Forth computer

Exceeding Brian Woodroffe's earlier expectations, his 6809-based Forth computer can be used with disc drives requiring high data-transfer rates – including Sony's microdrive and 8in floppy-disc drives. Access times for standard drives can also be reduced using a minor hardware modification.

by B. Woodroffe

After using the computer for a while I became discontented with the disc system because the software caused a one second delay each time access to the disc was required. This waiting time is needed to allow the drive to reach its operating speed – the software doesn't know whether the disc is running or stopped before access – and much of the benefit of the virtual memory system is lost because of this delay. Forth keeps data from a number of disc sectors in memory. When data from a sector that is not in the main memory is requested by the program, Forth overwrites one of the buffers with the required data. But if data in the buffer has been changed Forth first sends the data back to the disc so there can be two 1s delays for one disc call.

Keeping the disc constantly rotating is the easiest way of avoiding this delay but this was rejected because it shortens the life of the drive even though the motor is a brushless d.c. type. The method chosen relies on the fact that disc access operations are not uniform in time i.e., they are likely to come in bursts, especially when loading or listing screens from a disc and as a result of the virtual-memory buffer replacement algorithm described above. Keeping the drive running for a short while after a disc access is made means that it is likely that the disc will be running when the next disc access is required.

Normally, the disc-drive motor is turned off by the disc-select signal going false (p.i.a. A port, D_6, 0 to 1) when the program finishes using the drive. In the modification the drive motor enable signal is held true for five seconds after the drive-select signal goes false by a monostable multivibrator triggered by the trailing edge of the p.i.a. signal. As the software always assumes that the drive is up to speed and available, even though the monostable i.c. might have completed its cycle, a means of ensuring that the WD1793 controller doesn't try to access the drive during the motor start period is required. During this period the ready signal is held false by a further monostable multivibrator fired by the drive-motor start signal. A low-pass filter after the NOR gate combines the two sources of motor-on signal to allow for the set-up time of the 5s monostable.

This small hardware modification, consisting of two s.s.i. devices relieves the software of all considerations of motor-start latency. To prevent erroneous triggering the two monostable multivibrators should be grounded separately.

Brian Woodroffe works in research and development at Hewlett Packard.

Interfacing 8in drives

I found it galling that my 1.5MHz 6809 Forth system could not be interfaced with faster 8in drives, especially as these are often available second-hand at bargain prices. I have not yet got an 8in drive but I have been fortunate enough to try one of the sub-5in drives from Sony which has the same data rate as 8in drives. There is as yet no *de jure* standard for microfloppy-disc drives[1] but within Hewlett Packard, the Sony drive is the *de facto* standard. The first problem is to build a data-service routine that services the disc at a rate better than 11μs/byte. Although nominal disc-data transfer normally takes 16μs/byte, Western Digital specify 11 and 13μs worst-case service times for write and read respectively.

The previously used software loop (*Wireless World*, June 1983) achieves far worse than 11μs, even with the M6809 direct-page register modified to make the controller i.c. accessible through direct addressing. Analysing the software loop shows that two functions are being carried out – a byte is transferred between the WD1793 controller and ram, and bytes are counted to determine when the sector operation is completed. If the second function could be dispensed with the remaining loop would be much smaller and faster. There would be a small penalty in that the ability to read from and write to consecutive sectors would be lost as no byte count is kept. The problem now is how to break out of the disc-service software loop. Fortunately the controller gives hardware help here in that the IRQ line is activated on completion of every command; the DRQ line is activated for each byte transfer. DRQ, connected to the M6809 FIRQ line, is used in the data transfer loop to make the processor clear its SYNC state, thus synchronizing controller/processor transfers operations.

