Lab 6 Considerations

55:036
Embedded Systems and System Software

Lab 6 Overview

- Develop a simple time clock
  - Employees can “clock-in” and “clock-out” by swiping their ID-card
  - When someone clocks in or out a message of the form:
    Employee 1234567890 Clocked Out at 2:34:24 PM on 03/30/07
    will be displayed on a terminal screen (RS-232)
  - The time/date can be changed from the terminal keyboard (RS-232)

Lab 6 Hardware

Terminal

Mag stripe reader

PIC

DS1305 Serial Real-time Clock

Lab 6 Hardware

Terminal

Mag stripe reader

PIC

RA0(AN0)

SPI

DS1305 Serial Real-time Clock
Lab 6 Hardware

The DS1305 Serial Real-Time Clock
- Keeps track of date/time
  - 12 or 24 hour format
  - Adjusts for different number of days in months and leap-years
- Has two alarm settings
  - Can interrupt processor at selected date/time
- Has battery-backup (& trickle-charging) capability that we won’t use in this lab
- Has on-board NV-RAM that probably is not needed for this lab
- Supports SPI and Maxim 3-wire serial interface protocols
  - We will use SPI

Figuring out the Right SPI Mode

You already know about this from lab 5
Figuring out the Right SPI Mode

Figure 5. SERIAL CLOCK AS A FUNCTION OF MICROCONTROLLER CLOCK POLARITY (CPOL)

SDO
SDI

SDO
SDI

- CKP=0 (clock idle low)
- CKE=0 (data transfer on leading edge)
- SMP=0 (sample at middle)

Writing to the DS1305

Figure 6. SPI SINGLE-BYTE WRITE
Writing to the DS1305

Can’t leave CE permanently enabled
Note: C18 SPI library functions don’t handle setting and clearing the CE. You will need to do this explicitly before/after each read/write.

Burst Read/Write Mode

RTC address is automatically incremented such that data bytes are read from successive RTC locations.

C18 SPI™ PERIPHERAL FUNCTIONS

Function Description:
- **CloseSPI**  Disable the SSP module used for SPI™ communications.
- **DataRdySPI**  Determine if a new value is available from the SPI buffer.
- **getcSPI**  Read a byte from the SPI bus.
  (same as ReadSPI)
- **getsSPI**  Read a string from the SPI bus.
- **OpenSPI**  Initialize the SSP module used for SPI communications.
- **putcSPI**  Write a byte to the SPI bus.
  (same as WriteSPI)
- **putsSPI**  Write a string to the SPI bus.
Powering the DS1305
To power the DS1305 from one source, with no battery back-up, configure the device as follows:

VCC2 \( \rightarrow \) VCC1
GND \( \rightarrow \) VBAT
VCCIF \( \rightarrow \) VCC
DS1305
GND

SPI Modes
This is the proper SPI mode to talk to the DS1305:
CKP = 0
CKE = 0
SMP = 0

Configuring the PIC SPI
OpenSPI(SPI_FOSC_64, MODE_00, SMPMID);
Correction!!! The correct initialization for SPI is:
OpenSPI(SPI_FOSC_64, MODE_01, SMPMID);
Mode_01 sets CKP=0, CKE=0, SMP=0

DS1305 Data Sheet states
Max Clock Frequency as:
600 KHz (for Vcc = 2.0 V)
2.0 MHz (for Vcc = 5.0 V)
Powering the DS1305

To power the DS1305 from one source, with no battery back-up, configure the device as follows:

\[
\begin{array}{c}
\text{VCC2} \\
\text{VCC1} \\
\text{GND} \\
\text{VBAT} \\
\text{VCCIF} \\
\text{VCC} \\
\text{DS1305} \\
\text{VCC} \\
\end{array}
\]

Now Let’s Consider the RS-232

• Basic functionality:

```
repeat forever {
    Wait for “T” or “D” command from terminal
    If command is “T” {
        Display “Time Change Requested” on terminal screen
        Display “Enter Hours (two digits)” on screen
        Read HOURS from terminal
        Display “Enter Minutes (two digits)” on terminal screen
        Read MINUTES from terminal
        Display “Enter Seconds (two digits)” on terminal screen
        Read SECONDS from terminal
        Display “Enter 0 for AM, 1 for PM)”
        READ AM_PM from terminal
        Update DS1305 time to HOURS:MINUTES:SECONDS:AM_PM
    } Else /* similar code for “D” command */
}
```

Should do some error checking here.

