HANDY RELEASE 0.8.1
11 May 1989

CONFIDENTIAL and PROPRIETARY

This document describes the V 0.8.1 release of the Handy development environment. It includes these sections:

- New SPRITES and RESPRITE
- Changing the Tick Duration of the Audio Clock
- Other 6502: Changes
- Handebug 1.58
- Handycraft 1.58
- What You Must Do To Start Using This Release

New SPRITES and RESPRITE

New sprite rendering macro, RESPRITE, and the SPRITES macro is rewritten to take advantage of the features supported by the new assembler.

RESPRITE is a short version of SPRITES. SPRITES sets up the sprite engine and causes your sprite list to be rendered. RESPRITE causes your sprite list to be rendered without setting up the sprite engine. This provides a way for programmers who make multiple SPRITES calls per game frame to cut down on a little bit of overhead with all sprite calls after the first one.

The sprite engine needs to be set up once per game frame, when the first sprite list is rendered. So this first call render sprites must use the SPRITES macro. All subsequent calls can use RESPRITE.

Note that there's no problem with making multiple SPRITES calls per frame, if your control flow is such that you're not sure which SPRITES call would be executed first.

Changing the Tick Duration of the Audio Clock

You might want the HMUSIC and HSFX to have a different audio frame rate (the rate at which the audio clock ticks) than the default one provided by the system. You can do this at any time, as long as you use safe techniques.

You can define the initial value for the audio timer by defining the constants AUDIO_CLOCK_MASK and AUDIO_PRELOAD, which constants correspond to the Handy hardware timer clock select and preload values. These constants are defined in hsfx.i, but can be defined by you before or after you include hsfx.i as long as you redefine them before the hsfx.src file is included. For example:

```
; Set up for an audio frame rate of 200 per second
AUDIO_CLOCK_MASK  .EQU   6
AUDIO_PRELOAD     .EQU   78
        .IN 6502:include/hsfx.i
        .IN 6502:src/hsfx.src
```

If you want to change the tick duration of the audio clock at run time, use the SET_AUDIO_CLOCK macro, which macro wants 2 arguments passed in the registers: your new audio clock mask in X and your new audio preload value in A. The clock mask selects the clock to be used by the timer. The preload value acts as a multiplier of the timer clock. If you multiply the

preload value by the clock frequency, you get the number of microseconds between audio clock ticks.

To go from number of ticks per second to the arguments you should supply to SET_AUDIO_CLOCK, use this formula:

- Divide your desired number of ticks per second by one million
- Get the inverse of this number (1 / x)
- Divide this number by your desired clock frequency to get the required preload value. Typically you will want a desired clock frequency of 64 microseconds, so typically you will divide the number by 64

Example use of this macro:

```
; This call specifies a clock mask of 6, which selects
; the 64 usec clock, and a preload of 64.  This will
; change the audio clock to tick 240 times per second
; (which is the "standard" audio frame rate)
    LDA  #64
    LDX  #6
    SET_AUDIO_CLOCK
; This mask of 6 and preload of 78 will change the audio
; clock to tick 200 times per second
    SET_AUDIO_CLOCK
    LDA  #78
    LDX  #6
```

The SET_AUDIO_CLOCK macro stops all music and sound effects. It presumes that you haven't linked your own code onto the audio timer interrupt handler's code. If you have linked code onto the handler, you'll have to link it again after the SET_AUDIO_CLOCK call.

Refer to the hardware documentation for a discussion of the sorts of values that you can give to the audio timer.

Also, by the way, AUDIO_TIMER can be redefined now. AUDIO_TIMER is the constant that's used by the system to describe which timer is used by the audio drivers. The default is timer 6, but you can set it to any reasonable timer.

Other 6502: Changes

New controls.src, with new game button/switch definitions when running under emulation. First software contribution by Needle-san, ha ha, ha ha ha.

The declaration of sysIntTable is moved from within the INITSYS macro in sys.mac where it was stupidly placed to sys.i where it ought to have been defined all along.

Harddefs.i now reflects the new sprite types required by the Shadow-E error in the chip. Hardware bug by Needle-san, ha ha, ha ha ha. For details about the bug, refer to Handy Spec manual, pages 18-19, and Handy Hardware Appendix 2, page 5.

Most of you won't feel the change of this types redefinition. However, there are two new sprite types: XOR_SPRITE is now XOR_SHADOW_SPRITE and BACKGROUND_SPRITE is now BACK_SHADOW_SPRITE. Both names for these new sprites are supported in harddefs.i, but with the first official release of the development environment the old, incorrect names won't be supported.

Handebug 1.58

The BOOT CART function now works with breakpoints. After your file 0 is loaded, Handebug establishes any enabled breakpoints before executing the code.

The symbol table display now allows you to find a symbol by typing its name (case-insensitive) in the symbol name field. Aah, at last.

Bug Fix:

- No longer is 38K lost every time you use the LOAD CART function.

Handycraft 1.58

Bug Fix:

- Emulator handles multiple SPRITES per display frame correctly.

What You Must Do To Start Using This Release

Zippo.