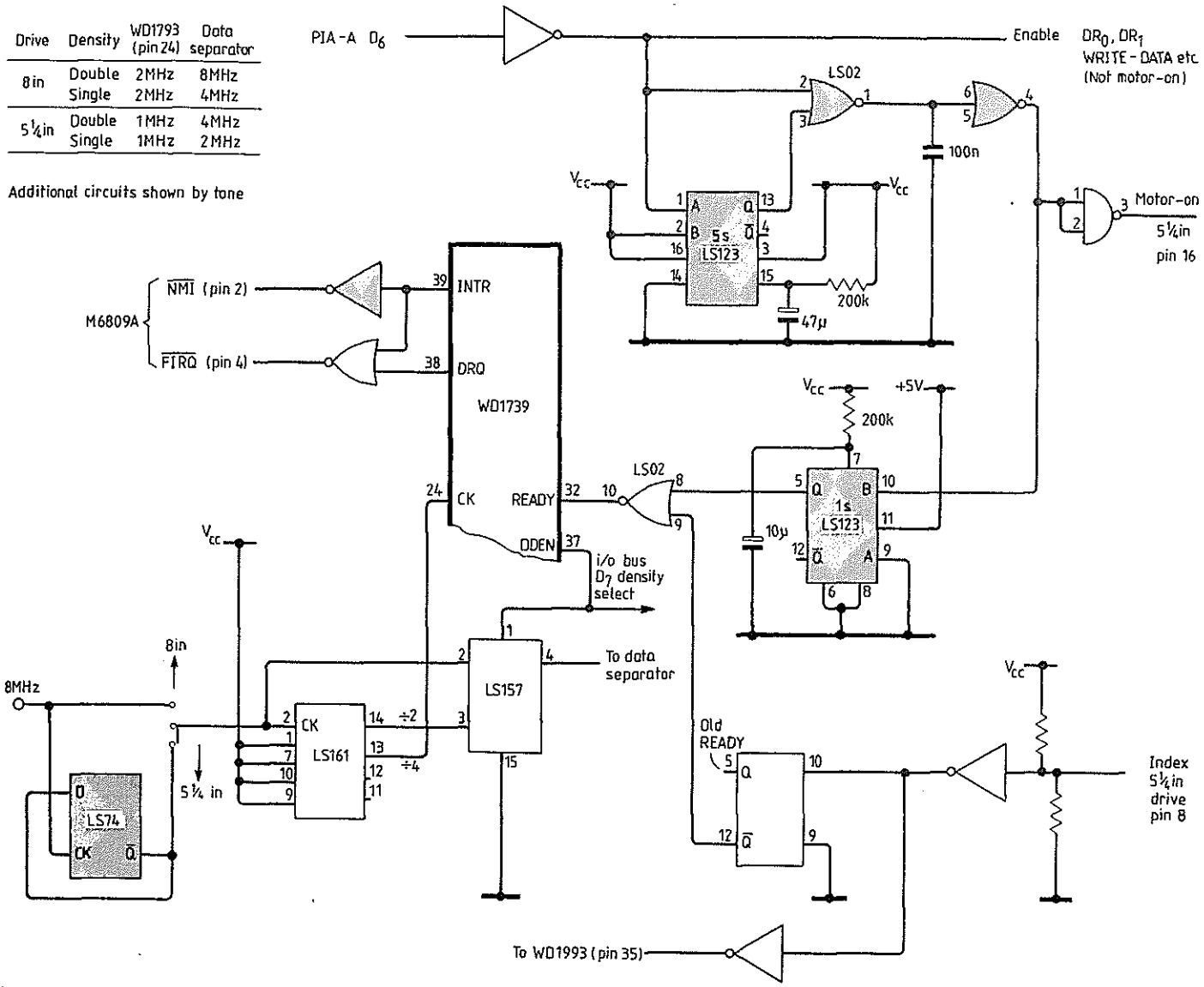
Interrupt request IRQ is tied to one of the other M6809 interrupt lines so that when a read or write-sector command is complete the processor aborts its current data-transfer loop activity and commences the interrupt routine. On application of the FIRQ signal the processor does not abort the data transfer loop and carry out the FIRQ routine because the program inhibits the FIRQ interrupt by holding the FIRQ mask bit in the condition-code re-

List 1. In this design the following Forth words are available.

