UPPSALA
UNIVERSITET

Sweet Shot Project Report
Microcontroller Programming Course, 2018
Embedded System, Uppsala University

Anoud Alshnakat
Asif Mohamed
Paul Sujeet
Vishnu Ullas

Contents

1.Introduction

Sweet Shot is an electronic system that is designed and implemented to shoot a piece of candy on the person entering a room. The project is based on wireless communication and operates using ATmega328p microcontroller unit. The idea is to create a fun system, yet applying most of the concepts that cover the syllabus.

2.Implementation

The Sweet Shot system consists of two modules: primary and secondary module. The primary module is set up at the top of the door frame to calculate the height of the person entering through the door using the ultrasound sensor. Based on the height of person, the system sends data to the secondary module.
The secondary module is placed away from the door to a certain distance. A cannon is connected to this module which consists of a stepper motor, solenoid and a servo motor. The Stepper motor helps to change the angle of the cannon while the servo motor and solenoid are utilized as the trigger of the cannon. The servo motor also reloads the cannon with candy. Using this cannon, we are able to shoot candy to the torso of the person. Fun stuff!

3.Design

The Sweet Shot circuit was built on two main sections, first one is suitably connected to the ATmega328p microcontroller and the Ultrasonic HC-SR04 and nRF24L01+ as a transmitter, which called the primary system, second one is connected to the stepper motor, solenoid, servo motor and nRF24L01+ as a receiver, which called secondary system.

In the primary system, the connections are wired as seen below in Figure 1, the MCU is getting powered up directly from an external battery, which is the main V_{cc} of the primary module. The nRF24L01+ module demands lesser voltage than the main V_{cc} i.e. only 3.3 V, so a voltage regulator LM317 is added to the circuit. The nRF24L01+ is also connected to the MCU by a Serial Peripheral Interface (SPI) bus and it will function as a transmitter module that the secondary module will receive. The distance sensor i.e. Ultrasonic HC-SR04 is connected to the MCU by General Purpose Input/Output (GPIO), with the pins PC4 and PC5.

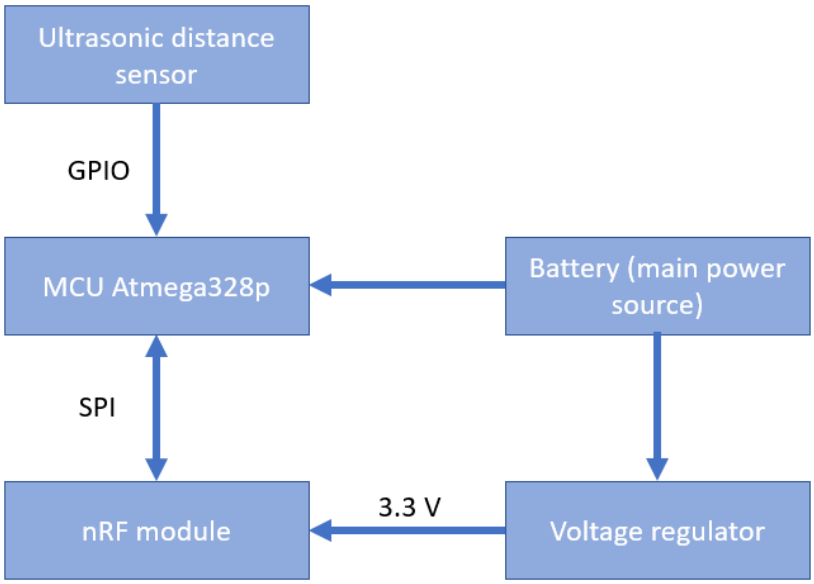

Figure 1

Moving to the secondary system, the connections are wired as seen below in Figure 2, it has four similar components in the circuit that exists in the primary system, namely the MCU, nRF24L01+ module, power source and voltage regulator. However, the mechanism of receiving the signal from the primary system as well as shooting the candy is added here. Looking at the Figure 2, the nRF24L01+ module will function as the receiver between communicating systems. The servo motor is connected to one of the GPIO pins of the MCU. Even though, the stepper motor and the solenoid are controlled by the MCU's GPIO pins, the MCU cannot supply sufficient current to control the poles of the stepper motor or the solenoid. Thus, we use a high-voltage, high-current Darlington transistor array to control the supply to these modules.

The cannon consists of the stepper motor, servo motor and solenoid. The base of the stepper motor is placed over a rigid body and the cannon is fixed on the motor itself. The cannon is just a long tube with the solenoid placed on one end, and the servo motor is placed on top of the tube to push the solenoid's piston. And the circuit is fixed at the base of this rigid structure.

