TH Logger

List of Authors: Robert Grooms, Matthew Harkness, Yunhao Chen, Keith Inch
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: rcgrooms@oakland.edu , mdharkne@oakland.edu , ychen2@oakland.edu, kjinch@oakland.edu

***Abstract*— The project summarized here is a temperature and humidity reader. The sensor we used was a humidity and temperature Sensor - RHT03 that was purchased through sparkfun.com. The output of the sensor is a single wire bidirectional digital interface that outputs a forty bit stream after is receives a start bit from the control board. The forty bit stream is then decoded through various modules we built into the top module to produce a display of the temperature or humidity that is selected by the switches on the Digilent FPGANexys 4 DDR Artix-7 FPGA: Trainer Board.**

I. INTRODUCTION

The scope of this project was to make a field-programmable gate array, FPGA, board to read and receive data from an external temperature and humidity sensor. The sensor used was the RHT03 sensor, built by Maxdetect. The motivation was from the idea that the weather channel never has the exact temperature or humidity for where you are at that exact moment. The people who want to know exactly what to wear for that day, without actually going outside, would use this. All the specifications of this RHT03 sensor are shown in Figure 1.

Model	RHT03	
Power supply	3.3-6V DC	
Output signal	digital signal via MaxDetect 1-wire bus	
Sensing element	Polymer humidity capacitor	
Operating range	humidity 0-100%RH;	temperature -40~80Celsius
Accuracy	**humidity +-2%RH**(Max +-5%RH);	temperature +-0.5Celsius
Resolution or sensitivity	humidity 0.1%RH;	temperature 0.1Celsius
Repeatability	humidity +-1%RH;	temperature +-0.2Celsius

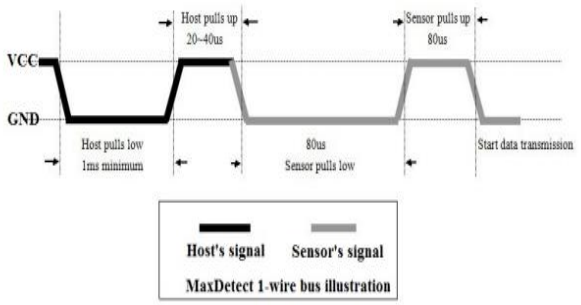
[2]. Figure 1: Specifications of RHT03 Sensor

This project directly utilizes the premise of several labs that were conducted in class. This project covers many topics from the course material, such as, finite state machines, a bidirectional bus, counters, registers, and a serializer. It also covers a topic that was not covered, which is a direct binary to seven-segment display convertor. This project can be applied to the everyday life of people who want to start there day with a current weather update. Although it cannot predict weather changes, it is useful enough to start your day.

II. METHODOLOGY

The RHT03 sensor was designed for a different language of coding, C, and a different board. This created issues with the timing of reading the data and many different techniques in VHDL had to be used to implement this sensor. When outlining the premise of this circuit design, we originally began with a different style of a design using a random access memory emulator. This was scraped do to the way this sensor needed to be initiated and read from. This changed into using many different counters and a main finite state machine to run this process. RHT03 works on a timing basis and outputs 40 bits of data, which takes a long time, compared to the speed of modern computers. To get this sensor to begin its process of reading the temperature and humidity, the FPGA board must send a start signal. Since this sensor sends and receives data through just one wire, a bidirectional bus was necessary.

