Sherman

Functionality

* The main function of the indoor irrigation system is to make sure that your flowers are getting watered at the right time. He’s doing that by having a configurable watering schedule.

Involved technology

* Arduino (language)
* NFC library (for Arduino): https://github.com/elechouse/PN532
* NFC Tools (NFC app for smartphones): https://play.google.com/store/apps/details?id=com.wakdev.wdnfc&hl=ro&gl=US

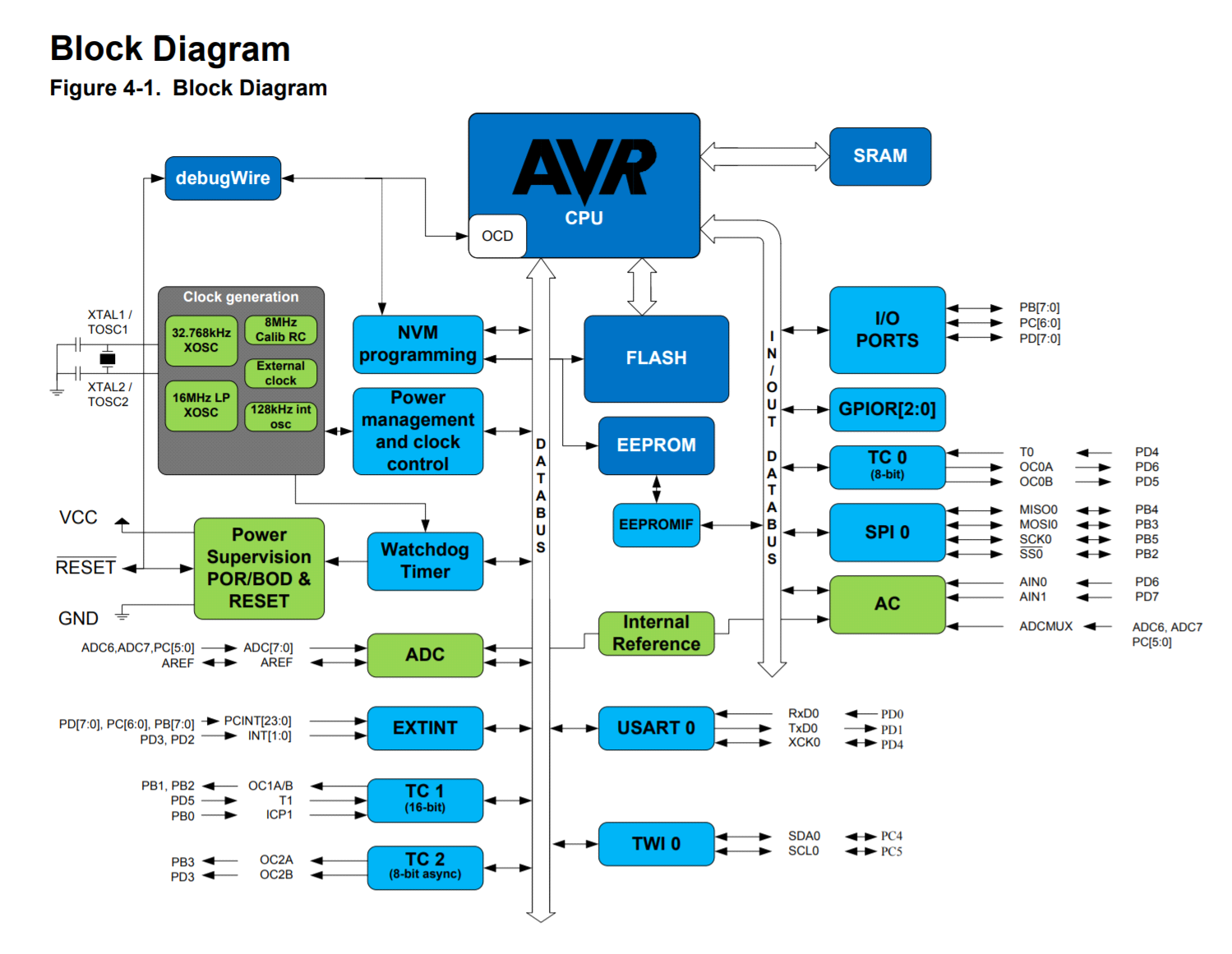
Used components

* Arduino UNO board
* 5V water pump
* RFID PN532 module
* SRD-05VDC-SL-C relay

Arduino UNO board

* The development board we used is Arduino Uno, with the following characteristics:

|  |  |
| --- | --- |
| Microcontroller | [ATmega328P](http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf) |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limit) | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) |
| PWM Digital I/O Pins | 6 |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 20 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 32 KB (ATmega328P) of which 0.5 KB used by bootloader |
| SRAM | 2 KB (ATmega328P) |
| EEPROM | 1 KB (ATmega328P) |
| Clock Speed | 16 MHz |
| LED\_BUILTIN | 13 |
| Length | 68.6 mm |
| Width | 53.4 mm |
| Weight | 25 g |



The Atmel MEGA328P microcontroller has the following features:

* High Performance, Low Power AVR® 8-Bit Microcontroller Family
* Advanced RISC Architecture
* 131 Powerful Instructions – Most Single Clock Cycle Execution
* 32 x 8 General Purpose Working Registers
* Fully Static Operation
* Up to 20 MIPS Throughput at 20MHz
* On-chip 2-cycle Multiplier
* High Endurance Non-volatile Memory Segments
* 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
* 256/512/512/1KBytes EEPROM
* 512/1K/1K/2KBytes Internal SRAM
* Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
* Data retention: 20 years at 85°C/100 years at 25°C(1)
* Optional Boot Code Section with Independent Lock Bits
* In-System Programming by On-chip Boot Program
* True Read-While-Write Operation
* Programming Lock for Software Security
* QTouch® library support
* Capacitive touch buttons, sliders and wheels
* QTouch and QMatrix™ acquisition
* Up to 64 sense channels

Peripheral Features

* Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
* One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
* Real Time Counter with Separate Oscillator
* Six PWM Channels
* 8-channel 10-bit ADC in TQFP and QFN/MLF package
* Temperature Measurement
* channel 10-bit ADC in PDIP Package
* Programmable Serial USART
* Master/Slave SPI Serial Interface
* Byte-oriented 2-wire Serial Interface (Philips I2C compatible)
* Programmable Watchdog Timer with Separate On-chip Oscillator
* On-chip Analog Comparator
* Interrupt and Wake-up on Pin Change
* Special Microcontroller Features
* Power-on Reset and Programmable Brown-out Detection
* Internal Calibrated Oscillator
* External and Internal Interrupt Sources
* Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
* I/O and Packages
* 23 Programmable I/O Lines
* 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF

Operating Voltage:

* 1.8 - 5.5V

Temperature Range:

* 40°C to 85°C

Speed Grade:

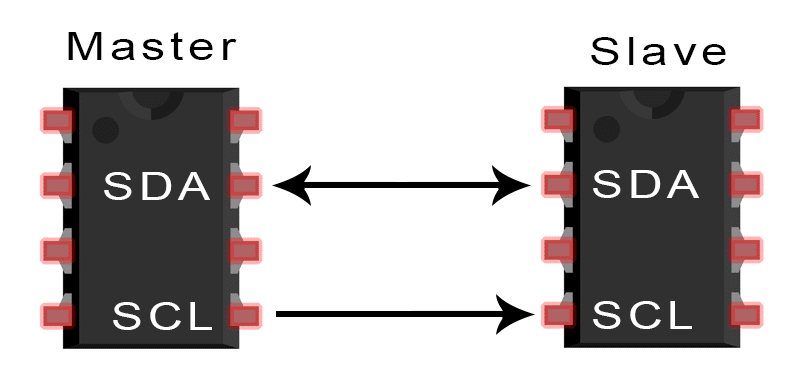
* 0 - 4MHz@1.8 - 5.5V, 0 - 10MHz@2.7 - 5.5.V, 0 - 20MHz @ 4.5 - 5.5V

Power Consumption at 1MHz, 1.8V, 25°C

* Active Mode: 0.2mA
* Power-down Mode: 0.1µA
* Power-save Mode: 0.75µA (Including 32kHz RTC)

