CHAPTER 6
DECODER BOARD TEST AND EVALUATION

This chapter describes the tests used to evaluate the performance of the decoder board hardware and software. At each stage of development, a test was performed to determine if the decoder board output response was correct given a test input. Other tests were performed to determine the efficiency of the decoder board output response.

6.1 Overview

When Grayson Electronics delivered the prototype decoder board, only minimal hardware testing had been performed. Grayson Electronics verified that power was distributed to all components, that the microcontroller could execute instructions from ROM and RAM, and that the LEDs could be illuminated. Virginia Tech would have to check the remaining components for proper operation.

The decoder board application software was developed in a series of stages until the software was complete. This series of stages made it possible to test the decoder board components in a prudent manner. As each stage was completed, a particular component was tested for correct operation. Therefore, the decoder board application software development and the component testing were accomplished concurrently.

6.2 Light Emitting Diode Blink Test

The first stage was to simply blink an LED at a one second rate. To accomplish this task, the boot program was modified so that it would continuously blink the LED. At this time, a ROM programmer was not available so the modified program was downloaded into external data RAM. However, when the decoder board was powered-up, the LED did not blink. It was learned that the microcontroller is hardwired to automatically fetch instructions from ROM upon power-up. Therefore, the WMI application software had to be modified to command the microcontroller to fetch instructions from external data RAM. Once this was done, the LED began to blink upon power-up.

6.3 Serial Link Test

The second stage was to send simulated IVDS messages from the decoder board to the WMI. The strategy was to store some arbitrary test messages in external data RAM, and then have the microcontroller read, packetize, and send these messages to the WMI. The WMI would then forward these messages to a PC which would display them on the computer monitor. This would verify the serial link between the WMI and the decoder board.

At this stage of development, the WMI application software was designed to accept a message in the form of two packets due to the limited HC05 buffer size. The two packet method solved the problem that if every message byte were a CAPP control code, then

each packet would be less than 32 bytes. Assuming worst case that each byte is a CAPP control code and thus must be preceded with a NULL byte, packet 1 would have a maximum of 21 bytes, and packet 2 would have a maximum of 27 bytes. The form of the two packets are shown below assuming that no bytes are escaped with a NULL byte.

Packet 1: STX Dummy Byte [3 Lock Bytes] Header Byte [7 Data Bytes] EOT
Packet 2: STX Dummy Byte [10 Data Bytes] CRC Byte CRC Byte EOT

As mentioned before, the dummy bytes are needed since these bytes get overwritten with the decoder board number. The dummy bytes for packets 1 and 2 were C0h and D0h, respectively. The WMI application program used these bytes to distinguish between the two packets and to establish the byte boundaries. Once the WMI received the two packets, only the data and CRC bytes were processed; the others were discarded.

Originally, ten different test messages were stored in external data RAM. The microcontroller continuously retrieved the test messages from RAM and sent them to the WMI. A one second delay was inserted between the packets to prevent HC05 buffer overflow. Also, the message bytes were ASCII characters so that they could be displayed on the computer monitor.

Once the test messages were successfully displayed, the CRC check was disabled so that any test message would be displayed. This removed the inconvenience of having to calculate the CRC bytes for every message sent to the WMI, and also aided troubleshooting if messages got corrupted. After disabling the CRC check, a test message of 23 ASCII A's (41h) was used, which is shown below in two packet form.

Packet 1: 02 C0 41 41 41 41 41 41 41 41 41 41 41 04
Packet 2: 02 D0 41 41 41 41 41 41 41 41 41 41 41 41 04

Additionally, the WMI application program was modified so that each message would be displayed on a separate line. Thus, if fifty test messages were sent, then fifty lines would have 23 A's displayed.

6.4 Interrupt Link Test

The next stage was to verify the interrupt link between the SSD and the microcontroller. The program was now modified to configure the Timer 0 interrupt to occur after eight RXDRDYn pulse edges, and to configure the SSD for infinite burst mode. The Timer 0 ISR would read a message byte from external data RAM and write it to another buffer in internal data RAM. After 23 bytes were written to the internal RAM buffer, the message was retrieved from the internal RAM buffer and sent to the WMI. The program continuously sent the test message.

