Lab 6 Temperature Sensor

This laboratory assignment accompanies the book, Embedded Microcomputer Systems: Real Time Interfacing, by Jonathan W. Valvano, published by Brooks-Cole, copyright © 2000.

Goals
- Design the hardware interface between a DS1620 temperature sensor and a microcomputer,
- Implement synchronous serial communication using simple I/O directly to the clock and data pins,
- Create the low-level device driver that could be used in other applications.

Review
- Valvano Section 3.3 about gadfly synchronization,
- Valvano Section 3.4.2 about accurate time delays,
- Valvano Section 3.4.8 about handshaking with the DS1620,
- Reread Lab 1 about binary fixed point format,
- DS1620 data sheets included with this Lab Manual,
- The chapter on the parallel port and output compare in the Motorola Reference Manual.

Starter files
- DS1620.C, DS1620.H, DSTEST.C

Background

One of the basic building components of a microprocessor-based control system is the sensor. In this lab, you will interface a DS1620 to your computer, and use it as part of a temperature controller. We will simulate a digital control system that applies heat to the room in order to maintain the temperature as close to a desired temperature setpoint, T*, as possible. This is a closed loop control system because the control signals (heat) depend on the state variables (temperature). Your system will communicate with the DS1620 to estimate the current temperature, T'. In this application, the actuator has only two states: *on* that warms up the room and *off* that does not apply heat. Read about the operation of the DS1620 in general and the T_{COM} signal in particular. For this control problem to function properly there must be a passive heat loss that lowers the room temperature when the heater is turned off. A typical digital control algorithm for this type of actuator is Bang-Bang. Other names for Bang-Bang include Two-position, On-off, or Binary Controller. There are two setpoint temperatures in a Bang-Bang controller, T_{HIGH} and T_{LOW}. The controller turns on the power (activate relay) if the temperature goes below T_{LOW} and turns off the power (deactivate relay) if the temperature goes above T_{HIGH}. The difference T_{HIGH}-T_{LOW} is called hysteresis. Hysteresis extends the life of the relay by reducing the number of times the relay opens and closes.