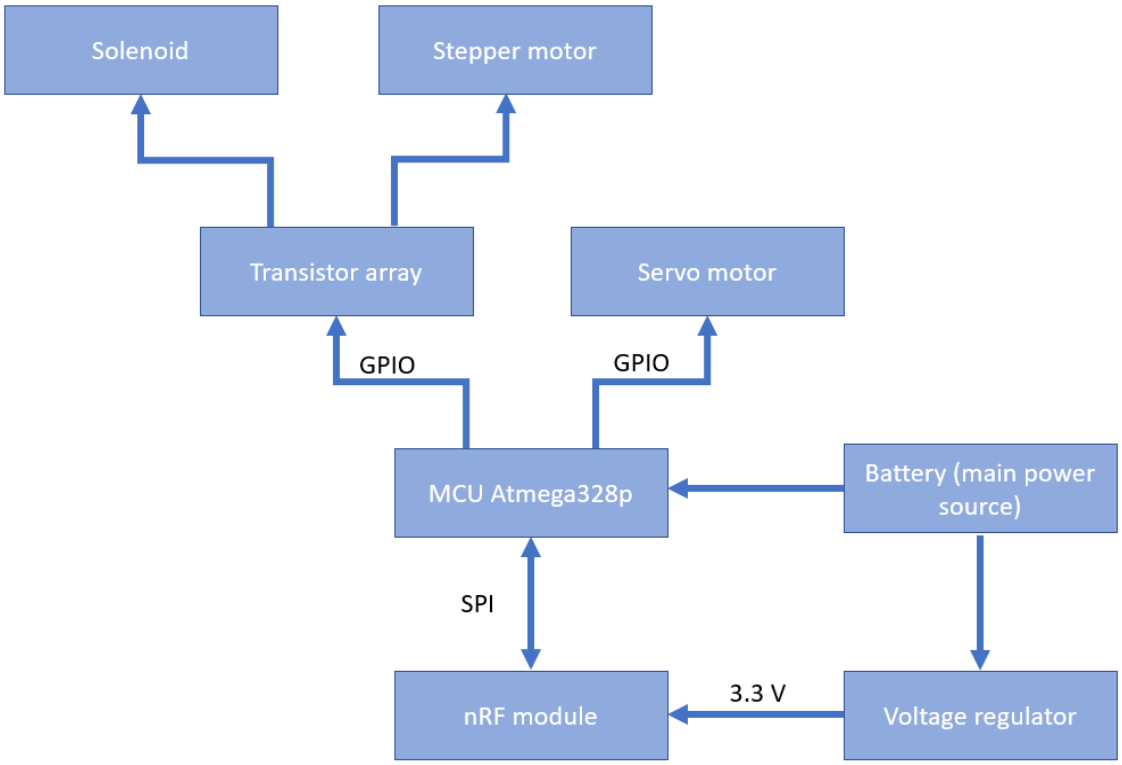

Figure 2

Project Hardware

The two circuits of the connected systems of the primary system and secondary system are shown in Figure 3 and Figure 4 respectively, it indicates the convenient wiring between the hardware parts.

