Microcontroller + FPLD Designs with the XS40 & XS95 Boards

October 31, 2000 (Version 2.0)

Application Note by D. Vanden Bout

Summary

This application note introduces you to the design and test of 8031 microcontroller + FPGA/CPLD designs using the XS40 and XS95 Boards™. The material presented in this appnote assumes you are familiar with the following topics:

- designing logic using the XC4000 FPGA or XC9500 CPLD (*The Practical XILINX Designer Lab Book* from Prentice-Hall is a good introduction to this topic);
- the architecture and electrical characteristics of the 8031 microcontroller (the 8031 is a ROM-less version of Intel's 8051, so check www.intel.com for more information);
- writing and debugging programs for the 8031.

Microcontroller + FPLD Design Flow

The basic design flow for building microcontroller + FPLD applications is shown in Figure 1. (The designs in this appnote are applicable to both the CPLD in the XS95 Board and the FPGA in the XS40 Board, so we will use FPLD as a single acronym for both types of programmable devices.)

Initially you have to get the specifications for the system you are trying to design. Then you have to determine what inputs are available to your system and what outputs it will generate.

At this point, you have to partition the functions of your system between the microcontroller and the FPLD. Some of the input signals will go to the microcontroller, some will go to the FPLD, and some will go to both. Likewise, some of the outputs will be computed by the microcontroller and some by the FPLD. There will also be some new intra-system inputs and outputs created by the need for the microcontroller and the FPLD to cooperate.

In general, the FPLD will be used for low-level functions where signal transitions occur more frequently and the control logic is simpler. A specialized serial transmitter/receiver would be a good example. Conversely, the microcontroller will be used for higher-level functions where the responses occur less quickly and the control logic is more complex. Reacting to commands passed in by the receiver is a good example.

Once the design has been partitioned and you have assigned the various inputs, outputs, and functions to the microcontroller and the FPLD, then you can begin doing detailed design of the software and hardware. For the software, you can use your favorite editor to create a .ASM assembly-language file and assemble it with asm51 to create a .HEX object file for the 8031 microcontroller on the XS Board. For the FPLD hardware portion, you will draw schematics and store them in .SCH files that are compiled it into a .BIT or .SVF bitstream file using the XILINX Foundation programming software. With the 8031 object file and the FPLD bitstream file in hand, you can download them to the XS Board using the XSLOAD program. XSLOAD stores the contents of the .HEX file into the 32 KByte RAM on the XS Board (128 KByte RAM on the XS+ Board) and then it reconfigures the FPLD by loading it with the bitstream file.

After the XS Board is loaded with the hardware and software, you need to test it to see if it really works. The answer usually starts as "No" so you need a method of injecting test signals and observing the results. XSPORT is a simple program that lets you send test signals to the XS Board through the PC parallel port. You can trace the reaction of your system to signals from the parallel port by programming the microcontroller and the FPLD to output status information on the LED digit (much like placing `printf` statements in your C language programs). This is admittedly crude but will serve if you don't have access to programmable stimulus generators and logic analyzers.

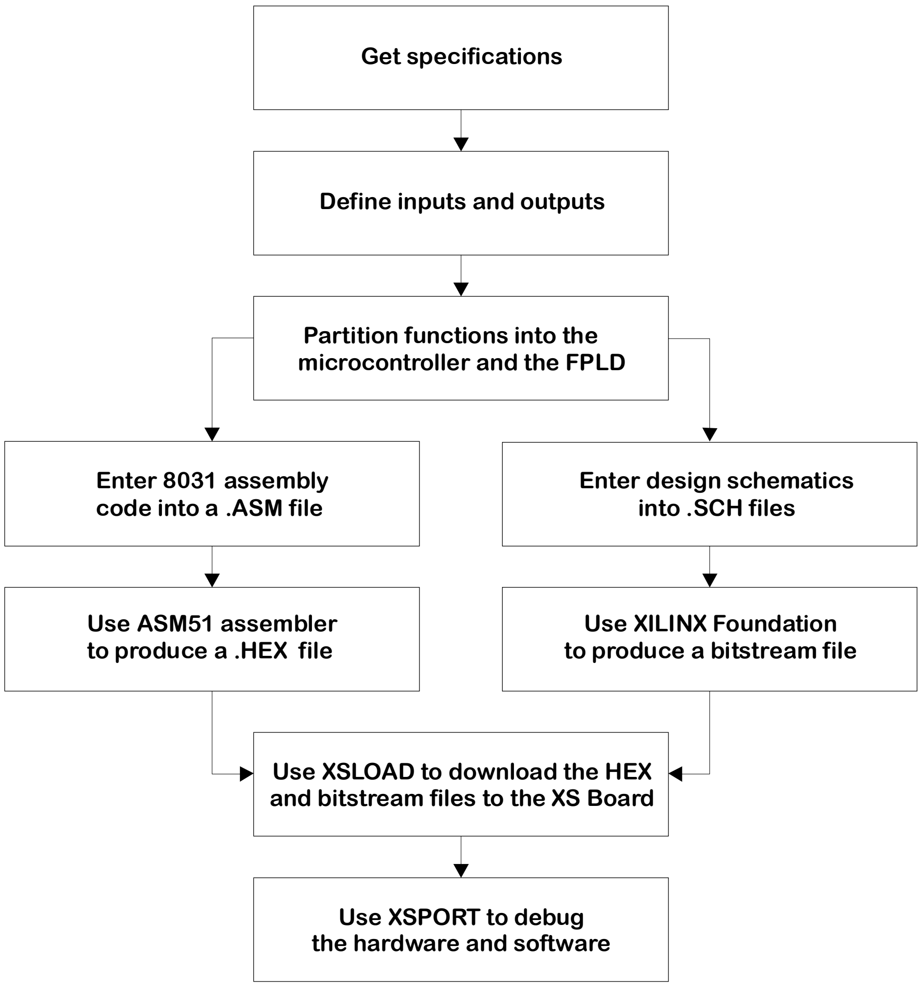
Get specifications
Define inputs and outputs
Partition functions into the microcontroller and the FPLD
Enter 8031 assembly code into a .ASM file
Enter design schematics into .SCH files
Use ASM51 assembler to produce a .HEX file
Use XILINX Foundation to produce a bitstream file
Use XSLOAD to download the HEX and bitstream files to the XS Board
Use XSPORT to debug the hardware and software

Figure 1: Microcontroller + FPLD design flow.

