Session 2525

Integrated Circuit Chip Testing
Engineering Design Projects K-12

Bill Monaghan, Ph.D, P.E.
College of Staten Island CUNY

For the past three summers I have been associated with the Science Discovery Center at the College of Staten Island, City University of New York. This program is sponsored by the New York State Education Department under a Dwight D. Eisenhower Title II Project. Selected high school students have the opportunity of doing a project in various disciplines under the guidance of the faculty.

The program meets six hours a day, four days a week for four weeks. The participants give a poster presentation to the community at large on the final day. The challenge for me was the identification of engineering design projects that could be completed in the given time frame and that would have interest for the participants.

Overview

Based upon my experience in teaching a microcomputer **SDK-86** based interfacing course, a hardware/software design project was chosen. The SDK-86 allows easy access to the system bus. The necessary lines are cabled to a solderless breadboard design station. The presence of switches, pulsers and LEDs on the design station enables one to do manual validation of the hardware interface. These features facilitate rapid prototyping of the hardware interface. All input and output devices are treated as IO ports. The SDK-86 has on-board serial communication capability with an RS-232C port. Software can be developed on a PC and downloaded to the SDK computer. This feature allows rapid prototyping of the software component of the project. The question still remained: "What is a viable design project?".

SAGE: Smart Automated Gate Evaluator

The use of Automated Testing Equipment (**ATE**) is pervasive within the integrated circuit chip industry. It was decided that the participants were to design and implement a multi-chip chip tester. The quad two-input **7408 AND, 7432 OR** and **7400 NAND** chips are pin compatible. This permits the construction of a multi-chip chip tester. SAGE, the realization of the above, has been reported in the literature[1] and will be briefly summarized.

Hardware Considerations

A major component of the interface is the decoder for port addresses. A 74154 decoder is used in the implementation. This decoder has two enable pins, E1* and E2*, and will decode four address lines. Its wiring is shown in Figure 1. The output port addresses are asserted low. For the decoder to be enabled, the M/IO* input must be low. The port addresses are therefore in "IO space". The M/IO* signal is low when the software is executing either an IN or OUT instruction. Simultaneously both A15 and A0 must be low. The SDK-86 board has several IO ports. The addresses for all on-board ports have A15 high. A low in the SAGE design will avoid any possible conflicts with these ports.

The reason for A0 being low is more subtle. If A0 is low, all decoded addresses will be even. The 8086 processor is designed to sample the low portion of the data bus, D0 to D7, for the transfer of 8 bits of information when the port address is even. One final comment is in order. The above design for port address decoding is non-absolute. Many possible port addresses will appear equivalent to the decoder. This decision was predicated on the need for simplicity within the existing time constraints. The ramifications of non-absolute decoding were thoroughly discussed and compared to a full absolute decoding implementation.

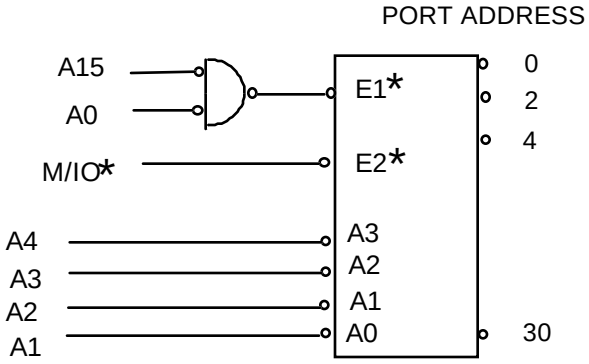

Figure 1
74154 Decoder

The port address 0 is chosen for both input and output in the SAGE implementation. Two 7475 level-sensitive latches are used to capture the test vector. A 74125 tri-state "electronic switch" isolates the results from the data bus. The unit under test (UUT) can be any of the pin-compatible chips. The completed SAGE hardware is given in Figure 2.

