MODERN EMBEDDED SYSTEMS PROGRAMMING

Coursework

Willy Herrera

Professor Richard Anthony

ES-ESP5200

25/11/2015

Coursework 2015/2016

Designing, Developing and Testing an Embedded Open-loop Controller

Report

Name: Willy Herrera
Group: Individual

The embedded application I have implemented is: A robotic arm controller with 2-stage security level clearance and positional recorder (RC2-SLPR).

Part A

1. Introduction

The system that has been developed is considered to be a safe robotic arm controller. The reason it is considered safe is because the user has to grant access from multiple states in the system in order to get a clearance for operating the robotic arm. In this project the robotic arm is represented by a servo motor. In order to achieve a robust and user friendly program, the system has been designed to operate as a state machine where only valid inputs can switch state. The total amount of states in this system equals three. Each state is mutually exclusive and therefore do not share methods between each other. Initial state is called RFID, second state is PIN-ID, third and final state is the robotic arm controller. The user guidance is handled by serial communication and a status RGB LED. Message prompts are sent to a computer monitor and the RGB light changes color as the user is taken through each state. An overview of the user requirements and use cases can be seen in the following Figure 1.

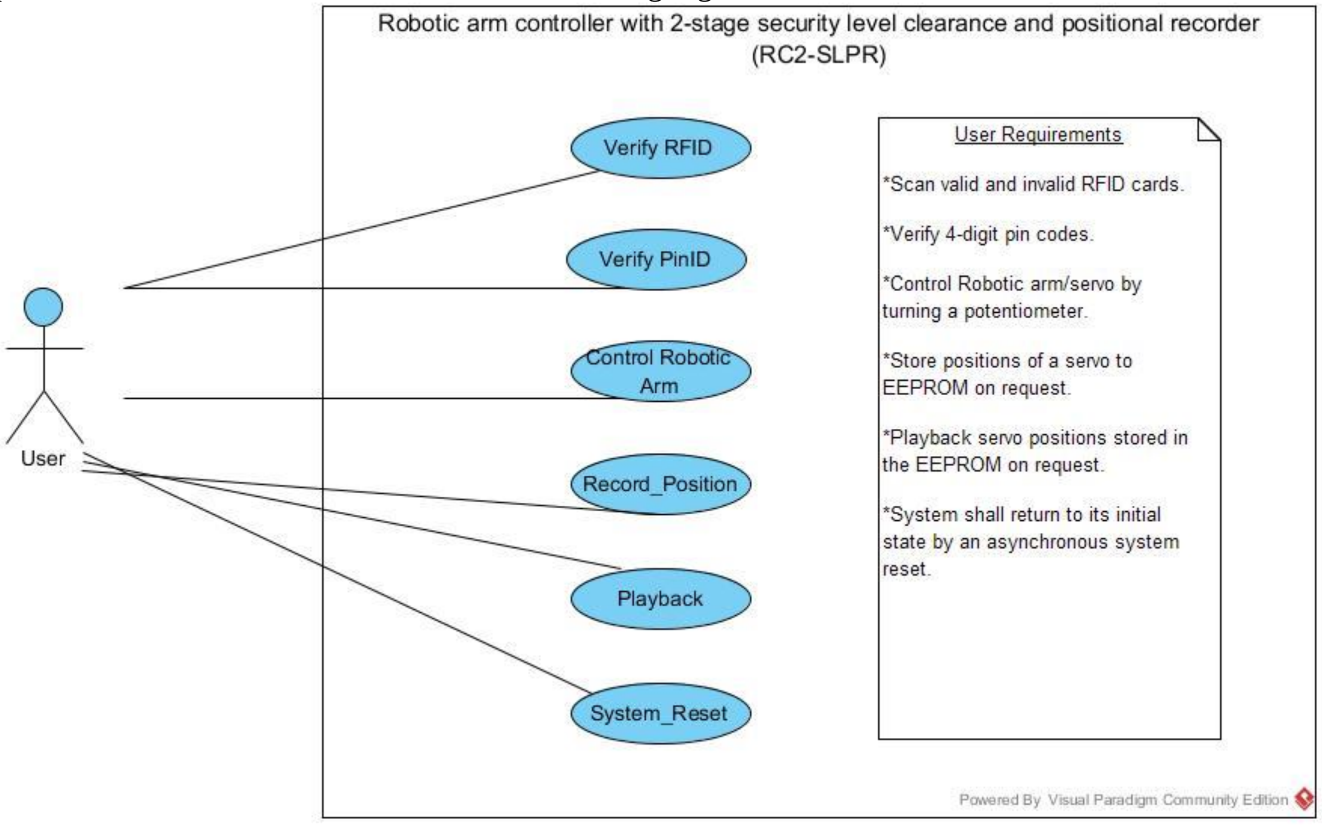

Robotic arm controller with 2-stage security level clearance and positional recorder (RC2-SLPR)
User
Verify RFID
Verify PinID
Control Robotic Arm
Record_Position
Playback
System_Reset
User Requirements
*Scan valid and invalid RFID cards.
*Verify 4-digit pin codes.
*Control Robotic arm/servo by turning a potentiometer.
*Store positions of a servo to EEPROM on request.
*Playback servo positions stored in the EEPROM on request.
*System shall return to its initial state by an asynchronous system reset.
Powered By Visual Paradigm Community Edition

Figure 1 Use Cases and User Requirements

The user starts in the initial state by scanning a valid RFID keycard. If the keycard is valid the system will move into the next state and prompt the user. On the contrary, if the keycard is invalid the system will

instead prompt the user that the used keycard is not valid and will therefore remain in the current state. Each invalid entry will also be subject to a 1.5 second buzzer tone. In the next state the user is asked to input a valid 4-digit pin on the keypad. The high level functionality of the keypad state is identical to the initial RFID state where the only way to move into the next state is by providing a valid input. Each key is associated with a unique tone which is played by a buzzer whenever a key is pressed. An invalid input will make the system remain in the same state as long as a timeout of 10 seconds has not occurred. If access has been granted and the system moves into the final state, the user gets full control of the robotic arm. By turning the controller the user can change the position of the robotic arm. In this project the controller itself is presented as a potentiometer. It is worth to mention that this final state is also the most complex in terms of its provided functionalities. Not only does it allow the user to move the robotic arm, but the system also allows positional recording of the robotic arm. The recording function can be turned on/off by an external switch. A second external switch allows the playback of the recorded positional data which forces the robotic arm to move autonomously. Finally the only way for the user to get back to the initial state is by pressing a third external button which is known as the system shutdown/reset switch. When pressed, this button will force the system into its initial state. After reset, the user is taken through the exact same procedures as the first time, making this a reliable system. The only difference between each run is whether or not there exist recorded positional data from earlier runs in the non-volatile memory section. This part of the memory is not designed to be flushed as the system restarts.
The following Figure 2 illustrates the full range of hardware and sensors that were used to implement the RC2-SLPR and which were also discussed in the previous section.

