Half Bit Map Support for C99.
And how to Use it.

.

The software in this package is all Public Domain, written by Bruce Harrison. All of the routines are designed to be called from C99 programs, using the C99 compiler and support files written by Clint Pulley. The Half Bit Map mode runs quite well on TI-99/4A computers, but may or may not work on Geneve (Myarc 9640) computers, or on TI-99/4A computers that use an 80 column card. The C99 system is Fairware, and users should send some contribution to Clint Pulley. His address is in the Manual that comes with C99..

.

Why Use Half Bit Map?.

.

The Half Bit Map mode could more properly be called the Enhanced Graphics Mode. Like the "normal" Graphics mode, which is the TI's "default" mode, the screen image consists of 24 rows of 32 8*8 pixel character positions. In that normal mode, color combinations may be assigned only to sets of 8 character definitions, so that for example the "H" through "O" will all be affected by changing one byte in the color table. In Half Bit Map, each character definition has its own eight byte color definition. Thus not only can an individual character's color be changed, but a character can have different colors for each of its eight rows. Obviously this opens up a world of possible ways to use the TI for games and such. When you add the capability for up to 32 sprites that move all by themselves, you've got a really powerful tool. Some time back, we released a set of object modules for the Assembly Programmer to take full advantage of Half Bit Map, but those were not "callable" from the C99 language. In answer to a request from Vern Jensen of Pasadena, CA, we have adapted and expanded those routines for use by C99 programs. The routines are collected into three separate libraries, called CHBSUB/O, CHBINP/O, and CHBSPR/O. CHBSUB/O is the entry point for Half Bit Map from C99. It includes routines to move in and out of Half Bit Map mode, into the normal Graphics mode, or back to C99's default Text Mode. It includes a number of utility routines to display strings, numbers, or characters on the Half Bit Map screen, and allows the programmer full use of the enhanced colorizing capabilities. CHBINP/O can't be used without CHBSUB/O. It provides the ability to take inputs into strings or numeric variables from the Half Bit Map screen. CHRSPR/O also can't be used without CHBSUB/O, but it adds full-blown Sprite capability like that available in Assembly programs using the normal Graphics mode..

.

What's on the Disk?.

.

Lots of Stuff! In addition to the three libraries in the form of D/F 80 object files, there are the corresponding source files with full annotation, so the programmer can see just exactly what each routine does, and how it does it. There are small files for #include lines in your C source, so that all the necessary REFs will be inserted for you by the C Compiler. There are also two demo programs that use most of the routines. They're included as C99 Source (CHB/C & CHI/C), as the resulting Assembly Source (CHB/S & CHI/S), as Object Files (CHB/O & CHI/O), and as E/A Option-5 Program files (CHB1/CHB2 & CHI). To give yourself a feel for this stuff, try running CHB1 and CHI from E/A Option-5. Here's what you'll see:.

.

When CHB1/2 starts, the program will immediately switch your computer to the Half Bit Map mode. The screen will be white with a dark blue edge. Next the program will put a border all the way around the screen two characters wide in Cyan. Then it will put some strings on the screen in various locations. Certain characters will have their colors changed. All of the upper case letters will be white on magenta. The numbers 1 through 3 will be in "rainbow" colors, and the "0" will be in white on dark red. Finally it will put five magnified Sprites on the screen and set four of them in motion. All of that will happen very quickly indeed, so it will appear to have happened all at once. Now just watch for a while. From time to time the cyan sprite will "wink" when all four of the others are on the same screen row. When the four moving sprites all converge at the center of the screen, the yellow "C" will turn around left to right! As you continue watching, when next the Sprites

converge at the middle of the screen, the yellow "C" will reverse again, back to normal. You'll notice a line near the bottom of the screen that says "press enter to exit". Don't! At least not yet. First press the space bar. You'll see the number 1 (rainbow color) appear in row 24, followed by "press ENTER to exit", where ENTER is in white on magenta, and the screen will be scrolled up by one row. Each time a key other than ENTER is pressed, the screen will scroll again, and the number that appears will increment. Notice that during the scroll, the sprites continue their motion, and that the scroll happens very quickly. There's actually a 2/60th second delay built into the program to prevent the scrolling from "running away" from you. As the screen scrolls, the five sprites continue as before, with the C reversing each time they all converge. When you've seen enough, press ENTER to get out of the program. The RE-RUN option will not appear, and you'll be back under E/A control..

.

CHI demonstrates the input routines from the CHBINP/O library. These include an ACCEPT AT type routine for strings, another for numbers, and two single-key input routines that echo the key struck to the screen. The ACCEPT AT type routines include full editing capability, so that Function-1 will delete the character under the cursor, Function-2 will initiate insert, Function-3 will erase the field, Function-S moves the cursor to the left, and Function-D to the right. Function-9, Function-8, Function-E snd Function-X will all exit the routine, as will ENTER..

.