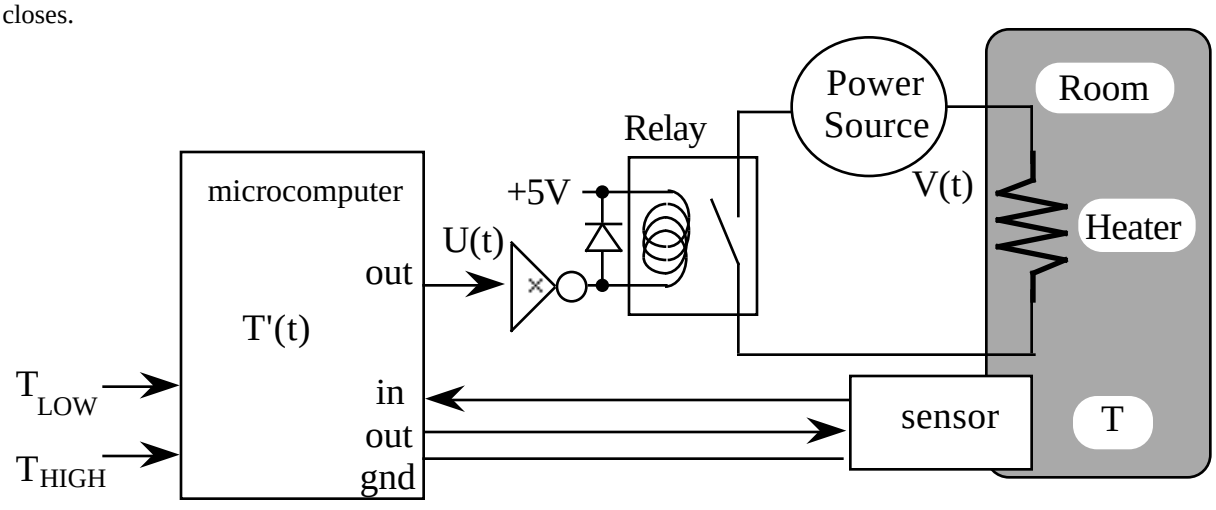

Figure 6.1. General Microcomputer-based Temperature Controller

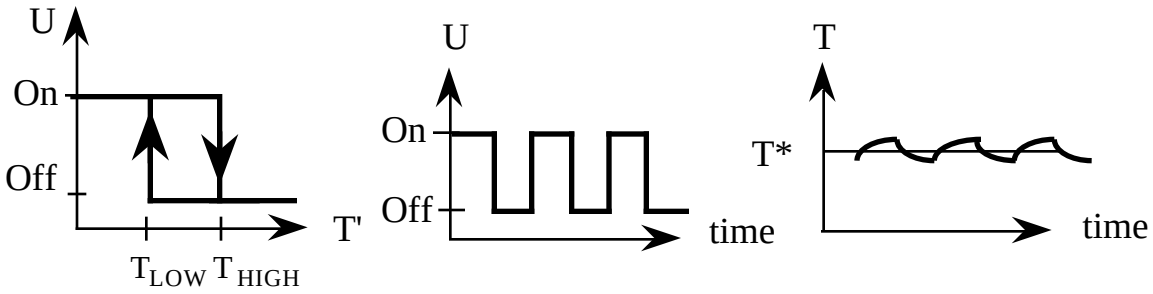

Figure 6.2. Algorithm for Bang-Bang Temperature Controller

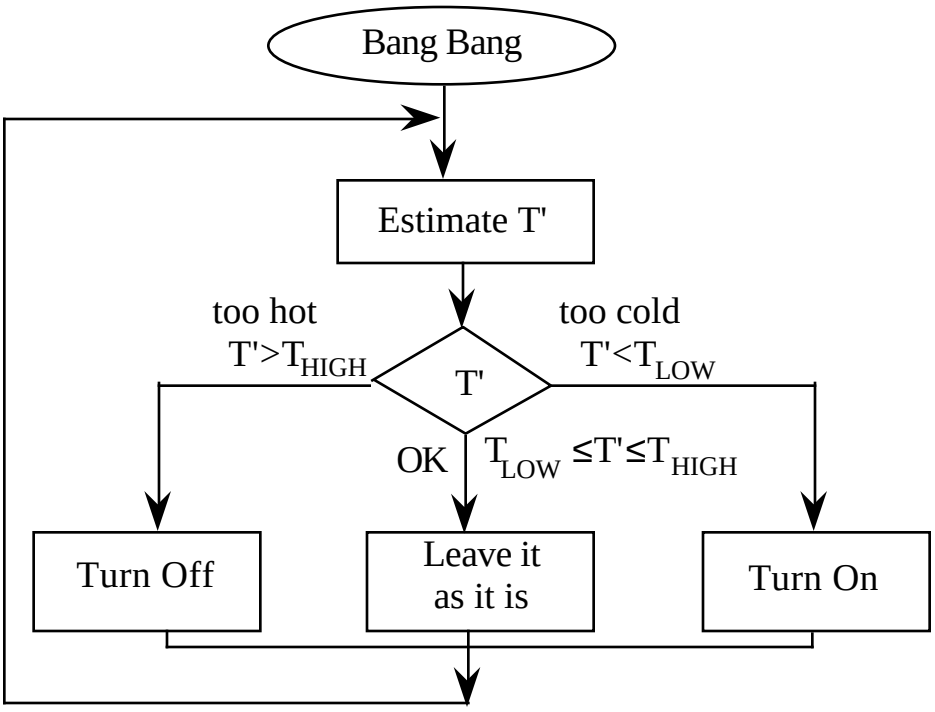

Figure 6.3. Software Algorithm for Software Based Bang-Bang Temperature Controller

Once programmed with the two setpoint temperatures, T_{HIGH} and T_{LOW}, the DS1620 will perform the above bang-bang algorithm automatically. The following figure shows an actual DS1620-based controller. The DS1620 can be programmed at the factory before installing the chip into a system. Here it is shown with a microcomputer that allows the operator to adjust the setpoints.

