KS10 FPGA Processor Manual

doyle (at) cox (dot) net

Table of Contents

1 Introduction 12
1.1 The DEC KS10 12
1.2 The KS10 FPGA 12
2 The KS10 FPGA Architecture 14
2.1 The KS10 FPGA Design Hierarchy 14
2.2 The KS10 FPGA Design Description 14
2.2.1 KS10 Bus Arbiter (ARB) 14
2.2.2 KS10 Console Interface (CSL) 14
2.2.3 KS10 Central Processing Unit (CPU) 14
2.2.3.1 KS10 CPU Interval Timer (TIMER) 15
2.2.3.2 KS10 CPU Virtual Memory Address (VMA) 16
2.2.3.3 KS10 CPU Priority Interrupt Controller (INTR) 16
2.2.3.4 KS10 CPU DBM Mux (DBM) 16
2.2.3.5 KS10 CPU Backplane Interface (BUS) 16
2.2.3.6 KS10 CPU Arithmetic Processor Flags (APR) 16
2.2.3.7 KS10 CPU Arithmetic Logic Unit (ALU) 17
2.2.3.8 KS10 CPU Microsequencer (USEQ) 19
2.2.3.9 KS10 CPU DBUS 24
2.2.3.10 KS10 CPU Instruction Register (IR) 25
2.2.3.11 KS10 CPU Step Count Adder (SCAD) 25
2.2.3.12 KS10 CPU RAMFILE 26
2.2.3.13 KS10 CPU Pager (PAGER) 26
2.2.3.14 KS10 CPU Page Fault Dispatch (PF_DISP) 26
2.2.3.15 KS10 CPU Next Instruction Dispatch (NI_DISP) 27
2.2.3.16 KS10 CPU Previous Context (PXCT) 27
2.2.4 KS10 Memory Controller (MEM) 27
2.2.5 KS10 IO Bus Adapter (UBA) 27
3 KS10 FPGA Backplane 28
3.1 KS10 FPGA Address Bus 29
3.2 KS10 FPGA Bus Cycles 31
3.2.1 APR / Timer Interrupt 31
3.2.2 KS10 FPGA External Interrupt 31
4 Console Microcontroller 32
4.1 Booting the KS10 FPGA 34
4.2 KS10 FPGA Console Commands 34
4.3 KS10 Console Software Operation 40
4.3.1 Field Programmable Gate Array (FPGA) Driver 40
4.3.2 Universal Asynchronous Receiver/Transmitter (UART) Driver 40
4.3.3 Synchronous Serial Interface (SSI) Driver 40
4.3.4 External Peripheral Interface (EPI) Driver 40
4.3.5 General-Purpose Input/Outputs (GPIOs) Driver 40
4.3.6 KS10 Driver 41
4.3.7 Secure Digital High-Capacity (SDHC) Card Driver 41
4.3.8 FAT32 Filesystem 41
5 KS10 FPGA Console Interface 42
5.1 KS10 FPGA Console Interface Registers 42
5.1.1 Console Microcontroller Interface 42
5.1.2 Console Interface Bus Design 43
5.1.3 Console Interface Register Memory Map 44
5.1.4 Console Control/Status Register 47
5.1.5 Console Data Register 49
5.1.6 Console Address Register 49
5.1.7 Console Instruction Register 49

5.1.8 DZ11 Console Control Register....50
5.1.9 RH11 Console Control Register....50
5.1.10 RH11 Debug Register....51
5.1.11 Debug Interface....52
5.1.11.1 Debug Control/Status Register (CSR)....52
5.1.11.2 Debug Breakpoint Address Register (DBAR)....53
5.1.11.3 Debug Breakpoint Mask Register (DBMR)....54
5.1.11.4 Debug Instruction Trace Register (ITR)....55
5.1.12 Firmware Version Register....56
5.2 Controlling the KS10....57
5.2.1 The RUN bit....58
5.2.2 The CONT bit....58
5.2.3 The EXEC bit....58
5.3 Console Interface Protocol....58
5.4 The Communications Area....59
5.4.1 Halt Switch....60
5.4.2 Keep Alive....60
5.4.3 Console TTY (CTY) Protocol....61
5.4.3.1 Console TTY (CTY) Input Protocol....61
5.4.3.2 Console TTY (CTY) Output Protocol....62
5.4.4 KLINIK Protocol....63
5.4.4.1 KLINIK Input Protocol....63
5.4.4.2 KLINIK Output Protocol....63
5.4.5 Boot RH11 Address....64
5.4.6 Boot Unit Number....64
5.4.7 Boot Magtape Parameters....64
6 KS10 Memory Controller....66
6.1 Memory Status Registers....66
6.2 SSRAM Memory Interface....67
7 KS10 IO Bridge (was Unibus) Implementation....70
7.1 IO Bus to KS10 Byte and Word Translation....70
7.2 IO Bus Bridge Paging Memory....71
7.3 IO Bus Bridge Paging....72
7.4 IO Bridge Control Status Register (UBACSR)....72
7.5 IO Bridge Maintenance Register....74
7.6 Bridged IO Devices....74
8 DZ11 Register Compatible Asynchronous Multiplexer....75
8.1 DZ11 Control and Status Register (CSR)....76
8.2 DZ11 Receiver Buffer Register (RBUF)....78
8.3 DZ11 Line Parameter Register (LPR)....79
8.4 DZ11 Transmit Control Register (TCR)....80
8.5 DZ11 Modem Status Register (MSR)....81
8.6 DZ11 Transmit Data Register (TDR)....81
8.7 Hardware Description....82
8.7.1 The Transmitter Scanner....82
8.7.2 The Baud Rate Generator....82
8.7.2.1 The Fractional-N Divider....82
9 RH11 Massbus Disk Controller....84
9.1 Definitions....85
9.1.1 Disk Clear Operations....85
9.1.1.1 IO Bridge Clear....85
9.1.1.2 Controller Clear....85
9.1.1.3 Drive Clear....85
9.1.1.4 Error Clear....85

9.2 RH11 Compatible Disk Controller....85
9.2.1 RH11 Control and Status #1 (RHCS1) Register....86
9.2.2 RH11 Word Count (RHWC) Register....88
9.2.3 RH11 Bus Address (RHBA) Register....88
9.2.4 RH11 Control and Status #2 (RHCS2) Register....89
9.2.5 RH11 Data Buffer (RHDB) Register....90
9.3 RP06/07 Disk Simulator....91
9.3.1 RP Control and Status #1 (RPCS1) Register....93
9.3.2 RP Disk Address (RPDA) Register....95
9.3.3 RP Drive Status (RPDS) Register....96
9.3.4 RP Error #1 (RPER1) Register....99
9.3.5 RP Attention Summary (RPAS) Register....102
9.3.6 RP Look Ahead (RPLA) Register....102
9.3.7 RP Maintenance (RPMR) Register....103
9.3.8 RP Drive Type (RPDT) Register....106
9.3.9 RP Serial Number (RPSN) Register....107
9.3.10 RP Offset (RPOF) Register....107
9.3.11 RP Desired Cylinder (RPDC) Register....108
9.3.12 RP Current Cylinder (RPCC) Register....109
9.3.13 RP Error Status #2 (RPER2) Register....110
9.3.14 RP Error Status #3 (RPER3) Register....112
9.3.15 RP Error Position (RPEC1) Register....113
9.3.16 RP Error Pattern (RPEC2) Register....113
9.4 RH11 Interrupts....114
9.5 RPXX Commands....114
9.5.1 Seek Function....114
9.5.2 Search Function....116
9.5.3 Offset Command and Return to Centerline Functions....116
9.5.4 Recalibrate Function....116
9.5.5 Unload Function....117
9.5.6 Pack Acknowledge Function....117
9.5.7 Read-in Preset Function....117
9.5.8 Release Function....117
9.5.9 Data Transfer Functions....117
9.5.9.1 Read header plus data....118
9.5.9.2 Read data....118
9.5.9.3 Write header plus data....118
9.5.9.4 Write header....118
9.5.9.5 Write check header plus data....118
9.5.9.6 Write check data....118
9.6 Disk Completion Monitor....118
9.7 Secure Digital (SD) Disk Controller....119
9.8 Secure Digital (SD) Capability Issues....119
9.8.1 SIMH Cylinder/Head/Sector (CHS) Disk Addressing....119
9.8.2 Cylinder/Head/Sector (CHS) Disk Address Increment....120
9.8.3 SIMH “Sector” Size....121
9.8.4 Disk Drive Parameters....121
9.8.5 RPxx/RMxx Disk Addressing....122
9.8.6 SD Disk Organization....123
10 LP20 Printer Controller....125
10.1 Control/Status A Register (CSRA)....125
10.2 Control/Status B Register (CSRB)....126
10.3 Bus Address Register (BAR)....127
10.4 Byte Counter Register (BCTR)....127

10.5 Page Counter Register (PCTR) 128
10.6 RAM Data Register (RAMD) 128
10.7 Column Counter Register (CCTR) / Character Buffer Register (CBUF) 129
10.8 Checksum Register (CKSM) / Printer Data Register (PDAT) 130
11 Executive Mode and IO Instructions 131
11.1 Executive Mode Instructions 132
11.1.1 Arithmetic Processor Interface (APR) Instructions 132
11.1.1.1 APR Identification (APRID) 132
11.1.1.2 Write APR (WRAPR) 133
11.1.1.3 Read APR (RDAPR) conditions 134
11.1.2 Priority Interrupt Controller (PI) Instructions 135
11.1.2.1 Write Priority Interrupt (WRPI) 135
11.1.2.2 Read Priority Interrupt (RDPI) 136
11.1.3 User Base Register (UBR) Instructions 137
11.1.3.1 Write to the User Base Register (WRUBR) 137
11.1.3.2 Read User Base Register (RDUBR) 138
11.1.4 Clear Page Table Entry (CLRPT) 139
11.1.5 Executive Base Register (EBR) Instructions 140
11.1.5.1 Write to the Executive Base Register (WREBR) 140
11.1.5.2 Read the Executive Base Register (RDEBR) 140
11.1.6 Shared Pointer Table (SPT) Base Address Register 142
11.1.6.1 Read Shared Pointer Table Base Address Register (RDSPB) 142
11.1.6.2 Write Shared Pointer Table Base Address Register (WRSPB) 142
11.1.7 Core Status Table (CST) Instructions 143
11.1.7.1 Read Core Status Table Base Register (RDCSB) 143
11.1.7.2 Write Core Status Table Base Register (WRCSB) 143
11.1.7.3 Write Core Status Table Mask Register (RDCSTM) 143
11.1.7.4 Write Core Status Table Mask Register (WRCSTM) 144
11.1.7.5 Read Core Status Table Process Use Register (RDPUR) 144
11.1.7.6 Write Core Status Table Process Use Register (WRPUR) 144
11.1.8 Timebase Instructions 144
11.1.8.1 RDTIM – Read Timebase 144
11.1.8.2 WRTIM - Write Timebase 145
11.1.9 Interval Timer Instructions 146
11.1.9.1 RDINT – Read Interval Timer 146
11.1.9.2 WRINT - Write Interval Timer 146
11.1.10 Halt Status Block Address Instructions 146
11.1.10.1 RDHSB - Read Halt Status Block Address 146
11.1.10.2 WRHSB - Write Halt Status Block Address 147
12 Diagnostics 148
12.1 Diagnostics Summary 149
12.1.1 DSUBA DECSYSTEM 2020 UNIBUS ADAPTER EXERCISER 149
13 Building the KS10 FPGA System 150
13.1 Tools 150
13.1.1 FTDI USB drivers. 150
13.1.1.1 FTDI Hardware 150
13.1.2 FPGA Tools 151
13.1.2.1 Xilinx ISE Webpack Version 14.7 151
13.1.2.2 Icarus Verilog 151
13.1.2.3 FPGA JTAG Programming Cable 151
13.1.2.4 Xilinx Chipscope (optional) 152
13.1.3 Software Tools 152
13.1.3.1 GCC tool suite ARM processors 152
13.1.3.2 GDB Debugger for ARM processors 152

13.1.3.3 OpenOCD On-Chip Debugger....153
13.1.3.4 Eclipse Integrated Development Environment....153

List of Figures

Figure 1 – DEC KS10 13
Figure 2 – DEC KS10 with Covers Removed 13
Figure 3 – KS10 FPGA Design Hierarchy 14
Figure 4 – KS10 FPGA CPU Block Diagram 15
Figure 5 – Timer Interface Diagram 15
Figure 6 – APR Interface Diagram 16
Figure 7 – ALU Interface Diagram 17
Figure 8 – DEC KS10 ALU Implementation 18
Figure 9 – KS10 FPGA ALU Implementation 18
Figure 10 – Microsequencer Interface 19
Figure 11 – Microsequencer Block Diagram 20
Figure 12 – Dispatch Interface Diagram 22
Figure 13 – Skip Interface Diagram 23
Figure 14 – Stack Interface Diagram 23
Figure 15 – Stack Block Diagram 24
Figure 16 – DBUS Interface Diagram 24
Figure 17 – DBUS Block Diagram 25
Figure 18 – SCAD Interface Diagram 25
Figure 19 – SCAD Block Diagram 26
Figure 20 – KS10 FPGA Bus Architecture 28
Figure 21 – KS10 FPGA Address Bus Illustration 29
Figure 22 – APR Interrupt Bus Cycle 31
Figure 23 – External Interrupt Bus Cycle 31
Figure 24 – KS10 Console Interface Block Diagram 42
Figure 25 – Console Microcontroller and KS10 FPGA Interface 43
Figure 26 – Console Microcontroller Interface Read/Write Cycle Timing Diagram 44
Figure 27 – Console Control/Status Register 47
Figure 28 – Console Data Register 49
Figure 29 – Console Address Register 49
Figure 30 – Console Instruction Register 50
Figure 31 – DZ11 Console Control Register 50
Figure 32 – RH11 Console Control Register 51
Figure 33 – RH11 Debug Register 51
Figure 27 – Debug Control/Status Register (DCSR) 52
Figure 34 – Debug Breakpoint Address Register (DBAR) 54
Figure 34 – Breakpoint Mask Register (DBMR) 54
Figure 35 – Instruction Trace Register 55
Figure 36 – KS10 Control State Diagram 57
Figure 37 – KS10 CTY Input Word (KS10 Memory Address 000032) 61
Figure 38 – KS10 CTY Output Word (KS10 Memory Address 000033) 62
Figure 39 – Memory Status Register (Read) 66
Figure 40 – Memory Status Register (Write) 66
Figure 41 - SSRAM Read Timing Diagram 68
Figure 42 - SSRAM Write Timing Diagram 69
Figure 43 – IO Bus Byte and Word Translation 70
Figure 44 – IO Bridge Paging RAM Write 71
Figure 45 – IO Bridge Paging RAM Read 71
Figure 46 – IO Bus Paging 72
Figure 47 – IO Bridge Control Status Register (UBACSR) 73
Figure 48 – IO Bridge Maintenance Register (UBAMR) 74
Figure 49 – DZ11 Block Diagram 76
Figure 50 – DZ11 Control and Status Register (CSR) 76
Figure 51 – DZ11 Receiver Buffer Register (RBUF) 78

Figure 52 – DZ11 Line Parameter Register (LPR) 79
Figure 53 – DZ11 Transmit Control Register TCR) 81
Figure 54 – Fractional-N Divider Block Diagram 83
Figure 55 – KS10 FPGA Disk Subsystem Architecture 85
Figure 56 – RH11 Control and Status Register #1 (RHCS1) 86
Figure 57 – RH11 Word Count Register (RHWC) 88
Figure 58 – RH11 Bus Address Register (RPBA) 88
Figure 59 – RH11 Control and Status Register #2 (RHCS2) 89
Figure 60 – RH11 Data Buffer Register (RPDB) 91
Figure 61 – RP Control and Status Register #1 (RPCS1) 93
Figure 62 – RP Disk Address Register (RPDA) 95
Figure 63 – RP Drive Status Register (RPDS) 96
Figure 64 – RP Error Register #1 (RPER1) 99
Figure 65 – RP Attention Summary Register (RPAS) 102
Figure 66 – RP Look Ahead Register (RPLA) 103
Figure 67 – RP Maintenance Register (RPMR) 104
Figure 68 – RP Drive Type Register (RPDT) 106
Figure 69 – RP Serial Number Register (RPSN) 107
Figure 70 – RP Offset Register (RPOF) 107
Figure 71 – RP Desired Cylinder Register (RPDC) 109
Figure 72 – RP Current Cylinder Register (RPCC) 109
Figure 73 – RP Error Status #2 (RPER2) 110
Figure 74 – RP Error Status #3 (RPER3) 112
Figure 75 – RP Error Position Register (RPEC1) 113
Figure 76 – RP Error Pattern Register (RPEC2) 113
Figure 77 – Sector Header Word #1 118
Figure 78 – Sector Header Word #2 118
Figure 79 – Sector Increment Algorithm 121
Figure 80 – SIMH/PDP10 Disk Image Hex Dump 121
Figure 81 – Disk Cylinder, Track, and Sector 123
Figure 82 – SD Card Storage Allocation 124
Figure 83 – Control/Status A Register (CSRA) 125
Figure 84 – Control/Status B Register (CSRB) 126
Figure 85 – Bus Address Register (BAR) 127
Figure 86 – Byte Count Register (BCTR) 128
Figure 87 – Page Count Register (PCTR) 128
Figure 88 – RAM Data Register (RAMD) 128
Figure 89 – Column Counter Register (CCTR) / Character Buffer Register (CBUF) 129
Figure 90 – Printer Data Register (PDAT) / Checksum Register (CKSM) 130
Figure 91 – APRID Instruction 132
Figure 92 – WRAPR Instruction 133
Figure 93 – RDAPR Instruction 134
Figure 94 – WRPI Instruction 135
Figure 95 – RDPI Instruction 136
Figure 96 – WRUBR (DATO PAG) Instruction 137
Figure 97 – RDUBR (DATI PAG) Instruction 138
Figure 98 – CLRPT Instruction 139
Figure 99 – WREBR (CONO PAG) Instruction 140
Figure 100 – RDEBR (CONI PAG) Instruction 140
Figure 101 – RDSPB Instruction 142
Figure 102 – WRSPB Instruction 142
Figure 103 – RDCSB Instruction 143
Figure 104 – WRCSB Instruction 143
Figure 105 – RDCSTM Instruction 143

Figure 106 – WRCSTM Instruction 144
Figure 107 – RDPUR Instruction 144
Figure 108 – WRPUR Instruction 144
Figure 109 – RDTIM Instruction 145
Figure 110 – WRTIM Instruction 145
Figure 111 – RDINT Instruction 146
Figure 112 – WRINT Instruction 146
Figure 113 – RDHSB Instruction 147
Figure 114 – WRHSB Instruction 147
Figure 115 – Directory Structure 150

List of Tables

Table 1 – APR Flags 17
Table 2 – KS10 Microcode Variations 21
Table 3 – “Page Fail” Dispatches 26
Table 4 – Bus Arbiter Operations 28
Table 5 – Address Flag Definitions 29
Table 6 – KS10 Bus Cycles 30
Table 7 – KS10 Console Command Summary 34
Table 8 – Console Interface Register Memory Map 44
Table 9 – Console Control/Status Register Definitions 47
Table 10 – DZ11 Console Control Register Definition 50
Table 11 – RH11 Console Control Register Definition 51
Table 12 – Console Control/Status Register Definitions 51
Table 9 – Debug Control/Status Register (DCSR) Definitions 52
Table 13 – Debug Breakpoint Address Register (DBAR) Definitions 54
Table 13 – Breakpoint Mask Register Definitions 55
Table 14 – Instruction Trace Register Definitions 56
Table 15 – Console Firmware Version Register Definitions 56
Table 16 – Control Operation from Halt State 57
Table 17 – KS10/Console Communications Area 59
Table 18 – KS10 Halt Switch Word (KS10 Memory Address 000030) 60
Table 19 – KS10 Keep Alive Word (KS10 Memory Address 000031) 60
Table 20 – KS10 CTY Input Word (KS10 Memory Address 000032) 61
Table 21 – KS10 CTY Output Word (KS10 Memory Address 000032) 62
Table 22 – KS10 KLINIK Input Word (KS10 Memory Address 000034) 63
Table 23 – KS10 KLINIK Output Word (KS10 Memory Address 000035) 63
Table 24 – KS10 RH11 Address Word (KS10 Memory Address 000036) 64
Table 25 – KS10 Boot Unit Number Word (KS10 Memory Address 000037) 64
Table 26 – KS10 Boot Magtape Parameter Word (KS10 Memory Address 000040) 65
Table 27 – Memory Status Register Definitions 66
Table 28 – SSRAM Read Timing Parameters 68
Table 29 – SSRAM Write Timing Parameters 69
Table 30 – UBA Address Translation 70
Table 31 – IO Bridge Paging RAM Definitions 71
Table 32 – IO Bridge Control Status Register (UBACSR) Definitions 73
Table 33 – IO Bridge Maintenance Register (UBAMR) Definitions 74
Table 34 – DZ11 Configuration 75
Table 35 – DZ11 Control and Status Register (CSR) – IO Address 760010 77
Table 36 – DZ11 Receiver Buffer Register (RBUF) – IO Address 760012 78
Table 37 – DZ11 Line Parameter Register (LPR) – IO Address 760012 79
Table 38 – DZ11 Transmit Control Register (TCR) – IO Address 760014 81
Table 39 – DZ11 Modem Status Register (TDR) – IO Address 760016 81
Table 40 – DZ11 Transmit Data Register (TDR) – IO Address 760016 82
Table 41 – RH11 Configuration 84
Table 42 – RH11 Controller Register Summary 86
Table 43 – RH11 Control and Status Register #1 (RHCS1) – IO Address 776700 86
Table 44 – RH11 Word Count Register (RHWC) – IO Address 776702 88
Table 45 – RH11 Bus Address Register (RHBA) – IO Address 776704 88
Table 46 – RH11 Control and Status Register #2 (RHCS2) – IO Address 776710 89
Table 47 – RH11 Data Buffer Register (RHDB) – IO Address 776722 91
Table 48 – RPxx Device Registers 91
Table 49 – Massbus Register Address Cross Reference 92
Table 50 – RP Control and Status Register #1 (RPCS1) – IO Address 776700 94
Table 51 – RP Disk Address Register (RPDA) – IO Address 776706 95

Table 52 – RP Drive Status Register (RPDS) – IO Address 776712 97
Table 53 – RP Error Register #1 (RPER1) – IO Address 776714 99
Table 54 – RP Attention Summary (RPAS) – IO Address 776716 102
Table 55 – RP Look Ahead (RPLA) – IO Address 776720 103
Table 56 – RP Maintenance Register (RPMR) – IO Address 776724 104
Table 57 – RP Drive Type Register (RPDT) – IO Address 776726 106
Table 58 – RP Serial Number Register (RPSN) – IO Address 776730 107
Table 59 – RP Offset Register (RPOF) – IO Address 776732 107
Table 60 – RP Desired Cylinder (RPDC) – IO Address 776734 109
Table 61 – RP Current Cylinder Register (RPCC) – IO Address 776736 109
Table 62 – RP Error Status Register #2 (RPER2) – IO Address 776740 110
Table 63 – RP Error Status Register #1 (RPER3) – IO Address 776742 112
Table 64 – RP Error Position Register (RPEC1) – IO Address 776744 113
Table 65 – RP Error Pattern Register (RPEC2) – IO Address 776746 114
Table 66 - RP06 Seek Timing Simulation 115
Table 67 – Disk Parameters 122
Table 68 – Control/Status A Register (CSRA) – IO Address 775400 125
Table 69 – Control/Status B Register (CSRB) – IO Address 775402 127
Table 70 – Bus Address Register (BAR) – IO Address 775404 127
Table 71 – Byte Count Register (BCTR) – IO Address 775406 128
Table 72 – Page Count Register (BCTR) – IO Address 775410 128
Table 73 – RAM Data Register (RAMD) – IO Address 775412 129
Table 74 – CCTR and CBUF Register 129
Table 75 – PDAT and CKSM Registers 130
Table 76 – APRID Bit Definitions 132
Table 77 – CONO APR (WRAPR) Bit Definitions 133
Table 78 – CONI APR (RDAPR) Bit Definitions 134
Table 79 – CONP PI (WRPI) Bit Definitions 135
Table 80 – CONI PI (RDPI) Bit Definitions 136
Table 81 – WRUBR (DATO PAG) Bit Definitions 137
Table 82 – RDUBR (DATI PAG) Bit Definitions 138
Table 83 – WREBR (CONO PAG) Bit Definitions ..... **Error! Bookmark not defined.**
Table 84 – RDEBR (CONI PAG) Bit Definitions 141
Table 85 – Diagnostic Status 148
Table 86 – SMMON Command Summary 154

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 12.5 MHz clock cycle.