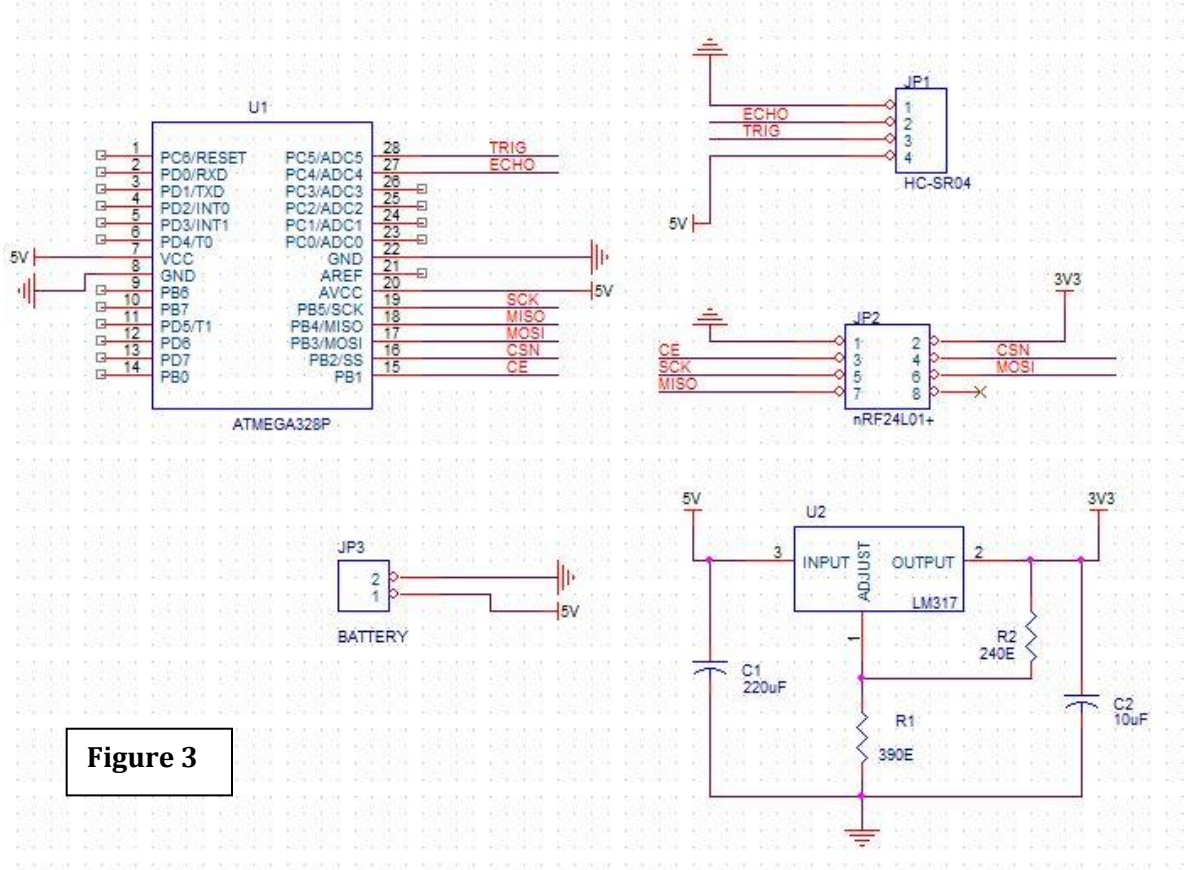

Figure 3

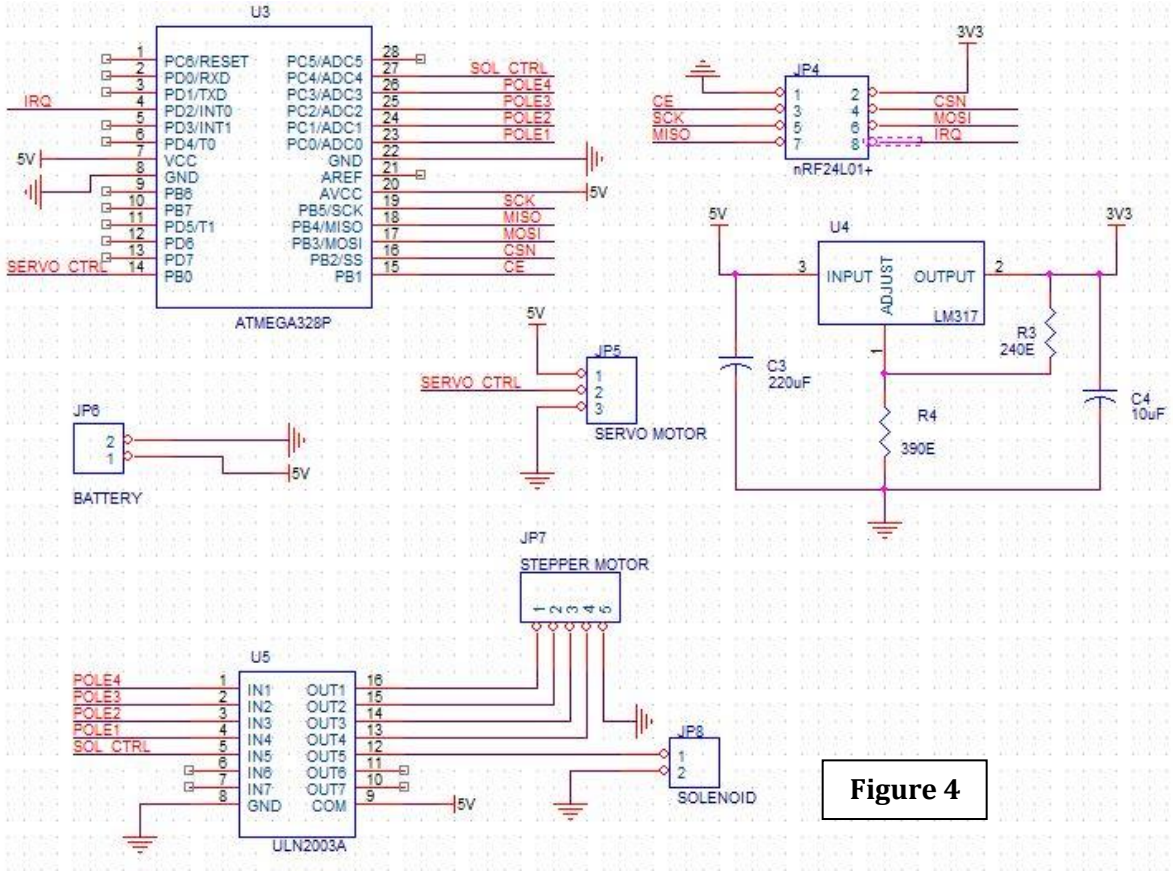

Figure 4

The exact used hardware parts are listed below, in addition to the essential features that is useful in the Sweet Shot project.

The MCU: ATmega328p

The ATmega328p is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. [1]

- Advanced RISC Architecture
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Six PWM Channels
- Six PWM Channels
- 6-channel 10-bit ADC in PDIP Package
- Two Master/Slave SPI Serial Interface
- Interrupt and Wake-up on Pin Change
- 23 Programmable I/O Lines
- Operating Voltage 1.8 - 5.5V

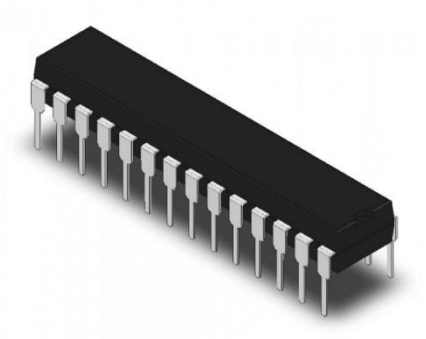

Figure 5

nRF24L01+

The Nordic nRF24L01+ integrates a complete 2.4GHz RF transceiver, RF synthesizer, and baseband logic including the Enhanced ShockBurst™ hardware protocol accelerator supporting a high-speed SPI interface for the application controller. [2]

The Nordic nRF24L01+ is a highly integrated, ultra-low power (ULP) 2Mbps RF transceiver IC for the 2.4GHz ISM (Industrial, Scientific and Medical) band. With peak RX/TX currents lower than 14mA, a sub µA power down mode, advanced power management, and a 1.9 to 3.6V supply range, the nRF24L01+ provides a true ULP solution enabling months to years of battery life from coin cell or AA/AAA batteries.

Figure 6

Ultrasonic HC-SR04

Ultrasonic distance sensor with transmitter and receiver mounted on a single board. The sensor will transmit 8 short pulses, calculate the time for the sound to bounce back and then sets the output high with a pulse length corresponding to the distance. [3]

- Supply voltage: 5VDC
- Current consumption: 2mA
- Sensing angle: 15°
- Sensing range: 2 - 400cm
- Resolution: 3mm

Figure 7

Stepper motor 28BYJ-48

The 28BYJ-48 is a small stepper motor suitable for a large range of applications that operates at 5V. It contains 4 phases.[4]

Figure 8

Servo motor MS-R-1.3-9

The servo motor operates at 4.8V~6V with 360º rotation.[5]

Figure 9

Transistor Array ULN2003A

The ULN2003A device is a high-voltage, high-current Darlington transistor arrays. Each consists of seven NPN Darlington pairs that feature high-voltage outputs with common-cathode clamp diodes for switching inductive loads.[6]

Main specifications:

- 500 mA rated collector current (single output)
- 50 V output (there is a version that supports 100 V output)
- Includes output fly-back diodes
- Inputs compatible with TTL and 5-V CMOS logic

