

1-Wire Controller

Xavier Miró Bruix

xavier@gwu.edu

CS 190

4/23/2006

Project Abstract

In this project I have implemented a 1-Wire Controller out of a Zilog Z8 microcontroller. The purpose of the project was implementing the functionality on the Z8 so that it could work with the sensors DS1820.

These sensors are high precision 1-Wire digital thermometers, and are produced by Dallas Semiconductor. They are little and easy to use because of their more important characteristic: 1-Wire technology only uses 1 wire (plus additional ground and optional additional power supply) and because of that the sensor only has 3 pins. One of the pin is for Ground, the other for Data and the last one for additional power.

In this project I also tried to implement the Search algorithm for 1-Wire devices. Even though there is a very complete description of the algorithm in Dallas Semiconductors website, I wasn't able to implement it.

The board that I was given also had some DS2401. These devices don't have any functionality. The only purpose of them is to give a unique 64-bit registration number in case that I implemented any of the search algorithms.

Status

Very few things worked as I hope they would work. It has been pretty discouraging working on it and seeing no progress. Once I started reading all the documentation I found out that many of the code lines were already implemented for me. In the Dallas' "1_Wire Communication Through Software" we have all the functions for sending and receiving data. The only thing we have to do is to program the ports so that they work as we want them to work. Then, once that was implemented, I had to worry about making the 1microsecond delay for the timings. Finally I found out that 6 "nop" operations in assembly as `asm("nop");` could produce a delay of 1 microsecond. Then, once I knew how many microseconds need each delay, it was just a matter of iteration in a loop of 6 "nop" instructions.

As I said, in the beginning I thought about using the Search Algorithm, something that at the end I didn't use. I worked with the Skip ROM command (doesn't deal with the ROM registers and supposes that there is only 1 device in the network) and the Read ROM command which gets the ROM register of 1 device in the network (only works when there is only 1 device in the network).

1-Wire Technology

This project is based on 1-Wire Technology, so I think it is appropriate to talk a little bit about this protocol. 1-Wire technology gets its name from one of its more important characteristics: a master device can control a 1-Wire device with just one wire. Well, let's say 2 wires, since we also need a ground wire.

1-Wire networks are made out of 3 different elements: a bus master controller, the wires and connectors, and the 1-Wire devices. The network is an open-drain master/slave architecture that uses a resistor pull-up to 5V supply at the master. It is a rigid protocol since no node can talk without the authorization of the master and no communication is allowed between slaves except through the master. Inside each component there is a ROM section with a 64-bit register that stores a unique serial number that will act as its address.

In order to accomplish the 1-wire idea, we need to drive data and power through the same wire. Data on the 1-Wire net is transferred by time slots.

Power

Power can be provided in two ways. The first possibility, and the most '1-Wire' way would be the called "parasite power". In this mode the power gets to the devices through the data wire. Whenever the data line is pulled high, the diode inside the device turns on and charges a capacitor. When the voltage drops at the pulled down line, the capacitor can start supplying power to the device until the line goes again high.

