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

NOTE: MST is referred to as MSR as MSTR is defined elsewhere in the registers for the ATmega328P, this was done to keep naming conventions for lights consistent with code.

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	LEDs	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 => MSR => 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 MSR phase, they will queue up during MSS. Since MSR has the shortest max time, and since they would have to wait for potentially 6TP, it makes sense to let MSR go next. I considered MSS => SRM => MSR 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. This alternative order would however be more efficient when SZ is active, as its likely that there would be increased traffic through SR during SZ time.

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	MSS	MSSY(flash), MSNY(flash), MSRY(flash), SRMY(flash)
1	0	0	1	0	MSS	MSS	MSSG, MSNG, MSRR, SRMR
2	0	**1**	1	0	MSS	MSS	MSSG, MSNG, MSRR, SRMR
3	**1**	1	1	0	MSS	MSS	MSSG, MSNG, MSRR, SRMR
4	1	1	1	0	MSS	MSS	MSSG, MSNG, MSRR, SRMR
5	1	1	1	0	MSS	MSS	MSSG, MSNG, MSRR, SRMR
6	0	1	1	0	MSS	MSS_2_MSR	MSSG, MSNG, MSRR, SRMR
7	0	0	1	0	MSS_2_MSR	MSS_2_MSR	MSSY, MSNY, MSRR, SRMR
8	0	0	1	0	MSS_2_MSR	MSS_2_MSR	MSSY, MSNY, MSRR, SRMR
9	0	0	1	0	MSS_2_MSR	MSS_2_MSR	MSSR, MSNR, MSRR, SRMR
10	0	0	1	**1**	MSS_2_MSR	MSR	MSSR, MSNR, MSRR, SRMR
11	0	0	**1**	1	MSR	MSR	MSSR, MSNG, MSRG, SRMR
12	0	0	1	1	MSR	MSR	MSSR, MSNG, MSRG, SRMR
13	0	0	1	1	MSR	MSR_2_SRM	MSSR, MSNG, MSRG, SRMR

14	0	0	1	1	MSR_2_SRM	MSR_2_SRM	MSSR, MSNY, MSRY, SRMR
15	0	0	1	1	MSR_2_SRM	MSR_2_SRM	MSSR, MSNY, MSRY, SRMR
16	0	0	1	1	MSR_2_SRM	MSR_2_SRM	MSSR, MSNR, MSRR, SRMR
17	0	0	1	1	MSR_2_SRM	SRM	MSSR, MSNR, MSRR, SRMR
18	0	0	1	1	SRM	SRM	MSSR, MSNR, MSRR, SRMG
19	0	0	1	0	SRM	SRM_2_MSR	MSSR, MSNR, MSRR, SRMG
20	0	0	1	0	SRM_2_MSR	SRM_2_MSR	MSSR, MSNR, MSRR, SRMY
21	0	0	1	0	SRM_2_MSR	SRM_2_MSR	MSSR, MSNR, MSRR, SRMY
22	0	0	1	0	SRM_2_MSR	SRM_2_MSR	MSSR, MSNR, MSRR, SRMR
23	0	0	1	0	SRM_2_MSR	MSR	MSSR, MSNR, MSRR, SRMR
24	0	0	1	0	MSR	MSR	MSSR, MSNG, MSRG, SRMR
25	0	0	0	0	MSR	MSR_2_MSS	MSSR, MSNG, MSRG, SRMR
26	0	0	0	0	MSR_2_MSS	MSR_2_MSS	MSSR, MSNY, MSRY, SRMR
27	0	0	0	0	MSR_2_MSS	MSR_2_MSS	MSSR, MSNY, MSRY, SRMR
28	0	0	0	0	MSR_2_MSS	MSR_2_MSS	MSSR, MSNR, MSRR, SRMR
29	0	0	0	0	MSR_2_MSS	MSS	MSSR, MSNR, MSRR, SRMR
30	0	0	0	0	MSS	MSS	MSSG, MSNG, MSRR, SRMR

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 so we aren't using busy loops in the middle of our state machine. 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 once to minimize their impact on system timing and performance, as both utilize busy loops for communication. We then set our curr_phase to next_phase, transitioning from the current to next phase as determined by our inputs. 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 by 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.	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. It is not used for timing in the main loop directly. 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 rate means we can have one timer for multiple variables that work independently from one another.

