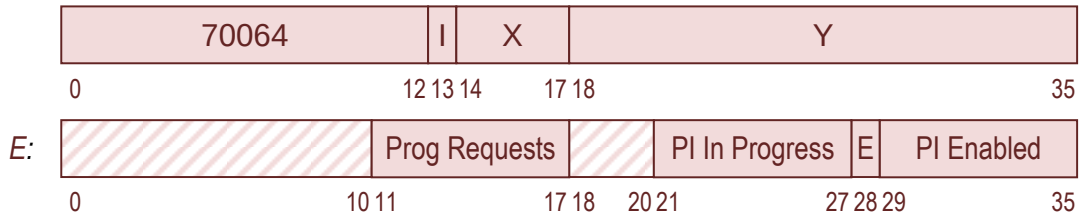


Figure 129 – RDPI (CONI PI) Instruction

Table 102 – RDPI (CONI PI) Bit Definitions

Bit(s)	Description	
0-10	Ignored	
11	Program Request	Channel 1
12		Channel 2
13		Channel 3
14		Channel 4
15		Channel 5
16		Channel 6
17		Channel 7
18-20	Ignored	
21	PI in Progress	Channel 1
22		Channel 2
23		Channel 3
24		Channel 4
25		Channel 5
26		Channel 6
27		Channel 7
28	PI system is enabled	
29	PI Enabled	Channel 1
30		Channel 2
31		Channel 3

Table 102 – RDPI (CONI PI) Bit Definitions		
Bit(s)	**Description**	
32		Channel 4
33		Channel 5
34		Channel 6
35		Channel 7

15.1.3 User Base Register (UBR) Instructions

15.1.3.1 Write to the User Base Register (WRUBR/DATO PAG)

This instruction loads the User Base Register from E. This operation invalidates the cache and clears the valid bits in the page tables.

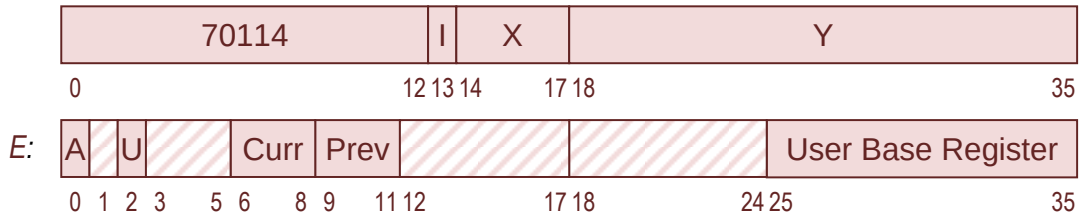


Figure 130 – WRUBR (DATO PAG) Instruction

Table 103 – WRUBR (DATO PAG) Bit Definitions		
Bit(s)	**Bit**	**Description**
0	A	Load the current and previous context AC blocks specified by bits 6-8 and 9-11 of E, respectively
1	0	Ignored
2	U	Load bits 25-35 into bits 16-26 in the User Base Register (UBR)
3-5	0	Ignored
6-8	Curr	Current AC Block
9-11	Prev	Previous Context AC Block
12-17	0	Ignored
18-24	0	Ignored

Table 103 – WRUBR (DATO PAG) Bit Definitions		
Bit(s)	**Bit**	**Description**
25-35	UBR	User base Register (Page Number)

15.1.3.2 Read User Base Register (RDUBR/DATI PAG)

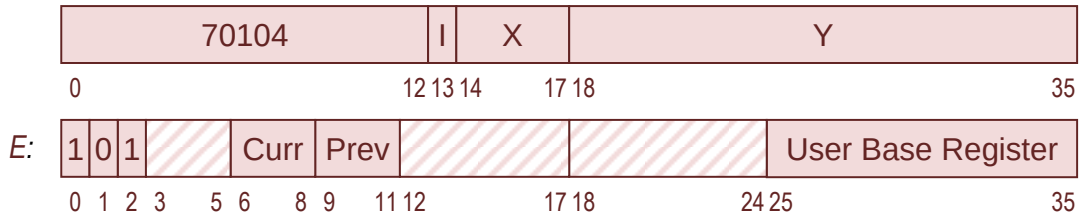


Figure 131 – RDUBR (DATI PAG) Instruction

Table 104 – RDUBR (DATI PAG) Bit Definitions	
Bit(s)	**Description**
0	Read as one.
1	Read as zero
2	Read as one.
3-5	Zero
6-8	Current AC Block
9-11	Previous Context AC Block
12-17	Zero
18-24	Zero
25-35	User base Register (Page Number)

15.1.4 Clear Page Table Entry (CLRPT/BLKO PAG)

This instruction clears the hardware page table so that the next reference to the word at E will cause a refill cycle.

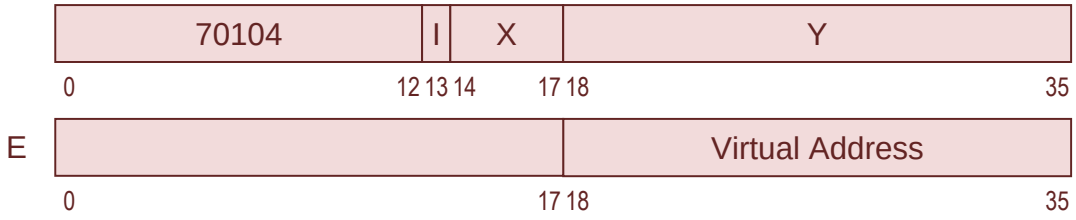


Figure 132 – CLRPT (BLKO PAG) Instruction

15.1.5 Executive Base Register (EBR) Instructions

15.1.5.1 Write to the Executive Base Register (WREBR/CONO PAG)

Configure the pager according to the effective address E. This operation invalidates the cache and clears the valid bits in the page tables.

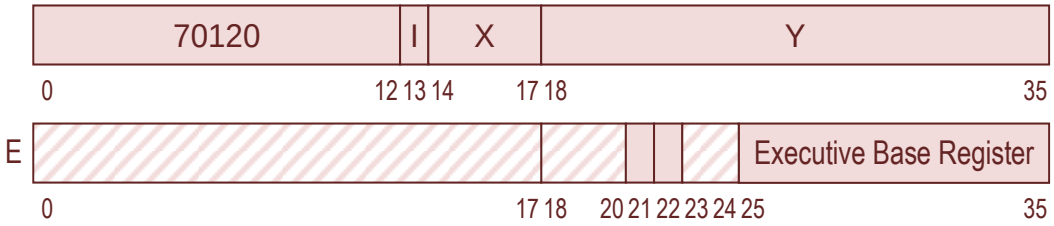


Figure 133 – WREBR (CONO PAG) Instruction

Table 105 – WREBR (CONO PAG) Bit Definitions		
Bit(s)	**BIT**	**Description**
18-20	-	Zero
21	T20PAG	Enable TOPS-20 paging
22	ENBPAG	Enable traps and paging
23-24	-	Zero
25-35	EBRPAG	Executive Base Address (Page Number)

15.1.5.2 Read the Executive Base Register (RDEBR/CONI PAG)

Read the status of the pager into the right half of location E.

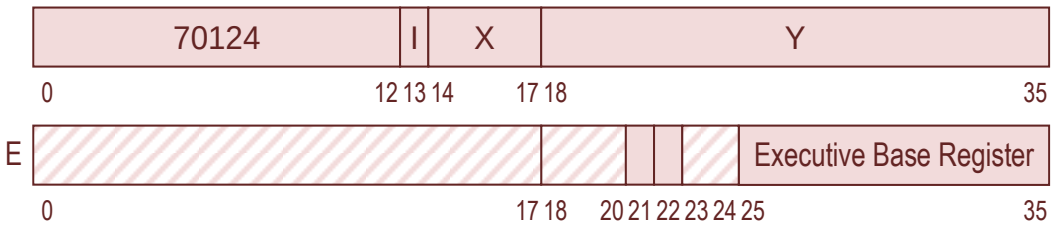


Figure 134 – RDEBR (CONI PAG) Instruction

Table 106 – RDEBR (CONI PAG) Bit Definitions		
Bit(s)	**BIT**	**Description**
18-20	-	Zero
21	T20PAG	Enable TOPS-20 paging
22	ENBPAG	Enable traps and paging
23-24	-	Zero
25-35	EBRPAG	Executive Base Address (Page Number)

15.1.6 Shared Pointer Table (SPT) Base Address Register

15.1.6.1 Read Shared Pointer Table Base Address Register (RDSPB)

Read the contents of the SPT base register into bits 14-35 of location E.

