Lab 2: Interfacing with a Sensor Device on an Embedded Computer System

Due Date: See the course schedule web page.

rev: 9/26/2017

Objective

- Learn how to interface sensors with embedded computer systems
- Understand bus protocols
- Understand the operation of GPIO ports

Description

By now you should have a working sensor device that can measure the light intensity in the surrounding environment, controlling a servo motor and indicating events. Moving forward, we would like to connect it to an embedded computing system to process the sensory data. In general, there are many possible ways to connect a customized device to an embedded computer, depending on the available interfaces on the computer and the sensor device. Examples include serial communication port, parallel port, USB, General Purpose I/O (GPIO), and even Ethernet. In this lab, we choose to use the GPIO port of an embedded computer (e.g. Intel Galileo) and design a customized bus protocol to acquire sensor data.

Our customized interface protocol governs the information exchange between the embedded computer and sensor device. At a higher level, the embedded computer sends commands to the sensor device, which responds with its data or status. At the bus signaling level, an LDR sensor device and a servo motor are connected to the GPIO ports of Intel Galileo, which provides data and control signals for implementing our customized bus protocol. The details of both the command protocol and bus protocol are explained as follows.

Command/Response Protocol
The Galileo board functions as the **master** of the bus protocol, and the PIC-based automation system responds to Galileo as a **slave** device.

(1) On the Galileo board, a user application on Yocto Linux issues commands to the PIC-based sensor device.

```
/* user commands */
#define MSG_RESET      0x0 /* reset the sensor to initial state */
#define MSG_PING       0x1 /*check if the sensor is working properly */
#define MSG_GET        0x2 /* obtain the most recent ADC result */
#define MSG_TURN30     0x3 /* turn the servo motor blade 30 degrees */
#define MSG_TURN90     0x4 /* turn the servo motor blade 90 degrees */
#define MSG_TURN120    0x5 /* turn the servo motor blade 120 degrees */
```

Command **MSG_RESET** will reset the state of the sensor.
Command **MSG_PING** checks if the sensor device is operational.
Command **MSG_GET** asks the sensor device to return the most recent ADC value.
Command **MSG_TURN**xxx requests the servo motor to turn the blade xxx degrees its

(2) On the sensor device, the PIC microcontroller responds to commands from the user application (on Galileo) with the following messages:

```
/* Sensor Device Responses */
#define MSG_ACK        0xE    /* acknowledgement to the commands */
#define MSG_NOTHING    0xF    /* reserved */
```

Usually a **MSG_ACK** message is replied from the sensor device after a command is received and corresponding actions are performed. Message **MSG_NOTHING** is a reserved message which should not appear in normal operations. **MSG_ACK** message follows the actual ADC value if the command from the embedded computer is **MSG_GET**.

An example command/response transaction between the sensor and the embedded computer is as follows. To obtain the most recent ADC result:

1. The user application (on Intel Galileo) sends **MSG_GET** to the sensor;
2. The sensor will output three 4-bit nibbles to the embedded system, trigger by the "Strobe" signal from the GPIO port; the user application (on Galileo) reads the three nibbles in sequence;
3. The sensor responds with **MSG_ACK** to tell the user application that it finishes reporting a 10-bit ADC result.

Bus Protocol

The PIC microcontroller on the sensor device is connected to the embedded computer through the GPIO port. A viable pin assignment is shown in Figure 1 although there can be certainly other possible assignments. Over the GPIO port, five signal lines (D3-D0 and Strobe) are available. The data bus (D3-D0) is **4-bit**. The control signal is **Strobe** driven by the Galileo board to synchronize the information exchange between the sensor device and the embedded system.

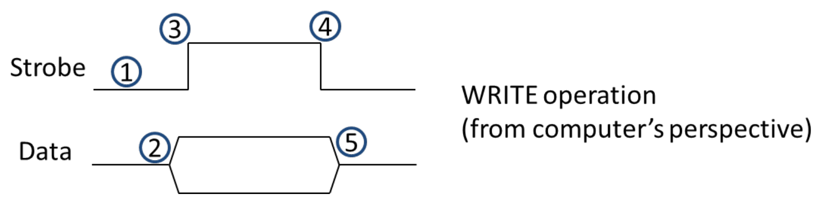

Figure 1. Write operation on the bus

The steps (in Figure 1) involved in the write operation are as follows.