To configure the SSD for infinite burst mode, the register value at address 30h was set to 60h. This caused the SSD to output the RXDRDYn signal as a continuous 40 KHz periodic pulse which made it convenient to use as an interrupt signal for test purposes.

6.5 Serial-To-Parallel Shift Register Test

Once the Timer 0 Interrupt was verified, the next stage was to verify the SPSR operation. For this test, a Motorola 68HC11 microcontroller was used to provide the data and clock inputs to the SPSR as well as the interrupt signal to the DS80C320. In other words, the 68HC11 would simulate the operation of the SSD.

To interface the 68HC11 to the decoder board, the SSD had to be isolated from the DS80C320 and the SPSR. To isolate the SSD, three signal traces on the decoder board were cut: RXDRDYn input to DS80C320, RXOUT input to SPSR, and SRCLK input to SPSR. Then, the appropriate 68HC11 signals were used.

The Serial Peripheral Interface (SPI) subsystem of the 68HC11 made it convenient to simulate the SSD operation.[5][7][8] The SPI subsystem provides a serial data line and a clock signal to shift out the data bits. Figure 6.1 shows the hardware connections to interface the 68HC11 to the decoder board. As the master device, the 68HC11 serially transmitted data bits to the decoder board at 62.5 KHz, using the Master-Out Slave In (MOSI) port for the data bits and the Serial Clock (SCK) for the timing signal. Although the 68HC11 transmitted the data bits faster than the SSD, 62.5 KHz was sufficient for verifying SPSR operation. The 68HC11 continuously sent the test message of 23 A's with sufficient delay between the messages to prevent HC05 buffer overflow.

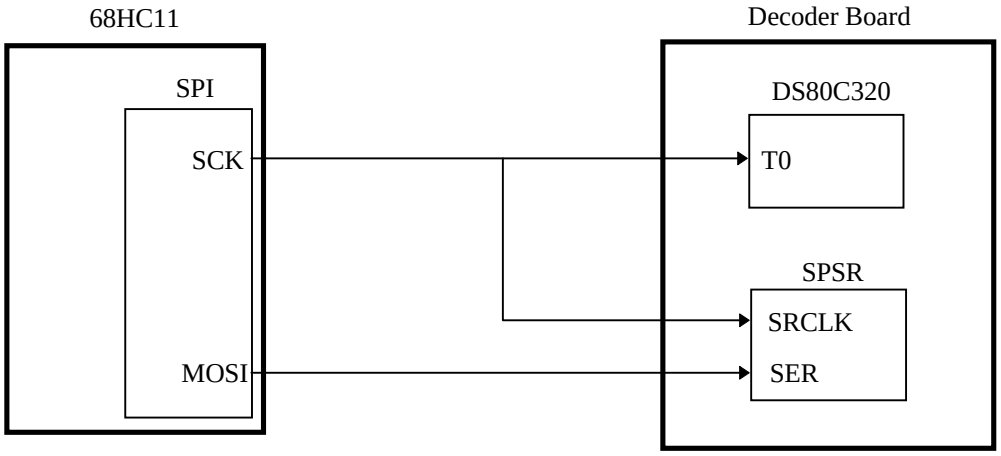

Figure 6.1 Motorola 68HC11 and Decoder Board Interface Block Diagram

Similar to the programs before, the Timer 0 interrupt would occur after counting eight clock pulses from the 68HC11. However, the Timer 0 ISR would now read the SPSR output and write the message byte to the internal RAM buffer. After 23 bytes were written, the DS80C320 would send the message to the WMI. The WMI then sent the message to the PC which displayed it on the computer monitor.

After SPSR operation was verified, all the hardware components except the SSD had been tested for correct operation, and the interface between the decoder board application software and the WMI application software had also been tested for correct operation. This was significant so that problems encountered in later stages of development could be attributed to the SSD, RF receiver, or RF transmitter.

