HCS12 Space Time

Matthew James Bellafaire
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
mbellafaire@oakland.edu

Nicholas Musienko
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
nmusienko@oakland.edu

***Abstract*—The goal of this project was to create a time keeping clock which does not require the user to manually set the time. A GPS module was used to obtain time from GPS satellites which was then communicated to the Dragon12 board using the SCI1 module. The NMEA sentence of the GPS module was parsed in a SCI interrupt routine which obtained the date and time from the data. An external I2C seven-segment display was used to display the time to the user. The end result is a time keeping device which automatically sets its time within 30 seconds of being powered on and maintains accurate time throughout the day.**

I. Introduction

This project implements both the Inter-Integrated Circuit (I2C) and Serial Communication Interface (SCI) peripherals on the Dragon12 board using the HCS12 micro-controller. An external Global Position System (GPS) module communicates over SCI at 9600 baud using the NMEA sentence format. Incoming data over SCI is buffered and then parsed by the software to extract the date and time information from the NMEA sentences. An external Inter-Integrated Circuit (I2C) seven-segment display is used to display the current time to the user. In order to communicate with the seven segment display the IIC module of the Dragon12 needed to be correctly configured and the required driving commands needed to be determined. For the purpose of debugging the project in development communication with the on-board LCD was also implemented to display the raw NMEA sentences and the current data and time.

The purpose of this project was to create a time keeping clock which would require the absolute minimum of user input to operate correctly. Using the GPS module to obtain time from GPS satellites allows the clock to display accurate time with no additional connectivity or user input required. The addition of the I2C display gives the user a clear reading of the current time even at a significant distance. The I2C display is much bigger and brighter than the display already on the Dragon12 board, and is much easier to obtain the time from as a user.

II. Methodology

A. SCI Communication with GPS Module

This project utilizes the Adafruit *Ultimate GPS Breakout* which is built around the MTK3339 GPS chipset. By default the GPS module utilized communicates over UART at a baud rate of 9600. The data output of the GPS module is given in the NMEA format, a raw output of a NMEA sentence can be seen in Figure 1. NMEA sentences are transmitted from the GPS module once every second, given the amount of time required to send the data at 9600 baud a polling approach to receiving data was determined to be impractical. In order to collect the data the SCI1 module of the Dragon12 was configured to receive mode at 9600 baud, the "Receiver Full Interrupt" was enabled and an ISR was created to buffer and parse the incoming data. SCI1 was used in favor of SCI0 to avoid conflicts with uploading code to the Dragon12 board.

```
$GPGGA,182336.000,4239.9818,N,08249.2428,W,1,07,1.06,176.4,M,-34.2,M,,*50
$GPGSA,A,3,08,28,30,01,07,27,17,,,,,,1.34,1.06,0.82*0D
$GPRMC,182336.000,A,4239.9818,N,08249.2428,W,0.30,231.59,140420,,,A*72
$GPVTG,231.59,T,,M,0.30,N,0.56,K,A*31
```

Fig. 1. NMEA sentence output from the GPS module, note that the $GPRMC line contains the current UTC time (182336.000) and the current date (140420).

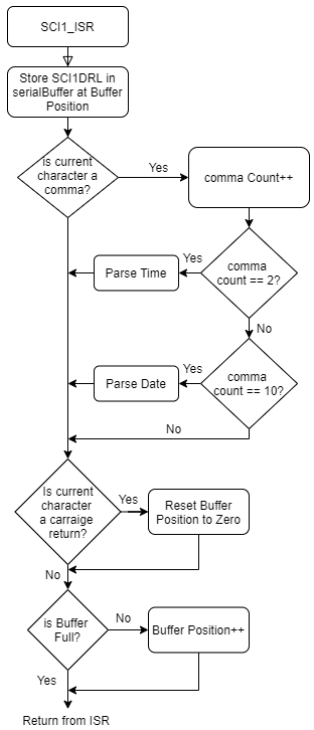

Fig. 2. Program flow chart of "Receiver Full Interrupt" responsible for receiving data from the GPS and parsing time/data information in *GPS.c*

A program flow chart of the ISR is shown in Figure 2, in addition to storing the data the ISR also parses time and date information from the NMEA sentence. NMEA sentences use commas to separate data fields with the position of each data field being fixed (eg. time is always after the first comma of the $GPRMC line), these commas are tracked and used to determine when data is parsed. In addition to this the comma's location is used as a relative position in the string to parse the desired characters from the NMEA string (eg. "$GPRMC,HHMMSS.000,..." has hour characters at 9 and 10 characters to the left of the second comma). The serial input buffer stores each incoming character until a "Carriage Return" character is received, at which time the buffer position resets to the beginning of its memory space. Since data is only read from the upper character to the lower character the buffer never requires clearing, incoming data only needs to overwrite previous data.

