Homework 10: Software Design Considerations

Due: Friday, March 31, at NOON

Team Code Name: ____J-Team ________________________________ **Group No. __10__**

Team Member Completing This Homework: _____Jonathan Chen____________________

E-mail Address of Report Author: ____________jjchen_________________ @ purdue.edu

NOTE: This is the last in a series of four "design component" homework assignments, each of which is to be completed by one team member. The completed homework will count for 10% of the team member's individual grade.

Evaluation:

Component/Criterion	**Score**	**Multiplier**	**Points**
Introduction & Summary	0 1 2 3 4 5 6 7 8 9 10	X 1	
Software Design Considerations	0 1 2 3 4 5 6 7 8 9 10	X 3	
Software Design Narrative	0 1 2 3 4 5 6 7 8 9 10	X 3	
List of References	0 1 2 3 4 5 6 7 8 9 10	X 1	
Appendices	0 1 2 3 4 5 6 7 8 9 10	X 1	
Technical Writing Style	0 1 2 3 4 5 6 7 8 9 10	X 1	
		TOTAL	

Comments:

1.0 Introduction

The RFID Xpr3ss system aims at improving the overall efficiency of the modern supermarket checkout system. The main goal is to eventually replace the UPC barcodes with RFID tags to reduce scanning time of each item and greatly simplify the checkout process by introducing solely electronic payment options. This system makes use of five significant hardware components, including a Freescale 9S12NE64 microcontroller [1], and is highly software intensive. While one of our main concerns is to keep check out time to a minimum, the visible portion of the software design (e.g. graphics displayed on LCD [2]) will also be an important factor in the overall appeal of the entire checkout process. The software will be a state machine at its core and will be controlled by a main polling loop, which will check flags set by simple interrupt handlers.

2.0 Software Design Considerations

2.1 Memory Mapping:

`$0000-$03FF`	Register Space
`$03FF-$1FFF`	RAM (7k) (inaccessible)
`$2000-$3FFF`	RAM (8k)
`>$2000-$2FFF`	Variables (heap)*
`>$3000-$3FFF`	Stack (SP starts at $4000)*
`$4000-$FF00`	Flash EEPROM (<48k)
`>$4000-$AFFF`	Main code*
`$FF00-$FFFF`	Interrupt Vectors
	* Approximations

2.2 External Interface mapping:

	Port:	Address:
Keypad:	Port H (0:3)	`$0258 (0:3)`
	Port PAD (0:3)	`$008F (0:3)`
LCD:	Port T (4:6)	`$0240 (4:6)`
	Port G (0:6)	`$0250 (0:6)`
	Port H (4)	`$0258 (4)`
Printer:	SCI0 Tx	`$00CF (SCI0_DRL)`
RFID reader:	SCI1 Rx	`$00D7 (SCI1_DRL)`
Ethernet:	EPHY (RxN, RxP, TxN, TxP)	Multiple Ports*

*Ethernet communication is provided by included libraries which perform many functions, such as maintaining a FIFO queue of data to be sent, and also placing the data on the appropriate pins at the appropriate time.

2.3 Utilization of integrated peripherals:

Built-in Ethernet Module (EPHY) – Enables communicating with remote networking stations, in this case, a remote database that contains customer and item information.

Serial Communications Interface Module (SCI0 and SCI1) [1] – Efficient communication with the thermal printer [3] and the RFID reader [4].

2.4 Organization of application code:

Interrupt-Flag-Driven, Polling State Machine (I.F.D.P.S.M)

State Machine:	The state machine will change state based on conditions at the end of the main loop. At each state certain flags will be cleared or asserted. For example, the keypad polling flag will be cleared if the keypad is not needed. Without a state machine, multiple timer channel interrupts would be needed, and that totally defeats the purpose of having a main polling, flag driven loop.
Polling:	The microcontroller communicates with the keypad by polling. The main loop calls a function that polls the keypad if a keypress is expected in that state. Polling is the best way to communicate with the keypad because of the fact that the main loop is flag driven.
Flag driven:	The main loop will be a giant IF structure, each IF statement checks for assertion of a flag. If a flag is set, then the related events will be handled, and required actions performed.
Interrupt routine:	This is the interrupt service routine called when an SCI interrupt alerts the processor that data is being written to the port. Using the interrupt to only set a flag is the most ideal for our application, as fair attention should be given to other peripherals that may require handling as well. In this way, the interrupt service routine will not consume large amounts of time and delay the main loop.