6.6 Spread Spectrum Decoder Test

The next stage was to verify SSD operation. Since Virginia Tech had little experience with the SSD, a Z2000 Spread Spectrum Development Kit was used to aid SSD development. The kit provided a Z2000 Evaluation Board which included a Z2000 circuit chip, and PC software to allow easy programming of the SSD control registers. In conjunction with the Evaluation Board, a Z2000 Loopback Board provided the functionality required to test the Z2000 Evaluation Board's RF section interface in a loopback mode. The combination of the Evaluation Board and the Loopback Board made it convenient to program the Z2000 control registers and output test data. A detailed description for setting up the Evaluation Board and Loopback Board is provided in the development kit user's manual.[13]

At first, the control registers were programmed with values that the RF Group considered applicable for IVDS. Using the development platform, a test message of 23 A's was generated so that the RXOUT data could be displayed on a Tektronix 318 Logic Analyzer. The logic analyzer has four 8-channel data probes designated as Pods A - D, one external clock input probe, and 256 bits per channel for memory.

To display the data on the logic analyzer, Channel 0 of Pod A and the clock probe were connected to pins RXOUT and /RXDRDY, respectively, on the evaluation board. Figures 6.2 - 6.4 show the logic analyzer configuration menus with the appropriate settings for capturing data. After these menus are configured, the user presses the START button and then sends the test message. The captured data is viewed in either the State Table or Timing Diagram menus. A complete description on how to configure the menus, to capture data, and to view data is provided in the operator's manual.[9]