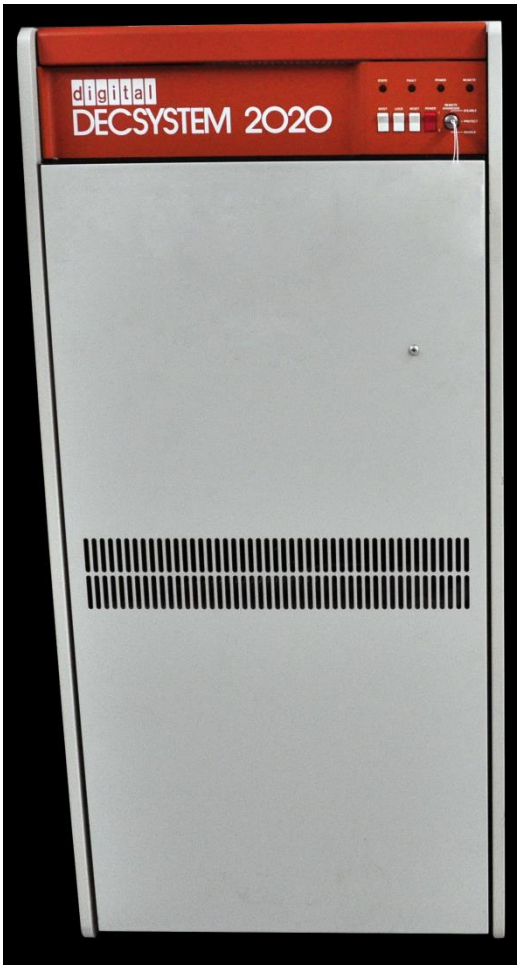


Figure 1 – DEC KS10

(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.

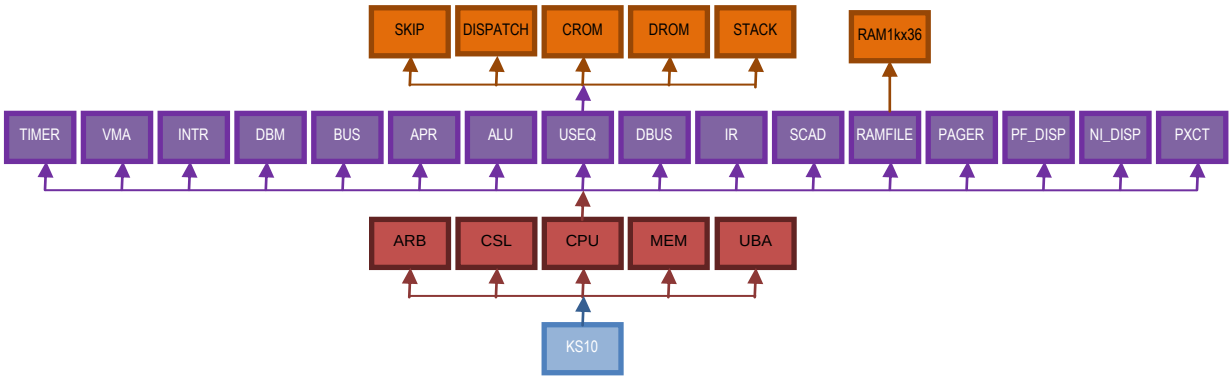


Figure 3 – KS10 FPGA Design Hierarchy

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.

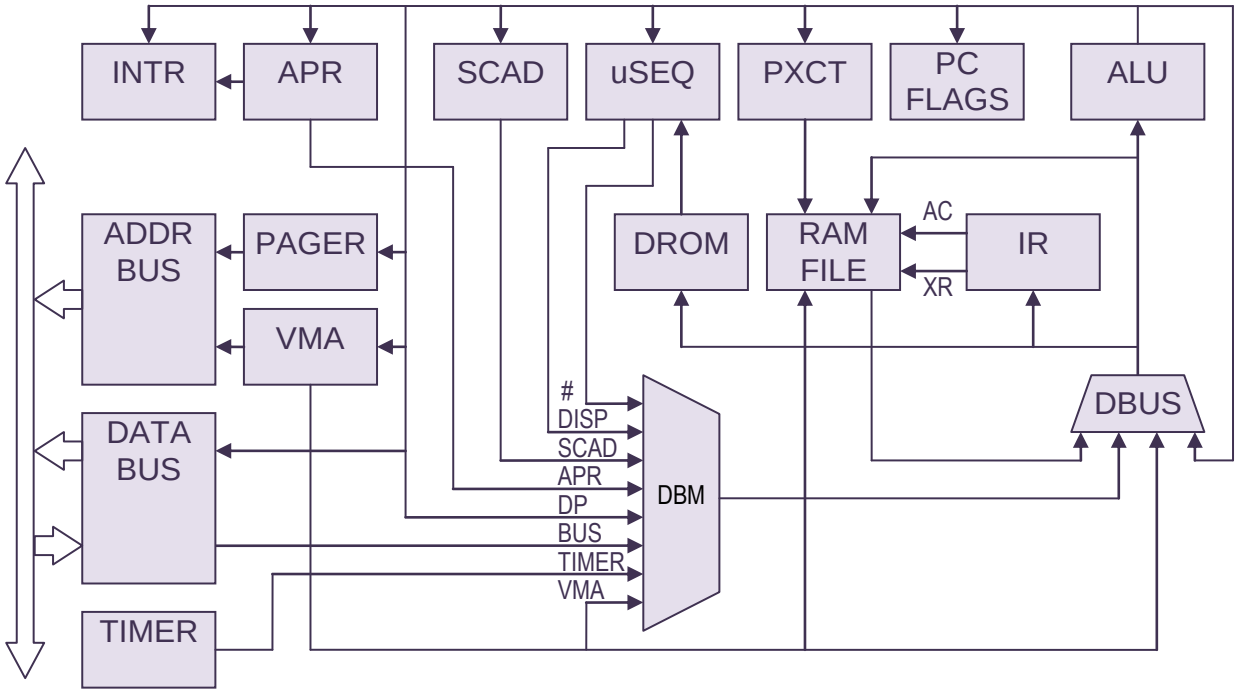


Figure 4 – KS10 FPGA CPU Block Diagram

2.2.3.1KS10 CPU Interval Timer (TIMER)

The PDP10 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.

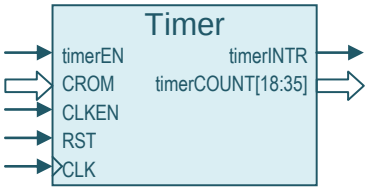


Figure 5 – Timer Interface Diagram

The timerINTR output is asserted for a single clock cycle on a timer overflow. i.e., whenever the MSB changes from '1' to '0'.

The KS10 hardware does not use the two LSBS or the upper 6-bits of the of the timerCOUNT output. These outputs must be set to zero. The microcode does read all of the bits and does not perform any masking.

The original KS10 interval timer was 12 bits and operated with a 4.096 MHz clock. The 12-bit timer generates an interrupt exactly every one millisecond.

This did not work well in practice. The RDTIME instruction also needs to convert timer output to time in 10 microsecond increments which would require a division by 40.96. Therefore the timer frequency was

changed by ECO to 4.100 MHz and the microcode was modified to perform a division by 41 to support the RDTIME instruction.

You can find the details of DEC KS10 timer design (and its problems) in a DEC memo authored by Dan Murphy entitled "Clocks for Dolphin" on bitsavers.org

The KS10 FPGA does not use a 4.100 MHz clock to drive the timer like the DEC KS10 implementation - this would have required an additional oscillator or crystal frequency reference. Instead, the KS10 FPGA implements a Fractional N divider to divide the 50 MHz clock using a 32-bit accumulator. The use of an accumulator instead of divider keeps the average output frequency correct. In other words, the accumulator maintains the fractional time when the count overflows.

The output of the Fractional N Divider is a 4.100 MHz clock that is *on-average* the correct frequency.

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. FPGA gates are cheap.

2.2.3.2KS10 CPU Virtual Memory Address (VMA)

TBD.

2.2.3.3KS10 CPU Priority Interrupt Controller (INTR)

TBD.

2.2.3.4KS10 CPU DBM Mux (DBM)

TBD.

2.2.3.5KS10 CPU Backplane Interface (BUS)

TBD.

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 6.

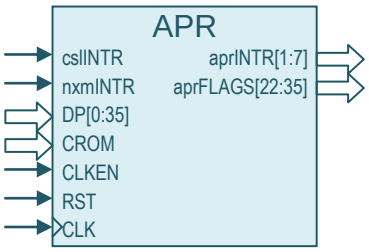


Figure 6 – APR Interface Diagram

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 11.1.1.

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**	**Notes**
22	Trap enable	Enable traps
23	Page enable	Enable paging
24	APR flag 24	This flag can be read/written but is otherwise not implemented.
25	Interrupt to KS10 console	
26	Power fail interrupt	This flag can be read/written but is otherwise not implemented.
27	NXM interrupt	This flag is asserted when non-existent memory is accessed.
28	Uncorrectable memory error	This flag can be read/written but is otherwise not implemented.
29	Correctable memory error	This flag can be read/written but is otherwise not implemented.
30	Interval timer interrupt	Asserted every 1.0 millisecond.
31	Interrupt from KS10 console	Asserted when the console transfers character to the KS10.
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 7.

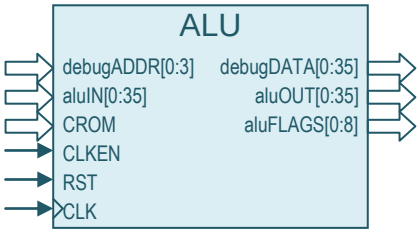


Figure 7 – ALU Interface Diagram

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 8.

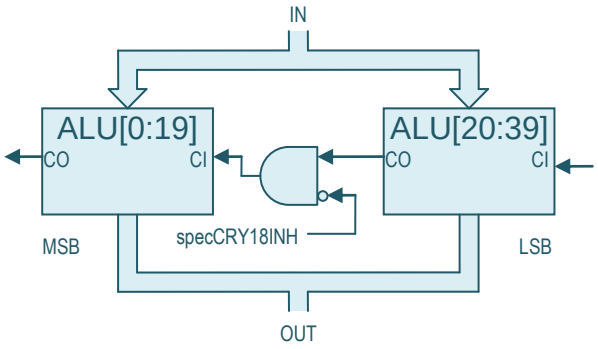


Figure 8 – DEC KS10 ALU Implementation

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 9.

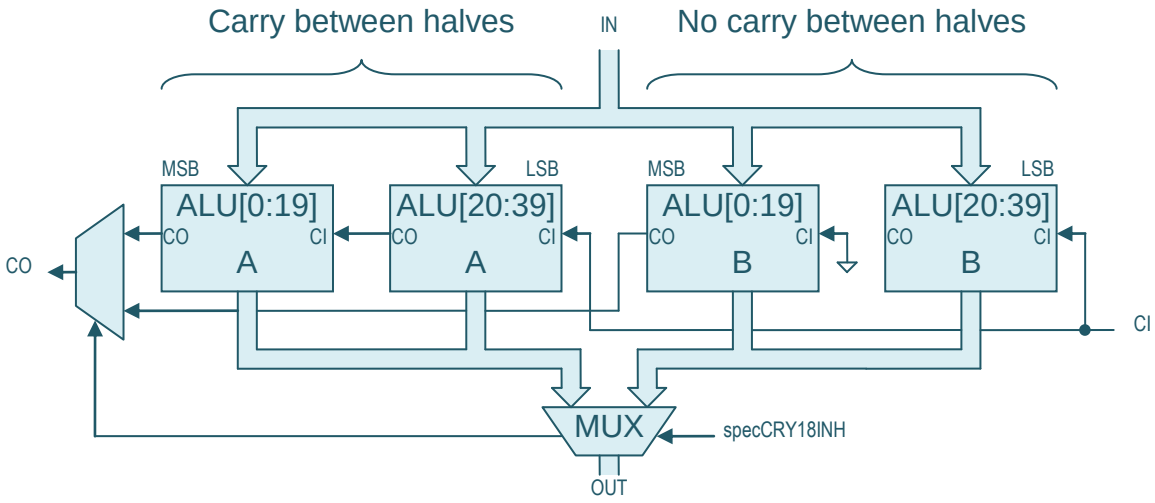


Figure 9 – KS10 FPGA ALU Implementation

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 8 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 10.

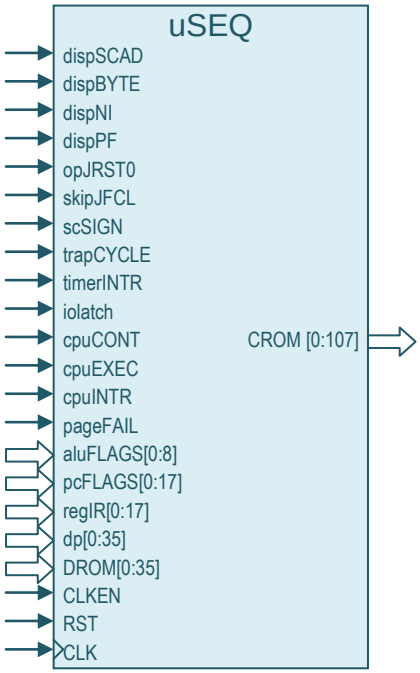


Figure 10 – Microsequencer Interface

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 in the microcode which is 7777 (octal). 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.

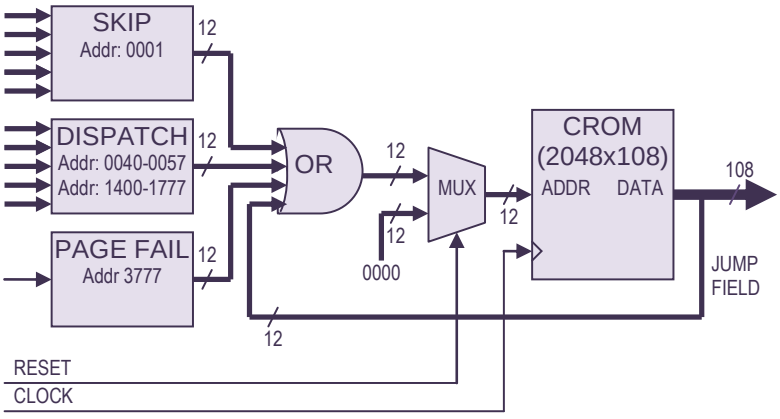


Figure 11 – Microsequencer Block Diagram

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.

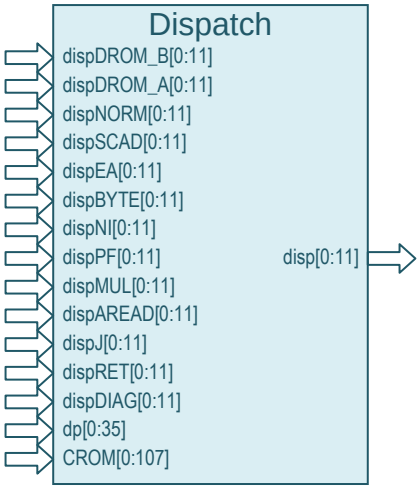


Figure 12 – Dispatch Interface Diagram

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.

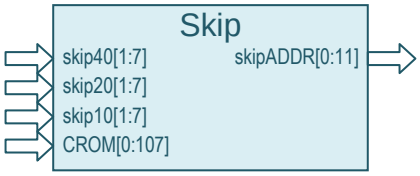


Figure 13 – Skip Interface Diagram

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 14 while a block diagram is illustrated below in Figure 15.

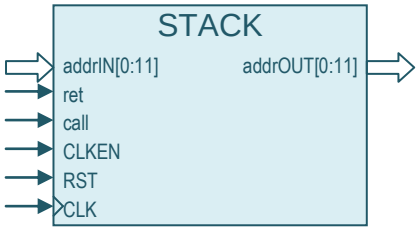


Figure 14 – Stack Interface Diagram

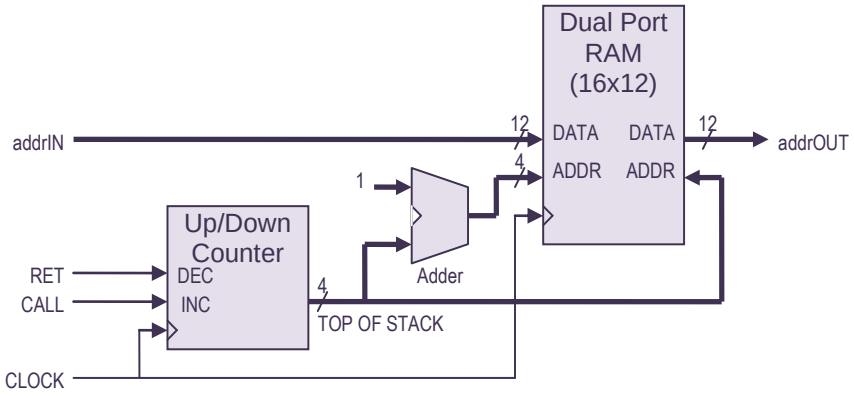


Figure 15 – Stack Block Diagram

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.

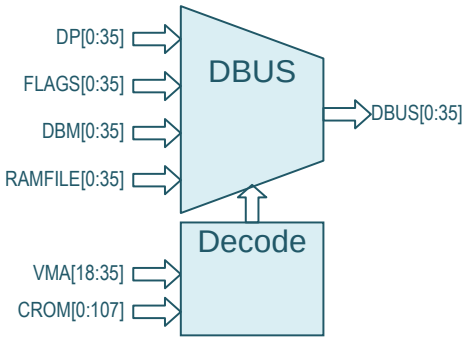


Figure 16 – DBUS Interface Diagram

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 17

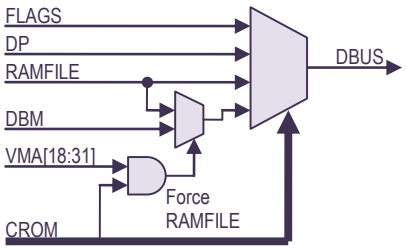


Figure 17 – DBUS Block Diagram

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.

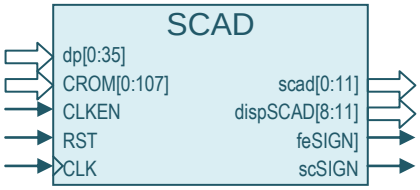


Figure 18 – SCAD Interface Diagram

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 19.

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

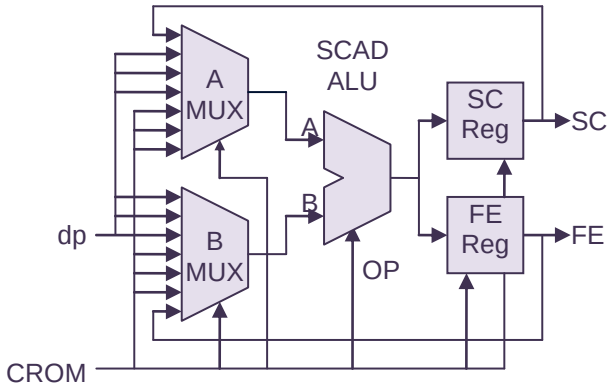


Figure 19 – SCAD Block Diagram

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.

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)

TBD.

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 3 – "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 4 summarizes the operation of the Bus Arbiter.

Table 4 – 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.