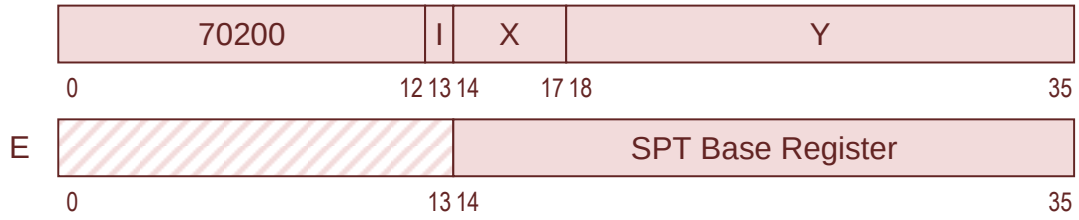


Figure 135 – RDSPB Instruction

15.1.6.2 Write Shared Pointer Table Base Address Register (WRSPB)

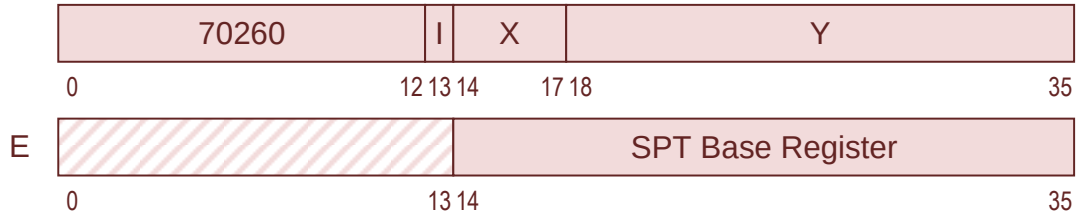


Figure 136 – WRSPB Instruction

15.1.7 Core Status Table (CST) Instructions

15.1.7.1 Read Core Status Table Base Register (RDCSB)

Read the contents of the Core Status Table Base Register into bits 14-35 of location E

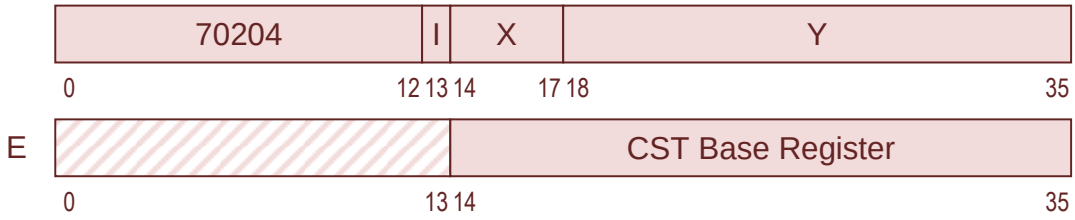


Figure 137 – RDCSB Instruction

15.1.7.2 Write Core Status Table Base Register (WRCSB)

Load the Core Status Table Mask Register from E.

The microcode will not use the CST if the CST Base Address is zero,

See also conditional compiles (`INHCST` and `NOCST`) in the KS10 microcode.

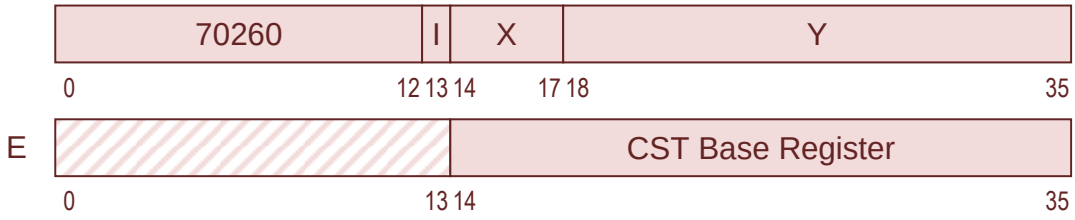


Figure 138 – WRCSB Instruction

15.1.7.3 Read Core Status Table Mask Register (RDCSTM)

Read the contents of the Core Status Table Mask Register into location E.

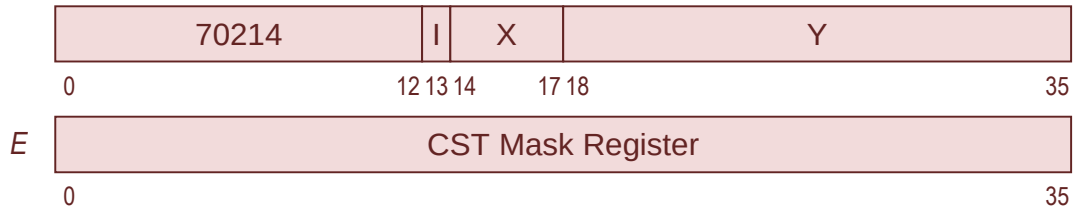


Figure 139 – RDCSTM Instruction

15.1.7.4 Write Core Status Table Mask Register (WRCSTM)

Load the Core Status Table Mask Register from E.

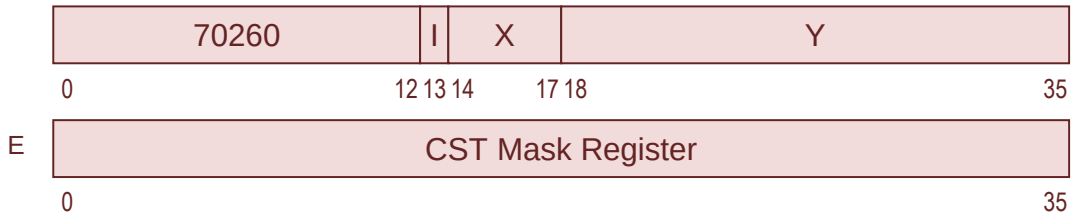


Figure 140 – WRCSTM Instruction

15.1.7.5 Read Core Status Table Process Use Register (RDPUR)

Read the contents of the Core Status Table Process Use Register into location E.

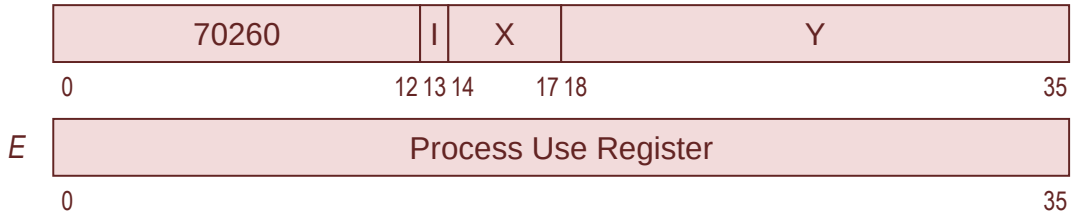


Figure 141 – RDPUR Instruction

15.1.7.6 Write Core Status Table Process Use Register (WRPUR)

Load the Core Status Table Process Use Register from location E.

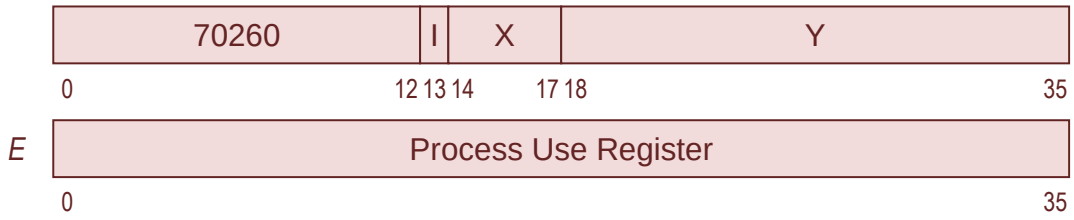


Figure 142 – WRPUR Instruction

15.1.8 Timebase Instructions

The timebase increments at 4.1 MHz.

15.1.8.1 RDTIM – Read Timebase

Read the contents of the time base registers, add the current contents of the millisecond counter to the double-word read, and place the result in location E, E+1.

Because the hardware is set up for signed arithmetic, bit 0 of the lower order word must be skipped.

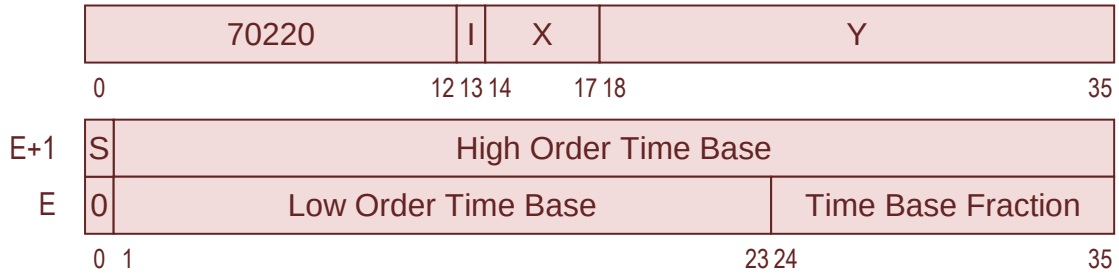


Figure 143 – RDTIM Instruction

15.1.8.2 WRTIM - Write Timebase

Read the contents of location E,E+1, clear the right twelve bits of the low order word read (the part corresponding to the hardware millisecond counter), and place the result in the time base registers in the workspace.

Because the hardware is set up for signed arithmetic, bit 0 of the lower order word must be skipped.