Microcontroller+FPLD Interconnections

The 8031 microcontroller and the FPLD on the XS Board are already connected together. These existing connections save you the effort of having to wire them yourself, but they also impose limitations on how your program and the FPLD hardware will interact. A high-level view of how the microcontroller, RAM, and FPGA on the XS40 Board are connected is shown in Figure 2. A similar view for the CPLD connections of the XS95 Board is shown in Figure 3.

The 100 MHz programmable oscillator output goes directly to a synchronous clock input of the FPLD. The FPLD processes the clock and retransmits it to the **XTAL1** input of the microcontroller.

The 8031 multiplexes the lower eight bits of a memory address with eight bits of data and outputs this on its **P0** port. Both the RAM data lines and the FPLD are connected to **P0**. The FPLD is configured to latch the address from **P0** under control of the **ALE** signal and send the latched address bits to the lower eight address lines of the RAM. Then the RAM and 8031 use the bus to exchange data.

Meanwhile, the upper eight bits of the address are output on port **P2** of the 8031. The 32 KByte RAM uses the lower seven of these address bits. (The 128 KByte RAM on the XS+ Boards use all eight bits.) The FPLD also receives the upper eight address bits and decodes these along with the **PSEN**, **RD** and **WR** control lines from the 8031 to generate the **CE** and **OE** signals that enable the RAM and its output drivers, respectively. Either of the **CE** or **OE** signals can be pulled high to disable the RAM and prevent it from having any effect on the rest of the XS Board circuitry.

An output from the FPLD controls the reset line of the microcontroller. The 8031 can be prevented from having any effect on the rest of the circuitry by forcing the **RST** pin high through the FPLD. (When **RST** is active, most of the 8031 pins are weakly pulled high.)

Many of the I/O pins of ports **P1** and **P3** of the 8031 connect to the FPLD and can be used for general-purpose I/O between the microcontroller and the FPLD. In addition to being general-purpose I/O, the **P3** pins also have special functions such as serial transmitters, receivers, interrupt inputs, timer inputs, and external RAM read/write control signals. If you aren't using a particular special function, then you can use the associated pin for general-purpose I/O between the microcontroller and the FPLD. In many cases, however, you will program the FPLD to make use of the special-purpose 8031 pins. (For example, the FPLD could generate an interrupt for the 8031.) If you want to use the special-purpose pin with an external circuit, then the FPLD I/O pin connected to it must be tristated.

An LED digit connects directly to the FPLD. The FPLD can be programmed so the microcontroller can control the LEDs either through **P1** or **P3** or by memory-mapping a register for the LED into the memory space of the 8031.

The PC can transmit signals to the XS Board through the eight data output bits of the printer port. The FPLD has direct access to these signals. The microcontroller can also access them if the FPLD is configured to pass the data output bits onto the FPLD I/O pins connected to the 8031.

Communication from the XS Board back to the PC also occurs through the parallel port. Four of the parallel port status pins are connected to three pins of **P1** and one pin of **P3**. Either the microcontroller or the FPLD can drive the status pins. The PC can read the status pins to fetch data from the XS Board.

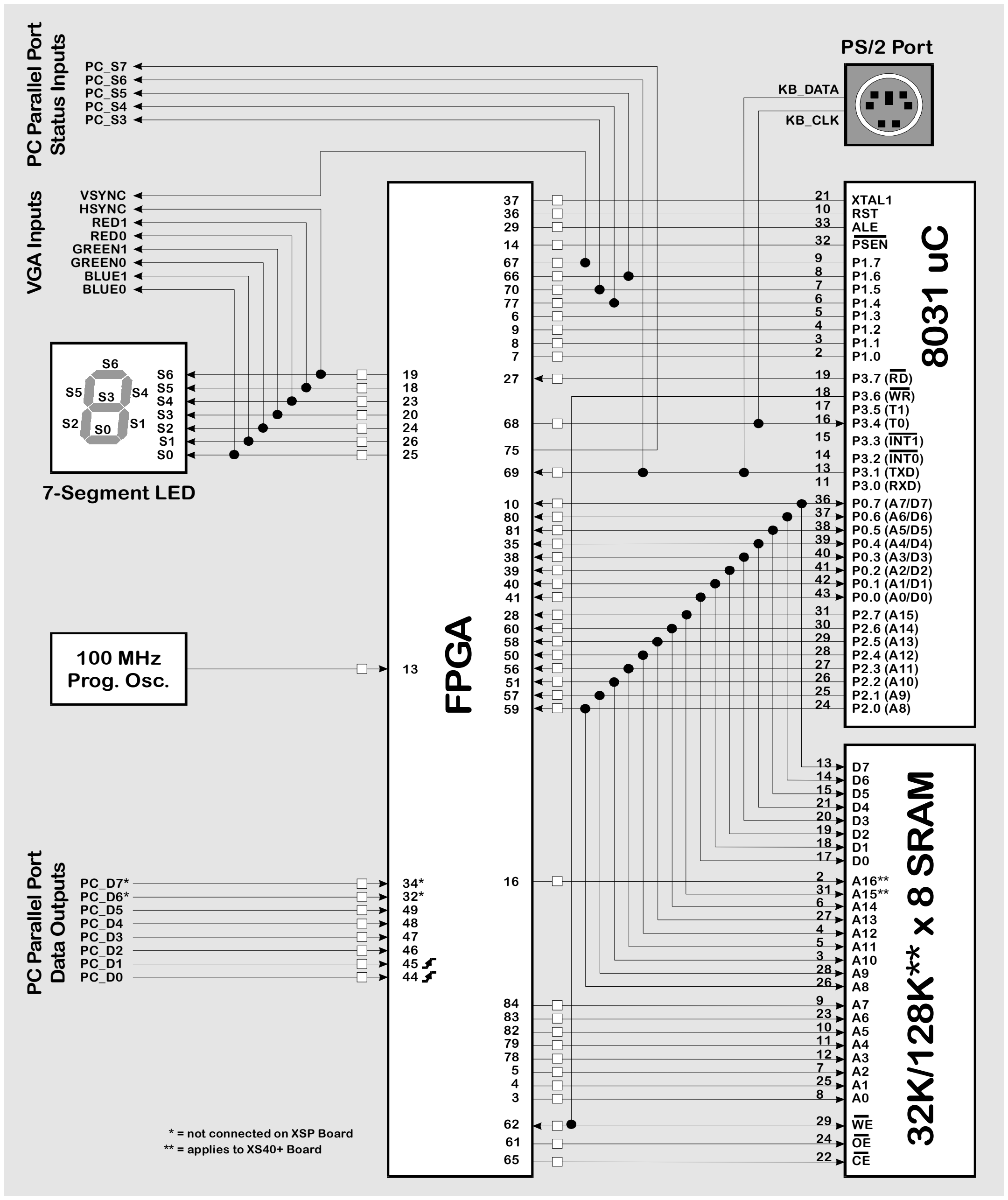

Figure 2: Connections between the 8031 microcontroller, RAM, and FPGA of the XS40 Board.