Communication between Arduino and RFID module

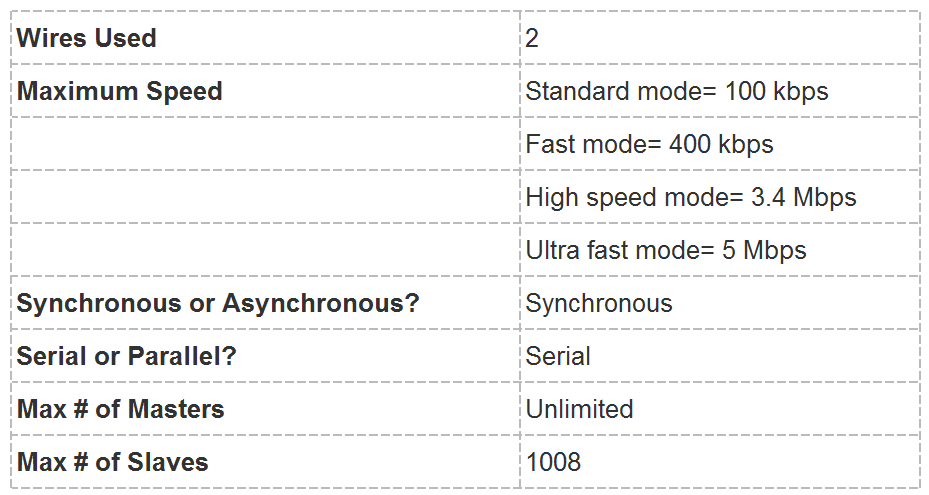
In the project, the **I2C** is used in order to transfer data between the development board and the RFID PN532 module.



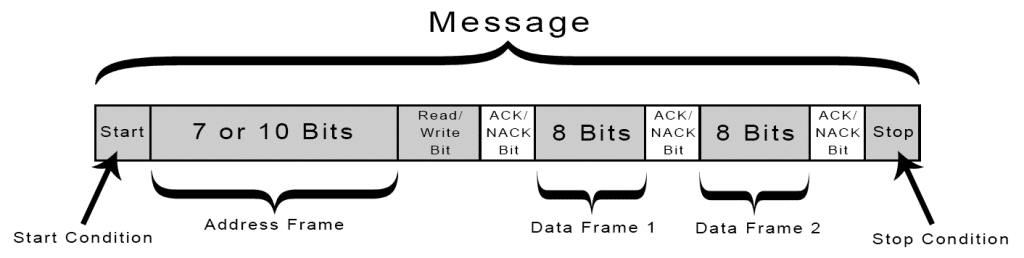
**SDA (Serial Data)** – The line for the master and slave to send and receive data.

**SCL (Serial Clock)** – The line that carries the clock signal.

I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line).



With I2C, data is transferred in messages. Messages are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:

[](https://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-I2C-Message-Frame-and-Bit-2.png)

**Start Condition:** The SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low.

**Stop Condition:** The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high.

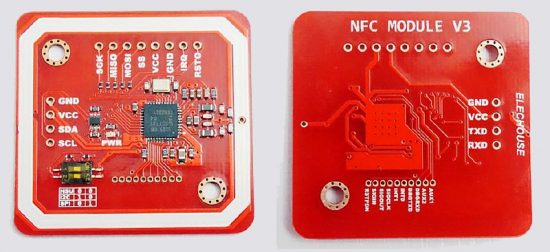
**Address Frame:** A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.

**Read/Write Bit:**A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).

**ACK/NACK Bit:** Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

RFID PN532 module

* Supports II2, SPI, and high-speed UART (HSU)
* RFID reader/writer mode support for:
  + Mifare 1K, 4K, Ultralight, and DesFire cards
  + ISO/IEC 14443-4 cards such as CD97BX, CD light, DesFire, and P5CN072 (SMX)
  + Innovision Jewel cards such as the IRT5001 card
  + FeliCa cards such as RCS\_860 and RCS\_854
* Built-in PCB antenna (covered by white paint)
* 5- to 7-cm communication distance
* On-board level shifter
* Standard 5-V TTL for I2C/UART and 3.3-V TTL SPI (VCC 3.3 to 5 V)
* Works as an RFID reader/writer
* Works as a 1443-A card or a virtual card
* Supports NFC with Android phone

