PPM tape interface for small computers

This article describes a method for recording programs and data from a microcomputer onto a cassette recorder using pulse position modulation (PPM). The advantages of this mode are that the interface required is simple and inexpensive, and the necessary computer routines are short.

by J. G. LANG

CSIRO Division of Applied Geomechanics, PO Box 54, Mt. Waverley, Vic. 3149.

The writer has built a microcomputer like the Miniscamp (Electronics Australia," April, May 1977) but based on the Signetics 2650 microprocessor. Like the Miniscamp, the data is entered through console switches and displayed on a row of eight LEDs, but an important difference is that the two functions of entering the HOLD state and reading data from the console switches are separated. The advantage of this is that the data switches can be read on the run. Another difference is that 512 bytes of RAM are installed, instead of the original Miniscamp's 256.

All such small console switch oriented systems have one thing in common. Before very long you start to look for some means of recording your programs so as to minimise the time-consuming operation of reloading programs through the console switches. This article describes the writer's solution using Pulse Position Modulation (PPM) on an ordinary audio cassette magnetic tape recorder.

On account of cost and availability an unmodified cassette recorder is the obvious medium, but a choice must be made between several available modes for recording digital data. From amongst phase modulation (PM), frequency-shift keying (FSK), pulse-duration modulation (PDM) and pulse-position modulation (PPM), the last was chosen. PPM has the advantages that only a very simple interface is involved and the elemental loading program can be made quite short. The length of this program is important as it has to be loaded by hand every time the microcomputer is powered up. Note that we are dealing with simple systems with limited RAM and no ROM.

PULSE POSITION MODULATION

With PPM a string of pulses is stretched out in time or along a length of magnetic tape as illustrated in Fig. 1. All the pulses have the same shape but the time interval between them is varied to indicate different values. For binary data we only need two intervals and those used here are 1667*us* (microseconds) for a logical "**zero**" and 3333*us* for a logical "**one**." On replay, the time between pulses is measured. If the interval is less than 2500*us*, then it is represented as a "**zero**," if more than 2500*us* then as a "**one**." 3333*us* between pulses is equivalent to 300Hz and 1667*us* is equivalent to 600Hz. The discriminating interval of 2500*us* is half way between the two nominal values and thus gives the greatest possible latitude in accommodating variations from the correct pulse positions.

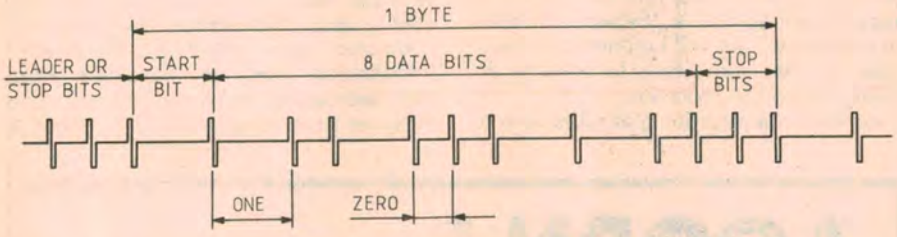

Fig. 1: typical pulse position modulation waveform showing the relationship between logical one and logical zero.

HARDWARE

The necessary interfacing hardware is simple and involves two circuits, one for writing to tape and the other for reading from tape.

The writing circuit is a pulse generator which delivers one symmetrical square wave pulse every time it is triggered. Both halves of a 74123 dual monostable are used in a cascaded arrangement. When a "negative-going" transition is applied to input pin 1 of the first mono it is triggered. When this mono recovers after a delay of 100*us* it triggers the second mono for a further delay of 100*us*. Resistor R3 is connected to the normally high output of the first mono and R4 to the normally low ouput of the second mono. At the junction of these two resistors the waveform shown on Fig. 2 is generated each time a trigger pulse is input.

If the pulse shaper were continually retriggered the output would be a continuous 5kHz square wave. While most cassette recorders will handle the fundamental of this waveform most will not do so well with the third, and higher harmonic components. Hence on replay, the waveform will be well rounded but this does not matter provided one definite pulse is returned for each pulse originally recorded.

For reading from a tape, the output from the cassette is connected to the circuit shown in Fig. 3 which uses a 555 timer IC connected as a Schmitt trigger. When the input to pin 2 goes below 1/3 Vcc the output at pin 3 goes high. It remains high until the input to pin 6 goes above 2/3Vcc when the output will change to low. Blocking capacitor C5 and resistors R6 and R7 put both inputs of the 555 at 1/2 Vcc when no signal is coming from the cassette recorder. When the volume gain of the recorder is increased until the signal swings above 2/3 and below 1/3 of Vcc the Schmitt trigger will produce a rectangular wave output at pin 3. Each time this waveform goes high the monostable formed by gates G1 and G2 generates a short duration negative going pulse. This pulse sets the RS flipflop formed by gates G3 and G4. The state of this flipflop is monitored at pin 8 by the

microcomputer through its sense input. The flipflop is cleared by the computer outputting a short negative pulse on its flag output.

