

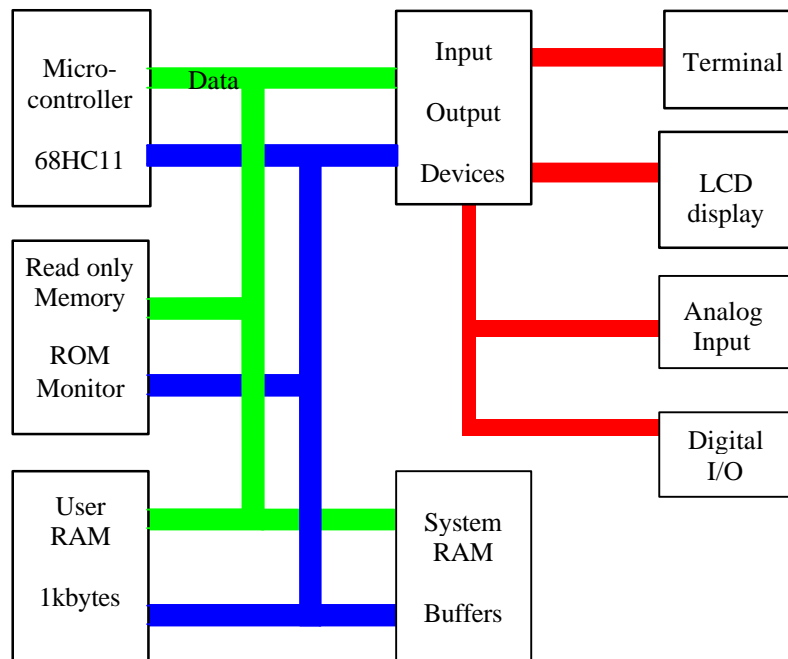
# **HB-BUFFALO DEVELOPMENT SYSTEM**

## **HANDYBOARD SYSTEM**

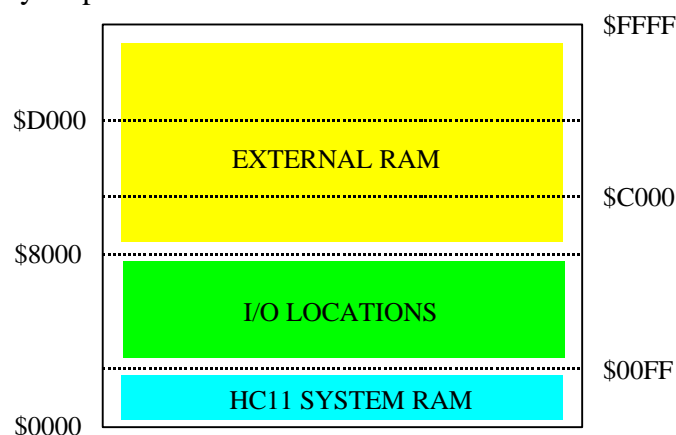
The HandyBoard Microprocessor Development System is a hardware based system developed by MIT\* to facilitate development of microcomputer systems based on the 68HC11 microcontroller. The system comprises all hardware elements necessary to develop and test a small (tiny) microcontroller system.

(\*Fred G. Martin, Massachusetts Institute of Technology)

The hardware configuration is:



The system RAM memory map is:



## **AVAILABLE INPUT AND OUTPUTS**

### **Input**

- Terminal – keyboard data entry
- 7 Analog voltage inputs
- 1 Frob Knob (Variable resistor attached to an analog voltage input channel)
- 6 digital logic inputs
- 2 switches (Stop and Start) on two digital inputs (7 and 8)

**Output**      Terminal - computer screen  
                 LCD display – 2 lines of 16 ASCII characters  
                 4 digital outputs  
                 4 motor drives (with LEDs)

## **HB-BUFFALO MONITOR**

The file, *Buf\_hb.S19*, reproduces *some* of the functionality of the *Motorola<sup>TM</sup> Buffalo* (of the EVB boards), on the HandyBoard.