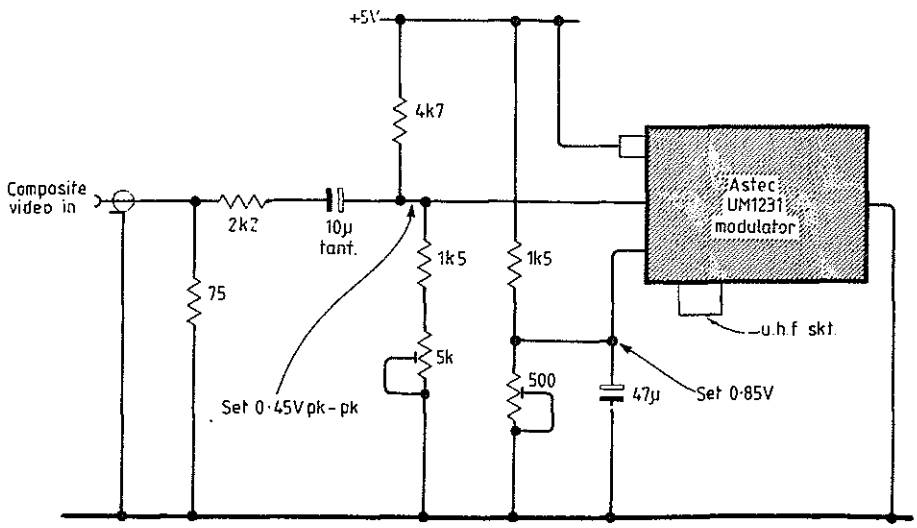
LIT	EXECUTE	BRANCH	0BRANCH	(LOOP)	(+LOOP)
(DO)	I	J	K	DIGIT	(FIND)
ENCLOSE	EMIT	KEY	?TERMINAL	CR	CMOVE
U<	U*	U/	AND	OR	XOR
SP@	SP!	RP!	;S	+	D+
MINUS	DMINUS	1+	2+	1-	2-
M*	*	M/	/MOD	/	MOD
*/MOD	*/	M/MOD	ABS	DABS	S->D
+-	D+-	0=	0<	LEAVE	>R
R>	R	OVER	DROP	SWAP	DUP
+!	@	C@	!	C!	TOGGLE
<BUILDS	DOES>	:	;	CONSTANT	VARIABLE
0	1	2	3	BL	C/L
FIRST	LIMIT	USER	+ORIGIN	S0	R0
TIB	WIDTH	WARNING	FENCE	DP	VOC-LINK
BLK	IN	OUT	SCR	OFFSET	CONTEXT
CURRENT	STATE	BASE	DPL	FLD	CSP
R#	HLD	COLUMNS	HERE	ALLOT	
,	C,	-	=	<	
>	SPACE	MIN	MAX	-DUP	
TRAVERSE	LATEST	LFA	CFA	NFA	PFA
!CSP	?ERROR	?COMP	?EXEC	?PAIRS	?CSP
?LOADING	COMPILE	[	]	SMUDGE	HEX
DECIMAL	(;CODE)	COUNT	TYPE	-TRAILING	"
(."	(.")	."	DEPTH	?STACK	?FREE
#0!	EXPECT	QUERY	X	FILL	ERASE
BLANKS	HOLD	PAD	WORD	(NUMBER)	NUMBER
FIND	(ABORT)	ERROR	ID.	CREATE	[COMPILE]
LITERAL	DLITERAL	INTERPRET	IMMEDIATE	VOCABULARY	DEFINITIONS
:	QUIT	ABORT	COLD	WARM	#DR
USE	PREV	+BUF	(LINE)	.LINE	MESSAGE
LOAD	-->		FORGET	BACK	BEGIN
ENDIF	THEN	DO	LOOP	+LOOP	UNTIL
END	AGAIN	REPEAT	IF	ELSE	WHILE
SPACES	<#	#>	SIGN	#	D.R
.R	D.	.	?	LIST	INDEX
DUMP	VLIST	R/BUF	R/SCR	SCR>BLK	USEBLK
DR0	DR1		DRIVE	R/W	;CODE
UPDATE	EMPTY-BUFFERS	BLOCK	FLUSH	FORTH	TASK
NOOP					

