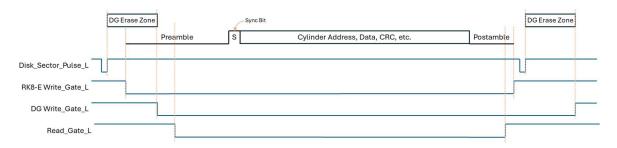
## Controller-independent drive emulator data format

Challenges for the previous emulator architecture:

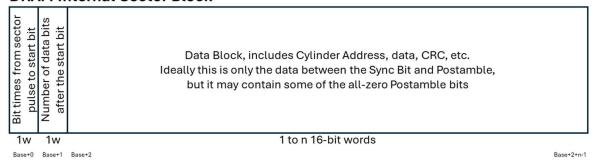
- Having the drive use controller-specific fields made it difficult to accommodate different controllers. It was necessary to know details of the timing of the disk bus signals for different controllers.
- The timing of some control signals, such as the Write Gate signal on Data General controllers, extend past the sector being written or read which was a complication for the emulator but solvable.

All disk formats in this family of drives and controllers have an all-zero preamble, a single one bit for the sync bit, some number of data bits stored in each sector which includes various fields in addition to data, and an all-zero postamble. Data General controllers keep the Write Gate signal asserted past the sector mark of the next sector.



The controller-independent format has the following data structure for each sector in DRAM and also each sector on the microSD card.

# **DRAM Internal Sector Block**



By referencing time from the sector pulse, the emulator will always output the start bit and data block at the correct time. It's just like bits on the surface of the disk platter.

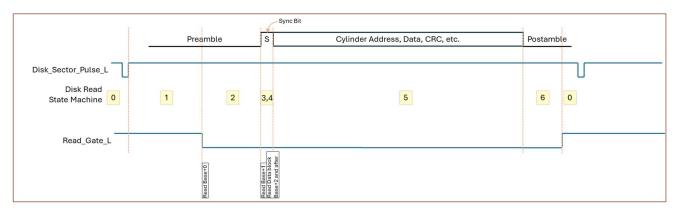
The fields are as follows:

- **Bit times from sector pulse to start bit** address Base+0, the number of bit times from the leading edge of the sector pulse to the start bit that follows the preamble. Since the preamble is an all-zero field, the emulator needs only to count the number of zero bits prior to the start bit and record the count in this field.
- Number of data bits after the start bit address Base+1, a 16-bit count of the number of bits in the following Data Block. The emulator knows that any bits following the Data Block are all-zero which is either all of the postamble of the last bits of the postamble.

• Data block – from addresses Base+2 up to Base+2+n-1, a block of 1 to n words which is the data for a sector on the disk which includes the Cylinder Address, sector data, and CRC. This field could also contain some of the postamble bits when sectors are written by the computer's disk controller. This is the same data that is presently stored in the DRAM and on the microSD card disk image file.

It is helpful to describe the events that occur during disk read and write operations to be able to implement these functions in FPGA logic.

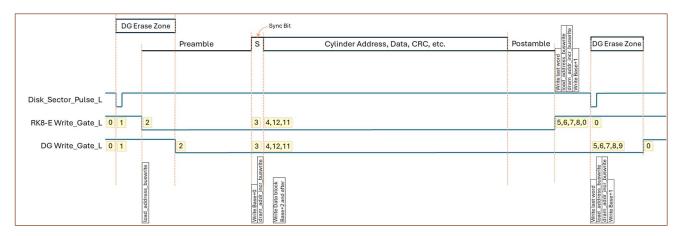
## **Disk Read Operation:**



The following sequence of events occurs when the disk controller reads a sector from the disk:

