Pertec Tape Controller, Version 1.0

Firmware version 1.2

April, 2024
Chuck Guzis

1. FOREWORD

This document describes the operation of my Pertec tape drive controller. This controller consists of the Pertec bus interface logic and an STM32F4 microcontroller, although any microcontroller of sufficient capabilities can be used.

Frequent reference is made to the Pertec interface description, located in the file 'Pertec 9-Track Tape Interface.pdf'. The library used for the microcontroller program is located at https://libopencm3.org/. The microcontroller data sheets can be found at http://www.st.com.

2. INTERFACE CONVENTIONS

The Pertec interface is typical for its time; 5 volt TTL, with transmitting done via open-collector logic terminated in 220 ohm to +5, 330 ohm to ground resistor networks. Logic is negative; that is, 0 volts is true, 5 volts is false.

There are about 52 unidirectional signal lines that are distributed over two 50-conductor cables with interleaved ground lines. Depending on vendor implementation, some of these signals may not be used. Others may have vendor-unique interpretations.

Unlike many protocols, the Pertec protocol does not involve handshaking. Data read is transferred from the tape formatter and marked by pulsing a strobe line true--there is no corresponding controller-to-formatter protocol signalling that the data has been accepted. Similarly, when writing, whatever data is present on the controller-to-formatter data lines is transferred to tape and the transfer marked by pulsing a "data written" strobe.

Various other signals may be pulsed during the read or write process; it is up to the controller to "catch" these signals.

Commands to the tape formatter involve data transfer or tape motion without data transfer. For data transfer (i.e. read and write), operations are initiated by the controller pulsing the "IGO" line; that is, bringing the level low for at 2 microseconds. Operation commences on the trailing edge of the pulse and "formatter busy" is asserted. The type of operation is determined by other command lines.

For example, a simple read in the forward direction involves pulsing IGO with no other lines asserted. As each byte is presented to the controller, the IRSTR line is pulsed. A read in the reverse direction involves asserting IREV with IGO.

Writing is initiated by asserting IWRT with IGO. As the formatter acceptseach byte of data from the controller, IWSTR is pulsed. However, see below for the use of ILWD (last word) to signal the end of a block.

All data transfer operations involve the formatter setting IDBY active for the duration of the transfer, after which IFBY is also de-asserted.

Non-data transfer, i.e. tape motion, operations do not set IDBY, but do set IFBY (formatter busy) and do not involve IGO. So, to rewind a tape to load point, IRWD is pulsed, IREW goes active, with IRDY (drive ready) going inactive. When load point is reached, IREW goes inactive and IRDY goes active.

Similarly, if IRWU is pulsed (some implementations require that IREW also be asserted), the tape rewinds and is unloaded and is ready for removal.

Block length is signaled on a read by IDBY going inactive during a read operation. On writing, ILWD must be asserted just before presenting the last byte of data in a block. Generally, ILWD can be asserted immediately after IWSTR goes inactive for the penultimate byte in a block.

Errors are signaled during a transfer; there are two general types. ICER is pulsed for a "corrected" error during reading. IHER is pulsed for a "hard" (uncorrected) error during both read and write operations. IFMK is pulsed during a read when a file mark (tape mark) is encountered.

There are other operations that can be implemented by a given drive/formatter combination; many of these are vendor-unique and are discussed in the documentation for your particular drive.

3. CONTROLLER IMPLEMENTATION

The controller as implemented here consists of two main parts. The first part is simply TTL level logic to interface to the tape formatter and consists mostly of flip-flops, receivers and random logic. The purpose is to reduce the 52 signal lines down to fewer I/O lines and to catch any pulses emitted by the formatter. The second part is a microcontroller that represents the "brains" of the controller.

3.1 INTERFACE LOGIC

Please refer to the schematic for this narrative.