Using this interrupt driven approach allows the time data parsing from the GPS to run in parallel with other tasks performed by the HCS12 micro-controller. Data is parsed using the configured ISR, date and time information is accessible outside if *GPS.c* by the use of interface functions. All GPS communication functions and data parsing code is contained in the *GPS.c* file of the project. Since the GPS module communicates time in UTC the ISR also calls a function to convert to EST time, all time and date information is accessible by interface functions in *GPS.c*.

Due to the timing constraints imposed by the I2C interface of the seven segment display some special considerations were required. It was determined to run the I2C module interrupts needed to be disabled for the duration of the data transmission from the Dragon12 board to the seven-segment display. Disabling interrupts for the duration of I2C transmission resulted in infrequent errors in receiving data from the GPS module due to data being lost in the NMEA sentence. In order to correct for these errors the time parsing function needed to be adjusted to check the newly parsed time before updating the time data string. The implementation of validating the parsed time before update was performed by checking each character in the new string and ensuring that all characters were numerical characters between 0 and 9. In the case that there is an error detected in the data parsed from the GPS module the time string is not updated. Using this implementation of data validation resulted in no observable errors in the output during operation. However, when an error occurs in the parsing the displayed time will be off by one second until the next NMEA sentence is transmitted from the GPS module, since seconds are not displayed for any purposes other than debugging this solution was deemed acceptable.

B. I2C Communication with Display

In order to display the time obtained by the GPS to the user, this project implements a 1.2" Seven-Segment Display from Adafruit. This particular display has a "backpack," or a printed circuit board that acts as an interface between the HCS12 and the seven-segment display. The driver IC on the backpack is the HT16K33 RAM Mapping 16x8 LED Controller Driver, which communicates exclusively over I2C. The HCS12 has an IIC module that has I2C bus compatibility, and this module is what was used in the final design.

1) Using the IIC Module for I2C Communication: I2C is a protocol originally developed by Philips that has since become one of the standard IC communication protocols. It is a two-wire protocol that links together master and slave devices synchronously and bidirectionally. Master devices can make read and write requests to slave devices. In this instance, the HCS12 will only behave as a master, only requesting data writes to the HT16K33 slave to set the LED states. This narrows the scope of the I2C implementation considerably.

To begin using I2C, the HCS12 module needs to be initialized. Values need to be set in three control registers: IBFD, IBAD, and IBCR. Doing this requires the following process:

1) Updating IBFD to select the required division ratio to obtain correct SCL frequency.
2) Updating IBAD to define the HCS12's slave address.
3) Setting IBCR to enable the IIC interface system.
4) Modify IBCR to select Master/Slave Mode, TX/RX mode, and whether to enable interrupts.

IBC[7:0] (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
MUL = 1				
00	20	7	6	11
07	40	10	16	21
0B	40	9	16	21
14	80	17	34	41
18	80	9	38	41
1D	160	25	78	81
1F	240	33	118	121
20	160	17	78	81
25	320	49	158	161
27	480	65	238	241
28	320	33	158	161
MUL = 2				
40	40	14	12	22
45	60	18	22	32
47	80	20	32	42
4B	80	18	32	42
54	160	34	68	82
58	160	18	76	82
5D	320	50	156	162
5F	480	66	236	242
60	320	34	156	162
MUL = 4				
80	80	28	24	44
85	120	36	44	64
87	160	40	64	84
8B	160	36	64	84
94	320	68	136	164
98	320	36	152	164

Fig. 3. I2C Divider and Hold Values

First, the IIC Frequency Divider Register, or IBFD, is set to the appropriate value from Figure 3. The main factor in determining this value comes from the desired speed of the I2C bus. The regular speed of an I2C bus is 100 kHz, and the bus speed of the HCS12 is set via PLL to 24 MHz. This requires a SCL divider value of 240, and the only value for IBFD that corresponds to this divider value is 0x1F.