- 1. The Emulator generates a periodic sector pulse.
- 2. At the leading edge of the sector pulse, the Emulator internally loads the fieldbitcount counter to 1, which tracks the number of bit times since the sector pulse. The Emulator begins to increment the fieldbitcount counter once per bit time.
- 3. If the controller asserts the Read Gate signal, then the Emulator reads the first word of the DRAM Internal Sector Block from DRAM at address Base+0 which is the value: "Bit times from sector pulse to start bit" and saves it in cthreshold. The fieldbitcount counter continues to increment once per bit time. If there is no read operation (Read Gate doesn't go active after the sector pulse) and the next sector pulse arrives without having Read Gate asserted, then the fieldbitcount counter is loaded with 1 again at the leading edge of the sector pulse. This happens often.
- 4. The Emulator detects when the fieldbitcount counter is equal to cthreshold (which was read from Base+0 in step #3) and outputs the Start Bit at the proper clock/data phase time on the serial read data signal, BUS\_RD\_DATA\_L.
- 5. The Emulator reads the second word of the **DRAM Internal Sector Block** from DRAM at address Base+1 which is "Number of data bits after the start bit" and saves it in cthreshold. This is to know when to stop the process of reading from DRAM and outputting serial data to the Disk Bus. The fieldbitcount counter is reset to 1 because it now keeps track of how many bits are transferred following the Sync Bit.
- 6. The Emulator now outputs serialized data read from the Data Block in the DRAM starting at address Base+2. The fieldbitcount counter is incremented once per bit time. If the fieldbitcount counter is equal to cthreshold (which was read from Base + 1 in step #5) then the emulator advances to step #7. If Read Gate goes inactive before all bits of the Data Block have been read from DRAM and serialized, then reset the data read state machine to its starting state.
- 7. While Read Gate is still in the active state, the Emulator outputs all-zero data to the Disk Bus until Read Gate goes inactive.
- 8. When Read Gate goes inactive, then reset the data read state machine to its starting state.

# **Disk Write Operation:**



The following sequence of events occurs when the disk controller writes a sector to the disk:

- 1. The Emulator generates a periodic sector pulse.
- 2. At the leading edge of the sector pulse, the Emulator internally loads the fieldbitcount counter to 1, which tracks the number of bit times since the sector pulse. The Emulator begins to increment the fieldbitcount counter once per bit time. Prior to receiving the composite write data-clock signal from the controller, the emulator uses the internal bit time reference signal, clkenbl read bit, as a bit time reference.
- 3. The controller asserts the Write Gate signal.
- 4. The Emulator decodes serial data on BUS\_WT\_DATA\_CLK\_L to recover clock and data.
- 5. The emulator continues to increment the fieldbitcount counter but now uses the bit enable signal from the decoded BUS\_WT\_DATA\_CLK\_L signal as a bit-time reference.
- 6. When the Sync Bit is detected on the decoded BUS\_WT\_DATA\_CLK\_L, the Emulator writes the value in the fieldbitcount counter (which is the bit count since the leading edge of the index pulse) into the first word of the **DRAM Internal Sector Block** at Base+0. The Emulator sets the fieldbitcount counter to 1 which will now be used to count the number of bits that follow the Sync Bit.
- 7. Emulator decodes serial data on BUS\_WT\_DATA\_CLK\_L and writes data words to the **DRAM** Internal Sector Block area called Data Block which begins at Base+2.
- 8. When the Controller de-asserts Write Gate, the Emulator writes the last word or partial word of decoded data to the Data Block and then writes the value in the fieldbitcount counter to the second word of **DRAM Internal Sector Block** at Base+1 and the internal state machine returns to its starting state. If the Emulator asserts the periodic sector pulse prior to the inactive edge of Write Gate, then the Emulator interprets this event as the end of the sector data and the Emulator writes the last word or partial word of decoded data to the Data Block and then writes the field\_bit\_count counter to the second word of **DRAM Internal Sector Block** at Base+1 and the internal state machine goes to a state where no more data is written to DRAM and it waits for Write Gate to go inactive. In the event where Write Gate goes inactive following the assertion of the sector pulse, the internal state machine returns to its starting state when Write Gate goes inactive.