---

*NOTE: Buffalo is (probably) trade Mark of Motorola Corporation.  
This program is provided 'as is' free of charge for educational use only.  
The Author accepts no responsibility for inaccuracies, errors, or whatever.*

---

The *HB\_BUFFALO* system is configured as follows:

**EXTERNAL RAM**    Used For HB\_Buffalo monitor, user programs, and data storage. (32kB)  
                         HB\_Buffalo -    4kB located    \$C000 to \$D000  
                         User Ram -     16kB located    \$8000 to \$BFFF  
                                        12kB located    \$D000 to \$FFFF

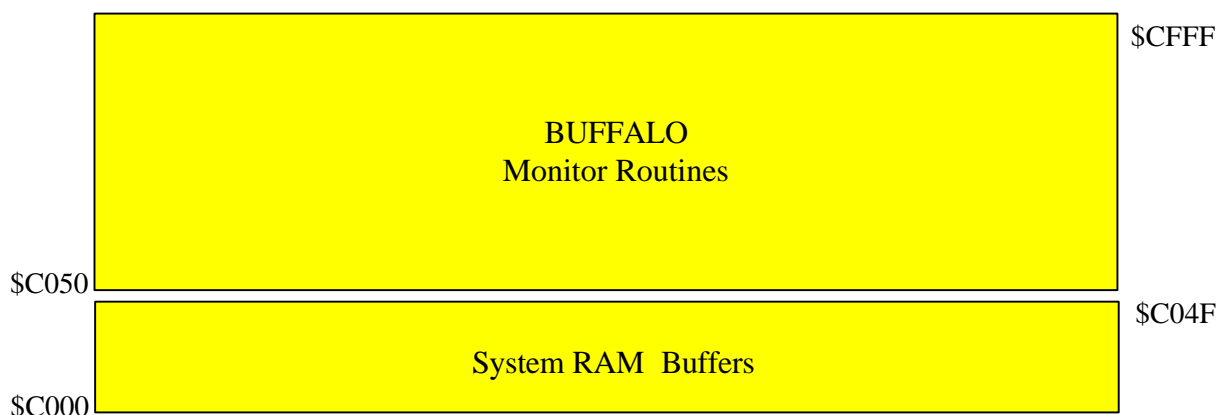
**SYSTEM RAM**     255 Bytes located at Zero Page: \$0000 to \$00FF  
                         Used For HB\_Buffalo monitor scratch pad memory, and display routines.

## **MONITOR ROUTINES AND BUFFERS**

The monitor, or HandyBoard Assembler operating system, is called “HB\_Buffalo”. It consists of a number of subroutines that can be called by the user to perform required functions. The “HB\_Buffalo” monitor routines allow for Machine Code program development and testing.

In addition to the monitor routines there are a number of locations in the system RAM that the user can use to pass information to the operating system routines (or receive information back). These monitor RAM locations are known generically as BUFFERS.

The memory map of the area of RAM set aside for use by the HB\_Buffalo monitor is as follows:



When the development system is powered up, the microprocessor starts executing the monitor routines. The monitor prompt (>) is displayed and the development system is ready to receive instructions by the keyboard. Routines in the monitor receive computer keypresses and send data to the computer display.

The monitor routines are accessible to the user. Monitor subroutines can be used by JSRing (or JMPing) to their start address.

Routines that might be commonly used are:

### **PROMPT    JMP \$C050**

JMPing to PROMPT will return control to the monitor. The system prompt will be displayed. This is the “correct” way to end a program.

The “Prompt” routine in the monitor commences at location \$C050

All programs should conclude with the instruction:    **JMP \$C050**

### **GET        JSR \$C053**

The terminal (computer keyboard) can be used to input data to your Machine Code program.

The monitor subroutine used is called GET. It’s starting address is \$C053.

The instruction    **JSR \$C053**    will execute this routine.

A call (JSR) to GET will cause your program to wait until a key is pressed on the terminal keyboard.

This is indicated on the terminal screen by the display of the special prompt “Data =”.

When a key is pressed, the key will be read and your program will return from the subroutine GET.

The hexadecimal value of the key pressed will be found in accumulator A. If the key was not a hex character then “\$00” will be in accumulator A.

The ASCII value of the key hit will be returned in the system RAM buffer located at \$C030. This RAM buffer is commonly called Key.    **KEY EQU \$C030**

The table following shows the ASCII codes for most keyboard characters. The ASCII code is given in hexadecimal. For example the ASCII code for the character “3” is \$33, that for “A” is \$41, that for “ ” (a space) is \$20.