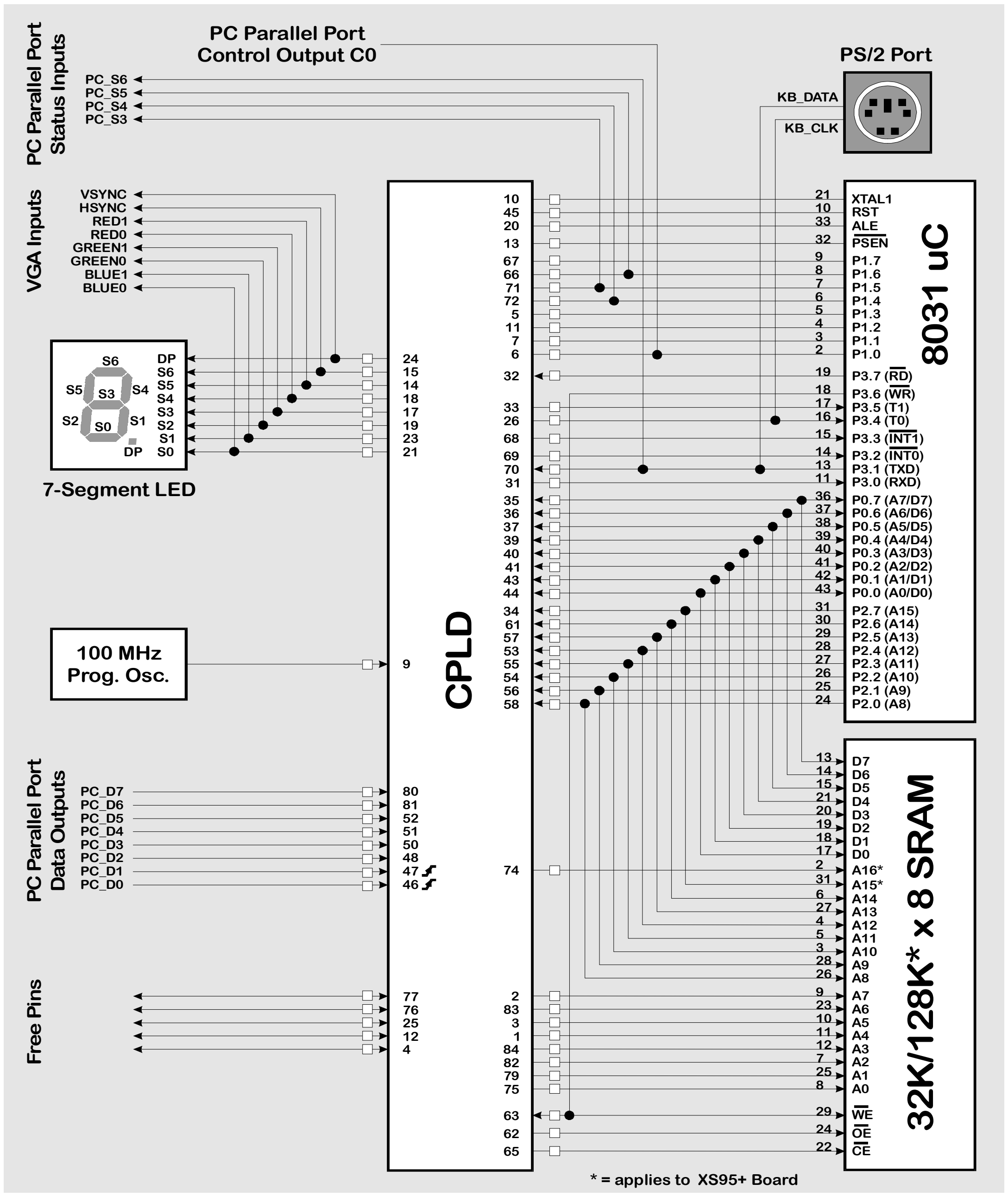

Figure 3: Connections between the 8031 microcontroller, RAM, and CPLD of the XS95 Board.

User-Constraint Files for the XS40 and XS95 Boards

The `UINFC40.UCF` file shown in Listing 1 details how the pins of the XC4000 FPGA connect to the other components on the XS40 Board. The `UINFC95.UCF` file shown in Listing 2 details how the pins of the XC9500 CPLD connect to the other components on the XS95 Board. (The line numbers in each listing are provided for explanatory purposes only. They are not part of the actual file contents.) Here is a line-by-line description of what is contained in these user-constraint files:

Line 1: The 50 MHz clock (this is the default frequency) from the external programmable oscillator enters through this pin.

Lines 4–11: These pins are outputs which control the individual segments of the LED digit. Note that the decimal-point segment (`LED<7>`) is not connected on the XS40 Board.

Lines 14–21: The eight data bits from the PC parallel port enter the FPLD through these pins. For the XS40 Board, the upper-most two data bits (`PC_D<6>` and `PC_D<7>`) cannot be specified in a user-constraints file because these are special-purpose pins. To access these data bits on the XS40 Board you will have to use the special-purpose schematic symbols **MD0** and **MD2**.

Line 24: The FPLD sends a clock signal to the 8031 clock input through this pin.

Line 25: The FPLD controls the active-high reset input of the 8031 through this pin.

Line 26: The FPLD monitors this pin for a falling edge which indicates it should latch the lower eight bits of an address from the multiplexed address/data bus of the 8031.

Line 27: The FPLD monitors this pin for a low level which indicates the 8031 is accessing the program segment of memory.

Line 28: The FPLD monitors this pin for a low level which indicates the 8031 is reading a byte from the data segment of memory.

Line 29: The FPLD monitors this pin for a low level which indicates the 8031 is writing a byte to the data segment of memory.

Lines 30–37: The multiplexed address/data bus of the 8031 appears on these pins of the FPLD. The FPLD can latch the lower eight bits of the address on the falling edge of `ALE_`. After that, a byte of data will appear on these pins.