The file size on microSD is only two words per sector larger than the previous rk05 file format size, so the load and unload times are about the same.

Essential fields needed in the microSD file header to support the functions described above are:

- Number of Cylinders
- Number of Sectors per Track
- Number of Heads
- Microseconds per sector
- Bit rate

The Bit Rate field is the only controller-specific information needed by the Emulator.

Need a format-specific converter utility for each disk controller type to convert from SIMH to the emulator file format. Maybe this is only needed for PDP-8 and PDP-11. For less-commonly-used formats it is probably sufficient to have a utility to create an empty (all-zero data) file with header parameters in the list above, and the lengths of data blocks need to only be approximate given the data rate and number of sectors per track. The emulator disks can be formatted on each machine or users can develop their own SIMH converter for other formats using the source code for the PDP-8 utilities.

#### rke File Format Description:

The file on the microSD card has a file extension of "rke". The emulator RUN/LOAD switch is toggled to the RUN position, the emulator searches for the first rke file that it finds and loads that file into the emulator's DRAM. The rke file consists of a group of header fields followed by a group of sector data blocks. The sector data blocks can vary in size slightly. This is determined by the original lengths of each block when the microSD file was created and the width of the write gate pulse produced by the disk controller that may have written new data to the disk while it was loaded in the drive.

The file consists of Header + Data Blocks. For 16-sector PDP-8 there are 203 \* 16 \* 2 = 6496 blocks. For 12-sector PDP-11 there are 203 \* 12 \* 2 = 4872 blocks.

The rke file header consists of the following fields:

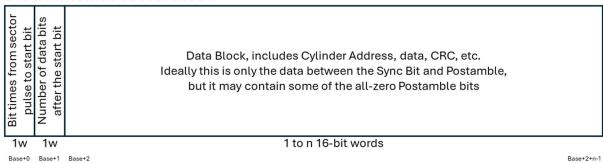
# bytes	Field Description	Hex Offset
10	char magicNumber[10] = "\x89RK05\r\n\x1A";	0x0
4	char versionNumber[4] = "1.1";	0xa
11	char imageName[11];	0xe
200	char imageDescription[200];	0x19
20	char imageDate[20];	0xe1
100	char controller[100];	0xf5
4	int bitRate; (big endian format)	0x159
4	int numberOfCylinders; (big endian format)	0x15d
4	int numberOfSectorsPerTrack; (big endian format)	0x161
4	int numberOfHeads; (big endian format)	0x165
4	int microsecondsPerSector; (big endian format)	0x169

Following the header there is a group of Sector Blocks beginning at offset  $365_{10}$  (0x16d). The number of Sector Blocks is:

numberOfCylinders × numberOfSectorsPerTrack × numberOfHeads

The format of each Sector Block is as follows:

#### **DRAM Internal Sector Block**



- uint16 bit\_times\_sector\_to\_start\_bit; (little endian format)
- uint16 data\_bits\_after\_start\_bit; (little endian format)
- uint16 data\_in\_sector[data\_word\_count]

The length of the Data Block in 16-bit words, data\_word\_count, can vary by a small amount in each sector because it is determined by the emulator measuring the length of the write gate pulse from the controller. A file might be initially created by converting a simH file to an rke file, in which case all Sector Blocks will be the same length. Then if that rke file is loaded into the emulator and some sectors are written by the disk controller, the sectors written by the disk controller can have a slightly different length than the sectors created by the simH conversion software. The number of 16-bit words in the Data Block is based on the data\_bits\_after\_start\_bit field value.

If (data\_bits\_after\_start\_bit % 16) == 0 then the Data Block length is (data\_bits\_after\_start\_bit / 16).

If (data\_bits\_after\_start\_bit % 16) != 0 then the Data Block length is (data\_bits\_after\_start\_bit / 16) + 1.

