System Support I Startup and Checkout

Support Software for HDOS 2.0:

SYSSUPP.H8D HDOS 2.0 disk image with support programs:

SSIGAL.JED	ATF16V8BQL or GAL16V8D programming for I/O addresses documented below
H8SPD.ABS	Z80 CPU speed change utility (or use any other such program that you prefer)
LPPAR.DVD	Parallel port printer driver disassembled by Glenn Roberts
PLAYIO.ABS	NOGDS music PLAY program, modified for DACs mapped as I/O ports
TESTADC.BAS	MBASIC program to test ADC and 8-channel multiplexer
H8ADC.ASM	HDOS ASM source for ADC test using INT 6 interrupts
AM9511.BAS	MBASIC program to do test the AM9511 APU
AM9511.DOC	Documentation from the MicroTran AM9511 library distribution disk
TIME.ABS	Compiled FORTRAN-80 code to time the MicroTran AM9511 library
TIME80.ABS	Compiled FORTRAN-80 code to time the MS F80 math library

NOGDS.H8D HDOS disk image of NOGDS music software v2.3, as originally distributed

I/O Addresses used by the System Support I Board:

DAC0	024Q/025Q (Write)
DAC1	026Q/027Q (Write)
ADC	040Q/041Q (Read/Write)
MUX	042Q/043Q (Write)
APU	064Q (Read/Write) – Data
	065Q (Read/Write) – Command/Status
APU_RST	066Q/067Q (Write) (soft reset of APU)
Printer	320Q-323Q (Read/Write)
CPU_SPD	362Q (Write) – mirrors the H8 CPU status port

Step 1: Test & configure I/O Wait State Generator

Jumper settings:

P11	I/O Select	jumper on pins 5-6 (right hand set of pins) for wait states generated on IORQ
P14	Mem Select	no jumpers (no memory wait states)
JP8	RDYIN*	no jumper (RDYIN* pullup is internal to v4 CPU board)
JP4	IO_WAIT	jumper (generate I/O Wait States)
JP5	MEM_WAIT	no jumper (no memory wait states)

Connect oscilloscope:

Ch1	JP4 IO_WAIT jumper
	5v/cm
Ch2	IORQ
	5v/cm
Horiz	500ns/cm
Trigger	Ch1 negative pulse edge

Pulse width setting:

Adjust RV3 to obtain a pulse width of approximately 700ns. At 2MHz, the IORQ pulse width should be approximately 1.6us (1600ns).

Testing (Leave the oscilloscope connected as above for the following tests):

Jumper the Z80 CPU board for a maximum clock speed of 16MHz.

Boot HDOS 2.0

Run "`H8SPD x`", where x = 2,4,8, or M (maximum). Try each faster speed, starting with 4MHz. The system should remain stable at all CPU speeds up to 16MHz. The oscilloscope trace should show a constant 700ns pulse width on JP4, while the IORQ pulse width should drop from 1.6us at 2MHz to about 800ns at 16MHz.

The wait state generator should prevent the IORQ pulse width from dropping below about 800ns. Without the wait states, the IORQ pulse width at 16MHz would be about 200ns, and such a short pulse is what causes I/O devices to fail at faster CPU speeds. By forcing the IORQ pulse to remain at or above ~800ns, the wait state generator allows even slow I/O devices to operate at faster CPU speeds.

NOTE: <u>The H8 bus was not designed or intended to operate at 16MHz.</u> But operation at this speed has been quite reliable with I/O wait states and the Trionyx backplane. The Z67-IDE+ has been tested to operate and boot reliably at 16MHz. Even very slow I/O boards such as the H8-4 serial card have been tested successfully at 16MHz when using I/O wait states. You may need to experiment with a longer pulse width (more wait states) if you are having trouble running reliably at 16MHz.

Your results may vary! Particularly if you are using off-CPU memory (such as the 512K memory card), I/O wait states may not be enough to permit reliable 16MHz operation. The System Support board can generate memory wait states, and you may need to experiment with memory waits for 16MHz

operation. Bear in mind that in many cases, adding memory wait states will completely offset the performance gains that are anticipated from the higher CPU speed! Memory wait states affect EVERY instruction, and throughput takes a big performance hit when they are introduced.