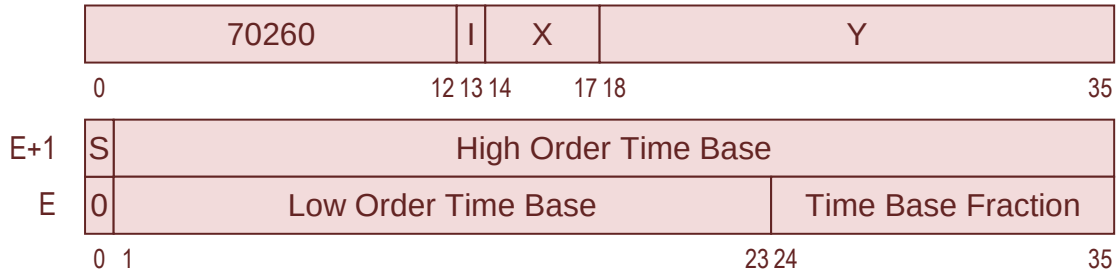


Figure 144 – WRTIM Instruction

15.1.9 Interval Timer Instructions

These instructions modify or query the interval timer.

15.1.9.1 RDINT – Read Interval Timer

The RDINT instruction reads the current value of the Interval Timer Period Register and stores the value in E. The period read is the same as that was supplied by WRINT

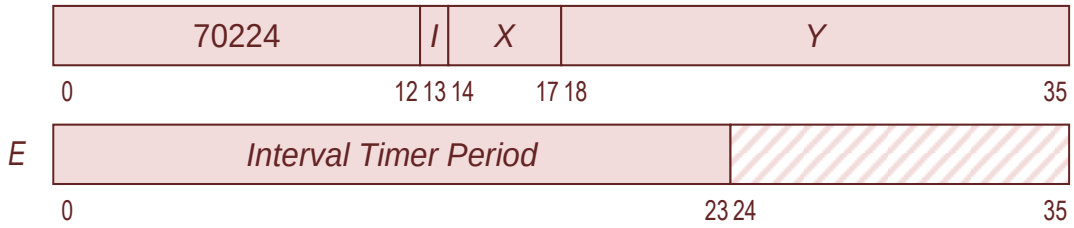


Figure 145 – RDINT Instruction

15.1.9.2 WRINT - Write Interval Timer

The WRINT instruction loads the Interval Timer Period Register and Interval Counter from E.

If the Interval Timer Period Register is set to zero, the Interval Timer function is disabled and the Period Counter does not decrement.

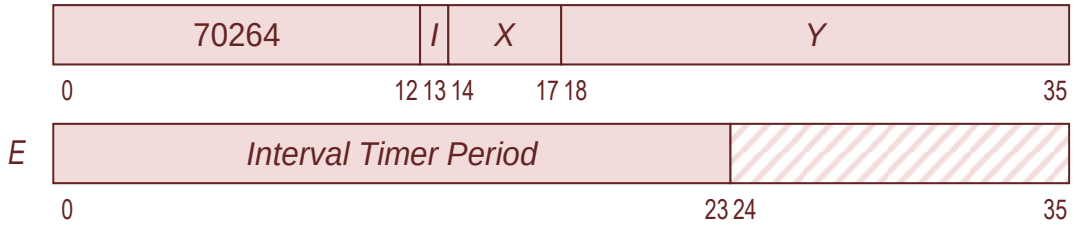


Figure 146 – WRINT Instruction

15.1.10 Halt Status Block Address Instructions

These instructions modify and query the location where the KS10 Halt Status Block is located.

The status consists of the sixteen AM2901 registers, the VMA register, the SC register and the FE register. The microcode initializes the address to o376000. TOPS20 sets the HSB address to o000400. TOPS10 sets the HSB address to o000424. The system should allocate 32 words for this storage.

FIXME: The microcode doesn't seem to actually store the SC register or the FE register.

15.1.10.1 RDHSB - Read Halt Status Block Address

Return the address of the Halt Status Block in E.

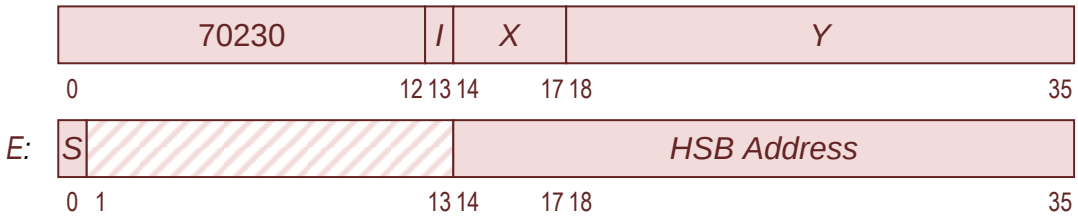


Figure 147 – RDHSB Instruction

15.1.10.2 WRHSB - Write Halt Status Block Address

Set the address of the Halt Status Block to E. If E is negative (bit 0 is set), the Halt Status address will not be modified.

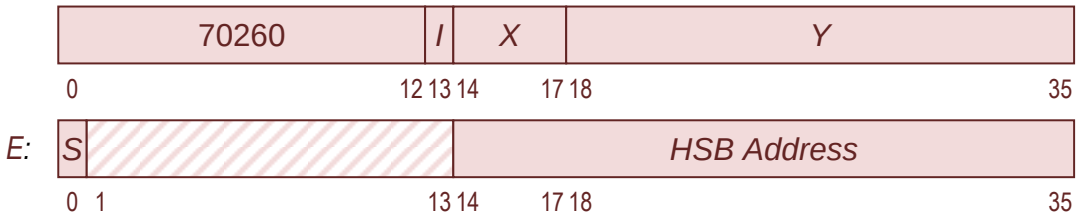


Figure 148 – WRHSB Instruction

16 Diagnostics

The following diagnostics have been executed in the simulator and on the target. The results are as follows.

Table 107 – Diagnostic Status

Test	Diagnostic Title	Sim Pass/ Fail	Target Pass/ Fail	Notes
DSKAAA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (1)	Pass	Pass	
DSKABA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (2)	Pass	Pass	
DSKACA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (3)	Pass	Pass	
DSKADA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (4)	Pass	Pass	
DSKAEA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (5)	Pass	Pass	
DSKAFA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (6)	Pass	Pass	
DSKAGA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (7)	Pass	Pass	
DSKAHA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (8)	Pass	Pass	
DSKAIA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (9)	Pass	Fail*	
DSKAJA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (10)	Pass	Pass	
DSKAKA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (11)	Pass	Pass	
DSKALA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (12)	Pass	Pass	
DSKAMA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (13)	Pass	Fail*	
DSKBAA0	DECSYSTEM 2020 BASIC INSTRUCTION RELIABILITY DIAGNOSTIC	FAIL	FAIL	
DSKCAA0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (1)	Pass	Fail*	
DSKCBA0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (2)	Pass	Pass	
DSKCCA0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (3)	Pass	Fail*	
DSKCDA0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (4)	Pass	Fail*	
DSKCEA0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (5)	Pass	Pass	
DSKCFC0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (6)	Pass	Pass	
DSKCGB0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (7)	FAIL	FAIL	
DSKDAB0	DECSYSTEM 2020 CPU & MEMORY RELIABILITY DIAGNOSTIC	FAIL	FAIL	
DSKEAA0	DECSYSTEM 2020 PAGING HARDWARE DIAGNOSTIC	Pass	FAIL	
DSKEBA0	KS10 - CACHE DIAGNOSTIC	FAIL	FAIL	
DSKECB0	DECSYSTEM KS10 KL-PAGING TEST	FAIL	FAIL	

DSKFAA0	DECSYSTEM 2020 INSTRUCTION TIMING DIAGNOSTIC	Pass	Pass	
DSMMAB0	DECSYSTEM 2020 KS10 1024K MEMORY DIAGNOSTIC	Pass	Pass	
DSMMBA0	DECSYSTEM 2020 BLT/FLOATING 1-0 MEMORY EXERCISER TEST	Pass	Pass	
DSMMCB0	DECSYSTEM 2020 FAST AC DIAGNOSTIC	Pass	Pass	
DSMMDC0	DECSYSTEM 2020 MEM DIAG	Pass	Pass	
DSRHAA0	DECSYSTEM 2020 MASSBUS ADAPTER EXERCISER	-	-	
DSRMAB0	DECSYSTEM-2020 RH11-RM03 Basic Device Diagnostic	Fail	Fail	
DSRMB0	DECSYSTEM-2020 KS10/RH11 RM03/RP06 Reliability Diagnostic	N/A	N/A	
DSRPAC0	DECSYSTEM 2020 KS10/RH11 – RP06 BASIC DEVICE DIAGNOSTIC	Pass	Pass	Note1
DSUBAC0	DECSYSTEM 2020 UNIBUS ADAPTER EXERCISER	Pass	Pass	Note1 Note 2
DSLTA	DECSYSTEM 2020 TELETYPE TEST (DSLTA)	Pass	Pass	
DSDZAB0	DECSYSTEM 2020 DZ11 ASYNC. LINE MUX DIAGNOSTICS (DSDZA)	Pass	Pass	Note 3
DSLPA	DECSYSTEM 2020 LINE PRINTER DIAGNOSTIC [DSLPA]	Pass	Pass	
DSDUA	DSDUA DECSYSTEM 2020 DUP-11 DIAGNOSTICS			

Note 1:
Some of the diagnostic modes are not fully implemented.

