Using microcontrollers and solid state sensors

Meirin Evans

9214122

School of Physics and Astronomy

The University of Manchester

Second Year Laboratory Report

May 2016

This experiment was performed in collaboration with *John Cobbledick*.

Abstract

Through using a microcontroller combined with electronic components, output voltage measurements from various circuits were made. Along with a Radio Frequency Identification Device (RFID), the microcontoller was used to convert the device's binary reading to an RFID tag's individual number.

1. Introduction

Microcontrollers are small computers which are connected via Universal Serial Bus (USB) to a larger computer which programs the microcontroller. They have an inbuilt circuit board to connect to other electronic components enabling them to measure the outputs as well as control components. Microcontollers can be used in inductor/capacitor/resistor circuits as well as with amplifiers, transistors and diodes. They can also be used with an oscilloscope's wave generator to produce a variety of voltages and the oscilloscope can measure the microcontroller's output voltage. Solid state sensors are used widely; examples include Radio Frequency Identification Devices (RFID), temperature sensors, alcohol sensors, proximity sensors, range sensors, frequency to light converters and gas sensors [1].

2. Theory

Microcontrollers can be used to construct a peak and hold circuit, shown in Figure 1. The circuit includes an oscilloscope signal, a diode and a capacitor (with negative terminal grounded) in series. A voltage reading is taken between the capacitor and diode. The circuit bears this name since the signal passes through the diode and charges the capacitor, giving a voltage across it, but when the capacitor discharges no current can pass through the diode in the other direction so it holds the same voltage. A higher capacitance means a higher time constant (longer discharge time) so the peak is held for longer.

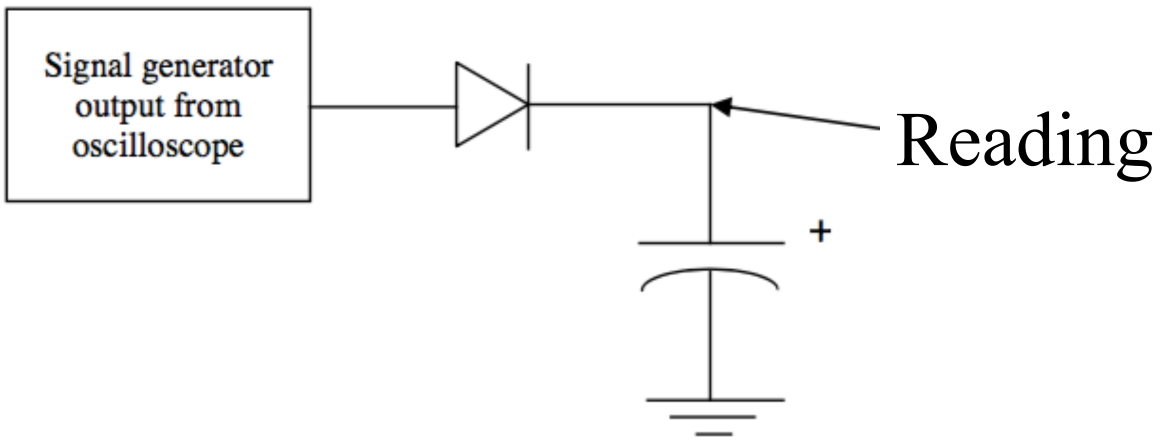


Fig 1. Circuit diagram for peak and hold. Includes an oscilloscope signal, a diode and a capacitor.

RFID tags have individual decimal numbers associated with them, for example school cafeteria tags as illustrated in Figure 2. For cafeteria tags, this number may then be linked to the holder's name and balance, which could then be displayed on a screen. When a tag is flashed within 1-2 cm of a sensor it reads the tag's number as a series of binary bits with different time lengths. This includes bits that notify the sensor as to when the signal starts and ends. It is usually sufficient to flash the tag in front of the sensor. The binary number is converted to hexadecimal (base 16) within the microcontroller, as shown in Table 1. Converting the microcontroller output to a string using a programming language gives the number in ASCII format. ASCII format is a system used by computers to dis-

Fig 2. Picture of RFID tags, like those used in school cafeterias [2].