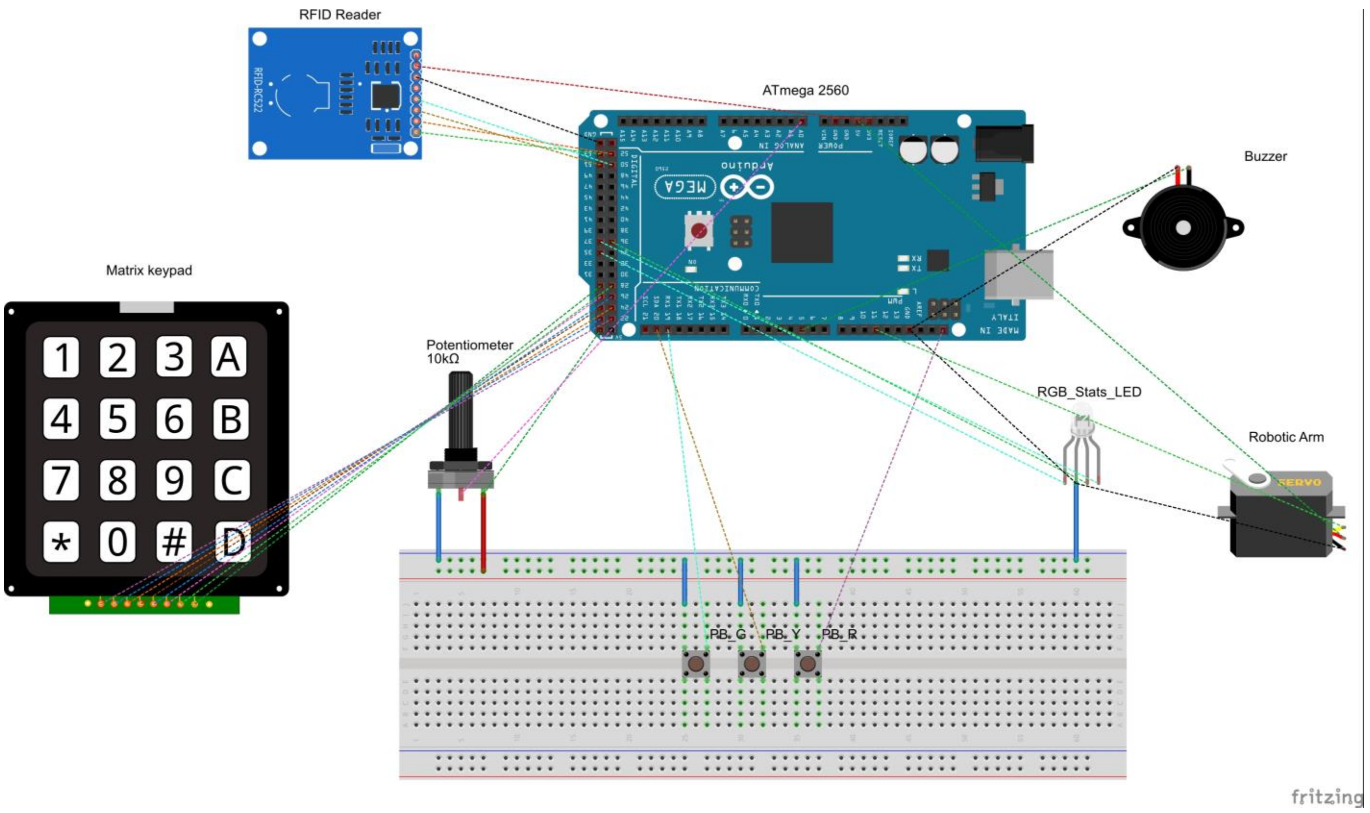


Figure 2 HW Components

It is worth to mention that the three push button abbreviations are chosen to underline the real life color of the push buttons. It is significant as for example the PB_R (pushbutton red) is the main asynchronous reset switch. Next the RB_Y is the asynchronous yellow switch which initiates a record function when the program is in the robotic arm controller state 3. Finally RB_G is an asynchronous playback function which like the record button can only be initiated during state 3. All of the switches are connected to hardware interrupts to effectively provide asynchronous input to the RC2-SLPR system. The ISR code routines are kept short which merely only set flags within the system. It is a good practice to avoid creating timing conflicts which can be easily created if for example heavy number crunching algorithms would have been placed in an ISR segment. More details on code implementation and specifics will be discussed in the detailed design part of this report.

Besides the hardware that are shown in Figure 2, there exist numerous hardware modules within the microcontroller that are being used. This Table 1, illustrate the internal hardware modules and their functionalities within the system.

Internal HW modules	Functionality
UART	Display message prompts for easier user navigation
ADC	Read potentiometer as controller input
Timer1	Provide PWM signals for Servo/timeout the keypad
Timer3	Sound buzzer tones
Hardware Interrupts INT0,INT1,INT2	Provide asynchronous reset, playback and record
SPI	Interface the RFID reader MFRC 522A

Table 1 illustrates the internal HW modules with respective functionality

Timing is key in RC2-SLPR. Table 1 also provide an overview of which internal modules are time dependent and what they do. As an example for the robotic arm to be responsive, it requires a 50 Hz pulse train which is implemented by Timer 1 in this system. More on timing requirements and their details, as well as the WCET limits of the RC2-SLPR will be analyzed in section 2 of this report. The following Figure 3 illustrate a flow chart of the RC2-SLPR.