Note 2:
Because the PAGER is broken, the DSUBA diagnostic must be executed with the INHPAG switch asserted. Asserting the INHPAG switch will execute the test with a 256K address space and with paging disabled. The INHPAG switch has the value 000100

Note 3:
The DZ11 uses diagnostic uses timing loops. This requires a patch to modify timing.

17 Building the KS10 FPGA System

The directory of the project is illustrated below in Figure 149

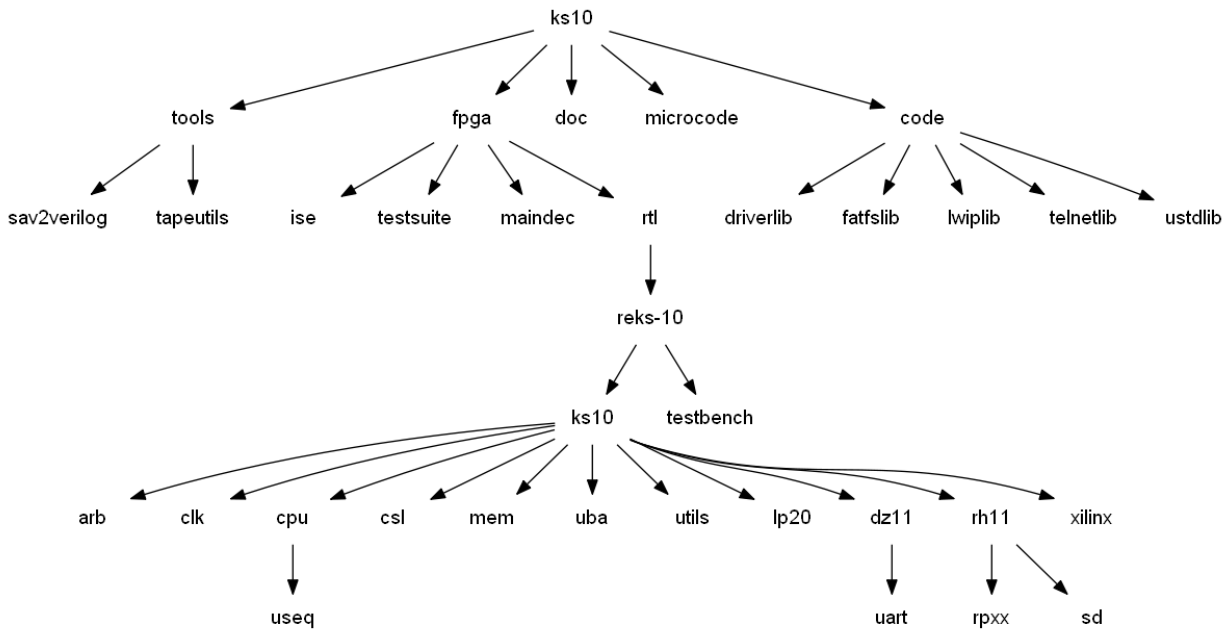


Figure 149 – Directory Structure

17.1 Tools

Linux (or Cygwin) development system including:

bash

make

rcs

awk

17.1.1 FTDI USB drivers.

This driver is required for the serial interface and for OpenOCD.

Include picture of FT-4232 configuration.

17.1.1.1 FTDI Hardware

Port 1

Port 2

Port 3

Port 4

17.1.2 FPGA Tools

The KS10 FPGA is currently only targeted toward a Xilinx Spartan 6 FPGA and therefore the FPGA tools are centered around the Xilinx toolset. Having said that, there is very little in the KS10 FPGA that is Xilinx specific so targeting alternate devices should be a relatively simple task.

17.1.2.1 Xilinx ISE Webpack Version 14.7

17.1.2.2 Icarus Verilog

Icarus Verilog is used for the regression testsuite because it is faster than Xilinx ISIM.

iverilog-20130827

17.1.2.3 FPGA JTAG Programming Cable

You will need a JTAG Programming Cable that is supported by Xilinx iMPACT. Please check to see if your JTAG Programming Cable is compatible.

I've chosen to invest in a Digilent JTAG HS2 Programming Cable because it supported by Xilinx iMPACT, Xilinx ChipScope, Xilinx EDK, and is supported by OpenOCD. See Section 17.1.3.3 for additional information about OpenOCD.

The makefile is designed such that programming the FPGA and/or programming the FPGA SPI Flash is part of the makefile script. For now, the makefile is pretty specific to my Digilent JTAG HS2 Programming Cable but the makefile can easily accommodate other Programming Cables as they are identified. Patches are welcome.

Information about the Digilent JTAG HS2 Programming Cable may be obtained from:

http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,395,1052&Prod=JTAG-HS2

The Digilent Plugin for Xilinx Tools must be installed before the Xilinx tools will recognize the Digilent JTAG HS2 Programming Cable. Please follow the installation procedure for that plug-in. The plug-in is available from:

https://digilentinc.com/Products/Detail.cfm?NavPath=2,66,768&Prod=DIGILENT-PLUGIN

Note: I may change the design such that the Console Microcontroller is responsible for programming the FPGA. In this case, the Console Microcontroller would load the FPGA firmware from the SD Card (or USB device) and program the FPGA as part of the Console boot sequence. I've chosen not to do this during development because my makefiles can build the FPGA firmware and program the FPGA directly with no file transfers to/from the SD Card – the process is just faster and less error prone that way. In this case, the programming cable would be optional.

I may change my mind about this if I can figure out how to install a TFTP server on the Console Microcontroller so that the Host Computer can TFTP the FPGA firmware directly to the SD Card.

17.1.2.4 Xilinx Chipscope (optional)

Xilinx ChipScope is an Embedded Logic Analyzer that may be used to debug the FPGA firmware.

Various parts of the KS10 FPGA system have been instrumented with ChipScope interfaces during the development and debugging process. These interfaces are conditionally compiled into the Verilog Design and are enabled by making changes to the Makefile. The debug macros that enable this instrumentation are summarized below.

```
#   -D CHIPSCOPE_CPU    Instrument the CPU
#   -D CHIPSCOPE_UBA    Instrument the UBA
#   -D CHIPSCOPE_SD     Instrument the SD Controller
#   -D CHIPSCOPE_MEM    Instrument the Memory Controller
#   -D CHIPSCOPE_CSL    Instrument the Console Interface
```

Because of FPGA resource limitations, only one interface can be instrumented at a time. Other modules of the KS10 FPGA may be instrumented later as required. The Xilinx IP "Cores" for the ChipScope Pro Integrated Controller (ICON), ChipScope Pro Virtual Input/Output (VIO), and ChipScope Pro Integrated Logic Analyzer (ILA) are located in the "`fpga/rtl/xilinx`" directory.

Be aware that using ChipScope to debug the KS10 FPGA is a task that is not for the timid.

The host computer (Linux or Windows) accesses the Embedded Logic Analyzer via the JTAG port of the FPGA. In my development environment, the ChipScope application is configured to use the Digilent JTAG HS2 Programming Cable. See Section 17.1.2.3 for additional information about the Digilent JTAG HS2 Programming Cable.

Other FPGA Programmer cables may be supported. Please check to see if your programming cable is compatible by the Xilinx tools.

Xilinx ChipScope is not part of the ISE Webpack and requires a separately purchased license. ChipScope is entirely optional and is not required to modify, build, program, or execute the FPGA firmware.

17.1.3 Software Tools

17.1.3.1 GCC tool suite ARM processors

The software was designed to be compiled using GCC compiler collection for ARM processors. The compiler that I use is:

```
$ arm-none-eabi-gcc –version
arm-none-eabi-gcc (GCC) 4.7.2
Copyright (C) 2012 Free Software Foundation, Inc.
```

17.1.3.2 GDB Debugger for ARM processors

17.1.3.3 OpenOCD On-Chip Debugger

The Host Computer uses OpenOCD to interact with the Console Microcontroller at a hardware level. OpenOCD is used to program the Flash ROM, as a GDB server, and to reset/reboot the Console Microcontroller remotely.

The Console Microcontroller hardware interface includes an FTDI FT-4232 Quad High Speed USB to Multipurpose UART/MPSSE IC. The first port (by default) is configured to emulate a JTAG port and this JTAG port is connected to the Console Microcontroller. The remaining three ports are used for RS-232 interfaces to the Console Microcontroller and to the DZ11 Terminal Multiplexer in the KS10 FPGA. This is described in detail in Section TBD.

Include picture from Windows Device Manager.