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ASCII	48	49	50	51	52	53	54	55	56	57	65	66	67	68	69	70

Table 1. Table showing binary, hexadecimal and ASCII equivalents for decimal 0-15.

play characters like numbers, letters and punctuation or to give instructions to the computer. Once the number is in ASCII it can be directly converted to decimal to give a recognisable number.

A 5 digit (for example) hexadecimal *ghijk* is converted to decimal through Equation 1;

$$ghijk = 16^4 g + 16^3 h + 16^2 i + 16 j + k \, . \tag{1}$$

The value of each digit is multiplied by increasing powers of 16 from the right, where *g* through *k* can be 0-9 or A-F (corresponding to 10-15). In ASCII, a 2 indicates the start of a signal whilst a 3 indicates the end. The last 2 digits in the hexadecimal number are a checksum and are therefore ignored when calculating the decimal number. The first 2 digits are also ignored as these are only used to calculate the checksum. The checksum is useful in verifying whether the information was correctly received. To do this, the tag number's hexadecimal digits are paired then an XOR operation is performed on all pairs together. To understand an XOR operation numbers are converted to binary and the binary digits of the numbers are then compared. An XOR gives 1 if digits are different and 0 if they are the same, as illustrated in Table 2. If the XOR operations of all the pairs corresponding to the tag number give the checksum, the data has been read correctly.

Hexadecimal	Decimal	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
3C	60	0	0	1	1	1	1	0	0
CC	204	1	1	0	0	1	1	0	0
F0	240	1	1	1	1	0	0	0	0

Table 2. Example of XOR operation on 2 hexadecimal numbers 3C and CC. The binary representations are put in columns of decreasing powers of 2 for ease in calculating the result. 3C XOR CC gives F0.

3. Experimental method

In the peak and hold circuit, capacitors of 1nF and 100nF were used to compare behaviour.

The microcontroller used was an Arduino UNO [3] board which could be programmed using the Arduino language, which is similar to C. An RDM630 [4] operating at 125kHz was used as an RFID sensor. Pictures of both are in Figure 2. The program converted the ASCII characters read by the Arduino to tag number, which was then printed to the Arduino's serial port. The program also did the XOR operations to obtain the tag's checksum and compared it with the read value. The program used to read the RIFD tags is in the Appendix. Viewing the series of bits read by the sensor on an oscilloscope showed that the first 2 digits of the hexadecimal number should be ignored in calculating the tag number.

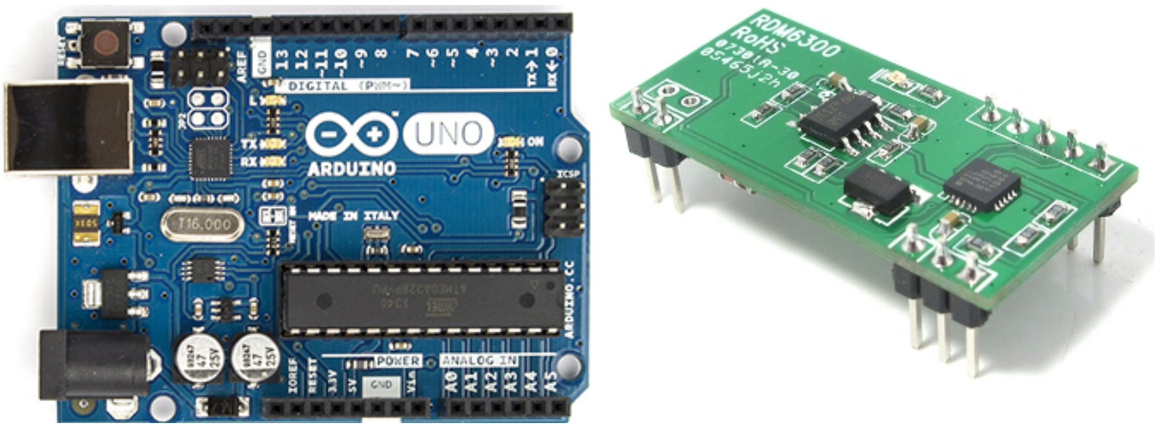


Fig 2. Picture of an Arduino UNO board [5] (left). Picture of RFID sensor RDM630 [6] (right).