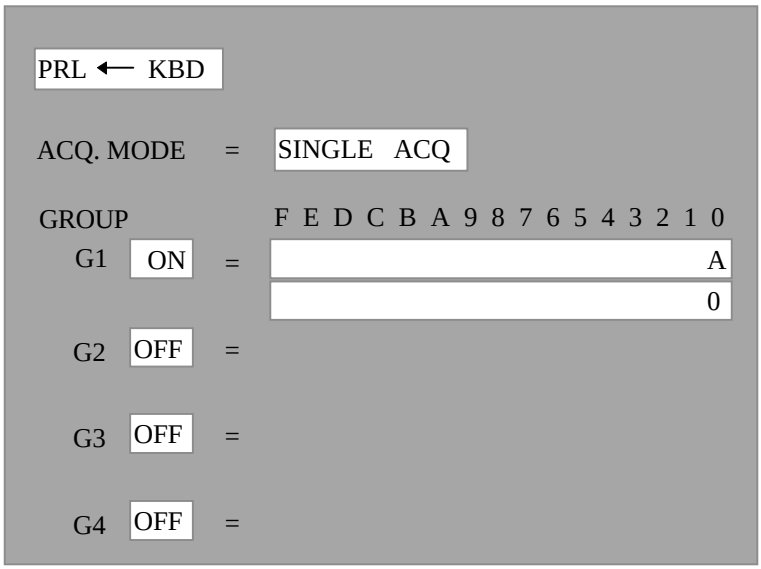

Figure 6.2 Logic Analyzer Setup Menu Settings for Capturing Bits

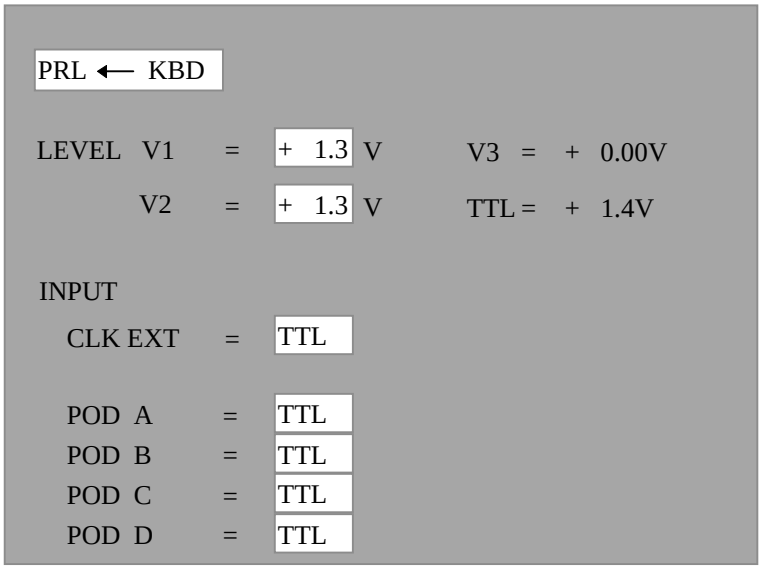

Figure 6.3 Logic Analyzer Threshold Menu Settings

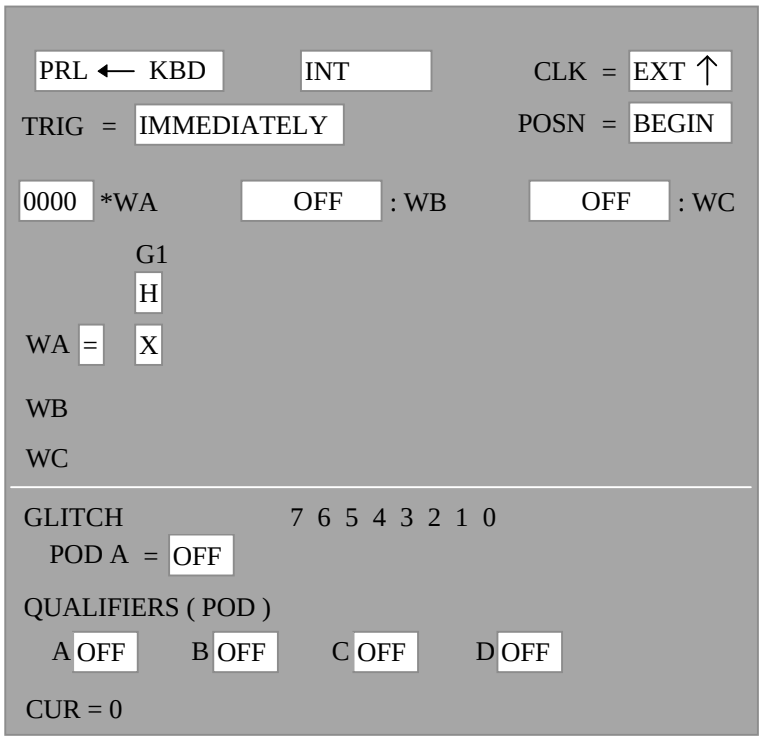

Figure 6.4 Logic Analyzer Trigger Menu Settings for Capturing Bits

The first attempts at configuring the SSD to output the test message were unsuccessful. Surprisingly, the logic analyzer always displayed a logic 1 for all 256 bits of memory, instead of the 184 bits that comprise 23 A's. On the assumption that the logic analyzer was improperly configured, a storage oscilloscope was used. The oscilloscope also showed the same results. To troubleshoot the problem, the test message was changed to a single byte containing 55h, which would appear as a square wave. Again, the logic analyzer showed all 256 bits as a logic 1 instead of only showing the first eight bits as alternating 0's and 1's, and the other 248 bits as undefined bits. The storage oscilloscope also displayed the data as logic 1s. This problem persisted until it was discovered that if the oscilloscope triggered on the falling edge, the square wave was displayed. But, if the oscilloscope triggered on the rising edge, the logic 1 data was displayed.

To further troubleshoot the problem, a frequency counter was connected to the /RXDRDY pin on the evaluation board. Surprisingly, the counter showed that the Z2000 Evaluation Board transmitted approximately 8000 clock pulses for a single byte. It was discovered that regardless of the message size, approximately 8000 clock pulses were transmitted. Apparently, the Zilog Development Kit software was designed to output this large number of clock pulses, though the technical documentation does not address this. Consequently, after many iterations of configuring the logic analyzer to capture data after

a specified number of clock pulses, the square wave appeared after 4352 clock pulses. This was the cause of the analyzer always displaying logic 1 for all 256 bits when it captured data beginning with the first clock pulse.