Drive	Density	WD1793 (pin 24)	Data separator
8in	Double	2MHz	8MHz
	Single	2MHz	4MHz
5¼in	Double	1MHz	4MHz
	Single	1MHz	2MHz

Additional circuits shown by tone



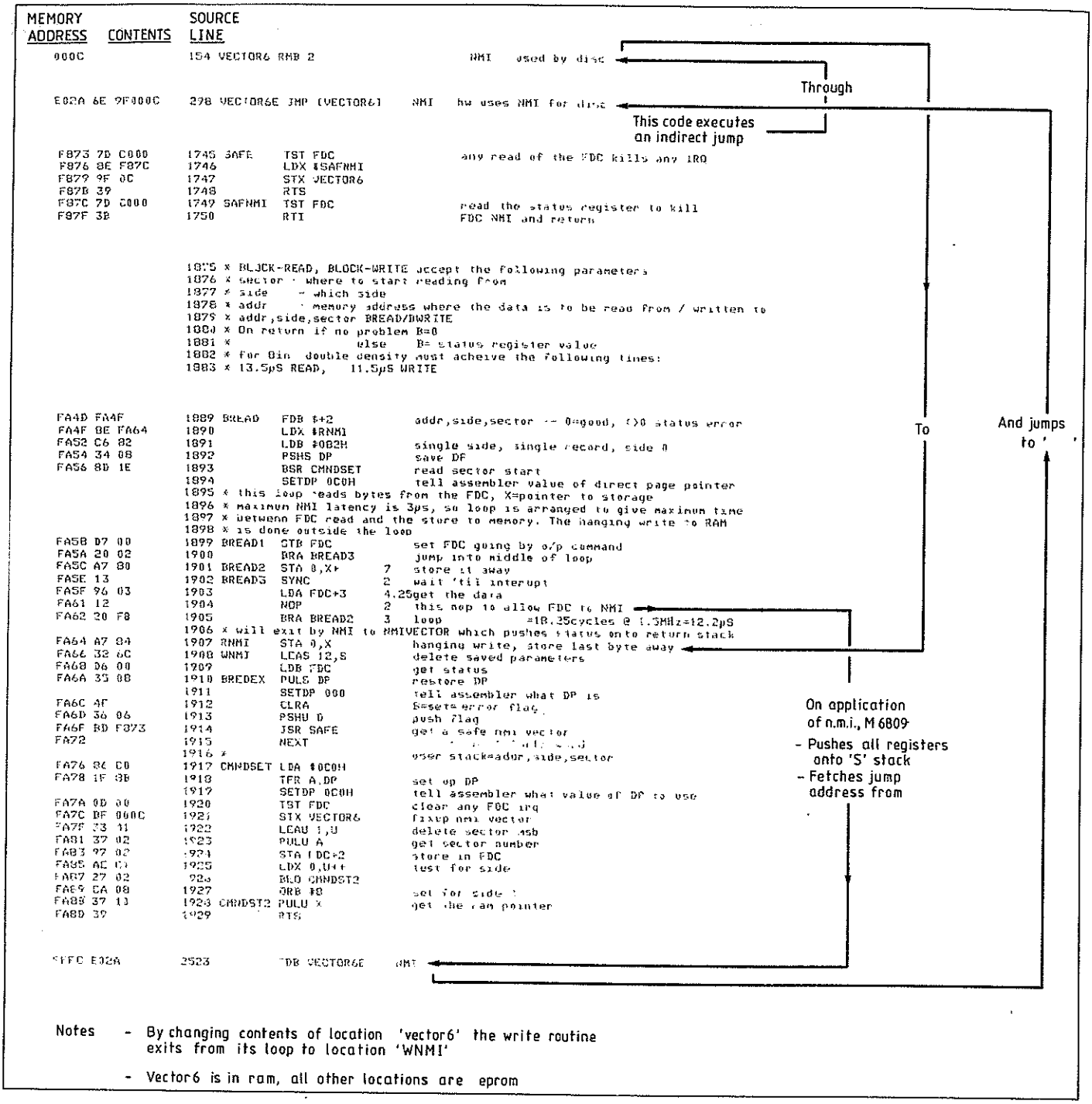
▲ *Disc interface modification speeds up overall access time by keeping the drive motor running for five seconds after the computer tells it to switch off. This significantly reduces the effect of a one second delay required for the motor to start up since disc-access operations tend to come in bursts.*



