Hyper about Slow Load Times?

. . . KIM Hypertape is an alternative

Jim Butterfield
14 Brooklyn Ave.
Toronto Ontario M4M 2X5

Whenever I meet a bunch of KIM users in my travels, I'm likely to notice a couple of guys off to one side whispering and gesturing in my direction. This means either of two things: I'm wearing odd socks (again), or I'm being identified as the creator of KIM Hypertape.

Hypertape, often called super tape by KIM users, is indeed a good thing. It speeds up the standard KIM-1 cassette tape interface by a factor of six times. This gives a speed of roughly 50 bytes per second loading or dumping. It saves time and tape. And it's completely compatible with the KIM cassette tape loader — no extra hardware or software is needed to read Hypertape.

But I must confess: I didn't do it alone. I didn't even plan to write Hypertape; it sort of happened. It's not that I mind the fame. It's kind of nice getting fan letters and acknowledgements in other people's programs. And I have no objection to nubile nymphs strewing rose petals in my path, either, although I haven't had too many of those yet.

Now, it's time to own up. I wasn't a man with a vision struggling against innumerable setbacks. I fell into it on my way to something else. It's like the story of Thomas Edison picking up the world's first light bulb, admiring it, and then bringing it to his lips and hollering, "Hello? Hello?"

It all started last fall, when I was having lunch with Julien Dubé, a friend and fellow KIM owner, and Rick Simpson, then manager of KIM-1 Product Support for MOS Technology, Inc. Rick was talking about the cassette interface. "Maybe we should have made it faster," he mused. "It could be speeded up by a factor of three, but . . ." At that moment the chopped chicken livers arrived, and the sentence was never finished. But the phrase had caught my imagination. A speedup of three times! Wow! But how would it be done? To solve the mystery, I would have to look into the workings of the KIM cassette load/dump programs.

Recording Basics

The KIM User Manual describes the cassette recording principles quite clearly. The system uses frequency shift keying (FSK). The two tones used are at frequencies of 3700 Hertz and 2400 Hertz respectively. During a dump to cassette, the tones are generated directly from the microprocessor as square waves — no oscillators are involved. In reading back from cassette, the signal is fed to an LM565 phase lock loop used as frequency discriminator (see Fig. 1). Everything else is done in software — timing, assembling of characters, storage of data and checksum. Handy to know, but not enough.

The next step was to dig into the software. How are the bits represented on tape? Still not hard to find; KIM is well documented. The so-called 2/1 scheme is used: To record a logic zero, send 3700 Hertz for five milliseconds duration followed by 2400 Hz for 2.5 ms. To record a logic one, send 3700 Hz for 2.5 ms followed by 2400 Hz for 5 ms (see Fig. 2). Either way, it's about 7.5 ms per bit, right? And each sequence commences with the higher frequency.

Now we're getting somewhere. The next step is to look at the tape load monitor program and see how it gets those bits back off the tape. Aha! Here's what KIM does: It compares the timing of the two parts, 3700 Hz versus 2400 Hz. If the 3700 Hz signal lasts longer, the bit must be zero; if the 2400 Hz is the long one, then the bit is logic one.

Now, pay attention, we're almost there. If the KIM loader doesn't care about the actual timing, but just wants to know which frequency lasts longer . . . we can speed the whole thing up! As long as we keep the right timing ratio between the two frequencies, the KIM monitor won't worry whether it's fast or slow. Since we're dealing with input and output at the bit level, we don't need to

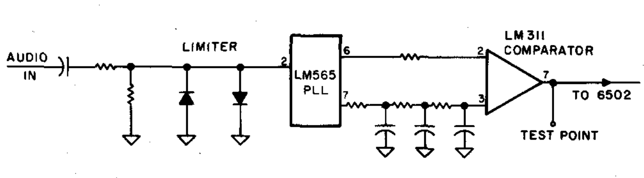

Fig. 1. KIM-1 audio tape input circuit.