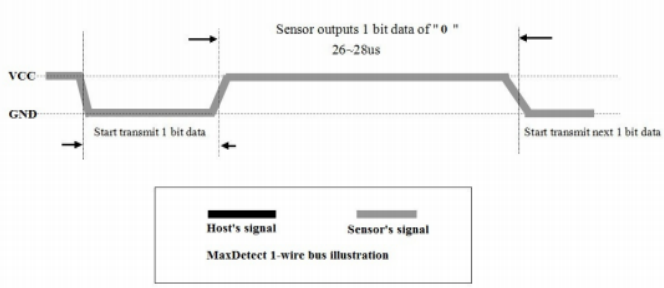
This bidirectional bus sends and receives its data directly into our finite state machine. This main finite state machine is essentially the control unit. This finite state machine outputs a one into the bus while the tri-state buffer going out is enabled and the input buffer is disabled, which is also a signal sent from this main FSM. Once this initiation signal is sent, the board pulls low, by sending a ‘0’, for ten milli-seconds, and then the board pulls high, by sending a ‘1’, for twenty microseconds. The finite state machine keeps track of this time by sending out a count signal, ‘1’, to the first and second counter. These are integer counters based on the period, 10 nanoseconds, of the 100 MHz input clock. After this happens, the sensor takes over and the input buffer is then enabled for the rest of the process, while disabling the output buffer. This is beginning process is shown in Figure 2.



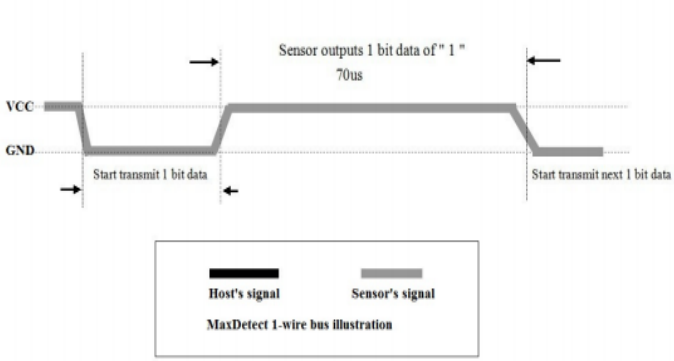
[2]. Figure 2: Initialization of RHT03

Once the sensor takes over, the finite state machine must interpret what the sensor is sending into the board. After the second counter finished, the finite state machine must wait for the data coming in to output a '0'. The state machine must keep count for 80 microseconds and then look for when the data coming is a '1'. The FSM then clears and utilizes the same 80 microsecond counter then looks to see if the data coming into the state machine is a '0'. Once the finite state machine recognizes its low state, it waits until the data goes high again. After all this, the actual data, to be collected, begins to be documented.

For this sensor, it is all based on time of how long the data coming in stays high. If the data coming in stays high for only 26 microseconds, then the output is a '0', shown in Figure 3. If it stays high for 70 microseconds, the data being read is a '1', shown in Figure 4. To keep track of this count, an integer counter is used based upon the frequency of the input clock and how long each clock tick period is. The initial counter is counted up to 26 microseconds and then the finite state machine checks to see if the data has gone low or is still high. When it is low at this point, the finite state machine shifts in a '0' into the 40-bit register, which is also enabled at the same time. This then initializes our 40-bit counter to keep track of all the bits coming in. If instead the data coming in was a '1' after the 26 microsecond counter, it enters another counter which counts up to 44 microseconds, to meet the 70 microseconds need to be a '1'. There is a fail safe after this point to check if the data is still high or has gone low. If the data has gone low, then the finite state machine enables and shifts in a '0'. If it is still high, then the data being shifted in is a '1'. The main finite state machine explained can be seen in the ASM form in Figure 6. This then goes to the 40-bit counter to keep track of all the bits coming in to tell when the process has stopped. Once all the bits are accounted for, there is a "swait" state that continuously loops around until the start button is pressed again to start the process all over again.



[2]. Figure 3: Detecting a '0'



[2]. Figure 4: Detecting a '1'

Once all this data is collected and output from the register, a signal is used to separate the 40-bits into two 16-bit sections and an 8-bit section. The 40-bit output consists of bits 39 down to 24 being the reading for the relative humidity. The next 16-bits, 23 down to 8, is the temperature reading. The last 8-bits is the check sum, which is the first 32-bits added together to make sure that the output is correct. The 8-bit output for the check sum is sent and displayed on the first 8 LED's on the board. Using the branches of the signal "dataout," the relative humidity and temperature bits are sent into their respective 16-bit register. The output of the two registers are then sent through a mux which allows us to toggle between reading the temperature and humidity through "sw<0>" on the board. Since this output is in binary, there had to be a way to convert this binary output into their respective seven-segment codes.