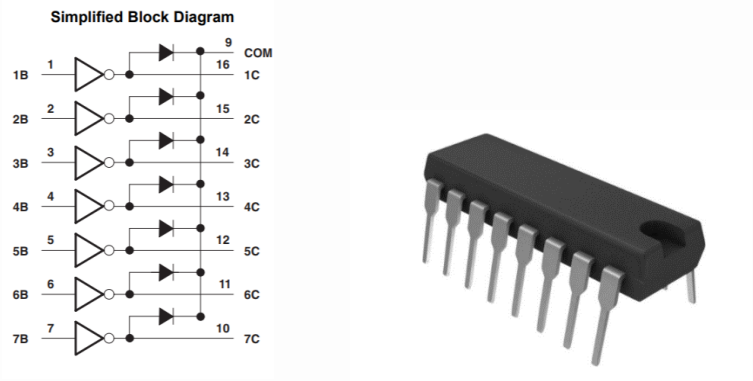

Figure 10

Small Push-Pull Solenoid

Solenoid is a device to induce linear motion for pushing, pulling or controlling switches and levers. This solenoid is designed to work directly with 5V which makes it a great match for embedded projects.[7]

Figure 11

Voltage regulator LM317

The LM317 is an adjustable 3−terminal positive voltage regulator capable of supplying excess of 1.5 A over an output voltage range of 1.2 V to 37 V. This voltage regulator is exceptionally easy to use and requires only two external resistors to set the output voltage. Further, it employs internal current limiting, thermal shutdown and safe area compensation, making it essentially blow−out proof.[8]

- Output Adjustable between 1.2 V and 37 V
- Internal Thermal Overload Protection
- Internal Short Circuit Current Limiting Constant with Temperature
- Output Transistor Safe−Area Compensation
- Floating Operation for High Voltage Applications

Figure 12

Project Software

The Sweet Shot project is divided as previously mentioned into two main systems, and each has a separate code.

Primary module

- **Headers**
  - As the nRF24L01+ module uses SPI interface to communicate with the MCU, we utilize a simple library in order to initialize and implement the communication between two different nRF24L01+ modules as *nrf24l01.c*.
  - We have created a simple library to initialise and control the HC-SR04 as *systemprim.c*.

- **Main Function**
  - Initialize the HC-SR04 to the respective pins and enable it as an interrupt.
  - Initializes the nRF24L01+ to Tx mode to transmit data to the destination address
  - Initialize a timer to calculate the time taken by the ultrasound to receive the trigger.
  - Enable the global interrupts as well.
  - Main while loop begins here and loops every 60ms. Every 60ms, the ultrasound is made to trigger a signal and if distance calculated at the less than 200 (standard height of door frame), it enables the nRF24L01+ module to send a message based on this height and waits for 1s.
  - Else the loop continues.

- **Interrupt service routine (PCINT1)**
  - A flag is used to make sure to reset the timer every 60 ms and re-enable the trigger in the ultrasound if no echo is received.
  - If there is a response, the timer value is saved and the distance is calculated. If the distance is more than 200 cm i.e. the max height of the door, it is doesn't enable the nRF24L01+ to send any data. However, if it is lesser, it will enable the nRF24L01+ to send respective strings based on the levels of height.

Secondary Module

- **Headers**
  - As the nRF24L01+ module uses SPI interface to communicate with the MCU, we utilize a simple library in order to initialize and implement the communication between two different nRF24L01+ modules as *nrf24l01.c*.
  - A library is created to initialize and control the stepper motor, servo motor and the solenoid as *systemsecon.c*.

- **Main Function**
  - Enable the interrupts.
  - Calls the functions to initialize the stepper motor, servo motor, and solenoid to the respective pins.
  - Initializes the nRF24L01+ to Rx mode to receive data from the source address.
  - Main while loop begins and checks for any data being received to enable the flag in the interrupt.
  - Based on the message received, a function is called which changes the angle of the stepper motor from the reference point and the solenoid is switched on. The servo motor, right away, moves 0 to 180 degrees and back to 0. The solenoid is then released to shoot the piston and the stepper motor returns to its reference point.

- **Process_system**
    - There are 4 different types of messages that the system can recognize.

Message	Height	Angle of stepper motor
AL	**151-160**	**45**
BL	**161-170**	**55**
CL	**171-180**	**65**
DL	**181-190**	**75**

4. References

1. ATmega328P - 8-bit AVR Microcontrollers. 2018. ATmega328P - 8-bit AVR Microcontrollers. [ONLINE] Available at: http://www.microchip.com/wwwproducts/en/ATmega328p.

2. eZ Systems. 2018. nRF24L01+ / 2.4GHz RF / Products / Home - Ultra Low Power Wireless Solutions from NORDIC SEMICONDUCTOR. [ONLINE] Available at: https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P.

3. 2018. HC-SR04 Ultrasonic Sensor Distance Measuring Module . [ONLINE] Available at: https://robokits.co.in/sensors/ultrasonic-sensor/hc-sr04-ultrasonic-sensor-distance-measuring-module

4. 28BYJ-48 Stepper Motor with ULN2003 driver and Arduino Uno. [ONLINE] Available at: http://42bots.com/tutorials/28byj-48-stepper-motor-with-uln2003-driver-and-arduino-uno/