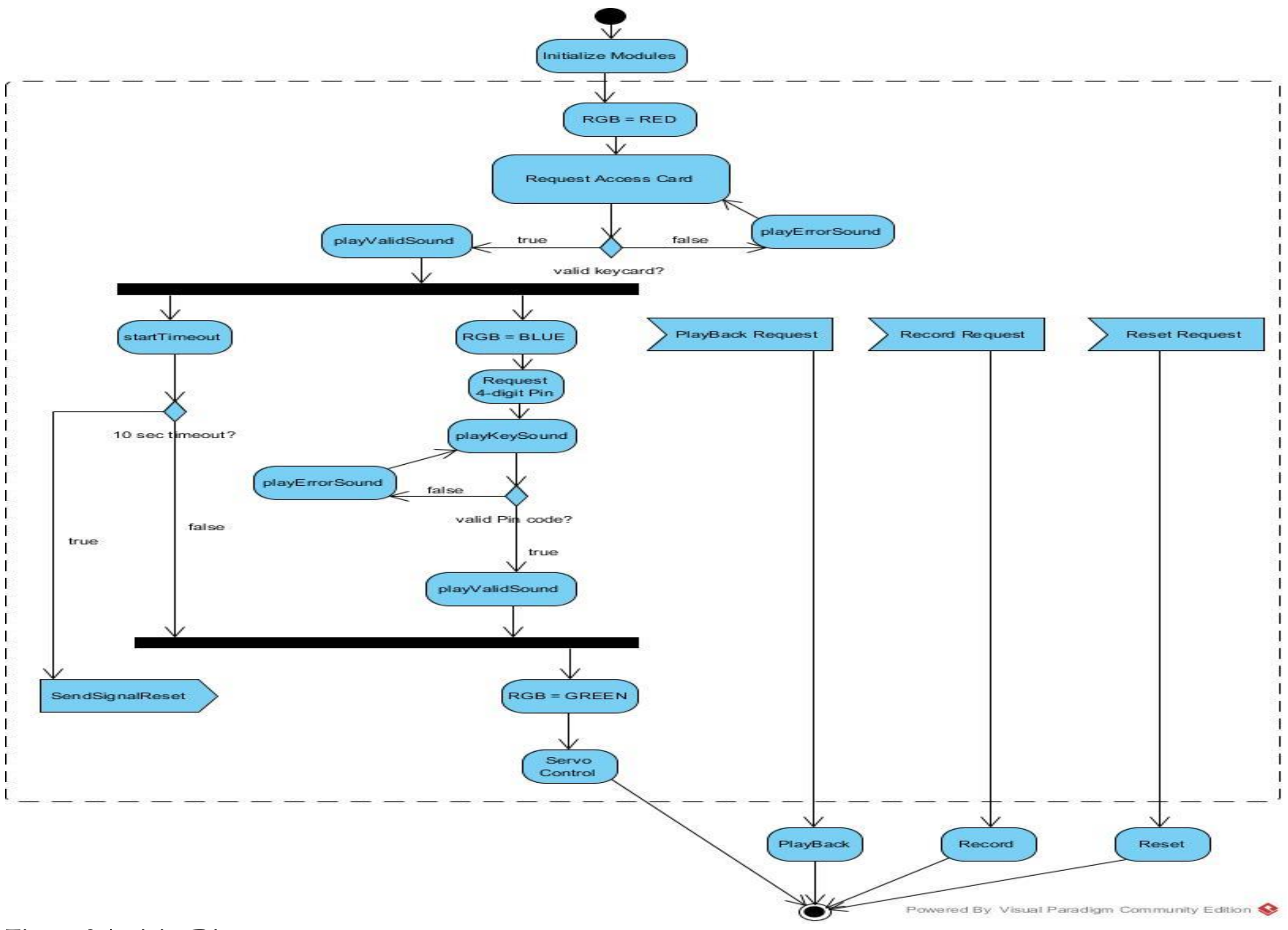


Figure 3 Activity Diagram

The flow chart, also known as an activity diagram in UML which show the different actions in a sequential fashion. This is a very powerful way of illustrating the program flow as the program counter is progressing. Notice the dotted line which form a square around most of the actions/signals of the system. This line marks the interruptible region of the program. This means that the three external interrupts which are presented as

"PlayBack Request", "Record Request" and "Reset Request" can all interrupt and re-direct the processor to the linked functions which are outside the dotted square. These could almost be seen as separate threads which are active on request for a playback, record or reset.

The main program starts by initializing all global variables and the different hardware modules that are going to be used. Table 1 showed all the main modules that were used for the RC2-SLPR. After successful initialization the status RGB led is turned red to indicate the first state of the program. It is worth to mention that the Uart module is not presented in this activity diagram as it does not present a significant functional role other than guiding the user via textual aid on a terminal.

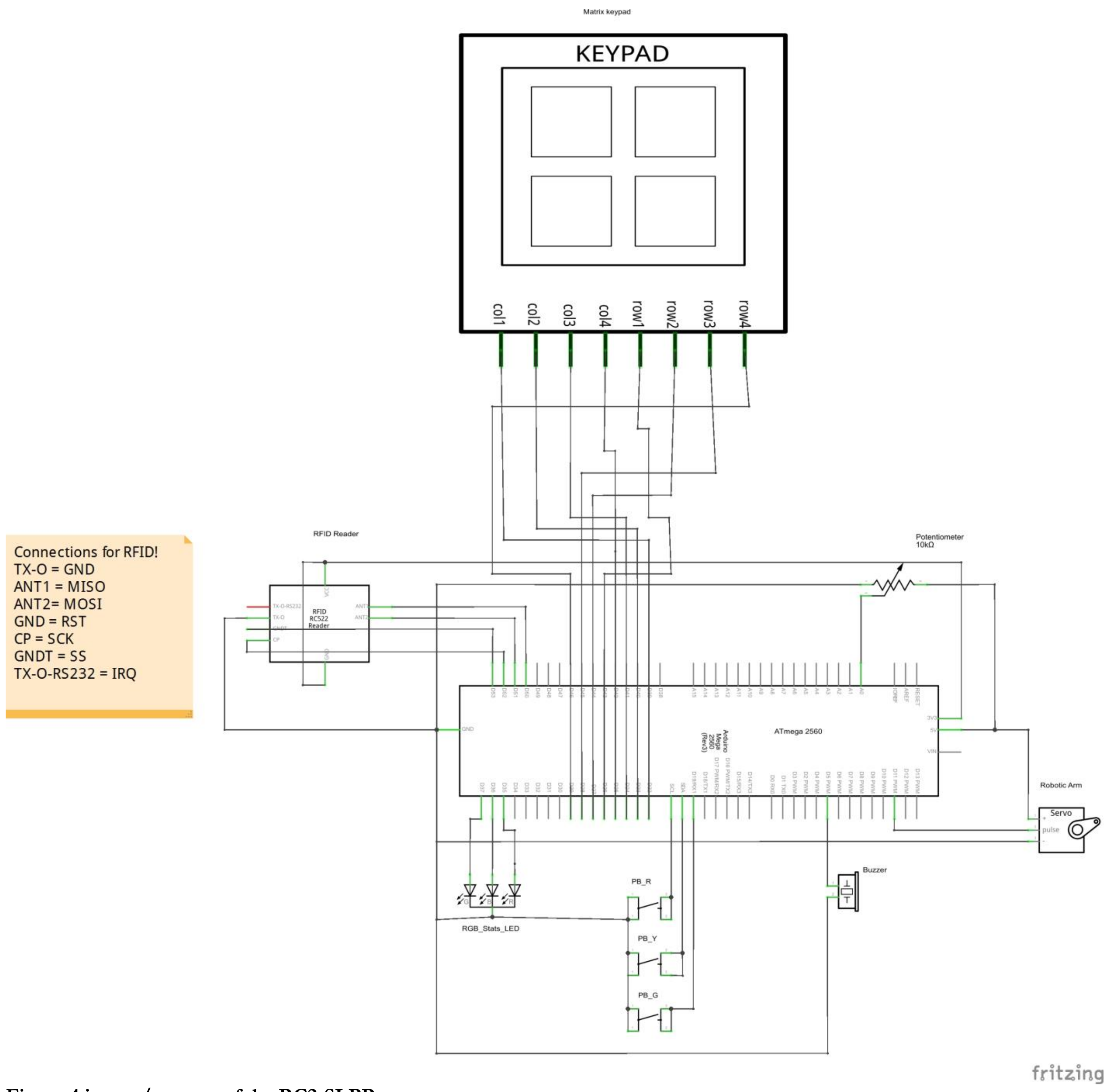
The first state handles the RFID reader which receives the serial data via the SPI protocol whenever the user presents his/her access card. This particular program has only one pre-registered access card for demonstration purposes, and its serial number is always compared to the serial number presented to the RFID reader. If there is no match, a buzzer will play an invalid tone and return to the "Request Access Card" activity. On the other hand, if there is a match then the buzzer will play a valid tone and move to the program counter into the second state.

The second state is indicated by a blue color displayed on the RGB led. At this point, the program create two threads. The first thread start a timer for a 10 second timeout. If the timer reaches the 10 second mark, a signal to reset is generated. The reset request would be accepted and therefore an interrupt would take the program counter back to state 1. The second thread handles the action of requesting a 4-digit pin code from the user. Similarly to state 1 where the RFID reader played a false tone and returned to the request action, the pin-code action keeps returning to its request state every time the entered pin does not match the pre-defined valid pin. Upon success, the buzzer play a success tone and the program counter moves into the third state of the system.