3.1.1 TAPE DRIVE CONNECTORS (Sheet 2)

All signal lines are shown. Only 2 are unconnected; ISGL which, on some drives signals 800 bpi NRZ mode; and IDENT, which indicates that the drive is reading the identification burst information. Neither controls drive operation and either does not appear on many drive implementations.

IFEN is strapped as permanently active. The use of multiple separate formatters on drives shared by multiple hosts is not common on current systems.

All other formatter-to-controller signals are terminated by 220/330 ohmresistor networks RN1, RN2 and RN3.

3.1.2 TAPE CONTROL REGISTERS (Sheet 3)

U10 and U11 are 74ABT574 8 bit latches that are capable of driving 60 ma loads. Other families' version of this latch may be used, provided that a 40 ma drive capability is ensured. They are connected to 16 control bits on the formatter interface. At startup, they are both tri-stated (high impedance), and enabled by bringing the CMDENA line low from the microcontroller. The upper or lower 8 bits of the 16 output signals are clocked from the microcontroller via the CMDSEL0 and CMDSEL1 lines. Since the latches are in high-impedance mode at startup, they cannot affect drive operation even if they are loaded with random values. As usual, these lines are low-active.

N.B. It is possible to reduce the number of GPIO bits required by connecting the command data lines to the MCU data in/out lines and loading either half of the control registers from the data lines. Logic will be a bit more complicated, however.

3.1.3 TAPE DATA REGISTERS AND DATA CONTROL (Sheet 4)

Data in and out to the formatter is latched by U1 and U2 (74ABT574). Both are connected to the microcontroller which must operate the common lines bidirectionally. The data direction (read or

write) is controlled by the DDIR signal from the microcontroller, which tri-state enables one or the other (OE is driven through inverter U3A). In the case of output during a write operation, odd parity is generated by U5, 74LS280. Many drives do not use this signal, but it's provided for completeness.

Read and write strobes (IRSTR and IWSTR) are caught on the trailing edge by flip-flops (74LS74) U4A and U4B. The output of these is incorporated in the status registers (see section 3.1.5, Sheet 6).

Both registers are reset by the microcontroller pulsing TACK low.

Note that the output register U2 must be latched with data before IGO and IWRT are asserted. Thereafter, WDEMTPY is asserted after each byte is accepted by the formatter. Prior to the last byte to be written is latched into U2, ILWD must be asserted.

For reading, RDAVAIL is set when the formatter makes a byte of read data available.

3.1.4 POWER/MICROCONTROLLER INTERFACE (Sheet 5)

Sheet 5 describes the headers dedicated to the microcontroller interface. These are wired to appropriate pins on the microcontroller card.

+5 volts is provided from two sources; the first is the external connection via J3; the second is supplied via the microcontroller card, which is, in turn supplied by the USB interface. Schottky diode D1 prevents external power coming in via J3 from over-driving the USB power.

Bypass capacitors (104) are supplied at every IC.

3.1.5 STATUS REGISTERS (Sheet 6)

Tri-state buffers U6 and U7 furnish status information from the formatter as well as the transient status for file mark, corrected error and hard error, latched by U8 and U9. These 3 status bits ar reset at the trailing edge of the IGO signal, as all three statuses occur during data transfer.

Output from U6 and U7 is selected by the SSEL microcontroller signal. The status bits are arranged such that the status selection need not be changed during the data phase of a read or write command.

4. MICROCONTROLLER OPERATION OUTLINE

The code required to run the Pertec interface is straightforward. The basic tasks are as follows:

1. Initialize clocks, GPIO, tape interface, SD card interface and USB communication interface.
2. Poll for next operation to be performed.
3. Perform the operation; go to step 2.

4.1 INITIALIZATION

On microcontroller powerup, all GPIO ports should be in input mode with weak pullup enabled. This prevents any undesired activity on the tape drive.