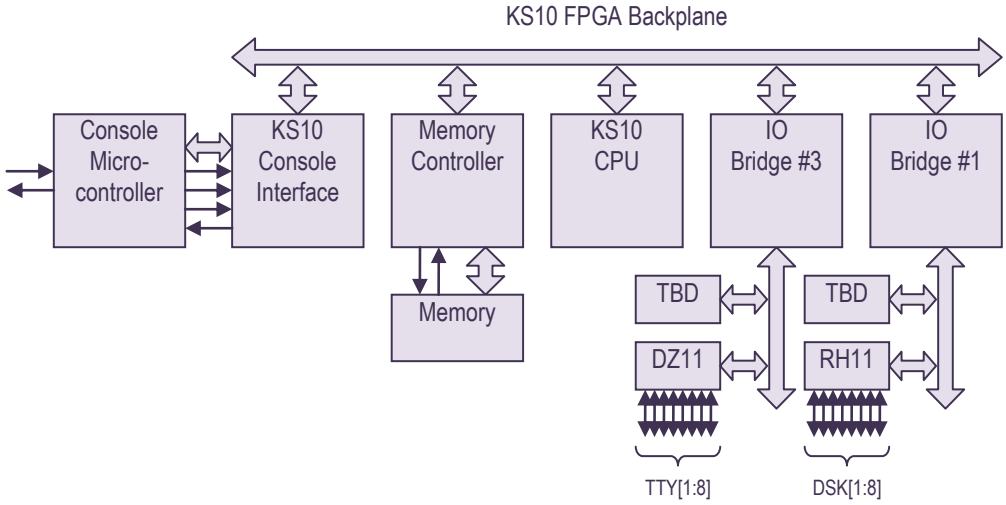


Figure 20 – KS10 FPGA Bus Architecture

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:

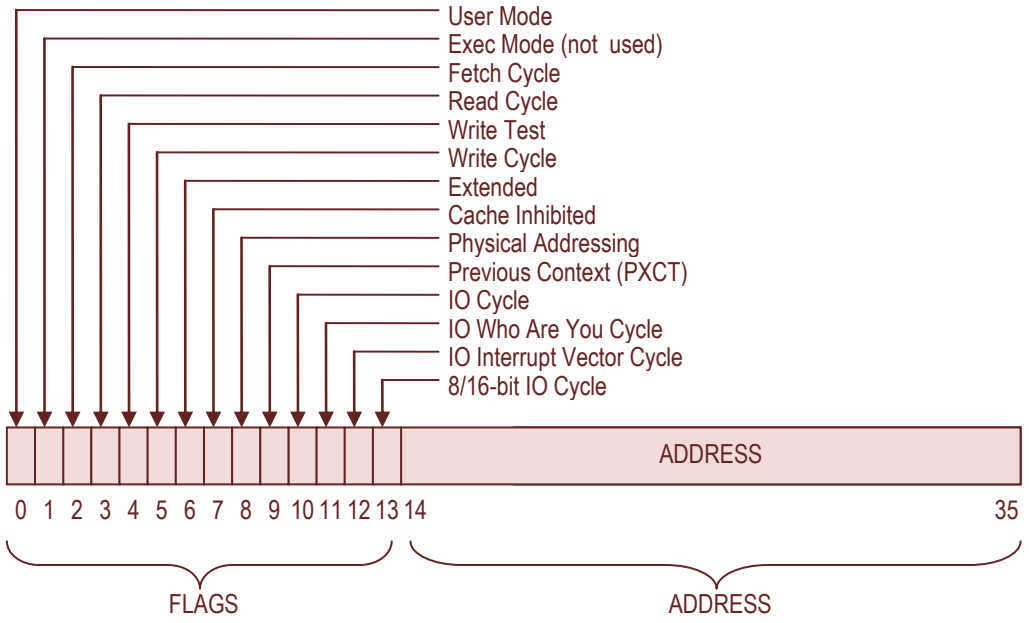


Figure 21 – KS10 FPGA Address Bus Illustration

Table 5 – Address Flag Definitions		
Bit	**Mnemonic**	**Description**
0	USER	1 = User Mode. This signal probably isn't useful for anything outside of the CPU.
1	EXEC	1 = Exec Mode. Not used. Always zero.
2	FETCH	1 = Instruction Fetch.
3	READ CYCLE	1 = Read Cycle (either IO or Memory)
4	WRTEST	1 = 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 CYCLE	1 = Write Cycle (either IO or Memory)

Table 5 – Address Flag Definitions		
Bit	**Mnemonic**	**Description**
6	EXTENDED	1 = Extended Address Cycle. This signal probably isn't useful for anything outside of the CPU.
7	CACHEINH	1 = Cache inhibit. This signal probably isn't useful for anything outside of the CPU or Cache Controller.
8	PHYSICAL	1 = Physical Address.
9	PREVIOUS	1 = VMA Previous Context (PXCT). This signal probably isn't useful for anything outside of the CPU.
10	IO CYCLE	1 = IO Cycle, 0 = Memory Cycle
11	WRU CYCLE	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 that 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 CYCLE	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 CYCLE	Unibus Byte IO operation. The LSB of the address is used to determine if the upper or lower byte is to be accessed.

Table 6 – KS10 Bus Cycles								
CYCLE TYPE	**FETCH BIT 2**	**READ BIT 3**	**WRTEST BIT4**	**WRITE BIT 5**	**PHYS BIT 8**	**IO BIT 10**	**WRU BIT11**	**VECT BIT 12**
Instruction Fetch	1	1	0	0	0	0	0	0
Memory Read	0/1	1	0	0	0/1	0	0	0
Memory Write	0	0	0/1	1	0/1	0	0	0
IO Read	0	1	0	0	1	1	0	0
IO Write	0	0	0	1	1	1	0	0
WRU Cycle	0	1	0	0	1	1	1	0
Vector Cycle		1	0	0	1	1	0	1

3.2 KS10 FPGA Bus Cycles

3.2.1 APR / Timer Interrupt

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

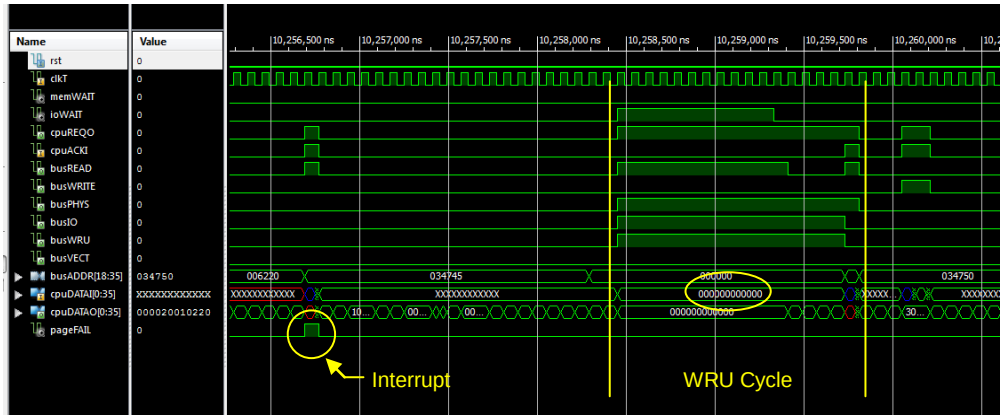


Figure 22 – APR Interrupt Bus Cycle

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.

3.2.2 KS10 FPGA External Interrupt

The KS10 FPGA external interrupt bus cycle captured from a DZ11 interrupt cycle is shown below in Figure 23.

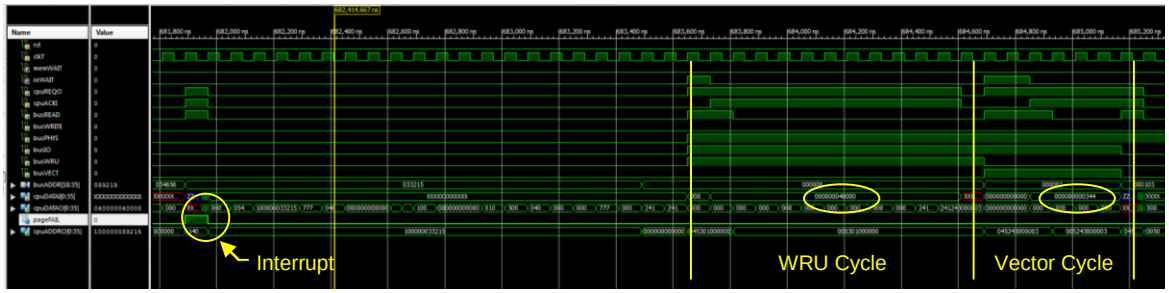


Figure 23 – External Interrupt Bus Cycle

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 Microcontroller

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.

The KS10 Console provides boot and debug functionality to the KS10 Processor. Because of the extensive changes that are required for the KS10 FPGA implementation, the KS10 FPGA Console is significantly different than the original DEC KS10 Console and uses a more modern microcontroller.

The console microcontroller controls how the firmware is loaded into the FPGA, loads the KS10 executable into memory, and starts the KS10 processor. Once the KS10 FPGA is operating, the console provides the CTY interface. The KS10 executable can be either one of the diagnostic programs or the monitor program.

The console microcontroller is a Texas Instruments/Stellaris LM3S9B96 (ARM Cortex-M3) single-chip microcontroller. This microcontroller operates at 50 MHz and includes:

- 256 KB Flash memory
- 96 KB SRAM
- External Peripheral Interface (EPI). The KS10 FPGA interface is memory mapped using the EPI interface.
- 3x Universal Asynchronous Receivers/Transmitters (UARTs). These are used for serial communication
- Synchronous Serial Interface (SSI). The Secure Digital (SD) Card interface is built on top of the SSI.
- Ethernet controller.
- Universal Serial Bus (USB) controller.

The Console Microcontroller is implemented using a mix of C and C++ software – most of the KS10 console software is implemented in C++ while the COTS FAT32 filesystem software is implemented in plain old C. All code is built using a GCC embedded ARM toolset. All the software interacts directly with the hardware – there is no operating system to provide support services. The code can be loaded into the LM3S9B96 microcontroller using the **openocd** application via the USB hardware that is included on the board.

Whereas the DEC KS10 could boot from 9 track magtapes or RPxx disks attached to the Massbus, the KS10 FGPA can boot from a FAT32 formatted SD Card connected to the Console, or from an SD Card emulating an RPxx disk attached to the KS10 FPGA. Both SD Card interfaces support the Secure Digital Card High Capacity (SDHC) devices. Software and firmware to support the other SD Card formats is untested and should be considered experimental.

The console software initializes the microcontroller, performs a few self-tests on the hardware, loads the Xilinx FPGA with firmware, displays the contents of the KS10 FPGA Firmware Revision Register, initializes the SD Card device, and mounts a FAT32 filesystem on the SDHC Card device.

When the console has been initialized, the console microcontroller starts the KS10 FPGA and the KS10 FPGA begins executing microcode. The KS10 microcode initializes the Arithmetic Logic Unit (ALU) registers, initializes variables stored in the RAMFILE, and performs a basic built-in test of the ALU. After this is completed, the KS10 FPGA microcode halts.

When the console microcontroller detects that the KS10 has halted, the console checks the KS10 status to ensure that the KS10 selftests have completed successfully, and waits for the operator to select a boot device and read the boot code into KS10 memory.

The console microcontroller can load code using PDP-10 Magtape images (.SAV files) from the SD Card.

Lastly, the console microcontroller sets a starting address for the KS10 to begin execution and starts the KS10. For some boot devices, the starting address can be obtained by reading the boot media.

The console boot sequence is as follows:

```
CPU : Console alive.
CPU : Device identifier is 0x10040203.
NET : MAC Address is 01:1a:b6:00:64:00
NET : Successfully started telnet task.
NET : Telnet started.
FPGA: Programming with firmware. . . . . .
FPGA: Programmed successfully.
CPU : EPI interface initialized.
FPGA: Firmware is REV 00.07
FPGA: Registers tested completed successfully.
SDHC: Card inserted.
SDHC: Card initialized successfully.
SDHC: FAT filesystem successfully mounted on SD media.
KS10: Halted.
KS10: Halt Cause: Microcode Startup. (PC=000000)
KS10>
KS10> SD DIR
05/29/2013  07:59 PM             2009 STATUS.TXT
07/11/2013  08:43 PM   <DIR>        0 DIAG
07/03/2013  06:36 PM            25095 DIAG/DSKEA.SAV
07/03/2013  06:36 PM            18520 DIAG/DSKEB.SAV
07/03/2013  06:36 PM            15860 DIAG/DSKEC.SAV
07/03/2013  07:04 PM             8730 DIAG/DSKFA.SAV
01/26/2013  09:16 AM            32625 DIAG/DSDZA.SAV
06/29/2013  04:39 PM            11900 DIAG/DSKAA.SAV
06/29/2013  04:40 PM            13385 DIAG/DSKAB.SAV
06/29/2013  04:40 PM             9285 DIAG/DSKAC.SAV
06/29/2013  04:40 PM            10500 DIAG/DSKAD.SAV
06/29/2013  04:40 PM             7780 DIAG/DSKAE.SAV
06/29/2013  04:40 PM            10225 DIAG/DSKAF.SAV
07/03/2013  06:33 PM             4560 DIAG/DSKAG.SAV
07/03/2013  06:33 PM            20775 DIAG/DSKAH.SAV
07/03/2013  06:33 PM            23965 DIAG/DSKAI.SAV
07/03/2013  06:33 PM            16375 DIAG/DSKAJ.SAV
07/03/2013  06:33 PM            30575 DIAG/DSKAK.SAV
07/03/2013  06:34 PM            33200 DIAG/DSKAL.SAV
07/03/2013  06:34 PM            20710 DIAG/DSKAM.SAV
07/03/2013  06:35 PM            62580 DIAG/DSKBA.SAV
07/03/2013  06:35 PM            16535 DIAG/DSKCA.SAV
07/03/2013  06:35 PM             8070 DIAG/DSKCB.SAV
07/03/2013  06:35 PM            16655 DIAG/DSKCC.SAV
07/03/2013  06:36 PM            31190 DIAG/DSKCD.SAV
07/03/2013  06:36 PM            24410 DIAG/DSKCE.SAV
```

```
07/03/2013  06:36 PM            45875 DIAG/DSKCF.SAV
07/03/2013  06:36 PM            30590 DIAG/DSKCG.SAV
07/03/2013  06:36 PM            70130 DIAG/DSKDA.SAV
KS10> BT DIAG/DSKDA.SAV
…
… The KS10 begins executing code at the starting address stored in the 'SAV' file.
…
```

This SD Card has mostly diagnostic programs loaded on it.

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 an SD Card attached to the Console Microcontroller or a disk drive attached to a Unibus Adapter. Magtapes are not supported.

The console microcontroller requires that the SD Card had been formatted with a FAT32 filesystem. It should be noted that the SD Card and the FAT32 filesystem is read-only. The KS10 Console code cannot change the data on the SD Card.

For now, the boot code *must* be in PDP10 .SAV file format. The PDP10 .SAV file format has been selected because it is portable and is fairly space efficient. Other file formats may be added in the future.

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 "ST" 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 7 – 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 RPxx disk drive. (See DS command).

Table 7 – KS10 Console Command Summary		
Command	**Argument**	**Description**
BT	1	Boot Diagnostic Monitor This command boots to the Diagnostic Monitor the from the selected RPxx disk drive. (See DS command).
CE	{1 | 0}	Cache Enable Enable/Disable KS10 cache.
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	xx	Clock Pulse. Not implemented. Clock the CPU clock xx times.
CS		Clock Start Not implemented. Start the CPU clock.
DB	xx	Deposit Bus Not implemented. Deposit xx onto KS10 bus.
DC	xx	Deposit CRAM Not implemented. Deposit xx into CRAM. Address previously loaded by LC command.
DF	xx	Deposit into CRAM memory. Not implemented. Deposit xx into CRAM address. Address and diagnostic function previous loaded by LC and LF command.
DK	xx	Deposit into 8080 memory. Not implemented. Deposit xx (8 bits) into 8080 memory. Address previously loaded by LK command.
DI	xx	Deposit into IO register Deposit 16-, 18- or 36-bit argument into an I/O register. Address previously loaded by an LK command.
DM	xx	Deposit KS10 Memory. Deposit 36-bit argument into KS10 memory. Address previously loaded by LA command.

Table 7 – KS10 Console Command Summary		
Command	**Argument**	**Description**
DN	xx	Deposit Next Deposit xx into next KS10 memory address. Address previously loaded by LA command.
DR		Deposit into 8080 Register. Not implemented.
DS		Show RH11 Boot Parameters Boot Parameters are: UBA = 1 BASE = 776700 UNIT = 0
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.
EC		Examine CRAM register Not implemented Examine contents of CRAM register.
EK		Examine contents of 8080 memory. Not implemented Address previously loaded by LK command.
EI		Examine I/O register. Address previously loaded 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
EM		Examine KS10 memory. Address previously loaded by LA command.
EM	xx	Examine KS10 memory. Examine contents of KS10 memory address xx.

Table 7 – 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
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	xx	KLINIK Not implemented. Enable remote link with mode xx.
LA	xx	Load Memory Address. Set KS10 FPGA memory address. The valid address range is 0000000-3777777.
LB		Load Bootstrap to Monitor. This command boots to the Monitor the from the selected RPxx disk drive. (See DS command). 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 RPxx disk drive. (See DS command). This is the same command as BT 1 since the microcode is not loadable.
LC		Load CRAM Address. Not implemented.
LF		Load Diagnostic Write Function. Not implemented.

<table>
<tr><th colspan="3">Table 7 – KS10 Console Command Summary</th></tr>
<tr><th>Command</th><th>Argument</th><th>Description</th></tr>
<tr><td rowspan="19">LI</td><td rowspan="19">xx</td><td>Load IO Address.
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><table>
<tr><th>IO Address</th><th>Description</th></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>
</table></td></tr>
<tr><td>LK</td><td></td><td>Load 8080 address
Not implemented.
Set 8080 memory address xx.</td></tr>
<tr><td>LR</td><td></td><td>Load 8080 register
Not implemented.
Set 8080 register address xx.</td></tr>
<tr><td>LT</td><td></td><td>Lamp Test
Not implemented.
Blink the indicator lights.</td></tr>
</table>

Table 7 – KS10 Console Command Summary		
Command	**Argument**	**Description**
MB		Magtape Boot Not implemented. Load the monitor boot program from the tape selected last
MK	xx	Mark microcode word Not implemented Mark microcode word at CRAM address xx.
MM		Not implemented.
MR		Master Reset. Reset the KS10.
MS		Not implemented.
MT		Not implemented.
MT	1	Not implemented.
PE		Not implemented.
PM		Not implemented.
PW		Not implemented.
RC		Not implemented.
RP		Repeat Not implemented. Repeat last command or last command string until any CTY key is depressed.
RP	xx	Repeat Not implemented. Repeat last command, or last command string, xx times.
SC		Not implemented.
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		Not implemented.
ST	xx	Start KS10 program at address xx. Console program enters user mode.
TE	{1 | 0}	Enable/Disable KS10 CPU Interval Timer
TP	{1 | 0}	Enable/Disable KS10 CPU traps.

Table 7 – KS10 Console Command Summary		
Command	**Argument**	**Description**
TT		Not implemented. Force KLINIK line from mode 3 to mode 2.
UM	xx	Unmark microcode word Not implemented. Unmark microcode word at CRAM address xx.
VD		Verify CRAM against disk. Not implemented.
VT		Verify CRAM against tape. Not implemented.
ZM		Zero Memory Deposit 0 into all KS10 memory locations.

4.3 KS10 Console Software Operation

This section describes some of components of the console software.

4.3.1 Field Programmable Gate Array (FPGA) Driver

The FPGA that has been selected for this implementation is RAM-based and is therefore volatile. When power is applied, the FPGA is un-programmed and essentially 'brain dead'. The FPGA firmware is stored in a serial Flash memory that is attached to the FPGA. The Console microcontroller initiates the FPGA programming operation and monitors the programming sequence to ensure that it completes successfully. Once programmed, the driver also provides limited built-in-test (BIT) reporting capabilities back to the console software.

Once the FPGA has been loaded with firmware and the FPGA is operating like a KS10, the operation of the FPGA is managed by the KS10 Driver.

4.3.2 Universal Asynchronous Receiver/Transmitter (UART) Driver

The UART Driver is implemented in the Microcontroller's Read Only Memory (ROM). The ROM provides an Application Programming Interface (API) to the UART hardware.

4.3.3 Synchronous Serial Interface (SSI) Driver

The SSI Driver is implemented in the Microcontroller's Read Only Memory (ROM). The ROM provides an Application Programming Interface (API) to the SSI hardware.

4.3.4 External Peripheral Interface (EPI) Driver

The EPI Driver is implemented in the Microcontroller's Read Only Memory (ROM). The ROM provides an Application Programming Interface (API) to the EPI hardware. The EPI is configured by the driver to implement a 8-bit multiplexed address and data bus. See the block diagram in Figure 25.

4.3.5 General-Purpose Input/Outputs (GPIOs) Driver

The GPIO Driver is implemented in the Microcontroller's Read Only Memory (ROM). The ROM provides an Application Programming Interface (API) to the GPIO hardware.

4.3.6 KS10 Driver

The KS10 driver provides a sane abstraction of the KS10 FPGA interfaces. The interface between the Console Microcontroller and the KS10 FPGA is described in Section 4 of this document.

The KS10 FPGA is controlled by a set of memory-mapped registers inside the FPGA. The KS10 Driver uses the EPI Driver to access the memory-mapped registers inside the KS10 FPGA.

The KS10 Driver also provides accessor functions to the various bits in the Console Control/Status Register.

4.3.7 Secure Digital High-Capacity (SDHC) Card Driver

Although the console software attempts to be compatible all types of SD Card, it is really only tested with modern cards that are compatible with the SDHC specification. To keep the driver simple, the driver operates in Serial Peripheral Interface (SPI) mode which is supported by the MCU's Synchronous Serial Interface (SSI). The SD Card interface which uses the SPI protocol is described by the document https://www.sdcard.org/downloads/pls/simplified_specs/part1_410.pdf in chapter 7.

4.3.8 FAT32 Filesystem

The FAT Filesytem is a port of the FatFS - Generic FAT File System available from http://elm-chan.org/fsw/ff/00index_e.html. This particular implementation of the FatFS has been configured to be read-only and does not support long filenames.

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.

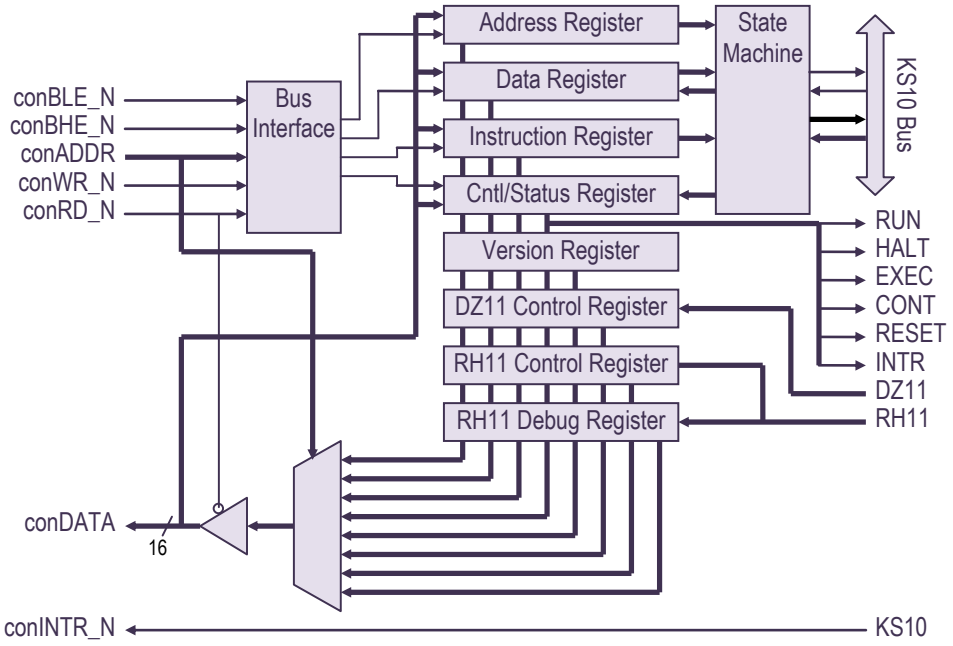


Figure 24 – KS10 Console Interface Block Diagram

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 25.