• Example of rke file contents	×
Eile Disk Edit View Options Registry Bookmarks Misc Help	
🗅 📂 🖫 🖟   🚜   magicNumber 🕨 🕦 versionNumber imageName	
000000	5a 45 52 isk formatted using the OS/8 ZER
000060 00 00 00 00 00 00 00 00 00 00 00	00 00 00
0000a0 00 00 00 00 00 00 00 00 00 00 00	00 00 00
000100 00 00 00 00 00 00 00 00 00 00 00	00 00 00
000140	02 fa 00 ËÄÎ.D³pÈÈú.
0001a0 numberOfSectorsPerTrack 4 9 microsecondsPerSector 7 f8 91 of 00 00 00 00 00 00 00 00 00 00 00 00 00	00 87 1f
0001e0 8c f9 d1 2a c8 numberOfHeads 70 01 02 2f c8 87 88 18 00 b0 00 82 0f a1 82 7c 88 08 02 00 000200 00 00 bc 00 00 00 00 00 00 00 00 00 00 00 00 00	17 20 f0   .ùÑ*Èp/Ȱ¡.  ð 89 98 88   .¼
000240	91 d3 08 ~('F©°À'È.OÎ(.fîéÆÈ.òm <sup>°</sup> .i'ó. 0d a8 1f â²n'.Iòø®n'.Iä.9@~(è.ñëÄnêò.¨.
000280	86 2e 20 .ì¢äëbÞè2Þá-«åMä ÿ¬./ðêÒ-hÏ-ê 91 04 f8åèô <sup>a⊚</sup> *ÞV.Þáâ]Þ 0-ä.N.ê-'.ø
0002c0 91 24 10 91 00 10 00 e0 4d 83 0c 08 00 00 68 91 1e ea bi 1e ec di 1e 87 el 9e c0 07 0002e0 f0 fa 7a 88 c0 00 e0 1d 20 f0 83 2c c2 fd 0b 08 6e 00 00 9e 4f 00 00 00 00 00 cd 00 000300 00 d8 8f 03 02 00 00 ff 2f 04 cc 24 11 d6 f4 60 fb 3f 0c 00 03 00 d6 f4 72 ee 9f 10 85	44 Oc 00 ðúz.à.à. ð.,âýn0Í.D
000320	15 d6 f4 D .ÖôrôÿÖôrç_HÏ4HÖôrö.O.@.Öô
000380 90 41 d6 f4 72 f9 4f 11 92 41 35 d6 f4 72 f9 4f 50 92 41 35 d6 f4 72 f7 8f Bit times from st 0003a0 fa 9f 40 00 04 00 d6 04 00 f5 1f 0c 50 03 00 d6 f4 72 f3 0f 15 43 02 00 d6 pulse to start bit	ector f4\72 . Aöôrùo.'A5öôrùoP'A5öôr÷.IÏ.Aöôr t 00\05 ú.@öPöôróCöôrò_L
0003c0 00 d6 04 00 f2 5f 20 10 03 00 d6 f4 72 f8 bf 48 06 43 35 d6 f4 72 f7 9f 30 b 00 00 sc 0003e0 5f 09 4c 02 10 d6 04 00 df 5f 20 10 03 00 0c 72 7a c9 ff 24 00 02 00 8c f4 72 f9 cf 18 000400 8c f4 72 fb 4f 55 4c 92 51 8c f4 72 fd ff 40 c5 35 49 8c f4 72 fd ef 24 05 25 1d 8c f4	<del>□</del>
000420	<u>00 00 δC [%x31</u> .ôrþ.Môrþ/QGôrþ0.N
000460 cd 03 40 d6 f4 7a ef 2f 09 13 05 00 d6 04 00 f1 af ea c0 5e ec 20 ef ac b6 cf 29 00 d8 000480 51 f5 15 00 00 00 00 00 00 cc 00 44 0c 00 00 d8 cf 1d 03 00 00 ff 1f 40 d3 04 cc d6 04 00004a0 08 c9 04 0c 46 f0 7a fc 1f 084c9 04 0c c6 f4 7a fc 1f 08 c9 04 0c 86 f1 7a fc 1f 40 38	00 f8 lf QõÌ.DøÏÿ.@ó.Ìöø.