After further analysis, it was discovered that the Development Kit software outputs a stream of logic 1s, and then has a logic 0 start bit after which the data appears. Once the data ends, the software then outputs the logic 1s again. Thus, an alternative way to display the data is to configure the logic analyzer to capture the data once the start bit appears. Figure 6.5 shows the Trigger Menu settings to capture data after the start bit.

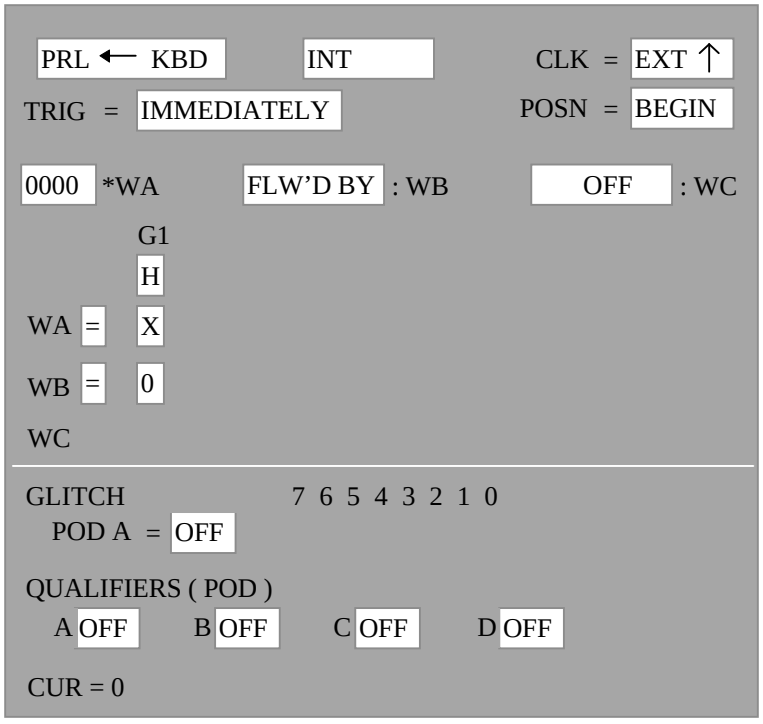

Figure 6.5 Logic Analyzer Trigger Menu Settings for Capturing Bits After Start Bit

6.7 Radio Frequency Link Test

Once the test message was displayed correctly on the logic analyzer, it was reasonably assured that the SSD was operating correctly. The next stage was to send a test message to the decoder board using the RF test transmitter and have the SPSR output displayed on the logic analyzer. At this stage of development, the program would read the SPSR output after eight clock pulses. Also, since the Timer 0 interrupt triggers on falling clock edges, the /RXDRDY signal was inverted before going to the Timer 0 interrupt pin.

The logic analyzer data probe was connected to the MAD0 - MAD7 pins on Test Port P2, and the clock probe was connected to the MPRDn pin, also on Test Port P2. Thus, when the Timer 0 ISR reads the SPSR, this read signal (MPRDn) triggers the clock probe so that the logic analyzer will capture the data byte on the test port. Figures 6.6 and 6.7 show the Setup Menu and Trigger Menu settings for the logic analyzer to read bytes; the other menu settings remained the same.

The RF test transmitter sent the customary test message of 23 A's. The logic analyzer displayed some interesting results. Instead of displaying all 23 bytes correctly, only a few of the bytes were displayed correctly. After several more attempts, the results were the same, except which bytes were displayed correctly was random. In other words, the interrupt signal was triggering the microcontroller to randomly read bytes correctly, which led to the conclusion that there was a timing problem between the SSD and the microcontroller.