2.5 Flowchart: (Refer to Appendix A)

2.6 Provisions made for debugging

- BDM
  - This header allows us to flash the chip with our code, as well as to have real-time access to the various registers and other parameters within the processor during run-time.
- Debug Mode
  - This is a mode that aids in the identification and troubleshooting of problems with the system. It cycles through the use of the required modules and

attempts to display status information on the LCD in both character and graphical mode. It also writes to the printer in case the LCD is not functioning properly. There will be added functionality as development progresses and potential pitfalls are identified.

3.0 Software Design Narrative (Refer to Appendix B for hierarchical diagram)

- Main ( )
  - Processor init ( )
    - This is the first function called before the main polling loop begins. It initializes all necessary register values for use in the design. These include Data Direction registers, Pull Device registers, EPHY registers, and SCI configuration registers. Most of the initialization values were either created by the code development IDE (Metrowerks CodeWarrior for HC12) [5], or taken from demo code provided with the processor evaluation kit [6].
    - Completion Status: Constantly under development. Needs to be designed as other functionality is implemented.
  - LCD init ( )
    - This module initializes all of the registers necessary for the LCD [2]. The LCD will be operated in both graphical and character modes, and hence this module has two sub-modules to initialize them separately as needed. The LCD is configured by writing to it with the RS signal asserted to select a configuration register in the LCD's RAM, and then subsequently writing to the LCD with the RS signal negated to write a specific value to those registers. The LCD will be transmitted to using various control signals, and an 8 bit data bus. The data bus needs to be split between 2 ports, Port G and Port H, since Port G only has 7 accessible pins in the 80 pin package of the Freescale MC9S12NE64 microcontroller [1]. The control signals will be provided by Port T on the microcontroller, since only 3 pins are needed (LCDreset, RS, and Enable).
    - Completion Status: Sub-module successfully ported and tested. Pin reassignment and integration still needed.
  - E-mail ( )
    - This module processes e-mail sending through the built-in Ethernet module. The algorithms involved are still in the design stage, as it was discovered that SMTP protocol is not ideal for a highly-portable solution. Not all institutions intended to use this product have access to an SMTP server, and hence, it has recently been decided that a proprietary protocol, paired with a customizable software application is a better fit for the design. This method of communication is still under development.
    - Completion Status: Still under development. Algorithm recently redesigned due to difficulties in development and unreasonable portability concerns.

- Handle RFID ( )
  - This module is called by the main polling loop when the RFID interrupt flag has been asserted by its corresponding interrupt service routine. It retrieves data, in the form of a serial number, sent by the RFID reader [4] and takes appropriate action depending on the state of the system. If an RFID scan is irrelevant to the current state, the data is ignored, and the receive buffer is cleared. In certain situations, where administrative assistance is required, an administrator may swipe his keyfob and invoke this special mode, which will allow them to input a price for a faulting item, or to perform other special tasks.
  - Completion Status: Base code successfully written and tested, some functionality still needs to be added.
- Print ( )
  - This module is called when a printed receipt is desired, or if an attempt at emailing the receipt was unsuccessful. It sends data to the thermal printer through the serial port, and acts quite a bit like the RFID handle module, except that it transmits instead of reading. The printer [3] is configured manually using timed button presses, and only needs to receive what characters to print, which almost entirely conform to ASCII standards.
  - Completion Status: Successfully coded and tested. Ready for integration.
- Compare PIN ( )
  - This module is called during the appropriate state in the main state machine. It will store the 4 key presses following a customer's keyfob being scanned, and once 4 have been entered, it will compare them against the 4-digit customer PIN which was read into SRAM when the keyfob was first scanned. It will report to the main loop when all keypresses have been captured, as to whether or not the PINs matched.
  - Completion Status: Successfully coded and tested. More keypad functions needed for other functions before integration.
- DB check ( )
  - This module handles the queries to the external database. It sends the RFID serial number to the external software, and it processes the data received from the database for user and item identification and verification. It also manipulates the data and stores it in appropriate data structures in SRAM.
  - Completion Status: Still in development. Will be written when the Ethernet code is finished.

- LCD_Main.c (Update LCD)
    - Contains all functions needed to communicate with LCD.
    - Completion Status: Mostly tested and completed. Advanced graphics will require additional functions.
    - Void LCD_graphic_ini ( ) (Graphical mode init)
        - This module initializes required registers for graphical mode. It sets the display address and cursor address and clears the whole RAM to prepare for graphics information. It also sets the number of bits to be displayed on each 8-pixel block.
        - Completion Status: Successfully coded and tested. Ready for integration.
    - Void LCD_char_ini ( ) (Character mode init)
        - This module initializes required registers for character mode. It sets the display address and cursor address and clears the whole RAM to prepare for text display, as well as character pitch, character spacing and number of characters in a row.
        - Completion Status: Successfully coded and tested. Ready for integration.
    - Void LCD_write_data (int *data*) (Write graphical data)
        - This module simply writes one byte into the LCD RAM. Input can be in integer or hexadecimal format for a better representation of the graphical perspective of the data.
        - Completion Status: Successfully coded and tested. Ready for integration.
    - Void LCD_print_string (char **str*) (Write character data)
        - This module converts a string and sends it byte-by-byte to the LCD. It writes the data similarly to LCD_write_data, but the data is limited to binary representation of ASCII values of characters. Since on any data write the LCD automatically increments the cursor address by one, there is no need to reset the cursor address every time a byte of data is written.
        - Completion Status: Successfully coded and tested. Ready for integration.