Lines 38–45: The FPLD can output the latched lower-byte of the address on these pins which are also connected to the address pins of the external RAM.

Lines 46–53: The FPLD monitors the upper-byte of the address output by the 8031 on these pins uses it to generate chip-selects based upon the address values it sees. These address bits also serve as the upper address byte for the external RAM. (The XS Board with 32 Kbyte RAM only uses seven of the upper address bits while the XS+ Board with 128 KByte RAM uses all eight).

Line 54: The FPLD outputs the most-significant address bit for the XS+ Board 128 KByte RAM on this pin. This pin is not used for the XS Boards with 32 KByte RAMs.

Line 57: The FPLD enables the output drivers of the external memory by placing a low level on this pin.

Line 58: The FPLD enables the external memory by placing a low level on this pin.

Lines 61-66: The CPLD on the XS95 Board has six free pins that aren't connected to anything else, so they are listed at the end of Listing 2. The pins of the XC4000 FPGA on the XS40 Board are all used so there are no entries for free pins in the `UINFC40.UCF` file.

Listing 1: User-constraint file for the XS40 Board (UINFC40.UCF).

```
NET CLK             LOC=P13; # CLOCK FROM EXTERNAL OSCILLATOR
#
# LED DRIVER OUTPUTS
NET LED<0>    LOC=P25;
NET LED<1>    LOC=P26;
NET LED<2>    LOC=P24;
NET LED<3>    LOC=P20;
NET LED<4>    LOC=P23;
NET LED<5>    LOC=P18;
NET LED<6>    LOC=P19;
//NET LED<7>  LOC=P30; # THERE IS NO CONNECTION TO THE DECIMAL-POINT
#
# DATA BITS FROM THE PC PARALLEL PORT
NET PC_D<0>   LOC=P44;
NET PC_D<1>   LOC=P45;
NET PC_D<2>   LOC=P46;
NET PC_D<3>   LOC=P47;
NET PC_D<4>   LOC=P48;
NET PC_D<5>   LOC=P49;
//NET PC_D<6> LOC=P32; # MUST USE SPECIAL-PURPOSE PINS FOR
//NET PC_D<7> LOC=P34; #   ACCESSING PC_D<6> AND PC_D<7>
#
# MICROCONTROLLER PINS
NET XTAL1     LOC=P37; # INPUT CLOCK
NET RST       LOC=P36; # ACTIVE-HIGH RESET
NET ALE_      LOC=P29; # ACTIVE-LOW ADDRESS LATCH ENABLE
NET PSEN_     LOC=P14; # ACTIVE-LOW PROGRAM-STORE ENABLE
NET RD_       LOC=P27; # ACTIVE-LOW READ
NET WR_       LOC=P62; # ACTIVE-LOW WRITE (ALSO CONTROLS RAM)
NET AD<0>     LOC=P41; # MULTIPLEXED ADDRESS/DATA BUS
NET AD<1>     LOC=P40;
NET AD<2>     LOC=P39;
NET AD<3>     LOC=P38;
NET AD<4>     LOC=P35;
NET AD<5>     LOC=P81;
NET AD<6>     LOC=P80;
NET AD<7>     LOC=P10;
NET A<0>      LOC=P3;  # DEMUXED LOWER BYTE OF ADDRESS
NET A<1>      LOC=P4;
NET A<2>      LOC=P5;
NET A<3>      LOC=P78;
NET A<4>      LOC=P79;
NET A<5>      LOC=P82;
NET A<6>      LOC=P83;
NET A<7>      LOC=P84;
NET A<8>      LOC=P59; # UPPER BYTE OF ADDRESS
NET A<9>      LOC=P57;
NET A<10>     LOC=P51;
NET A<11>     LOC=P56;
NET A<12>     LOC=P50;
NET A<13>     LOC=P58;
NET A<14>     LOC=P60;
NET A<15>     LOC=P28;
NET A16       LOC=P16; # MS address bit of the 128K RAM
#
# RAM CONTROL PINS
NET OE_       LOC=P61; # ACTIVE-LOW OUTPUT ENABLE
NET CE_       LOC=P65; # ACTIVE-LOW CHIP ENABLE
```

Listing 2: User-constraint file for the XS95 Board (UINFC95.UCF).

```
NET CLK             LOC=P9; # CLOCK FROM EXTERNAL OSCILLATOR
#
# LED DRIVER OUTPUTS
NET LED<0>    LOC=P21;
NET LED<1>    LOC=P23;
NET LED<2>    LOC=P19;
NET LED<3>    LOC=P17;
NET LED<4>    LOC=P18;
NET LED<5>    LOC=P14;
NET LED<6>    LOC=P15;
NET LED<7>    LOC=P24;
#
# DATA BITS FROM THE PC PARALLEL PORT
NET PC_D<0>   LOC=P46;
NET PC_D<1>   LOC=P47;
NET PC_D<2>   LOC=P48;
NET PC_D<3>   LOC=P50;
NET PC_D<4>   LOC=P51;
NET PC_D<5>   LOC=P52;
NET PC_D<6>   LOC=P81;
NET PC_D<7>   LOC=P80;
#
# MICROCONTROLLER PINS
NET XTAL1     LOC=P10; # INPUT CLOCK
NET RST       LOC=P45; # ACTIVE-HIGH RESET
NET ALE_      LOC=P20; # ACTIVE-LOW ADDRESS LATCH ENABLE
NET PSEN_     LOC=P13; # ACTIVE-LOW PROGRAM-STORE ENABLE
NET RD_       LOC=P32; # ACTIVE-LOW READ
NET WR_       LOC=P63; # ACTIVE-LOW WRITE (ALSO CONTROLS RAM)
NET AD<0>     LOC=P44; # MULTIPLEXED ADDRESS/DATA BUS
NET AD<1>     LOC=P43;
NET AD<2>     LOC=P41;
NET AD<3>     LOC=P40;
NET AD<4>     LOC=P39;
NET AD<5>     LOC=P37;
NET AD<6>     LOC=P36;
NET AD<7>     LOC=P35;
NET A<0>      LOC=P75; # DEMUXED LOWER BYTE OF ADDRESS
NET A<1>      LOC=P79;
NET A<2>      LOC=P82;
NET A<3>      LOC=P84;
NET A<4>      LOC=P1;
NET A<5>      LOC=P3;
NET A<6>      LOC=P83;
NET A<7>      LOC=P2;
NET A<8>      LOC=P58; # UPPER BYTE OF ADDRESS
NET A<9>      LOC=P56;
NET A<10>     LOC=P54;
NET A<11>     LOC=P55;
NET A<12>     LOC=P53;
NET A<13>     LOC=P57;
NET A<14>     LOC=P61;
NET A<15>     LOC=P34;
NET A16       LOC=P74; # MS address bit of 128K RAM
#
# RAM CONTROL PINS
NET OE_       LOC=P62; # ACTIVE-LOW OUTPUT ENABLE
NET CE_       LOC=P65; # ACTIVE-LOW CHIP ENABLE
#
# THESE ARE FREE PINS THAT DON'T CONNECT TO ANYTHING ELSE
//NET FREE<0> LOC=P4;
//NET FREE<1> LOC=P12;
//NET FREE<2> LOC=P25;
```

```
//NET FREE<3>LOC=P74; # also used as MS address bit of 128K RAM
//NET FREE<4>LOC=P76;
//NET FREE<5>LOC=P77;
```