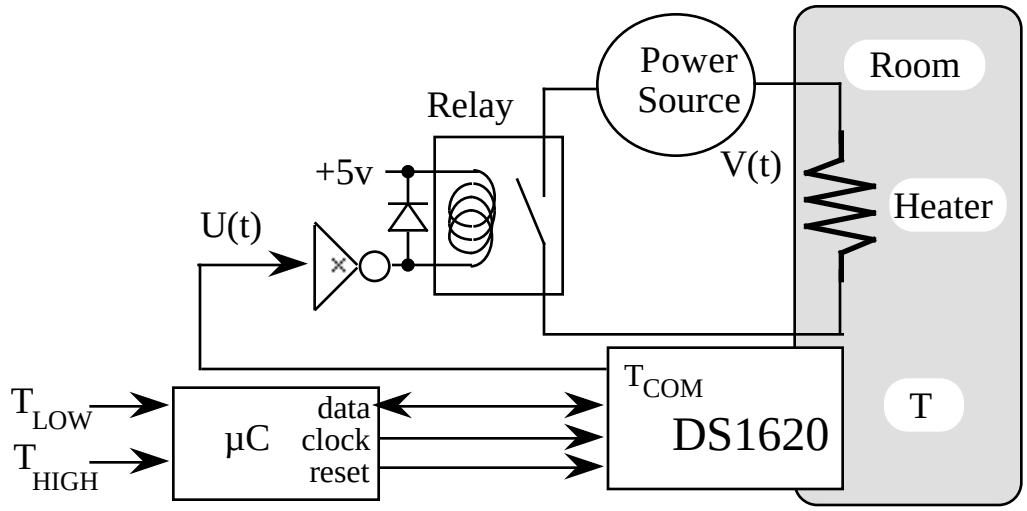

Figure 6.4. DS1620-based Temperature Controller

Instead of a relay and heater, you will connect three LED's to the DS1620. The middle LED simulates the control to the heater that would add or not add thermal energy to the room. The other two will help debug your system. The two setpoint temperatures, T_{HIGH} and T_{LOW}, will be entered using the `InFDec()` routine you developed in Lab 1. The microcomputer will send the two setpoints to the DS1620. Continuous mode will be started, and you should be able to observe the controller action on the three LED's. There are five types of communications that you can perform with the DS1610. See Table 3 and Figures 3,4 of the DS1620 data sheets. Information is sent LSB first.

1) Execute Function

This type of communication involves sending an 8-bit instruction from the 6812 to the DS1620 and no data. The two examples of this type are StartConvertT (0xEE) and StopConvertT (0x22). For these commands, you simply send the 8 bits.

2) Send Command

This type of communication involves sending both an 8-bit instruction and an 8-bit command from the 6812 to the DS1620. The only example of this type is WriteConfig. For this command, you first send the 8 bits (0x0C), then you send the 8 bits of data.

3) Receive Status

This type of communication involves first sending an 8-bit instruction to the DS1620 then receiving back an 8 bit data from the DS1620. The only example of this type is ReadConfig. For this command, you first send the 8 bits, next you switch the direction register bit for the data pin so it is an input, and then you receive the 9 bits of data.

4) Send Data

This type of communication involves sending both an 8-bit instruction and a 9-bit data from the 6812 to the DS1620. The two examples of this type are WriteTH and WriteTL. For these commands, you first send the 8 bits, then you send the 9 bits of data.

5) Receive Data

This type of communication involves first sending an 8-bit instruction to the DS1620 then receiving back a 9 bit data from the DS1620. The three examples of this type are ReadTH ReadTL and ReadTemperature. For these commands, you first send the 8 bits, next you switch the direction register bit for the data pin so it is an input, and then you receive the 9 bits of data.