- RFID interrupt ( )
    - This routine sets a flag which will be used by the main loop to call the Handle RFID module. This is ideal, as it will take very little time to actually service the interrupt, and it will allow the flow control to be maintained within the main polling loop. The actual reading of the data and storing in memory will take place within the Handle RFID module.
    - Completion Status: Successfully coded and tested. Ready for integration.

4.0 Summary

The RFID Xpress system is a very software intensive design. While a single file program might provide space efficiency and run slightly faster, the heavily modular design used in the RFID Xpress system allows for extensive debugging of each separate part, resulting in a much smoother transition from development to integration. This paper described how the Interrupt-Flag-Driven, Polling State Machine algorithm is used to efficiently control the operation of the Keypad, LCD, Printer and RFID Reader, while optimizing portability, modularity, and CPU utilization.

List of References

[1] Freescale MC9S12NE64 Microcontroller
http://www.freescale.com/files/microcontrollers/doc/data_sheet/MC9S12NE64V1.pdf

[2] CrystalFontz CFAG240128D Graphic LCD
http://www.crystalfontz.com/products/240128d/CFAG240128DFMIT.pdf

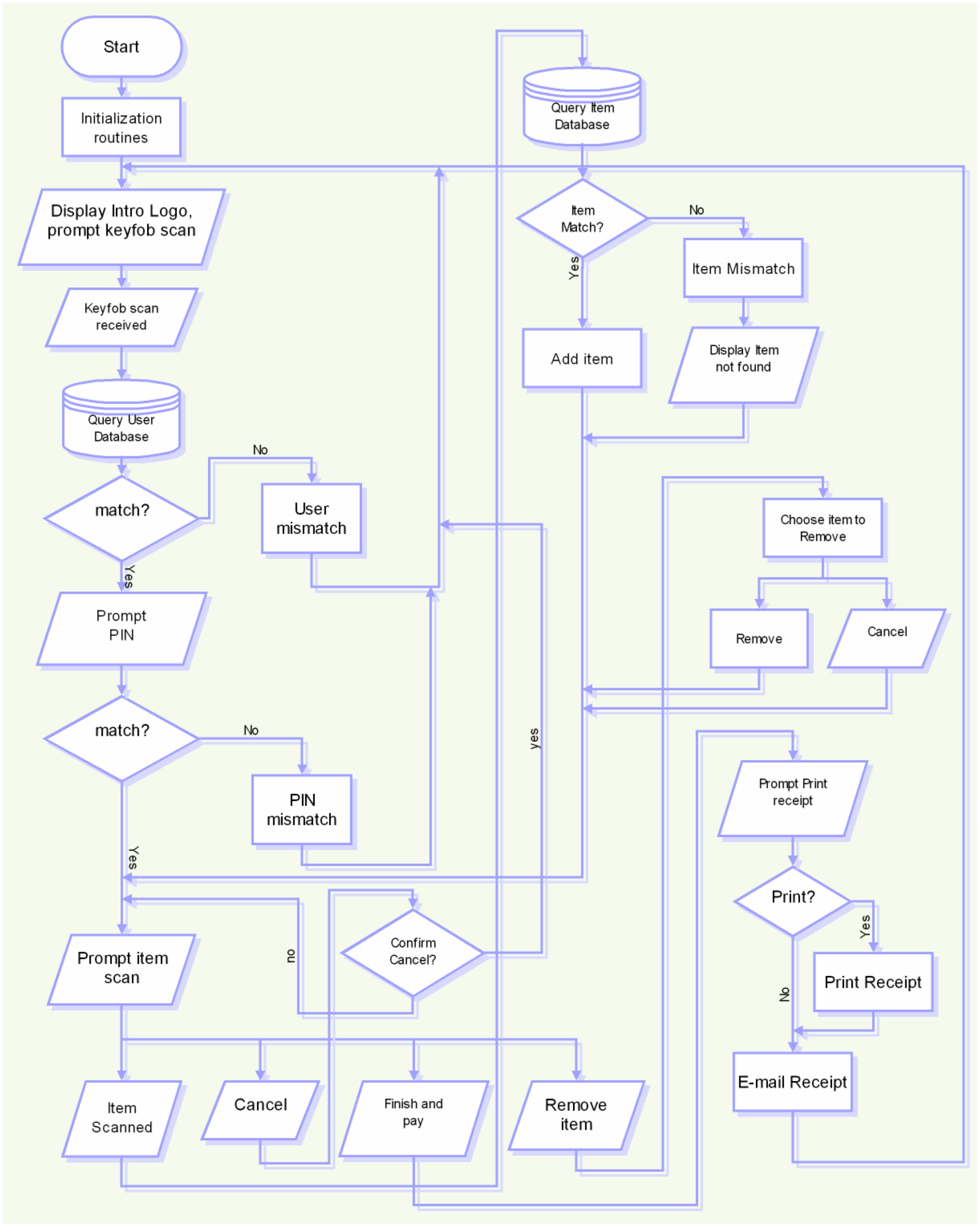
[3] Star Micronics NP-211 Thermal Kiosk Receipt Printer
http://www.starmicronics.com/printers/printers_pages/support/manuals/NP_manuals/NP211SM.pdf