General-Purpose FPLD+Microcontroller Interface Circuitry

Before designing a circuit to perform a specific function, it makes sense to develop a generic framework which contains the circuitry that is needed in every design. This circuitry is shown in Figure 4.

At the top of Figure 4 is the circuitry for bringing the eight bits data from the PC parallel port into the FPLD. Data on the **PC_D[7:0]** bus is passed through a standard byte-wide input buffer (**IBUF8**) and appears on the internal **PC_D_IN[7:0]** bus. The same type of circuit is used to pass the upper address byte from the 8031 (**A[15:8]**) onto the internal address bus (**A_IN[15:8]**). Single-bit input buffers are used to transfer the **ALE_**, **PSEN_**, **RD_**, and **WR_** signals onto the internal **ALE_IN_**, **PSEN_IN_**, **RD_IN_**, and **WR_IN_** signals, respectively.

The multiplexed address/data bus (**AD[7:0]**) is bidirectional because the 8031 uses it to both write and read bytes of data to and from the RAM and FPLD. An **IBUF8** buffer is used to read the values on this bus onto the internal address/data bus (**AD_IN[7:0]**) of the FPLD. The FPLD also has to be able to write to the **AD[7:0]** bus in order to send data back to the 8031 or the external memory. For this reason, a tristate buffer (**OBUFE8**) is attached to the **AD[7:0]** pins. The value on the internal data bus (**D_OUT[7:0]**) will be forced onto **AD[7:0]** when the buffer driver-control signal (**D_DRV**) is high.

The FPLD is responsible for de-multiplexing the 8031's address/data bus so that a stable address can be sent to standard addressable devices like the external RAM. A byte-wide output register (**OFD8**) is used to latch the lower byte of the address from the **AD_IN[7:0]** bus on the falling edge of **ALE_IN_**. The latched address is output on the **A[7:0]** pins.

The 8031's reset (**RST**) is driven by the value output on the least-significant bit of the PC parallel port (**PC_D_IN0**). This lets us reset the 8031 using the XSPORT utility.

The active-low chip-select of the external RAM (**CE_OUT_**) is driven by the most-significant address bit (**A15_IN**). When **A15**=0, the RAM is selected which effectively maps it into the address range `0000H-7FFFH`. (This simple chip-select scheme wastes much of the 128 KByte RAM on an XS+ Board.) The outputs of the RAM are enabled by a low level on **OE_** whenever the 8031 fetches an instruction (**PSEN_IN_**=0) or performs a data-read operation (**RD_IN_**=0).

The most-significant address bit of the 128 KByte RAM on an XS+ Board is driven by the 8031 program store enable (**PSEN**). This partitions the RAM into a lower 64 KByte code segment and an upper 64 KByte data segment. (Only the lower 32 KBytes is available in each segment because of the simple RAM chip-select scheme we are using.)

The programmable oscillator output enters the FPLD through the **CLK** input. The oscillator has a factory default frequency of 50 MHz so a divide-by-three circuit generates a 16.7 MHz clock that is passed to the 8031 through the **XTAL1** pin.