The decoder, while simple in theory, was in fact quite substantial, as instead of creating a binary-to-bcd converter, we found it simpler to write select statements for each possible input, and for each of the digits. First off, the sensors data sheet informed us that if the reading was negative, then the 16th bit would be a one, and the other 15 bits would be displayed the same. This made our job a bit simpler as we would not need to correct for the 1s or 2s compliment. So a simple statement was written to check for a value of the 16th bit and set the value of that display to be a dash if so. For the remaining digits we only looked at the remaining 15bits, so as not to have to double our workload.

Being that the sensor could only read up to 80 C or 100% Humidity, we saw no reason to exceed a reading of 99.9, which would be output by the sensor as 999, and as a result only wrote statements up to 1024, or the 11th bit. It seemed like an inefficient sensor design to send 16 bits when only 12 would be relevant (11 plus the "negative") but regardless. So select statements were written for each of the 3 digits given every possible input up to 999 (or "000001111100111"), and all others were set to display a dash. Then another bit was added to the 7bit outputs for each so that a decimal point could be added onto the second digit to handle our division by 10.

Once the converter finds every possibility and converts this 16-bits, it outputs four 7-bit signals. Each output has its purpose in depicting what and there to display each number. One signal is for the ones place, one for the tens place, one for the hundreds place, and one just in case the temperature reads a negative number. Since this input to the converter is a large decimal number, the sensor must divide by ten to calculate the actual reading. We did this by always setting the tens place seven-segment display to having the decimal point on. This made the coding easier for not having to actually compute this in code. The four 7-bit signals are sent into a multiplexer, which is controlled by the serializer finite state machine, shown in Figure 5. This code we obtained from Dr. Llamocca [1]. This state machine initializes a counter, a decoder, and the multiplexer. The idea of this is to quickly enable each seven-segment display, while displaying each output of the multiplexer. The finite state machine uses the counter to cycle through the mux and the decoder so quickly that the human eye cannot tell that they are not actually all on at once. This serializer had to be used because the FPGA board does not allow the user to display each output separately on each display and turn them on at the same time [1]. To show how all these components interact with each other, the top-level design is shown in Figure 7.

