PROGRAMMING

This section contains information on how to program various operating modes for TAPS. It also provides information on the layout of these programs so that a knowledgeable user could modify existing programs or write entirely new programs for new operating modes.

The CPU used in the new TAPS is faster and more capable than the previous processor -- but it has less memory available for program storage. In order to maintain the functions of the older TAPS systems in this new design, a different approach was taken.

The program stored in the FLASH EPROM on the CPU chip is now just a driver program. It does not accomplish any of TAPS functions itself. These programs have been separated from the orginal code and saved as complete programs on the compact-flash RAM card (CF-RAM). The driver program knows how to read from the CF-RAM to place these programs into the RAM.

When power is first applied to TAPS, the driver program does basic initialization and then waits for about 4 seconds for an input on the serial port. If a keypress is detected, it then asks if the user wants to change operating modes. If the answer is Y (or y), a menu screen is displayed and the user can choose which operating mode is desired. The driver program then loads this program from CF-RAM and executes this program.

```
PRESS ANY KEY TO CHANGE MODES, K TO EXIT PROGRAMY

CURRENT MODE IS 0  - CAST
DO YOU WANT TO CHANGE MODES (Y/N)?

       MODES
0     CAST
1     EXTERNAL SOUNDER
2     INSTRUMENT
3     RAW CAST
4     TEST

ENTER NEW MODE CODE: 1
```

If no keypress is detected, the driver program inspects the RAM to see if a program is present. It does this by calculating a byte checksum over a specific part of RAM (containing program instructions) and comparing this value to a stored value in another part of RAM. If these values agree, then this code is executed.

Thus, if TAPS were previously programmed with the desired mode, the user need do nothing to have it begin operating about 5-6 seconds after power is applied.

Selecting a different operating mode causes the driver program to load a RAM image from the CF-RAM into working RAM. This image includes the byte checksum for the section of code and an execution address. Once the code is loaded, the execution address is retrieved and executed.

Obviously, if you want to change operating modes in TAPS you have to be connected to the serial port with an operating terminal program (HyperTerm or equivalent) before you install the shorting plug on TAPS. If you miss the 4 second window to type a key you will have to pull the shorting plug, wait a couple of seconds, and re-install it.

OPERATING PARAMETERS

Each of the programs that produce the operating modes of TAPS shares a data structure which contains parameters such as the number of pings per average, turn-on depth, etc. Not all of the programs use all of this data but all programs share this data structure. This data structure is saved to the EEPROM on the CPU chip whenever it is modified. It is read into RAM from the EEPROM whenever power is applied.

Since different operating programs use different subsets of this data structure, changing to a new operating mode can result in some strange selections of parameters. Thus, when an operating program is run for the first time, the program will display the programming screen appropriate for that mode to force you to accept and/or change these parameters.

For example, in the screen shown on the previous page the user has elected to shift from CAST to SOUNDER mode. This will result in the following screen from TAPS:

```
LOADING CODE INTO RAM  ...
LOADING EXT SOUNDER MODE CODE
..................................................

REPROGRAM OPERATING PARAMETERS:
ENTER NEW DATA OR <CR> TO ACCEPT CURRENT VALUE

# PINGS/AVERAGE =  6   32
MAXIMUM RANGE = 19.000
# DEPTH BINS =  150   200
NEW MAXIMUM RANGE = 25.250
OUTPUT MODE IS BINARY
ENTER 0 FOR BINARY / 1 FOR ASCII-HEX:

SELECT EXTERNAL SENSOR TYPE:
    0 = NO SENSOR INSTALLED
    1 = FREQUENCY OUTPUT SENSOR
    2 = VOLTAGE OUTPUT SENSOR

EXTERNAL SENSOR # 1 TYPE = 2 0
EXTERNAL SENSOR # 2 TYPE = 1 0

MAKE THESE CHANGES PERMANENT (Y/N)? Y
```

The first part of the screen is simply informative, noting that the code is being loaded. The remainder of the screen is the programming screen for SOUNDER MODE. In this mode, parameters like the turn-on depth are not used and so are not displayed. Generally, we use more pings per average in SOUNDER MODE than in CAST mode (since each range bin consists of one sample per ping). Parameters specific to this mode may be left over from the last time this mode was setup (e.g., # depth bins) or may be from the previous mode (e.g., # pings/average). In this example, we have changed the number of pings per average, the number of depth bins, and the instrument setups.