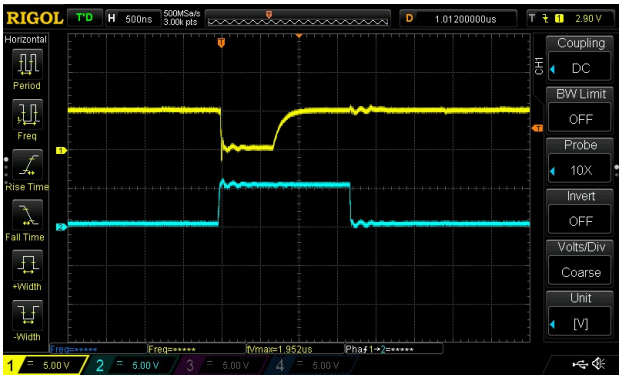
Test platform notes:

The hardware used to test the System Support board wait state generator is as follows:

Trionyx backplane with OEM Heath power supply
Front Panel v2.0
Z80 CPU v4 with onboard RAM and DUART serial ports
System Support board
82C55 PPIO board
H37/67 I/O controller with H17 daughter card

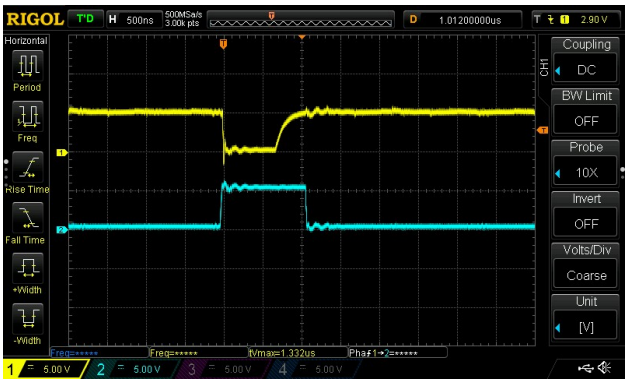
Individual Heath boards that have been tested at 16MHz:

OEM Heath front panel
WIP



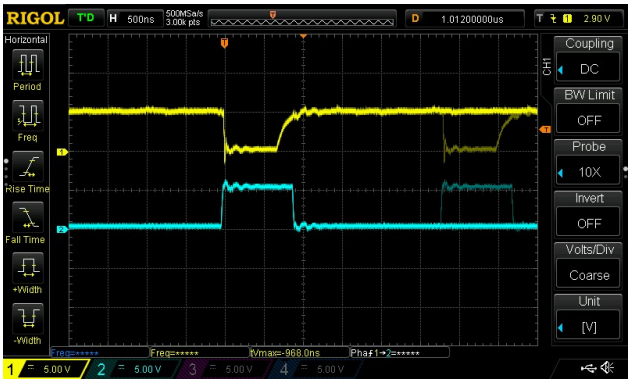
Yellow – JP4 IO_WAIT
Cyan – IORQ

CPU Speed = 2MHz
Adjusted for ~700ns pulse width



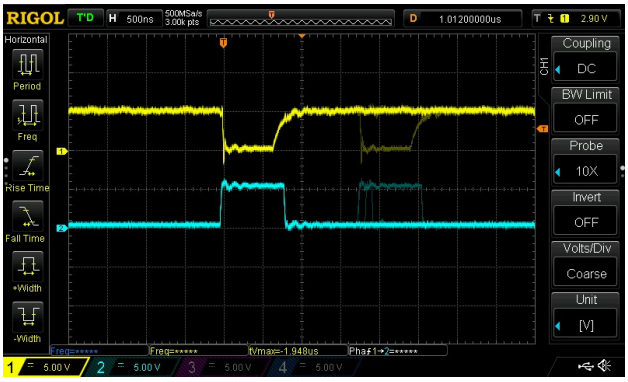
Yellow – JP4 IO_WAIT
Cyan – IORQ

CPU Speed = 4MHz
Adjusted for ~700ns pulse width



Yellow – JP4 IO_WAIT
Cyan – IORQ

CPU Speed = 8MHz
Adjusted for ~700ns pulse width



Yellow – JP4 IO_WAIT
Cyan – IORQ

CPU Speed = 16MHz
Adjusted for ~700ns pulse width

Step 2: CPU Speed Indicator (for use with v2.0 Front Panel)

Jumper settings:

JP3	H17/BANK	Jumper 1-2 (RIGHT) to display H17 Side Select Jumper 2-3 (LEFT) to display Bank Select status

Connect a 5-conductor cable between P7 on the System Support board and P6 (FP LEDS) on the v2.0 Front Panel board, Pin 1 to Pin 1.