Open On-Chip Debugger (openocd-0.8.0)

OpenOCD version 0.8.0 is available for download from
http://sourceforge.net/projects/openocd/files/openocd/0.8.0/

17.1.3.4 Eclipse Integrated Development Environment

Appendix A – SMMON Commands

Table 108 – SMMON Command Summary	
Command	**Argument** **Description**
D	**Execute a command script from the Load Device. SMMON will respond by printing FILE.EXE. Respond by typing the filename of the command script.**
	SMMON CMD - **D** FILE.EXT - **SMCPU** LH SWS - 0 DSKAA. DSKAB. DSKAC. …

Table 108 – SMMON Command Summary	
Command	**Argument** **Description**
F	**Directory**
	SMMON CMD - **F** BEWARE.TXT 2 CONVRT.SAV 32 CZDLDD.BIN 26 CZDLDG.BIN 27 CZDMCC.BIN 32 CZDMEC.BIN 28 CZDMED.BIN 28 CZDMFB.BIN 31 CZDMFC.BIN 32 CZDMGD.BIN 27 CZDMHB.BIN 19 CZDMHC.BIN 19 CZDPBC.BIN 26 CZDPCD.BIN 22 CZDPDD.BIN 21 CZDPEB.BIN 11 CZM9BA.BIN 8 CZM9BD.BIN 12 CZQMCF.BIN 27 CZQMCG.BIN 27 DECX11.BIN 62 DECX11.CNF 1 DECX11.MAP 2 DIAG.DN22 1 DMPBOT.BIN 1 DS65B.BIN 63 DSANA.SAV 137 DSANAL.KMC 7 DSANAM.KMC 6 DSANB.SAV 35 DSCDA.SAV 51 DSDUA.SAV 74 DSDZA.SAV 51 DSKAA.SAV 19 DSKAB.SAV 21 DSKAC.SAV 15 DSKAD.SAV 17 DSKAE.SAV 13 DSKAF.SAV 16 DSKAG.SAV 8 DSKAH.SAV 33 DSKAI.SAV 38 DSKAJ.SAV 26 DSKAK.SAV 48 DSKAL.SAV 52 DSKAM.SAV 33 DSKBA.SAV 98 DSKCA.SAV 26 DSKCB.SAV 13 DSKCC.SAV 27 DSKCD.SAV 49 DSKCE.SAV 39 DSKCF.SAV 72 DSKCG.SAV 48 …
G	**Start or restart execution of the program that is in memory.**

Table 108 – SMMON Command Summary	
Command	**Argument** **Description**
H	**Help**

	SMMON CMD - **H**
	
	NORMAL START = 20000
	RESTART/ABORT = 20001
	PRINT TEST TITLE = 20002
	RESTART CURR TEST = 20003
	
	COMMANDS;
	STD=START DIAGNOSTIC
	STM=REINITIALIZE START
	STL=START LOADER
	START=START DIAGNOSTIC
	SFSTRT=SPECIAL FEATURE START
	PFSTRT=POWER FAIL START
	REE=REENTER
	DDT=DDT
	START1=SPECIAL START 1
	START2=SPECIAL START 2
	START3=SPECIAL START 3
	START4=SPECIAL START 4
	START5=SPECIAL START 5
	SMMON=LOAD SMMON
	SMMAG=LOAD SMMAG
	SMAPT=LOAD SMAPT
	
	R=RESELECT, X=XPN, I=INTERNAL, T=TTY, D=DEVICE,
	S=SINGLE, F=DIR, L=LIST, G=GO
	
	DEVICES;
	UBA #
	  0 = UBA 1, RH ADR 776700
	  1 = UBA 1, RH ADR 776700
	  2 = UBA 2, RH ADR 776700
	  3 = UBA 3, RH ADR 776700
	  # = UBA ADDRESS
	?= IDENTIFY DISKS, DSK:?= MASTER DIRECTORY

Table 108 – SMMON Command Summary	
Command	**Argument** **Description**
I	**Execute or re-execute command script that is in memory.**
	SMMON CMD - **I** DSKAA. SMMON PASS 1 DSKAA. SMMON PASS 2 DSKAA. SMMON PASS 3 DSKAA. …

Table 108 – SMMON Command Summary

Command	Argument Description
L	**List**

```
SMMON CMD - L

FILE.EXT - BEWARE.TXT

                JULY 1981 KS10 DIAGNOSTIC UPDATE
                --------------------------------

THE MASTER DIAGNOSTIC MAGTAPE DSXLA HAS BEEN REVISED

FROM    REVISION         0.4    FROM REVISION   0.5

THE FOLLOWING CHANGES HAVE BEEN MADE TO DSXLA

DIAGNOSTIC NAME         OLD REV LEVEL           NEW REV LEVEL
---------------         -------------           -------------

DSANA.SAV               NEW                     0.2
DSANB.SAV               NEW                     0.2
DSLPA.SAV               0.5                     0.7
DSPCA.SAV               NEW                     0.1
DSPCB.SAV               NEW                     0.1
DSTUA.SAV               0.2                     0.3

THE FOLLOWING 11 DIAGNOSTICS WERE ADDED TO THE DSXLA DIAGNOSTIC
MAGTAPE. THEY ARE ALSO ADDED TO THE KS10 FICHE LIBRARY.

CZDLDG.BIN              NEW                     0.7
CZQMCG.BIN              NEW                     0.7
CZDPBC.BIN              NEW                     0.3
CZDPCD.BIN              NEW                     0.4
CZDPDD.BIN              NEW                     0.4
CZDPEB.BIN              NEW                     0.2
ZDPFB0.BIN              NEW                     0.2
CZDMCC.BIN              NEW                     0.3
CZDMED.BIN              NEW                     0.4
…
```

Table 108 – SMMON Command Summary	
Command	**Argument** **Description**
R	**Reselect Boot Device**
	UBA # - **0<CR>** DISK:<DIRECTORY> OR DISK:[P,PN] – **CR>**
	UBA # - **0<CR>** DISK:<DIRECTORY> OR DISK:[P,PN] - **?<CR>** DSKB RP06 TOPS10 DSKC RP06 TOPS10 PS 1,0 RP06 TOPS20 DISK:<DIRECTORY> OR DISK:[P,PN] - **PS:?** ACCOUNTS.DIRECTORY BACKUP-COPY-OF-ROOT-DIRECTORY.IMAGE BOOTSTRAP.BIN DIAGNOSTICS.DIRECTORY DSKBTTBL. F-S.DIRECTORY INDEX-TABLE.BIN MFG.DIRECTORY NEW-SUBSYS.DIRECTORY NEW-SYSTEM.DIRECTORY OPERATOR.DIRECTORY RED.DIRECTORY REL.DIRECTORY ROOT-DIRECTORY.DIRECTORY SPOOL.DIRECTORY SUBSYS.DIRECTORY SYSTEM.DIRECTORY UETP.DIRECTORY UNSUPPORTED.DIRECTORY
S	**Load and run a single program** The specified program will be loaded and run the number of iterations as specified in the program by “ITERAT”.

Table 108 – SMMON Command Summary	
Command	**Argument** **Description**
	SMMON CMD - **S** FILE.EXT - **DSKCA** DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC #1 (DSKCA) VERSION 0.1, SV=0.3, CPU#=4097, MCV=130, MCO=470, HO=0, KASW=000000 000000 TTY SWITCH CONTROL ? - 0,S OR Y <CR> - 0 SWITCHES = 000000 000000 PC = 033002 RESULT = 367411 000000 C(AC) FAILED FAULT NUMBER = CA30603 PC = 034002 RESULT = 174400 000000 C(AC) FAILED FAULT NUMBER = CA31703 PC = 035002 RESULT = 200400 000000 C(E) FAILED FAULT NUMBER = CA34401 END PASS 1. END PASS 2. END PASS 3. END PASS 4. END PASS 5. END PASS 6. END PASS 7. END PASS 8. END PASS 9. END PASS 10. END PASS 11. END PASS 12.

Table 108 – SMMON Command Summary	
Command	**Argument Description**
T	**Execute a command script from the TTY**
	SMMON CMD – **T** NAME PASSES RH SW ITERATIONS **DSKAA 10 0 1** **^Z** DSKAA. SMMON PASS 1 DSKAA. SMMON PASS 2 DSKAA. SMMON PASS 3 DSKAA. SMMON PASS 4 DSKAA. SMMON PASS 5 DSKAA. SMMON PASS 6 DSKAA. SMMON PASS 7 DSKAA. SMMON PASS 8 CMD'S REQUIRED
X	**XPN Directs SMMON to run through the expanded command set dialogue. Refer to the section on the expanded command which follows Table 5.**

Appendix B – Building a RP06 Red Pack from Tape in SIMH

The Reliability Exerciser and Diagnostic Pack (RED PACK) is a very useful diagnostic tool that DEC provided to its Field Service Engineers to help maintain and repair the KS10.

The procedure for creating the RED PACK using the SIMH simulator was provided by Peter Hettkamp as a response to my query on the alt.sys.pdp10 newsgroup. Thanks Peter.