From the point of view of the computer, the operation of reading consists of repeatedly looping back to check the state of the flipflop. When the flipflop goes high the computer notes the fact, then promptly resets the flipflop. After each reset the computer starts counting the number of program loops before the next pulse causes the flipflop to be set again.

When reloading it is convenient if the signal going to the computer can be heard so the loading program can be started in the leader and the recorder stopped after the file has finished. However many cassette recorders only have one outlet and when this is used the internal speaker is muted. In these cases, a small external speaker can be connected in parallel with the input to the interface of Fig. 3. If this speaker has its own volume control in the form of a variable series resistor, then a suitable listening level can be set after the output of the recorder has been set to the level required by the interface.

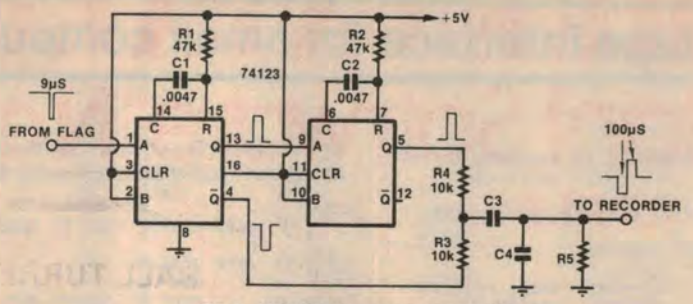

FIG. 2 : SYMMETRICAL PULSE GENERATOR

Fig. 2: the pulse generator circuitry. Both halves of a 74123 dual monostable in a cascaded arrangement are used to produce the waveform shown at the output.

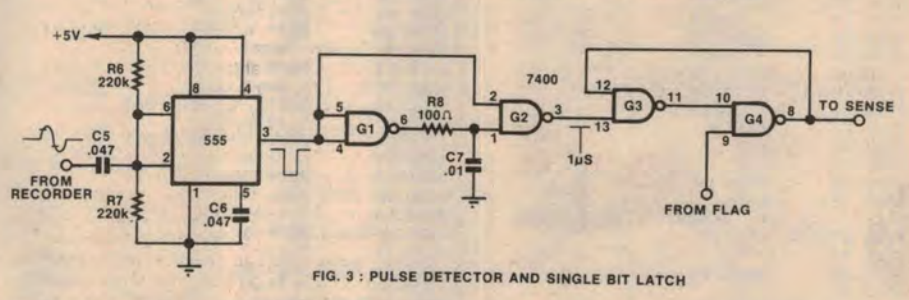

FIG. 3 : PULSE DETECTOR AND SINGLE BIT LATCH

Fig. 3: the tape reading circuitry. It uses just two ICS (a 555 timer and a 7400 quad NAND gate), three capacitors, and three resistors.

DATA RECORDING SYSTEM

How can we use the ability to measure the time between pulses to record and recover data on cassette tape? After trying a number of alternatives the following system has been adopted by the writer:

A "zero" bit is an interval of 1667 *us* and a pulse; a "one" bit is an interval of 3333 *us* and a pulse. Each data byte is recorded as a start bit (always a one), eight data bits and two stop bits (always zeros). The data bits are transmitted serially with the least significant bit (LSB) first and the most significant bit (MSB) last.

A record contains a maximum of 256 bytes, preceded by a three second leader of 1800 **zeros** and followed by a gap of at least one second duration.

In such a system, there is an arbitrary choice between whether the shorter interval will represent a **one** or a **zero.** The advantages of choosing the zero to be shorter interval are that the time for a given byte is shorter (one start, two stop bits), there are more zeros than ones on the average when the data is a program (the most used instructions have the simpler codes) and the bootstrap program is marginally shorter.

With this method of recording, the time taken by a byte will depend on the number of ones and zeros in the data. On occasions when it is necessary for all bytes to take the same time this can be done by adding an additional stop bit to the end of the byte for every zero in the data bits.

The type of bit used for the leader must be that with the shorter interval. This is so reading can be commenced anywhere in the leader and the first interval measured will always be less than the shorter interval.

The **start** bit must be of the opposite type to the leader so the beginning of a byte can be detected. The **stop** bits at the end of a byte are like a short leader between bytes and allow a little slack to pick up synchronisation at the next **start** bit. At the end of a byte the program will take momentarily longer as the byte just received is stored or otherwise disposed of.

Using a gap to mark the end of a record is very convenient with PPM as the gap can be detected while counting program loops to determine the type of bit. If the number of loops exceeds the time for a **one** by a suitable margin then this can be interpreted as an end-of-record gap. In practice it is convenient to say an end-of-record gap has occurred when the loop counter overflows and wraps around to zero.