In the third and final state of the program, the RGB light a solid green color to imply that the robotic arm controller is now active (Servo Control). After having passed the 2-stage security clearance levels, the user can now freely rotate the potentiometer in order to change the position of the robotic arm. At this stage the user can interrupt the controller by any of the three external interrupts that are provided. Finally the last circular node indicates program completion, however in embedded systems a program never completes rather it goes back to its first state which in this case would be to wait for yet another access card. More in-depth details about the program execution can be illustrated by alternative diagrams such as state diagrams and timing diagrams etc. these can be found in section 2 of the report.

2. Design and Development

In this section, more details about inputs/outputs, register initialization/setup and more advanced system diagrams will be presented and analyzed. To begin with, the following schematic shown in Figure 4 highlight all the different input/output ports that are being utilized by the RC2-SLPR system.



Connections for RFID!
TX-O = GND
ANT1 = MISO
ANT2= MOSI
GND = RST
CP = SCK
GNDT = SS
TX-O-RS232 = IRQ

Figure 4 inputs/outputs of the RC2-SLPR

As could be seen in Figure 2, the RC2-SLPR system is composed of numerous hardware components which are simulating the real life environment. In order to show the mapping between the various hardware components and their real life counterparts, Table 2 is given below to address this topic as well as stating the various functionalities that are provided.

Component/Sensor	**Real Life representation**	**Functionality**	**Data Direction**	**Interface_Module**
MFRC 522A	RFID Reader	Scan access cards	In	SPI
RGB LED	RGB LED	System State Indicator	Out	GPIO
Piezo element	Buzzer	Play key press/valid/invalid tones	Out	Timer 3
Servo	Robotic Arm	Simulate a moving arm	Out	Timer 1

4x4 Keypad	Keypad	Request user pin-code	In	GPIO
10 kohm Potentiometer	Robotic Arm Controller	Control the moving arm	In	ADC

Table 2 illustrates the external HW components and their attributes

In order to get a detailed port setup overview of all the internal and external hardware modules, the following Table 3 will show all the relevant port configurations of the RC2-SLPR.

Module	**Hardware**	**Register setup**	**Access port**	**Special remark**
Timer1	Servo control	TCCR1 configured for fast PWM, no prescaler, channel A ICR1 sets operating frequency to 50 hz OCR1 sets servo position TIMSK irrelevant ----------------------- TCCR1 configured for Normal mode, use prescaler = 8. TIMSK use overflow interrupt, will overflow every 500 msec	PB5 as output for channel A	Timer1 has 2-modes of operation. It multiplexes between generating PWM signals and Normal timing for timeouts
Timer3	Piezo element	TCCR3 configured for fast PWM, no prescaler, channel A&B ICR3 sets tone frequencies, provided by lookup tbl OCR3 sets servo position TIMSK irrelevant	PE3 as output for channel A, channel B not used	Input capture pin ICP3 PE7 should be configured as output to prevent random noise
Hardware INT 0..2	External INT's	EICRA configure INT0..2 for falling edge triggering EIMSK Enable INT0..2 EIFR clear flags	PD0,PD1,PD2 as inputs with internal pullups attached	
RGB Status Led	RGB Led	DDRC as outputs PORTC initially set to low	PC0 Output Red PC1 Output Green PC2 Output Blue	

Keypad	4x4 Keypad	DDRA = 0xF0;
		Columns as inputs, Rows as outputs
		PORTA = 0xFF;
		Attach pullups to columns and set rows to be high
			PA 0..3 Column IN
			PA 4..7 Row Out	

UART	UART0	UCSR0, double USART Tx speed, Rx,Tx interrupts enabled, 8 bit data size, asynchronous UBRR0 set to 9600 baud	Rx0 on PE0 Tx0 on PE1	
RFID	MFRC-522A	SPI configuration Mosi,Miso,SCK,SDA,RST,GND,VCC	RST set to high SDA(SS) = PB0	The module operates at 3.3V
ADC		ADMUX, left adjust, AVCC as ref, ch0 ADCSR, ADC enable, Auto trigger, Int enabled, pre-scaler = 32 DIDR, disable digital input on ch0 ADCSRA, start conversion	PF0 as analog input, channel 0	Left adjust makes use of 8bit data

Table 3 Register map for RC2-SLPR, all configurations are 100% compatible with the MCU clocked at 1 MHz

The RC2-SLPR utilizes numerous time variant modules which are effectively multiplexed and share a common processor. Therefore the following Table 4, show the prioritizations of all the time based modules. The table summarizes details about periodic Timers as well as asynchronous interrupts and their functionality within the system. Finally the WCET is stated in the modules in which time is critical. Notice that Table 4 lists the respective priorities in relevance to each other, the lower number represents a higher priority.

Module Type	**Priority**	**Function**	**WCET(attention to time units)**
Interrupt 0	1	Reset	20 us
Interrupt 1	2	Record	-
Interrupt 2	3	Playback	-
SPI	6	Read Access Card	48 ms
Timer 1	4	Generate PWM/Timeout	10 s
Timer 3	5	Generate PWM	-
ADC interrupt	8	Servo Controller	-
UART interrupt	7	User textual guidance	-

Table 4 illustrates all the time dependent components of the RC2-SLPR and their preferences

In order to demonstrate the significance of the timing aspect in RC2-SLPR, two different types of timing diagrams have been developed in UML. The first timing diagram in Figure 5 is mostly concerned about the sequence of program events with respect to time. As with all previous diagrams of the RC2-SLPR system, the UART based user guidance system is not presented.

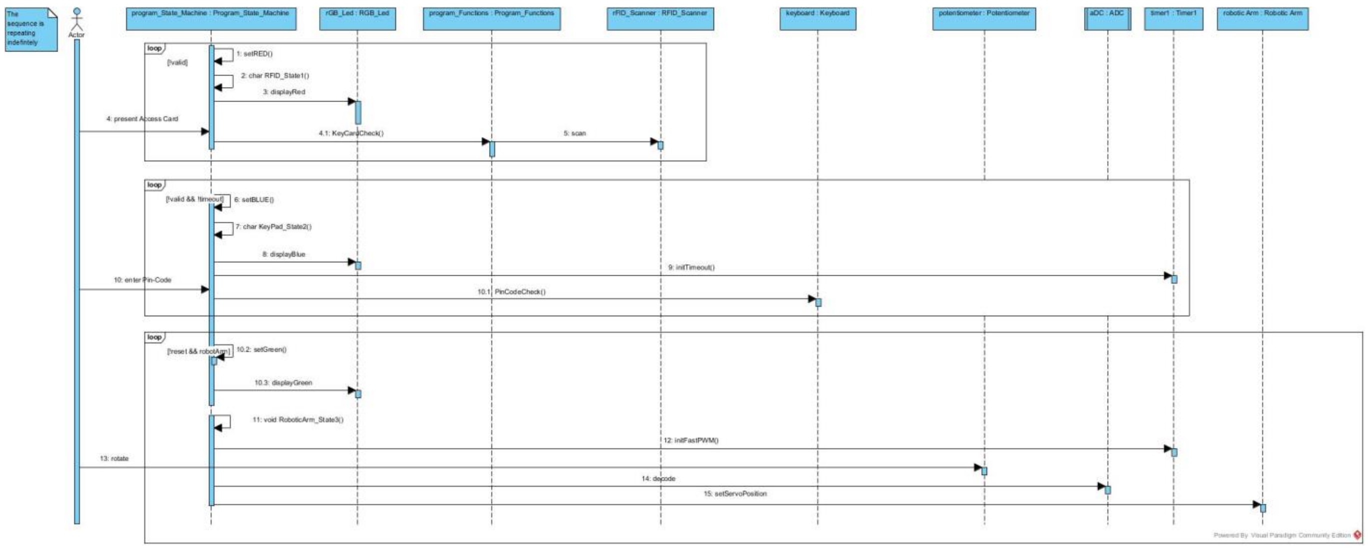


Figure 5 Sequence diagram of RC2-SLPR

The sequence diagram is composed of three states that define the architecture of the RC2-SLPR. When the system is powered on it automatically starts in the initial state. The RGB module turns red to indicate the first state, thereafter it waits for the user to present an access card. If the access card is valid the system proceeds to the second state, if not it simply stays in the same state and the user can try again.