6) Describe how you used the ADC.

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

My ADC has a prescalar of 64 as defined by ADCSRA, with the clock frequency of the ATmega328P this means the frequency is 16MHz/64, which is 250kHz. I should have used a prescaler of 128 to achieve a sampling frequency of 125kHz, a slower frequency would result in a more accurate reading and less wasted power. I chose this frequency so the value in ADCH would update more frequently and ensure ADCH always had a result to read. My ADC is set to free running mode, this is another 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.

(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. We update time_period_value once a time period has elapsed so we aren't changing the value of the current time period, this could cause even more inaccuracies.

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.

(a) 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

(b) 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 inefficient. I am also simplifying the process of displaying each number on the LCD display by using their ASCII values. I do this by using division and mod operations to extract each digit from a number, then adding 48 before passing it to the LCD_Write_Chr() function. 48 is added as the ASCII value for 0 is 48, the value for 1 is 49 and so on. Adding 48 to a number gives us the corresponding ASCII value for that number.

(c) 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. We only update each position once per loop. I have also eliminated the need for the LCD_Clear() function, as I am only updating LCD positions that have a character displayed there already, so it automatically overwrites the data there. I also eliminated the need to turn numbers in to chars or arrays by calling LCD_Write_Chr() and using the ASCII value for the given number.

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.

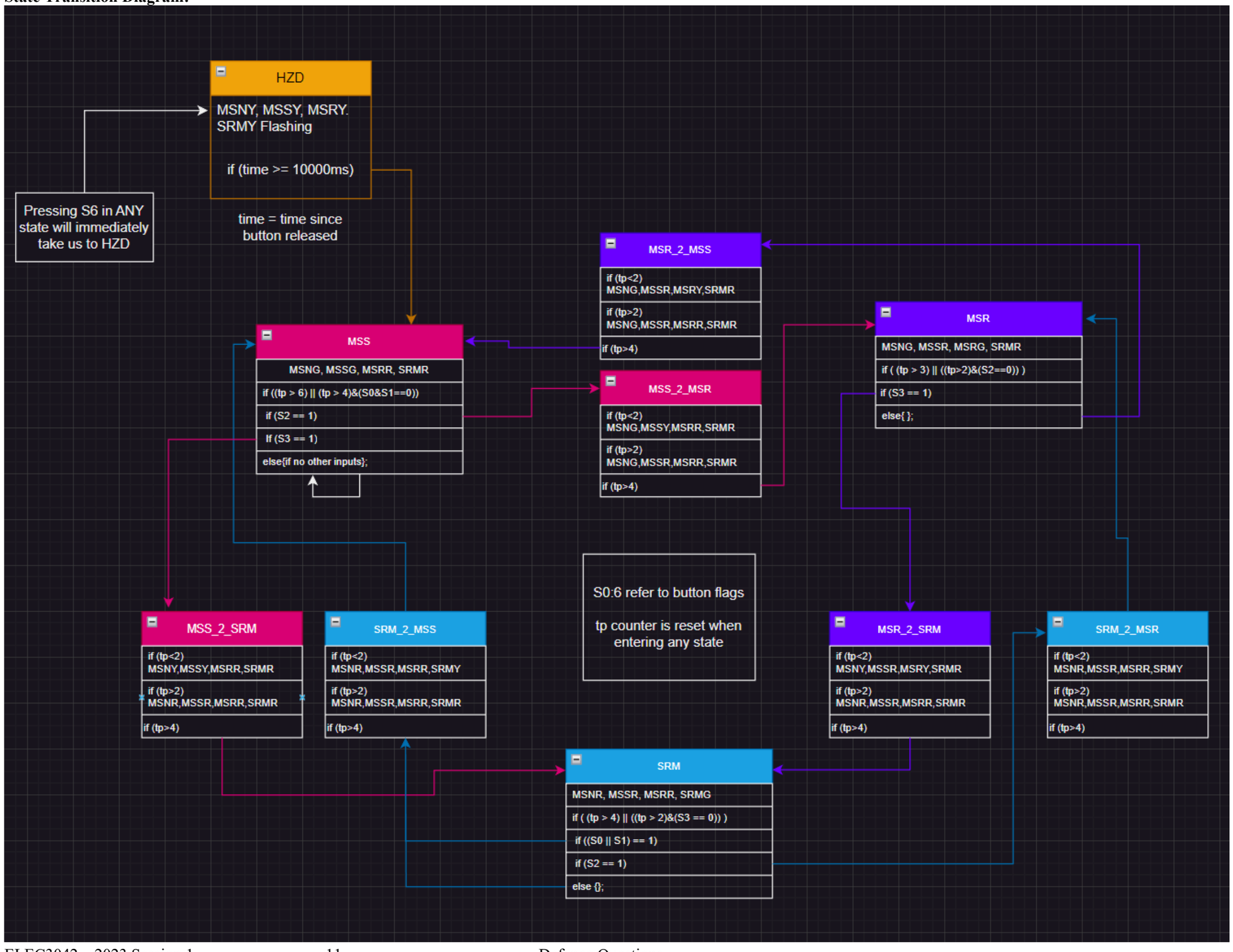
Functions associated with I2C communication also have a busy loop waiting for a flag to be ready in the I2C_wait() function. This is used in the LCD_Position(), LCD_Write(), and LCD_Write_Chr() functions. I have attempted to minimise the impact of the time wasted by these busy loops by using the functions 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:



Inputs					Curr_Phase	Next_Phase	Outputs				Flashing?
S0	S1	S2	S3	S6			MSN	MSS	MSR	SRM	
X	X	X	X	1	X	HZD	Y	Y	Y	Y	Y
X	X	X	X	0	HZD	MSS	Y	Y	Y	Y	Y
X	X	0	0	0	MSS	MSS	G	G	R	R	N
X	X	1	0	0	MSS	MSS_2_MSR	G	G	R	R	N
X	X	0	1	0	MSS	MSS_2_SRM	G	G	R	R	N
X	X	1	1	0	MSS	MSS_2_MSR	G	G	R	R	N
X	X	X	0	0	MSR	MSR_2_MSS	R	R	G	R	N
X	X	X	1	0	MSR	MSR_2_SRM	R	R	G	R	N
X	X	0	X	0	SRM	SRM_2_MSS	R	R	R	G	N
0	0	1	X	0	SRM	SRM_2_MSR	R	R	R	G	N
1	0	1	X	0	SRM	SRM_2_MSS	R	R	R	G	N
0	1	1	X	0	SRM	SRM_2_MSS	R	R	R	G	N
1	1	1	X	0	SRM	SRM_2_MSS	R	R	R	G	N
X	X	X	X	0	MSS_2_SRM	SRM	Y => R	Y => R	R	R	N
X	X	X	X	0	MSS_2_MSR	MSR	Y => R	Y => R	R	R	N
X	X	X	X	0	MSR_2_SRM	SRM	R	R	Y => R	R	N
X	X	X	X	0	MSR_2_MSS	MSS	R	R	Y => R	R	N
X	X	X	X	0	SRM_2_MSS	MSS	R	R	R	Y => R	N
X	X	X	X	0	SRM_2_MSR	MSR	R	R	R	Y => R	N

Y => R Indicates the colour changes mid state based on time