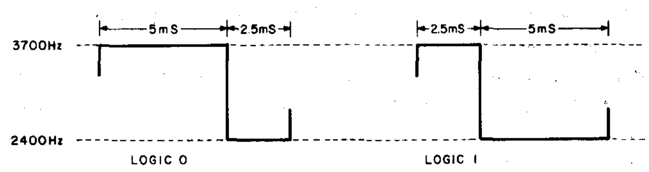

Fig. 2. Timing of normal KIM tape signals.

```
; OUTPUT 3700 HZ TO TAPE
; 9 PULSES 138 USEC EACH
;
ONE     LDX #9          N = 9 pulses
        PHA             Save A in stack
ONE1    BIT CLKRDI      Check timer
        BPL ONE1        Wait for timeout
        LDA #126        Next timing . .
        STA CLK1T        . . into timer
        LDA #$A7        Bit 7 on
        STA SBD          . . to output
ONE2    BIT CLKRDI      Wait for timeout
        BPL ONE2
        LDA #126        Next timing . .
        STA CLK1T        . . into timer
        LDA #$27        Bit 7 off
        STA SBD          . . to output
        DEX             one less cycle
        BNE ONE1        go back if more
        PLA             bring A back
        RTS             return from subroutine
```

```
; OUTPUT 2400 HZ TO TAPE
; 6 PULSES 207 USEC EACH
;
ZRO     LDX #6          N = 6 pulses
        PHA
ZRO1    BIT CLKRDI
        BPL ZRO1
        LDA #195
        STA CLK1T
        LDA #$A7
        STA SBD
ZRO2    BIT CLKRDI
        BPL ZRO2
        LDA #195
        STA CLK1T
        LDA #$27
        STA SBD
        DEX
        BNE ZRO1
        PLA
        RTS
```

Program A. Original KIM routines for sending the two frequencies to audio output. Note that the labels ONE and ZRO do not *indicate that we are sending logical one or zero from memory.*

```
; OUTPUT FREQ TO TAPE
; Y REGISTER SAYS WHICH FREQUENCY
;
ZON     LDX NPUL,Y      get N from table
        PHA             save A
ZON1    BIT CLKRDI      Check timer
        BPL ZON1        Wait for timeout
        LDA TIMG,Y      get timing from table
        STA CLK1T        . . into timer
        LDA #$A7        Bit 7 on
        STA SBD          . . to output
ZON2    BIT CLKRDI      Check timer
        BPL ZON2        Wait for timeout
        LDA TIMG,Y      timing from table
        STA CLK1T        . . into timer
        LDA #$27        Bit 7 off
        STA SBD          . . to output
        DEX             one less cycle
        BNE ZON1        go back if more
        PLA             bring A back
        RTS             return from subroutine
```

Program B. I've combined the two original KIM routines using the Y index register to indicate which frequency is involved. Note that I've done nothing new — just saved memory and punch-up time. But wait, ZON1 and ZON2 look very similar. Can I save even more?

```
; OUTPUT FREQ TO TAPE
; Y REGISTER SAYS WHICH FREQUENCY
; LOCATION GANG STARTS AT $27 HEX
;
ZON     LDX NPUL,Y      get half-pulses from table
        PHA             save A
ZON1    BIT CLKRDI      check timer
        BPL ZON1        wait for timeout
        LDA TIMG,Y      get timing from table
        STA CLK1T        . . put in timer
        LDA GANG        $27 or $A7
        EOR #$80        flip it over
        STA SBD         output it
        STA GANG        and save new GANG value
        DEX             one less half-cycle
        BNE ZON1        go back if more
        PLA             bring A back
        RTS             end of subroutine
```

Program C. Further consolidation of the coding. We're now counting half-cycles instead of cycles, and this turns out to be a big help.

worry about details of tape formats: special characters, checksums and items like that. They will all eventually be sent as bits — and it's the format of the bits we're dealing with. To telescope those bit signals, we must return to the audio dump program. So long as we write them properly onto tape, we know that the load program will track them correctly.

Counting Cycles

A little arithmetic, or failing that, a look at the KIM manual, shows that logic zero consists of 18 pulses at the higher frequency followed by six pulses at the lower. For logic one the numbers become nine and 12 (see Fig. 3). As I noted these numbers, the words kept echoing through my head, ". . . factor of three . . ." Suddenly the penny dropped. All the above pulses are multiples of three — so you can reduce the number of cycles to 2/3 or to 1/3 without getting into fractions. KIM sends all its cycles direct from software. So all you'd need to do is to change the loop counters and . . . hmmm, it might work. Of course, Rick's phrase was, ". . . factor of three, but . . ." But what? Would the phase lock loop be too sluggish to take the speed increase?

Did all these exciting discoveries send me rushing to the coding sheets to see if I could produce triple-speed tape? Nope. Next day, I was chatting to Julien Dubé again. "Funny thing," I said in my wise and knowing way. "I think there's a way to increase tape speed by at least 50 percent — it might even go up to triple speed." As usual, Julien made a good audience as I outlined my detective work of the previous evening. "What's more," I concluded triumphantly, "you'd hardly need to write it. Just copy out the ROM programs to RAM, change the pulse counters, and you have it!"

I thought no more of the conversation until late that evening when Julien called me. "Works fine," he reported. "Good at triple speed, too. Why did you suspect there might be a problem?" I found this disconcerting. Not only had Julien been listening to me earlier in the day, he'd gone right ahead and done it. I collected my thoughts.