5. Olimex. 2018. MS-R-1.3-9. [ONLINE] Available at: https://www.olimex.com/Products/RobotParts/ServoMotors/MS-R-1.3-9/

6. ULN2003A. [ONLINE] Available at: http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=slrs027&fileType=pdf

7. Solenoid - 5v (small). [ONLINE] Available at: https://www.sparkfun.com/products/11015

8. LM317 3/4 Pin 1.5A Adjustable Positive Voltage Regulator. [ONLINE] Available at: http://www.ti.com/product/LM317

5. Appendix

Nrf24l01.h

```
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>

#ifndef _NRF24L01_H
#define _NRF24L01_H


typedef struct {
    int pipe_number;
    uint8_t data[32];
    uint8_t length;
} nRF24L01Message;

typedef struct {
    volatile uint8_t *port;
    uint8_t pin;
} gpio_pin;

typedef struct {
    gpio_pin ss; // slave select
    gpio_pin ce; // chip enabled
    gpio_pin sck; // serial clock
    gpio_pin mosi; // master out slave in
    gpio_pin miso; // master in slave out
    uint8_t status;
} nRF24L01;


nRF24L01 *nRF24L01_init(void);
void nRF24L01_begin(nRF24L01 *rf);
uint8_t nRF24L01_send_command(nRF24L01 *rf, uint8_t command, void *data,
    size_t length);
uint8_t nRF24L01_write_register(nRF24L01 *rf, uint8_t reg_address, void *data,
    size_t length);
uint8_t nRF24L01_read_register(nRF24L01 *rf, uint8_t regAddress, void *data,
    size_t length);
uint8_t nRF24L01_no_op(nRF24L01 *rf);
uint8_t nRF24L01_update_status(nRF24L01 *rf);
uint8_t nRF24L01_get_status(nRF24L01 *rf);
void nRF24L01_listen(nRF24L01 *rf, int pipe, uint8_t *address);
bool nRF24L01_data_received(nRF24L01 *rf);
bool nRF24L01_read_received_data(nRF24L01 *rf, nRF24L01Message *message);
int nRF24L01_pipe_number_received(nRF24L01 *rf);
```

```
void nRF24L01_transmit(nRF24L01 *rf, void *address, nRF24L01Message *msg);
int nRF24L01_transmit_success(nRF24L01 *rf);
void nRF24L01_flush_transmit_message(nRF24L01 *rf);
void nRF24L01_retry_transmit(nRF24L01 *rf);
void nRF24L01_clear_interrupts(nRF24L01 *rf);
void nRF24L01_clear_transmit_interrupts(nRF24L01 *rf);
void nRF24L01_clear_receive_interrupt(nRF24L01 *rf);

#endif
```