The second state is similar to the first but here it is represented by a blue color on the RGB module. A 10-second timeout is initiated as soon as the second state becomes active. The user is allowed to enter as many 4-digit pin code combinations within 10 seconds. If the user is not successful with entering a valid code within the timeout limit, the systems restarts in the initial state. On the other hand if the pin-code is valid the system progresses to the final robotic arm state.

The third and final state is indicated by a green light on the RGB module. This indicates that the system is now active and the user can operate the robotic arm by the use of a potentiometer. By rotating the potentiometer the ADC converts the variable resistance to voltage on the output. The output is then implemented into a control algorithm to represent different pulse widths which define the positions of the robotic arm. Finally the third state is active until the user press the reset button which takes the system back into its initial state.

In order to effectively demonstrate how time is relevant to each operations, the timing diagram in Figure 7 show the relative timing of each operation in terms of an arbitrary time unit t(u).

By analyzing Figure 7, it is noticeable that the different lifelines are based on the time dependent modules that were listed in Table 4. User activities along with signals that are interpreted as valid/invalid, stay active for the majority of the time. The first lifeline represents the user as the customer. Customer events are always time consuming where the shortest event is presented as the press of a reset button. Different system modules respond to the customers actions. The responses are quick but remain active for long periods of time after being activated. However the slowest system response which accounts for several time units, is the actual decoding procedure of the presented RFID card ("Card Detected"). This can be compared to a quick system response such as the setting the "Robot Control" to active state a couple time units after pin validation. Finally the absolute quickest function with respective to time, are the hardware interrupts which are presented in Figure 7 as INT0 and INT1. An alternative compact version of the same timing diagram can be seen in Figure 6.