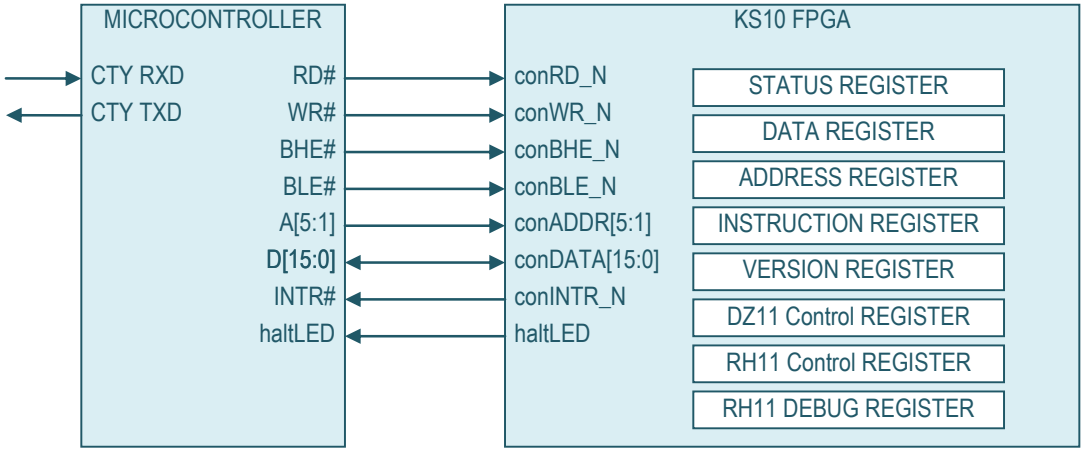


Figure 25 – Console Microcontroller and KS10 FPGA Interface

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 26.

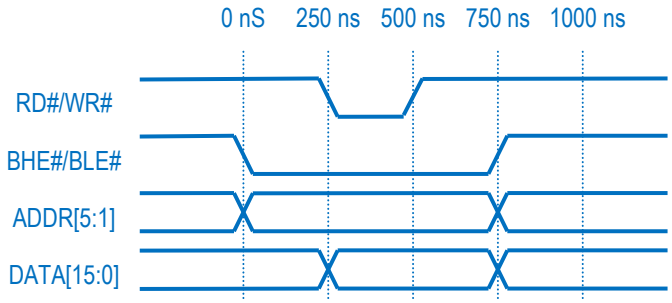


Figure 26 – Console Microcontroller Interface Read/Write Cycle Timing Diagram

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 8.

Table 8 – 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 Control/Status Register bits 28-35
0x11	Console Control/Status Register bits 20-27
0x12	Console Control/Status Register bits 12-19
0x13	Console Control/Status Register bits 4-11
0x14	Console Control/Status Register bits 0-3
0x15	Reserved
0x16	Reserved
0x17	Reserved
0x18	Console Instruction Register bits 28-35
0x19	Console Instruction Register bits 20-27
0x1a	Console Instruction Register bits 12-19
0x1b	Console Instruction Register bits 4-11
0x1c	Console Instruction Register bits 0-3
0x1d	Reserved
0x1e	Reserved
0x1f	Reserved
0x20	DZ11 Control Register bits 56-63
0x21	DZ11 Control Register bits 48-55
0x22	DZ11 Control Register bits 40-47
0x23	DZ11 Control Register bits 32-39
0x24	DZ11 Control Register bits 24-31
0x25	DZ11 Control Register bits 16-23
0x26	DZ11 Control Register bits 8-15
0x27	DZ11 Control Register bits 0-7
0x28	RH11 Control Register bits 56-63
0x29	RH11 Control Register bits 48-55
0x2a	RH11 Control Register bits 40-47
0x2b	RH11 Control Register bits 32-39
0x2c	RH11 Control Register bits 24-31
0x2d	RH11 Control Register bits 16-23
0x2e	RH11 Control Register bits 8-15
0x2f	RH11 Control Register bits 0-7
0x30	RH11 Debug Register bits 56-63
0x31	RH11 Debug Register bits 48-55
0x32	RH11 Debug Register bits 40-47

0x33	RH11 Debug Register bits 32-39
0x34	RH11 Debug Register bits 24-31
0x35	RH11 Debug Register bits 16-23
0x36	RH11 Debug Register bits 8-15
0x37	RH11 Debug Register bits 0-7
0x38	Debug Control/Status Register bits 28-35
0x39	Debug Control/Status Register bits 20-27
0x3a	Debug Control/Status Register bits 12-19
0x3b	Debug Control/Status Register bits 4-11
0x3c	Debug Control/Status Register bits 0-3
0x3d	Reserved
0x3e	Reserved
0x3f	Reserved
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	Instruction Trace Register bits 8-15
0x57	Instruction Trace 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.

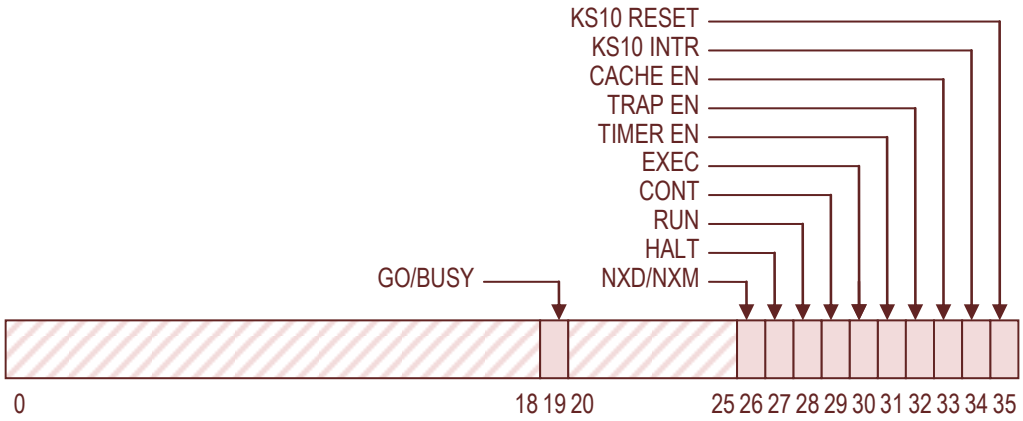


Figure 27 – Console Control/Status Register

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

Table 9 – Console Control/Status Register Definitions				
Bit	**Mnemonic**	**R/W**	**Init**	**Description**
0-18	Reserved	R/W	-	Reserved. Writes ignored. Always read as zero.

Table 9 – Console Control/Status Register Definitions				
Bit	**Mnemonic**	**R/W**	**Init**	**Description**
19	GO/BUSY	R/W	0	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. The Console Address Register and Console Data Register should not be modified when this bit is asserted.
20-25	Reserved	R	-	Reserved. Writes ignored. Always read as zero.
26	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.
27	HALT	R	-	HALT Status. Writes ignored. Returns HALT status.
28	RUN	R/W	0	KS10 Run. See Section 5.2.1 for a description of the operation of this bit. Note: The RUN bit is automatically cleared by the KS10 microcode when a HALT condition occurs.
29	CONT	R/W	0	KS10 Continue. See Section 5.2.2 for a description of the operation of this bit. Note: The CONT bit is automatically cleared by the KS10 microcode once it is sampled.
30	EXEC	R/W	0	KS10 Execute. See Section 5.2.3 for a description of the operation of this bit. Note: The EXEC bit is automatically cleared by the KS10 microcode once it is sampled.
31	TIMER EN	R/W	0	This bit enables the KS10 one millisecond interval timer.
32	TRAP EN	R/W	0	This bit enables KS10 traps.
33	CACHE EN	R/W	0	This bit enables the KS10 cache.
34	KS10_INTR	R/W	0	This bit generates an interrupt from the Console to the KS10. This signal only needs to be asserted for a single clock cycle in order to generate a KS10 interrupt. Always read as zero.

Table 9 – Console Control/Status Register Definitions				
Bit	**Mnemonic**	**R/W**	**Init**	**Description**
35	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. The console microcontroller must initialize the Console Instruction Register and the Console CTY interface (at least) before negating the KS10 RESET signal. Note: 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.

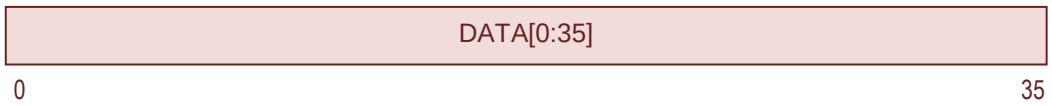


Figure 28 – Console Data Register

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 29.

Figure 29 – 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.

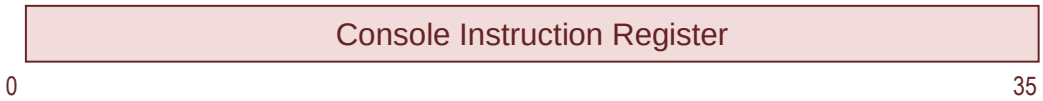


Figure 30 – Console Instruction Register

5.1.8 DZ11 Console Control Register

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.

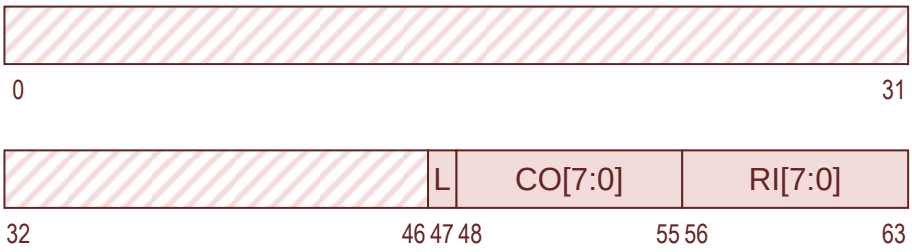


Figure 31 – DZ11 Console Control Register

Table 10 – DZ11 Console Control Register Definition			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0-46	Reserved	R/W	Reserved Unused bits are read/write
47	LOOP	R/W	Loopback. This loops TXD and RXD outside of the DZ11.
48-55	CO[7:0]	R/W	Carrier Sense These bits are reflected in the DZ11 Modem Status Register bits 15 through 8.
56-63	RI[7:0]	R/W	Ring Indication These bits are reflected in the DZ11 Modem Status Register bits 7 through 0.

5.1.9 RH11 Console Control Register

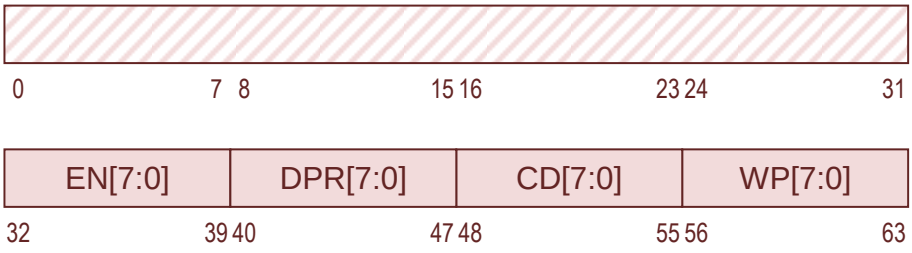


Figure 32 – RH11 Console Control Register

Table 11 – RH11 Console Control Register Definition			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0-46	Reserved	R/W	Reserved Unused bits are read/write
32-39	EN[7:0]	R/W	Drive Enabled
40-47	DPR[7:0]	R/W	Drive Present
48-55	CD[7:0]	R/W	Card Detect
56-63	WP[7:0]	R/W	Write Protect

5.1.10 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.

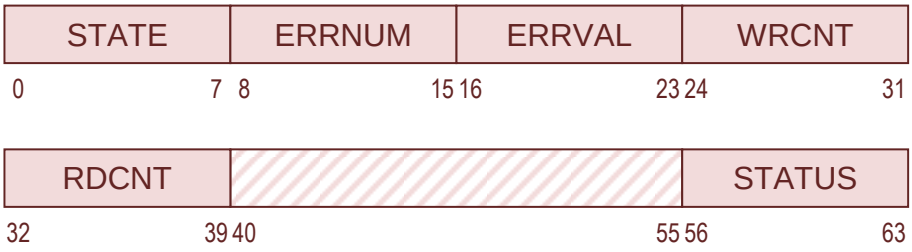


Figure 33 – RH11 Debug Register

Table 12 – Console Control/Status 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
24-31	WRCNT	R	Number of Disk Writes.
32-39	RDCNT	R	Number of Disk Reads.
40-55	Reserved	R	Reserved. Always read as zero.
56	DISK0	R	
57	DISK1	R	
58	DISK2	R	
59	DISK3	R	

60	DISK4	R	
61	DISK5	R	
62	DISK6	R	
63	DISK7	R	

5.1.11 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.

5.1.11.1 Debug Control/Status Register (CSR)

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

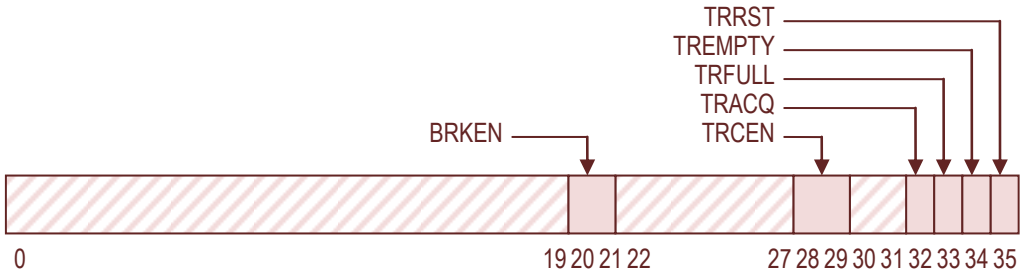


Figure 34 – Debug Control/Status Register (DCSR)

Table 13 – Debug Control/Status Register (DCSR) Definitions				
Bit	**Mnemonic**	**R/W**	**Init**	**Description**
0-19	Reserved	R	-	Reserved Writes ignored. Always read as zero.
20-21	BRKEN	R/W	0	Breakpoint Enable 00 – Breakpoints disabled 01 – Breakpoint on Address Match 10 – Breakpoint on Trace Buffer full. 11 – Reserved
22-27	-	R	0	Reserved Writes ignored. Always read as zero.

Table 13 – Debug Control/Status Register (DCSR) Definitions				
Bit	**Mnemonic**	**R/W**	**Init**	**Description**
28-29	TREN	R/W	0	Trace Enable 00 – Trace disabled 01 – Trace on Address Match 10 – Trace Enabled 11 – Reserved
30-31	-	R	0	Reserved Writes ignored. Always read as zero.
32	TRACQ	R	0	Trace Acquiring Data Writes ignored. This signal is asserted when the Trace Buffer is acquiring data. This signal cleared when the Trace Buffer is reset (DCSR[RESET] asserted).
33	TRFULL	R	0	Trace Buffer Full Writes ignored. This signal is asserted when the Trace Buffer is full.
34	TREMPTY	R	1	Trace Buffer Empty Writes ignored. This signal is asserted when the Trace Buffer is empty.
35	TRRESET	W	0	Trace Buffer Reset Asserting this bit will reset the Trace Buffer state. The EMPTY bit will be asserted and the FULL bit will be negated. Always read as zero.

5.1.11.2 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 35 – Debug Breakpoint Address Register (DBAR)

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

5.1.11.3 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 36 – Breakpoint Mask Register (DBMR)

<table>
<tr><th colspan="4">Table 15 – Breakpoint Mask Register 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.</td></tr>
<tr><td>
<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>

5.1.11.4 Debug Instruction Trace Register (ITR)

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.

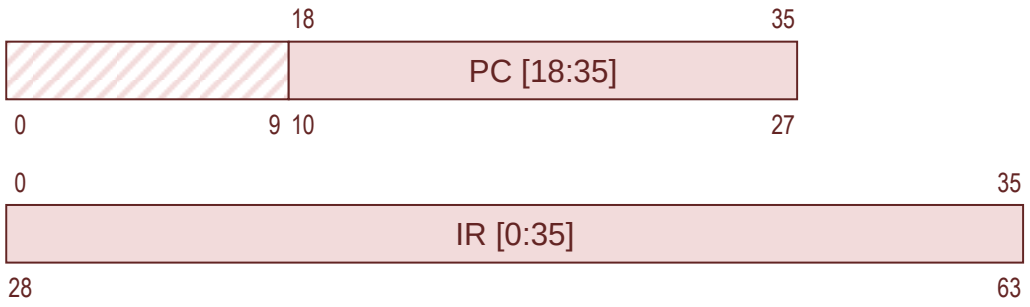


Figure 37 – Instruction Trace Register

Table 16 – 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.

5.1.12 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.

Table 17 – 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	0xAE	‘.’	R	Always ‘.’ + 0x80
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 0xAE (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 38 below.

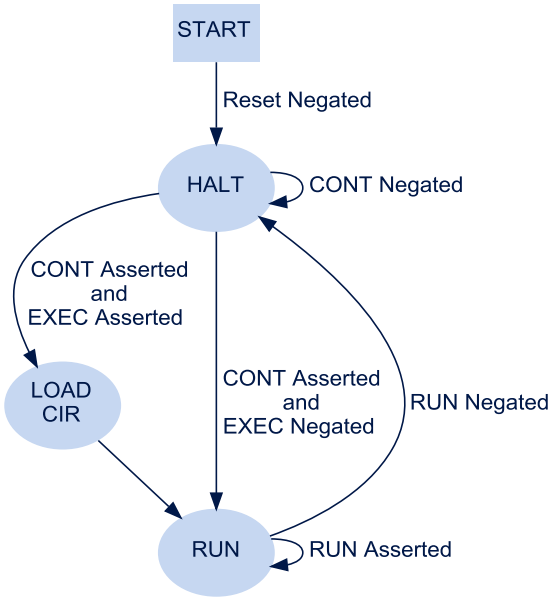


Figure 38 – KS10 Control State Diagram

When in the *HALT State*, Table 18 enumerates the types of operations that the KS10 will perform when the control bits are set in the various states.

Table 18 – 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 also 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 a sequence of 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 9.

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 29.

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 19.

Table 19 – 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

Table 19 – KS10/Console Communications Area		
Address	**Summary**	**Detailed Description**
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 20 – KS10 Halt Switch Word (KS10 Memory Address 000030)		
Bit(s)	**Mnemonic**	**Description**

5.4.2 Keep Alive

Table 21 – KS10 Keep Alive 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	Manufacturing mode.
14-19	-	Reserved
20-27	KPALIV	Keep alive word.

Table 21 – KS10 Keep Alive Word (KS10 Memory Address 000031)		
Bit(s)	**Mnemonic**	**Description**
28-31	-	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.

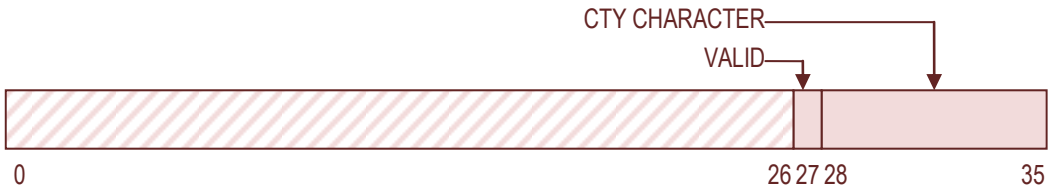


Figure 39 – KS10 CTY Input Word (KS10 Memory Address 000032)

Table 22 – 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 9.

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.

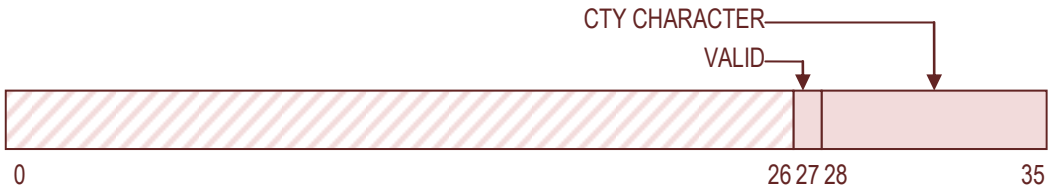


Figure 40 – KS10 CTY Output Word (KS10 Memory Address 000033)

Table 23 – 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 9.

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 24 – 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 25 – 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 26 – 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 27 – 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 28 – 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.

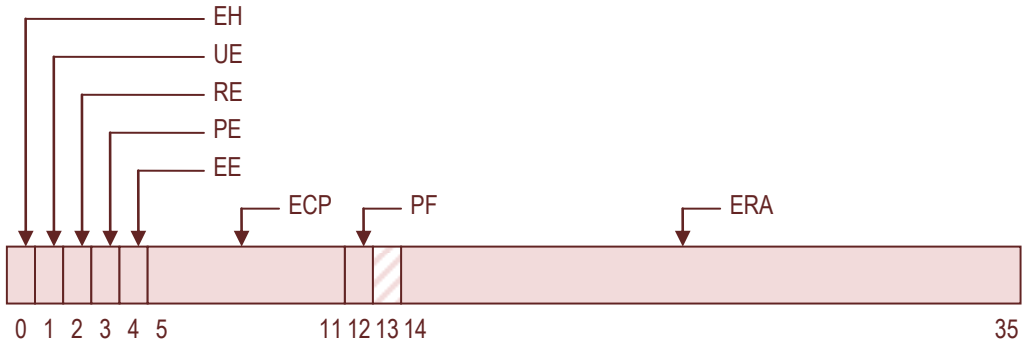


Figure 41 – Memory Status Register (Read)

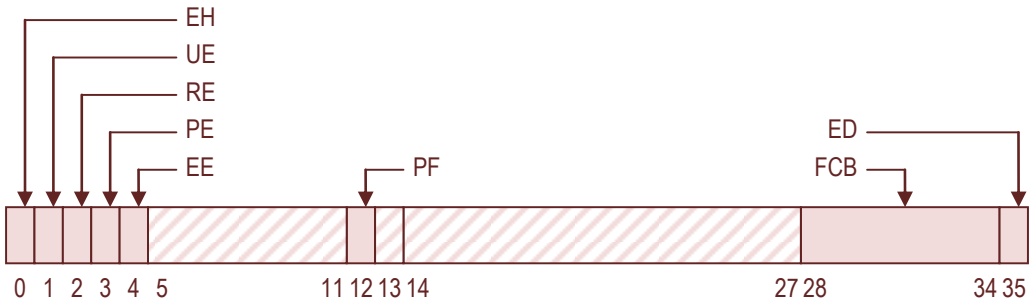


Figure 42 – Memory Status Register (Write)

Table 29 – Memory Status Register Definitions			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0	EH	R	Error Hold. Not implemented. Always read as 0. Writes are ignored.
1	UE	R	Uncorrectable Read Error aka BAD DATA. Not implemented. Always read as 0. Writes are ignored.

Table 29 – Memory Status Register Definitions			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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

For now, 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 43.

The timing diagram of a write cycle is illustrated below in Figure 44.

