2023 S1 ELEC3042 Defence Questions

Name: Tyler Johnson

Student ID: 46978518

***Include state diagram(s) that describe the function of your FINAL system as an appendix. Label them clearly and reference them as part of your answers below.***

**Declaration:**

Did you write the code yourself? YES / NO

Did you do Extension 1? YES / NO

Did you do Extension 2? YES / NO

*List the state diagrams included in the appendix. List the inputs, number of states, and outputs of each state diagram. Indicate which state diagram(s) describe the flow of traffic.*

|  |  |  |  |
| --- | --- | --- | --- |
| State Diagram Name | Inputs | Outputs | Number of States |
| Traffic Phase State Machine  (Traffic flow) | S0:6, POT | LEDs, LCD Display | 10 |

**State Diagram Questions:**

*1) (a) If all sensors are triggered, what is the order of the phases of traffic according to your state diagrams?*

The phases will change according to the order: MSS => MST => SRM

*(b) Why was the order of phases in (a) selected? Why was this order better than the alternatives? Discuss with reference to the alternatives.*

I chose this order as it makes intuitive sense to try and let as much traffic through Main Street as possible. Since cars cannot turn right off main street northbound until MST phase, they will queue up during MSS. Since MST has the shortest max time, and since they would have to wait for potentially 6TP, it makes sense to let MST go next. I considered MSS => SRM => MST as an alternative order, but traffic from SR is already slowed by both the pedestrian crossing and school zone. Not as many cars would be approaching from SR as frequently, and fewer cars will be queueing at the intersection as there isn’t as much space.

*2) Complete, in detail, the timeline of the state transitions that occur for the sequence of events listed below. Assume that the system time period is set at 1 s, and at time 0 the system exits the hazard state. Further, assume that it takes 1 s for each car to pass the sensor.*

|  |  |  |
| --- | --- | --- |
| Time (seconds) | Input Sensor Event | Number of cars |
| 0 | S2 (Main Street Right Turn) | 1 |
| 2 | S1 (Main Street Northbound) | 5 |
| 3 | S0 (Main Street Southbound) | 3 |
| 10 | S3 (School Road) | 1 |
| 11 | S2 (Main Street Right Turn) | 3 |

*Fill in the table according to your state diagram.You MUST list state transitions according to your state diagrams. A vague description of what happens will not suffice for this question.*

*You should NOT change the* ***1*** *in the inputs column. However, you will need to fill in an additional 1 according to the number of cars that have not passed through the intersection.*

*The Time column is wall clock time passed, not line duration.*

*If needed, add additional lines to the bottom of the table.*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Inputs | | | | | State | | Output |
| Time (s) | S0 | S1 | S2 | S3 | Current | Next | *(List which LEDs are ON or flashing)* |
| 0 | 0 | 0 | **1** | 0 | Hazard |  | MSSY(flash), MSNY(flash), MSTY(flash), SRRR(flash) |
| 1 |  |  |  |  |  |  |  |
| 2 |  | **1** |  |  |  |  |  |
| 3 | **1** |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |
| 10 |  |  |  | **1** |  |  |  |
| 11 |  |  | **1** |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |  |
| 17 |  |  |  |  |  |  |  |
| 18 |  |  |  |  |  |  |  |
| 19 |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |
| 21 |  |  |  |  |  |  |  |
| 22 |  |  |  |  |  |  |  |
| 23 |  |  |  |  |  |  |  |
| 24 |  |  |  |  |  |  |  |
| 25 |  |  |  |  |  |  |  |

*Add rows as necessary*

**Implementation Questions:**

*3) What occurs in your main loop? Justify why each of these actions is in the main loop.*