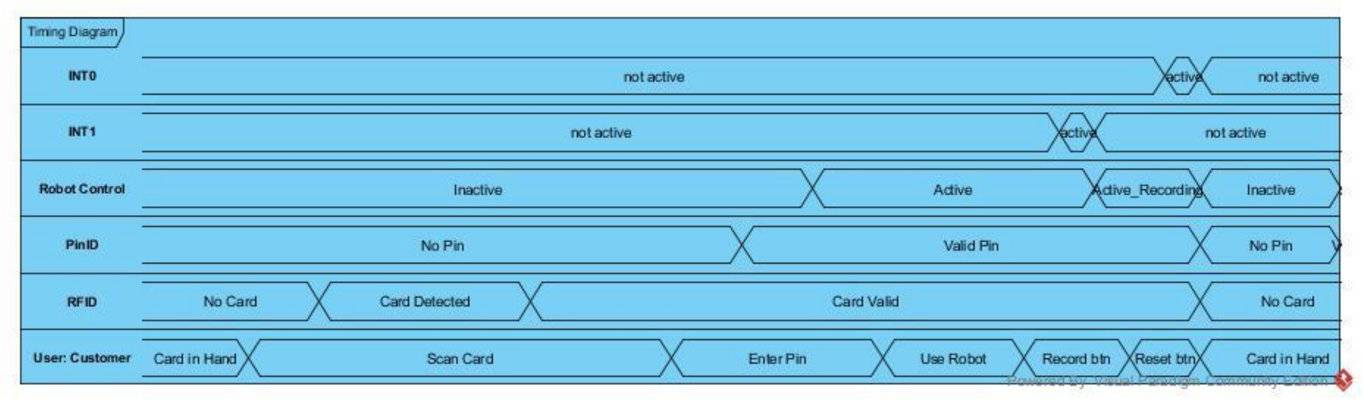


Figure 6 compact timing diagram

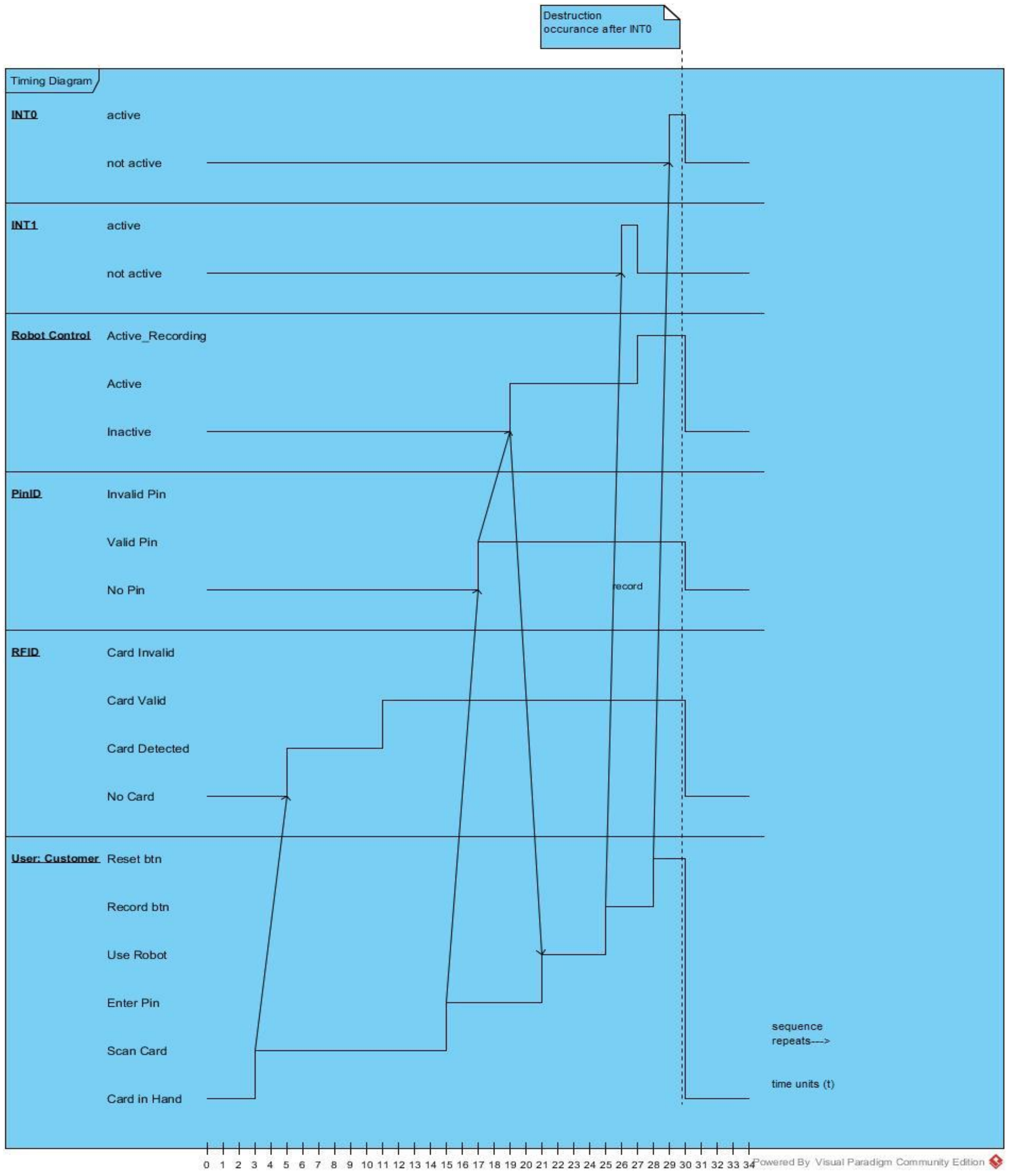


Figure 7 timing diagram

It should be clear by now that the reliability of RC2-SLPR is heavily dependent on its triple state architecture. The architecture based on three states was intentionally designed so that maintenance and troubleshooting could be located and fixed with minimal effort. It could only be realized if the three main states of the RC2-SLPR were truly or close to mutually exclusive. With this design of the architecture each state could focus on its own responsibility within the system. The following Figure 8 show the architectural class diagram of the RC2-SLPR.

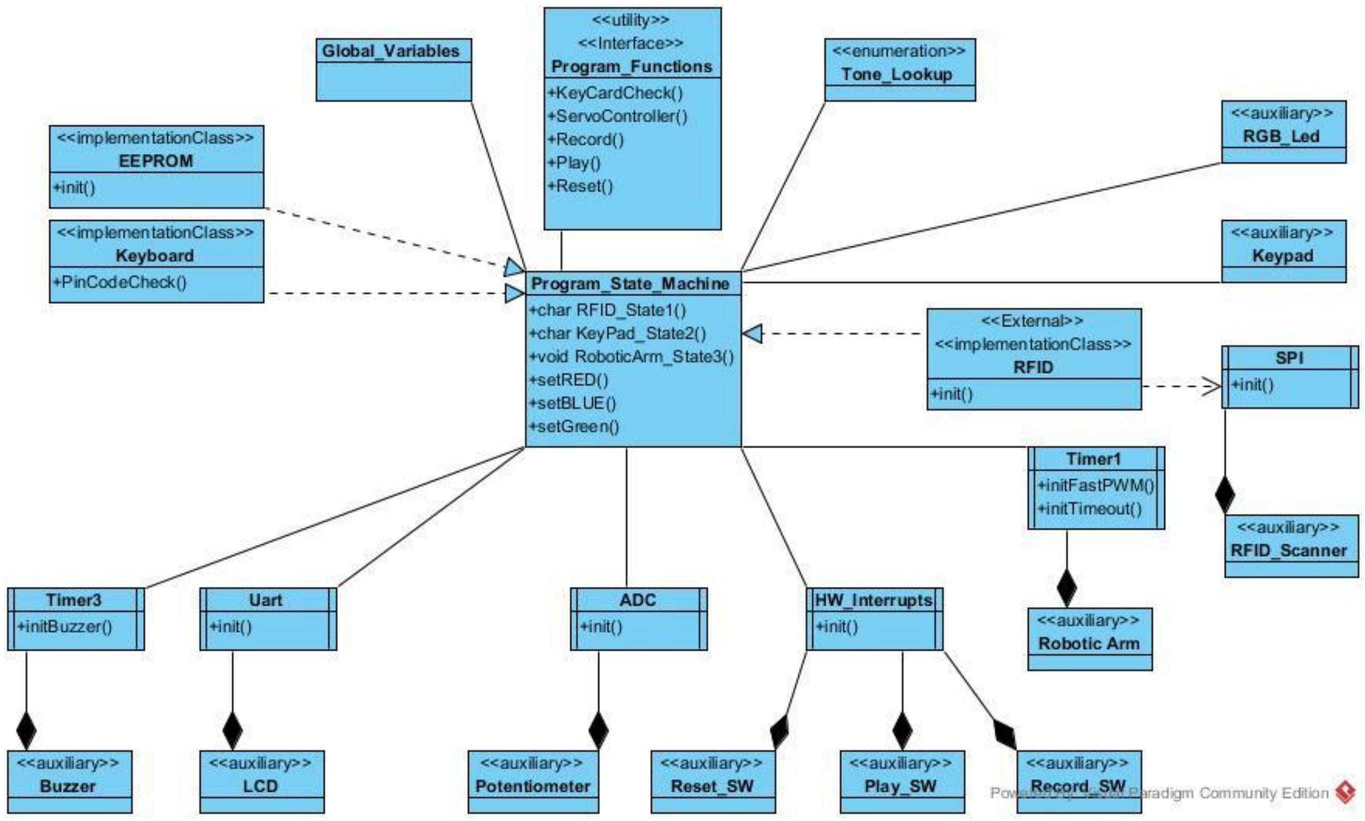


Figure 8 RC2-SLPR SW/HW Architecture

Software modules of the RC2-SLPR are developed from ground up with an exception of one module. The implementation class of the RFID module was modified from its original author who is referenced in this report in order to function with the particular ATmega 2560 processor, which is utilized by the RC2-SLPR. Additionally all the modules presented in Figure 8 can be found as separate .h files in the project folder.

Finally before proceeding to the test section of this report, it is worth to look at a UML based state machine diagram of the RC2-SLPR. Figure 9 show an excellent view of program execution from a state machine perspective which is also the way the system was intentionally designed. It is important to notice the special state #3 “ServoController”. This state can be classified as a submachine state which itself is a miniature state machine. In order to be able to record the robotic arm positions or perform a playback, the activation of these states is only realizable from the “ServoController” state (state #3).