Download and unzip the RED PACK tape image and help file from pdp-10.trailing-edge.com as follows:

```
$ wget http://pdp-10.trailing-edge.com/tapes/red405a2.tap.bz2 red405a2.tap.bz2
$ wget http://pdp-10.trailing-edge.com/red405a2/11/red/red20.hlp red20.hlp
$ bunzip2 red405a2.tap.bz2
```

Use your favorite editor to create a command script for SIMH. I use the following script:

```
$ cat red405a2.cmd

set rp0 rp06
att rp0 red405a2.rp06
set tu0 format=e11
set tu0 locked
att tu0 ./red405a2.tap
```

Start SIMH

```
$ ./pdp10 red405a2.cmd
```

```
PDP-10 simulator V4.0-0 Beta        git commit id: 8204a203
sim> boot tu0
```

The RED405A2 tape is a KS10 boot tape. The first file on a bootable tape is for the KS10 console processor. The pdp10 emulator does not make complete sense out of it. Interrupt it by typing ^E. Then boot again from tu0. This time, the tape is positioned at the second file on the tape, which is MTBOOT:

```
^E

Simulation stopped, PC: 775451
sim> boot tu0

MTBOOT >/L
```

The /G143 is used to load the initial monitor. Note: the parameters are set per the instructions in "red20.hlp" file that was downloaded.

```
MTBOOT >/G143

[FOR ADDITIONAL INFORMATION TYPE "?" TO ANY OF THE FOLLOWING QUESTIONS.]

DO YOU WANT TO REPLACE THE FILE SYSTEM ON THE PUBLIC STRUCTURE? Y

DO YOU WANT TO DEFINE THE PUBLIC STRUCTURE? Y

HOW MANY PACKS ARE IN THIS STRUCTURE: 1

ON WHICH "CHANNEL,UNIT" IS LOGICAL PACK # 0 MOUNTED: ?
[ENTER A PAIR OF NUMBERS SEPARATED BY A COMMA THAT SPECIFY THE
 CHANNEL AND UNIT UPON WHICH THE APPROPRIATE PACK IS MOUNTED.
 THE FOLLOWING IS A LIST OF VALID CHANNEL,UNIT PAIRS:
0,0 ;TYPE=RP06
0,1 ;TYPE=RP06,OFFLINE
0,2 ;TYPE=RP06,OFFLINE
0,3 ;TYPE=RP06,OFFLINE
0,4 ;TYPE=RP06,OFFLINE
0,5 ;TYPE=RP06,OFFLINE
0,6 ;TYPE=RP06,OFFLINE
0,7 ;TYPE=RP06,OFFLINE
]

ON WHICH "CHANNEL,UNIT" IS LOGICAL PACK # 0 MOUNTED: 0,0

DO YOU WANT THE DEFAULT SWAPPING SPACE? N

HOW MANY PAGES FOR SWAPPING? 5000

DO YOU WANT THE DEFAULT SIZE FRONT END FILE SYSTEM? N

HOW MANY PAGES FOR THE FRONT END FILE SYSTEM? 0

DO YOU WANT THE DEFAULT SIZE BOOTSTRAP AREA? N

HOW MANY PAGES FOR THE BOOTSTRAP FILE? 100

[STRUCTURE "PS" SUCCESSFULLY DEFINED]

[PS MOUNTED]

%%NO SETSPD

System restarting, wait...
ENTER CURRENT DATE AND TIME: 23-NOV-2015 21:24

YOU HAVE ENTERED MONDAY, 23-NOVEMBER-2015 9:24PM,
 IS THIS CORRECT (Y,N) Y
```

```
WHY RELOAD? RED 405 BUILD

<SYSTEM>ACCOUNTS-TABLE.BIN NOT FOUND - ACCOUNT VALIDATION IS DISABLED

RUNNING DDMP


NO SYSJOB
NO EXEC
```

At this point, press ^C to grab the computer's attention. The monitor responds with a mini-exec prompt. We use this mini-exec to load the exec from the tape. Please note that you only type G, the mini-exec fills in the "ET FILE "... The error message at this point is expected, the tape drive needs to space past the current file mark. We repeat the GET FILE command

```
MX>GET FILE MTA0:

INTERRUPT AT 0
MX>GET FILE MTA0:
```

This loaded the TOPS-20 EXEC. Enter an S, the mini-exec completes the START command.

```
MX>START

 TOPS-20 Command processor 4(560)
```

Enable capabilities.

```
@ENABLE
```

At this point, we have an empty public structure file system on the RP06, and a running TOP-20 EXEC. Up to this point, this followed the usual procedure to install TOPS-20.

Create a file named SERIAL which contains the serial number of this KS10:

```
$COPY TTY: SERIAL
 TTY: => SERIAL..1

4097
^Z
```

Copy the installer command script from MTA0:

```
$COPY MTA0: INSTALL.CMD
 MTA0: => INSTALL.CMD.1 [OK]
```

Execute the installer command script. Everything runs automatically.

```
$TAKE INSTALL.CMD
 <SUBSYS>MIC.EXE.1 SAVED
 MTA0: => RESTORE.MIC.1 [OK]
 END OF INSTALL.CMD.1
```

```
$;<UETP.LIB>RESTORE.MIC.1,  7-JUN-79 14:43:12, EDIT BY EIBEN
;USES THE ORIGINAL RED-TAPE 1 TO BUILD RED-PACK
DAYTIME
 MONDAY, NOVEMBER 23, 2015 21:28:52
$TERMINAL NO PAGE
$ENA
$RU MTA0:
DLUSER>STR PS:
DLUSER>LOA MTA0:

DONE.
DLUSER>EXIT
$RU MTA0:
DUMPER>TAPE MTA0:
DUMPER>REST <*>*.*


DUMPER TAPE # 1, RED405-ORIGINAL :<DIAGNOSTICS>, FRIDAY,  7-AUG-81 2139
LOADING FILE(S) INTO PS:<DIAGNOSTICS>

END OF SAVESET
DUMPER>REST <*>*.*


DUMPER TAPE # 1, RED405-ORIGINAL :<F-S>, FRIDAY,  7-AUG-81 2142
LOADING FILE(S) INTO PS:<F-S>

END OF SAVESET
DUMPER>REST <*>*.*


DUMPER TAPE # 1, RED405-ORIGINAL :<RED>, FRIDAY,  7-AUG-81 2142
LOADING FILE(S) INTO PS:<RED>

END OF SAVESET
DUMPER>REST <*>*.*


DUMPER TAPE # 1, RED405-ORIGINAL :<SUBSYS>, FRIDAY,  7-AUG-81 2143
LOADING FILE(S) INTO PS:<SUBSYS>

END OF SAVESET
DUMPER>REST <*>*.*


DUMPER TAPE # 1, RED405-ORIGINAL :<SYSTEM>, FRIDAY,  7-AUG-81 2148
LOADING FILE(S) INTO PS:<SYSTEM>

END OF SAVESET
DUMPER>REST <*>*.*
```

```
DUMPER TAPE # 1, RED405-ORIGINAL :<UETP.LIB>, FRIDAY,  7-AUG-81 2149
LOADING FILE(S) INTO PS:<UETP.LIB>

END OF SAVESET
DUMPER>REST <*>*.*


DUMPER TAPE # 1, RED405-ORIGINAL :<UETP.LIB>, FRIDAY,  7-AUG-81 2153
LOADING FILE(S) INTO PS:<REL>

END OF SAVESET
DUMPER>REST <*>*.*


DUMPER TAPE # 1, RED405-ORIGINAL :<UETP.MIC>, FRIDAY,  7-AUG-81 2153
LOADING FILE(S) INTO PS:<UETP.MIC>

END OF SAVESET
DUMPER>REST <*>*.*


DUMPER TAPE # 1, RED405-ORIGINAL :<UNSUPPORTED>, FRIDAY,  7-AUG-81 2153
LOADING FILE(S) INTO PS:<UNSUPPORTED>

END OF SAVESET
DUMPER>REW
DUMPER>EXIT
$TV

*;Y$$
INPUT FILE: <OPERATOR>SERIAL
6 CHARS
*BJ$1XZ$$
*
;Y$$
INPUT FILE: <UETP.LIB>PS-MICRO.MIC
1315 CHARS
*SSERIAL $K$GZ$$
*
;U$$
OUTPUT FILE: <UETP.LIB>PS-MICRO.MIC
*;Y$$
INPUT FILE: <UETP.LIB>RED-MICRO.MIC
1362 CHARS
*SSERIAL $K$GZ$$
*
;X$$
OUTPUT FILE: <UETP.LIB>RED-MICRO.MIC
```

```
$RU <SYSTEM>MAKDMP

MAX SIZE OF MEMORY IN K (512): 512
$DO PS-MICRO
$;<UETP.LIB>PS-MICRO.MIC.1,  7-JUN-79 14:15:50, EDIT BY EIBEN
;WRITES 8080-FILE AREA
DAYTIME
 MONDAY, NOVEMBER 23, 2015 21:29:30
$TERMINAL NO PAGE
$ENA
$CONN <DIAGNOSTICS>
$RUN <DIAGNOSTICS>SMFILE.EXE

DECSYSTEM 2020 DIAGNOSTICS FE-FILE PROGRAM
VERSION 0.3, TOPS-20, KS10, CPU#=4097
[FOR HELP TYPE "HELP"]
SMFILE>;        WE WANT TO UPDATE THE BOOTSTRAP AREA ON PS:
WRI SET PS:<ROOT-DIRECTORY>BOOTSTRAP.BIN
SMFILE>;        WE WANT TO START CLEAN
WRITE RESET
SMFILE>;        WE HAVE TO READ M-CODE IN FIRST
READ KS10.ULD
SMFILE>;        WHAT VERSION OF M-CODE ARE WE HANDLING?
EXA CRAM 137
SHOULD BE:      000137/357707070000011700003777604541O2

C    J   #  ALU S/D  A/B RBM SPEC DISP SKIP T C SC FE FM MC DV MP C/LR M
0 3577 000117 3 771 0005 437   00   70   70 0 0  0  0  0  0  0  0  4 0 0

SMFILE>;        WE HAVE TO SPECIFY A DEC-CERTIFIED SERIAL-NUMBER
SERIAL 4097
SMFILE>WRITE CRAM
SMFILE>;        AND THE BOOT FOR THE MONITOR
WRITE BOOT <SYSTEM>SMBOOT.EXE
SMFILE>;        AND OUR DIAGNOSTIC BOOT
WRITE DIAGBT SMMON.EXE
SMFILE>;        AND BOOT-CHECK 2
WRITE BC2 SMBC2.EXE
SMFILE>;        AT LAST WE HAVE TO UPDATE THE HOME-BLOCKS
WRITE DONE

[HOME BLOCKS SET]
SMFILE>;        NOW WE WRITE OUT THE SAME M-CODE FOR LATER USE ON MAGTAPE
OUTPUT CRAM PS:<SYSTEM>KS10.RAM
SMFILE>;        WE WANT A NEW M-CODE FOR DIAGNOSTICS
OUTPUT CRAM PS:<DIAGNOSTICS>SMTAPE.RAM
SMFILE>;        WE ALSO NEED THE MONITOR-MAGTAPE BOOT
OUTPUT MTBOOT <SYSTEM>SMMTBT.EXE PS:<SYSTEM>MTBOOT.RDI
SMFILE>;LETS CHECK A LITTLE BIT , WHAT WE HAVE DONE
```