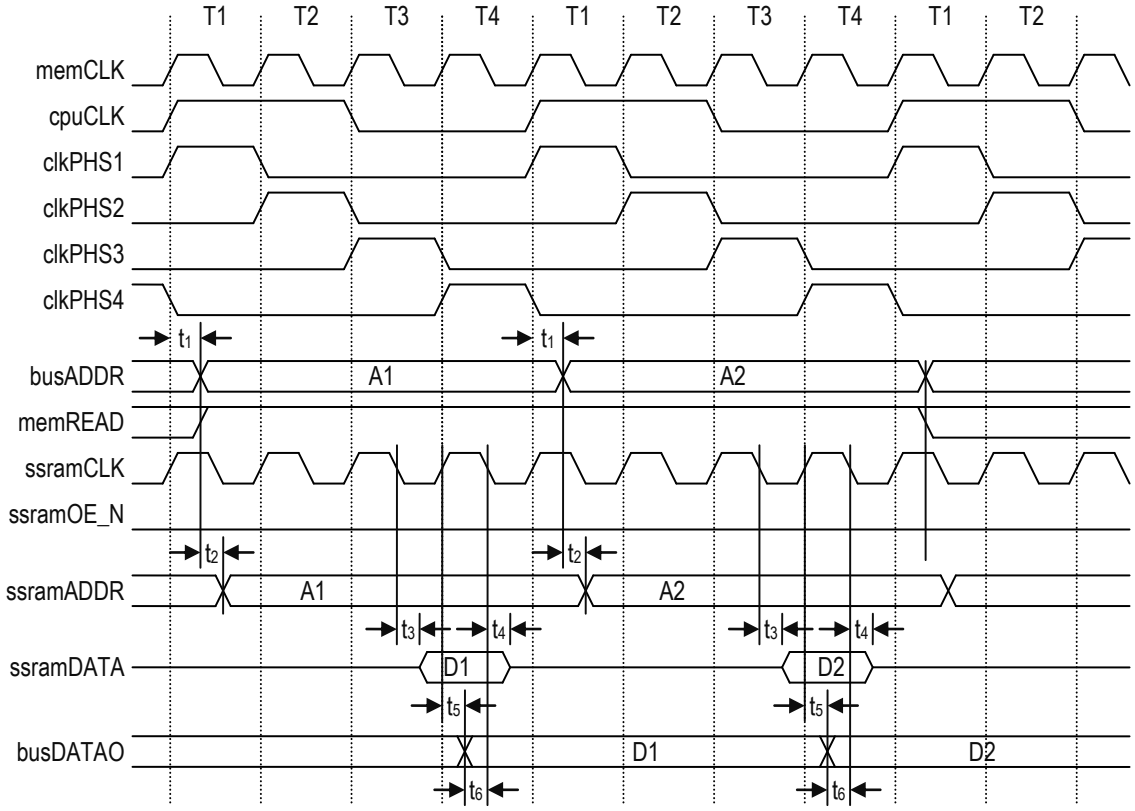


Figure 43 - SSRAM Read Timing Diagram

Table 30 – 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

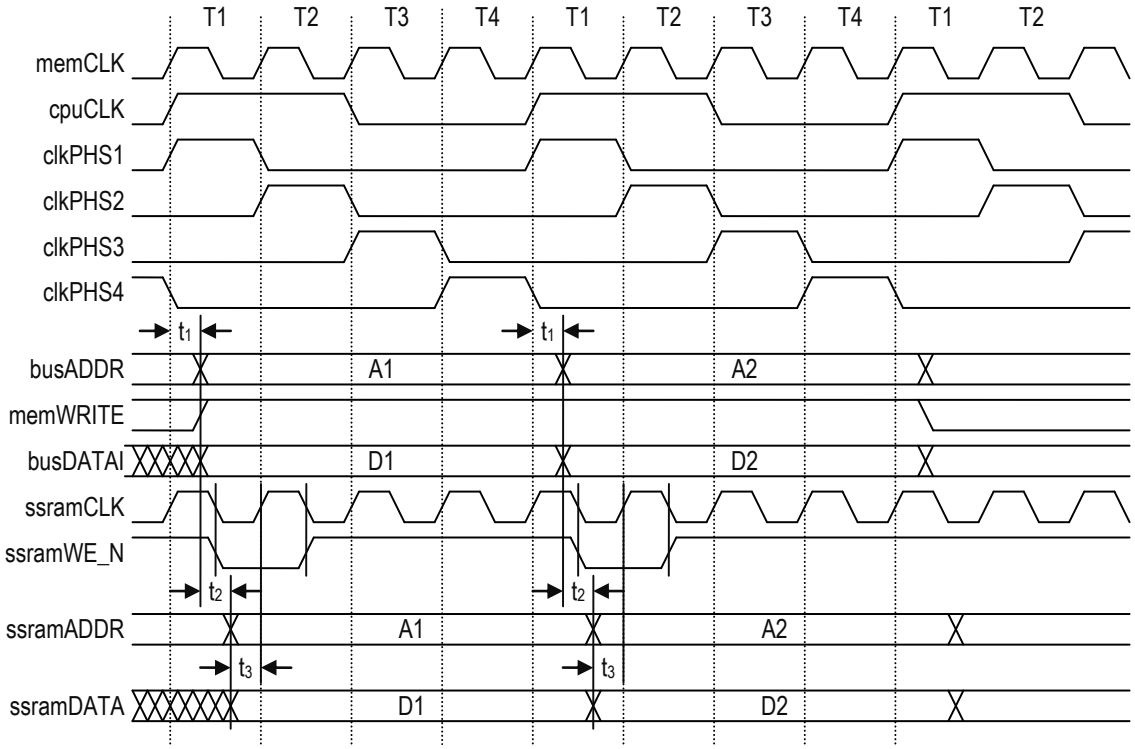


Figure 44 - SSRAM Write Timing Diagram

Table 31 – 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 Bridge (was Unibus) Implementation

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] The DECSYSTEM 2020 UNIBUS ADAPTER EXERCISER (DSUBA) will test UBA1, UBA3, and UBA4.

The KS10 FPGA implements register-compatible IO Bridges but does not attempt to implement the Unibus hardware or protocol inside the FPGA.

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

7.1 IO Bus to KS10 Byte and Word Translation

The UBA can provide translation between the 36-bit KS10 bus and 16-bit and 8-bit peripherals. This translation is illustrated below in Figure 45.

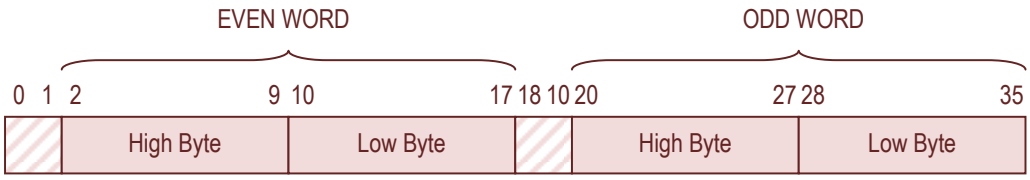


Figure 45 – IO Bus Byte and Word Translation

Table 32 – UBA Address Translation

UBA A1	UBA A0	OP	UBA Data Description
0	0	Byte	Even word, low byte
0	1	Byte	Even word, high byte
1	0	Byte	Odd word, low byte
1	1	Byte	Odd word, high byte
0	0	Word	Even word
1	0	Word	Odd word

[1] Personal correspondence with Timothe Litt.

7.2 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 46. The format during a read operation is detailed in Figure 47.

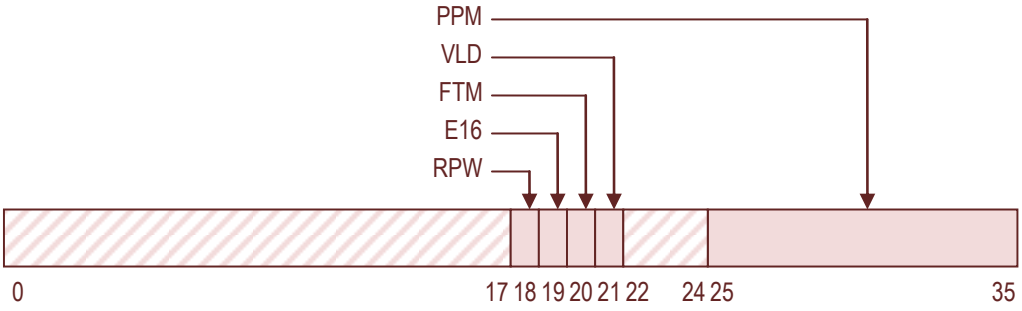


Figure 46 – IO Bridge Paging RAM Write

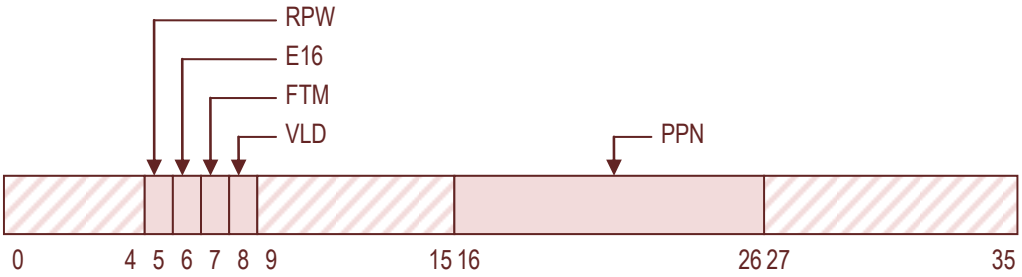


Figure 47 – IO Bridge Paging RAM Read

Table 33 – 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.

Table 33 – IO Bridge Paging RAM Definitions			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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.

7.3 IO Bus Bridge Paging

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.

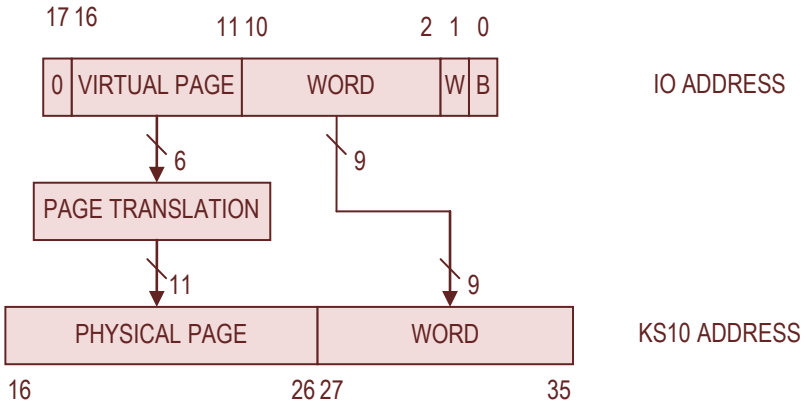


Figure 48 – IO Bus Paging

7.4 IO 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.

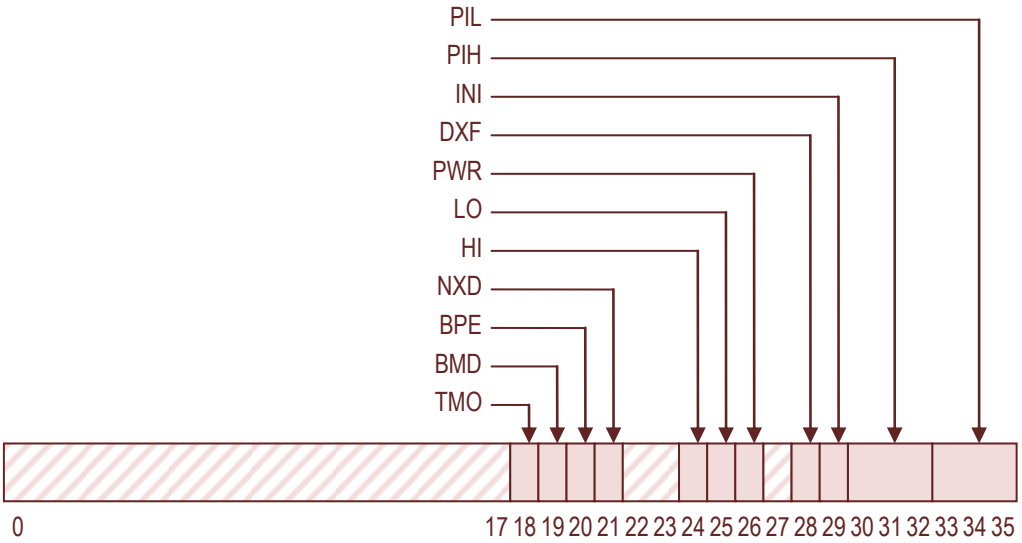


Figure 49 – IO Bridge Control Status Register (UBACSR)

Table 34 – 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.
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.

Table 34 – IO Bridge Control Status Register (UBACSR) Definitions			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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	Hi priority PIA. Cleared by writing a one to INI.
33-35	PIL	R/W	Lo priority PIA. Cleared by writing a one to INI.

7.5 IO 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 50 – IO Bridge Maintenance Register (UBAMR)

Table 35 – 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.

7.6 Bridged IO Devices

In an actual KS10 implementation, all of the normal IO is implemented by Unibus Devices. The KS10 implements an IO bridge that bridges the KS10 backplane bus to Unibus. In the KS10 FPGA implementation, this same architecture is replicated – except that the IO bus is significantly different than Unibus.

For now only two Unibus-compatible devices are implemented: the DZ11 Terminal Multiplexer and the RH11 Massbus Disk Controller.

Refer to the block diagram in Figure 20 for additional information about this bus structure.

These devices are described in the following sections.

8 DZ11 Register Compatible 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 36.

Table 36 – DZ11 Configuration

Device	UBA	Interrupt	Interrupt Vector	Base Address
DZ11 #1	UBA3	5	000340	760010
DZ11 #2	UBA3	5	000350	760020
DZ11 #3	UBA3	5	000360	760030
DZ11 #4	UBA3	5	000370	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 51.

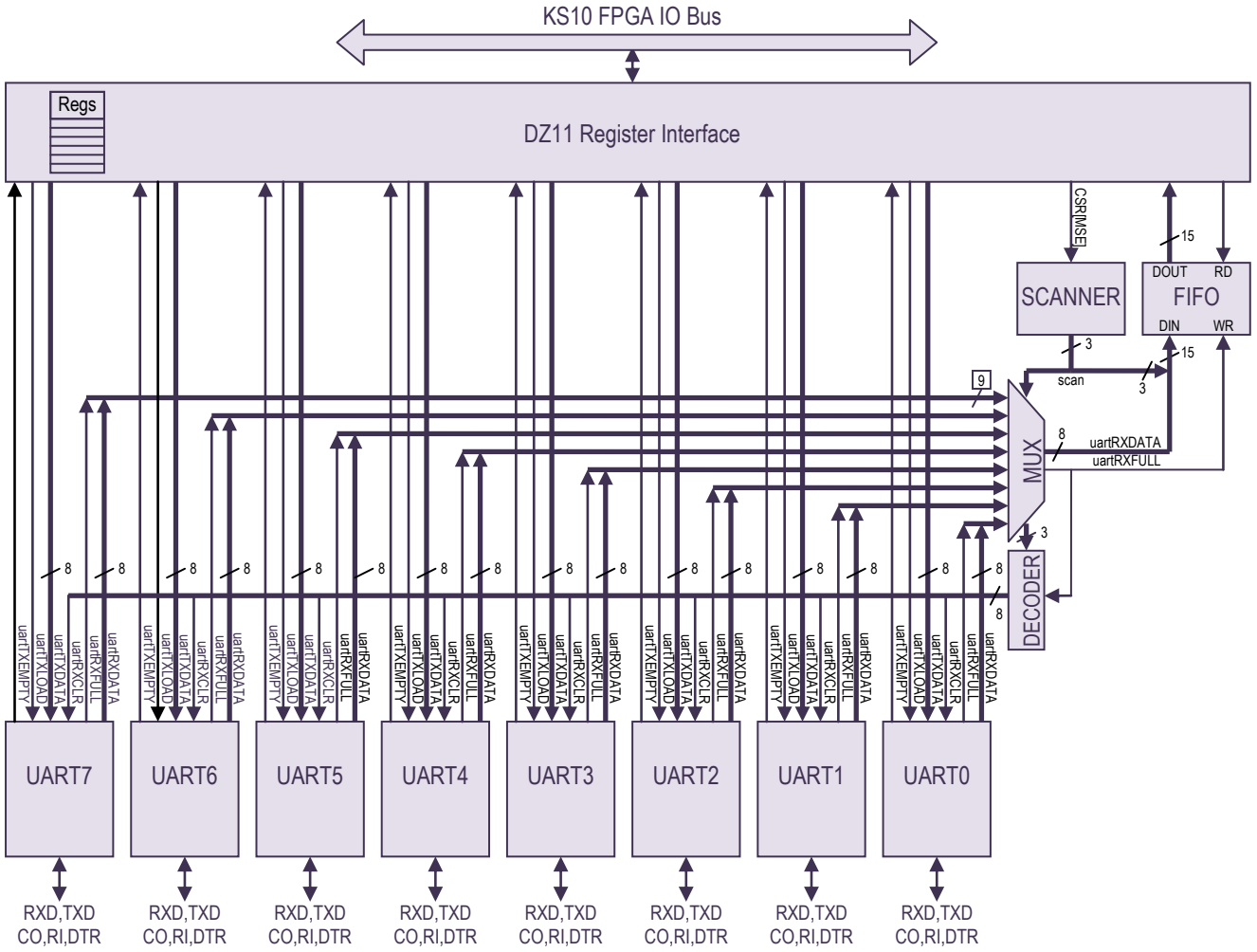


Figure 51 – DZ11 Block Diagram

8.1 DZ11 Control and Status Register (CSR)

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

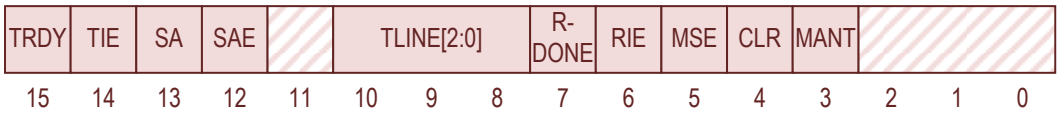


Figure 52 – DZ11 Control and Status Register (CSR)

Table 37 – 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. When this bit is asserted and CSR[TIE] is asserted, a transmitter interrupt is generated. This bit is cleared by CSR[CLR]), devRESET (from the IO Bus Bridge), or by loading the transmitter buffer. It is also cleared by negating the line (TCR[LIN]) associated with CSR[TLINE].
14	TIE	R/W	Transmitter Interrupt Enable. Asserting this bit enables transmitter interrupts. This bit is cleared by CSR[CLR] and by devRESET (from the IO Bus Bridge).
13	SA	R	Silo Alarm. This bit is asserted when at least 16 characters have been written into the receive FIFO. This signal (CSR[SA]) will generate a receiver interrupt when CSR[RIE] is asserted. This bit is cleared by asserting CSR[CLR], negating CSR[SAE], asserting devRESET (from the IO Bus Bridge), or by reading RBUF. 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.
12	SAE	R/W	Silo Alarm Enable. Asserting this bit enables the SILO Alarm (SA) interrupt. This bit is cleared by negating CSR[SAE], asserting CSR[CLR], or by asserting devRESET (from the IO Bus Bridge).
11	Unused	R	Read as zero.
10-8	TLINE[2:0]	R	Transmit Line. When CSR[TRDY] is asserted, these bits indicate which of the transmitter can accept a character to transmit.
7	RDONE	R	Receiver Done. This bit is asserted if the FIFO is not empty. This signal (CSR[RDONE]) will generate a receiver interrupt when CSR[SAE] is negated and CSR[RIE] is asserted.
6	RIE	R/W	Receiver Interrupt Enable. Asserting this bit enables receiver interrupts. This bit is cleared by CSR[CLR] and by devRESET (from the IO Bus Bridge).

Table 37 – DZ11 Control and Status Register (CSR) – IO Address 760010			
Bit(s)	**Mnemonic**	**R/W**	**Description**
5	MSE	R/W	Master Scan Enable. This bit enables the receiver, transmitter, and FIFO. This bit is cleared by CSR[CLR] and by devRESET (from the IO Bus Bridge).
4	CLR	R/W	Clear. When asserted, this bit clears the Receiver FIFO, all UARTS, and the CSR. This triggers a 15µS one-shot in the KS10.
3	MAINT	R/W	Maintenance Mode. When this bit is asserted, the transmitted serial data is looped back to the receiver. This bit is cleared by CSR[CLR] and by devRESET (from the IO Bus Bridge).
2-0	Unused	R	Read as zero.

8.2 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.

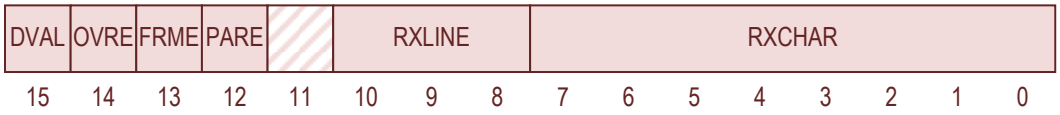


Figure 53 – DZ11 Receiver Buffer Register (RBUF)

Table 38 – 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. Always read as zero.
12	PARE	R	Parity Error. Always read as zero.
11	Unused	R	Always read as zero.
10-8	RXLINE	R	Received line. This field indicates the line number of the received character.

Table 38 – DZ11 Receiver Buffer Register (RBUF) – IO Address 760012			
Bit(s)	**Mnemonic**	**R/W**	**Description**
7-0	RXCHAR	R	Received character.

8.3 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.

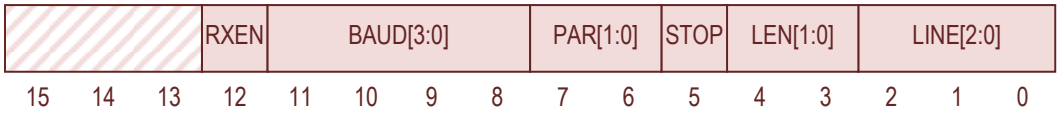


Figure 54 – DZ11 Line Parameter Register (LPR)

Table 39 – 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. When asserted, this enables the UART receiver clock selected by LPR[LINE]. The RXEM bits are cleared by CSR[CLR] and by devRESET (from the IO Bus Bridge).

Table 39 – 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.	

8.4 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.

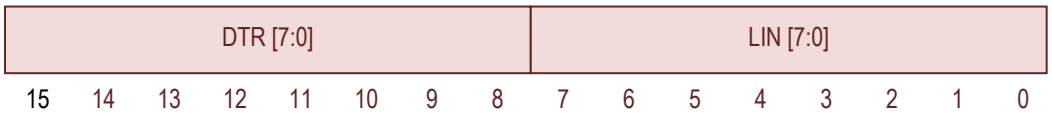


Figure 55 – DZ11 Transmit Control Register TCR)

Table 40 – 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.