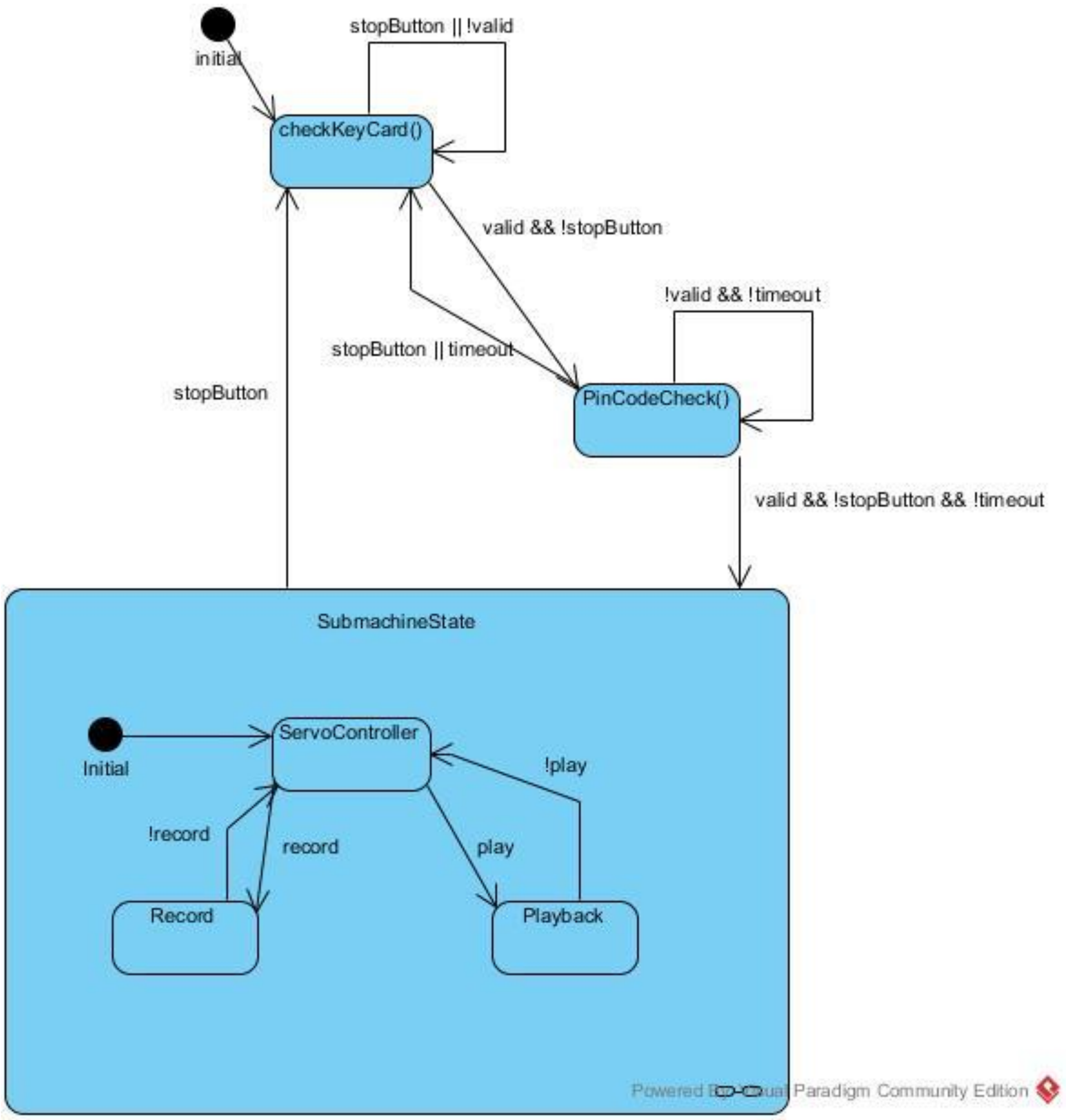


Figure 9 State Machine diagram of RC2-SLPR

3. Testing

The RC2-SLPR has been tested extensively throughout its development process. Its architecture which was shown in Figure 7,8 made the verification and validation process much simpler. Each module could individually be verified to be functioning properly and thereafter validated with accordance to the customer specifications. To assist in testing individual modules, the use of low level variable analysis was aided by the use of an AVR Dragonboard Debugger. There were certain sections of the system in which time was critical and therefore had to have an acceptable WCET time. This information was previously shown in Table 4 for the timing section of this report. The RC2-SLPR is considered robust as its three state architecture is simple and independent. Therefore it makes it convenient to handle one state at a time and fix the relevant modules to that state. Table 1 summarized the main functionality of the RC2-SLPR, additionally Table 3 provided an extensive configuration of the modules for correctness of operation and usability.

Finally the following Black Box tests were performed to validate the user requirements from Figure 1.

Test Case Description	Test Case Id	Test Description	Expected Results	Actual Results	Test Status Pass/Fail	Severity
Verify RFID	B001	Provide valid	System shall	Valid output	PASS	High

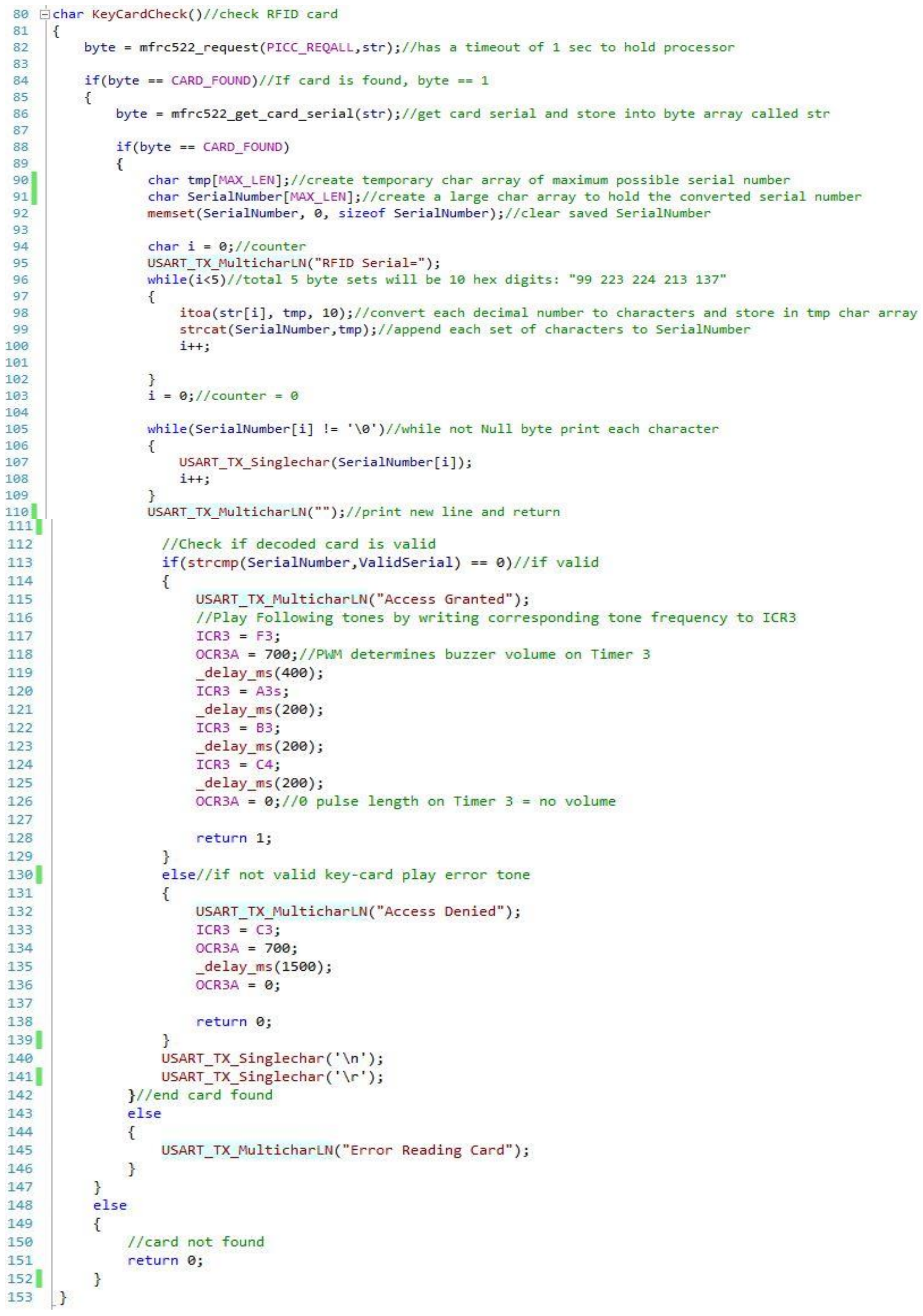
		RFID card	return true	generated		
		Provide Invalid RFID card	System shall return false	Invalid output generated	PASS	High
Verify PinID	B002	Enter valid pin code	System shall return true	Valid output displayed	PASS	High
		Enter invalid pin code	System shall return false	Invalid output displayed	PASS	High
Control Robotic Arm	B003	Move Pot to rotate robotic arm	Robotic arm move with no jitter	Robotic arm moved accurately	PASS	High
		Stress the Potentiometer	Robotic arm is responsive	Robotic arm moved instantly	PASS	High
Record Position	B004	Activate Record Mode	Positions shall be stored into EEPROM	Function stored ADC values to EEPROM	PASS	Medium
Playback	B005	Activate Playback Mode	Robot should move with playback	Robotic arm moved with dataset from read EEPROM	PASS	Medium
Reset	B006	Activate System Reset	System Resets	System jumped to its initial state	PASS	High