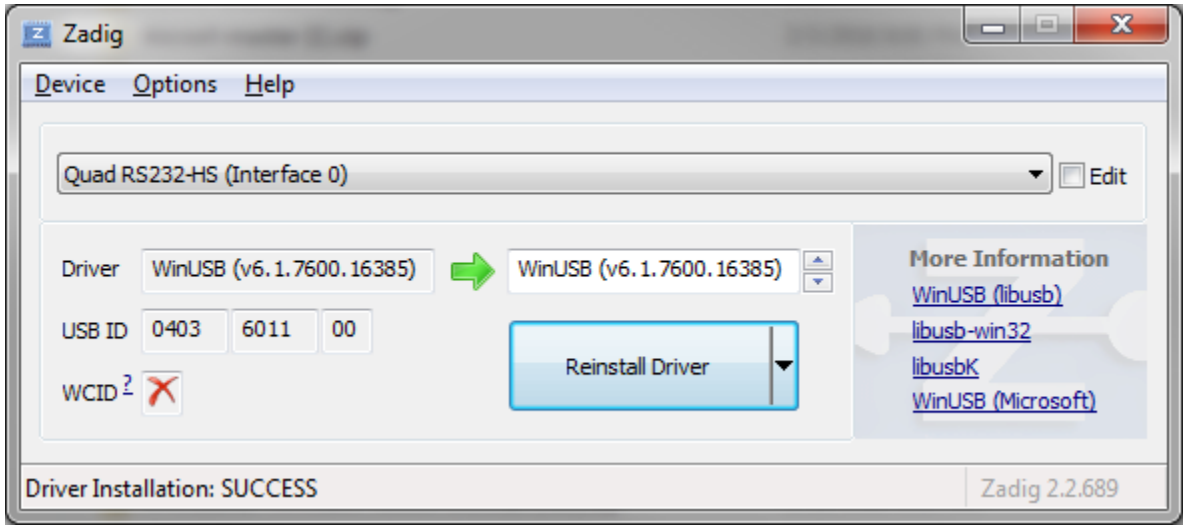
```
WRI SET PS:<ROOT-DIRECTORY>BOOTSTRAP.BIN
SMFILE>;        WHAT DISK WERE WE USING?
INF DISK

USING PS        RP06

SMFILE>;        DID WE APPROACH BOUNDARIES?
INF FREE

FRONT-END FREE PAGES = 28

SMFILE>;        WHERE ARE WE LOCATED?
INF FEFILE

DISK ADDRESS IN HOME BLOCK = 100000 000574
LENGTH IN HOME BLOCK =       000000 000620
8080 POINTER IN HOME BLOCK = 000100 000000

SMFILE>;        DID WE GO INTO "FUTURE" STUFF?
INF IND

THE FOLLOWING FRONT-END INDIRECT FILES EXIST:

FRONT-END FREE PAGES = 28

SMFILE>;        ..AND FINALLY GO BACK TO MONITOR-MODE
EXIT
$DISABLE
@DAYTIME
 MONDAY, NOVEMBER 23, 2015 21:29:53
@TERMINAL PAGE
@UNL MTA0:
@CONN
@DAYTIME
 MONDAY, NOVEMBER 23, 2015 21:29:57
@TERMINAL PAGE
@
[MICEMF - END OF MIC FILE: RESTORE.MIC.1 ]
@KMIC
@
Simulation stopped, PC: 000003 (SOJG 2,3)
sim> QUIT
Goodbye
```

Appendix C – USB Driver Mayhem

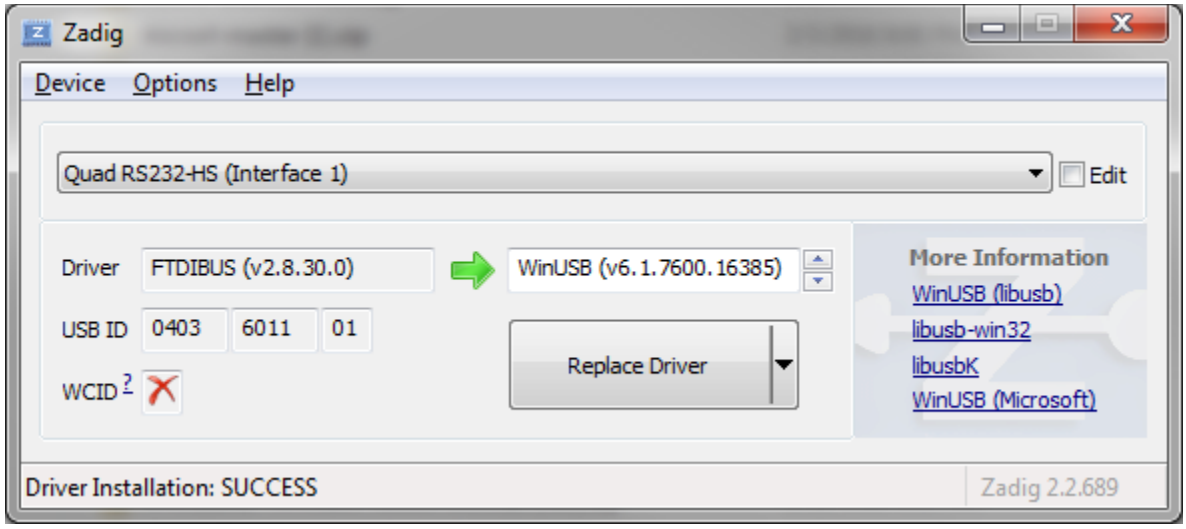
Get Zadig from http://zadig.akeo.ie/

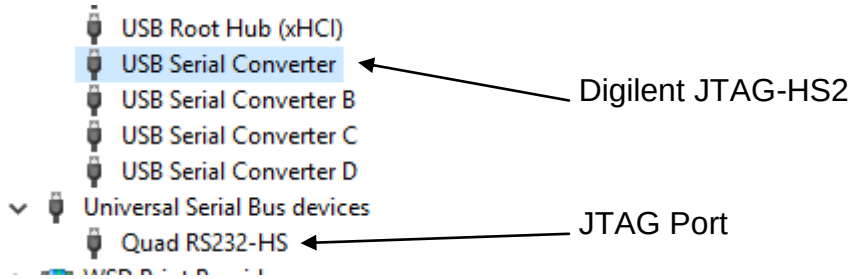
Select the Quad RS232+HS device.