Next, the IIC Bus Address Register (or IBAD) is set to the value that the HCS12 should respond to in slave mode. Since this state will never occur in this application, this value is negligible.

Lastly, the IIC Bus Control Register (IBCR) needs to be set to put the HCS12 into the correct modes. The IBEN bit in Figure 4 is set to enable the IIC module. Since this application needs only to transmit data over the bus, the Tx/Rx bit is set to enable transmitting. The MS/SL bit selects whether the module

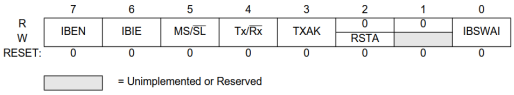

Fig. 4. IIC-Bus Control Register (IBCR) Diagram

operates as a master or a slave, and is set each time a command is to be sent.

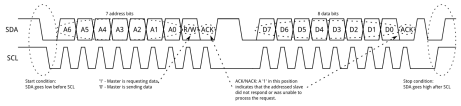

Fig. 5. I2C Bus Transmission Protocol and Waveforms

A proper I2C data transmission has several parts, as shown in Figure 6. First, it has a start condition that allows the master to take control of the bus. Then, it sends the seven address bits of the requested slave alongside a bit that tells the slave whether the command is a read or a write. If the slave receives this byte, it pulls the bus low for an acknowledge bit before preparing for the next transmission. This process, minus the initial start condition, repeats itself to send out the correct number of bytes along the bus. Once complete, a stop condition occurs to release the bus from the master's control.

The HCS12's IIC module actually comes with hardware that generates the proper start and stop conditions. By changing the MS/SL bit in the IBCR from 0 to 1, a start condition is generated along the bus. Similarly, toggling the bit from 1 to 0 generates a stop condition. There are also a number of other features that make I2C transmission easier. The IBB bit of the IBCR is set high if a start condition is detected, and cleared if a stop condition is detected. This allows the HCS12 to see if the I2C bus is busy before initiating its own data transfer. The IBIF bit of the IBCR is set high if an error state occurs or if a byte transfer successfully occurs, which allows the HCS12 to see if the slave is ready for a new transmission. Finally, the RXAK bit in that same register allows the HCS12 to see if the slave actually received the byte, clearing if a successful acknowledge signal was received.

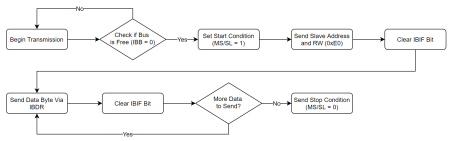

Fig. 6. I2C bus data transmission logic diagram used to send data to the HT16K33

With the module correctly initialized, the process to send a string of data over the I2C bus is relatively straightforward. The start condition occurs, then the address of the slave is sent. The HT16K33 has a standard address of 0x70, but when the read/write bit is added, the necessary byte becomes 0xE0. Then, data is written into the data register (IBDR) and transmitted over the bus until the transmission finishes. The functions written to handle I2C initialization and transmission of data are all found in *I2C.c* within the source code.

2) Communicating with the HT16K33: The HT16K33 is a general purpose LED driver, and as such has a general start-up process. In order to initialize the IC and turn on the 7-segment display, the following commands must be sent:

1) System Set - Turns on the Internal Oscillator
2) Display Set - Turns on the Display (and also sets a blinking frequency)
3) Brightness Set - Sets the brightness of the LEDs

Once these commands are accepted by the HT16K33, the display will be on and ready to display data. The HT16K33 has 8 memory locations mapped to LEDs on the 7 segment display. Each of these memory locations are 16 bits wide, and are addressed by bytes. This means that the first memory location is addressed 0x00 and 0x01, the second 0x02 and 0x03, up through 0x0E and 0x0F. Only the last 8 of these bits are connected to the segment LEDs on the 7-segment display, and only the first 5 of these memory locations are important for this application. The first, second, fourth, and fifth memory locations are connected to the first, second, third, and fourth digits on the 7 segment display, respectively. The third memory location is responsible for controlling the colon between the numbers and the upper and lower pips on the left side of the display.