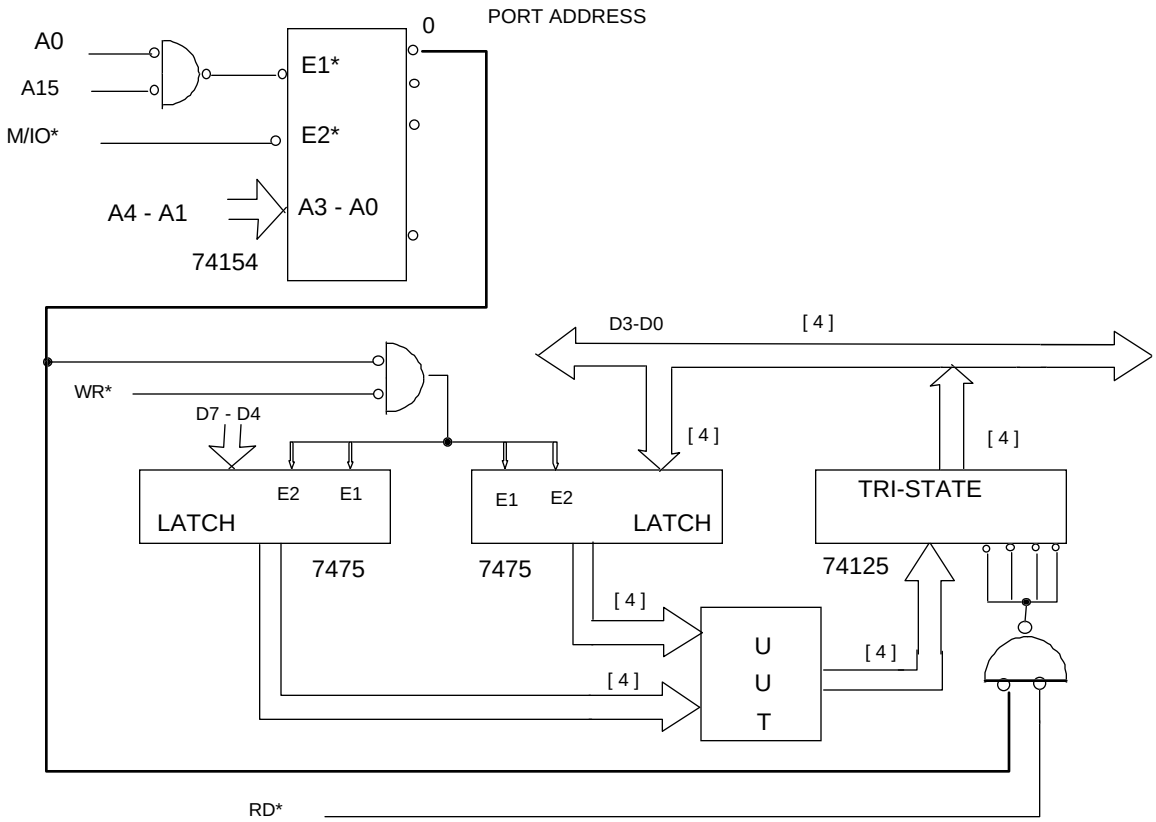

Figure 2
SAGE Realization

Software Considerations

The software is relatively straightforward. All chips are exercised with the same test vector, 11100100. A stored table of correct responses is constructed and used to compare the outputs of the UUT. Each table entry is one byte wide (8 bits) and contains the appropriate correct responses for the chip. For illustrative purposes consider the OR entry in the table: 01101110. The rightmost four bits, low nibble, is the correct response to 11100100. Three 2-bit right circular rotations of the test vector will generate all the remaining vectors needed for an exhaustive testing of a given UUT. Three 1-bit right circular rotations of the appropriate correct response vector will move the correct responses into the low nibble.

The IN and OUT instructions are the crucial components of the software. These instructions require that the address of an input or output port be placed in the DX register of the microprocessor prior to their execution. A typical sequence for an input is:

```
MOV   DX,0
IN    AL,DX
```

The DX register is 16 bits wide while AL is 8 bits. The semantics of IN can be stated as follows:

1. The content of DX is placed on the address lines A15...A0 and sent to the system bus.
2. A control signal M/IO* is asserted low and sent to the system bus.
3. The read control signal RD* is asserted low and sent to the system bus.
4. The microprocessor waits for signal stabilization and then samples the data bus.