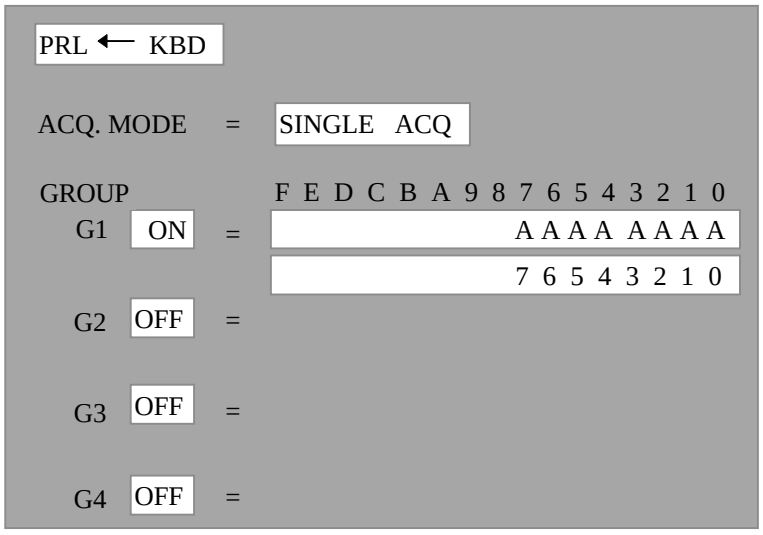

Figure 6.6 Logic Analyzer Setup Menu Settings for Capturing Bytes

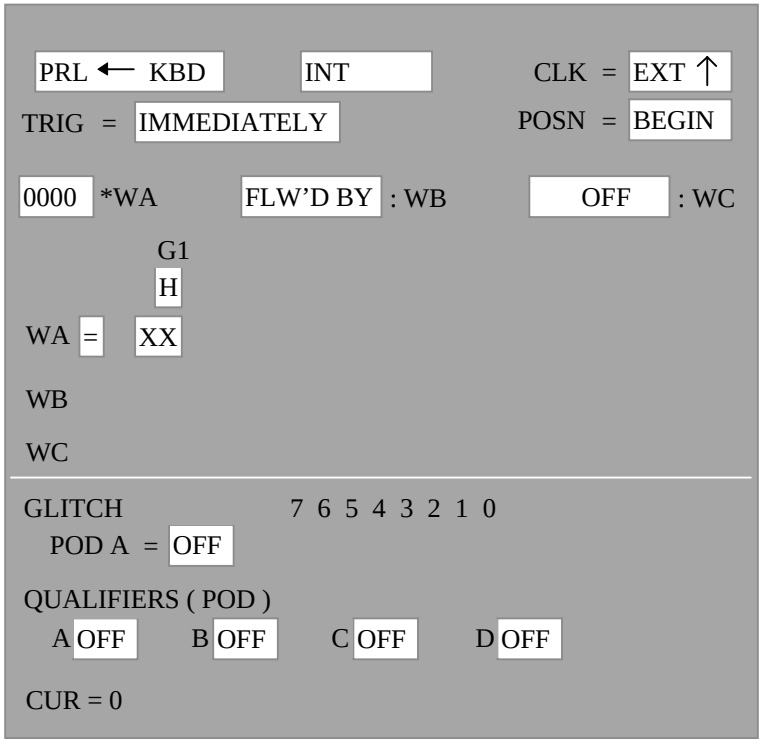

Figure 6.7 Logic Analyzer Trigger Menu Settings for Capturing Bytes

Referring to Figure 4.7, the 0.25 microsecond pulse width of RXDRDYn was found not to satisfy the DS80C320 hold time for edge triggered interrupts. The DS80C320 requires edge triggered interrupts to be held for at least one machine cycle, which equals four oscillator periods. At a clock rate of 11.059 MHz, four oscillator periods equals 0.362 microseconds.

To verify that this was indeed the problem, Figure 6.8 shows a test circuit used to stretch the pulse width of RXDRDYn. After implementing the pulse stretcher circuit, the logic analyzer displayed all 23 bytes correctly. Therefore, the digital clock pulse stretcher circuit in Figure 4.3 was developed and programmed into the PAL so that the interrupt hold time would be satisfied.