8.5 DZ11 Modem Status Register (MSR)

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

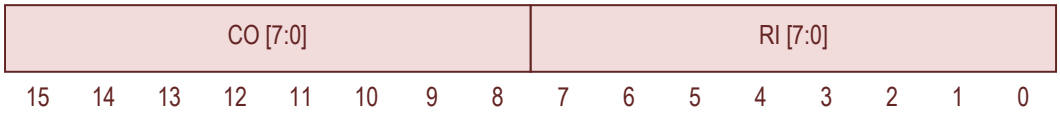


Table 41 – DZ11 Modem Status Register (TDR) – IO Address 760016			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-8	CO[7:0]	R	Carrier Detect. These bits reflect the state of the 8 Carrier Detect (CO) signals. Note: these signals are controlled by the DZ11 Console Control Register but are otherwise unused.
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.

8.6 DZ11 Transmit Data Register (TDR)

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

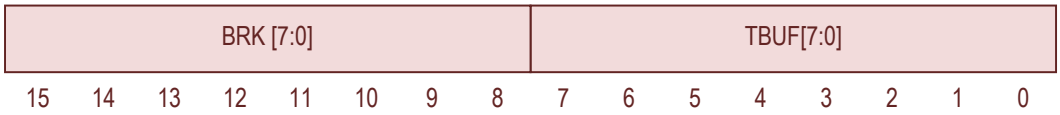


Table 42 – DZ11 Transmit Data Register (TDR) – IO Address 760016			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-8	BRK[7:0]	W	Break. Not implemented. Writes have no effect.
7-0	TBUF[7:0]	W	Transmitter Buffer. Data that is written to the port is written to the UART transmitter buffer that is indicated by CSR[TLINE] register bits.

8.7 Hardware Description

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

8.7.1 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.

8.7.2 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.

8.7.2.1The 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 56.

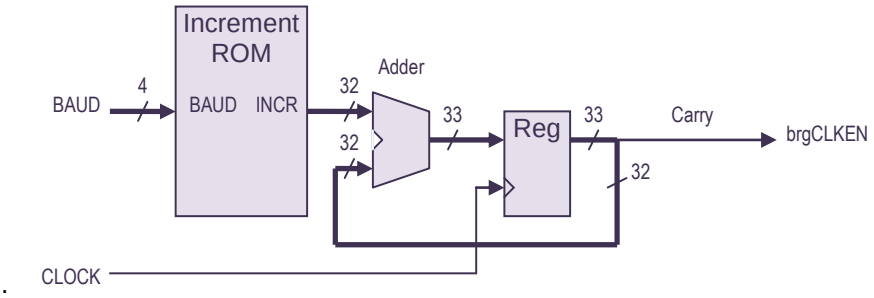


Figure 56 – Fractional-N Divider Block Diagram

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

9 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 43.

Table 43 – 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 57.

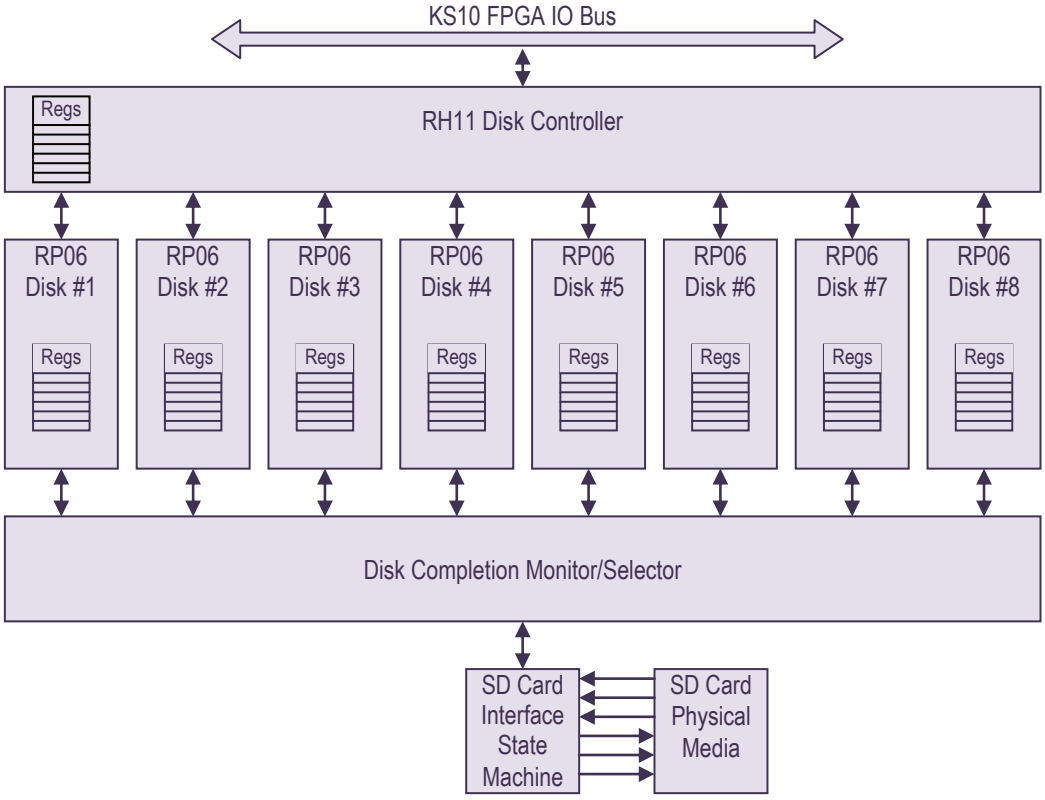


Figure 57 – KS10 FPGA Disk Subsystem Architecture

The subsections are described in the following document sections.

9.1 Definitions

9.1.1 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.

9.1.1.1IO Bridge Clear

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

9.1.1.2Controller Clear

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

9.1.1.3Drive Clear

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

9.1.1.4Error 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]).

9.2 RH11 Compatible Disk Controller

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 44 – 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

9.2.1 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.

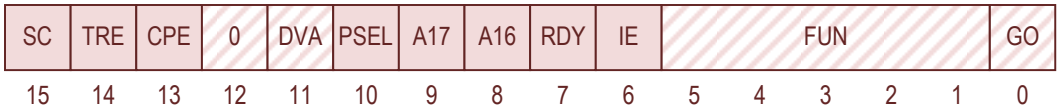


Figure 58 – RH11 Control and Status Register #1 (RHCS1)

Table 45 – 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.

Table 45 – RH11 Control and Status Register #1 (RHCS1) – IO Address 776700			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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*.
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.

9.2.2 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 word that is transferred increments the register. The transfer is complete when the Word Count register is zero.

The disk always reads and writes full sectors. On writes, partial sectors are filled with zero.

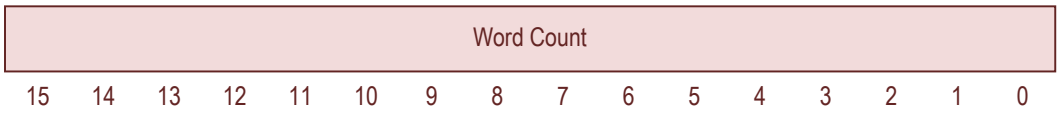


Figure 59 – RH11 Word Count Register (RHWC)

Table 46 – 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.

9.2.3 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 48. 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 60 – RH11 Bus Address Register (RPBA)

Table 47 – 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).

Table 47 – RH11 Bus Address Register (RHBA) – IO Address 776704			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0	BA	R	Bus Address (Bit 0). Writes ignored. Always read as zero.

9.2.4 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.

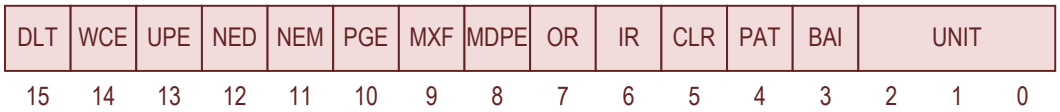


Figure 61 – RH11 Control and Status Register #2 (RHCS2)

Table 48 – 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. Never set. All 8 drives are always implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Error Clear*.

Table 48 – RH11 Control and Status Register #2 (RHCS2) – IO Address 776710			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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*.
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*.

9.2.5 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.

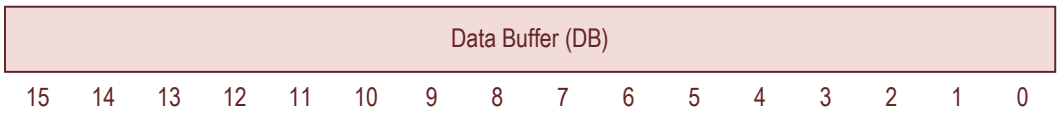


Figure 62 – RH11 Data Buffer Register (RPDB)

Table 49 – 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.

9.3 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 50 – 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

Table 50 – RPxx Device Registers				
Unibus Address	**RP Register Name**	**RM Register Name**	**R/W**	**Register Description**
		RMHR	R/W	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 51. This is provided for reference only. In this implementation, the MASSBUS registers are decoded directly from the IO Bus address.

Table 51 – Massbus Register Address Cross Reference				
Reg # (decimal)	**Reg # (octal)**	**RP Register Name**	**RM Register Name**	**Compatibility Comment**
0	0	RPCS1	RMCS1	

Table 51 – Massbus Register Address Cross Reference				
Reg # (decimal)	**Reg # (octal)**	**RP Register Name**	**RM Register Name**	**Compatibility Comment**
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.
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	

9.3.1 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.

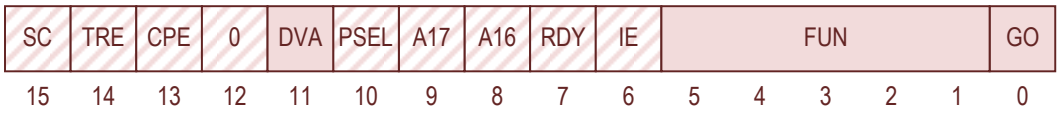


Figure 63 – RP Control and Status Register #1 (RPCS1)

<table>
<tr><th colspan="4">Table 52 – 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>Description</th></tr>
<tr><td>15</td><td>SC</td><td>R</td><td>See RH controller.</td></tr>
<tr><td>14</td><td>TRE</td><td>R/W</td><td>See RH controller.</td></tr>
<tr><td>13</td><td>CPE</td><td>R</td><td>See RH controller.</td></tr>
<tr><td>12</td><td>0</td><td>R</td><td>See RH controller.</td></tr>
<tr><td>11</td><td>DVA</td><td>R</td><td>Drive available. Always read as 1</td></tr>
<tr><td>10</td><td>PSEL</td><td>R/W</td><td>See RH controller.</td></tr>
<tr><td>9</td><td>A17</td><td>R/W</td><td>See RH controller.</td></tr>
<tr><td>8</td><td>A16</td><td>R/W</td><td>See RH controller.</td></tr>
<tr><td>7</td><td>RDY</td><td>R</td><td>See RH controller.</td></tr>
<tr><td>6</td><td>IE</td><td>R/W</td><td>See RH controller.</td></tr>
<tr><td rowspan="19">5-1</td><td rowspan="19">FUN</td><td rowspan="19">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>
</table>

Table 52 – RP Control and Status Register #1 (RPCS1) – IO Address 776700				
Bit(s)	**Mnemonic**	**R/W**	**Description**	
			32-33	Illegal function(s)
			34	Read data
			35	Read header and data
			36-37	Illegal function(s)
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???	

9.3.2 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.

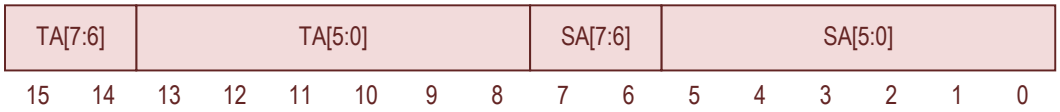


Figure 64 – RP Disk Address Register (RPDA)

Table 53 – RP Disk Address Register (RPDA) – IO Address 776706			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15-14	TA[7:6]	R	Zero. (RPxx drive type) Writes ignored. Read as zero.
		R/W	Track Address upper bits. (RMxx drive type) Incremented with TA[5:0] below. Cleared by Read-in Preset command. This register is **NOT** reset by either the *IO Bridge Clear or Controller Clear*.
13-8	TA[5:0]	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 69 for the highest numbered track. Cleared by Read-in Preset command. This register is **NOT** reset by either the *IO Bridge Clear or Controller Clear*.

Table 53 – RP Disk Address Register (RPDA) – IO Address 776706			
Bit(s)	**Mnemonic**	**R/W**	**Description**
7-6	SA[7:6]	R	Zero. (RPxx drive type) Writes ignored. Read as zero.
		R/W	Sector Address upper bits. (RMxx drive type) Incremented with SA[5:0] below. Cleared by Read-in Preset command. This register is **NOT** reset by either the *IO Bridge Clear or Controller Clear*.
5-0	SA[5:0]	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 69 for the highest numbered sector. Cleared by Read-in Preset command. This register is **NOT** reset by either the *IO Bridge Clear or Controller Clear*.

9.3.3 RP Drive Status (RPDS) Register

This register reports the status of the disk drive.

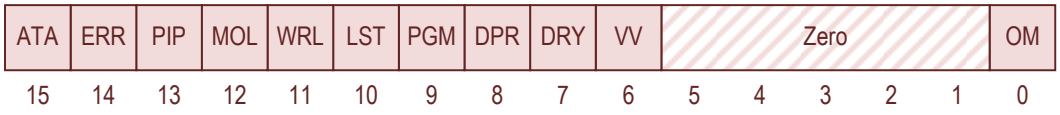


Figure 65 – RP Drive Status Register (RPDS)

Table 54 – 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.
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.

Table 54 – RP Drive Status Register (RPDS) – IO Address 776712			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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.
6	VV	R	Volume Valid. Set by Pack Acknowledge command or the Read-in Preset command. Cleared by inserting the SD Card into the SD Reader (i.e., cleared when the SD Card transitions from not present to present). This is **NOT** cleared by Cleared by IO Bridge Clear, Controller Clear, or Drive Clear command. Note: 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.
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

Table 54 – RP Drive Status Register (RPDS) – IO Address 776712			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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.
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.

9.3.4 RP Error #1 (RPER1) Register

This register contains the error status of the addressed drive.

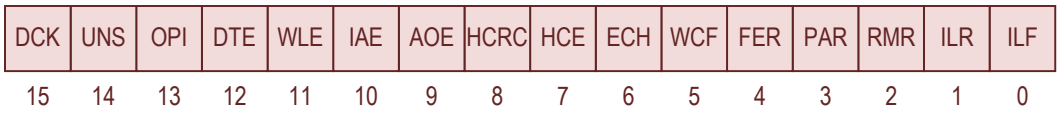


Figure 66 – RP Error Register #1 (RPER1)

Table 55 – 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 55 – 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*.
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*.

Table 55 – RP Error Register #1 (RPER1) – IO Address 776714			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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*.
0	ILF	R/W	Illegal function. Set by executing an illegal function per Table 52 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*.

9.3.5 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 67 – RP Attention Summary Register (RPAS)

Table 56 – 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].

9.3.6 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 68 – RP Look Ahead Register (RPLA)

Table 57 – RP Look Ahead (RPLA) – 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’.		
5-4	LAE	R	Look Ahead Extension		
			Bit 5	**Bit 4**	**Description**
			0	0	First quarter
			0	1	Second quarter
			1	0	Third quarter
			1	1	Fourth quarter
3-0	Zero	R	Always read as zero		

9.3.7 RP Maintenance (RPMR) Register

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

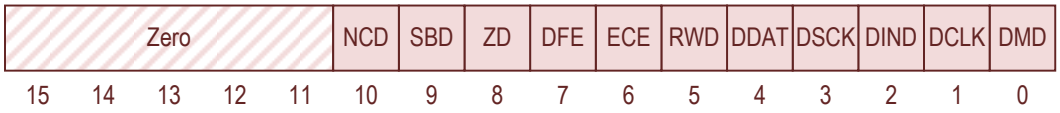


Figure 69 – RP Maintenance Register (RPMR)

Table 58 – 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.
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.

Table 58 – RP Maintenance Register (RPMR) – IO Address 776724			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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. 4592^{nd} to 4623^{nd} bits (16-bit mode) 2. 5104^{th} to 5135^{th} 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.
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).

Table 58 – RP Maintenance Register (RPMR) – IO Address 776724			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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).
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.

9.3.8 RP Drive Type (RPDT) Register

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

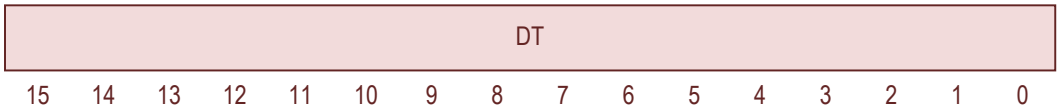


Figure 70 – RP Drive Type Register (RPDT)

Table 59 – RP Drive Type Register (RPDT) – IO Address 776726				
Bit(s)	**Mnemonic**	**R/W**	**Drive**	**Register Contents**
15-0	DT	R	RM03	020024
			RP04	020020
			RP05	020021
			RP06	020022
			RM80	020026
			RM05	020027
			RP07	020042

9.3.9 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.

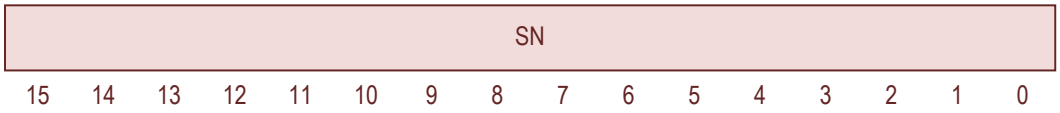


Figure 71 – RP Serial Number Register (RPSN)

Table 60 – 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

9.3.10 RP Offset (RPOF) Register

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

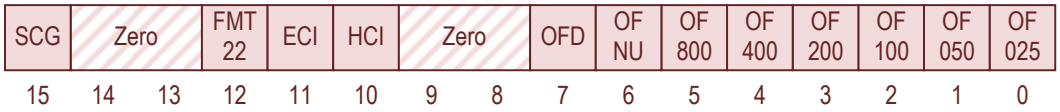


Figure 72 – RP Offset Register (RPOF)

Table 61 – 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.

Table 61 – RP Offset Register (RPOF) – IO Address 776732			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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 9.3.6. 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.
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.

9.3.11 RP Desired Cylinder (RPDC) Register

The cylinder is specified in this register.

Figure 73 – RP Desired Cylinder Register (RPDC)

Table 62 – 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*.

9.3.12 RP Current Cylinder (RPCC) Register

The RPCC register returns the Current Cylinder.

Figure 74 – RP Current Cylinder Register (RPCC)

Table 63 – 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.

9.3.13 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.

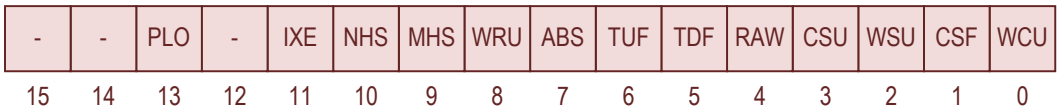


Figure 75 – RP Error Status #2 (RPER2)

Table 64 – 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*.
13	PLO	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*.

Table 64 – RP Error Status Register #2 (RPER2) – IO Address 776740			
Bit(s)	**Mnemonic**	**R/W**	**Description**
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*.
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*.

9.3.14 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.

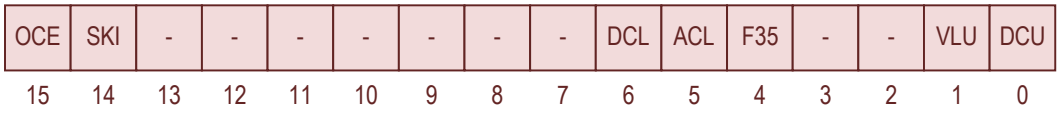


Figure 76 – RP Error Status #3 (RPER3)

Table 65 – RP Error Status Register #1 (RPER3) – IO Address 776742			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	OCE	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*.
13-7	-	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	R/W	Velocity Unsafe. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*

Table 65 – RP Error Status Register #1 (RPER3) – IO Address 776742			
Bit(s)	**Mnemonic**	**R/W**	**Description**
0	DCU	R/W	DC Unsafe. Not implemented. Cleared by *IO Bridge Clear, Controller Clear*, or *Drive Clear*.

9.3.15 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 77 – RP Error Position Register (RPEC1)

Table 66 – 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.

9.3.16 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 78 – RP Error Pattern Register (RPEC2)

Table 67 – 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.

9.4 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.

9.5 RPXX Commands

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

9.5.1 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 68 - 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
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

[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.

causes the controller to execute an auto-recalibrate operation. The SKI error and auto-recalibration operation is tested by DSRPA TEST-270.

9.5.2 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.

9.5.3 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.

9.5.4 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.

9.5.5 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.

9.5.6 Pack Acknowledge Function

Sets Volume Valid (RPDS[VV]).

9.5.7 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.

9.5.8 Release Function

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

9.5.9 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

[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.

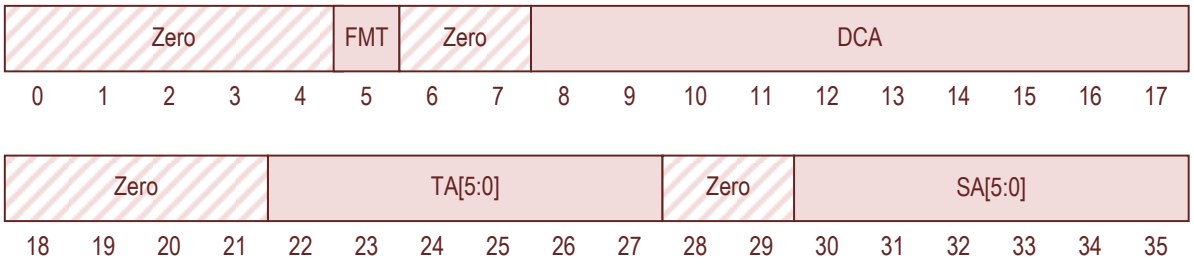


Figure 79 – Sector Header Word #1

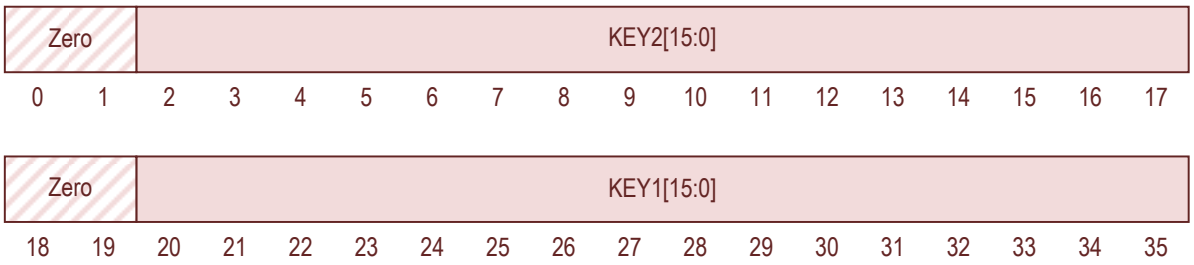


Figure 80 – Sector Header Word #2

9.5.9.1 Read header plus data

9.5.9.2 Read data

9.5.9.3 Write header plus data

9.5.9.4 Write header

9.5.9.5 Write check header plus data

9.5.9.6 Write check data

9.6 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 57.