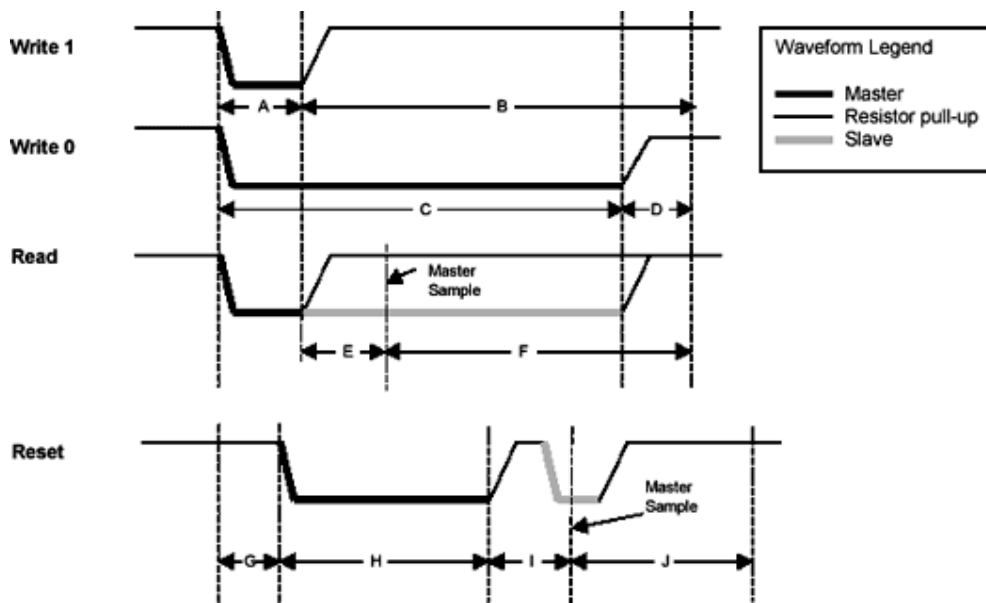
Sometimes "parasite power" is not possible. The power needed by the device when the line is low may be bigger than the power that can be provided by the capacitor. In this case we should provide an external power to the system, not through the data line. There are other scenarios when we don't have the need of just having 1 wire, and we can run safer by having an extra power supply and not having to rely on the capacitor.

Timing and Signals

As I said before, data is transferred by time slots. A time slot is the time it takes to perform one bit of communication. In the figure below, we can see each of the basic operations through the 1-Wire. The reset operation pulls down the line for 480 μ s, and then waits for an answer from the slave devices. If there is an answer the line will be

low again when the master samples. If the line is high, it means that either no slave is on the line or the reset wasn't heard by the devices.

Then there are the read and write operations. If the master wants to write a '1', it must hold down the line for 6 μ s, and let it high again for 64 μ s. If the master wants a '0' in the line, it has to pull the line low for 60 μ s. In case that the master wants to read, it pulls down the line for 6 μ s, wait 9 μ s and sample the data. All these values are the recommended ones, but a complete table of values can be found below the timing figure.



| Parameter | Speed | Min (μ s) | Recommenc | Max (μ s) |
|-----------|-----------|----------------|-----------|----------------|
| A | Standard | 5 | 6 | 15 |
| | Overdrive | 1 | 1.5 | 1.85 |
| B | Standard | 59 | 64 | N/A |
| | Overdrive | 7.5 | 7.5 | N/A |
| C | Standard | 60 | 60 | 120 |
| | Overdrive | 7 | 7.5 | 14 |
| D | Standard | 8 | 10 | N/A |
| | Overdrive | 2.5 | 2.5 | N/A |
| E | Standard | 5 | 9 | 12 |
| | Overdrive | 0.5 | 0.75 | 0.85 |
| F | Standard | 50 | 55 | N/A |
| | Overdrive | 6.75 | 7 | N/A |
| G | Standard | 0 | 0 | 0 |
| | Overdrive | 2.5 | 2.5 | N/A |
| H | Standard | 480 | 480 | 640 |
| | Overdrive | 68 | 70 | 80 |
| I | Standard | 63 | 70 | 78 |
| | Overdrive | 7.2 | 8.5 | 8.8 |
| J | Standard | 410 | 410 | N/A |
| | Overdrive | 39.5 | 40 | N/A |

Specification

As I said in the beginning, for this project I worked with 3 different hardware devices. The first is the Zilog Z8F6403 which is part of the Zilog Z8 Encore! family. These devices are 8-bit multipurpose microcontrollers with Flash memory. Some of its characteristics can be summarized in this list: 64K Flash memory, 4K RAM memory, 5 timers, 60 general I/O pins, full-duplex SPI, I²C and 2 UART ports... This is what makes Zilog Z8F6403 a perfect candidate for many embedded systems applications, like the one we developed in this project.