The interface is organized as several 8-bit ports, set up as follows:

1. Data port (J4), bidirectional, initially in input mode.
2. Status port (J5), input mode.
3. Control port (J6), output mode
4. Register select port (J7), output mode

On some microcontrollers, particularly ARM-based ones, GPIO ports are generally 16 bits. Writing a value to an 8 bit half of a 16-bit register is a bit unconventional. One first sets all bits in the 8 bit half to ones, then clears the appropriate bits that are desired to be zeroes.

The register select (J7) port should be first initialized with all high-level bits (ones). This has the effect of disabling the control port output.

Next, the control port (J6) should be written with all ones in both 8 bit halves, using the register select bits (CMDSEL0 and CMDSEL1) set appropriately.

Register select bit TACK should be pulsed to reset the data status bits.

SD card, communications and clock should be set up as appropriate for the choice of microcontroller and software.

4.2 TAPE POSITIONING

Tape can be rewound, unloaded, or spaced in the forward direction by first ensuring that the tape drive is online and ready. This is done by polling for the IRDY bit active. If a rewind is attempted, it can be skipped if the ILDP (load point) status is active.

The appropriate command bits are set in the control port using the register select (CMDSEL0 and CMDSEL1) bits. Before setting, the control port output should be disabled (CMDENA bit), the control register loaded, then enabled.

Operation status should be monitored until complete.

4.3 TAPE READING

Tape reading operation is similar to the tape positioning protocol, with the difference that IGO is pulsed at the beginning of the operation. Further, TACK should also be pulsed to clear the "read data available" flag. All error flags are reset upon release of IGO. Tape motion ensues, with IFBY being set. When data has been processed by the formatter, IDBY goes active and each byte read setting the RDAVAIL flag in the status register. The data byte can then be read from the data register and stored. The transfer is then acknowledged by pulsing TACK. At the conclusion of the tape block transfer, IDBY is released. On many streaming drives, it is not necessary to wait for IFBY to clear; the next command can be issued immediately. Error flags and file mark indication are available in status register.

Most tape formatters read an entire block, process it, then transfer it to the host. As a consequence, transfers occur in bursts, with idle time between. This should leave more than enough time to write the block to the SD card; in our implementation on the STM32F407, the data write to SD is initiated using DMA transfer, such that SD writing can overlap with tape reading.

4.4 TAPE WRITING

Tape writing is initiated in a manner similar to tape reading, with several important differences. The data register is set to output mode; output data is latched into the data register before the write operation is initiated. Upon data being transferred to the tape drive, the WDEMPTY bit is set in the status register and the next data byte should be latched into the data register. Before the last byte of a block is written, the ILWD bit should be set in the control register. This can be done as soon as the WDEMPTY bit is set for the preceding byte. Note that the IFPT (file protect) bit will be set in the status register if the tape has not been write-enabled (no "ring").

Tape marks are written by asserting IWRT and IWFM as well as IGO. This will result in a single file mark being written. The IEOT status should be tested before each block write to avoid writing past end-of-tape.

5.0 MISCELLANEOUS NOTES

A few implementation-specific details should be noted here.

The FAT32 file system implemented on the SD card, is realized using Elm Chan's FATFS. Single file size is limited to 2GB, which should not be a problem, even at 6250 GCR density.

Our implementation time- and date-stamps each file as it is created. This is done by virtue of the microprocessor card used containing a clock backup battery.

By default, tape image files are recorded in SIMH TAP format.

The source code contains provisions for host interface via USB CDC ACM or, alternatively, serial asynchronous I/O at 115200 bps. The selection is done by specifying a compile-time option.

Transfer from SD card to host can be done in three ways. The SD card can be removed and inserted into a suitable SD reader on the host system. As an alternative, YMODEM 1K transfer is implemented. Many terminal programs, such as Linux minicom support this protocol.

Finally, if the usb connection feature is employed, the SD card file system is exposed on the host as a USB-connected drive. Care should be used when modifying the contents of this drive from the host, as the data will not be flushed to the card until the card is either sync-ed or dismounted on the host. Not doing so can lead to file system corruption.