**ASCII TABLE**

00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	\	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	“	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	\$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	‘	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	-
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	-	6F	o	7F	RO

## **PUT JSR \$C056**

Put copies the ASCII data in the display buffer to the HandyBoard LCD Display.

The sequence of operations to output data is:

1. Store ASCII display codes, for characters to be displayed, in the display buffer (DISBUF). The display buffer is 32 bytes wide. One byte is needed to control the display pattern for each character.  
This display buffer starts at system RAM location: **DISBUF EQU \$C000.**
2. JSR to the monitor display routine (PUT). This routine commences at location \$C056. The instruction **JSR \$C056** will access this routine.  
The ASCII characters referenced by the ASCII display codes contained in DISBUF will be copied to the LCD display screen.  
Once copied, the display will not be changed until the next call to PUT.  
  
Note: The monitor uses DISBUF and PUT to display its own commands. Use of Terminal commands, such as MM, MD, or GO, will overwrite any message you may have loaded into the display buffer.

## **CLRDISP JSR \$C059**

Clears the display buffer by filling the buffer with spaces (blanks). A call to PUT is then needed to send this blank data to the actual display.

## **DISNUM JSR \$C062**

Writes the byte value in accumulator **A** to two consecutive memory locations pointed to by the X Index register. That is; if A contains \$0C, the memory locations X and X+1 will have the two letters '0C' stored. This is useful for writing Byte or Word values to the display buffer.

## **SENDCH JSR \$C05C**

Sends a character in the accumulator **A** to the serial line, and thus sent to the terminal program.

## **SENDSTR JSR \$C05F**

Sends a NUL (\$00) terminated string to the serial line, thus terminal program.  
The string is pointed to by the Index Register, and must be terminated by \$00.

## **SOUND JSR \$C065**

Plays a tone of the HandyBoard speaker. The tone is governed by the byte value stored at **FREQ**, and plays for a duration specified by **DUR**.

*DUR location \$C031      FREQ location \$C032*

## **RDKNB JSR \$C068**

Reads the status of the knob, (0-255), and returns that value in the buffer **KNBPOS** and in the buffer **ADPOS**.

*KNBPOS location \$C033*

**READAD    JSR    \$ C06E**

Reads an Analog to Digital port, (0-7). The port to be converted (0-7) is set by the value sent in Accumulator B. The converted value is returned in the Accumulator B, and is stored in the buffer ADPOS.

*ADPOS location \$C035*


## **HB BUFFALO OPERATION**

The HandyBoard Development System is designed to be operated attached to a terminal. In most cases the terminal will consist of a PC running a terminal emulation package.

To establish a connection to the HandyBoard, the HB\_Buffalo monitor code (BUF\_HB.S19 or BUF\_HB.MON) must be first downloaded. This is achieved by either:

1. Using the downloader provided with the HandyBoard. Such as: DL.EXE or HBDL.EXE.
2. Using the downloader section incorporated with the Terminal program of my HANDYBD.EXE program, for Editing, Assembling, and Downloading of machine code programs.

This is achieved by:

Activate the terminal screen by the button: 

Activate the download section by the button: 

Follow the directions provided on the screen to download the monitor.

On powering up the handyBoard, the Prompt '*HandyBoard Buffalo V2.1*' should display on the terminal program.

Note: Any other terminal program can be used to communicate to the HB\_Buffalo operating system, such as Windows95 HyperTerm.

Terminal settings: 9600 Baud, 8 bits, 1 Stop bit, No Parity.

## **COMMANDS IMPLEMENTED**

Once *HB\_Buffalo* has been downloaded and is in operation, then at the monitor prompt (>), the following commands are available:

Note: When entering addresses or data, hexadecimal values are used, without the leading "\$".  
All monitor commands are finished with an <Enter>.

### **LOAD**

Load S-records. Monitor waits for an S-Record to be downloaded via the serial line.

(Note: NO check is made on the addresses, I/O, ROM and RAM will attempt to be modified)

BF <addr1> <addr2> [<data>]

Block Fill. Fills the memory locations from *addr1* to *addr2*, with the byte *data*.