Testing:

As you switch CPU speeds, the appropriate LED on the front panel should light. From bottom to top on the front panel: LED1=2Mhz, LED2=4Mhz, LED3=8Mhz, LED4=Maximum MHz, LED5= your selection for H17 Side Select or Bank Select (JP3).

H17 Side Select note: LED "ON" indicates side 0.

Step 3: Parallel Printer Port (skip if no printer is installed)

Jumper settings:

JP15	8255 Mode	For 8255 Mode 1 (preferred), jumper 1-2 (LEFT) For 8255 Mode 0, jumper 2-3 (RIGHT) This setting must match the expected settings for your driver or BIOS.
JP12	RESET	Most software/printers use the RESET signal, so install a jumper here
JP11	/LF	If you have custom software that uses the /LF control, install a jumper here (this control line is not used by Heath software)

Connect a standard IBM-PC motherboard parallel port cable, terminating in a DB25 female connector, to P17. Connect your parallel printer cable's DB25 male connector to the motherboard cable just installed.

Testing:

Boot HDOS 2.0

Copy "LPPAR.DVD" to your boot disk as LP.DVD (or whatever device name you prefer).

Reboot HDOS 2.0

Send a test print (`COPY LP:=SYSHELP.DOC` will work for a typical HDOS 2.0 installation)

Step 4: HA-8-2 Music Board

Jumper settings:

None

Testing:

Set the volume controls RV1 and RV2 to their mid-point.

Connect an amplified speaker or stereo amplifier/receiver to the 1/8" stereo mini-jack at CON1.

Boot HDOS 2.0

Mount the NOGDS Distribution Software (version 2.3) disk, or copy it to the desired boot or work disk.

Replace PLAY.ABS on the NOGDS disk with PLAYIO.ABS from the System Support disk. The NOGDS PLAY.ABS uses memory mapping for the music card. The System Support board uses I/O mapping (for eventual CP/M compatibility). You must use PLAYIO.ABS with the System Support board!

Assuming PLAYIO and CLGAS.PLA exist on your SY0: drive, run "`PLAYIO CLGAS`"

If the program and/or PLA file are on different drives, specify them on the command line. For example:

```
SY1:PLAYIO SY4:CLGAS
```

"Classical Gas" (written by Mason Williams) should begin playback. Adjust the volume controls RV1 and RV2 to a setting that is not so high that it overloads the input of your audio device.

Testing DACs with MBASIC:

```
10 FOR I=0 TO 255
20 OUT &O24,I
30 OUT &O26,255-I
40 NEXT I
50 END
```

The raw (unfiltered) DAC outputs are available at P12. The MBASIC program will (quickly) step through all 256 possible DAC values (DAC 0 will be ascending, DAC 1 will be descending). If you hook an oscilloscope or voltmeter to the DAC0 and DAC1 pins, you should observe voltage varying between 0v and about -5v.

Step 5: ADC0804 Analog-to-Digital converter plus 8-channel input multiplexer

Jumper settings:

None

Connect up to 8 analog inputs (voltage range 0-5v) to P10. The analog voltages are measured with respect to analog ground on pin 10 of P10. *Voltage above +5v may damage the 4051 multiplexer or ADC0804 converter!*

Testing:

Boot HDOS 2.0

Run MBASIC

Load and run program "TESTADC.BAS":

```
10 REM TEST SYSTEM SUPPORT ADC AND MULTIPLEXER
20 REM LOOP OVER 8 INPUT CHANNELS (0-7)
30 FOR I=0 TO 7
40 REM OUTPUT CHANNEL SELECT TO MULTIPLEXER
50 OUT &O42,I
60 REM (ANY) WRITE TO ADC PORT STARTS A-D CONVERSION
70 OUT &O40,0
80 PRINT TAB(I*8);
90 REM RANGE IS (APPROXIMATELY) 0-5V, SCALE 8-BIT VALUE TO VOLTAGE
100 PRINT USING "#.##";INP(&O40)*5/256;
110 NEXT I
120 PRINT
130 GOTO 30
140 END
```

This program continuously displays the voltage measured at each of the 8 input channels.

NOTE: TESTADC.BAS assumes that the ADC conversion started in line 70 is complete by the time it is read in line 100. MBASIC is slow enough that these timing constraints are met. Assembly language programs will need to insert delays and/or use interrupts to guarantee that valid values are available. See the 4051 and ADC0804 datasheets for timing details.

Application Notes:

A typical use of the ADC would be to read joystick values. The joystick potentiometer would be connected between +5v (pin 1) and Ground (pin 10) of P13. The potentiometer wiper would be connected to the desired channel input (Ch 0=Pin 2, Ch 7=Pin 9). A centered joystick would produce an ADC value of about 128.