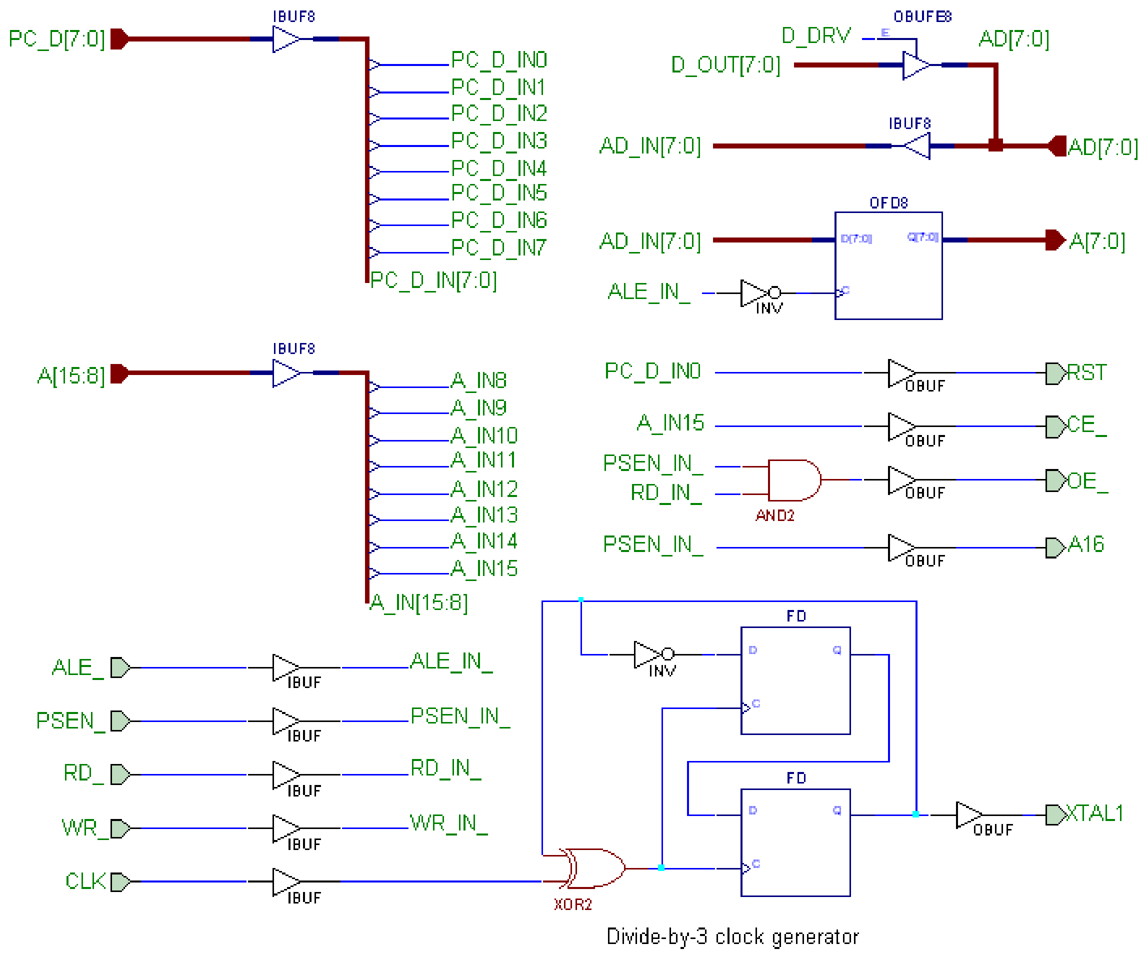

Figure 4: General-purpose interface circuitry.

LED Display Design

Displaying patterns on the LED digit is a simple design that will show you how to work with the XS Board. The specifications for the design are that it should sequentially light each LED segment for about a second, extinguish it, and proceed with the next segment. After the last segment is tested, the entire procedure is repeated in an endless loop.

The system needs a clock input to time the interval each segment is active. Eight outputs are needed to drive the eight LED segments.

Now we have to partition the functions for our test system between the FPLD and the microcontroller. Obviously, we can build the entire test system using only the FPLD. A 25-bit counter in the FPLD can be incremented by the 50 MHz oscillator. The counter will roll-over every 0.67s at which time a rotating eight-bit shift register (also in the FPLD) will shift its contents. If this shift register is initially loaded with `00000001` and the output of each register bit drives one of the LED digital segments, then all eight segments will be individually tested after eight shifts. Looping the upper bit of the shift register back into the least-significant bit of the shift register insures the testing will repeat as long as the clock is active.

Since we wanted to show how to design a system using both the FPLD and the microcontroller, let's partition it a bit differently. The FPLD will still be used to drive the LED digit segments, but the sequencing of the test will be moved to the microcontroller. The microcontroller will write a rotating bit pattern to the LED and time each step in the test sequence using a delay loop. The FPLD will be used to implement a writable register within the address space of the microcontroller.

The schematic for the writable LED register is shown in Figure 5. The input data bus (**AD_IN**) bus carries the data value from the 8031 into the LED register (**FD8CE**). The register is enabled for writing if the upper four address bits are all high. So any access by the 8031 to the address range `F000H-FFFFH` will enable the register. The data is loaded into the LED register at the end of a write operation by the rising edge on the **WR_IN_** signal. The output of the register goes through a byte-wide output buffer (**OBUF8**) that drives the pins attached to the LED display. Because the register is writable but not readable, the **D_DRV** signal is tied to ground to disable the output buffers that would send data from the FPLD to the 8031.

The LED register schematics are exported as .EDN files (EDIF format) that are mapped, placed, and routed by the XILINX Foundation software to create an LEDREG.BIT bitstream file (for XS40 Boards) or an LEDREG.SVF file (for XS95 Boards).

Once the LED register hardware is designed and implemented, we can write the assembly code for the 8031 (Listing 3). The program works as follows:

Line 4: An address for the LED register implemented in the FPLD is defined for the 8031.

Line 8: A byte of the internal RAM in the 8031 is reserved to serve as a counter in the `WAIT` subroutine.

Line 11: The microcontroller program is set to start at address `0000H` which is the address the 8031 jumps to immediately after a reset. All 8031 programs for the XS Boards must be stored in the RAM. The FPLD was configured to map the RAM into the range `0000H—7FFFH` so there would be some RAM to hold the instructions where the 8031 expects to find them.

Line 16: The accumulator is initialized with the bit pattern `00000001`. The '1' bit in the pattern will be rotated through all eight positions and loaded into the LED register each time to test all the segments of the LED digit.

Lines 19–20: The address of the LED register is written into the pointer used for accessing external memory, and then the value in the accumulator is indirectly written to the register.

Line 22: A call is made to the `WAIT` subroutine which inserts a delay of approximately one second. This gives us time to observe the segments of the LED digit before they get loaded with the next pattern.

Line 24: The contents of the accumulator are rotated so that the next LED segment will be activated on the next loop through the program.

Line 26: This transfers the program back to the start of the loop so the next LED segment can be checked.

Lines 30–43: The `WAIT` subroutine counts through $10 \times 256 \times 256 = 655{,}360$ loop iterations at approximately 1.5 µs per iteration. This inserts a delay of about a second. The subroutine pushes and pops the `A` and `B` registers so they retain the same values upon exit from the subroutine.

The assembly code in the LEDDISP.ASM file is assembled using the command:

```
C:> ASM51 LEDDISP.ASM
```

This command will create the file LEDDISP.HEX. (Make sure you have a copy of the mod52 file in the directory where you are performing the assembly. mod52 contains definitions for many of the internal register addresses of the 8052/8031. You can find mod52 in the directory created when you installed the XSTOOLs software utilities.)

Now you can download the LEDDISP.HEX and LEDREG.BIT or LEDREG.SVF files to the XS Board. The downloading cable must be attached between the PC printer port and the J1 header of the XS Board. The actual downloading process for the XS95 Board is accomplished with the command:

```
C:> XSLOAD -P 1 LEDDISP.HEX LEDREG.SVF
```

And the downloading process for the XS40 Board is done with this command:

```
C:> XSLOAD -P 1 LEDDISP.HEX LEDREG.BIT
```

XSLOAD loads the RAM with the program found in the LEDDISP.HEX file. Then it configures the FPLD with the circuit description stored in the .SVF or .BIT file. The command given above assumes the XS Board is hooked to LPT1 so it uses the option `-P 1`. If the XS Board is hooked to LPT2 or LPT3, then use `-P 2` or `-P 3`, respectively. (If you don't use the `-P` option at all, XSLOAD will use LPT1 as the default printer port.)

Once you have the XS Board completely loaded, you would naturally ask: "Now what?" For this simple example all you have to ensure is that the microcontroller is reset correctly. Recall that parallel port pin **D0** is connected to the **RST** line of the 8031 through the FPLD. The 8031 can be reset and released to start executing its program with the following command sequence:

```
C:> XSPORT 1
C:> XSPORT 0
```

After you execute these two commands, you should see the LED segments being sequentially activated at one second intervals.