▲ *U.h.f. modulator connects to the video-controller circuit output (see June issue) so that the computer can be used with a standard tv set.*

gister set. The controller interrupt-request line IRQ, being connected to the processor's non-maskable interrupt input NMI, can never be masked so when this line is true the processor must be interrupted and goes to the routine requested by IRQ. The processor IRQ interrupt-request input could have been used but I wanted to leave it free for expansion.

For sector read and write operations the disc controller interrupts on transfer of the last byte. In the case of a sector-write operation the data-transfer routine is finished when the last byte is written into the controller. In sector read operations the data transfer routine is finished not when the last byte is read from the controller, but when it is written into the ram sector buffer. So when a sector is written the controller interrupts after all data transfers have taken place but when sectors are read the controller causes a jump out of the software loop before the last memory-storage operation is carried out. Worse still, latency before the controller interrupt is variable so when an interruption is made, whether or not the memory has been updated remains in doubt. The solution I chose was to code the data-transfer loop so as to maximize the time between reading data from the controller and writing it into the memory. Inclusion of a no-operation, NOP, increases the data-transfer loop time to just under the allowable maximum of 13µs to ensure that the controller always interrupts before the processor can write the byte into memory; the first operation of the interrupt routine is to write that byte into memory. Unfortunately this writing operation done outside the data-transfer loop means that the interrupt routines for sector reading and writing must be different.




```
MEMORY          SOURCE
ADDRESS  CONTENTS  LINE
  000C             154 VECTOR6 RMB 2                   NMI   used by disc

  E02A 6E 9F000C   298 VECTOR6E JMP [VECTOR6]    NMI   hw uses NMI for disc

  F873 7D C000     1745 SAFE    TST FDC                 any read of the FDC kills any IRQ
  F876 8E F87C     1746         LDX #SAFNMI
  F879 9F 0C       1747         STX VECTOR6
  F87B 39          1748         RTS
  F87C 7D C000     1749 SAFNMI  TST FDC                 read the status register to kill
  F87F 3B          1750         RTI                     FDC NMI and return


                   1875 * BLOCK-READ, BLOCK-WRITE accept the following parameters
                   1876 * sector - where to start reading from
                   1877 * side   - which side
                   1878 * addr   - memory address where the data is to be read from / written to
                   1879 * addr,side,sector BREAD/BWRITE
                   1880 * On return if no problem B=0
                   1881 *                 else    B= status register value
                   1882 * For 8in  double density must acheive the following times:
                   1883 * 13.5µS READ,   11.5µS WRITE


  FA4D FA4F        1889 BREAD   FDB $+2               addr,side,sector -- 0=good, <>0 status error
  FA4F 8E FA64     1890         LDX #RNMI
  FA52 C6 82       1891         LDB #082H           single side, single record, side 0
  FA54 34 08       1892         PSHS DP             save DP
  FA56 8D 1E       1893         BSR CMNDSET         read sector start
                   1894         SETDP 0C0H          tell assembler value of direct page pointer
                   1895 * this loop reads bytes from the FDC, X=pointer to storage
                   1896 * maximum NMI latency is 3µs, so loop is arranged to give maximum time
                   1897 * between FDC read and the store to memory. The hanging write to RAM
                   1898 * is done outside the loop
  FA58 B7 00       1899 BREAD1  STB FDC             set FDC going by o/p command
  FA5A 20 02       1900         BRA BREAD3          jump into middle of loop
  FA5C A7 80       1901 BREAD2  STA 0,X+      7     store it away
  FA5E 13          1902 BREAD3  SYNC          2     wait 'til interupt
  FA5F 96 03       1903         LDA FDC+3     4.25get the data
  FA61 12          1904         NOP           2     this nop to allow FDC to NMI
  FA62 20 F8       1905         BRA BREAD2    3     loop            =18.25cycles @ 1.5MHz=12.2µS
                   1906 * will exit by NMI to NMIVECTOR which pushes status onto return stack
  FA64 A7 84       1907 RNMI    STA 0,X             hanging write, store last byte away
  FA66 32 6C       1908 WNMI    LEAS 12,S           delete saved parameters
  FA68 D6 00       1909         LDB FDC             get status
  FA6A 35 08       1910 BREDEX  PULS DP             restore DP
                   1911         SETDP 000           tell assembler what DP is
  FA6C 4F          1912         CLRA                Reset error flag
  FA6D 36 06       1913         PSHU D              push flag
  FA6F BD F873     1914         JSR SAFE            get a safe nmi vector
  FA72             1915         NEXT                [illegible]
                   1916 *                           user stack=adr,side,sector
  FA76 86 C0       1917 CMNDSET LDA #0C0H
  FA78 1F 8B       1918         TFR A,DP            set up DP
                   1919         SETDP 0C0H          tell assembler what value of DP to use
  FA7A 0D 00       1920         TST FDC             clear any FDC irq
  FA7C BF 000C     1921         STX VECTOR6         fixup nmi vector
  FA7F 33 11       1922         LEAU 1,U            delete sector msb
  FA81 37 02       1923         PULU A              get sector number
  FA83 97 02       1924         STA FDC+2           store in FDC
  FA85 AC C1       1925         CMX 0,U++           test for side
  FA87 27 02       1926         BEQ CMNDST2
  FA89 CA 08       1927         ORB #8              set for side 1
  FA8B 37 11       1928 CMNDST2 PULU X              get the ram pointer
  FA8D 39          1929         RTS


  FFFC E02A        2523         FDB VECTOR6E    RMI
```