0004c0 04 00 ed 5f 50 cf 00 00 d6 04 00 ea cf 08 c1 03 10 d6 04 00 f8 5f 48 c5 04 44 81 f0 7a 0004e0 89 83 3d 8c 74 73 fd 5f 1c 89 83 3d c2 74 4d f2 6f 3d 84 64 49 41 03 00 ef 1f 14 92 30	fa 5f 1ci_Pïöêï.Áöø_HÅ.D.ðzú 51 8e f0=.tsý=ÂtMòo=.dIAï'0Q.ð
000500 7a e8 3f 3c cd 03 40 d6 04 00 fc 3f 3d 09 04 40 d6 04 00 f3 1f 40 d3 04 c8 d6 f4 7a ec 000520 f4 c8 d6 f4 7a fb ff Bit times from sector 04 00 f8 1f 48 06 03 00 d6 f4 72 ed ff 30 44 00 00 000540 f0 3f 35 81 00 50 d6 nulse to start bit 92 00 04 d6 f4 72 f5 1f 4c 92 00 00 d6 04 00 e8 1f	d6 f4 72   ôÈÖôzûÿÖø.HÖôríÿODÖôr
000560 14 d6 04 00 ad 9f 30 00 00 fd 3f 30 Number of data bits 00 fd 3f 54 81 00 4c 89 00 00 ff 3f 30 Number of data bits 00 fd 3f 54 81 00 4c 89 00 00 ff 3f 30	04 00 e5 .Ö0ÖÉ/-FÖô0Öå 81 00 4c ?TÙ.Lý?0ù.Lý?TLÿ?0L
0005a0 89 00 00 ff 1f 08 c9 04 0c 46 05 after start bit 05 20 d6 04 00 f6 4f 14 54 02 00 d6 04 0005c0 1b 00 00 d6 04 00 ec 2f 19 13 00 00 00 00 d6 04 00 df ef 14 8f 00 3c d6 04 00 df ef 14 cf 50 10 d6 04 00	4c 25 d6öì/öæÿ\$\$[./.L%ö
000600	f6 8f 51 À^ì זּÏ).Øé;>Q×Î.Dö.Q 00 f0 3fÿÏP.ÖßKñ4ÖñÖð?
000640	f2 6f 3dø?.@ö×?öø^.=ÂtMòo=
0006a0 00 f3 1f 40 d3 04 c8 d6 f4 7a ec 1f 40 d3 f4 c8 d6 f4 7a fb ff Bit times from sector 04\ 00 f8 0006c0 03 00 d6 f4 72 ed ff 30 44 00 00 d6 f4 72 f0 3f 35 81 00 50 d6 pulse to start bit 92\ 00 04	1f 48 06 .ó.@ó.Èöôzì.@óôÈöôzûÿöø.H. d6 f4 72öôríÿODöôrð?5pöé.0'öôr
0006e0   f5 1f 4c 92   00 00 d6 04   00 e8 1f 34   d2 c0 14 d6   04 00 ad 9f   30   <del>00 03 00 04 04 04   00 05   26   00 00 04   00 05   26   00 05   26   00 05   26   00 05   26   00 05   26   00 05   26   00 05   26   00 05   26   00 05   26   00 05   26   00 05   26   00 05   26   00 05   26   00 05   26   00 05   00 05   26   00 05   00</del>	2d 46 <b>1</b> õ.L'Őè.4òà.ÖOÖÉ/-F
Offset 1791=0x6ff Bits=00000001 Unsign after start bit	ANSI / OVR / L Size: 2585773