Typing <CR> (or ENTER) will keep the displayed values. Entering a new value will change that parameter. At the end,

you will be given the opportunity to save the entire data structure to EEPROM.

Note that if you don't save the data structure, the next time power is applied the old parameters will be read from EEPROM and used for data collection.

MEMORY MAPS

The CPU used in TAPS is a Motorola 68HC12. This chip is an 8/16 bit processor with a normal memory space of 64 kB. The lower 32 kB is devoted to registers, an EEPROM, a small on-chip RAM, and external RAM; TAPS uses a non-volatile RAM chip with a real-time clock. FLASH EPROM. The upper 32 kB is setup for flash RAM which contains the FORTH O/S and the driver program.

Data manuals and programming information are included in the Programming section of the TAPS CD.

A memory map for this processor is shown below:

FROM	TO	SIZE	CONTAINS
0000	00FF		System registers
0100	07FF		Used by FORTH
0800	0BFF	1024	FREE RAM
0C00	0CFF		do not use
0D00	0FFF	768	EEPROM
1000	7FEF	28672	FREE RAM for programs/data
7FF0	7FFF	16	RTC registers
8000	B7FF	14336	FLASH EPROM for programs
B800	F5FF	15872	kernel in flash (FORTH)
F600	F7EF	494	FLASH EPROM for programs
F7F0	F7FF		Interrupt Vectors
F800	FFEF		serial boot loader
FFF0	FFFF		main vectors

The driver program is loaded into FLASH starting at $B800. Operating programs are loaded into RAM starting at $1000. In general, data and variable storage is at low memory locations, with code following.

External RAM is comprised of a Non-Volatile RAM chip with a Real-time-clock embedded. This RTC uses the upper 16 bytes of memory for the data registers, from $7FF0 to 7FFF.

External I/O is accomplished via digital registers on the CPU chip. These registers include those shown in the table on the following page. Note that many digital registers consist of a data register for reading/writing data and a data direction register that determines, bit-by-bit, if a register is read or write. In some cases, mixed used of a register (Port T is an example) requires that the DDR be re-set after each use.

Details of the ports used by TAPS and the state of each port at initialization is shown in the table following the port listing.

The CPU is part of a single-board-computer manufactured by New Micros, Inc. of Dallas, TX. Their web site is www.newmicros.com. The card part number is NMIS/L-0912. Data on this card and some information on the Forth system embedded in Flash is provided in the Programming section of the cd.

68HC12 ports and registers.

ADRS	**REGISTER**
0056	Port P
0057	Port P DDR
006F	Port AD
00AE	Port T
00AF	Port T DDR
00FE	Port DLC
00FF	Port DLC DDR
00C0	baud rate (16 bits)
00d0	SPI 1 control reg
00D1	SPI 2 control reg
00D2	SPI baud rate
00D3	SPI status reg
00D5	SPI data reg
00D6	Port S
00D7	Port S DDR
0080	Timer I/O select
0084	Timer counter reg
0086	Timer control reg
008B	Timer Control Reg 4
008D	Timer interrupt mask 2
008E	Timer interrupt flag reg 1
008F	Timer interrupt flag reg 2
0090	Timer input compare reg 1
0091	Timer input compare reg 2

PROGRAMMING

68HC12 Port usage in TAPS

BIT	PIN #	I/O	PRIMARY FUNCTON	TAPS-NG	I/O	INIT
PT0	31	I/O	IO Capture	FREQ 1	I	
PT1	32	I/O	IO Capture	FREQ 2	I	
PT2	29	I/O	IO Capture	M0 - MUX	O	0
PT3	30	I/O	IO Capture	M1 - MUX	O	0
PT4	27	I/O	IO Capture	M2 - MUX	O	0
PT5	28	I/O	IO Capture	G0 - GAIN	O	0
PT6	25	I/O	IO Capture	G1- GAIN	O	0
PT7	26	I/O	IO Capture	G2 - GAIN	O	0
PS0	55	I/O	RxD	RxD	I	
PS1	56	I/O	TxD	TxD	O	
PS2	53	I/O			O	
PS3	54	I/O			O	
PS4	51	I/O	MISO	MISO	I	
PS5	52	I/O	MOSI	MOSI	O	
PS6	49	I/O	SCLK	SCLK	O	
PS7	50	I/O	\CS		I	
AD0	65	I	AtoD 0	\BUSY - CF	I	
AD1	66	I	AtoD 1	\BUSY - ADC2	I	
AD2	63	I	AtoD 2	\CFDET	I	
AD3	64	I	AtoD 3	VERROR	I	
AD4	61	I	AtoD 4		I	
AD5	62	I	AtoD 5		I	
AD6	59	I	AtoD 6	FREQ1	I	
AD7	60	I	AtoD 7	FREQ2	I	
PP0	34	I/O	PW0/General IO	RUN/STOP	O	1
PP1	33	I/O	PW1/General IO	SHTDWN	O	1
PP2	36	I/O	PW2/General IO	\ADC2	O	1
PP3	35	I/O	PW3/General IO	\ADC1	O	1
PP4	38	I/O	General I/O	\DAC	O	1
PP5	37	I/O	General I/O	FREQ SELECT	O	1
PP6	40	I/O	General I/O	\DDS	O	1
PP7	39	I/O	General I/O	\XGATE	O	1
PDLC0	41	I/O	DLCRx	reserved	I	
PDLC1	44	I/O	DLCTx			