Nrf24l01.c

```
#include <stdlib.h>
#include <stdbool.h>
#include <avr/io.h>
#include <util/delay.h>
#include <string.h>
#include "nrf24l01.h"
#include "nrf24l01-mnemonics.h"


static void copy_address(uint8_t *source, uint8_t *destination);
inline static void set_as_output(gpio_pin pin);
inline static void set_high(gpio_pin pin);
inline static void set_low(gpio_pin pin);
static void spi_init(nRF24L01 *rf);
static uint8_t spi_transfer(uint8_t data);


nRF24L01 *nRF24L01_init(void) {
    nRF24L01 *rf = malloc(sizeof(nRF24L01));
    memset(rf, 0, sizeof(nRF24L01));
    return rf;
}

void nRF24L01_begin(nRF24L01 *rf) {
    set_as_output(rf->ss);
    set_as_output(rf->ce);

    set_high(rf->ss);
    set_low(rf->ce);

    spi_init(rf);

    nRF24L01_send_command(rf, FLUSH_RX, NULL, 0);
    nRF24L01_send_command(rf, FLUSH_TX, NULL, 0);
    nRF24L01_clear_interrupts(rf);

    uint8_t data;
    data = _BV(EN_CRC) | _BV(CRCO) | _BV(PWR_UP) | _BV(PRIM_RX);
    nRF24L01_write_register(rf, CONFIG, &data, 1);

    // enable Auto Acknowlegde on all pipes
    data = _BV(ENAA_P0) | _BV(ENAA_P1) | _BV(ENAA_P2)
         | _BV(ENAA_P3) | _BV(ENAA_P4) | _BV(ENAA_P5);
```

```
    nRF24L01_write_register(rf, EN_AA, &data, 1);

    // enable Dynamic Payload on al pipes
    data = _BV(DPL_P0) | _BV(DPL_P1) | _BV(DPL_P2)
         | _BV(DPL_P3) | _BV(DPL_P4) | _BV(DPL_P5);
    nRF24L01_write_register(rf, DYNPD, &data, 1);

    // enable Dynamic Payload (global)
    data = _BV(EN_DPL);
    nRF24L01_write_register(rf, FEATURE, &data, 1);

    // disable all rx addresses
    data = 0;
    nRF24L01_write_register(rf, EN_RXADDR, &data, 1);
}

uint8_t nRF24L01_send_command(nRF24L01 *rf, uint8_t command, void *data,
    size_t length) {
    set_low(rf->ss);

    rf->status = spi_transfer(command);
    for (unsigned int i = 0; i < length; i++)
        ((uint8_t*)data)[i] = spi_transfer(((uint8_t*)data)[i]);

    set_high(rf->ss);

    return rf->status;
}

uint8_t nRF24L01_write_register(nRF24L01 *rf, uint8_t reg_address, void *data,
    size_t length) {
    return nRF24L01_send_command(rf, W_REGISTER | reg_address, data, length);
}

uint8_t nRF24L01_read_register(nRF24L01 *rf, uint8_t reg_address, void *data,
    size_t length) {
    return nRF24L01_send_command(rf, R_REGISTER | reg_address, data, length);
}

uint8_t nRF24L01_no_op(nRF24L01 *rf) {
    return nRF24L01_send_command(rf, NOP, NULL, 0);
}

uint8_t nRF24L01_update_status(nRF24L01 *rf) {
    return nRF24L01_no_op(rf);
}

uint8_t nRF24L01_get_status(nRF24L01 *rf) {
    return rf->status;
}

bool nRF24L01_data_received(nRF24L01 *rf) {
    set_low(rf->ce);
    nRF24L01_update_status(rf);
    return nRF24L01_pipe_number_received(rf) >= 0;
```

```
}

void nRF24L01_listen(nRF24L01 *rf, int pipe, uint8_t *address) {
    uint8_t addr[5];
    copy_address(address, addr);

    nRF24L01_write_register(rf, RX_ADDR_P0 + pipe, addr, 5);

    uint8_t current_pipes;
    nRF24L01_read_register(rf, EN_RXADDR, &current_pipes, 1);
    current_pipes |= _BV(pipe);
    nRF24L01_write_register(rf, EN_RXADDR, &current_pipes, 1);

    set_high(rf->ce);
}

bool nRF24L01_read_received_data(nRF24L01 *rf, nRF24L01Message *message) {
    message->pipe_number = nRF24L01_pipe_number_received(rf);
    nRF24L01_clear_receive_interrupt(rf);
    if (message->pipe_number < 0) {
        message->length = 0;
        return false;
    }

    nRF24L01_read_register(rf, R_RX_PL_WID, &message->length, 1);

    if (message->length > 0) {
        nRF24L01_send_command(rf, R_RX_PAYLOAD, &message->data,
            message->length);
    }

    return true;
}

int nRF24L01_pipe_number_received(nRF24L01 *rf) {
    int pipe_number = (rf->status & RX_P_NO_MASK) >> 1;
    return pipe_number <= 5 ? pipe_number : -1;
}

void nRF24L01_transmit(nRF24L01 *rf, void *address, nRF24L01Message *msg) {
    nRF24L01_clear_transmit_interrupts(rf);
    uint8_t addr[5];
    copy_address((uint8_t *)address, addr);
    nRF24L01_write_register(rf, TX_ADDR, addr, 5);
    copy_address((uint8_t *)address, addr);
    nRF24L01_write_register(rf, RX_ADDR_P0, addr, 5);
    nRF24L01_send_command(rf, W_TX_PAYLOAD, &msg->data, msg->length);
    uint8_t config;
    nRF24L01_read_register(rf, CONFIG, &config, 1);
    config &= ~_BV(PRIM_RX);
    nRF24L01_write_register(rf, CONFIG, &config, 1);
    set_high(rf->ce);
}

int nRF24L01_transmit_success(nRF24L01 *rf) {
```

```
    set_low(rf->ce);
    nRF24L01_update_status(rf);
    int success;
    if (rf->status & _BV(TX_DS)) success = 0;
    else if (rf->status & _BV(MAX_RT)) success = -1;
    else success = -2;
    nRF24L01_clear_transmit_interrupts(rf);
    uint8_t config;
    nRF24L01_read_register(rf, CONFIG, &config, 1);
    config |= _BV(PRIM_RX);
    nRF24L01_write_register(rf, CONFIG, &config, 1);
    return success;
}

void nRF24L01_flush_transmit_message(nRF24L01 *rf) {
    nRF24L01_send_command(rf, FLUSH_TX, NULL, 0);
}

void nRF24L01_retry_transmit(nRF24L01 *rf) {
    // XXX not sure it works this way, never tested
    uint8_t config;
    nRF24L01_read_register(rf, CONFIG, &config, 1);
    config &= ~_BV(PRIM_RX);
    nRF24L01_write_register(rf, CONFIG, &config, 1);
    set_high(rf->ce);
}

void nRF24L01_clear_interrupts(nRF24L01 *rf) {
    uint8_t data = _BV(RX_DR) | _BV(TX_DS) | _BV(MAX_RT);
    nRF24L01_write_register(rf, STATUS, &data, 1);
}

void nRF24L01_clear_transmit_interrupts(nRF24L01 *rf) {
    uint8_t data = _BV(TX_DS) | _BV(MAX_RT);
    nRF24L01_write_register(rf, STATUS, &data, 1);
}

void nRF24L01_clear_receive_interrupt(nRF24L01 *rf) {
    uint8_t data = _BV(RX_DR) | rf->status;
    nRF24L01_write_register(rf, STATUS, &data, 1);
}

static void copy_address(uint8_t *source, uint8_t *destination) {
    for (int i = 0; i < 5; i++)
        destination[i] = source[i];
}

inline static void set_as_output(gpio_pin pin) {
    volatile uint8_t *ddr = pin.port - 1;
    *ddr |= _BV(pin.pin);
}

inline static void set_as_input(gpio_pin pin) {
    volatile uint8_t *ddr = pin.port - 1;
    *ddr &= ~_BV(pin.pin);
```

```
}

inline static void set_high(gpio_pin pin) {
    *pin.port |= _BV(pin.pin);
}

inline static void set_low(gpio_pin pin) {
    *pin.port &= ~_BV(pin.pin);
}

static void spi_init(nRF24L01 *rf) {
    // set as master
    SPCR |= _BV(MSTR);
    // enable SPI
    SPCR |= _BV(SPE);
    // MISO pin automatically overrides to input
    set_as_output(rf->sck);
    set_as_output(rf->mosi);
    set_as_input(rf->miso);
    // SPI mode 0: Clock Polarity CPOL = 0, Clock Phase CPHA = 0
    SPCR &= ~_BV(CPOL);
    SPCR &= ~_BV(CPHA);
    // Clock 2X speed
    SPCR &= ~_BV(SPR0);
    SPCR &= ~_BV(SPR1);
    SPSR |= _BV(SPI2X);
    // most significant first (MSB)
    SPCR &= ~_BV(DORD);
}

static uint8_t spi_transfer(uint8_t data) {
    SPDR = data;
    while (!(SPSR & _BV(SPIF)));
    return SPDR;
}
```

systemPrim.h

```
#ifndef _SYSTEMPRIM_H_
#define _SYSTEMPRIM_H_

#define TRIGGER_PERIOD 10
void sensorInit();
void triggerSensor();

#endif /* SYSTEMPRIM_H_ */
```

systemPrim.c

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <string.h>
```

```
#include "systemPrim.h"

void sensorInit() {
	DDRC = 0xFF;                                        // Port C second nibble
all output.
	DDRC &= ~(1<<DDC5);                                 // Set Pin C5 as input to
read Echo
	PORTC |= (1<<PORTC5);                               // Enable pull up on C5
	PORTC &= ~(1<<PC4);                                 // Init C4 as low
(trigger)

	PRR &= ~(1<<PRTIM1);                                // To activate timer1
module
	TCNT1 = 0;                                                // Initial timer
value
	TCCR1B |= (1<<CS10);                                // Timer without
prescaller. Since default clock for atmega328p is 1Mhz period is 1uS
	TCCR1B |= (1<<ICES1);                               // First capture on rising
edge