If no data is specified zero is used.

(Note: NO check is made on the addresses, I/O, ROM and RAM will attempt to be modified)

MOVE <Sadr1> <Dadr2> [<num>]

Block move. Copies memory from *Saddr1* to *Daddr2*. A total of *num* locations are copied,

If no number is specified, one byte is copied.

(Note: NO check is made on the addresses, I/O, ROM and RAM will attempt to be modified)

MD <addr1> [<addr2>]

Memory dump. Displays the memory from *addr1* to *addr2* on the terminal.

Or the 16 Bytes from *addr1*, if *addr2* is not specified.

MM <addr> <data>

Memory modify. Modifies the memory location *addr*, to the data value *data*.

(Note: NO check is made on the addresses, I/O, ROM and RAM will attempt to be modified)

CALL <addr>

Call user subroutine. Monitor does a JSR to *addr*.

GO <addr>

Execute user code. Monitor does a JMP to *addr*.

### WARNING

Some interrupts are setup. These interrupt vector locations are all set to the monitor restart PROMPT routine.

No protection of the Buffalo monitor is in effect. That is, the user code (downloaded or running) or monitor use (memory modify or block filling) could overwrite any RAM location, including the monitor locations (\$C000-\$D000).

### START AND STOP BUTTONS

The state of the two push buttons can be determined by reading from the Digital Input location. This location is not uniquely determined (fully decoded) in the HandyBoard memory map. It exists anywhere between \$7000 and \$7FFF.

A read from any location in the range \$7000 to \$7FFF will return the current state of the eight digital inputs. These include the two buttons, which are at digital inputs 6 and 7.

The “**Start**” button controls the most significant bit (bit 7) of the digital input byte. In its normal (or up position) the MSB is returned as a ‘zero’. If the “Start” button is actually depressed then the MSB will be returned as a ‘one’. Remember that it is a push button, it only returns a ‘1’ if read while it is actually pushed.

The following code would determine the state of the “Start” button:

```
STR TBN    LDAA    $7000    ; read the digital input byte
           BMI     SET      ; if MSB set, then branch to SET
UNSET      xxxx                ; code if NOT set
```

The “**Stop**” button controls the second most significant bit (bit 6) of the digital input byte. In its normal or up position the bit returns as a ‘zero’. If the “Stop” button is actually depressed then bit 6 will be returned as a ‘one’. Remember that it is a push button, only a ‘1’ is returned if read while it actually is pushed.

The following code would determine the state of the “Stop” button:

```
STOPBN     LDAA    $7000    ; read the digital input byte
           LSLA                ; shift left, to make second MSB the new MSB
           BMI     SET      ; if new MSB set, then branch to SET
UNSET      xxxx                ; code if NOT set
```

The following code could also be used to determine the state of the “Stop” button:

```
STOPBN     LDAA    $7000    ; read the digital input byte
           ANDA    #$40      ; mask out second bit (bit 6)
           BNE     SET      ; if it was set, then branch to SET
UNSET      xxxx                ; code if NOT set
```

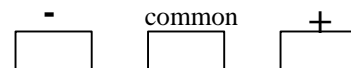
## MOTOR DRIVE OUTPUT PORT

There are four motor drive outputs built into the HandyBoard. These are intended for the operation of small dc motors. The four motor outputs each consist of a pair of outputs which can be set to +12, 0, or -12 volts.

Control of the motor outputs is through the Motor Output location. This location is not uniquely determined (fully decoded) in the HandyBoard memory map. It exists anywhere between \$7000 and \$7FFF.

A write to any location in the range \$7000 to \$7FFF will set the state of the Motor Outputs. The bit pattern written to the Motor Outputs determined whether +12V, 0 or -12V is available at the motor output connectors.

The motor output connections for each motor are:



If the left hand output is “activated” it is set to -12V else it is zero (common). If the right hand output is “activated” it is set to +12V else it is zero (common).

The control byte consists of two nibbles. The bit pattern in the low nibble determines whether the right hand or left hand outputs are being addresses. A ‘1’ addresses the corresponding right hand connection, a ‘0’ the left hand connection. The bit pattern in the high nibble determines whether the chosen connection is activated or not.