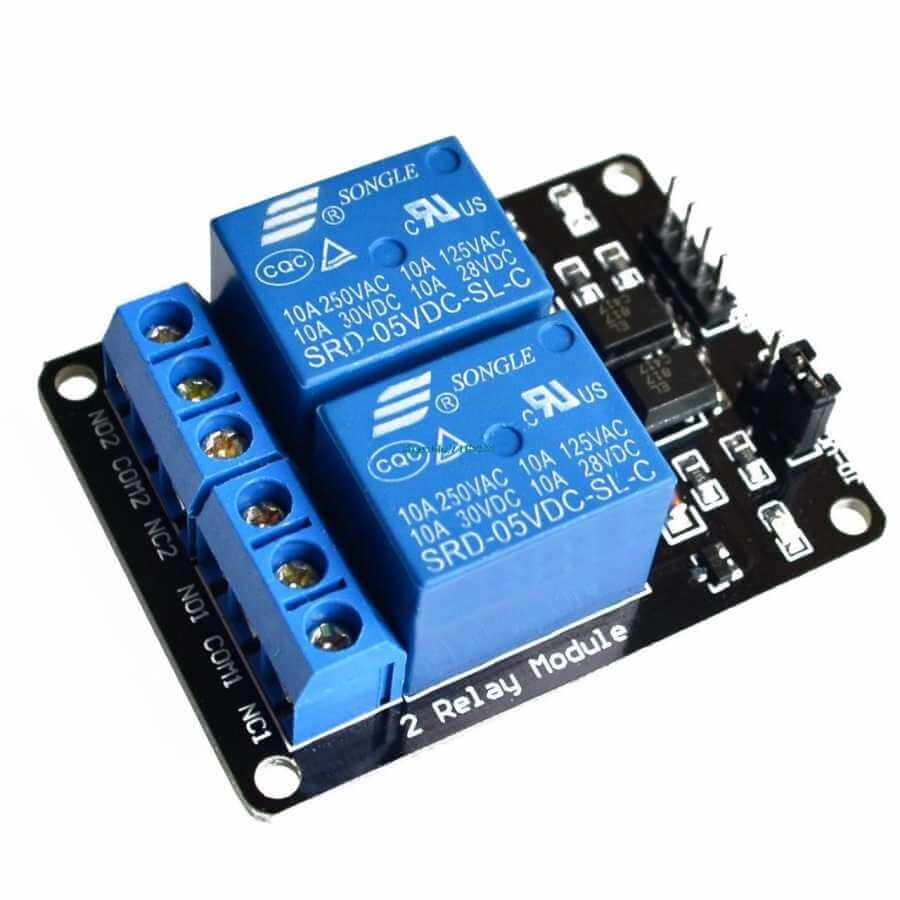


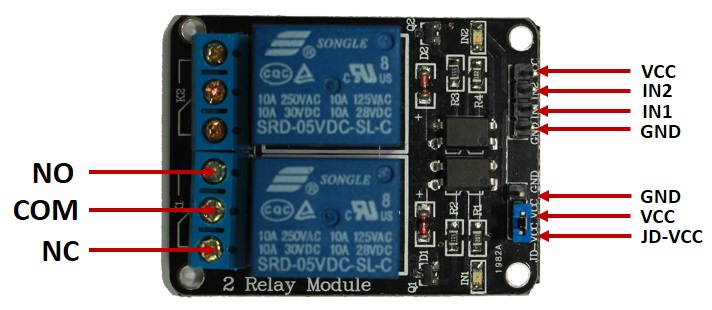
3 - 6V water pump



SRD-05VDC-SL-C relay

* Operating voltage: 5V
* Dimensions: 50 x 26 x 18.5mm
* Maximum load: AC 250V/10A, DC 30V/10A





* **COM**: common pin (connected to the signal you are planning to switch)
* **NC (Normally Closed):** pin is connected to the COM pin by default, unless you send a signal from the Arduino to the relay module to break the connection
* **NO (Normally Open):** pin is open by default, unless you send a signal from the Arduino to the relay module to make the connection.
* **GND**: goes to ground
* **IN1**: controls the first relay
* **IN2**: controls the second relay
* **VCC**: goes to 5V

Programming (Arduino)

#include <Wire.h>

#include <PN532\_I2C.h>

#include <PN532.h>

#include <NfcAdapter.h>

#define PUMP\_PIN 8

#define BUTTON\_PIN 3

#define ONE\_SECOND 1000

enum mode\_type {Timer, Sensor};

PN532\_I2C pn532i2c(Wire);