4. Results

Graphs of the input and output voltage shapes for both capacitors are shown in Figure 3.

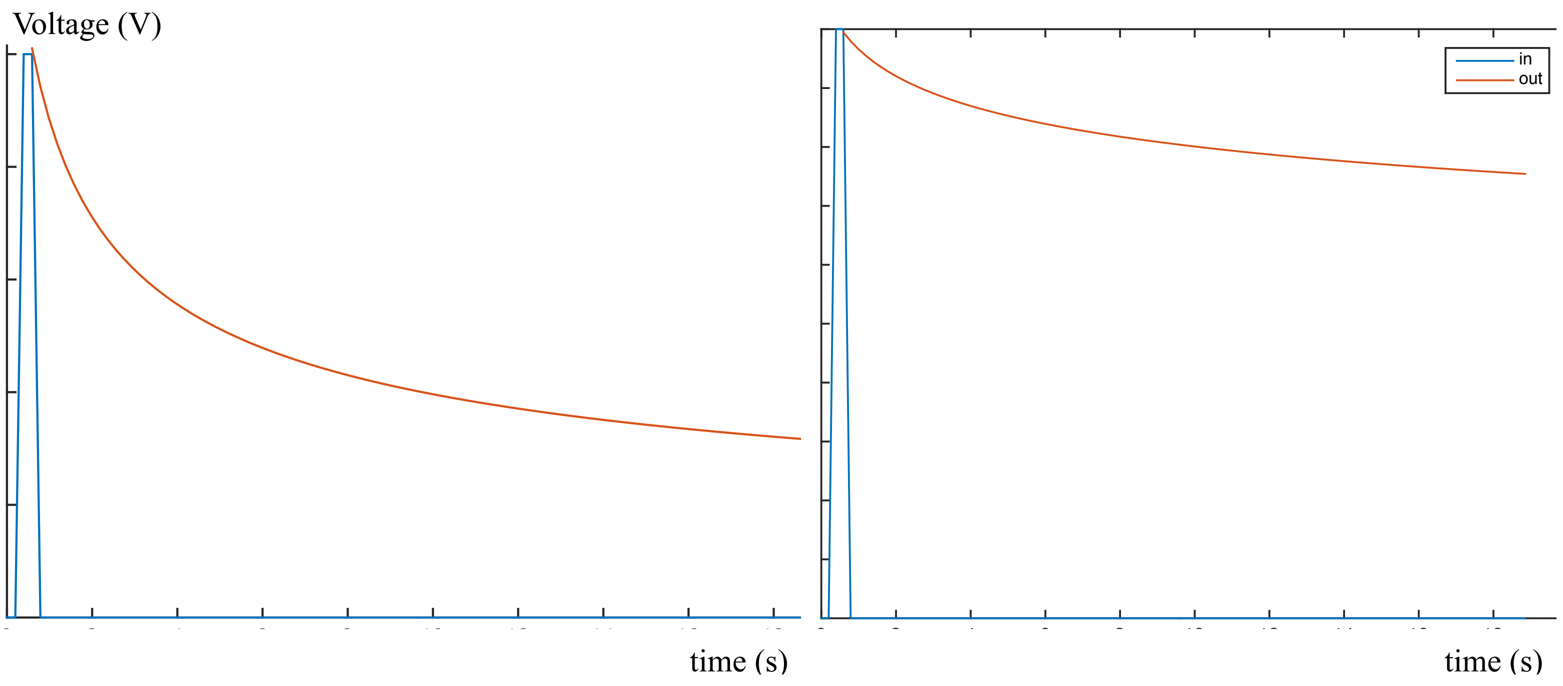


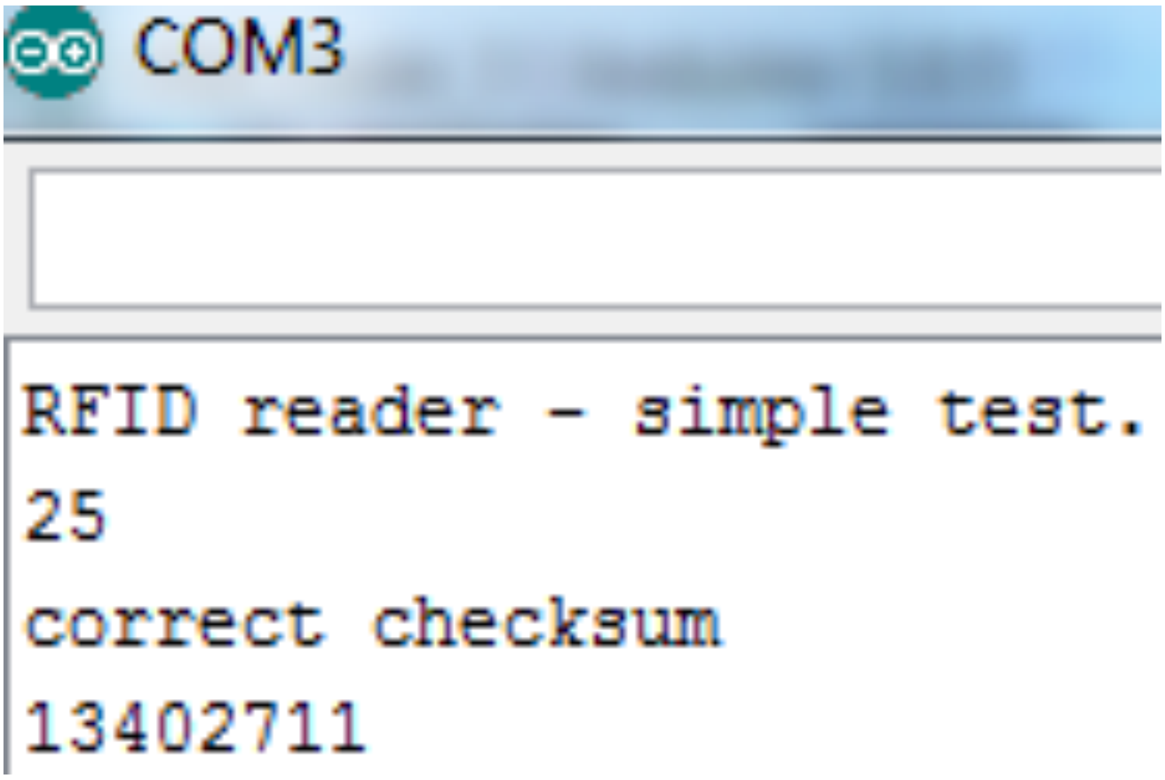
Fig 3. Plots of input and output voltage against time. Left is 1nF, right is 100nF. 100nF holds the peak for longer than 1nF.

The results from reading RFID tags are shown in Table 3. All the tags' calculated checksums agreed with the initial reading.

Hexadecimal	3C00CC825725	3C00CEF5C3C4
ASCII	2 51 67 48 48 67 67 56 50 53 55 50 53 3	2 51 67 48 48 67 69 70 53 67 51 67 52 3
Tag number	13402711	13563331

Table 3. Table showing conversions from hexadecimal to ASCII to tag number for RFID tags.

A screenshot of an example output from the Arduino's serial is shown in Figure 4. It shows an introductory comment, the tag's checksum, a statement saying whether the checksum was read correctly and the tag's number.



```
RFID reader - simple test.
25
correct checksum
13402711
```

Fig 4. Screenshot of the Arduino's output on the serial for one of the tags used. Includes an introductory comment, the tag's checksum, a statement saying whether the checksum was read correctly and the tag's number.