Notes - By changing contents of location 'vector6' the write routine exits from its loop to location 'WNMI'

- Vector6 is in ram, all other locations are eprom

Diagram of program flow during a sector read on the Forth computer.

The processor starts its interrupt sequence by pushing appropriate registers onto the stack and jumping to code pointed to by a vector in high memory. In this Forth system high memory is eprom so the interrupt vectors cannot be changed to point to different routines. I remedied this by making the interrupt vectors point to code which executes a jump to a location determined by a value stored in ram. By changing data in the ram location the non-maskable interrupt vector can be altered during program execution. This practice is unstructured and therefore unfashionable, but it is highly effective.

Normally an interrupt routine is completed by a return-from-interrupt instruction which restores processor register values to those prior to the interrupt, i.e. restore context. In this case the interrupt vector is being used as a jump instruction to jump out of the data-transfer loop so the first operation in the NMI routine is to delete saved registers (LEAS 12,S). As the controller is connected to the non-maskable interrupt line there is the potential for the occurrence of an interrupt when one is not required. To prevent this the NMI vector points to a safe routine when not in use which reads the controller status register, clearing the cause of the interrupt before carrying out a more normal return from interrupt, RTI, operation.

Extra signals to the disc drives, e.g. track 42, should be inverted and buffered using standard t.t.l. open-collector drivers (7438) as used for the WGATE signal. Extra input signals from the drive such as READY should be buffered using say two LS14 gates, and terminated as previously shown. I should have used the drive's ready signal, eliminating one of the monostable multivibrators connected to the index line but I did not. Also I connected my motor-on signal (5.25in drive, pin16) to the Sony drive head-load line, pin 14, whereas I should have used hold, HLD on pin 28 of the controller. These minor modifications were made because I still intend to use 5.25in drives as they currently offer better value for money than the Sony drives at one-off prices.

The matter of write-precompensation has not yet been resolved. I found by trial and error that for the Sony drive at least write-precompensation is not mandatory. It might be necessary for older 8in drives, and in commercial products to minimize the number of attempts to read the disc. Details of precompensation circuits are given in the Western Digital handbook and reference two.