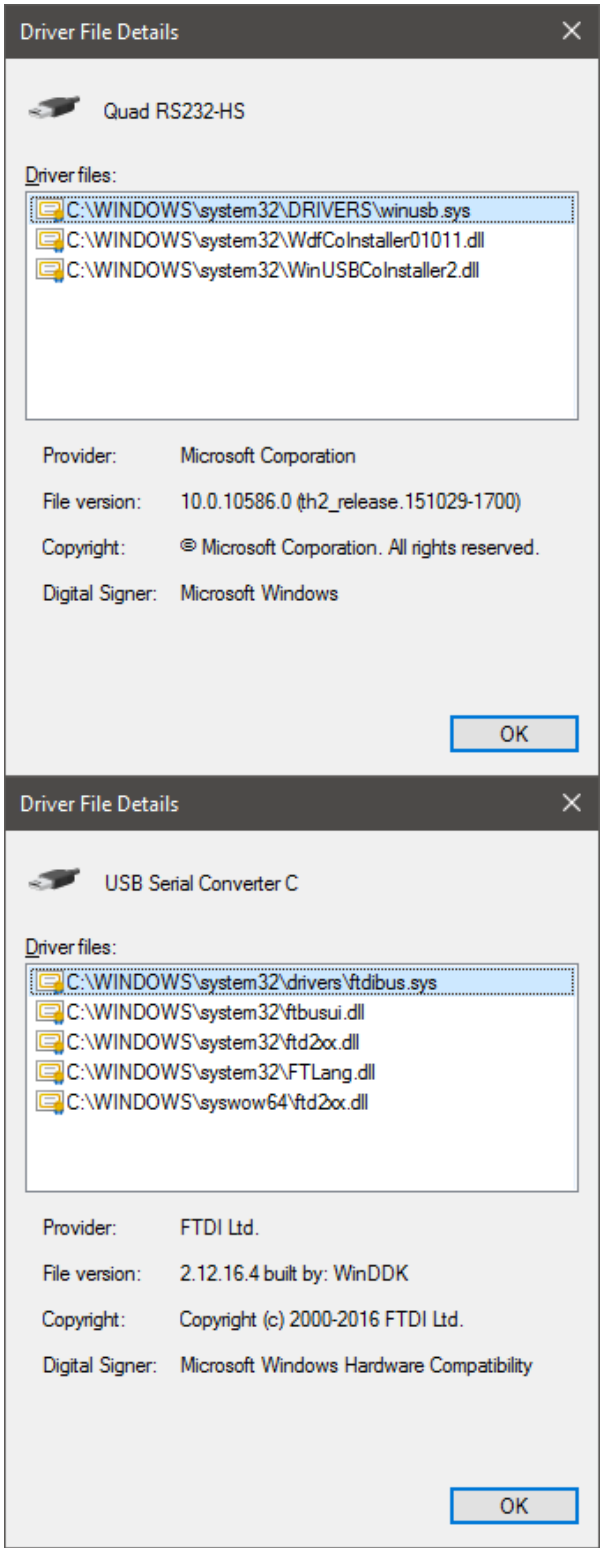
The MCU JTAG is on Interface 0. It should be configured to use WinUSB as follows:

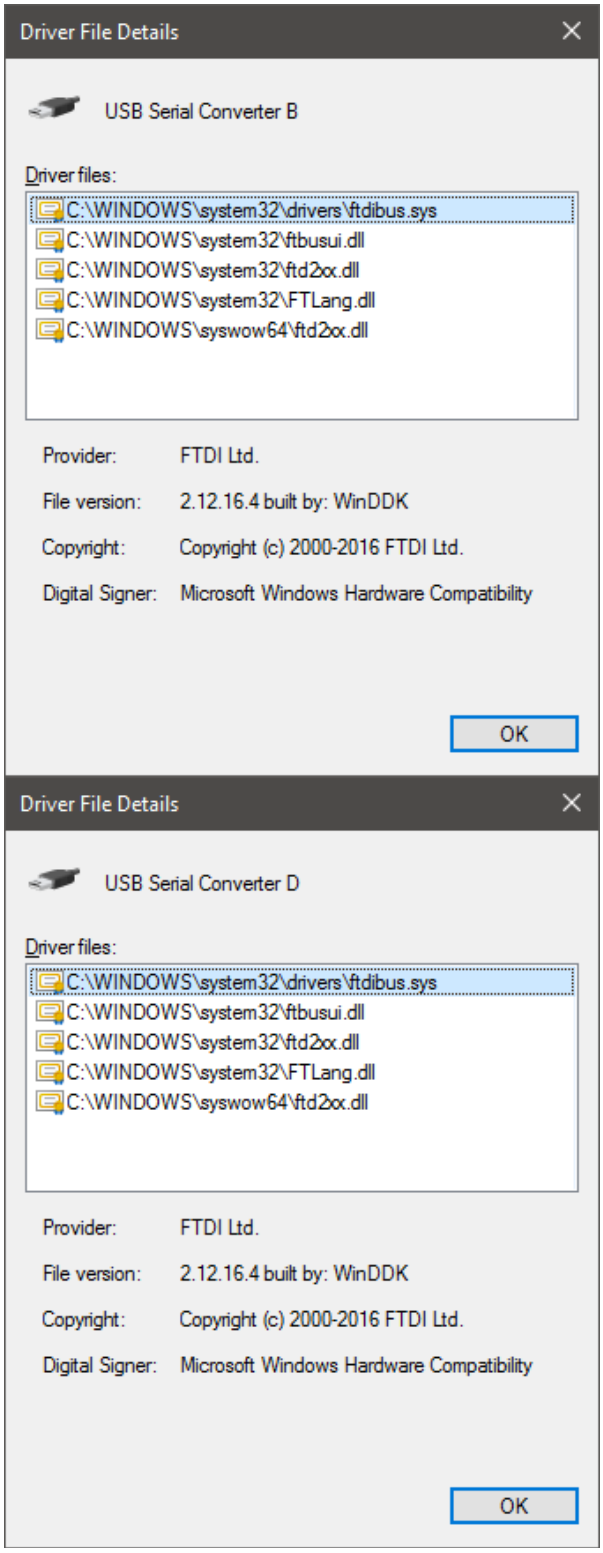


The serial ports are on Interface 1, 2, and 3. They should be configured to use the FTDIBUS drivers as follows:bt




USB Root Hub (xHCI)
USB Serial Converter
USB Serial Converter B
USB Serial Converter C
USB Serial Converter D
Universal Serial Bus devices
Quad RS232-HS
Digilent JTAG-HS2
JTAG Port


Driver File Details
Quad RS232-HS
Driver files:
C:\WINDOWS\system32\DRIVERS\winusb.sys
C:\WINDOWS\system32\WdfCoInstaller01011.dll
C:\WINDOWS\system32\WinUSBCoInstaller2.dll
Provider: Microsoft Corporation
File version: 10.0.10586.0 (th2_release.151029-1700)
Copyright: © Microsoft Corporation. All rights reserved.
Digital Signer: Microsoft Windows
OK


Driver File Details
USB Serial Converter B
Driver files:
C:\WINDOWS\system32\drivers\ftdibus.sys
C:\WINDOWS\system32\ftbusui.dll
C:\WINDOWS\system32\ftd2xx.dll
C:\WINDOWS\system32\FTLang.dll
C:\WINDOWS\syswow64\ftd2xx.dll
Provider: FTDI Ltd.
File version: 2.12.16.4 built by: WinDDK
Copyright: Copyright (c) 2000-2016 FTDI Ltd.
Digital Signer: Microsoft Windows Hardware Compatibility
OK

Driver File Details
USB Serial Converter C
Driver files:
C:\WINDOWS\system32\drivers\ftdibus.sys
C:\WINDOWS\system32\ftbusui.dll
C:\WINDOWS\system32\ftd2xx.dll
C:\WINDOWS\system32\FTLang.dll
C:\WINDOWS\syswow64\ftd2xx.dll
Provider: FTDI Ltd.
File version: 2.12.16.4 built by: WinDDK
Copyright: Copyright (c) 2000-2016 FTDI Ltd.
Digital Signer: Microsoft Windows Hardware Compatibility
OK

Driver File Details
USB Serial Converter D
Driver files:
C:\WINDOWS\system32\drivers\ftdibus.sys
C:\WINDOWS\system32\ftbusui.dll
C:\WINDOWS\system32\ftd2xx.dll
C:\WINDOWS\system32\FTLang.dll
C:\WINDOWS\syswow64\ftd2xx.dll
Provider: FTDI Ltd.
File version: 2.12.16.4 built by: WinDDK
Copyright: Copyright (c) 2000-2016 FTDI Ltd.
Digital Signer: Microsoft Windows Hardware Compatibility
OK

Appendix D – Monitor Notes

Info from Richard Cornwell

TOPS-10 6.03 runs on KA and KI

TOPS-10 7.03 runs on KS and uses TOPS-10 pager.

TOPS-10 7.04 runs on KS and uses more resources than TOPS-10 7.03

TOPS-10 7.04 runs on KS and uses Tops20 pager

What you want to do is read the MIG guide