From the above it is seen that there is a certain rationale running through the choice of how the intervals and bits are assigned for a system.

COMPUTER PROGRAMMING

Fig. 4 shows 2650 routines to save and load 256 byte blocks. The mnemonic code is written for a simple single byte assembler for which the operator characters have the following meanings:

%	assemble relative address
<	high-order byte of 15-bit address
>	low-order byte of 15-bit address
/	adjacent elements are to be ORed together
;	semi-colon and remainder of line are a comment
@	add in the indirect address bit

X1, X2, X3 are the indexing mode bits

All variable and constant identifiers commence with a letter while all numbers commence with a numeric character. If this character is a zero then the value is in hexadecimal, otherwise it is decimal. (This simple rule is definitive and saves the need to specify the number base by additional characters).

An address label must commence in the first character position of a line. Any other field must be preceded by a space.

With the computer for which the routines were first written, a WRTD (write data) or WRTC (write control) will cause the value in the designated register to be latched into the display LEDs. In addition the WRTC instruction will also cause the computer to go into the "hold" (pause) state until the "continue" switch is pressed. Thus while a block of data is being read, the number of bytes read is displayed. If all goes well, the computer will stop with the correct number of bytes showing in the display.

The program routine for writing to tape is reasonably straight forward but the reading routine needs some explanation particularly at the part where discrimination is made between ones and zeros.

Register R1 is used to count the number of program loops (each of 27*us*) before another pulse arrives. At CIL4, R1 is given an initial value of 36 decimal. If 92 loops are made the count goes to 128 which puts a **one** in the MSB of R1. This will occur after 2500*us*. Thus for intervals shorter than 2500*us* there will be a zero in the MSB and for intervals greater or equal to 2500*us* the MSB will be a one.

PPM tape interface for small computers

```
                ;
                ;RECORDING ON CASSETTE IN PPM.
                ;   J.G.LANG. 1979-1108.
                ;
                ;WRITE BLOCK Ø TO CASSETTE.
                ;
01A0 20 CIW0  EORZ/R0
01A1 93         LPSL
01A2 03         STRZ/R3   ADDR. POINTER
01A3 3B         BSTR/UN   LEADER
01A4 0F           %LDR
01A5 0F CW2     LODA/R3   NEXT BYTE
01A6 60           X3/<BLK0
01A7 00              >BLK0
01A8 3B         BSTR/UN
01A9 16           %WBYT
01AA DB         BIRR/R3
01AB 79           %CW2
01AC B3         WRTC/R3   HOLD
01AD 1B         BCTR/UN   DO AGAIN
01AE 71           %CIW0
01AF C0         NOP
01B0 C0         NOP
01B1 C0         NOP
01B2 C0         NOP
01B3 C0         NOP
                ;3 SEC. LEADER OF ZERO BITS.
01B4 05 LDR     LODI/R1
01B5 07            7
01B6 06         LODI/R2
01B7 00            0
01B8 3B LDR2    BSTR/UN
01B9 1C           %ZERO
01BA FA         BDRR/R2
01BB 7C           %LDR2
01BC F9         BDRR/R1
01BD 7A           %LDR2
01BE 17         RETC/UN
01BF C0         NOP
```