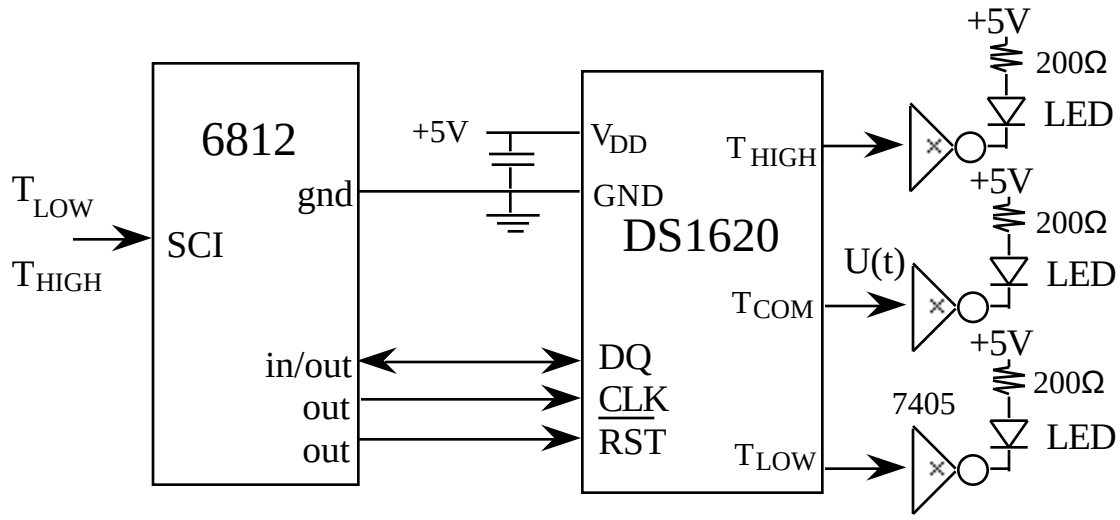

Figure 6.5. Simulated microcomputer-based temperature controller that you will build.

Preparation

Show the required hardware connections. Label all hardware chips, pin numbers, and resistor values. Ask your TA for the name and location of a demo program that communicates with the DS1620. Modify the port and bit locations to make your hardware.

Write the low-level DS1620 software interface routines. At the lowest level you should be able to send commands and send/receive data. At the next level you will develop commands to read temperature and write setpoints. Pass data into/out of these programs using signed 16-bit binary fixed-point format. Refer to Table 1 of the DS1620 data sheets and Table 1.1 in Lab1 for more information about this binary fixed-point format. You must have a separate DS1620.H and DS1620.C files to simplify the reuse of these routines. You are not allowed to perform serial port I/O (e.g., `InFDec OutString printf`) within the DS1620.C files. These operator interactions will occur in the main program. Write a main program that inputs desired temperature setpoints from the user, and transmits them to the DS1620. Implement a simple interpreter that allows the operator to perform each of the individual 9 operations with the DS1610. In addition, add a thermometer mode that runs a continuous loop repeating these steps over and over until the operator stops the command (use `InStatus`)

- read the current temperature from the DS1620,
- convert the temperature to ˚F and displays it on the PC screen (using `OutFDec`).

Procedure

Run the demo program to test the hardware interface. Start with the lowest level routines and test your DS1620.C functions in small pieces. Write a main program that performs the same operation over and over so that you can observe the synchronous serial communication on a dual channel scope.

Checkout

You should be able to demonstrate your ability to execute all 9 functions individually. Connect the Dual Channel scope to CLK,DQ and explain the signals generated when running thermometer mode.

Hints

1) Make sure the wires are securely attached to your board.
2) You can increase the temperature of the DS1620 with your finger, and decrease it with a fan. You could use one of those frozen cubes you put in your cooler, but I suggest you avoid using liquids (e.g., ice) in this lab.