PDLC2	43	I/O	General I/O		O	
PDLC3	46	I/O	General I/O		O	
PDLC4	45	I/O	General I/O		O	
PDLC5	48	I/O	General I/O	TRANS PWR	O	0
PCLC6	47	I/O	General I/O	INST PWR	O	0
PE0		I/O	\XIRQ		I	
PE1		I/O	\IRQ		I	
PE2		I/O	R/W		O	
PE3	13	I/O	\LSTRB/TAGLO			
PE4		I/O	E-CLOCK		O	
PE5	16	I/O	IPIPEO/MODA		I	
PE6	15	I/O	IPIPE1/MODB		I	
PE7		I/O	\DBE			

In this table, an O denotes an output and an I denotes an input. The values in the right-hand column are zeros and ones, logical values used to initalize the DDRs.

The tables above can be used in conjunction with the CPU/IO schematics to trace signals for troubleshooting purposes. In this case, a logic 0 is a voltage below 1 V while a logic 1 is a voltage above 2.4 V. Test points are provided to monitor most of these signals.

PROGRAM STRUCTURE

The operating programs are written in Forth. The kernel of this operating system/language is loaded into the Flash RAM starting at $B800. Additional Forth 'words' are defined by the operating programs using the basic vocabulary included in the kernel. Forth can be learned by anyone who is motivated to do so. It is one of the more terse languages available, using some standard characters (such as . and ,) to stand for basic operations (print and store, respectively). It is quite economical of memory space, however, and is quite fast in execution. In previous versions of TAPS, Forth was embedded in ROM on the CPU chip. In this version, it is embedded in the Flash EPROM on the CPU chip and can be overloaded if desired.

In all of the operating programs used in TAPS, variable storage begins at address $1010. Generally speaking, the space starting here is used for variables, constants, and data arrays. Code begins after all of the variable and constant storage is defined. The space between the start of external RAM ($1000) and $1010 is used to hold data used by the driver program.

The two bytes at $1000 hold the execution address (code-field-address in Forth) for the main word of the operating program. Loading this address onto the stack and EXECUTE-ing it will cause the operating program to run.

The two bytes at $1002 hold a byte-wise checksum for the program code located between $3B00 and 4B00. This checksum is computed when the program is first loaded into the CF-RAM card. The driver program computes this checksum again each time TAPS is powered-up; if the values match, it uses the execution address to start the operating program. If the values do not match, the driver

program will display the MODE selection menu.

The two bytes at $1004 hold a first-run flag. This flag is set to True ($FFFF) when the operating program is loaded from CF-RAM. During the initalization part of the operating program, this value is tested and, when TRUE, the reprogram code is executed. This is how the program forces the user to verify and/or change the operating parameters when a new mode is installed.

COMPACT FLASH

The compact flash memory card provides space for both operating programs and data. Details of the interfacing are provided in the Programming section of the cd under Programming Info.

Used as an extended memory device, the compact flash card is composed of 512 byte sectors which must be read or written as an entirety. Some insight into the use of buffers to hold data prior to writing to the compact flash may be obtained by studying the code for CAST and RAWCAST modes.

On the 128 MB card, there are 262,144 sectors. Each sector may be directly found from a long-word address. TAPS uses sectors at address $1000 and up for data. A long-word counter in NV-RAM is used to hold the current sector address and is updated each time a data sector is written. Since there are 258,048 of these sectors available, no error checking is done to see if the current data address is greater than the end of the memory. It is assumed that data will be downloaded long before memory is full.