The second device is the 1-wire DS2401. This device is a low-cost element with a 64-bit registration number on its ROM memory. It is mostly used for registration and identification of equipment because of this unique 64-bit number. In my project it was used in some phase of it, when trying to implement the search algorithm. Besides that application, they don't have any importance in the project.

The third device is the 1-wire DS1820. This element gives the project all its functionality. It can accurately give current temperature. Its wide range of temperatures (from -55°C to 125°C) and its $\pm 0.5^\circ\text{C}$ accuracy makes it a perfect candidate for the project.

The implementation of the project is done in C language, with some embedded assembly language. The most important (even trivial) use of the embedded assembly is when I calculate the delays, where I use 6 nop instructions for a 1microsecond delay. The project is divided in different modules. There is main module where the flow of the program is controlled. Basically it is an endless while loop where the temperature is constantly being calculated. In order to do so, I had to follow the transaction sequence for accessing the DS1820:

- 1) Initialization: Consists of a reset pulse transmitted by the bus master followed by presence pulse(s) transmitted by the slave(s)
- 2) ROM command: These commands operate on the unique 64-bit ROM codes of each slave devices and allow the master to single out a specific device present on the 1-wire bus.
- 3) DS1820 function command: These commands allow the master to initiate temperature conversion, to read from and write to the scratchpad memory and to determine the power supply mode.

In order to calculate the temperature and get it to the Z8, 2 of these transaction sequences are needed:

| Master Mode | Data (LSB First) | Comments |
|-------------|-----------------------|--|
| TX | Reset | Master issues reset pulse. |
| RX | Presence | DS1820 responds with presence pulse |
| TX | CCh | Master issues Skip ROM command |
| TX | 44h | Master issues Convert T command |
| RX | 0 ... (several 0?) | DS1820 responds 0s while the temperature is being read |
| RX | 1 | DS1820 has finished reading temperature |
| TX | Reset | Master issues reset pulse. |
| RX | Presence | DS1820 responds with presence pulse |
| TX | CCh | Master issues Skip ROM command |
| TX | BEh | Master issues Read Scratchpad command |
| RX | 9 data bytes | Master reads entire scratchpad. The master recalculates CRC. If they match, continue. If not, read again |

The first one is to order the sensor to calculate the temperature. Reset followed by a presence pulse (initialization phase), then a Skip ROM command is sent. This command addresses all devices on the bus simultaneously, but as long as only 1 device is on the network, everything will work right. Last command of the first transaction is the Convert T. After sending it, the master waits for a '1' to come from the slave, which would mean that the operation has finished successfully. Once the temperature is calculated, a second transaction begins. The 2 first phases are the same, but the DS1820 function command phase is different. The master is going to read the scratchpad to get the memory where the last temperature reading is stored. It has to read 9 bytes (8 bytes from memory and the 9th byte of CRC). If the CRC matches with the information stored in the first 8 bytes, then the temperature store in the first 2 bytes of the memory is displayed in the led matrices of the Z8.

There is another module that has all the Timing functions for the 1-Wire technology. Its functions are stored in `Wire1_Timing.c` (with the definitions in `Wire1_Timing.h`). We can find here the function that calculates the 1microsecond delays, and the function that sets the speed of the communication. In the project I have been working with the standard speed (there is an alternative 'overdrive' speed). Depending of the speed, different time slots will be defined.