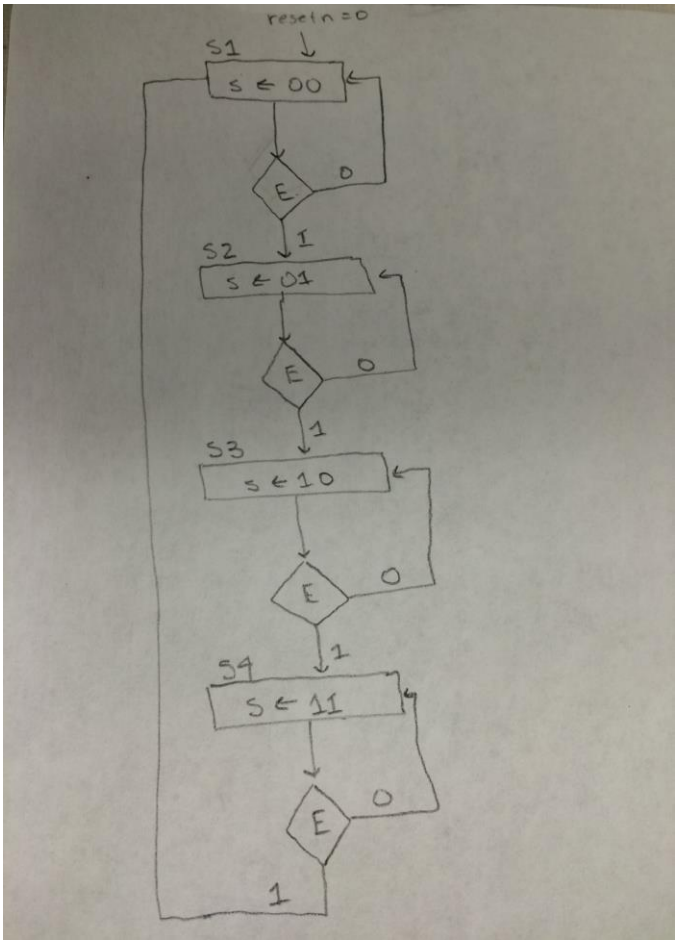


Figure 5: FSM of the Serializer in ASM form

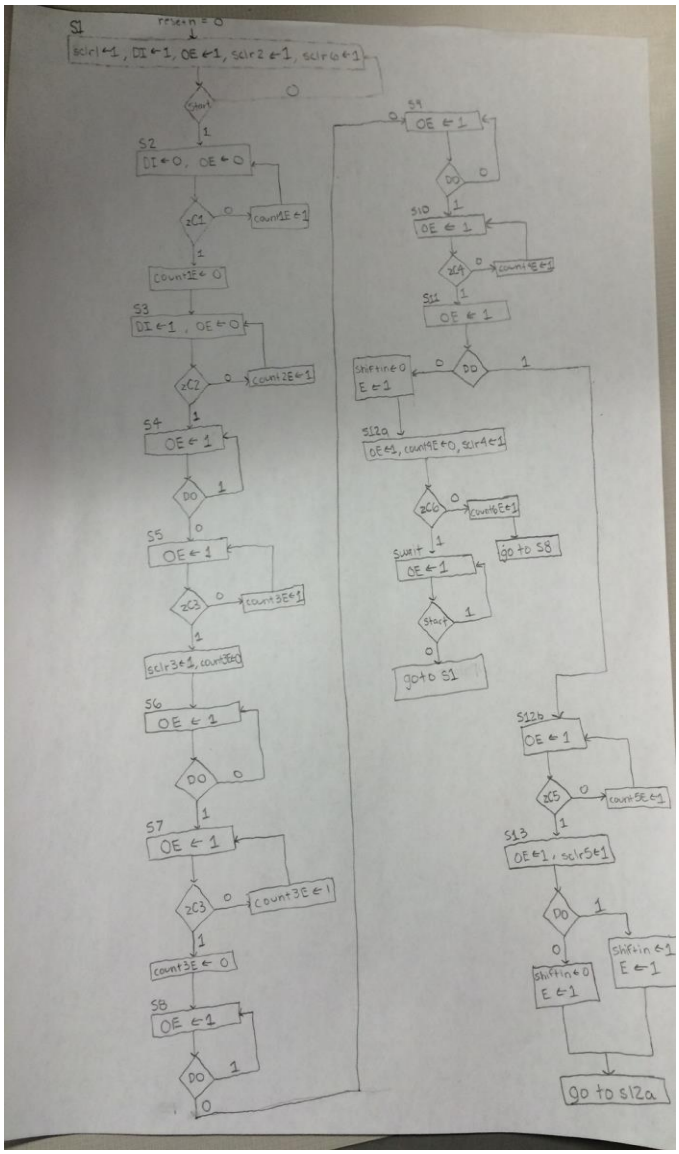


Figure 6: Main FSM in ASM form

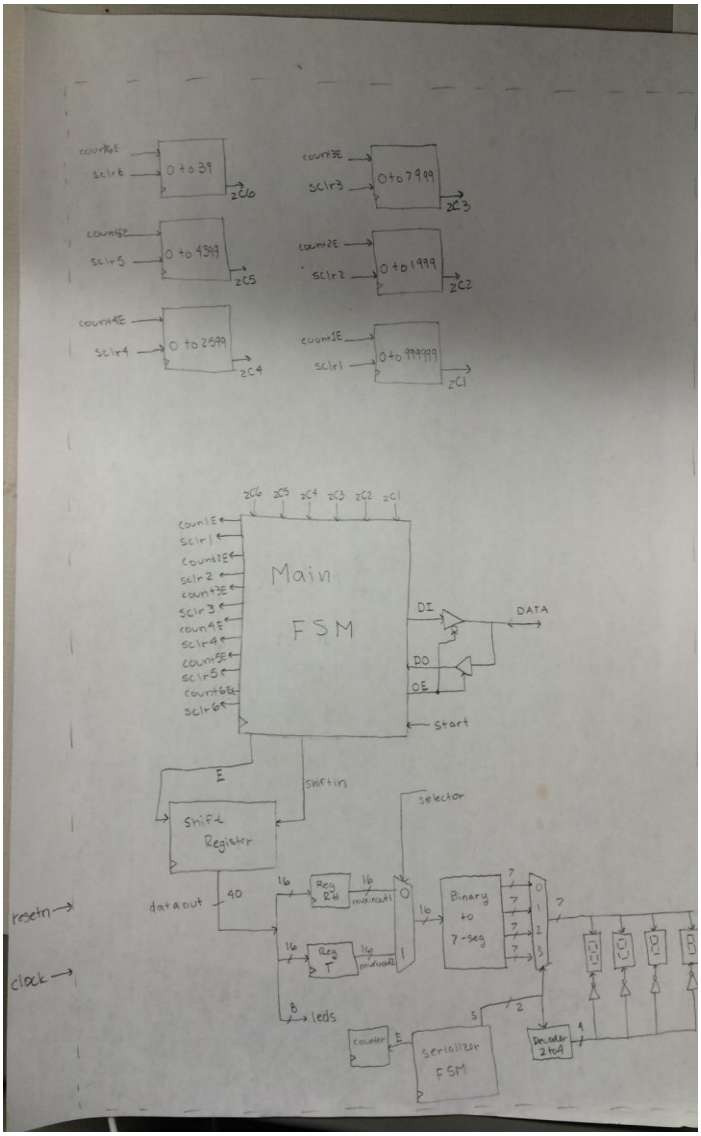


Figure 7: Top-Level Circuit Design

III. EXPERIMENTAL SETUP

To make sure that this was functioning before we hooked up the sensor, we made a testbenh using ISE. We ran this testbench through the simulation tool in ISE to make sure that the control finite state machine was switching states correctly and the timing was on. Since the data was going to be different for each input of a '0' and a '1', we continuously switched between a '1' and a '0' for each of the 40-bits. To tell if the timing was correct, we set the top level design side by side with the simulation to make sure that it was sending the signals and switching states correctly. Once this simulation was verified, we could plug the sensor in and test it. Had we plugged the sensor in prior to doing this, we could have blown the sensor if the timing was off. We expected there to be issues with the timing and this is where debugging was a must.

IV. RESULTS

For the results, we obtained the simulation of the whole design. These results were exactly what we had expected. It was difficult to figure out how to obtain these results, but it was done through understanding the testbenches for an ’inout’ signal data input. Learning more about bidirectional buses and how to implement them in a testbench helped this part. Also, understanding how a finite state machine operates and what to implement this in a testbench helped. Once you make sure you input the correct data to begin the process of the state machine it was easy to see. Only one thing was going to possibly unexplainable. If the sensor was not sending out the correct data, then we would not be able to tell what the issue was. Luckily the sensor we had was perfectly functioning and the answers that displayed on the seven-segments correlated well with the actual temperature and humidity. The good part was that everything was visually seen in the simulation. This allowed us to know if it was going to work well before we programmed the board and tested it. The only possibility of an issue when programming the board was to make sure that the constraints file was correctly written. Once the constraints file was done, the only thing left was to test it. The simulation played perfectly on the board, just as expected. The results of this data can be seen in figures 8 and 9. This shows the states changing at the correct times as the data coming in oscillates from being a ‘1’ and a ‘0’.