(1) The computer pulls the Strobe signal low. The PIC microcontroller gets ready to read a 4-bit **command** message.
(2) The computer outputs a command on the data bus
(3) The computer raises the Strobe signal to indicate the command is ready on the bus. The PIC microcontroller starts reading the value (i.e. a **command**) from the bus. The computer maintains the value for at least 10ms.
(4) The computer ends the write operation by pulling the Strobe signal to low again. By this time, the PIC should have already finished reading the value.
(5) The computer stops putting the command on the bus. The write operation concludes.

Note that the GPIO port of the computer should be in low impedance mode when outputting the command, while the PIC should put the data pins in high impedance mode since it is receiving the value.

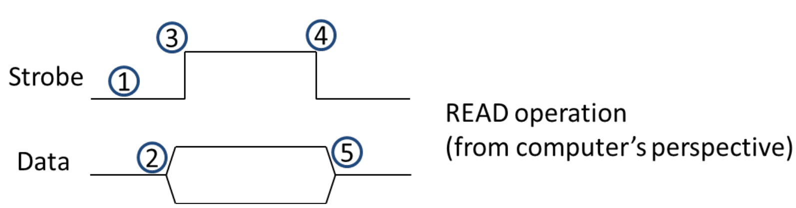

Figure 2. Read operation on the bus

Figure 2 illustrates the five steps involved in a read operation from the computer (although the timing diagram appears to have the same pattern as Figure 3, there are differences in the steps):

(1) The computer pulls the Strobe signal low to get ready for read operation
(2) The PIC microcontroller on the sensor device outputs a 4-bit value (either **MSG_ACK** or a portion of the ADC value) when it sees a low on the strobe line.
(3) The computer raises the Strobe signal and starting reading the value from the data bus. The PIC checks the Strobe signal and learns that the computer has started reading the value.
(4) The computer pulls the Strobe signal low again to indicate that the value has been read.
(5) The PIC microcontroller sees the Strobe pulled low and stops outputting the 4-bit value.

Note that the read operation can be repeated for multiple times (e.g. three times for obtaining a 10-bit ADC value).

What You Need to Accomplish in this Lab

1. Design a user space application on Yocto Linux to directly access the GPIO port of an Intel Galileo board and control the Strobe signals and data bus as described in Figures 1 and 2.
2. Wire up the sensor/motor controller device (PIC) with the GPIO port (Galileo).
3. Design firmware for the PIC microcontroller on the sensor/motor controller device to support the read and write operations as described in Figures 1 and 2. PIC automation system should be able to accept commands from Galileo Board and executing actions on PIC (turning the Servo Motor Blade to certain degree) , responding back to the appropriate command with an ADC value or ACK or NOTHING message.
4. On the Intel Galileo embedded system, implement the bus protocol, and display the 10-bit ADC result as a readable decimal number obtained through the MSG_GET command.
5. On the Intel Galileo embedded system, implement a simple user options menu for each action (PING Action , RESET Action, Get ADC value, Turn Motor Blade)

Intel Galileo Board and GPIO

You will need to setup Linux on an Intel Galileo Gen2 board, and complete the GPIO programming using C. For detailed instructions on Linux system setup and GPIO programming, please refer to the instructions_Lab2.txt and Micro2_Lab_Introduction.pdf files in the lab assignment repository http://github.com/yanluo-uml/micro2.git

Deliverables

- specified *" Micro2_Lab_Introduction.pdf " file*

Deadlines:

- specified on course syllabus on Piazza

References:

[1] PIC16F18857 datasheet.
http://www.microchip.com/wwwproducts/en/PIC16F18857

[2] Intel Galileo Gen2 Pin Configuration Table:
https://communities.intel.com/thread/55920

[3] Yocto Linux image and installation instructions,
https://github.com/yanluo-uml/micro2/lab2.

[4] Useful links:

https://emutex.com/educational/71-getting-started-with-intel-galileo-gen-2

- Handling Intel Galileo **Gen 1** GPIO Port
http://www.malinov.com/Home/sergey-s-blog/intelgalileo-programminggpiofromlinux

NOTE:

- Be aware that Intel Galileo Gen1 and Gen2 are different in terms of GPIO configuration and I/O internal expanders.
- Check its datasheet for more information.
- Arduino code is not allowed for any Galileo project.
- Make sure you use 3.3V I/Os ports, **NOT 5V** (use the jumper next to Ethernet port to adjust).