	PCICR = (1<<PCIE1);                                 // Enable PCINT[14:8] we
use pin C5 which is PCINT13
	PCMSK1 = (1<<PCINT13);                              // Enable C5 interrupt
}

void triggerSensor(){

	PORTC |= (1<<PC4);                                  // Set trigger high
	_delay_us(TRIGGER_PERIOD);                    // for 10uS
	PORTC &= ~(1<<PC4);                                 // to trigger the
ultrasonic module
}
```

systemsecon.h

```
/*
 * systemsecon.h
 *
 * Created: 29-01-2018 19:32:50
 *  Author: Asif Mohd
 */


#ifndef SYSTEMSECON_H_
#define SYSTEMSECON_H_
#define STEPPERMOTOR_DELAY  3
#define STEPPERMOTOR_PORT PORTC
#define STEPPERMOTOR_DDR  DDRC


//Initialization of Steper motor
void stepperInit();
//Initialization of Servo motor
void servoInit();
//Initialization of the Solenoid and Button for reference point
```

```
void solenoidInit();
//Functions to move the Stepper
int stepperMotorForward(int count);
void stepperMotorBackward(int count,int state);
//Functions to move the servo motor
void servoControl(int event);



#endif /* SYSTEMSECON_H_ */
```

systemsecon.c

```
#include <stdbool.h>
#include <avr/io.h>
#include <util/delay.h>
#include "systemsecon.h"


uint8_t stepArray[]={0b1000,0b1100,0b0100,0b0110,0b0010,0b0011,0b0001,0b1001};
int temp,step;

void stepperInit(){
	STEPPERMOTOR_DDR    = 0x0f;
	STEPPERMOTOR_PORT   = 0x00;
}

void servoInit(){
	DDRB |= (1 << PB0);
	PORTB ^= (0 << PB0);
}

void solenoidInit(){
	DDRC |= (1 << PC4);
	DDRC &= ~(1 << PC5);
	PORTC ^= (0 << PC4);
}

int stepperMotorForward(int count){
	temp=0;
	step=0;
	while(1){
		STEPPERMOTOR_PORT = stepArray[step];
		temp++;
		step++;
		if (step==8 && count != temp)
		step=0;
		else if (count == temp)
		return step;
		_delay_ms(STEPPERMOTOR_DELAY);
	}
}

void stepperMotorBackward(int count,int state){
```

```
	temp = 0;
	step=state-2;
	while(1){
		if (step < 0 && count != temp)
		step=7;
		STEPPERMOTOR_PORT = stepArray[step];
		temp++;
		step--;
		if ((count == temp)||((PINC & (1<<PC5)) == 1))
		break;
		_delay_ms(STEPPERMOTOR_DELAY);
	}
}

void servoControl(int event){
	switch(event){
		case 0:			//Move servo motor to 0 degrees
		for (int i=0;i<50;i++)
		{
			PORTC ^= (1 << PC5);
			_delay_us(1000);
			PORTC ^= (0 << PC5);
			_delay_ms(20);
		}
		break;

		case 1:			//Move servo motor to 135 degrees
		for (int i=0;i<50;i++)
		{
			PORTC ^= (1 << PC5);
			_delay_us(2500);
			PORTC ^= (0 << PC5);
			_delay_ms(20);
		}
		break;
	}
}
```

Main.c in Primary System

```
/*
 * Sweet_shot_prim.c
 *
 * Created: 29-01-2018 16:44:24
 * Author : Asif Mohd
 */

 /*
ATmega328P Pinout

                    PC6|1   28|PC5            TRIG
                    PD0|2   27|PC4            ECHO
                    PD1|3   26|PC3
                    PD2|4   25|PC2
                    PD3|5   24|PC1
```

```
                    PD4|6   23|PC0
                    Vcc|7   22|Gnd
                    Gnd|8   21|Aref
                    PB6|9   20|AVcc
                    PB7|10  19|PB5            SCK
                    PD5|11  18|PB4            MISO
                    PD6|12  17|PB3            MOSI
                    PD7|13  16|PB2      CSN
                    PB0|14  15|PB1            CE
*/

//Neccesary header Files
#define F_CPU 1000000UL
#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <string.h>
#include "systemPrim.h"
#include "nrf24l01.h"

//Global variables and functions
volatile bool rf_interrupt = false;
volatile bool send_message = false;
volatile uint8_t height;
inline void tx_message(uint8_t distance,nRF24L01 *rf);
uint8_t to_address[5] = { 0x01, 0x01, 0x01, 0x01, 0x01 };

//Setup for nRF24L01+ to Atmega328p connections
nRF24L01 *setup_rf(void) {
	nRF24L01 *rf = nRF24L01_init();
	rf->ss.port = &PORTB;
	rf->ss.pin = PB2;
	rf->ce.port = &PORTB;
	rf->ce.pin = PB1;
	rf->sck.port = &PORTB;
	rf->sck.pin = PB5;
	rf->mosi.port = &PORTB;
	rf->mosi.pin = PB3;
	rf->miso.port = &PORTB;
	rf->miso.pin = PB4;
	nRF24L01_begin(rf);
	return rf;
}

//Main Loop
int main(void) {
	//Ultrasound sensor is initiated
	sensorInit();
	nRF24L01 *rf = setup_rf();
	// Enable Global Interrupts
	sei();
	//Loop begins
	while(1){
```

```
            _delay_ms(60);                          // To allow sufficient time
between triggers of the sensor (60ms min)
            triggerSensor();                //Trigger the sensor
            if (rf_interrupt) {             //Check for Interrupt INT0 enabling the
flag for sending message
                  rf_interrupt = false;
                  int success = nRF24L01_transmit_success(rf);
                  if (success != 0)
                  nRF24L01_flush_transmit_message(rf);
            }
            if (send_message) {             //Sends the respective height detected
by the sensor through nRF24L01+
                  send_message = false;
                  tx_message(height,rf);
            }
      }
}