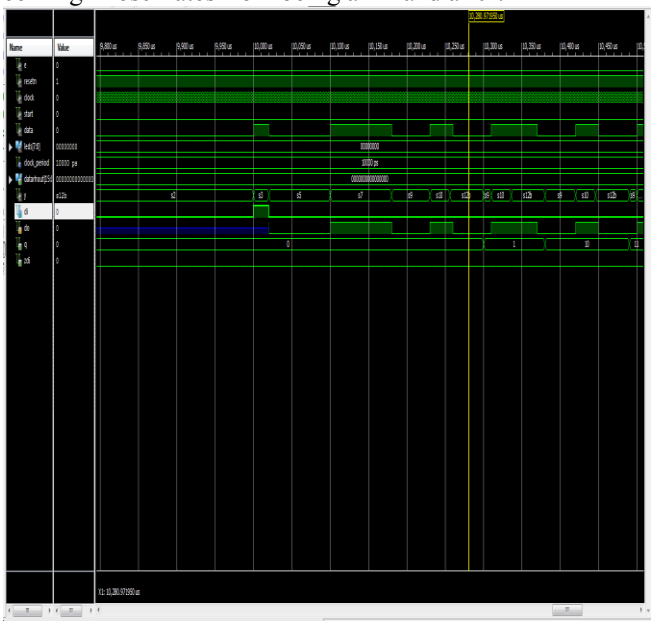


Figure 8: First Simulation Piece

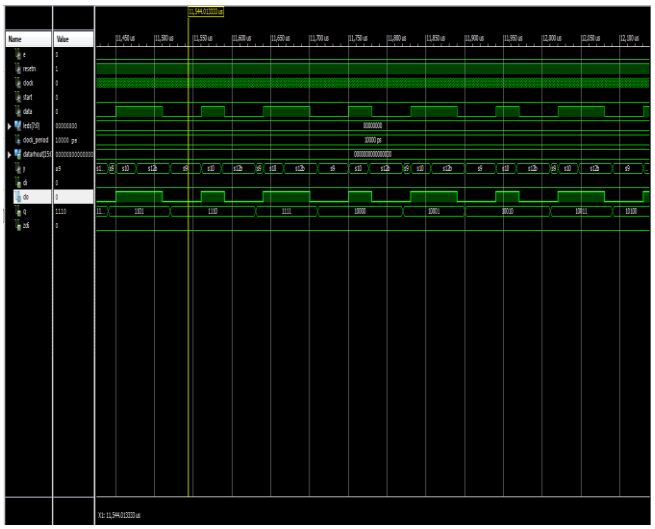


Figure 9: Second Simulation Piece

V. CONCLUSIONS

This project really made it clear on how to implement VHDL coding for an external sensor. This gave a clear understanding on how to bring a sensor, meant for a different language and board, to VHDL and an FPGA board. It really helped give a better understanding of debugging and what to look for when it comes to debugging via simulation model. This project could have been improved had the timing been right on down to the nanosecond, but it was close enough to be within the range of uncertainty allotted for such sensor. Could have also been improved if the sensor was a higher quality sensor. Every issue that we ran into was solved through help from the teacher or self-taught.

REFERENCES

[1]. VHDL Coding Tutorial- Daniel Llamocca

http://www.secs.oakland.edu/~llamocca/VH

DLforFPGAs.html

[2]. Liu, Thomas. "Digital Relative Humidity & Temperature Sensor RHT03." Humiditycn. MaxDetect Technology Co., Ltd. Web.14,Apr.2016.<https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Weather/RHT03.pdf>.