9.7 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.

9.8 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.

9.8.1 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:

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

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 9.8.3.

This can be re-written as:

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

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.

9.8.2 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 81 – Sector Increment Algorithm

9.8.3 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 82 – 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.

9.8.4 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 69 below summarizes the disk parameters for some common disk drives.

Table 69 – 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
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.

9.8.5 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.

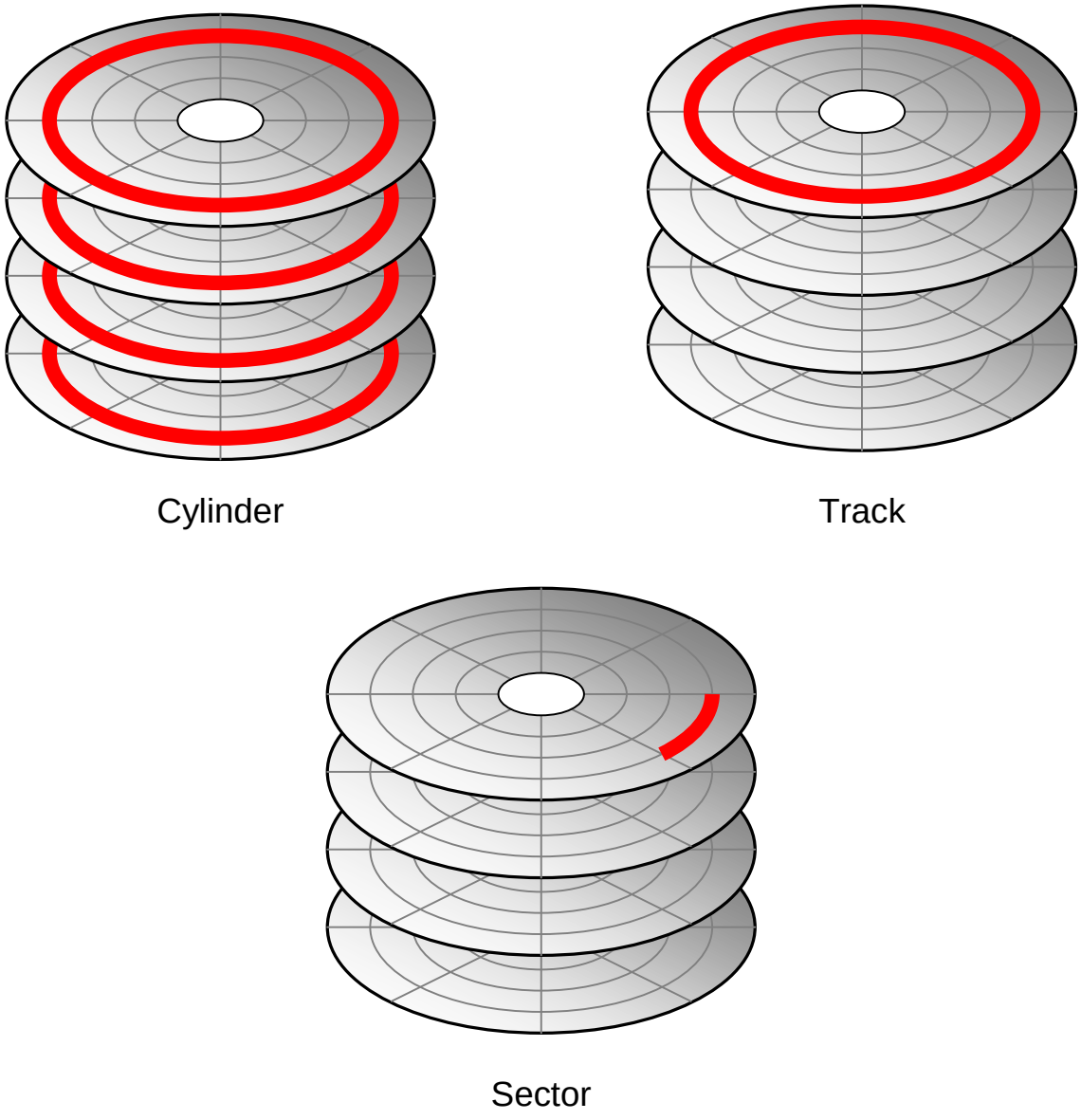


Figure 83 – Disk Cylinder, Track, and Sector

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.

9.8.6 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 84.

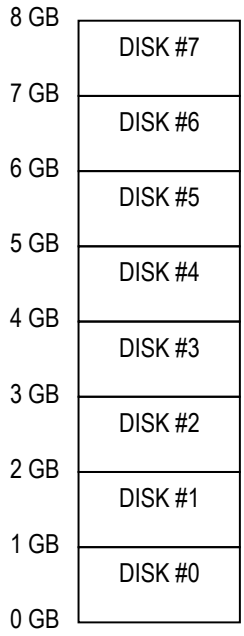


Figure 84 – SD Card Storage Allocation

10 LP20 Printer Controller

10.1 Control/Status A Register (CSRA)

The Control/Status A Register is byte addressable.

Figure 85 – Control/Status A Register (CSRA)

Table 70 – Control/Status A Register (CSRA) – IO Address 775400			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	ERR	R	Error This bit is asserted under the following conditions: 1. DAVFU Not Read (CSRB[DVOF]), or 2. Offline (CSRB[OFFL]), or 3. Line Printer Parity Error (CSRB[LPE]), or 4. Memory Parity Error (CSRB[MPE]), or 5. RAM Parity Error (CSRB[RPE]), or 6. Unibus Time-out Error (CSRB[MTE]), or 7. Demand time-out error (CSRB[DTE]), or 8. Go error (CSRB[GOE]) is asserted. Writes ignored.
14	PCZ	R	Page Counter Zero This is asserted when the page count is zero. An interrupt will be created when interrupts are enabled (CSRA[IE] asserted) and this transitions to asserted. GO (CSRA[GO]) will be reset when this transitions to asserted. Writes ignored.
13	UNDC	R	Undefined Characters. An interrupt will be created when interrupts are enabled (CSRA[IE] asserted) and this transitions to asserted. GO (CSRA[GO]) will be reset when this transitions to asserted. Writes ignored.
12	DVON	R	DAVFU Ready This is asserted when the DAVFU is loaded and used properly. Writes ignored.

Table 70 – Control/Status A Register (CSRA) – IO Address 775400			
Bit(s)	**Mnemonic**	**R/W**	**Description**
11	ONLN	R	On-line Writes ignored.
10	DHLD	R/W	Delimiter Hold
9	ECLR	W	Error Clear
8	INIT	W	Local Initialize
7	DONE	R	Done Writes ignored.
6	IE	R/W	Interrupt Enable When this bit is asserted, the following conditions will cause an interrupt: 1. Error (CSRA[ERR]) transitions to asserted, or 2. Page Zero (CSRA[PGZ]) transitions to asserted, or 3. Undefined Characters (CSRA[UNDC]) transitions to asserted, or 4. DONE (if DONE was set by byte counter going to zero). 5. DAVFU Ready (CSRA[DVON]) changes state, or 6. On-line (CSRA[ONLN]) changes state.
5-4	ADDR	R/W	Bus Address Extension [17:16]
3-2	MODE	R/W	Mode
1	PAR	R/W	Parity Enabled
0	GO	R/W	Go Note: CSRA[GO] must be asserted after all of the other bits in CSRA are set to the desired state.

10.2 Control/Status B Register (CSRB)

The Control/Status B Register is byte addressable.

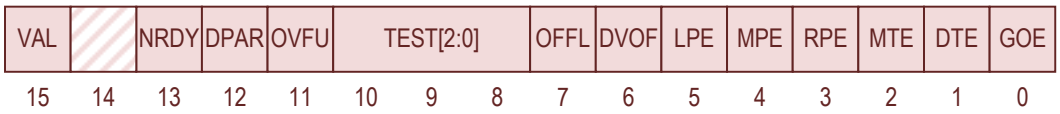


Figure 86 – Control/Status B Register (CSRB)

Table 71 – Control/Status B Register (CSRB) – IO Address 775402			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15	VAL	R	Valid Data
14	-	R	Reserved. Always read as zero.
13	NRDY	R	Printer Not Ready
12	DPAR	R	LPT Data Parity
11	OVFU	R	Optical vertical format unit
10-8	TEST	R/W	Test
7	OFFL	R	Off-line
6	DVOF	R/W	DAVFU not ready
5	LPE	R	Line Printer Parity Error
4	MPE	R	Memory Parity Error
3	RPE	R	RAM Parity Error
2	MTE	R	Unibus Time-out Error
1	DTE	R	Demand Time-out Error
0	GOE	R/W	Go Error

10.3 Bus Address Register (BAR)

The Bus Address Register is word addressable.

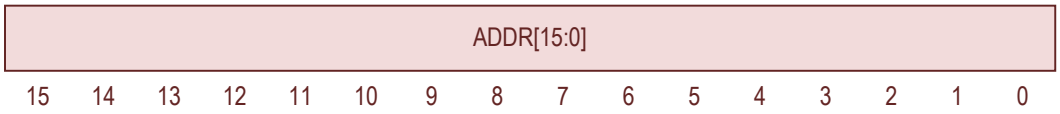


Figure 87 – Bus Address Register (BAR)

Table 72 – Bus Address Register (BAR) – IO Address 775404			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15:0	ADDR	R/W	Bus Address

10.4 Byte Counter Register (BCTR)

The Byte Counter Register is word addressable.

Figure 88 – Byte Count Register (BCTR)

Table 73 – Byte Count Register (BCTR) – IO Address 775406			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15:12	-	R	Reserved
11:0	COUNT	R/W	Byte Counter

10.5 Page Counter Register (PCTR)

The Page Counter Register is word addressable.

Figure 89 – Page Count Register (PCTR)

Table 74 – Page Count Register (BCTR) – IO Address 775410			
Bit(s)	**Mnemonic**	**R/W**	**Description**
15:12	-	R	Reserved
11:0	COUNT	R/W	Page Counter

10.6 RAM Data Register (RAMD)

The RAM Data Register is word addressable.

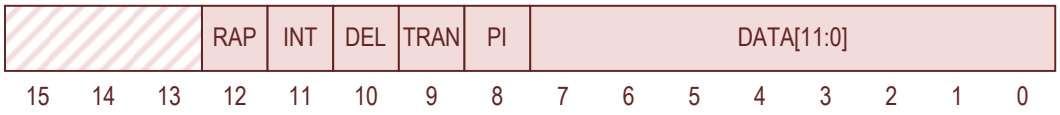


Figure 90 – RAM Data Register (RAMD)

Table 75 – 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 …
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.
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

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

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

Figure 91 – Column Counter Register (CCTR) / Character Buffer Register (CBUF)

Table 76 – 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

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

The Checksum Register and Printer Data Register are byte addressable.

Figure 92 – Printer Data Register (PDAT) / Checksum Register (CKSM)

Table 77 – 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	DATA	R	Printer Data

11 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	-	-	-	-	-	-	-	-

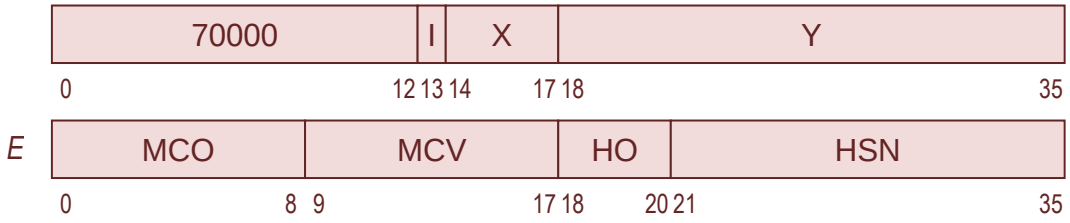
AC Field Assignments			
AC	**700**	**701**	**702**
00	APRID (BLKI APR)	UUO	RDSPB
01	UUO	RDUBR (DATI PAG)	RDCSB
02	UUO	CLRPT	RDPUR
03	UUO	WRUBR (DATO PAG)	RDCSTM
04	WRAPR (CONO APR)	WREBR (CONO PAG)	RDTIM
05	RDAPR (CONI APR)	RDEBR (CONI PAG)	RDINT
06	UUO	UUO	RDHSB
07	UUO	UUO	UUO
10	UUO	UUO	WRSPB
11	UUO	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

11.1 Executive Mode Instructions

11.1.1 Arithmetic Processor Interface (APR) Instructions

11.1.1.1 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 93 – APRID (BLKI APR) Instruction

Table 78 – 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
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

11.1.1.2 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:

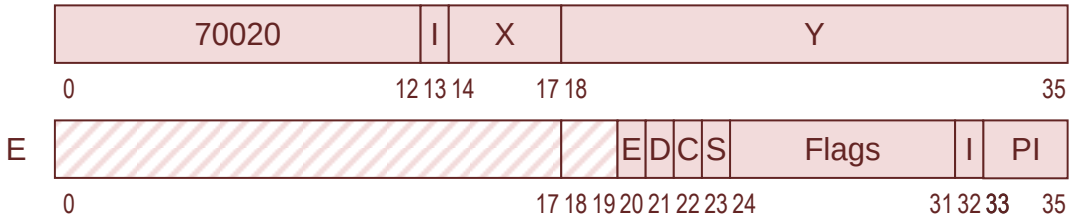


Figure 94 – WRAPR (CONO APR) Instruction

Table 79 – 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.
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.

11.1.1.3 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:

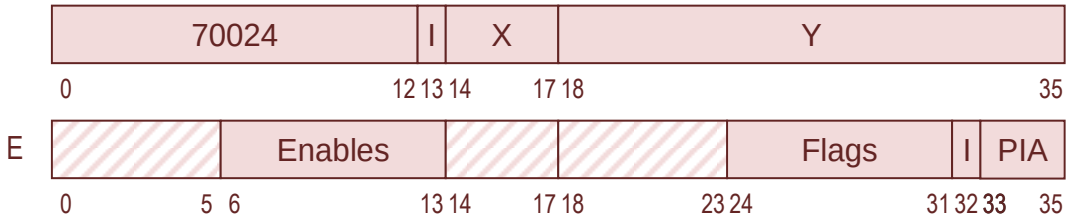


Figure 95 – RDAPR (CONI APR) Instruction

Table 80 – 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.
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)

Table 80 – RDAPR (CONI APR) Bit Definitions		
Bit(s)	**Description**	
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.

11.1.2 Priority Interrupt Controller (PI) Instructions

11.1.2.1 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

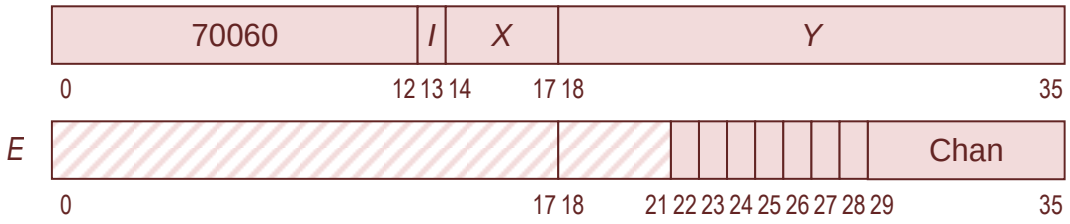


Figure 96 – WRPI (CONO PI) Instruction

Table 81 – 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

Table 81 – WRPI (CONO PI) Bit Definitions		
Bit(s)	**Description**	
29	Select	Channel 1
30		Channel 2
31		Channel 3
32		Channel 4
33		Channel 5
34		Channel 6
35		Channel 7

11.1.2.2 Read Priority Interrupt (RDPI/CONI PI)

This instruction stores the PI status in the word addressed by E.

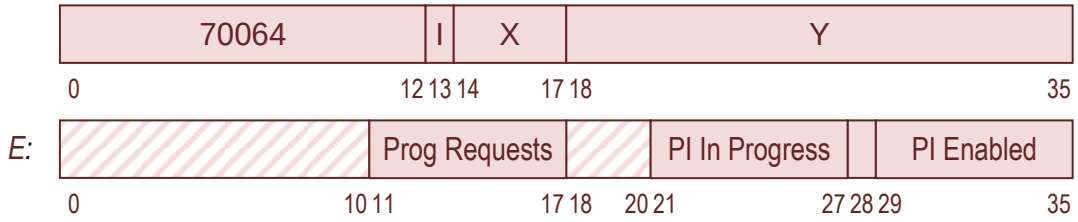


Figure 97 – RDPI (CONI PI) Instruction

Table 82 – 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

Table 82 – RDPI (CONI PI) Bit Definitions		
Bit(s)	**Description**	
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
32		Channel 4
33		Channel 5
34		Channel 6
35		Channel 7

11.1.3 User Base Register (UBR) Instructions

11.1.3.1 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.

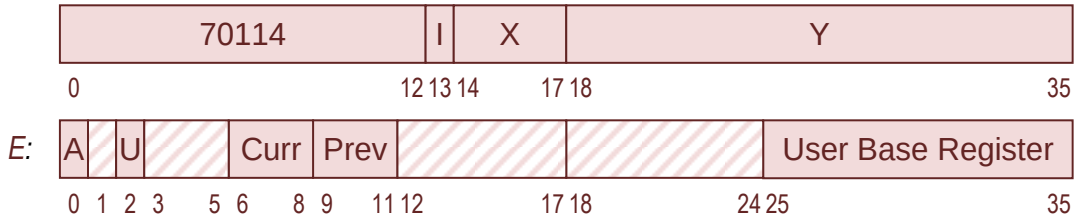


Figure 98 – WRUBR (DATO PAG) Instruction

Table 83 – 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

Table 83 – WRUBR (DATO PAG) Bit Definitions		
Bit(s)	**Bit**	**Description**
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
25-35	UBR	User base Register (Page Number)

11.1.3.2 Read User Base Register (RDUBR/DATI PAG)

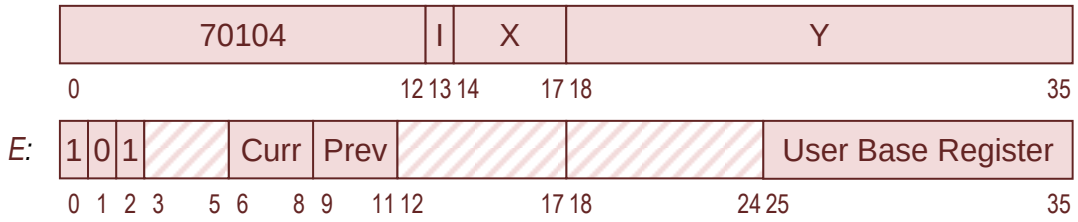


Figure 99 – RDUBR (DATI PAG) Instruction

Table 84 – 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)

11.1.4 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.

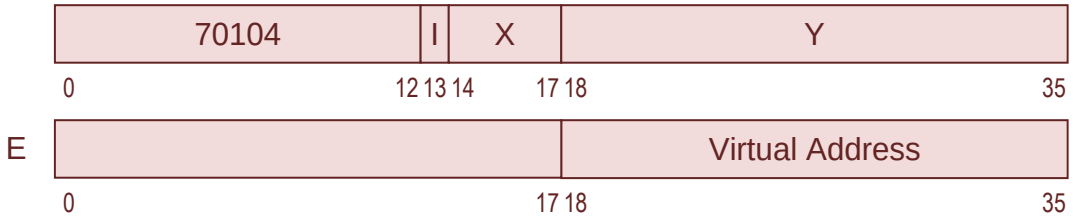


Figure 100 – CLRPT (BLKO PAG) Instruction

11.1.5 Executive Base Register (EBR) Instructions

11.1.5.1 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.

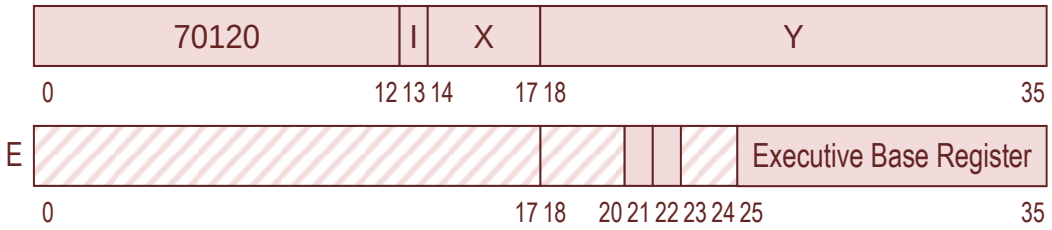


Figure 101 – WREBR (CONO PAG) Instruction

Table 85 – 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)

11.1.5.2 Read the Executive Base Register (RDEBR/CONI PAG)

Read the status of the pager into the right half of location E.

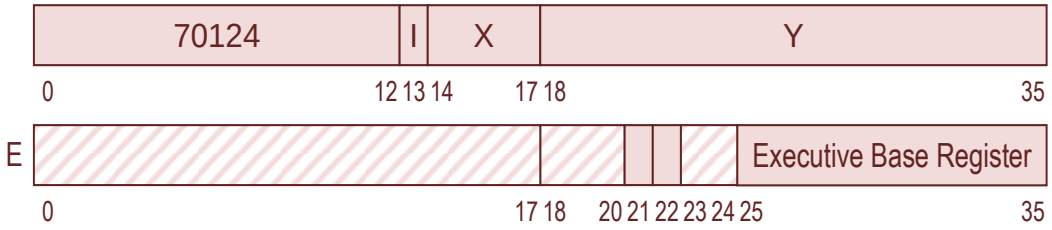


Figure 102 – RDEBR (CONI PAG) Instruction

Table 86 – 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)

11.1.6 Shared Pointer Table (SPT) Base Address Register

11.1.6.1 Read Shared Pointer Table Base Address Register (RDSPB)

Read the contents of the SPT base register into bits 14-35 of location E.