Table 5 Black Box test results

Table 5 show the results of a black box test for each user requirement. It is important to notice that some requirements have a higher severity degree if not working properly. In fact 2/6 use cases had the lowest severity level measured to medium. This was concluded from a functional perspective of a real world robotics system. The minimum requirement for a high security level access robotic arm to function properly was to first grant access to the user in a secure way, thereafter allow the user to operate the robot arm. Any additional features such as recording and playing the robotic arm would merely be seen as optional features. These features would not have to be bullet proof in order for the user to operate the robotic arm. Lastly perhaps the most important feature would be to allow the user to perform a system reset. From a safety perspective, the reset switch could be seen as the most critical part of the system and therefore be given the highest available rank on the severity scale.

4. Program code listings

The following code snippet illustrate how the RC2-SLPR decode and validate RFID cards. The comments provided should make the code self-explanatory.



```c
char KeyCardCheck()//check RFID card
{
    byte = mfrc522_request(PICC_REQALL,str);//has a timeout of 1 sec to hold processor

    if(byte == CARD_FOUND)//If card is found, byte == 1
    {
        byte = mfrc522_get_card_serial(str);//get card serial and store into byte array called str

        if(byte == CARD_FOUND)
        {
            char tmp[MAX_LEN];//create temporary char array of maximum possible serial number
            char SerialNumber[MAX_LEN];//create a large char array to hold the converted serial number
            memset(SerialNumber, 0, sizeof SerialNumber);//clear saved SerialNumber

            char i = 0;//counter
            USART_TX_MulticharLN("RFID Serial=");
            while(i<5)//total 5 byte sets will be 10 hex digits: "99 223 224 213 137"
            {
                itoa(str[i], tmp, 10);//convert each decimal number to characters and store in tmp char array
                strcat(SerialNumber,tmp);//append each set of characters to SerialNumber
                i++;

            }
            i = 0;//counter = 0

            while(SerialNumber[i] != '\0')//while not Null byte print each character
            {
                USART_TX_Singlechar(SerialNumber[i]);
                i++;
            }
            USART_TX_MulticharLN("");//print new line and return

            //Check if decoded card is valid
            if(strcmp(SerialNumber,ValidSerial) == 0)//if valid
            {
                USART_TX_MulticharLN("Access Granted");
                //Play Following tones by writing corresponding tone frequency to ICR3
                ICR3 = F3;
                OCR3A = 700;//PWM determines buzzer volume on Timer 3
                _delay_ms(400);
                ICR3 = A3s;
                _delay_ms(200);
                ICR3 = B3;
                _delay_ms(200);
                ICR3 = C4;
                _delay_ms(200);
                OCR3A = 0;//0 pulse length on Timer 3 = no volume

                return 1;
            }
            else//if not valid key-card play error tone
            {
                USART_TX_MulticharLN("Access Denied");
                ICR3 = C3;
                OCR3A = 700;
                _delay_ms(1500);
                OCR3A = 0;

                return 0;
            }
            USART_TX_Singlechar('\n');
            USART_TX_Singlechar('\r');
        }//end card found
        else
        {
            USART_TX_MulticharLN("Error Reading Card");
        }
    }
    else
    {
        //card not found
        return 0;
    }
}
```

Part B

5. Critical evaluation and conclusion

The completed RC2-SLPR system satisfied all the use cases in Figure 1. Apart from the validated requirements from Table 5, the system also make use of several extended solutions which were included in the final design. Table 6 highlight the extended solutions of the RC2-SLPR.

Extended Solutions		
Technique	**Present Yes/No**	**How/Where**
Use of multiple timer	Yes	Timer1 for PWM, Timer3 for Tone Frequency generation
Multiple interrupts, with appropriate prioritization	Yes	Refer to Table 4
Multiplexing in/out to the same port	Yes	Timer1 is multiplexed with different setup configurations for use with PWM/Timeout modes
Bitwise manipulation of I/O on ports	Yes	Keypad Module uses extensive bit manipulation to check signals for COL/ROW
Use the ADC/ACOMP	Yes	ADC converts potentiometer inputs to volts.
Use of more-complex peripherals such as an LCD/Keypad	Yes	Keypad, Buzzer, RFID etc. are used
Organizing program code into libraries to improve the organization and re-use of code	Yes	All modules from figure 8 are represented as separate .h files with an exception of the RFID which is referenced as an external static library
In-built diagnostics (ex LEDs to show program state)	Yes	The RGB module together with the UART represent the different states for the user

Table 6

As previously mentioned, the main architecture of the RC2-SLPR is formed around a state machine. This design concept made it easier to realize both the use cases and the extended solutions shown in Table 6. The final design of the RC2-SLPR is very pleasing. Top features include robustness and ease of maintenance. The existing software modules can easily be fixed/upgraded and replaced. The system was tested extensively during its development. Register contents and function were debugged and verified with the help of AVR Dragonboard. The user requirements were finally validated with black box testing which could be seen in Table 5. There are however certain aspects of the system which could have been improved in both testing and development. If the system was to become a production unit, it would have been essential to further white box testing of each module in order to ensure its robustness. Furthermore there exist a shortage of limits within the system. Even though the system provides a finite timeout of 10 seconds in state 2, additional limits to the system should be implemented. For example there should be a limit to the number of tries that the user can enter an incorrect pin-code even if the user is sent back to the RFID state after just 10 seconds of timeout. The system should accumulate the number of sequential incorrect tries until it reaches a set limit which could block that specific RFID access card for one hour or more. Additionally the referenced external RFID library module could be developed from ground up in order to further suppress unnecessary logic that might exist. Finally there should also be a timeout for state 3. A timer could start due to inactivity of the robotic arm controller and when expired reset the system.

I have had the great pleasure to experience the development of a prototype based on a real time system. It was incredible to see how the small sensors and other peripherals could be joined together to perform more complex tasks. With the vast power and flexibility that the AVR development suite offers, the limit of what could actually be implemented was only restricted to my imagination. Previously I have mostly worked with bootloaders which hide the details of register settings in a micro-controller. Now I feel that every resource on the micro-controller can be utilized efficiently in order to meet the requirements of a design. The overall project has made me even more confident as a programmer. Finally this experience has allowed me to gain more insight in efficiently utilizing micro-controller datasheets to find and set desired registers.

References

- Asif Shimon, MIFARE RC522 module library for Atmega48, https://github.com/asif-mahmud/MIFARE-RFID-with-AVR
- Atmel-ATmega640-1280-1281-2560-2561_datasheet
- Atmel Studio 6
- Fritzing, open-source circuit builder
- Real-Time Systems: Design principles for distributed embedded applications, Hermann Kopetz, 2011
- Systems Programming: 1st Edition Designing and Developing Distributed Applications, Richard Anthony, 2015
- Visual Paradigm 12.0 Community Edition