"OK," I said, "The problem is that the phase lock loop is likely coming to the limit of its tracking capability. Put a meter on the output of the LM311 comparator. Normally it's 2.5 V, but at high speed it will start to pull because of bias distortion."

Julien called back very quickly. "It's solid on 2.5 V

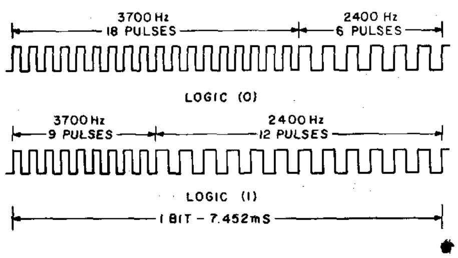

Fig. 3. KIM standard audio signals.

at the highest speed," he reported. "Loads without error, too. Funny thing — I've listened to the tape itself and it sounds totally different than ordinary KIM tape."

On to the Coding Sheets

I think it was that last comment that got me. How can it sound different when it's the same two frequencies? Besides, the phase lock loop behavior intrigued me: How could it track on only two cycles? How much further could it go? At this stage, triple speed, we were sending a minimum of only two pulses at one frequency and three at the other. Could I speed that up without getting into fractions? I couldn't see how. Can you send half a pulse? That sounds like the paradox of one hand clapping. I was unable to see it.

Let's pick over the original KIM monitor coding for sending the two frequencies. It's shown in Program A. Well written, but since we need to change it anyway, let's see what we can do with it. Subroutines ONE and ZERO are almost identical. They differ in only two items: nine cycles versus six cycles, and 126 microseconds of delay in the timer versus 195 microseconds. Automatic programming reflex number one: Consolidate them and put the two variables in a table.

Assuming we have that squared away (see Program B), there's another piece of duplicate coding: The sequences at ONE1 and ONE2 (and their counterparts in ZRO) are almost the same. This time, the difference is in hexadecimal 27 versus A7. These values are sent to the output register to make bit 7 (the tape output) go on and off, generating the square wave that we record on tape. Automatic programming reflex number two: when you have a bit going back and forth like that, use an EOR (Exclusive OR) instruction to flip it over and back.

That last part is more than just efficient coding; it has important consequences for us to follow through. Previously, we generated a square wave by having a piece of program to turn the bit on, followed by a piece of program to turn it off again. That makes one full cycle of the square wave. But if we go the EOR route (see Program C), we'll flip the bit over and generate one-half of the square wave. That's what we've been looking for: a way to generate half a pulse. We've opened the door to sixfold speedup.

Now all the pieces have come together, and the coding comes easily. We have the number of half-cycles for each frequency in a table, so we can easily adjust the program for other speeds. At maximum — Hypertape — speeds, we'll be sending as little as one pulse at the lower frequency and 1.5 pulses at the higher frequency (Fig. 4). Can the phase lock loop track it? You bet it can — and the 2.5 V test point stays steady as a rock.

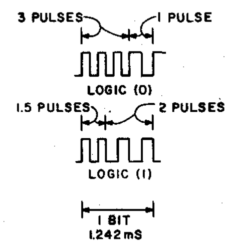

Fig. 4. Hypertape audio signals.

Wrapping It Up

The test runs were a bit eerie. Even when you do the arithmetic, it doesn't seem right for a 30-second program to load in five seconds. At first, it all happened so quickly that I was sure there was something wrong. But it checked out OK, and Hypertape became a reality.

Tests of various tape recorders revealed that a few of them won't carry Hypertape, apparently because their frequency response is too poor to carry the high sidebands of the signal. A related problem occurs in exchanging tapes from one cassette unit to another: Slight head misalignment causes those vital high frequencies to be lost. It's a good practice for KIM tape swappers to drop their speed to a paltry three times normal to eliminate this potential problem. Of course, the documented and tidied up program (Program D) was