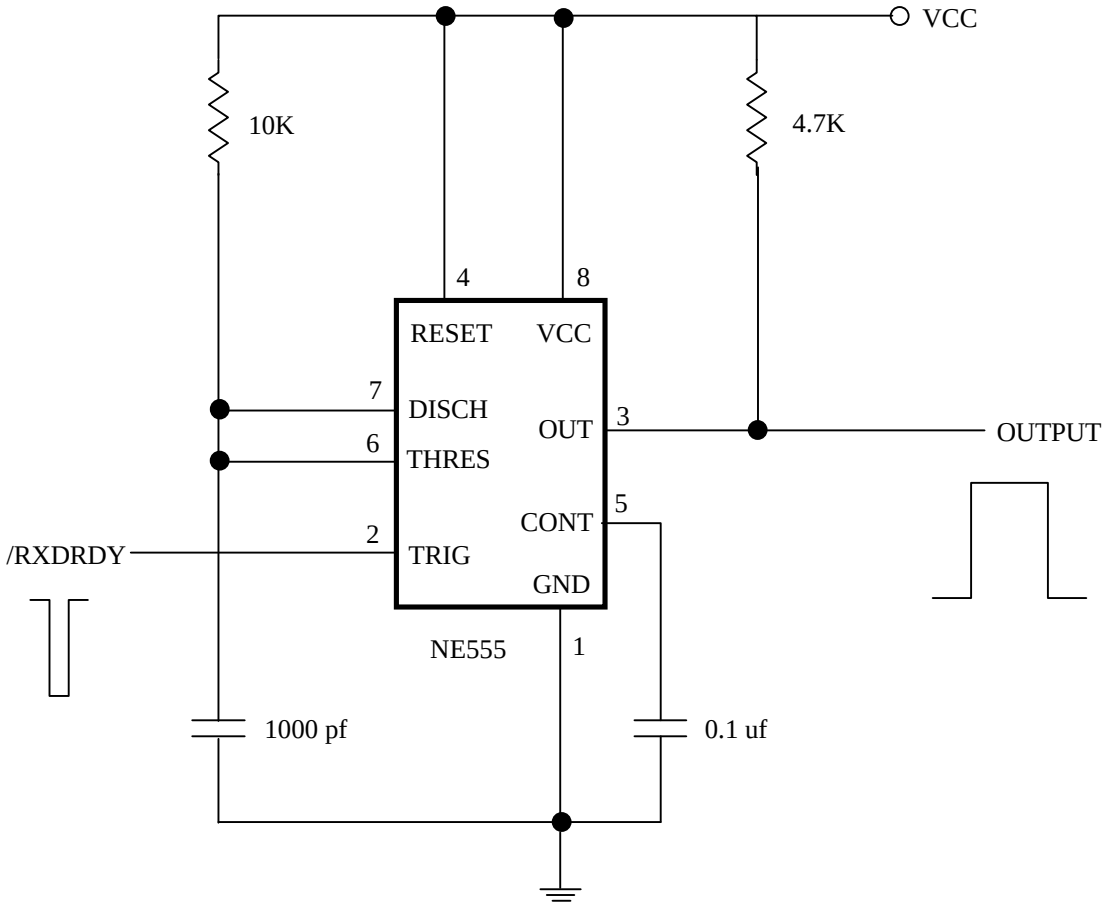

Figure 6.8 Analog Pulse Stretcher Circuit

6.8 Efficiency Tests

Once the interrupt timing problem was corrected, all the decoder board components were operating correctly. The next stage had the test transmitter send 1000 messages to be displayed on the computer monitor. This test would determine if the decoder board application software would process all the messages and send them correctly to the WMI. At this stage, the decoder board application software would receive a message, disable the Timer 0 Interrupt, and then send the message to the WMI. This was done to ensure that the Serial Transmit Interrupt and the Timer 0 Interrupt would not interfere with one another.

To allow sufficient time for the decoder board application software to process the 1000 messages, a 50 millisecond delay was inserted between messages. Before the messages were sent, the Telemate software was configured so that a log file would record all

messages received. After sending the messages, the log file showed that the decoder board received and sent all 1000 messages to the WMI.
To achieve minimal delay time needed between messages, the Timer 0 ISR was modified so that when a byte was received, it would immediately be sent to the WMI. This meant that the Timer 0 and Serial Transmit interrupts would overlap. When the 1000 messages were sent, the decoder board application program crashed after receiving the first few messages. After troubleshooting, the problem was found that some working registers were not saved before the Timer ISR executed. Unlike the 68HC11, the DS80C320 does not save working registers when an ISR is executed. The programmer must save these registers to the stack.