In order to actually implement proper communication to the 7 segment display, an unsigned integer array of size 8 was declared as a buffer. The command for sending data to the first memory location in the HT16K33 is 0x00, which sets the pointer to the low byte of the first memory location. Via a function, this buffer can then be sent to the 7-segment display. First, a start condition occurs and the device address is sent via I2C, and then the command to write to the HT16K33's memory. Then, the low byte of the first array element is sent to the display, followed by the high byte. Next, each array element of the buffer is sent in a similar fashion until the end of the array is reached and a stop condition is forced.

Using this function, other functions were written to allow for translating ASCII characters, decimal numbers, and hexadecimal numbers to a format recognizable by the display. Functions also allow for a single digit or all 4 to be changed at once, independent control of the colon and pips, and for the digits to be individually or completely turned off.

C. LCD Display Output

In order to both provide a secondary output source for data and provide a tool in visualizing internal variables, the LCD onboard the Dragon12 was selected for use. To interface with the LCD display, there are a number of data bus pins that the HCS12 needs to drive. Each of these important pins are driven by PORTK of the HCS12.

- Register Select (RS) - If RS is 0, data is written into the command code register. If RS is 1, data is written into the data register.
- Read/Write (R/W) - Allows the user to decide whether to read or write information to the LCD.

- D4-D7 - The four data pins necessary to send information to the LCD. There are technically 8, but only 4 are necessary.
- Enable (E) - The pin used to latch data into the requested register. When new data is supplied to the data pins, a high-low pulse must be applied to the Enable pin for the LCD to latch the data.

Thus, the process to send a new byte of data involves setting the RS and R/W bits appropriately for the request, then pushing the top 4 bits to the data pins. E then gets pulsed to latch the set of data. This is then repeated for the lower nibble. The function that implements this behavior is the *sendLCD()* function within *LCD.c*.

Initializing the LCD display involves sending a variety of commands that enable the LCD, force it into the 4-bit data bus mode, and set a couple other features. All of these can also be found within the *LCD.c* file.

Once the LCD was ready to receive and display data, some utility functions were created to make using it easier. These functions can clear and/or home the LCD screen, change the location of the current character, write a character at the current cursor location, and write an entire line of characters at once. Each of these functions build on the *sendLCD()* function. With these, displaying the NMEA sentences and parsed data in real time is possible and made easy.

III. Experimental Setup

During the process of developing the software for this project the code was tested with each new addition to the code. First, the LCD display protocols were created and verified to display text declared from within the program. Then, during testing of the SCI module's code, the raw NMEA sentences were printed directly to the LCD to ensure proper communication. The parsed Date and Time data from the NMEA sentences was also printed to the LCD to aid in the development of the I2C communication. After this, the I2C communication was developed and test data was written to the 7-segment display to ensure all written functions operate as defined. In the final testing the Dragon12's on-board LCD displays the first 16 characters of the NMEA sentence on the first line and both the current time and date on the second line. Finally, the time data was printed to the I2C display in order to verify that all components of the system work together as intended in the project design.

The final validation of the system was achieved by uploading the completed code to the Dragon12 board, connected as shown in Figure 7 and allowing the program to run for an extended period of time. In the results presented here the program was allowed to run from 10AM to 8PM while being monitored closely.

IV. Testing and Results

Testing of of the overall system was performed with the outlined procedure. During testing the system was allowed to run without any user input. Throughout the day the time displayed on both the I2C and LCD displays remained accurate

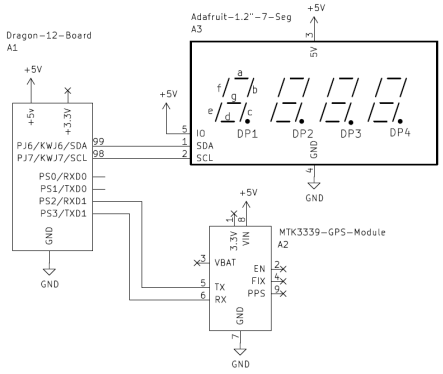

Fig. 7. Schematic of external modules connections to the Dragon12 board, the connections shown here were used for the implementation and testing of the project. (Components connected internally on the Dragon12 board are not shown)

when compared with the clock of a windows PC. Additionally there was no evidence of communication issues between the GPS module and the Dragon 12 board, throughout the testing the NMEA sentences were properly displayed on the LCD module.