Since the outputs of the tri-state device in Figure 2 are connected to the data bus, the above sequence will read these outputs into the AL register. Thus the outputs of the UUT are read and compared to the correct response vector.

The output sequence is quite similar to the input. Steps 1 and 2 are unchanged. Steps 3 and 4 become:

3. The microprocessor places the content of the AL register on the data bus.
4. After all signals stabilize, the processor issues the WR* signal.

This procedure will load the external 7475 latches with the contents of AL and this test vector is used as input to the UUT.

USReT: Universal Shift Register Tester

The 74194 is a 4-bit bidirectional Universal Shift Register (**USR**). Two of them can easily be cascaded to form a 8-bit register. The chip has an asynchronous master reset input, MR*. The operating mode of the register is determined by two inputs, s_1 and s_0, and is given by the following table:

S_1	S_0	Operating Mode
0	0	Hold
0	1	Shift Right
1	0	Shift Left
1	1	Parallel Load

A clock pulse input (active rising edge) performs the chosen operation. Dsr and Dsl inputs are used in shift operations. The parallel load inputs are designated Q_0 to Q_7, Q_0 being the leftmost bit. Since USReT was realized after successful completion of SAGE, the SAGE interface was modified as shown in Figure 2. Particular attention should be given to the various port addresses. Output port address 2 is used to perform a master reset of the USR and output address 4 clocks the USR. Output address 6 controls the 7475 latches and input port address 0 reads the USR.