The last stage was to verify that the decoder board could indeed handle heavy message traffic. Therefore, the delay time between messages was reduced to five milliseconds. After a 1000 messages were sent, the log file showed that a significant number of messages were lost. At this stage of development, the two packet scheme was still being used. A timing analysis revealed that the decoder board was taking more time to send a message than it took to receive it. The problem was excessive overhead due to the two packet scheme.

The decoder board receives data bits at 40 KHz or one bit every 25 microseconds. Therefore, the time to receive a 23 byte (184 bits) message is 184 x 25 us which equals 4.6 milliseconds. The decoder board transmits data bits at 57.6 KHz or one bit every 17.36 microseconds. The two packet scheme adds two STX bytes, two EOT bytes, and two dummy bytes as overhead to the original 23 byte message. Also, the microcontroller adds a start bit and a stop bit for each byte transmitted. Consequently, there are 290 bits required to transmit the original message. With the added overhead, the time to transmit two packets is 290 x 17.36 us which equals 5.03 milliseconds. Obviously, this overhead had to be reduced so that the decoder board would send messages out faster than they arrive.

The overhead was reduced in two ways. First, the WMI application program does not need the Lock and Header bytes to process an IVDS message. Consequently, these bytes are discarded, leaving 19 bytes to be sent to the WMI. The next objective was to find a method to send the message as one packet. If none of the message bytes are control codes, then the message could be sent as one packet since the number of bytes would be less than 32. Since the control codes are less than 6h, any byte greater than this value would not be escaped. Consequently, bit-stuffing the message so that the MSB of each byte is set to 1 ensures that no data byte will be escaped with the NULL byte.

After bit-stuffing, the message length becomes 22 bytes. As one packet, only one STX byte, one dummy byte, and one EOT byte are added for a total of 25 bytes. Adding the start and stop bits, the total number of bits sent to the WMI is now 250 bits. The time to transmit the packet is 250 x 17.36 us which equals 4.34 milliseconds. Therefore, the

decoder board sends messages faster than they arrive, which eliminated the decoder board as the bottleneck.

6.9 Summary

A series of tests were performed to verify the operation of the decoder board components and to evaluate the performance of the software. Each test became more complex as the development of the software matured.

The LED blink test was the first test. This simple test verified that the microcontroller could execute a program that was downloaded into RAM. The only purpose of the program was to blink an LED at a one second rate.

The serial link test verified that the microcontroller could send a data packet to the WMI over the serial port. The main routine of the program continuously sent packets that contained 23 ASCII A's.

The interrupt link test verified that the SSD could interrupt the microcontroller. When the microcontroller counted eight clock pulses from the SSD, the Timer 0 ISR would read a byte from external RAM and store it to internal RAM. After 23 bytes were stored, the microcontroller sent the bytes in a packet to the WMI.

The SPSR test verified the operation of the SPSR. Using the 68HC11 to simulate the SSD, the microcontroller would retrieve data from the SPSR and send it to the WMI. Once this test was completed, the operation of all the decoder board hardware components were verified except the SSD.

The SSD test verified the operation of the SSD. This test verified that the data values loaded into the control registers of the SSD were appropriate for the IVDS system. This test was performed in stand-alone mode using the Z2000 development kit. The development kit software sent test data to the SSD, and the output of the SSD was displayed on a Logic Analyzer.

The RF link test verified that decoder board could get messages from the RF test transmitter via the receiver. The output of the SPSR was displayed on the logic analyzer. It was during this test that a timing problem between the SSD and the microcontroller was discovered. Consequently, the pulse stretcher circuit was needed to correct the problem. Once this test was completed, all the decoder board components were operating correctly.

The efficiency tests verified that the decoder board could handle heavy message traffic. These tests revealed the overhead problem when sending messages in two packet form, and the problem of some working registers not properly saved when an interrupt occurred. Once these problems were corrected in the software, the decoder board could receive and send messages to the WMI during heavy message traffic.