5. Discussion

The main difficulty in converting the hexadecimal reading of an RFID tag to its tag number was in the step of converting hexadecimal to ASCII. Particularly troublesome was splitting the whole hexadecimal number into its individual digits to then put into arrays. Problems arose when trying to access array elements. To overcome this a do-while loop was used to read the tag's hexadecimal digits whilst a for loop was used to do conversions, as opposed to a for loop for both.

Another detail requiring particular attention is the use of long format rather than int when calculating the final number. The maximum decimal using int is 32767, obtained from the fact that an int is held in 16 bits and 2^{16} is 65536, but negative numbers as well as 0 need to be included. A long, however, is held in 32 bits and 2^{32} is 4294967296, giving a maximum long of 2147483647. The final tag numbers were between the maximum values of int and long therefore long was appropriate. Int was used in other steps to speed up the program, though it probably did not change the speed much.

6. Summary

Microcontrollers are very useful devices to manipulate electronic components, which are themselves easily manipulable using programming. They are most useful when used with solid state sensors to read, convert and output information from the sensors.

References

[1] Lalauze, R. & Pijolat, C., "A new approach to selective detection of gas by an SnO_2 solid-state sensor", *Sensors and Actuators,* Volume 5, Issue 1, 1984.

[2] http://www.globalspec.com/learnmore/data_acquisition_signal_conditioning/data_input_devices/rfid_tags, *RFID Tags Information*, IHS Engineering360, Accessed 30/03/2016.

[3] https://www.arduino.cc/en/Main/ArduinoBoardUno, *Arduino UNO & Genuino UNO*, Arduino, Accessed 21/03/2016.

[4] http://www.seeedstudio.com/depot/datasheet/RDM630-Spec..pdf, *RDM630 Specification,* Seed Studio, Accessed 21/03/2016.

[5] https://www.arduino.cc/en/Guide/Windows, *Getting Started with Arduino and Genuino on Windows*, Arduino, Accessed 21/03/2016.

[6] http://store.iteadstudio.com/index.php?main_page=product_info&products_id=6, *125kHz RFID module RDM630 - UART*, iStore, Accessed 21/03/2016.

The number of words in this document is 1547.

This document was last saved on 8/5/2016 at 21:31.

Appendix

```
// allow use of mySerial
#include <SoftwareSerial.h>

// use pin 10 on Arduino
SoftwareSerial mySerial(10, 11); // RX, TX.

// create arrays of ints
int fig[14];
int check[5];

// setup loop
void setup()
{
  // Open hardware serial port for Serial Monitor.
  Serial.begin(9600);
  // debug statement
  Serial.println("RFID reader - simple test.");

  // Set the baud rate for the SoftwareSerial port.
  mySerial.begin(9600);
}

// main loop
void loop()
{
  // initialise j
  int j = 0;
  // long format for tag num
  long dec;
  // do while loop for reading
  do {
    if (mySerial.available() > 0) {
      fig[j] = mySerial.read();
      j++;
    }
  } while (j < 14);
  // for loop to do conversions
  for (int i=0; i < 14; i++) {
    if (fig[i] > 47 && fig[i] < 58) {
      fig[i] = fig[i]-48;
    }
    else if (fig[i] > 64 && fig[i] < 71) {
      fig[i] = fig[i]-55;
    }
  }
  // for loop to pair digits
  for (int k = 1; k < 12; k = k + 2) {
    check[k]= fig[k]*16 + fig[k+1];
  }
  // initialise csum
  int csum = 0;
  // for loop for checksum
  for (int h = 1; h < 10; h = h + 2){
    csum = csum^check[h];
  }
  // print checksum
  Serial.println(csum,HEX);
  if (csum == check[11]) {
  Serial.println("correct checksum");
  }
  // calculate tag num
  dec = fig[10] + fig[9]*16 + fig[8]*power(16,2) + fig[7]*power(16,3) +
            fig[6]*power(16,4) + fig[5]*power(16,5);
  // print tag num
  Serial.println(dec);
}

// user defined power func
long power(int a, int b) {
  long c = a;
  for (int p = 0; p < b - 1; p++){
    c = a*c;
  }
  return c;
}
```