The operating programs are loaded into compact flash at sector addresses beginning at $100 (see the table, below). Each program is a binary image of the RAM space from $1000 to $7FFF. These images are created by loading the individual programs into RAM and then loading and executing the SAVEPRGM.4TH program. This program requires entry of the compact flash sector number at which to start saving the code image and then execution of the main word of this routine ; e.g.

```
200 SAVE-PROGRAM
```

would save the external sounder mode code to compact flash.

MODE	PROGRAM	SECTOR
CAST	WCAST.4TH	100
SNDR W/ TVG	SEXTSNDR.4TH	200
SNDR W/O TVG	NOSNDR.4TH	300
INSTRUMENT	INSTRUM.4TH	400
RAWCAST	RAWCAST.4TH	500
TEST	TAPSTEST.4TH	600

PROGRAM OPERATION

Some insight into the operation of this CPU can be obtained from the data provided by New Micros in the paper MAX12FORTH.pdf located in the Programming Info section on the cd. This information will be summarized and expanded here. Features peculiar to TAPS will be explained in some detail.

The CPU in TAPS follows a standard Motorola startup procedure. The FORTH kernel code pre-empts part of this startup by checking for a quick-start sequence in the EEPROM at address $0FFE. If the byte sequence $A55A is found at this location, the word at EEPROM location $0FFC is loaded as an execution address and executed. One of the benefits of having this quick-start code sequence available is to write the write-once registers on the CPU chip.

In TAPS, the code FFLASH.4th has been loaded into EEPROM and the code field address of this routine put at location $0FFC. This routine sets the baud rate to 19200 baud, enables external RAM access, and pauses a short time. When this code is completed, execution returns to FORTH.

Following this startup routine, the kernel program searches memory at specific locations for an autostart flag word ($A55A or $A44A). If either flag is found, the next word is taken to be a code field address and execution begins at that address. If the flag word was $A55A, this code is executed repeatedly, should an exit or error occur; if the flag word was $A44A, then an exit from this code will return operation to the FORTH kernel.

The end of the program DRIVER.4th contains these words to be executed:

```
a44a d00 ee!
' main cfa d02 ee!
```

These commands put the re-entry autostart flag in EEPROM followed by the code field address for the word MAIN. Thus, after the quick start code has executed, control will pass to the driver program. Should the user specify an exit, program control will pass to FORTH again.

The driver program is fairly simple. It displays the line

```
PRESS ANY KEY TO CHANGE MODES, K
TO EXIT PROGRAM
```

and waits for a user input. If any key except K is pressed, then a new-program dialog ensues and the user can pick from the menu of programs which one to load.

After the program is loaded, or if no keypress was detected, the driver program computes a byte-wise checksum over the address range $3B00-4B00 and compares this to the value stored at address $1002. If the values match, the address located at $1000 is loaded and execution resumes at that point.

If the user has entered a K, the driver program ceases execution and control is passed to the FORTH kernel. This option is provided so that new programs can be loaded into either the FLASH-EPROM on the CPU chip or into the compact flash memory.

LOADING PROGRAMS

There are two distinct methods for loading code into TAPS: loading the driver program into FLASH-EPROM and loading operating code into RAM and thence into the compact flash memory. These methods differ because of the final destination.

Normally, re-loading code does not require that TAPS be opened. Should a problem arise, it will be necessary to open TAPS to get at certain jumpers. In extremis, it may be necessary to use a Background Debug Monitor (BDM) to wrest control of the CPU. This latter happens mostly when the correct procedures are not followed.

A suitable terminal program will be necessary to download code to TAPS. New Micros offers a free terminal program called NMITerm that runs on PC's. We have used it to download FORTH (operating) code. It is a bit tricky to setup and may or may not work with S-

files (see later). We keep an old DOS machine alive strictly for the purpose of running programs that allow us control of properties that modern terminal programs (like, ugh, HyperTerm) do not. We use BitCom, which is probably no longer available. There is a free program called BitWare that claims to include BitCom as a terminal program but we haven't tried it. ProComm is a similar DOS-based terminal program that would probably work.

The essential controls are filtering (to convert all typed or downloaded text to upper case), character echo pacing (for operating program downloads only), and line pacing. The line pacing for operating program downloads should be a specific character (LF or $10) while the line pacing for S-files or the driver program should be timed (20-30 mSec).