The next module is the one in charge of the basic communication functions in 1-Wire technology. Its functions are stored in `Wire1_Basic.c` (with the definitions in `Wire1_Basic.h`). The functions discussed in this module are from the notes in "1-Wire Communication Through Software" by Dallas Semiconductor. Here we can find the following functions:

- `int OWTouchReset(void)` : This function generates a reset, return 1 if no presence detect was found, return 0 otherwise.
- `void OWWriteBit(int bit)` : Send a 1-Wire write bit. Provide 10us recovery time.
- `int OWReadBit(void)` : Read a bit from the 1-Wire bus and return it. Provide 10us recovery time.
- `void OWWriteByte(int data)` : Write 1-Wire data byte
- `int OWReadByte(void)` : Read 1-Wire data byte and return it
- `int OWTouchByte(int data)` : Write a 1-Wire data byte and return the sampled result.
- `void OWBlock(unsigned char *data, int data_len)` : Write a block 1-Wire data bytes and return the sampled result in the same buffer.
- `int OWOverdriveSkip(unsigned char *data, int data_len)` : Set all devices on 1-Wire to overdrive speed. Return '1' if at least one overdrive capable device is detected.

The next module would be the one for calculating the CRC. The information for this module was retrieved from the notes in "Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor iButton Products - Maxim/Dallas". There is a function called `Do_CRC` which given an array of integers, it calculates the CRC. With the calculation of these CRC, we can detect these errors:

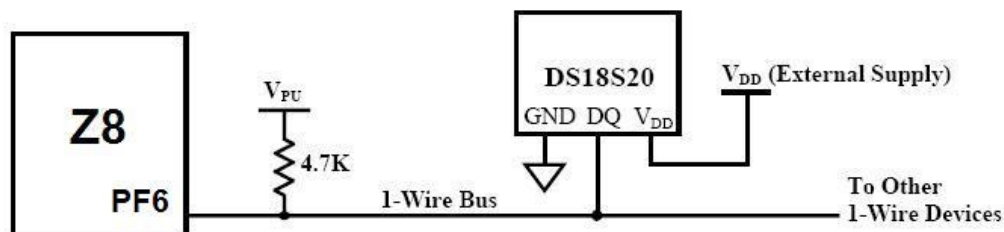
1. Any odd number of errors anywhere within the 64-bit number.
2. All double-bit errors anywhere within the 64-bit number.
3. Any cluster of errors that can be contained within an 8-bit "window" (1-8 bits incorrect).
4. Most larger clusters of errors.

If the values were correct, and the given CRC is introduced in the process of re-calculating the CRC, we should get 0x00 as a result. If so, return a 0. If not, return -1.

Finally there are some other modules, which are re-uses of older modules. There is a LEDs module for printing on the LEDs, there is a GPIO module for initializing the GPIO operations, and there is a functions.c module for printing the temperature on the LEDs.