That's a lot to absorb all at once, so we're going to break it down for you into individual routines, and explain what each does, how to call it from your C99 code, and what parameters get supplied or returned. We'll start with the routines in CHBSUB:.

.

CHBSUB/O Routines.

.

sethb() This needs no parameters. It blanks the screen, puts the computer into Half Bit Map, saves the existing character set, clears the Half Bit Map screen, sets all character colors to black on white, then unblanks the screen and returns to your C99 code. You'll then have a blank white screen with a dark blue border..

.

hbcc(fc,bc) This changes the default colors for the Half Bit Map screen. Foreground color becomes fc, background color becomes bc. This may be used either before or after sethb(), as it changes both the present color schemes for all Half Bit characters and the default colors used by sethb(). This can also be used to "undo" all existing colorizings, setting all characters to fc and bc colors..

.

setgm() This too needs no parameters. It blanks the screen, puts the computer in "normal" graphics mode, clears the graphics screen, resets the original character definitions, then unblanks to a green screen with all character colors set to black on green, as is normal for the graphics mode from E/A..

.

settm() Ditto no parameters. This blanks the screen, clears the screen, resets characters, etc. and puts you back into the default C99 condition with a 40-character Text screen in white on dark blue. This and the above are designed to allow your program to mode-switch gracefully, but you must first use sethb() before using either setgm() or settm()..

.

hbstr(row,col,"Quoted String") Here we need parameters. All three must be provided. Row and Col may be just numbers or INT variables. Range for Row is 1 through 24, and for Col is 1 through 32. The string should be contained in quotes as shown. If the string contains the \n, the next part will be displayed on a new line. The routine checks the validity of Row, Col, and won't display anything beyond the end of the screen. If the string would extend beyond the screen, it will be truncated to fit..

.

hbchr(row,col,char) Places a single character whose ASCII value is char at row, column..

.

hbcls() No parameters. This clears the screen in HB mode..

.

hbclf(row,col,rpts) This clears an area of the screen starting at row, col, and extending rpts locations from there..

.

hcolor(char,fc,bc,rpt) Changes the color scheme for one or more characters starting at char to foreground color fc and background color bc. Does the number of successive colors given by rpt. For example, hcolor('w',16,13,4) would color the lower case w, x, y, and z to white on dark green. All four parameters must be supplied, even if the fourth is 1..

.

hrbow(char,rpt,fc1,bc1,fc2,bc2, ... fc8,bc8) Changes the color scheme of characters starting with char and extending rpt characters with a color scheme specified by eight pairs of foreground/background colors. Colors are given in the scheme as used in Basic or Extended Basic. Each number as indicated by fc1,bc1, etc. must range from 1 through 16. The colors fc1,bc1 color the top row of the character, advancing downward to fc8,bc8, which color the bottom row. This is useful for giving characters different colors for each row of the character. See an example in CHB/C..

.

calchr(char,"pattern string") This works like CALL CHAR in Basic or Extended Basic. The pattern string may be up to 16 characters of hex, just as in Basic. If the string is shorter than 16 characters, the rest of the character pattern will be set to zeros..

.

hbscrl() No parameters. This moves everything currently on the screen up one row, and blanks row 24. The contents of row 1 disappear..

.

hbvchr(row,col,char,rpt) This places char on the screen starting at row, col, and repeating downward for rpt times. Like the Basic CALL VCHAR, this will wrap around from the end of the screen to the screen origin and continue repeats from there..

.

hbhchr(row,col,char,rpt) This works the same as the CALL HCHAR in Basic. Like the one above, this wraps around from the bottom to the top of the screen..

.

hbdint(row,col,integer) This places a signed integer quantity on screen in decimal notation. Integer can be either a number or the name of an INT variable. The number as displayed will range from -32768 through +32767. For positive numbers, there will be no space left before the digits, therefore either the first numeric character or the - sign will appear at row, col..

.

That's it for the CHBSUB routines. These should allow you to do all sorts of displaying on the Half Bit Map screen. Next we move to the Sprite services..

.

CHBSPR/O Routines.

.

sprite(num,ypos,xpos,char,color,yvel,xvel) This requires a lot of parameters, so let's take them one at a time. num means a number from 0 through 31, as there are 32 sprites available at any time. They should be assigned in order, from 0 through the number needed. The parameter ypos is in dot-row coordinates, and may range from 0 through 191. Xpos is the dot-column position, and may range from 0 through 255. Char is a character number from 0 through 255. The sprites' characters have their own definitions separate from the normal screen characters, but the normal screen characters' patterns are available for the sprites also, unless you change them. Color ranges from 1 to 16, as for the Basic color schemes. Yvel and xvel, which must be included even if both are zero, may range from -128 through +127. Putting a sprite with non-zero velocity on screen sets up the user interrupt that provides motion. Your program, however, must provide some means of allowing interrupts frequently so that the sprites will continue moving. See the section of the source file CHB/C, in which LIMI 2 and LIMI 0 are included in the loop while awaiting a keystroke..