- The MSB of each nibble addresses motor connectors 1
- The second MSB of each nibble addresses motor connectors 2
- The third MSB of each nibble addresses motor connectors 3
- The LSB of each nibble addresses motor connectors 4

For example:

- Writing %1000 1000 would turn on right hand connector of motor 1 making it +12V
- Writing %1000 0000 would turn on left hand connector of motor 1 making it -12V
- Writing %0100 0100 would turn on right hand connector of motor 2 making it +12V
- Writing %0100 0000 would turn on left hand connector of motor 2 making it -12V
- Writing %0110 0100 would turn on **right hand** connector of motor **2** making it +12V and **left hand** connector of motor **3** making it -12V

### **Array of 4 red and 4 green LEDs**

The group of 4 red and 4 green LEDs on the HandyBoard show the state of the Motor Drive Output Port. A write to any location in the range \$7000 to \$7FFF will set the state of the Motor Outputs and hence determines which combination of the 8 LEDs is turned on.

The LEDs are arranged in 4 pairs of a red and green LED. From each pair either the red LED or the green LED can be turned on, but not both at the same time. Any combination of the pairs can have a LED turned on.



The following code would turn on each red LED in turn from the top down.

```
CHASE    LDAA    #$1F    ; all LEDS set to RED by $F in low nibble
*                                     top LED set ON by 1 (=0001) in high nibble
                                     STAA    $7000    ; write pattern to Motor Outputs
                                     LDAA    #$2F    ; all LEDS set to RED by $F in low nibble
*                                     second top LED set ON by 2 (=0010) in hi nibble
                                     STAA    $7000    ; write pattern to Motor Outputs
                                     LDAA    #$4F    ; all LEDS set to RED by $F in low nibble
*                                     second bottom LED set ON by 4 (=0100) in hi nib
                                     STAA    $7000    ; write pattern to Motor Outputs
*                                     LDAA    #$8F    ; all LEDS set to RED by $F in low nibble
                                     bottom LED set ON by 8 (=1000) in high nibble
*                                     STAA    $7000    ; write pattern to Motor Outputs
                                     BRA     CHASE
```

The following code would turn on a bar of green LED in turn from the top down.

```
BAR       LDAA    #$10    ; all LEDS set to GREEN by $0 in low nibble
*                                     top LED set ON by 1 (=0001) in high nibble
                                     STAA    $7000    ; write pattern to Motor Outputs
                                     LDAA    #$3F    ; all LEDS set to GREEN by $0 in low nibble
*                                     top two LEDs set ON by 3 (=0011) in high nibble
                                     STAA    $7000    ; write pattern to Motor Outputs
                                     LDAA    #$7F    ; all LEDS set to GREEN by $0 in low nibble
*                                     top 3 LEDs set ON by 7 (=0111) in high nibble
                                     STAA    $7000    ; write pattern to Motor Outputs
                                     LDAA    #$8F    ; all LEDS set to GREEN by $0 in low nibble
*                                     all LED set ON by F (=1111) in high nibble
                                     STAA    $7000    ; write pattern to Motor Outputs
                                     BRA     BAR
```

The following code would “flash” the second bottom LED from RED to GREEN.

```
FLASH     LDAA    #$40    ; all LEDS set to GREEN by $0 in low nibble
*                                     ; second bottom LED set ON by 4 (=0100) in hi nib
                                     STAA    $7000    ; write pattern to Motor Outputs
                                     LDAA    #$4F    ; all LEDS set to RED by $F in low nibble
*                                     ; second bottom LED set ON by 4 (=0100) in hi nib
                                     STAA    $7000    ; write pattern to Motor Outputs
                                     BRA     FLASH
```

---

*NOTE: Buffalo is (probably) trade Mark of Motorola Corporation.  
The HB-Buffer reproduces (some) functionality of Buffer for the HandyBoard.*

*The Author accepts no responsibility for inaccuracies, errors, or whatever.  
This program is provided ‘as is’ free of charge for educational use only.*

---

Author:

Charles Hacker,  
Griffith University - Gold Coast,  
Parklands Drive, Southport,  
Queensland, 4215.  
Australia.  
Email: C.Hacker@mailbox.gu.edu.au