[4] Intersoft Corp WM-RO-MR2 Medium Range RFID Reader
http://www.intersoft-us.com/dnload/WMROMR2.pdf

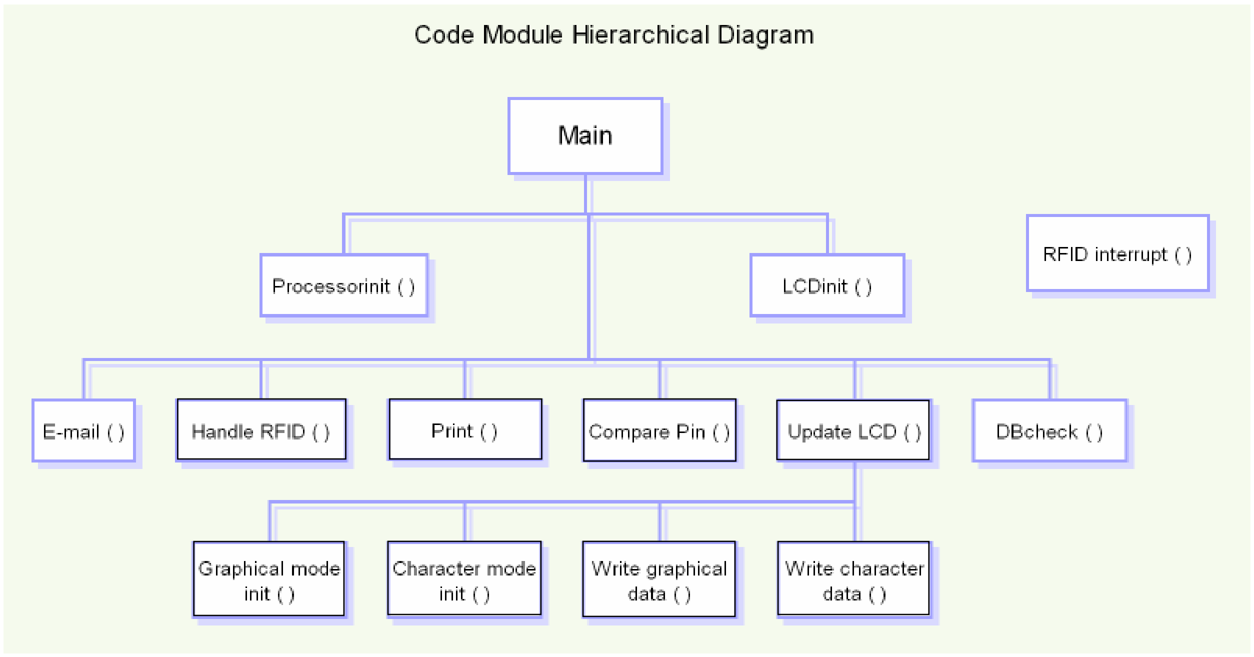
[5] Metrowerks CodeWarrior Development Studio
http://www.freescale.com/files/soft_dev_tools/doc/data_sheet/950-00081.pdf

[6] Freescale DEMO9S12NE64 evaluation kit
http://www.freescale.com/files/microcontrollers/doc/user_guide/DEMO9S12NE64UM.pdf

Appendix A: Flowchart/Pseudo-code for Main Program



Appendix B: Hierarchical Block Diagram of Code Organization



*Boxes for completed code are hot-linked to source