With the RC values used on the board, the ADC conversion time is approximately 175us, measured from the trailing edge of the ADC WRITE pulse to the leading edge of the ADC INTR pulse. Software sample frequencies approaching ~5000 samples/sec are possible.

TESTADC.ASM is provided as an example of how to use the ADC0804 with interrupt handling (on INT 6).

Step 6: AM9511 Arithmetic Processing Unit (Arithmetic Coprocessor)

Jumper settings:

JP6 APU_WAIT Install a jumper here (holds CPU until the APU is able to respond)

Testing:

Boot HDOS 2.0

Run MBASIC

Load and run program “AM9511.BAS”:

```
10 REM TEST AM9511 APU 32-BIT MULTIPLY
20 P=&O64
30 REM OUTPUT TO PORT &O66 DOES SOFT RESET OF AM9511
40 OUT &O66,0
50 PRINT "  A","  B","STATUS","  A*B","ERROR"
60 FOR J=1 TO 20
70 REM GET RANDOM 8-BIT INTEGER FOR MULTIPLIER
80 A=INT(RND(1)*255)
90 A1=A
100 PRINT A,
110 OUT P,A
120 OUT P,0
130 OUT P,0
140 OUT P,0
150 REM GET RANDOM 12-BIT INTEGER FOR MULTIPLICAND
160 A=INT(RND(1)*4096)
170 A2=A
180 PRINT A,
190 REM PUSH ONTO STACK AS 32-BIT INTEGER
200 OUT P,A MOD 256
210 OUT P,(A - (A MOD 256))/256
220 OUT P,0
230 OUT P,0
240 REM SEND "MULTIPLY" COMMAND TO AM9511
250 OUT P+1,&H2E+&H80
260 REM GET STATUS BYTE (WILL BE ZERO IF COMPLETE - OK FOR SLOW MBASIC)
270 PRINT INP(P+1),
280 REM POP 32-BIT RESULT VALUE OFF STACK
290 A=INP(P)
300 A=256*A+INP(P)
310 A=256*A+INP(P)
320 A=256*A+INP(P)
330 REM PRINT APU RESULT AND ERROR DIFFERENCE (COMPARE APU AND MBASIC
MATH)
340 PRINT A,A-(A1*A2)
350 OUT P+1,0
360 A=INP(P+1)
370 NEXT J
380 PRINT
390 GOTO 50
400 END
```

This program loops continuously through a simple integer multiplication. It confirms basic APU communication and function. The last column is the difference between the results from the APU and the internal MBASIC math library. The difference should be zero in each case. Ctrl-C to terminate the program.

Application Notes:

LINK-80 libraries are available to use the AM9511 APU with Microsoft FORTRAN-80 or BASIC-80 compiled languages.

The AM9511 APU uses a different floating point format than the Microsoft languages, so the libraries must translate before and after the subroutine calls to the APU.

The MicroTran LINK-80 library is copyrighted, and I do not have permission to distribute or reproduce it.

The MicroTran library included detailed patching instructions for Benton Harbor BASIC to replace the internal software math code with APU calls.

TIME.ABS is a FORTRAN-80 program compiled and linked with the MicroTran AM9511 library. It times the execution speed of each operation as documented in the file "AM9511.DOC" from the MicroTran library distribution. The results from TIME.ABS can be compared against the results from TIME80.ABS, which times the execution of each operation using the Microsoft F80 math library.

As Z80 CPU speeds have increased, the performance advantage of the AM9511 APU has disappeared for the integer and even some floating point arithmetic. The transcendental functions are still much faster with the APU. But looking beyond just speed improvements, the APU can offer significant advantage in code reduction or simplification. For example, converting angular servo target positions to the PWM pulse width required to achieve the position would typically require either floating point or at least 32-bit integer multiplication and division. These operations can be easily implemented using the APU in Z80 assembly language with very few instructions.

Hardware Notes:

The Intel C8231A is pin-equivalent to the AM9511.

AM9511s were produced supporting clock speeds from 2Mhz to 4MHz. The APU runs asynchronous to the Z80 clock. Match the oscillator speed to the APU speed.

APUs run HOT to the touch. You can optionally use an adhesive-mounted heatsink.

The APU can generate interrupts at the completion of an operation. Unless you have an application that has something else to do while waiting for the APU (multiple threads), there's no advantage to using interrupts compared to simply polling the APU status register.