PN532 nfc(pn532i2c);

uint8\_t watering\_time = 2; //in seconds

mode\_type mode = Timer;

uint32\_t current\_time = 0; //in seconds;

uint32\_t watering\_interval = 1; //in minutes

uint32\_t intervals\_array[10] = {1, 5, 15, 30, 60, 120, 240, 480, 960, 1400}; //minutes

uint32\_t watertime\_array[10] = {1, 2, 3, 5, 8, 10, 15, 30}; //seconds

uint16\_t humidity\_level;

NfcAdapter nfc2 = NfcAdapter(pn532i2c);

NdefMessage msg;

NdefRecord rcd;

byte payload\_array[100];

int payload\_length = 0;

volatile int interrupt\_called; //used like a flag to be able to create new config

void read\_message()

{

if (nfc2.tagPresent())

{

NfcTag tag = nfc2.read();

tag.print();

msg = tag.getNdefMessage();

rcd = msg.getRecord(0);

rcd.getPayload(payload\_array);

if(payload\_array[3]=='6' && payload\_array[4]=='9') //first 2 bytes from NDEF should be 6 and 9

{

payload\_length = rcd.getPayloadLength();

Serial.println("Received tag from irrigation system app");

process\_message();

}

else if(payload\_array[3]=='7' && payload\_array[4]=='0')

{

payload\_length = rcd.getPayloadLength();

Serial.println("Received tag from irrigation system app");

process\_message();

}

else

{

Serial.println("Received tag from unknown device");

}

interrupt\_called = 0;

delay(1000);

}

else

{

// PN532 probably timed out waiting for a card

Serial.println("Waiting for a device...");

}

}

void process\_message()

{

/\*

Message structure:

payload\_array[3] & payload\_array[4]: used for recognising app and mode

payload\_array[5] : used for watering\_interval

payload\_array[6]: used for watering time

\*/

uint32\_t temp\_watering\_interval, temp\_watering\_time, temp\_mode;

temp\_watering\_interval = intervals\_array[payload\_array[5] - 48];

temp\_watering\_time = watertime\_array[payload\_array[6] - 48];

temp\_mode = (payload\_array[3]=='7');

change\_config(temp\_mode, temp\_watering\_interval, temp\_watering\_time);

}

void change\_config(uint8\_t i\_mode, uint32\_t i\_watering\_interval, uint8\_t i\_watering\_time)

{

mode = i\_mode;

watering\_interval = i\_watering\_interval;

watering\_time = i\_watering\_time;

current\_time = 0;

Serial.print("watering\_interval = ");

Serial.println(watering\_interval);

Serial.print("watering\_time = ");

Serial.println(watering\_time);

Serial.print("mode = ");

Serial.println(mode);

}

void crazy\_loop()

{

Serial.println(current\_time);

if(mode == Timer)

{

if(current\_time >= (watering\_interval \* 60 \* ONE\_SECOND))

{

water();

current\_time = ONE\_SECOND + watering\_time \* ONE\_SECOND;

}

else

{

current\_time = current\_time + ONE\_SECOND;;

}

}

else if(mode == Sensor)

{

if(humidity\_level < 400)

{

water();

}

}

delay(ONE\_SECOND);

}

void water()

{

digitalWrite(PUMP\_PIN, LOW);

delay(watering\_time \* ONE\_SECOND);

digitalWrite(PUMP\_PIN,HIGH);

delay(ONE\_SECOND);

interrupt\_called = 0;

}

void ISR\_button()

{

interrupt\_called = 1;

}

void setup(void)

{

Serial.begin(115200);

Serial.println("Initialising");

nfc.begin();

uint32\_t versiondata = nfc.getFirmwareVersion();

Serial.println(versiondata);

//commented to test sleep mode

if (! versiondata)

{

Serial.print("Didn't find PN53x board");

while (1); // halt

}

// Got ok data, print it out!

Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);

Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);

Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);

// Set the max number of retry attempts to read from a card

// This prevents us from waiting forever for a card, which is

// the default behaviour of the PN532.

nfc.setPassiveActivationRetries(0xFF);

// configure board to read RFID tags

nfc.SAMConfig();

Serial.println("Waiting for an ISO14443A card");

//configure interrupt pin

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

attachInterrupt(digitalPinToInterrupt(BUTTON\_PIN),ISR\_button,FALLING);

pinMode(PUMP\_PIN, OUTPUT);

digitalWrite(PUMP\_PIN, HIGH);

interrupt\_called = 0;

}

void loop(void)

{

if(interrupt\_called == 1)

{

read\_message();

}

else

{

crazy\_loop();

}

}