Implementation & Construction

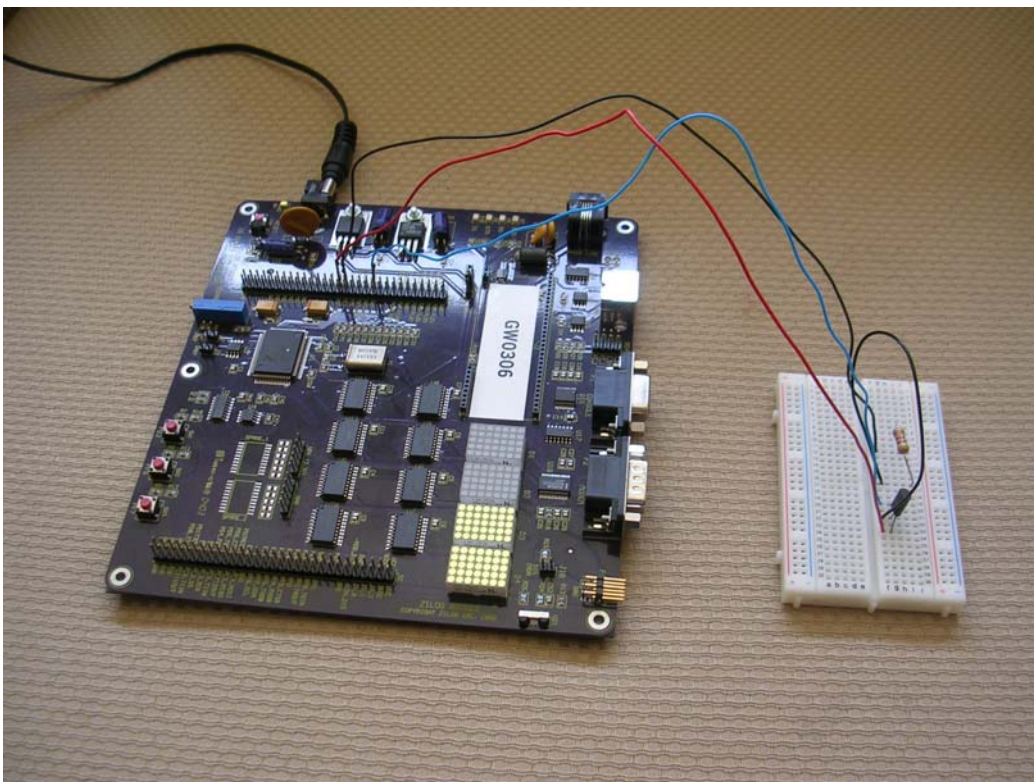
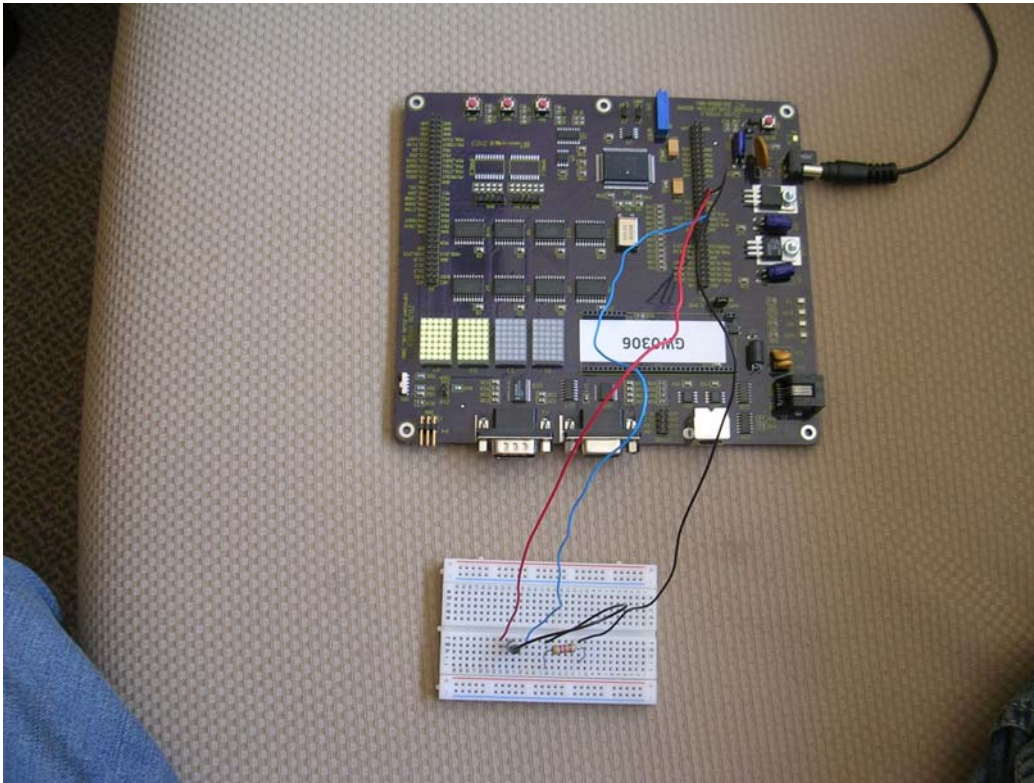
This would be a block diagram of the network.