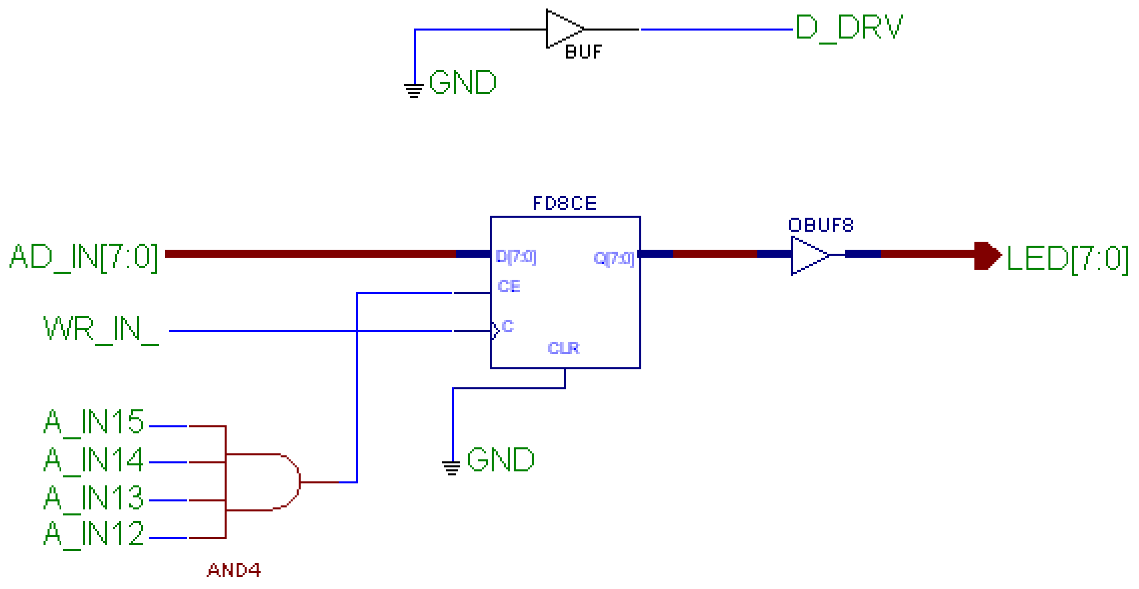

Figure 5: Circuitry for the LED register.

Listing 3: LED tester assembly code for the 8031 microcontroller (LEDDISP.ASM).

```
        $MOD51

STACKTOP EQU    70H    ; start of stack (grows up)
LEDREG   EQU    0F000H ; LED register address

        DSEG
        ORG     30H
CNTR:   DS      1      ; counter for wait routine

        CSEG
        ORG     0000H  ; program starts at 0 after reset
START:
        ; initialize stack pointer ...
        MOV     SP,#STACKTOP
        ; and the initial bit pattern to display
        MOV     A,#1
LOOP:
        ; show the bits on the LED digit
        MOV     DPTR,#LEDREG
        MOVX    @DPTR,A
        ; wait long enough so we can see the bits ...
        CALL    WAIT
        ; then rotate the bit pattern ...
        RL      A
        ; and then do it all again
        JMP     LOOP


; this subroutine waits about 1 second
WAIT:
        PUSH    ACC
        PUSH    B
        MOV     cntr,#10
WAIT1:
        MOV     B,#0
WAIT2:
        MOV     A,#0
        DJNZ    ACC,$
        DJNZ    B,WAIT2
        DJNZ    CNTR,WAIT1
        POP     B
        POP     ACC
        RET

        END
```

Random Number Generator Design

The LED display example in the previous section is very simple. All that was needed was a memory-mapped, writable register in the FPLD. What if we needed to perform the opposite operation and pass data from the FPLD back into the microcontroller?

The example in this section will show the design of a simple, eight-bit random number generator (RNG). The 8031 will write a seed value to the RNG. Then it will read a sequence of random numbers from the RNG. The bit pattern output from the RNG will be displayed on the LED digit.

The inputs and outputs for this design are the same as for the LED display: a clock input and eight LED driver outputs.

The design will be partitioned as follows:

- The FPLD will hold the LED register (as before) and will also be used to implement a RNG as a linear-feedback shift-register (LFSR). The random number seed is written into the LFSR, and every subsequent read of the LFSR will return a new random number.

- The microcontroller will be responsible for initializing the random number generator and then reading random numbers from the LFSR and writing the eight-bit patterns to the LED register in an endless loop. The microcontroller will insert a one-second delay between each write to the LED register so we have time to observe the random numbers.

The FPLD circuitry is shown in Figure 6.

The random number generation circuitry is included below the LED register from the previous section. The main component of the random number generator is an eight-bit loadable shift-register (**SR8CLE**). Because it will change its value whenever it is written or read, the shift-register is clocked with the logical AND of the **RD_IN_** or **WR_IN_** strobes so that a clock pulse is generated when either strobe goes low. In order to clock all the flip-flops in the shift-register at the same time, a clock buffer (**BUFG**) is inserted between the **AND** gate and the clock input. This reduces clock skew between the flip-flops of the shift-register and allows reliable shifting of the bits. (The **BUFG** is not needed for CPLD-based designs and should be removed if you are targeting the XS95 Board.)

The shift-register can be loaded with a seed by having the 8031 write to an address in the range `C000H-CFFFH`. A write to this address range places a logic 1 on the load control input (**L**) of the shift-register, sends the seed value over the **AD_IN** bus, and generates a rising edge on the register's clock input (**C**) at the end of the write operation.

The random number in the shift-register is fetched by having the 8031 read an address in the range `E000H-EFFFH`. A read to this address range forces the **D_DRV** signal high, thus turning on the buffer drivers in the FPLD (see Figure 4) that will send the value on **D_OUT** to the 8031.

Reading the random number generator also causes it to generate a new random number. If the address that is read is in the range `E000H-EFFFH`, then the shift-enable input (**CE**) will be raised to a logic 1. The low pulse on **RD_IN_** will clock the shift-register. This causes the contents of the shift register to be shifted left and the value on the serial-input (**SLI**) is placed in the least-significant bit of the register. The new LSB is formed from the exclusive-OR of bits 3, 4, 5, and 7 of the old value in the shift-register. This combination of bits insures that the shift-register will pass through 255 distinct states before repeating.