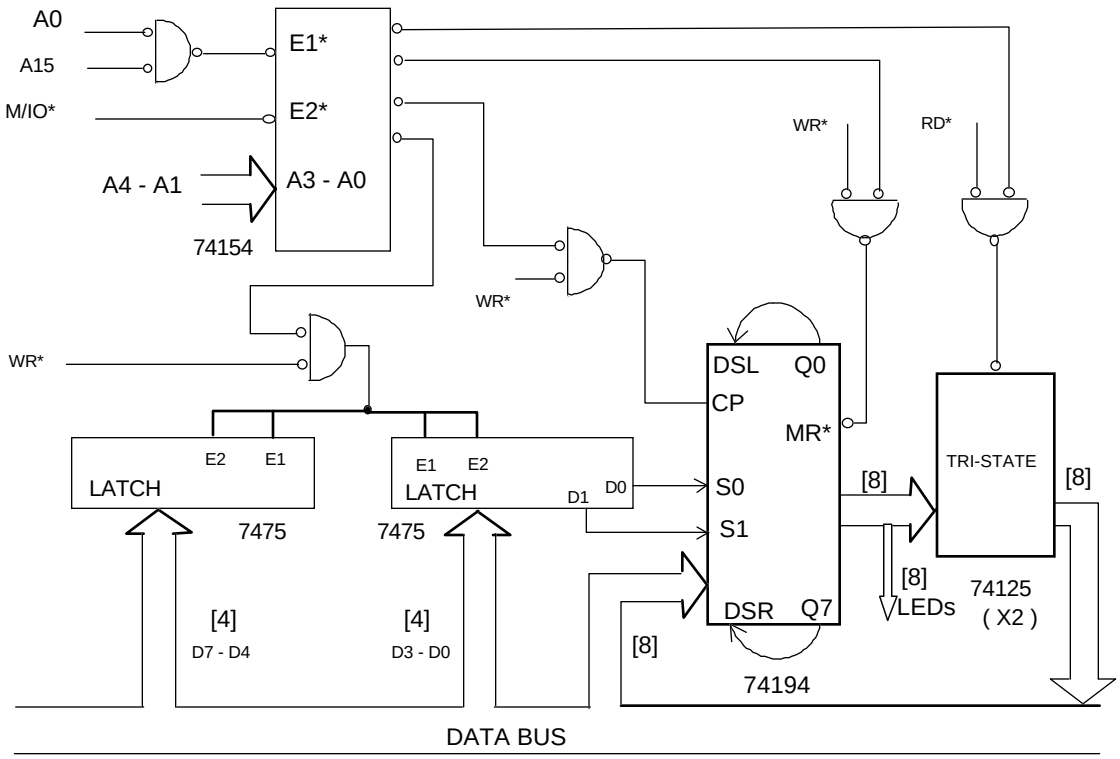

Figure 3
USReT ATE Interface

USReT ATE Interface

The current design utilizes the 7475 latch as a **mode register**. The D_1D_0 bits written to this register determine the operating mode of the USR. As an example, the following instructions will place the USR in parallel load mode:

MOV AL,3; MOV DX,6; OUT DX,AL.

If the following instructions are now executed, the USR will latch the 2EH datum:

MOV AL,2EH; MOV DX,4; OUT DX,AL.

At any time the latched contents of the USR may be read by executing:

MOV DX,0; IN AL,DX.

In a similar fashion a master reset of the USR is accomplished by executing:

MOV DX,2; OUT DX,AL.

A HOLD operation is achieved by writing a 0 to the mode register. The sequence of *a read, a write* followed by *another read of the USR* will not show any change in its stored value. Writing a 1 or a 2 to the mode register will configure the USR for shift operations. Dsr and Dsl are typically independent inputs to the USR; tying Q_0 to Dsl and Q_7 to Dsr produces circular shift operations. This is a useful arrangement for testing

purposes. Similar to the SAGE interface, this hardware can also be developed and tested in manual mode.

Software Considerations: Parallel Load Testing--Exhaustive Technique

A 3H is written to the mode register. A loop is then executed that performs *a write to the USR, a read* and then *a compare* between a copy of the written value and the read value for values between 0 and 0FFH inclusive. Any discrepancy indicates a faulty USR. By introducing timing loops in the software and connecting the outputs of the USR to LEDs, the progression of the tests can be monitored. Stuck-at-1 faults are easily emulated by disconnecting the appropriate lead. This exhaustive technique loops 256 times, a strategy **not** effective for memory cell testing of millions of 8-bit locations. This realization leads into a discussion and implementation of a time-efficient algorithm for parallel load operations.

Software Considerations: Parallel Load Testing--Walking 0/1 Technique

A common testing strategy is to write a field of zeroes and then walk a 1 through this field. The sequence of test values written to the USR is as follows:

00000000, 00000001, 00000010, 00000100
00001000, 00010000, 00100000, 01000000, 10000000.

Testing is performed for each value. A 0 is then walked through a field of ones. This version of testing , while not complete, is performed using 18 tests and in approximately 7% of the previous time. Both strategies are implemented in USReT; the choice of which to use is made at execution time.

Software Considerations: Data Shift , Hold and Master Reset Testing

The data value, 01010101, is written to the USR. The mode is then changed to a Dsr operation. Two shift operations are performed with testing after each shift. Because the wiring of the USR in the testbed produces circular rotations, a successful test indicates that each cell of the USR can effectively shift in both a 0 and a 1 from the right. The procedure is then repeated for a Dsl operation.

The Hold and Master Reset testing is relatively straightforward and requires no unusual strategies. At any stage a failed test identifies a faulty USR.

Conclusion

Since both assembler language programming and interfacing are new to the students, most find the SAGE project to be quite challenging. One participant moved very quickly through the design and implementation of SAGE and was the major inspiration to extend the UUT to a USR. Having gained experience with both projects, I believe the USReT tester to be the preferred choice. In either case, microcomputer-based hardware/ software projects are certainly viable for students in the 11th and 12th grades.

[1] Smart Automated Gate Evaluator, W. Monaghan, Proceedings of the Middle Atlantic Section Meeting, ASEE, 1994.

BILL MONAGHAN
Dr. Monaghan is Chair of the Department of Applied Sciences at the College of Staten Island, CUNY. His email address is monaghan@postbox.csi.cuny.edu.