ISR(PCINT1_vect) {
      if (bit_is_set(PINC,PC5)) {                                   // Checks if echo
is high
            TCNT1 = 0;                                                    // Reset
Timer
            PORTC |= (1<<PC3);
      }
      else {
            uint16_t numuS = TCNT1;                                 // Save Timer value
            uint8_t oldSREG = SREG;
            cli();                                                        // Disable
Global interrupts
            height = 200 - numuS/58;                                      //Calculate
the height
            if(height > 191 && height < 151){
      //Enable flag only if height is in the restricted limit for application

            rf_interrupt = true;
            send_message = true;
            }
            SREG = oldSREG;                                               // Enable
interrupts
      }
}

//Function to send respective message based on the height
void tx_message(uint8_t distance,nRF24L01 *rf){
      nRF24L01Message msg;
      if (distance < 160)
      {
            memcpy(msg.data,"AL" , 3);
      }
      else if (distance < 170)
      {
            memcpy(msg.data,"BL" , 3);
      }
```

```
	else if (distance < 180)
	{
		memcpy(msg.data,"CL" , 3);
	}
	else if (distance < 190)
	{
		memcpy(msg.data,"DL" , 3);
	}
	msg.length = strlen((char *)msg.data) + 1;
	nRF24L01_transmit(rf, to_address, &msg);
	_delay_ms(1000);
}
```

Main.c in Secondary System

```
/*
 * Sweet_Shot_secon.c
 *
 * Created: 29-01-2018 19:31:18
 * Author : Asif Mohd
 */

/*
ATmega328P Pinout

/Reset		PC6|1   28|PC5
			PD0|2   27|PC4		SOLENOID
			PD1|3   26|PC3		POLE4
	IRQ		PD2|4   25|PC2		POLE3
			PD3|5   24|PC1		POLE2
			PD4|6   23|PC0		POLE1
			Vcc|7   22|Gnd
			Gnd|8   21|Aref
			PB6|9   20|AVcc
			PB7|10  19|PB5		SCK
			PD5|11  18|PB4		MISO
			PD6|12  17|PB3		MOSI
			PD7|13  16|PB2		CSN
	SERVO	PB0|14  15|PB1		CE
*/

//Header files
#define F_CPU 1000000UL
#include <inttypes.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <string.h>
#include <util/delay.h>
#include "nrf24l01.h"
#include "nrf24l01-mnemonics.h"
#include "systemsecon.h"

// Steps of stepper motor for various Levels
#define LEVEL1 500
```

```
#define LEVEL2 1000
#define LEVEL3 1500
#define LEVEL4 2000

// Setup for nRF24L01 & interrupt
nRF24L01 *setup_rf(void);
volatile bool rf_interrupt = false;

// Functions for system
inline void process_message(char *message);
inline void process_system(int step);

// Main Loop
int main(void) {

	// Setup for Stepper motor and Electronic Lock
	stepperInit();
	servoInit();
	solenoidInit();
	// Enable GLobal interrupts
	sei();

	// Initialization of nRF24L01
	uint8_t address[5] = { 0x01, 0x01, 0x01, 0x01, 0x01 };
	nRF24L01 *rf = setup_rf();
	nRF24L01_listen(rf, 0, address);
	uint8_t addr[5];
	nRF24L01_read_register(rf, CONFIG, addr, 1);

	// Loop to Listen to Data
	while (true) {
				PORTD ^= (0 << PD0);
		// To check if interupt is enabled from nRF24L01
		if (rf_interrupt) {
			rf_interrupt = false;
			//------------------------------------------------------------
				PORTD ^= (1 << PD0);

			// Read the data from Source address
			while (nRF24L01_data_received(rf)) {
				nRF24L01Message msg;
				nRF24L01_read_received_data(rf, &msg);
				// Process the data received
				process_message((char *)msg.data);

			}
			// Continue to listen for data
			nRF24L01_listen(rf, 0, address);
		}
		_delay_ms(500);
	}
	return 0;
}

// Port Setup for the nRF24L01
```

```
nRF24L01 *setup_rf(void) {
	nRF24L01 *rf = nRF24L01_init();
	rf->ss.port = &PORTB;
	rf->ss.pin = PB2;
	rf->ce.port = &PORTB;
	rf->ce.pin = PB1;
	rf->sck.port = &PORTB;
	rf->sck.pin = PB5;
	rf->mosi.port = &PORTB;
	rf->mosi.pin = PB3;
	rf->miso.port = &PORTB;
	rf->miso.pin = PB4;
	// interrupt on falling edge of INT0 (PD2)
	EICRA |= _BV(ISC01);
	EIMSK |= _BV(INT0);
	nRF24L01_begin(rf);
	return rf;
}

// Based on message received, the system is enabled from Level 1-4
void process_message(char *message) {
	if (strcmp(message, "AL") == 0)
	process_system(LEVEL1);
	else if (strcmp(message, "BL") == 0)
	process_system(LEVEL2);
	else if (strcmp(message, "CL") == 0)
	process_system(LEVEL3);
	else if (strcmp(message, "DL") == 0)
	process_system(LEVEL4);

}

// The system rotates the stepper motor to the respective angle,
// shoots the candy and then returns the stepper motor
// back to original state
void process_system(int step) {
	int stepCurrentState;
	stepCurrentState = stepperMotorForward(step);
	PORTC ^= (1 << PC4);
	servoControl(1);
	_delay_ms(100);
	servoControl(0);
	_delay_ms(1000);
	PORTC ^= (0 << PC4);
	_delay_ms(100);
	stepperMotorBackward(step,stepCurrentState);
	_delay_ms(100);
}

// nRF24L01 interrupt
ISR(INT0_vect) {
	rf_interrupt = true;
}
```