The RNG schematics are exported as .EDN files (EDIF format) that are mapped, placed, and routed by the XILINX Foundation software to create a RANDGEN.BIT bitstream file (for XS40 Boards) or a RANDGEN.SVF file (for XS95 Boards).

Once the RNG hardware is designed and implemented, we can write the assembly code for the 8031 as shown in Listing 4. It is very similar to the one in the previous section so we will only discuss the differences:

Line 5: The memory address from which new random numbers are read is defined for the rest of the program.

Line 6: The memory address for writing the seed value to the random number generator is defined here.

Lines 18–20: The accumulator is loaded with a seed value that is then loaded into the random number generator.

Lines 23–24: A random number from the LFSR is indirectly read into the accumulator.

Lines 26–27: The bit pattern read from the random number generator is written to the LED register so we can see it.

As was shown before, you can download the RNDDISP.HEX and the RANDGEN.SVF files to the XS95 Board with the command:

```
C:> XSLOAD -P 1 RNDDISP.HEX RANDGEN.SVF
```

or to the XS40 Board with this command:

```
C:> XSLOAD -P 1 RNDDISP.HEX RANDGEN.BIT
```

The 8031 is reset and released to start executing its program with these commands:

```
C:> XSPORT 1
C:> XSPORT 0
```

After this, you should see the LED segments being activated in random patterns. You can record several patterns and check that they match what would be generated by the LFSR of Figure 6.

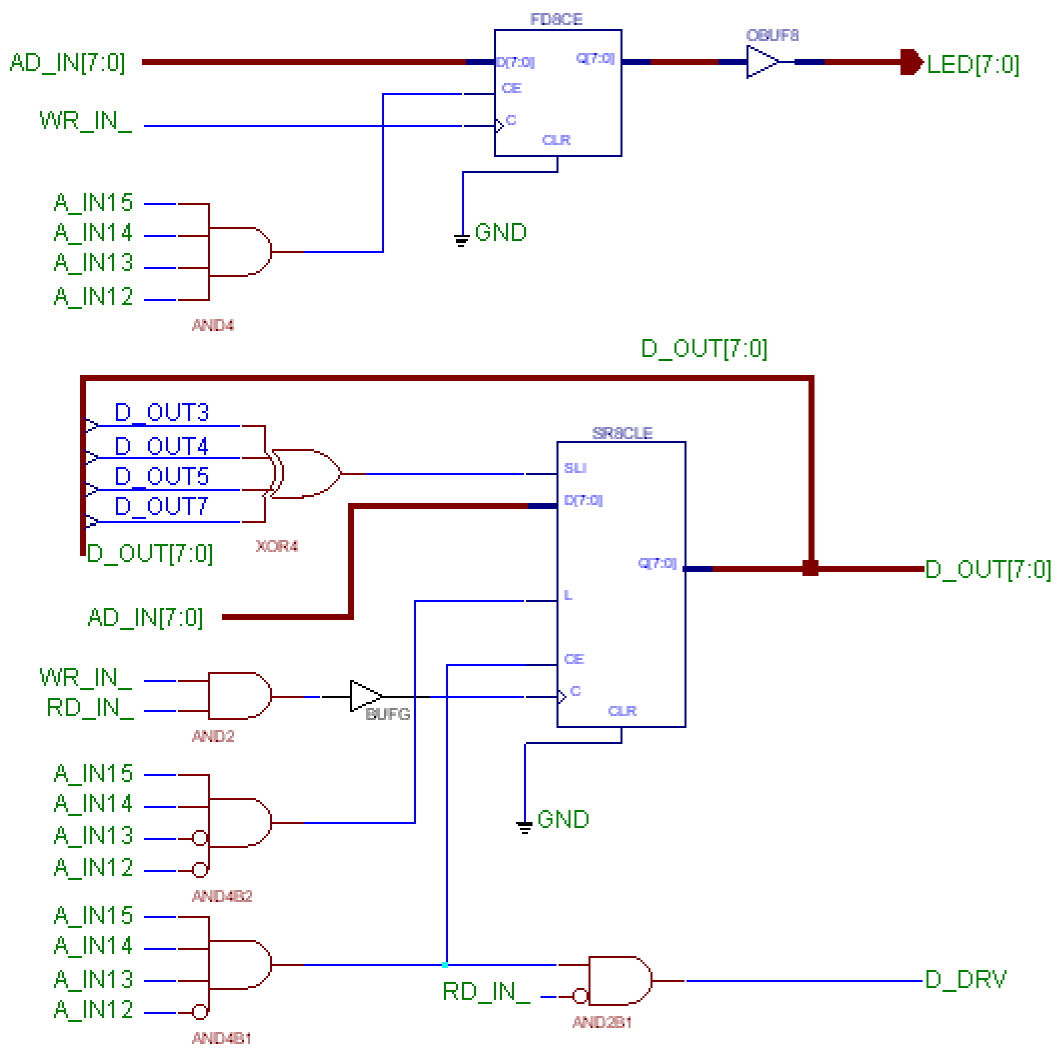

Figure 6: Random number generator circuitry.

Listing 4: Random number generator test assembly code for the 8031 microcontroller (RNDDISP.ASM).

```
        $MOD51

STACKTOP EQU    70H    ; start of stack (grows up)
LEDREG   EQU    0F000H ; LED register address
RNDGEN   EQU    0E000H ; address for reading random num. gen.
SEED     EQU    0C000H ; address for writing random num. gen. seed

        DSEG
        ORG     30H
CNTR:   DS      1      ; counter for wait routine

        CSEG
        ORG     0000H  ; program starts at 0 after reset
START:
        ; initialize stack pointer ...
        MOV     SP,#STACKTOP
        ; and the random number seed
        MOV     A,#1
        MOV     DPTR,#SEED
        MOVX    @DPTR,A
LOOP:
        ; read a new random number from the FPGA ...
        MOV     DPTR,#RNDGEN
        MOVX    A,@DPTR
        ; and show the random bits on the LED digit
        MOV     DPTR,#LEDREG
        MOVX    @DPTR,A
        ; wait long enough so we can see the bits ...
        CALL    WAIT
        ; and then do it all again
        JMP     LOOP


; this subroutine waits about 1 second
WAIT:
        PUSH    ACC
        PUSH    B
        MOV     cntr,#10
WAIT1:
        MOV     B,#0
WAIT2:
        MOV     A,#0
        DJNZ    ACC,$
        DJNZ    B,WAIT2
        DJNZ    CNTR,WAIT1
        POP     B
        POP     ACC
        RET

        END
```