Input handling occurs first in the while(1) loop. I check if an external interrupt (INT0) has occurred using the Expander\_Interrupt\_Occured flag and call checkButtonInputs() to determine which if any buttons are pressed. We determine inputs at the beginning of each loop as we need to process them accurately in the state machine. After that we enter the state machine, the behaviour of which is described by the state machine and truth table attached in the appendix. Immediately after the state machine we call updateLEDs(), which sends values from GPIOA\_LEDs and GPIOB\_LEDs to the port expander registers GPIOA and GPIOB respectively. This is done to limit the number of calls to the spi\_send() function, which wastes time with a busy loop. Instead of sending new data each time we want to change an LED we only send LED data twice, once for GPIOA and once for GPIOB. We store the state of the LEDs we want in variables in the meantime until the state machine has run through, so we are certain that what we are sending is purely necessary. Afterwards we call updateLCD() which reads from the current state of our timers and the system to determine what to display on the LCD. Both updateLCD() and updateLEDs() are run at the end of the main loop only once to minimize their impact on system timing and performance, as both utilize busy loops for asynchronous communication. We then set our curr\_phase to next\_phase, this is done to transition from one phase to another, and helps avoid confusion for both myself and the system. Our curr\_phase can be read from to get an accurate reading of where in the system we are, next\_phase can be redefined freely in the state machine without affecting our LCD display or our current state until after the state machine is finished.

*4) How are you using the hardware timer? Explain the purpose of the timer and why this timer was chosen. In what mode are they configured, and how often (if at all) do they interrupt?*

|  |  |  |  |
| --- | --- | --- | --- |
| Timer | Purpose and Justification | Mode | Frequency of Interrupt |
| Timer0 | Not Used | N/A | N/A |
| Timer1 | Not Used | N/A | N/A |
| Timer2 | Timer2 is used for keeping time in both HZD phase and in all other phases where time is controlled buy the POT. I chose timer2 as if the system is put to sleep timer2 will still run. Although I am not using sleep currently, I chose timer2 so that if I had the time for extensions, I wouldn’t have to change timers. I decided to condense all the utility of using multiple timers into just one to reduce the number of interrupts and conserve system resources. I was also assured that using multiple timers would result in a lower mark. | CTC | 1 interrupt/1ms |

*5) Describe in detail how timing works in your system. How are you keeping track of passing time? Why was this a good way of keeping system time?*

Timer2 generates an interrupt every 1ms, each time it runs it increments two variables, timer2\_ms\_counter and HZD\_ms\_counter. HZD\_ms\_counter is used to keep time in the HZD phase, it is reset every time we transition to HZD phase and whenever S6 is released. This is used to ensure that timing for the HZD phase is independent to the timing of the rest of the system and operates exclusively on seconds.

Timer2\_ms\_counter increments at the same rate but is used to determine if a time period has passed. When the value of timer2\_ms\_counter reaches the value of time\_period\_value the variable is reset and both time\_periods\_elapsed and time\_periods\_elapsed\_LCD are incremented by 1. We then read from ADCH to redefine the value of time\_period\_value as determined by the POT. time\_periods\_elapsed\_LCD never resets and is exclusively for displaying on the LCD screen. time\_periods\_elapsed resets along with timer2\_ms\_counter whenever a new state is entered using the resetTimer() function.

This is an effective method of keeping time as it means we are doing relatively little work in our ISR and we eliminate the need to use multiple timers to keep track of time. Additionally keeping separate variables incremented at the same time means fewer interrupts are being generated.

*6) Describe how you used the ADC.*

*(a) How often does your ADC sample the POT? Why did you choose this frequency?*

My ADC is set to free running mode, this is an issue as it caused inaccuracies in the reading of ADCH. This is a known issue that could be solved by triggering a conversion manually and checking if a conversion is completed before starting the next one. ADCH is being read at the same rate as timer2\_ms\_counter reaches time\_period\_value. I chose this rate as it means we aren’t redefining the value of a time period while in the middle of a time period, so turning the POT will affect the next time period instead of the current one. This was done to minimize errors in timing and to ensure consistency.

*(b) How did you implement the variation of duration of a period in response to POT changes?*

I used a variable named time\_period\_value to store the current ms value of the length of a time period. When the POT is turned all the way to the max value ADCH will read as 255, when all the way off it will read 0. Using the equation 100+((ADCH/10)\*36) We can remap these values from a minimum of 100ms to a maximum of 1000ms, as was the range defined in the requirements. The value of time\_period\_value is updated based on this equation.