Now Let’s Consider the RS-232

• Basic functionality:

```
repeat forever {
    Wait for “T” or “D” command from terminal
    If command is “T” {
        Display “Time Change Requested” on terminal screen
        Display “Enter Hours (two digits)” on screen
        Read HOURS from terminal & echo
        Display “Enter Minutes (two digits)” on terminal screen
        Read MINUTES from terminal & echo
        Display “Enter Seconds (two digits)” on terminal screen
        Read SECONDS from terminal & echo
        Display “Enter 0 for AM, 1 for PM)”
        READ AM_PM from terminal & echo
        Update DS1305 time to HOURS:MINUTES:SECONDS:AM_PM
    } Else /* similar code for “D” command */
}
```

Can’t allow “blocking input” to interfere with other important functions of the application.

Should do some error checking here.
A FSM for the RS-232 Interface (First Cut)

Pseudo-Code for RS FSM States

RS_IDLE:
if (no char. in RCREG)
    RS_Next_State = RS_IDLE;
else
    RS_Next_State = RS_GET_CMD;
break;

RS_GET_CMD:
ch = character read from UART;
Write ch to the terminal screen
if (ch != 'T' or 'D') {
    Write "Bad Command" to terminal screen;
    RS_Next_State = RS_IDLE;
} else
    if (ch == 'T') {
        Write "Time Change Requested" on terminal screen
        Write "Enter Hours (two digits)" to terminal screen
        RS_Next_State = RS_GET_HR;
    } else {
        Write "Date Change Requested"
        Write "Enter Month (two digits)" to terminal screen
        RS_Next_State = RS_GET_MO
    }
break;

RS_GET_HR:
for (i=0; i<2; ++i) {
    New_HR[i] = next char. read from UART;
    Write HR[i] to terminal screen;
}
if (all chars in New_HR are digits & make sense) {
    Write "Enter Minutes (two digits);"
    RS_Next_State = RS_GET_MIN;
} else {
    Write "Bad input" to terminal screen;
    Write "Enter Hours (two digits);" to terminal screen;
    RS_Next_State = RS_GET_HR;
}
break;
Pseudo-Code for RS FSM States

RS_GET_MIN:
for (i=0; i<2; i++) {
    New_MIN[i] = next char. read from UART;
    Write MIN[i] to terminal screen;
}
if (all chars in New_MIN are digits & make sense) {
    Write “Enter Minutes (two digits):” to terminal screen;
    RS_Next_State = RS_GET_MIN;
} else {
    Write “Bad input” to terminal screen;
    Write “Enter Minutes (two digits):” to terminal screen;
    RS_Next_State = RS_GET_MIN;
}
break;

Pseudo-Code for RS FSM States

RS_GET_SEC:
for (i=0; i<2; i++) {
    New_SEC[i] = next char. read from UART;
    Write SEC[i] to terminal screen;
}
if (all chars in New_SEC are digits & make sense) {
    Write “Enter 0 for AM, 1 for PM;” to terminal screen;
    RS_Next_State = RS_GET_A_P;
} else {
    Write “Bad input” to terminal screen;
    Write “Enter Seconds (two digits):” to terminal screen;
    RS_Next_State = RS_GET_SEC;
}
break;

Pseudo-Code for RS FSM States

RS_GET_A_P:
New_AM_PM = next char. read from UART;
Write New_AM_PM to terminal screen;
if (NEW_AM_PM == '0' or '1') {
    RS_Next_State = RS_UPDATE_T;
} else {
    Write “Bad input” to terminal screen;
    Write “Enter AM/PM (0 for AM, 1 for PM):” to terminal screen;
    RS_Next_State = RS_GET_A_P;
}
break;

Pseudo-Code for RS FSM States

RS_UPDATE_T:
Update the RS1305 time to:
RS_Next_State = RS_IDLE;
break;
**Possible Lab 6 Program Structure**

ISR sets Card_Swiped to indicate that a complete User-ID has been read from a card. The User-ID is placed by the ISR into the array ID.