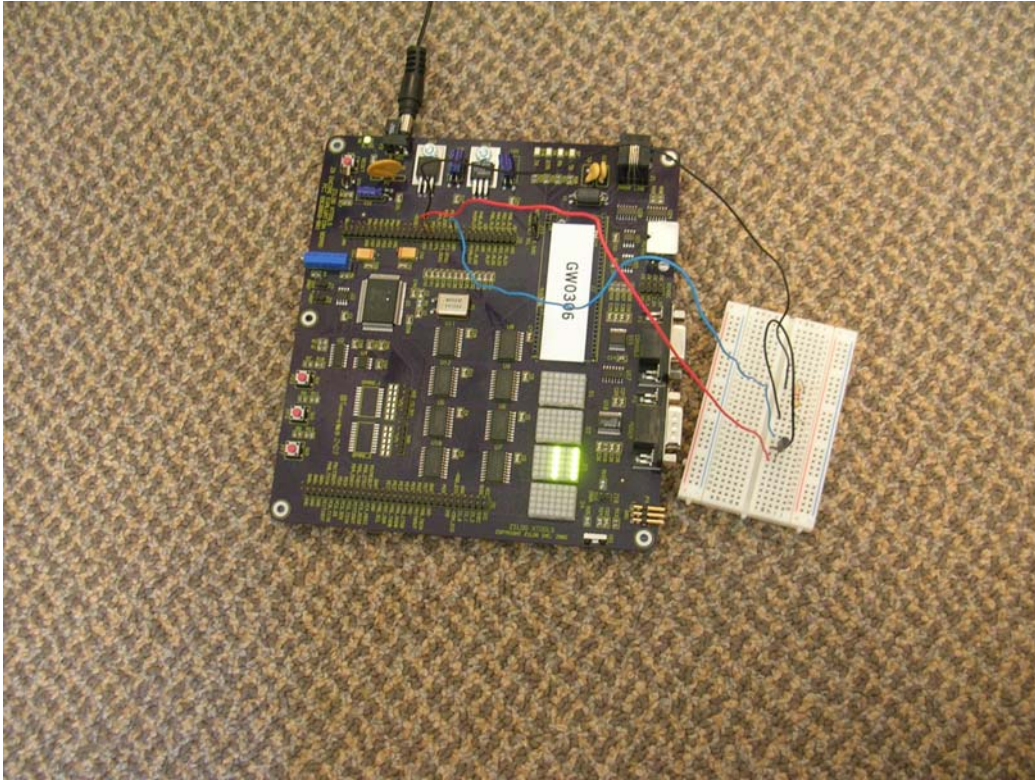


We can see a simple 1-Wire network. The Z8 will be working as the bus master. There is a pull-up resistor for the network, and then a DS1820 sensor that gets the data from the data bus coming from the Z8. We can see it gets the power supply from an external source.

The interactions with the Z8 are done through the port F6. This port has to be configured for both input and output since it is going to be sending information to the sensor and getting the information back. I couldn't configure it to be both input and output, but because of the timing I know when input and output are going to be. When we are having input, the function "int inp(void)" is called and it changes the direction of the port before reading. When we have an output, the function "int outp(void)" is called and the direction of the port changes before the writing.

Some pictures of the network





Retrospective

I think that one of the good decisions made in the project was trying to keep it simple, and not trying to work with a lot of sensors. I learnt that timing is not trivial, that things have to be planned or that code samples from documentation can help a lot.

If I could do things again, I would have started earlier. There are things that are not a matter of time, but usually everything is a matter of time. Also I would have planned better my semester (it seems like it doesn't have anything to do with embedded systems, but it does). Being here for just one year and not knowing which classes to take made me take some classes like Embedded Systems, Database Systems with Professor Narahari or Discrete Systems Simulation with Professor van Dorp which are very time consuming, and there are some weeks when something for all the classes is due.

Attachments

Included with this final report are all programs (application and test program as source code) and data sheets for all components.