To load a new version of an operating program, follow these steps (which assume you have a terminal program properly configured to download code to TAPS):

1. Power TAPS and press a K when the startup dialog is seen. You should see a response like:

```
MaxFORTH 5.0 >
```

2. Download the new code to TAPS. You should use either the NMITerm program or a suitable terminal program that allows pacing with character echo and LF line pacing.

3. When the code has completely loaded (without errors), you can execute the code to test it if you wish. This is how the operating code was developed.

Otherwise, download the program SAVEPRGM.4th. This code will load just above the operating code previously loaded. Then type the sector address for this code (from the table on page 7) and the word SAVE-PROGRAM. Hit ENTER and the program will print out its progress as it saves the code image to compact flash memory starting at the sector address you gave it.

4. Restart TAPS and press any key but K. Answer Y to the load new code question and load the new operating code.

Loading a new driver program is a bit more complicated. If the following steps are followed carefully, however, this should be a simple process.

1. Power TAPS and press a K when the startup dialog is seen. You should see a response like:

```
MaxFORTH 5.0 >
```

2. Type the following exactly as shown:

```
HEX
FFFF D00 EE!
D00 10 DUMP
```

This should result in a line of hex characters displaying the contents of 16 bytes beginning at the start of EEPROM. It is very important that the autostart flag at $D00 be over-written. If the first two bytes at $D00-1 are not $FF FF then the words above must be entered again. DO NOT PROCEED until the data dump shows the autostart flag has been over-written.

3. Type the following:

```
FLASH
```

and change the baud rate on the terminal program to 9600. Hit ENTER a time or two until you see the line

```
ERASE (E) OR PROGRAM (P)?:
```

Type an E. After a short pause (while the FLASH EPROM is erased), this line will re-appear.

4. Type a P. At this point, you need to start downloading the Forth kernel (F12V50L.S19). This is an S-file record that must be downloaded using timed pacing rather than the character/line pacing used to download programs. Set the line pacing to about 20-30 mSec.

When the download begins, you will see groups of asterisks appear on the terminal screen. Wait until the previous dialog line re-appears:

```
ERASE (E) OR PROGRAM (P)?:
```

5. Restart TAPS. Change the terminal baud rate back to 19200. You should now see the response

```
MaxFORTH 5.0 >
```

shortly after startup.

6. Download the program DRIVER.4TH. Use the pacing from the previous download. It will be substantially slower than the operating program downloads but the re-definition of some standard FORTH words causes error messages that would otherwise stop the download.

7. Restart TAPS. You should now see the driver program dialog upon startup.

WRESTING CONTROL

If the directions above are not followed precisely, it is possible to lock up the CPU and lose control. This typically happens when the driver program is changed but the autostart flag is left in EEPROM. The hints here are intended to help but, of course, the solution depends upon the problem.

A Background Debug Monitor is a useful device/program for fixing problems related to autostart vectors and such. We use a unit from Axiom Manufacturing in Garland TX (AX-BDM12) but almost any BDM setup for the 68HC12 CPU will probably work. There is a BDM connector on the CPU card -- obviously, you will have to open TAPS and remove the electronics unit to get access to this connector.

When we are developing code -- with the inevitable code lock-ups that entails -- we remove the CPU card from the electronics cage and work with it alone on the bench. We use a 12V DC supply to provide board power (J1 pin 1 = +, pin 2 = ground). We have made an adaptor to plug onto J6 that runs to a DB-9 serial connector. Pin 1 is SI (serial data in) and pin 2 is SO (serial data out). Pins 3-4 are ground. SI, SO, and GND are the only pins required for a simple serial interface to a PC.

The file EEPROM.S19 in the Folder DRIVER MISC CODE in the folder FORTH CODE in the Programming section of the CD can be used with a BDM to download over the EEPROM at address $0D00 to kill any autostart vectors if the procedures above do not work. There is generally a reset control on the BDM that lets the BDM program interrupt the CPU and view and alter memory contents.

Another possibility -- if you can exit the driver program with a K but cannot get to the FORTH prompt, try typing a CTRL-G and then short the reset pin to ground (on the BDM connector -- see the NMIS schematic and the BDMComm.pdf file to locate these adjacent pins). The CTRL-G reset skips the autostart sequence and may allow recovery to FORTH. Clearly, this requires TAPS to be open as well.

Sometimes it is possible to enter the FLASH reprogramming code and then do a CTRL-G reset escape (50-50 chance). This requires installing a jumper on the PDLC-0 pin and ground (just across from this pin on the 72-pin connector on the SBC itself). Set the terminal to 9600 baud. After the jumper is installed, do a

reset (short the reset pin on the BDM connector to ground) and you should see the FLASH reprogramming line. Remove the jumper from PDLC-0. Type CTRL-G and short the reset pin again. You may now see the FORTH prompt (and you may need to change to 19200 baud to see it).

Of course, if you plan to develop new programs or even, perhaps, to practice downloading programs, it might be wise to do so with the spare CPU/IO card. In this event, you will still have a working TAPS while you are wrestling with the joys of SBC programming!

MODIFYING PROGRAMS

The information in the sections above is provided largely in case revised programs are supplied by us to replace existing programs. In that case, the instructions for downloading the programs should be sufficient.

In the event that a user wishes to create a new operating mode -- e.g., save raw data in SOUNDER mode -- then the information on program structure and CPU memory structure, along with the examples from the existing programs, may be of some benefit. Whenever possible, of course, the existing programs should be modified.

However, it is certainly possible to reprogram the CPU to operate under a completely different O/S, such as C, and run programs in a completely different fashion from those provided. We started using FORTH simply because that was the O/S bundled in ROM on the first SBC's we used. Learning FORTH seemed simpler than replacing the CPU chip or designing our own CPU card. This need not stop a determined user from starting over from scratch with this system.

One possible modification that the user might wish to make is to change the channel frequencies and/or gains. The code segment show below is contained in every operating program in the section defining constants. It consists of three parts: prefix codes, gain/mux/dds codes, and frequencies for display purposes.

The DDS is loaded with two frequencies at a time: the transmit frequency (times 2) and the local oscillator frequency (transmit frequency plus 35 kHz). Each frequency consist of 4 bytes of data. Prefix codes are used to send the frequency bytes to the DDS chip. Each byte of frequency code requires a corresponding byte of prefix to direct it to the proper register.

In addition to the frequency codes, the frequency table contains the MUX codes to select the proper input to the receiver and the proper transmitter channel, and GAIN codes to set the final gain stage in the receiver.

The layout of the frequency table is explained in the header in the code segment. The first two bytes are the MUX and GAIN codes, followed by eight bytes of frequency code. The assembly code routine, *send-freqs*, is included in each of the TAPS operating programs to send the frequency codes to the DDS. This routine sends a prefix byte, a data byte, a prefix byte, etc. until the prefix byte equals zero, signifying the end of the table.

Changing gain codes or frequencies requires computing new values for the frequency table entries. The values embedded in TAPS were computed using the spreadsheet, TAPS-NG freq table.xls, included on the cd in the Programming section. This spreadsheet computes the frequency codes for the DDS and converts these values to hexadecimal for entry in the code table. It also computes the offsets for the MUX and GAIN codes. Note that the second byte in the frequency table is 00 for all channels.

```
( ------------------ prefix table for dds codes ---------------------
hex
create prefix
3322 , 3120 , 3726 , 3524 , 0000 ,

( ---------------------- frequency tables ---------------------------
(                            5/09/05
( layout is ttdd ffff ffff llll llll, where
(     tt is the mux/gain code
(     dd is a dummy byte [0]
(     ffff ffff is the code for 2*xmit freq
(     llll llll is the code for the lo freq
hex
create freqs
8000 , 02b6 , ae7d , 0189 , 374c ,      (  265 khz    freqs +  0
8400 , 044d , 013b , 0254 , 60aa ,      (  420 khz    freqs +  10
4800 , 072b , 020c , 03c3 , 6113 ,      (  700 khz    freqs +  20
6c00 , 0b43 , 9581 , 05cf , aace ,      ( 1100 khz    freqs +  30
7000 , 12f1 , a9fc , 09a6 , b50b ,      ( 1850 khz    freqs +  40
9400 , 1f21 , 2d77 , 0fbe , 76c9 ,      ( 3040 khz    freqs +  50

(  companion table of xmit freqs in khz

decimal
create khz
265 , 420 , 700 , 1100 , 1850 , 3040 ,
hex
decimal
```

Gains in the receiver can be set at 1, 2, 4, 8, and 16 X. Adjusting these values will change the receiving sensitivity by as much as 24 dB. The original values were selected to adjust the system gains to resemble earlier TAPS systems gain distributions across the frequencies.