---

**Lab 6 Considerations Continued**

- **Main Program Loop Responsibilities:**
  A. Monitor UART for time or date change requests from terminal and handle these requests as per the RS FSM
  B. Monitor the Card-Swiped flag for indication that card has been swiped and take appropriate actions:
     - Read current date/time from DS1305
     - Determine whether "clock-in" or "clock-out" event
     - Construct appropriate display string
     - Write the display string to the UART
     - Display "Thank You" on the LCD for 5 seconds

---

**A FSM for Main Loop Responsibility B**

---

**A Minor Problem**

- How do we “run” both FSMs (the RS FSM and the TC FSM) simultaneously in the main program loop??

```c
while(1) {
    switch ( /* RS FSM */
    }
    switch ( /* TC FSM */
    }
```

*Will this work??*
A Minor Problem

• How do we “run” both FSMs (the RS FSM and the TC FSM) simultaneously in the main program loop??

```c
while(1) {
    switch {
        /* RS FSM */
    }
    switch {
        /* TC FSM */
    }
}
```

Will this work??

Why not??

The FSMs can interfere with each other—e.g. the RS FSM can block for an arbitrarily long period of time waiting for terminal input.

Pseudo-Code for RS FSM States

```c
RS_GET_CMD:
    ch = character read from UART; /* while(!DataRdyUSART());
    Write ch to the terminal screen     ch = ReadUSART(); */
    if (ch != 'T' or 'D') {
        Write “Bad Command” to terminal screen;
        RS_Next_State = RS_IDLE;
    }
    else {
        if (ch == 'T') {
            Write “Time Change Requested” on terminal screen
            Write “Enter Hours (two digits)” to terminal screen
            RS_Next_State = RS_GET_HR;
        }
        else {
            Write “Date Change Requested”
            Write “Enter Month (two digits)” to terminal screen
            RS_Next_State = RS_Get_MO
        }
    }
    break:
```

Pseudo-Code for RS FSM States

```c
RS_GET_CMD:
    ch = character read from UART; /* while(!DataRdyUSART());
    Write ch to the terminal screen     ch = ReadUSART(); */
    if (ch != 'T' or 'D') {
        Write “Bad Command” to terminal screen;
        RS_Next_State = RS_IDLE;
    }
    else {
        if (ch == 'T') {
            Write “Time Change Requested” on terminal screen
            Write “Enter Hours (two digits)” to terminal screen
            RS_Next_State = RS_GET_HR;
        }
        else {
            Write “Date Change Requested”
            Write “Enter Month (two digits)” to terminal screen
            RS_Next_State = RS_Get_MO
        }
    }
    break:
```
Pseudo-Code for RS FSM States

RS_GET_CMD:
ch = character read from UART; while(!DataRdyUSART());
Write ch to the terminal screen ch = ReadUSART();
if (ch != 'T' or 'D') {
Write “Bad Command” to terminal screen;
RS_Next_State = RS_IDLE;
} else
  if (ch == 'T') {
Write “Time Change Requested” on terminal screen
Write “Enter Hours (two digits)” to terminal screen
RS_Next_State = RS_GET_HR;
} else {
Write “Date Change Requested”
Write “Enter Month (two digits)” to terminal screen
RS_Next_State = RS_Get_MO
}
break;

Fixing the “Minor Problem”

- Need to eliminate “busy-waiting” loops from FSMs
  - Will leave it to you to figure out exactly how to do this
  - It’s not difficult—e.g. look at how the Idle states of both FSMs wait for events without blocking
- In general, need to be sure that the main program loop will always iterate fast enough to meet the needs of both FSMs.
- One additional issue:
  - What happens if someone is updating the time/date from the terminal at the same time a card is being swiped:
    - terminal output can be intermixed (cosmetic issue)
    - Something worse?????