.

motion(num,yvel,xvel) Changes the velocities of the sprite given by num to yvel, xvel. This works just like the CALL MOTION in Extended Basic, except that num ranges from 0 through

31..
.
yposit(num) This is normally used to set an integer variable, as for example y=yposit(3) would set variable y to the dot-row position of sprite 3..
.
xposit(num) Same as above, but reports the dot-column position into a variable..
.
locspr(num,dot-row,dot-col) Places Sprite identified by num at the specified dot coordinates. Doesn't change velocities..
.
pattrn(num,char) Sets the character definition of a sprite identified by num to the pattern of char. For example if we perform pattrn(3,'F'), sprite number 3 will become an upper case "F"..
.
magnif(maglev) Changes the size of all sprites. the parameter may only be 1 through 4. Setting 1 means all sprites are just normal size. 2 doubles height and width. 3 and 4 are 4-character sprites of normal and doubled size..
.
revmo(num) This is our own unique idea. Just this one instruction reverses the motion of sprite indicated by num in both horizontal and vertical directions. If either velocity is zero or -128, that won't change. (-128 is a special number, which becomes itself again when changed from negative to positive by the Assembly code in REVMO.).
.
coinc(num1,num2) Reports coincidence of two sprites indicated by num1 and num2. Can be used to report into an INT variable as in k=coinc(1,2). The variable k will be 1 if the sprites are within ten pixels of each other in both row and column. Otherwise k will be zero. This could also be used without a reporting variable as in if coinc(1,2) ... .
.
delspr(num) Deletes the sprite indicated by num and all higher numbered sprites. To delete all sprites, use delspr(0). Deleting all sprites will also de-activate the user interrupt that governs sprite motion..
.
rev1(char) Reverses the sprite pattern of char from left to right. This works for single character sprites. This affects only character patterns used for sprites, not normal screen characters..
.
rev4(char) Reverses a four-character definition as used for a sprite with magnify set to 3 or 4 from left to right. As with rev1, this affects only characters used with sprites..
.
sprchr(char,"pattern string") This acts like a CALL CHAR for one of the characters used for sprites. The pattern string may be up to 16 characters of hex notation, just as for normal characters. If the string is less than 16 characters, the remainder of the pattern will be blank..
.
CHBINP/O Routines.
.
x=accnum(row,col,clrsig) This routine is normally used to input a number into an INT variable. It accepts only six characters, because the range of numbers in an INT variable is limited to -32768 through 32767. Leaving the field blank will put a zero into the variable. Scientific notation is accepted, so for example the entry 2.3E2 will result in 230 being placed in the variable. The parameter clrsig signals whether the field is to be cleared before taking input. If non-zero, the routine clears the field. If zero, the field is not cleared, so a default entry can be accepted..
.
accstr(row,col,maxlen,clrsig,buffer) This accepts a string into a buffer initiated as a CHAR array. You must take care that maxlen supplied here is one less than the dimension of the CHAR array. The string is reported out at buffer in ASCIZ format, in keeping with C's practice. A null entry becomes simply a 0 byte at the start of buffer. As above, clrsig <>0 means the field will be cleared before entry is taken. If clrsig=0, the field will not be cleared. Both this routine and the one above check the validity of Row, Col, etc. They will

exit without taking input if the parameters are out of range..
.
k=hbgkf(row,col) This puts a flashing cursor at row, col, then waits for a single keystroke. The key pressed is echoed at row, col, and reported to the variable..
.
k=hbgk(row,col) Nothing appears at row, col until a key is pressed, then the key is echoed at row, col. The key is also reported into the INT variable..
.

Easy to use.

.
The library CHBSUB/O is the main "entry level" for using the Half Bit Map mode with your C99 programs. Neither of the other two libraries can be used without it. If your program needs to take inputs in Half Bit Map, then you'll need to use both the CHBSUB/O and CHBINP/O libraries. If you need to use sprites in Half Bit Map, then both CHBSUB/O and CHBSPR/O will need to be used. If your program needs both input and sprite capabilities in Half Bit Map, then all three object files will be needed. Here's a quick "how to". Put the files HBSO and HBINO on the same disk as your source C99 file, and add the SPRIO file if you'll need sprites. Put #include directives in your C99 source file for dsk1.hbso, dsk1.hbino, and dsk1.sprio. Then put in the instructions as shown above, ending each with ; as required. After Compiling and Assembling, use the files CHBSUB/O, CHBINP/O (and CHBSPR/O if needed) in the Option 3 loading process, preferably before loading CSUP. When it's all put together by the loader, your finished product will have capability you've never dreamed of. We hope you who are doing C99 programming will find this product useful. Should you need help applying any of this, contact the author:.
.

Bruce Harrison.
5705 40th Place.
Hyattsville MD 20781.
U.S.A..
Phone (301) 277-3467

.....