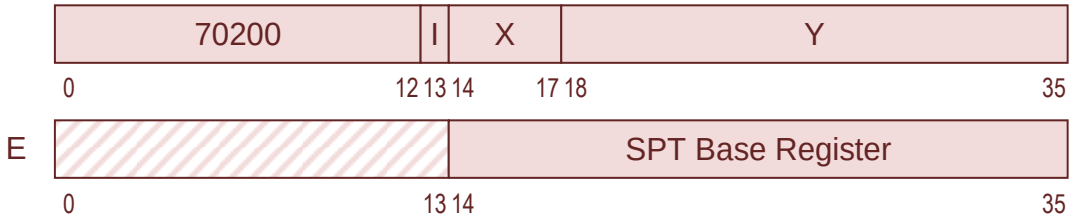


Figure 103 – RDSPB Instruction

11.1.6.2 Write Shared Pointer Table Base Address Register (WRSPB)

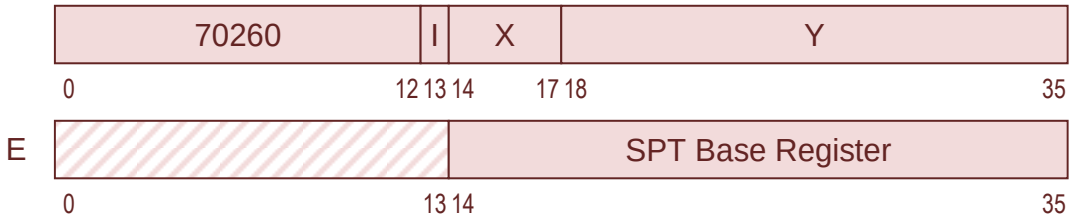


Figure 104 – WRSPB Instruction

11.1.7 Core Status Table (CST) Instructions

11.1.7.1 Read Core Status Table Base Register (RDCSB)

Read the contents of the Core Status Table Base Register into bits 14-35 of location E

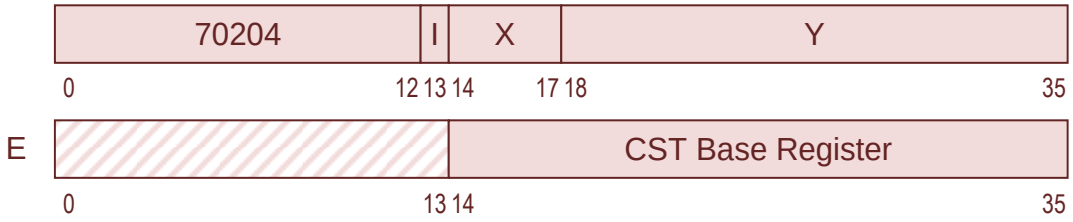


Figure 105 – RDCSB Instruction

11.1.7.2 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.

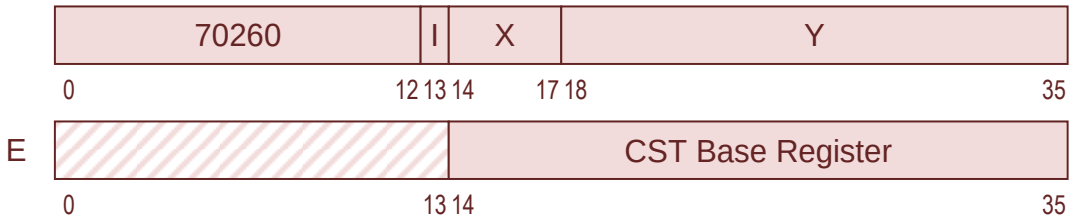


Figure 106 – WRCSB Instruction

11.1.7.3 Read Core Status Table Mask Register (RDCSTM)

Read the contents of the Core Status Table Mask Register into location E.

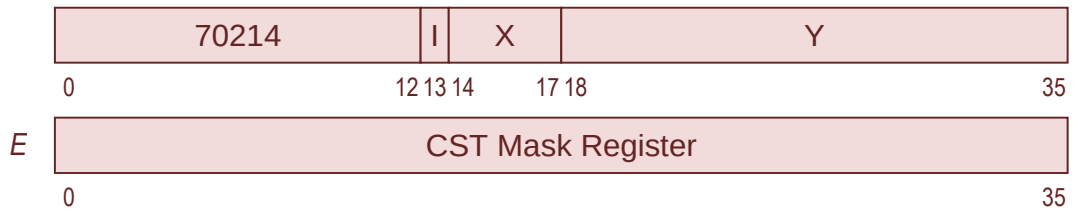


Figure 107 – RDCSTM Instruction

11.1.7.4 Write Core Status Table Mask Register (WRCSTM)

Load the Core Status Table Mask Register from E.

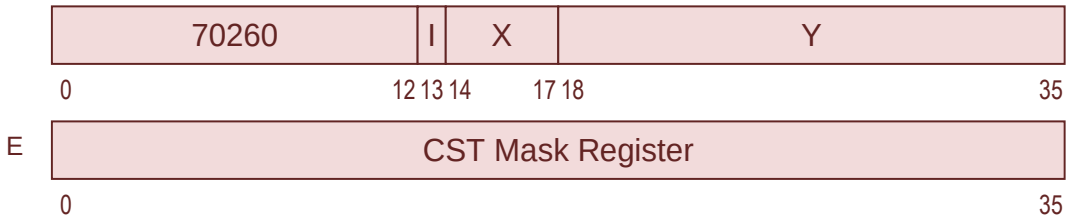


Figure 108 – WRCSTM Instruction

11.1.7.5 Read Core Status Table Process Use Register (RDPUR)

Read the contents of the Core Status Table Process Use Register into location E.

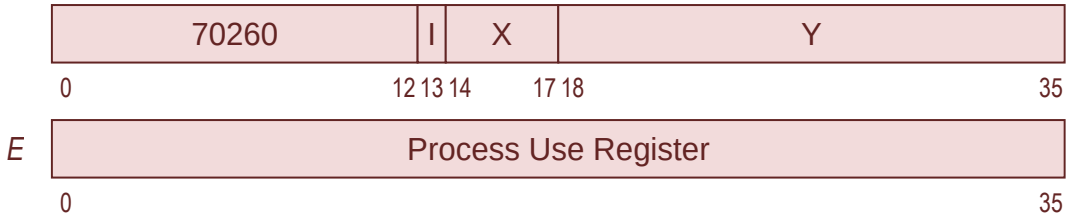


Figure 109 – RDPUR Instruction

11.1.7.6 Write Core Status Table Process Use Register (WRPUR)

Load the Core Status Table Process Use Register from location E.

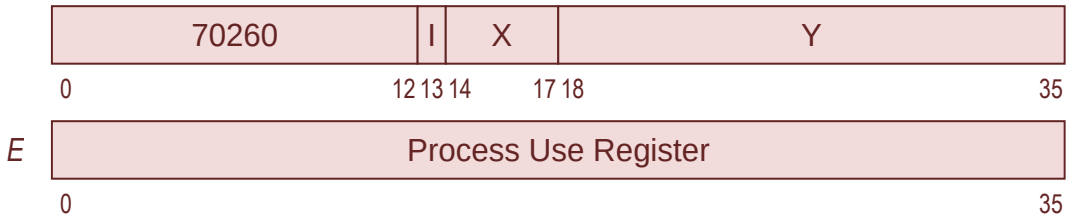


Figure 110 – WRPUR Instruction

11.1.8 Timebase Instructions

The timebase increments at 4.1 MHz.

11.1.8.1 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.

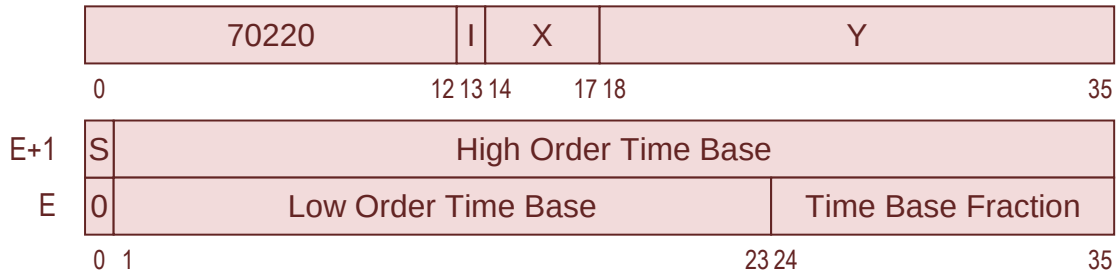


Figure 111 – RDTIM Instruction

11.1.8.2 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.

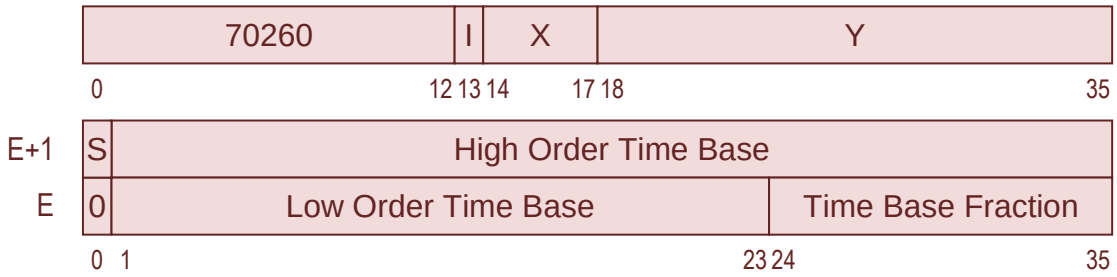


Figure 112 – WRTIM Instruction

11.1.9 Interval Timer Instructions

These instructions modify or query the interval timer.

11.1.9.1 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

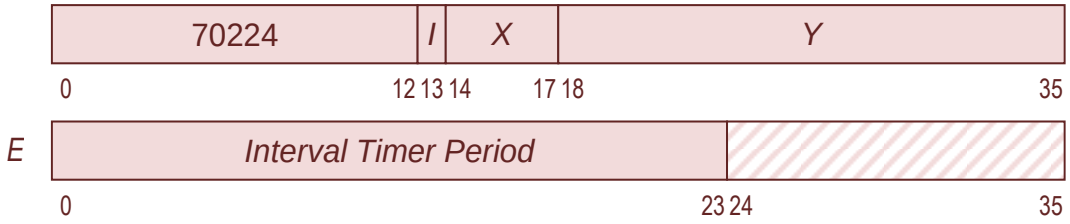


Figure 113 – RDINT Instruction

11.1.9.2 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.

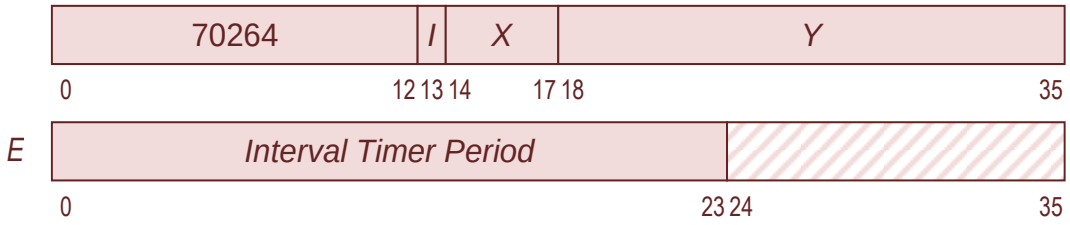


Figure 114 – WRINT Instruction

11.1.10 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.

11.1.10.1 RDHSB - Read Halt Status Block Address

Return the address of the Halt Status Block in E.

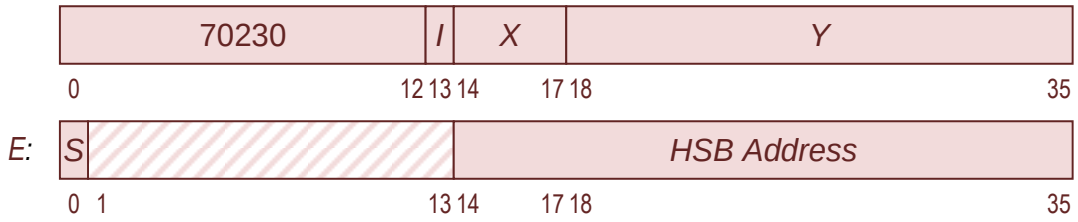


Figure 115 – RDHSB Instruction

11.1.10.2 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.

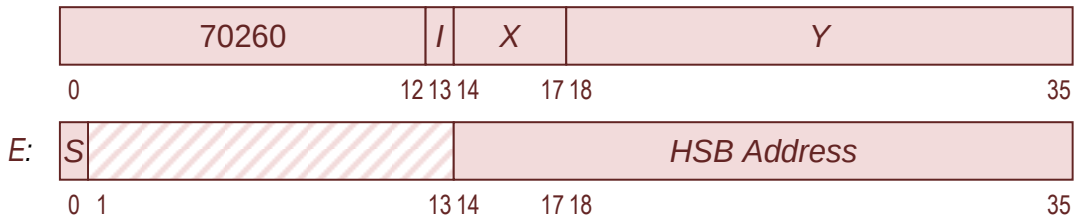


Figure 116 – WRHSB Instruction

12 Diagnostics

The following diagnostics have been executed in the simulator and on the target. The results are as follows.

Table 87 – Diagnostic Status

Test	Diagnostic Title	Sim Pass/ Fail	Target Pass/ Fail	Date of last test
DSKAAA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (1)	Pass	Pass	25 Nov 15
DSKABA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (2)	Pass	Pass	25 Nov 15
DSKACA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (3)	Pass	Pass	25 Nov 15
DSKADA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (4)	Pass	Pass	25 Nov 15
DSKAEA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (5)	Pass	Pass	25 Nov 15
DSKAFA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (6)	Pass	Pass	25 Nov 15
DSKAGA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (7)	Pass	Pass	25 Nov 15
DSKAHA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (8)	Pass	Pass	25 Nov 15
DSKAIA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (9)	Pass	Fail*	25 Nov 15
DSKAJA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (10)	Pass	Pass	25 Nov 15
DSKAKA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (11)	Pass	Pass	25 Nov 15
DSKALA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (12)	Pass	Pass	25 Nov 15
DSKAMA0	DECSYSTEM 2020 BASIC INSTRUCTION DIAGNOSTIC (13)	Pass	Fail*	25 Nov 15
DSKBAA0	DECSYSTEM 2020 BASIC INSTRUCTION RELIABILITY DIAGNOSTIC	FAIL	FAIL	25 Nov 15
DSKCAA0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (1)	Pass	Fail*	25 Nov 15
DSKCBA0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (2)	Pass	Pass	25 Nov 15
DSKCCA0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (3)	Pass	Fail*	25 Nov 15
DSKCDA0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (4)	Pass	Fail*	25 Nov 15
DSKCEA0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (5)	Pass	Pass	25 Nov 15
DSKCFC0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (6)	Pass	Pass	25 Nov 15
DSKCGB0	DECSYSTEM 2020 ADVANCED INSTRUCTION DIAGNOSTIC (7)	FAIL	FAIL	25 Nov 15
DSKDAB0	DECSYSTEM 2020 CPU & MEMORY RELIABILITY DIAGNOSTIC	FAIL	FAIL	25 Nov 15
DSKEAA0	DECSYSTEM 2020 PAGING HARDWARE DIAGNOSTIC	Pass	FAIL	25 Nov 15
DSKEBA0	KS10 - CACHE DIAGNOSTIC	FAIL	FAIL	25 Nov 15
DSKECB0	DECSYSTEM KS10 KL-PAGING TEST	FAIL	FAIL	25 Nov 15
DSKFAA0	DECSYSTEM 2020 INSTRUCTION TIMING DIAGNOSTIC	FAIL	FAIL	25 Nov 15
DSLTA	DECSYSTEM 2020 TELETYPE TEST	????	Pass	25 Nov 15
DSMMAB0	DECSYSTEM 2020 KS10 1024K MEMORY DIAGNOSTIC	FAIL	FAIL	15 Nov 15

DSMMBA0	DECSYSTEM 2020 BLT/FLOATING 1-0 MEMORY EXERCISER TEST	FAIL	FAIL	15 Nov 15
DSMMCB0	DECSYSTEM 2020 FAST AC DIAGNOSTIC	FAIL	Pass	15 Nov 15
DSMMDC0	DECSYSTEM 2020 MEM DIAG	FAIL	FAIL	15 Nov 15
DSRHAA0	DECSYSTEM 2020 MASSBUS ADAPTER EXERCISER	N/A	N/A	N/A
DSRMAB0	DECSYSTEM-2020 RH11-RM03 Basic Device Diagnostic	N/A	N/A	N/A
DSRMB0	DECSYSTEM-2020 KS10/RH11 RM03/RP06 Reliability Diagnostic	N/A	N/A	N/A
DSRPAC0	DECSYSTEM 2020 KS10/RH11 – RP06 BASIC DEVICE DIAGNOSTIC	Pass	FAIL	25 Nov 15
DSTUA	DECSYSTEM-2020 RH11-TM02/03-TU45/TU77 Basic Diagnostic	N/A	N/A	N/A
DSTUB	DECSYSTEM-2020 RH11-TM02/TM03-TU45/TU77 Magtape Reliability Diagnostic	N/A	N/A	N/A
DSUBAC0	DECSYSTEM 2020 UNIBUS ADAPTER EXERCISER	Pass	FAIL	25 Nov 15
DSDZAB0	DECSYSTEM 2020 DZ11 ASYNC L MUX	Pass	FAIL	25 Nov 15
*Fail – These test will pass if paging is inhibited (LH Switches = 000100)				

12.1 Diagnostics Summary

12.1.1 DSUBA DECSYSTEM 2020 UNIBUS ADAPTER EXERCISER

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.

```
* SMMON [DSQDC] - DECSYSTEM 2020 DIAGNOSTIC MONITOR - VER 0.3 *
SMMON CMD - STD

DECSYSTEM 2020 UNIBUS ADAPTER EXERCISER  [ DSUBA ]
VERSION 0.4, SV=0.3, CPU#=4097, MCV=130, MCO=30, HO=0, KASW=000000 000000

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

LH SWITCHES <# OR ?> - 000100
RH SWITCHES <# OR ?> - 000000
SWITCHES = 000100 000000
```

13 Building the KS10 FPGA System

The directory of the project is illustrated below in Figure 117

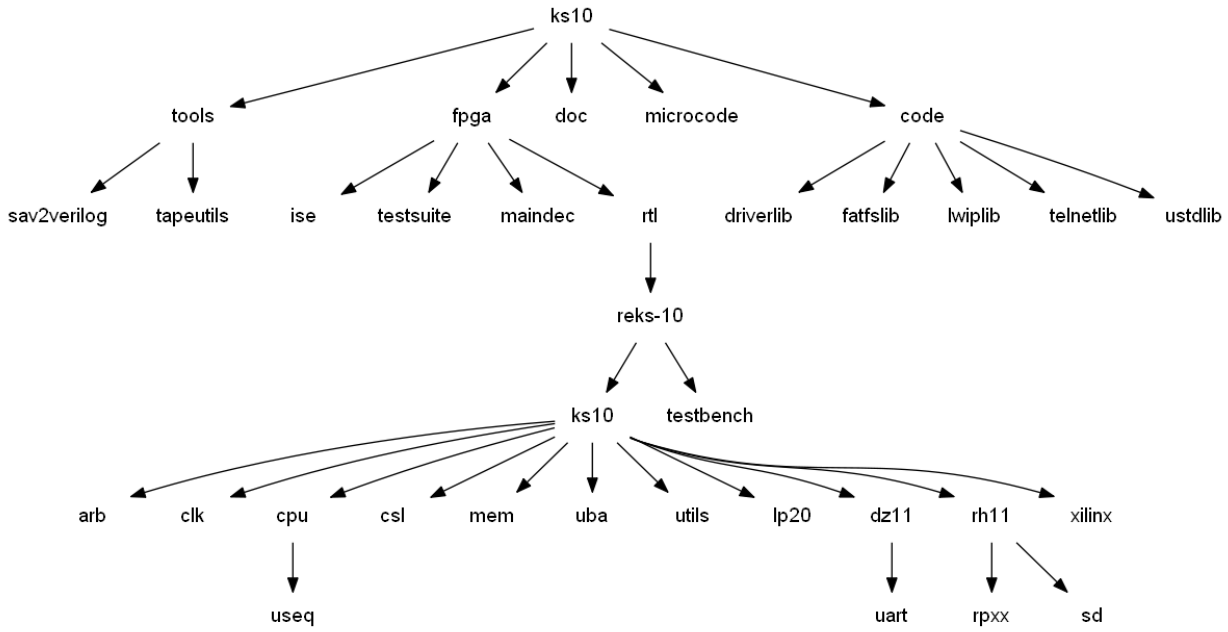


Figure 117 – Directory Structure

13.1 Tools

Linux (or Cygwin) development system including:

bash

make

rcs

awk

13.1.1 FTDI USB drivers.

This driver is required for the serial interface and for OpenOCD.

Include picture of FT-4232 configuration.

13.1.1.1 FTDI Hardware

Port 1

Port 2

Port 3

Port 4

13.1.2 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.

13.1.2.1 Xilinx ISE Webpack Version 14.7

13.1.2.2 Icarus Verilog

Icarus Verilog is used for the regression testsuite because it is faster than Xilinx ISIM.

iverilog-20130827

13.1.2.3 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 13.1.3.3 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.

13.1.2.4 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 13.1.2.3 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.

13.1.3 Software Tools

13.1.3.1 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.
```

13.1.3.2 GDB Debugger for ARM processors

13.1.3.3 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/

13.1.3.4 Eclipse Integrated Development Environment

Appendix A – SMMON Commands

Table 88 – 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 88 – 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 88 – 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 88 – 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 88 – 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 88 – 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 88 – 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 88 – 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?
```

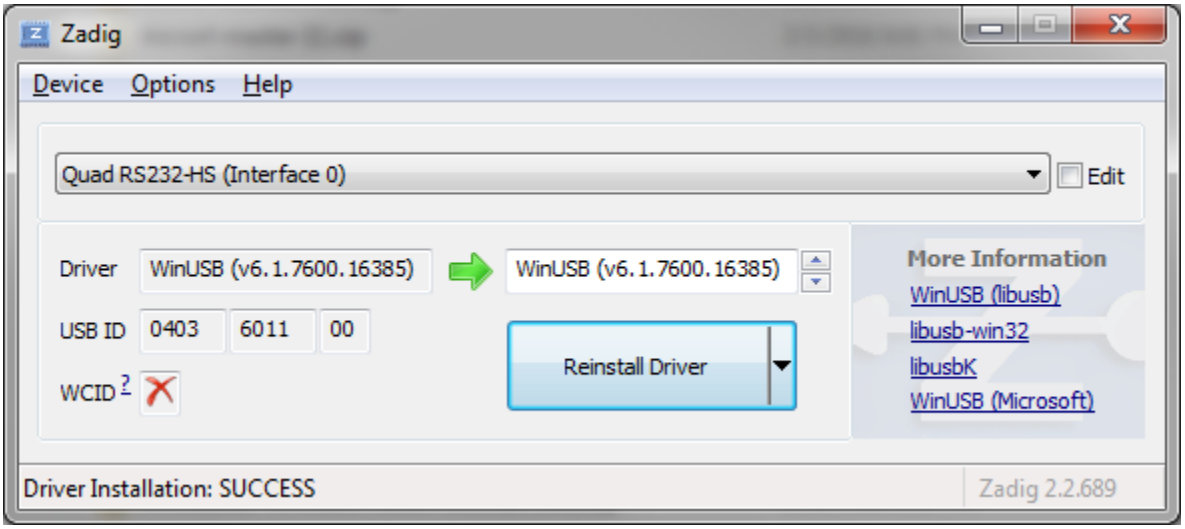
```
INF IND

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: