CHAPTER 7
DISCUSSION

This chapter summarizes the decoder board hardware/software development, describes the anomalies, and provides recommendations for further work.

7.1 Summary

The decoder board is an integral device for linking the Consumer Control (CC) subsystem and the Host subsystem. In brief, the decoder board decodes Intermediate Frequency (IF) spread spectrum messages and then serially transmits these messages to the WMI. To perform this function, the following hardware components are used on the decoder board: Microcontroller, Spread Spectrum Decoder (SSD), Programmable Array Logic, Analog-to-Digital Converter, Random Access Memory, Read-Only Memory (ROM), D-Latch, and Serial-to-Parallel Shift Register (SPSR). The two most important components are the microcontroller and the SSD. The SSD downconverts, despreads, and demodulates the IF spread spectrum messages, and the microcontroller initializes and enables all peripherals, and packetizes the messages for transfer to the WMI.

Since there are four receiver/decoder pairs in the Cell Repeater (CR) subsystem, each decoder board has its own program stored in ROM. The programs are identical except for the initialization values for the radio receiver synthesizers. Each radio receiver gets tuned to one of the following carrier frequencies: 906, 912, 918, or 924 MHz. No two radio receivers should be tuned to the same carrier frequency. Therefore, the programs initialize their respective synthesizers so that the radio receivers are tuned to one of the above carrier frequencies.

The decoder board program can be divided into four main parts: Main Routine, Serial Interrupt Service Routines (ISR), Timer 2 ISR, and Timer 0 ISR. When power is applied to the decoder board, the microcontroller executes the main routine. The main routine configures the microcontroller for use in IVDS by initializing its stack pointer, buffer pointers, timer control registers, peripherals, user variables, interrupts, and watchdog timer. After this initialization is complete, the microcontroller waits for interrupts.

When a serial interrupt occurs, the microcontroller either executes the Transmit ISR or the Receive ISR depending on the state of the serial interrupt status bit. The Transmit ISR retrieves a data byte from the Transmit buffer, initiates serial transmission of the byte, and updates the Transmit buffer out-pointer. The Receive ISR retrieves a data byte from the serial port register, and either stores it to the Receive buffer or performs some action based on the byte value. If a Timer 2 interrupt occurs, then a timeout has occurred between the microcontroller and the SSD. The Timer 2 ISR re-initializes the radio receiver and the SSD, and resets all user variables so that the microcontroller can receive a new IVDS message.

The essential part of the decoder board program is the Timer 0 ISR. If a Timer 0 Interrupt occurs, then the SSD has signaled the microcontroller that data is available at the output of the SPSR. After the microcontroller has received the Lock and Header bytes, the Timer 0 ISR receives seven bits of data, bit-stuffs the MSB to 1, and then packetizes the bytes for serial transmission to the WMI. A packet has to conform to the Condensed Address Packet Protocol (CAPP) and also satisfy the HC05 buffer limitations. Consequently, each packet has to have a start of transmission byte (02h), an end of transmission byte (04h), and its length limited to 32 bytes. In addition, if any data byte happens to be a CAPP control code, then the data byte must be preceded with a NULL byte.

Originally, the WMI application program accepted an IVDS message in two packet form so that a message would not exceed the buffer limitations. However, the two packet method increased the communication overhead between the decoder board and the WMI, which caused the decoder board to transmit IVDS messages at a slower rate than it was receiving them. To transmit an IVDS message in one packet, the Timer 0 ISR was modified to retrieve seven bits at a time and then bit-stuff the MSB to 1. The bit-stuffing algorithm ensured that no data bytes would have to be preceded with a NULL byte so that a bit-stuffed IVDS message could be contained in one packet that would always be less than 32 bytes.

During the development of the decoder board hardware and software, three major problems were encountered. The first problem occurred during verification of the SSD operation when using the Z2000 Evaluation Board and its application software. For some reason, the application software outputs approximately 4352 clock pulses before any SSD output data appears on a display device. Until this was discovered, it was very difficult to determine if the SSD was configured correctly for IVDS. The second major problem occurred during data transfer between the microcontroller and the SSD. The interrupt signal from the SSD to the microcontroller did not satisfy the timing requirements of the microcontroller. The low-level pulse width of the RXDRDYn interrupt signal had to be stretched so that the microcontroller would recognize an edge level triggered interrupt correctly. The last major problem occurred during a test to see if the decoder board could handle heavy message traffic. The test showed that the decoder board missed a significant number of messages that were sent to it. The problem turned out to be the excessive communication overhead incurred when the two packet scheme was used. As mentioned above, the one packet scheme which uses the bit-stuffing algorithm reduced the overhead so that decoder board could output messages faster than it was receiving them. Once the decoder board was no longer a bottleneck, no major problems were encountered thereafter.