Fig. 4: the 2650 software routines used to save and load 256 byte blocks. The program has to be loaded by hand every time the microcomputer is powered up.

```
                ;WRITE BYTE (R0) TO CASSETTE.
                ;
01C0 C2 WBYT    STRZ/R2
01C1 C0         NOP
01C2 3B         BSTR/UN   START BIT
01C3 0E           %ONE
01C4 05         LODI/R1   BIT CNTR
01C5 08            8
01C6 52 WB4     RRR/R2    NEXT BIT
01C7 BA         BSFR/N
01C8 0D           %ZERO
01C9 3A         BSTR/N
01CA 07           %ONE
01CB F9         BDRR/R1
01CC 79           %WB4
01CD 3B         BSTR/UN   STOP BIT
01CE 07           %ZERO
01CF 3B         BSTR/UN   STOP BIT
01D0 05           %ZERO
01D1 17         RETC/UN
01D2 04 ONE     LODI/R0
01D3 B4            180
01D4 F8 LN7     BDRR/R0
01D5 7E           %LN7
01D6 04 ZERO    LODI/R0
01D7 B4            180
01D8 F8 LN8     BDRR/R0
01D9 7E           %LN8
01DA 74         CPSU      PULSE
01DB 40           FLAG
01DC 76         PPSU
01DD 40           FLAG
01DE 20         EORZ/R0   MAKE POSITIVE
01DF 17         RETC/UN
```

```
                ;LOAD BLOCK Ø FROM CASSETTE.
                ;
01E0 77 CIL0    PPSL
01E1 08           WC
01E2 07         LODI/R3   POINTER
01E3 FF           -1
01E4 20 CIL2    EORZ/R0   NEXT BYTE
01E5 05 CIL4    LODI/R1   NEXT BIT
01E6 24           36
01E7 74         CPSU      PULSE TO
01E8 40           FLAG    CLEAR LATCH
01E9 76         PPSU
01EA 40           FLAG
01EB B4 CIL6    TPSU      COUNT LOOPS TILL
01EC 80           SENS    NEXT PULSE SETS
01ED 98         BCFR/Z    LATCH
01EE 0B           %CIL8
01EF D1 CIL7    RRL/R1    HIGH BIT TO CARRY
01F0 50         RRR/R0    HENCE TO BYTE
01F1 52         RRR/R2
01F2 9A         BCFR/N    START + 8 BITS?
01F3 71           %CIL4   NOT YET
01F4 CF         STRA/R3   STORE THE BYTE
01F5 20           X1<BLK0
01F6 00              >BLK0
01F7 F3         WRTD/R3   DISPLAY BYTE COUNT
01F8 1B         BCTR/UN
01F9 6A           %CIL2
01FA D9 CIL8    BIRR/R1   BUMP LOOP COUNT
01FB 6F           %CIL6   LOOP IF NOT ZERO
01FC B3         WRTC/R3   HOLD
01FD 1F         BCTA/UN   GOTO BCC CHECK
01FE 01           <CHK0
01FF 80           >CHK0
```

At CIL7 this bit is first rotated into the Carry, then into the MSB of Register R0 in which the data byte is being built up. As each new bit is received the pattern of bits in R0 is pushed right until the start bit (a **one**) is pushed out into the Carry. This is tested by seeing if R2 goes negative when the Carry bit is rotated into it. When the start bit has been pushed right through and out of R0 the remaining pattern is the required byte of data.

If the interval between pulses exceeds 5940*us* then the count will overflow and "wrap around" to zero which is interpreted as an end-of-record gap. This is tested at CIL8 as the loop counter is incremented.

ALLOCATION OF MEMORY

With 512 bytes of RAM it is convenient to think of these as two blocks, Block 0 and Block 1, each of 256 bytes. To avoid constantly getting tangled up or having to make unnecessary transfers, Block 0 (0000-00FF) is commonly used for the current user program and the first quarter of Block 1 (0100-013F) for the associated scratchpad or working area.

The remainder of Block 1 (0140-01FF) is used for utility routines for transferring and checking data.

After the system is powered up, the console switches are used to enter an elemental loader at the end of Block 0, and a Branch Absolute instruction at location 0000. The elemental loader for loading Block 1 differs for the loader of Block 0 shown in Fig. 4 only at the absolute address, ie, 00F5 becomes 21 hex and 00FD, 00FE and 00FF become C0 hex (no operation). When this elemental loader is executed a 256-byte block of utility routines is read from cassette and stored in Block 1.

The CIL0 routine copies a block into Block 0. CIW 0 will write Block 0 out onto tape. The routine DAD moves progressively through a designated part of memory displaying the address and data (contents) of each location in turn. This is done by displaying first the high order byte of the address, then the low order byte, then the contents. Before each byte is displayed, its complement is flashed momentarily so that repeated patterns will be noticed, and the contents are displayed for three times as long as the address bytes so that they can be differentiated.

The load from cassette routines do not include any parity or redundancy checks as it is desirable that the elemental loader be as short as possible. Checking against transfer errors is done immediately after loading by running the routine CBCC which calculates a Block Control Character (by exclusively ORing, then rotating left) on a 256-byte block. Two versions of the routine are included, one for checking Block 0 and the other for Block 1.

As each routine is completed it enters the HOLD state. These routines are linked so that a CONTINUE will cause a jump to the next commonly used routine.

In conclusion it has been shown how, by using the techniques described above, and with just a minimum of hardware, quite a useful little microcomputer operating system can be built up which will considerably extend the usefulness of any console switch-based microcomputer.

IF YOU PROGRAM MICROPROCESSORS, THESE SHEETS ARE FOR YOU

Writing microprocessor programs on blank or plain ruled paper is messy and time consuming. To make the job easier, Electronics Australia has produced custom-designed programming sheets for use with virtually any microprocessor system. The sheets feature columns for a 4-digit address, up to six digits of instruction code or data, labels, mnemonics, and comments. There is space for 46 lines per sheet, and the sheets are provided with space for program title, date and sheet number. All for less than 5 cents per sheet, posted!

PRICED AT $2.00 FOR 50 SHEETS
(plus 40c for postage anywhere in Australia)

Available only from Electronics Australia,
57 Regent St, Sydney (P.O. Box 163, Beaconsfield, NSW 2014)