When developing the LCD display protocols, the only general delay function available was one that gave a delay in milliseconds. The timing involved with the LCD was determined to operate on a scale much smaller than this, so after much trial and error a delay function that operates on a microsecond scale was added.

There were also several difficulties in initializing I2C communication to the HT16K33 driver on the 7-segment display. The advertised default I2C address for the HT16K33 is 0x70, and this was the address that was sent along the bus prior to each command byte(s) for most of development. This did not work, and the HT16K33 would not acknowledge the presence of a new byte. However, 0x70 is actually just the most significant 7 bits of the "address" byte necessary to send, and the least significant bit is the read/write bit. This extra zero shifts the entire address to the right 1 bit, and once 0xE0 was sent, the display functioned as intended.

Also, when developing the I2C interface functions, internal register-defined flags would not be read properly. This would cause the program to hang and in some extreme cases the HCS12 would become unresponsive until reset. It was discovered that disabling interrupts during functions that sent data over I2C and re-enabling them after fixed this problem. These function calls are so relatively small and low-impact that this had no effect on the interrupt-driven processes to read data from the GPS.

In integrating the GPS data to the whole system, it was found that some readings captured by the SCI module on the HCS12 were unexpected, not following the pattern of the typical NMEA string. These improper data strings would cause unpredictable behaviors within the 7-segment display. So, several error suppression techniques were created and tested. The most reliable of these seemed to be checking

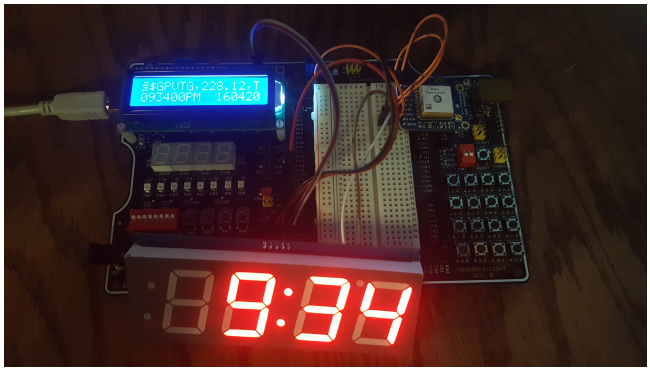

Fig. 8. Image of Dragon12 board running the code outlined in this document with the external I2C seven-segment display (bottom), UART GPS module (upper right), and on-board LCD (upper left), LCD displays the first 16 characters of the NMEA sentence on the first line and also time and date respectively on the bottom line.

the formatting of the string received over the SCI module and disregarding the data if it did not match proper NMEA formatting.

The GPS module must be able to receive signals from the GPS satellites, when these signals are blocked by significant obstructions to the module's view of the sky the project cannot attain accurate time. There is also a small amount of time that is required on startup for the GPS module to gain a fix on the GPS satellites and display accurate time. Indoors this lag is no more than 5 seconds on average, however it can vary widely depending on how obstructed the module is from a clear view of the sky. In some test conditions, it was found that the time from start-up to an accurate time message could reach almost a half-hour if there is too much obstruction.

V. Conclusions

While there were issues encountered in implementing the I2C and UART communication protocols on the Dragon12 board they were all able to be overcome. The project displays the correct time parsed from the GPS module and prints it correctly to the seven segment display. It also prints the time and date to the LCD display. In addition, there is no evidence of any significant error in the displayed time when compared with internet-synchronized clocks. There is also next to no errors visible on the 7-segment display and the time is shown almost always correctly. While there is a small delay in the project achieving accurate time on startup, this error was to be expected from the GPS module. The final implementation of this project achieved all the desired design goals.

References

[1] H.-W. Huang, *The HCS12/9S12: an introduction to software and hardware interfacing*. Delmar Cenage Learning, 2010.

[2] M. A. Mazidi and D. Causey, *HCS12 microcontroller and embedded systems using Assembly and C with CodeWarrior*. Pearson/Prentice Hall, 2009.

[3] Motorola, Inc., *HCS12 Inter-Integrated Circuit(IIC) Block Guide V02.06,* 2.06, Sept. 1999 [Revised Aug. 2002].

[4] Motorola, Inc., *HCS12 Serial Communications Interface (SCI) Block Guide V02.05,* 2.05, Jun. 1999 [Revised Oct. 2001].