For drives that keep the disc rotating, such as Sony's and most 8in drives, the disc speed-up hardware previously described should not be fitted but the drive should be connected with the motor-on

List 2. Forth words specific to this system.

```
Note the movement of data onto and off the stack is shown thus
      s1 s2 ..ABC.. r1 r2 r3
  Forth word 'ABC' takes two values off the stack (s2=top of stack) and produces
  three results (r3=new top of stack).
  VEMIT,VKEY,V?TER   - a user variable containing the execution vector for
                       EMIT,KEY, ?TERMINAL respctively
                       ..VEMIT..addr   ..VKEY..addr   ..V?TER..addr
  DPMAX              = a user variable containing the maximum address for the
                       dictionary.
                       ..DPMAX..addr
  SMAX               = a user variable containing the maximum depth of data
                       stack allowed for this user
                       ..SMAX..addr
  P!,P@              = a WORD to store a byte to, read a byte from one of the
                       user ports
                       data addr..P!..          addr..P@..data
  DENSITY            = a constant =0 for single density; -1 for double density
                       ..DENSITY..boolean
  DR-SEL             = a WORD to select the drive value top of stack
                       value..DR-SEL..
  DE-SEL             = a WORD to deslect the disc drives
                       ..DE-SEL..
  VERIFY             = a variable which if true causes a read after write
                       verify of disc writes
                       ..VERIFY..addr
  SIDE/DISC          = a constant returning the number of sides per disc
                       ..SIDE/DISC..value
  SEC/TRK,TRK/SIDE   = constants returning the number of sectors per track and
                       tracks per disc respectively.
                       ..SEC/TRK..value       ..TRK/SIDE..value
  SIDE               = a WORD to select the side of the disc depending on the
                       value at top of stack
                       value ..SIDE..
  ?WRITE             - if disc is write-protected issue error message 10 and
                       abort execution to return to terminal mode
                       ..?WRITE..
  CMND               = a WORD to execute "n-1" command at top of stack
                       value ..CMND..
  RATE               = a CONSTANT which equals the disc drive stepping rate as
                       coded for the WD1793
                       ..RATE.. value
  SEEK               - steps the floppy drive to seek the track that is at top
                       of stack
                       value ..SEEK..
  STEP               = a WORD to step the floppy disc, IN if TOS=1
                                                       OUT if TOS=-1
                                                  same direction if TOS=0
                       value ..SEEK..
  R-ADR              = reads the address mark off the disc, either returns
                       a/ flag=true if unsuccessful (=status of WD1793)
                       b/ sector_type, sector, side, track , false flag
                       ..R-ADR.. type sector side track 0        or
                       ..R-ADR.. true
  BREAD,BWRITE       - similar to FORTH BLOCK_READ, BLOCK_WRITE except
                       additional parameters are included, sector within a
                       track,
                       returns WD1793 status 0-succesful
                       address  side  sector ..BREAD.. status
  SEC-R/W            = similar to FORTH R/W except no limit checking is done.
                       Returns to top of stack a flag. 0=successful operation
                                                       1=crc error
                                                       2=seek error
                                                       3=not ready
                       addr sector flag ..SEC-R/W.. status
  TRK-WR             - a WORD to write a whole track to disc, used in formatting
                       a disc. Takes from the stack the address of the track
                       image. Return WD1793 status, 0=successful
                       addr ..TRK-WR.. status
  TRK-BLD            - a WORD to build in memory a byte image of a track prior
                       to be written out by TRK-WR. Takes as input the side and
                       track number whose image is to be formed and produces
                       the address of where the image is.
                       side track ..TRK-BLD.. addr
  ?DISC              - a WORD to find out what sort of disc is on the drive.
                       sets CONSTANTs B/BUF,DENSITY,SEC/TRK to suit.
                       ..?DISC..
  MS                 - a WORD to delay execution by the number of milliseconds
                       that is top of stack
                       value ..MS..
  SKEW               - currently a no operation word, that can be patched to
                       allow interleaving of the sectors when formatting a disc.
                       Intended useage is that it will modify the sector number
                       currently at top of stack.
                       sector_no ..SKEW.. sector_no
```

signal permanently true, i.e. grounded.