Program D. The final, polished, complete version for reading and writing data in the Hypertape format.

```
                    ;hypertape writer starts here
0100 A9 AD          DUMP      LDA #$AD        LDA command
0102 8D EC 17                 STA VEB
0105 20 32 19                 JSR INTVEB      set up sub
0108 A9 27                    LDA #$27
010A 85 F5                    STA GANG        flop flag for SBD
010C A9 BF                    LDA #$BF
010E 8D 43 17                 STA PBDD        directnl registr
0111 A2 64                    LDX #$64        send 100 . .
0113 A9 16                    LDA #$16        . . SYNC chars
0115 20 61 01                 JSR HIC
0118 A9 2A                    LDA #$2A        send START (*)
011A 20 88 01                 JSR OUTCHT
011D AD F9 17                 LDA ID          send pgm ID
0120 20 70 01                 JSR OUTBT
0123 AD F5 A7                 LDA SAL         & start addrs
0126 20 6D 01                 JSR OUTBTC
0129 AD F6 17                 LDA SAH
012C 20 6D 01                 JSR OUTBTC
012F 20 EC 17       DUMPT4    JSR VEB
0132 20 6D 01                 JSR OUTBTC      send byte . .
0135 20 EA 19                 JSR INCVEB      move to next . .
0138 AD ED 17                 LDA VEB+1
013B CD F7 17                 CMP EAL         is it last byte?
013E AD EE 17                 LDA VEB+2
0141 ED F8 17                 SBC EAH
0144 90 E9                    BCC DUMPT4      no, repeat
0146 A9 2F                    LDA #$2F        yes, send . .
0148 20 88 01                 JSR OUTCHT      . . END (/)
014B AD E7 17                 LDA CHKL
014E 20 70 01                 JSR OUTBT
0151 AD E8 17                 LDA CHKH        checksum
0154 20 70 01       EXIT      JSR OUTBT
0157 A2 02                    LDX #$02        send two . .
0159 A9 04                    LDA #$04        EOT characters
015B 20 61 01                 JSR HIC
015E 4C 5C 18                 JMP DISPZ       & we're done
                    ;subroutines
0161 86 F1          HIC       STX TIC
0163 48             HIC1      PHA
0164 20 88 01                 JSR OUTCHT      send character
0167 68                       PLA             . . bring it back
0168 C6 F1                    DEC TIC
016A D0 F7                    BNE HIC1        repeat as needed
016C 60                       RTS
016D 20 4C 19       OUTBTC    JSR CHKT        compute checksum
0170 48             OUTBT     PHA             save the character
0171 4A                       LSR A
0172 4A                       LSR A           . . and take its
```

fired off to the KIM User Notes for more extensive field testing. Acknowledgement was given to Julien Dube' for his help.

As you can see, Hypertape's speed came from putting the bits more compactly onto tape. There are still other areas where the signal can be made more efficient. For example, each byte of storage is translated into two hexadecimal characters. That's a waste of two-to-one, since 16 bits are used to store eight. Then there's the question of the 2/1 coding scheme; that uses three bit-times to store each bit. And of course, we haven't touched on the question of data compression. There are still worlds to conquer. But I think I'll take it easy for a while. After all, there's something to be said for full compatibility with the KIM monitor. Then again, if Julien isn't doing anything next month . . . . ■

```
0173 4A                       LSR A           four left bits . .
0174 4A                       LSR A
0175 20 7D 01                 JSR HEXOUT      write 'em
0178 68                       PLA             now the 4 right bits
0179 20 7D 01                 JSR HEXOUT
017C 60                       RTS
                 ;
017D 29 0F       HEXOUT       AND #$0F        remove unwanted bits
017F C9 0A                    CMP #$0A        convert to ASCII
0181 18                       CLC             . . by adding:
0182 30 02                    BMI HEX1
0184 69 07                    ADC #$07        $37 if A to F;
0186 69 30       HEX1         ADC #$30        $30 if numeric.
0188 A0 07       OUTCHT       LDY #$07        For the 8 bits:
018A 84 F2                    STY COUNT
018C A0 02       TRY          LDY #$02        send 3 units
018E 84 F3                    STY TRIB        starting at 3700 Hz
0190 BE BE 01    ZON          LDX NPUL,Y      # of half cycles
0193 48                       PHA
0194 2C 47 17    ZON1         BIT CLKRDI      Wait for previous . .
0197 10 FB                    BPL ZON1        cycle to complete
0199 B9 BF 01                 LDA TIMG,Y      Get timing to the . .
019C 8D 44 17                 STA CLK1T       next pulse (7E or C3)
019F A5 F5                    LDA GANG
01A1 49 80                    EOR #$80        Flip between 1 & 0
01A3 8D 42 17                 STA SBD
01A6 85 F5                    STA GANG
01A8 CA                       DEX             Sent all cycles?
01A9 D0 E9                    BNE ZON1        no, go back
01AB 68                       PLA             yes, recall char
01AC C6 F3                    DEC TRIB        one less to send
01AE F0 05                    BEQ SETZ        branch if last one
01B0 30 07                    BMI ROUT        branch if no more
01B2 4A                       LSR A           Take next bit
01B3 90 DB                    BCC ZON         . . if it's a one . .
01B5 A0 00       SETZ         LDY #0          Switch to 2400 Hz
01B7 F0 D7                    BEQ ZON         unconditional return
01B9 C6 F2       ROUT         DEC COUNT       one less bit
01BB 10 CF                    BPL TRY         any more? go back
01BD 60                       RTS
                 ;frequency/density controls
01BE 02          NPUL         .BYTE $02       Two pulses! One cycle!
01BF C3 03 7E    TIMG         .BYTE $C3,$03,$7E
```