7.2 Anomalies

After the decoder board could successfully receive messages from the RF transmitter and send these messages to the WMI, the first anomaly appeared. At random times, the

decoder board would transmit garbage messages continuously for a few seconds when the RF transmitter was not sending messages. This problem occurred even if the RF transmitter was powered off. At this stage of development, the decoder board application program was not checking for Lock and Header Bytes, and the CRC check was disabled. Therefore, any "message" that the SSD detected would get transmitted to the WMI.

Though not proven conclusively, the cause of this anomaly was that the receiver gain was too high, thus overdriving the A/D converter which caused the SSD to detect false messages. After the RF group appropriately adjusted the receiver gain, this anomaly rarely appeared, and after time, completely disappeared.

The second anomaly occurred when the SSD would detect false messages, and the microcontroller would find a Lock byte followed by a Header Byte in these false messages. Although it is normal for the SSD to occasionally detect false messages, the probability that the microcontroller should find the Lock and Header bytes in these false messages is quite low. However, a garbage message would randomly appear on the computer monitor at least once every two or three minutes.

The cause of this second anomaly was never conclusively found. But the problem may have been rooted in the software handshaking between the decoder board application program and the WMI application program. During a test when a 1000 messages were sent to the WMI, it was noticed that some messages were corrupted when the WMI application software displayed status information while the messages were also being displayed. Once the display of status information was disabled, the messages no longer were corrupted and at the same time, garbage messages no longer randomly appeared.

The last anomaly involves the bit-stuffing algorithm. After a message is bit-stuffed, the last five bits of the unstuffed message is contained in the last byte of the bit-stuffed message. To get these last five bits, the Timer 0 interrupt occurs after five RXC clock edges and then the Timer 0 ISR reads these bits from the SPSR. However, these last five bits are only correctly retrieved when the Timer 0 interrupt occurs after four RXC clock edges. Since a problem has not occurred, the cause of this anomaly is still not known.

7.3 Recommendations For Further Work

Currently, the decoder board application program decodes IVDS messages that have a fixed length. If the need should arise, the application program may be modified to decode messages that have variable length. This requires the use of the RXACTIVE signal that indicates the size of a message burst. RXACTIVE goes high immediately before the first data bit appears at the RXOUT pin and then immediately goes low once the last data bit appears. Consequently, the application program could count the number of bits received while RXACTIVE is high, search for the Lock and Header bytes to find the byte boundaries, and then send the packetized data to the WMI.

The drawbacks to decoding variable length messages are twofold. First, the WMI application program would have to be altered to account for the variable lengths. This will add additional processing time of the messages which may not be desirable. Secondly, the decoder board may lose messages if the processing time to search for byte boundaries is found to be excessive.

The cause of the third anomaly should be further investigated. Although this anomaly is not causing a problem, another problem during later stages of IVDS system development may result. Consequently, it would be prudent to clear this anomaly before other problems appear.

Lastly, it would be beneficial to enhance the decoder board software so that the WMI application software can send commands to the decoder board to tune the radio receivers to different carrier frequencies. This would remove the inconvenience of reprogramming the decoder ROMs each time the user wants to change the carrier frequencies. The easiest way to implement this feature would be to add a look-up table algorithm in the decoder board software. Then, the WMI application software could send a two byte command followed by the desired carrier frequency data to tune a radio receiver to a new carrier frequency. Likewise, it would also be beneficial to allow the WMI application software to send commands to change the register contents of the SSD. This would also remove the inconvenience of reprogramming the ROMs each time the user wants to change the configuration of the SSD.

7.4 Overall Accomplishments

The major accomplishments include the digital pulse stretcher circuit, the Timer 0 ISR, the Timer 2 ISR, and the use of external RAM as a storage buffer. The pulse stretcher circuit is important because it corrected a timing problem between the SSD and the DS80C320 microcontroller. Without the pulse stretcher circuit, the SSD could not correctly send data to the microcontroller.

The Timer 0 ISR is important because it allows the decoder board to send messages to the WMI faster than it takes to receive them from the radio receivers. Specifically, the Timer 0 ISR reduces the communication overhead between the microcontroller and the WMI. Without this overhead reduction, the decoder board would have been a bottleneck resulting in many messages lost during periods of heavy message traffic.

The Timer 2 ISR is important because it prevents the decoder board from getting hung-up if a message terminates abnormally. If a message terminates abnormally, the Timer 2 ISR will detect this after the timeout period and reset the decoder board to receive a new message. Otherwise, the decoder board would receive part of a message that terminated abnormally and then erroneously append part of the next message to the first message. Once this condition occurs, the decoder board would continuously do this, so it is imperative to have the Timer 2 ISR reset the decoder board if a timeout occurs.

The use of external RAM as a storage buffer enhanced the capability of the decoder board. The external RAM has significantly more memory than the internal RAM. By the modifying the original Transmit ISR to store messages to external RAM, the decoder board can store a greater number of backlogged messages if the WMI blocks data transmission from the decoder board.