Software issued (first revision) assumes the presence of disc speed-up hardware and includes the faster data-transfer loop. I will supply a drive pin connection list and format program for the Sony drive that can be modified for 8in drives to readers sending an s.a.e. to me at 632 Queensferry Road, Edinburgh. The Forth word BLD-TRK in eprom is only suitable for mini-floppy disc drives.

Thanks to Hewlett Packard for the use of their test equipment and Sony for the loan of a microdrive. Software used, based on the FIG model, was prepared on an HP64000 microprocessor development system.

Integrated circuits 87 and 88 were missing from last month's components list. They are 2114 static rams. In the photograph of power-supply spikes, vertical sensitivity for all but the clock signal is 0.5V/div.

References

1. J. Bovin, Floppy incompatibility, *Systems International*, May 1983, p.61.
2. J. Hoeppner and L. Wall, Encoding/decoding techniques double floppy disc capacity, *Computer Design*, Feb. 1980, pp. 127-135.

Brian Woodroffe plans to describe the Forth language in a subsequent series.

Wireless World Forth computer

Introduction, c.p.u. and memory circuits, May 1983, pp. 53-8.
Circuit description, video-controller circuit and peripherals, June 1983, pp. 55-8.
Software, disc controller and power supply circuits, July 1983, pp. 58-61.
Construction tips, August 1983, pp.44,45.

Complementary current mirror

Current mirrors with transistors of the same type of conductance are well known[1] and widely used in integrated circuits[2]. It is possible to create the configuration with similar properties using complementary transistors also, Fig. a. Accepting the usual assumptions[2] that

$$I_{C1}=I_{S1}\exp|V_{BE1}|/V_T$$
$$I_{C2}=I_{S2}\exp V_{BE2}/V_T$$
$$I_{B1}=I_{C1}/\beta_{F1}$$

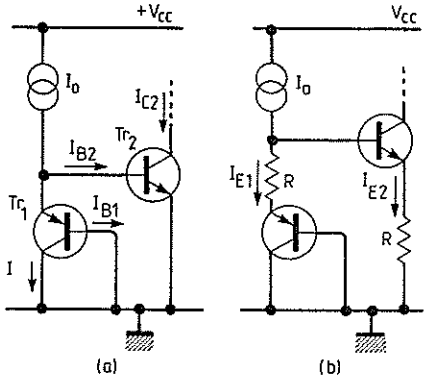
and $I_{B2}=I_{S2}/\beta_{F2}$, the output current is

$$I_{C2}=\frac{I_0}{\left(1+\frac{1}{\beta_{F1}}\right)\frac{I_{S1}}{I_{S2}}+\frac{1}{\beta_{F2}}}$$

with $|V_{BE1}|=V_{BE2}$. If the technology allows two complementary transistors with $I_{S1}=I_{S2}$, then

$$I_{C2}=\frac{I_0}{1+\frac{1}{\beta_{F1}}+\frac{1}{\beta_{F2}}}\approx I_0\left(1-\frac{1}{\beta_{F1}}-\frac{1}{\beta_{F2}}\right)$$

as in the ordinary current mirror. Usually n-p-n and p-n-p transistors in an integrated technology are produced by different methods and parameters I_{S1} and I_{S2}



Complementary transistor current mirror with matched transistor (a) and matched emitter resistances (b).

are not matched. But the discrete current mirror with matched resistors in emitter circuits works reasonably well, Fig. b. In this circuit

$$I_{E1}R+|V_{BE1}|=V_{BE2}+I_{E2}R$$

and

$$I_{E1}=I_{E2}+\frac{|V_{BE1}|-V_{BE2}}{R}.$$

If transistors are designed for complementary operation, say 2N4401 and 2N4403, and emitter resistors are matched to within 1% the error is 2 to 3% without preliminary transistor matching. The gain β_{F1} of a discrete p-n-p transistor is usually high and the collector currents happen to be matched also. – I. M. Filanovsky, University of Alberta.

1. F.J. Lidgey. Looking into current mirrors. *Wireless World*, October, 1979, vol. 68, pp. 51-58.
2. P. Gray, R. Meyer. Analysis and design of analog integrated circuits. Wiley, 1977.