*7) How many interrupts do you use? List each interrupt and describe its purpose.*

|  |  |
| --- | --- |
| Interrupt | Purpose |
| PCINT0\_vect | Sets a flag that an interrupt has occurred on PCINT0 |
| TIMER2\_COMPA\_vect | Keeping time in the system |
| INT0\_vect | Sets a flag that an interrupt occurred on the port expander |

*8) Describe how your system identifies which, if any, of the buttons (sensors) are pressed. How often are the buttons checked? How did you handle multiple buttons (sensors) being pressed simultaneously?*

When a button is pressed that is connected to the port expander, an interrupt is generated using an external interrupt on INT0, all that’s done in the INT0 ISR is the Expander\_Interrupt\_Occured flag is set high. This flag is then read in the main loop and runs the checkButtonInputs() function. This function reads from the INTCAPA and INTCAPB registers which capture the pin states at the moment the interrupt occurred. These registers are compared with known defined values for each button and set flags corresponding to each button high. Each button has its own flag to avoid confusion. S6 is read in a similar way, PCINT0 ISR sets a flag high which is read in the main loop. In order to handle buttons being held down between states, we call the checkButtonInputs() function in the each of the “Main” traffic states to determine if each button is still pressed. Otherwise, the flag is set low. Multiple buttons being pressed at the same time isn’t a concern, as when an interrupt occurs the INTCAP registers captures the state of every GPIO pin.

*9) Explain how you are using the LCD display.*

1. *What information is being displayed in the 32 character positions of the display?*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S0 | S1 | S2 | S3 | S4 | S5 | S6 |  |  |  |  | Time\_periods\_elapsed\_LCD | |
| Phase | | | Colour |  |  |  |  |  |  |  |  | Time\_period\_value |

1. *How often is the display being updated? How is the update being implemented?*

The value for Colour is being updated in each of the “main” traffic states, however every other value is being updated at the end of our main while loop using the updateLCD() function. This function is called once every loop. We are only writing to a specific address once per loop, this however is still somewhat inefficient. This was done to make sure that the LCD isn’t wasting time while in the main loop to keep inputs tight.

1. *How does your frequency of LCD display updates impact system performance?*

The frequency of LCD display updates does impact system performance, although I have not noticed it causing timing errors in my testing. I have attempted to minimise the amount of wasted time in the middle of the state machine by updating the LCD display at the end of the while loop, so we aren’t stuck in busy loops in the middle of the state machine. So far this

*10) A busy loop is any piece of code that loops waiting for an event, such as a flag being made ready. Have you used busy loop? If yes, describe each busy loop and justify its use. (Check in non-obvious places, like SPI or TWI waiting for a flag to be ready on byte send.)*

Busy loops are used in my code, but they are exclusively used for I2C and SPI communications. spi\_transfer has a while loop that waits till the SPIF flag goes low as is required to determine if a transfer is complete. For this circumstance I have attempted to minimize the number of spi\_read() and spi\_write() calls used in the main loop. There are a number of spi\_read() calls in the setup\_PortExpander() function, however this is only run once to initialise SPI communications. Data that is going to be sent to the port expander is stored in GPIOA\_LEDs and GPIOB\_LEDs before being sent to the port expander at the end of the while loop by the updateLEDs() function. I have also attempted to minimise the amount of spi\_read() calls made, placing them inside if statements so they are only called when necessary.

Functions associated with I2C communication also have a busy loop waiting for a flag to be ready in the I2C\_wait() function. This is used in the LCD\_Position(), LCD\_Write(), and LCD\_Write\_Chr() functions. I have attempted to minimise the impact of the time wasted by these busy loops by using the functions only where necessary. Displaying numbers on the LCD could have been achieved in a much more efficient way, however this would require a significant amount of code or the use of libraries outside of what was allowed. I have also reduced the amount of time in busy loops by eliminating the need for the LCD\_Clear() function by only writing to certain positions and using variables of fixed length so that writing to that position overwrites what was printed previously. Given more time and a deeper understanding of I2C and SPI communication I would have been able to improve the code’s efficiency.

**Appendix:**

(*Insert your state diagram(s) and state transition table(s) here.*

***Handwritten state diagrams and transition tables will not be accepted.*** )

**State Transition Diagram:**

A picture containing text, screenshot, diagram, design

Description automatically generated