Lucas Plager

**Microprocessor for Fire Alarms**

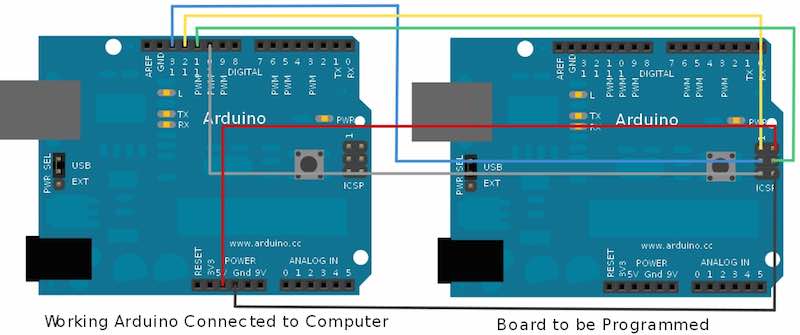
Each fire alarm needs it’s own microprocessor to handle computing for the wireless signals and sensors that will be part of our smoke sensor design. The microprocessor will need to be low power and will not need a very high amount of processing power. For this reason, we have decided to use the ATmega328 microcontroller developed by Atmel. This chip is a low-power CMOS transistor microcontroller. The ATmega328 uses a simple 8-bit RISC architecture for executing simple instructions. This architecture is perfect for reading inputs from simple sensors and proving output signals to components such as LEDs. The RISC architecture that is used by the ATmega328 contains 131 instructions that operate in a single clock cycle. This microprocessor is capable of 20 million instructions per second when running at 20 MHz. Another important component of this chip is that is contains 23Kbytes of programmable flash memory. This memory is where the bootloader and functional code for our program will be stored. This flash memory is re-programmable allowing us to update and change the code as many times as needed during testing. The ATmega328 also comes with six sleep modes which may come in handy for us if we need to save battery life and power consumption on our system. This chip runs at an operating voltage of between 1.8 and 5.5V according to the data sheet provided by Atmel. The last of the important characteristics on this chip for our fire alarm system is that it contains 23 programmable I/O lines. These lines will be used to connect to the various peripherals of our alarm such as the buzzer, smoke sensors, LEDs, batteries, and wireless peripherals. One final reason that we chose this microprocessor is that it is the same one used in the Arduino Uno. This means that we can use a lot of the same resources that are provided with the Arduino Uno without having to worry about compatibility issues. Lastly, we can use an Arduino Uno to load a bootloader onto our chip alleviating the issues of choosing and writing our own bootloader. (http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P\_Datasheet.pdf)

**Bootloader**

Because we needed to purchase multiple microprocessors for use with many custom fire alarms we are going to build it was not feasible to purchase multiple Arduino Uno boards. This lead us to purchase ATmega328 processors that did not have a bootloader installed so we will need to install one ourselves. The purpose of a bootloader is to be a small program that runs when the system is powered on before the main program is run. This bootloader software will tell our fire alarm system to wait for the software on our programming computer to sent a new program to our fire alarm to be loaded. Our custom program would then be loaded into the flash memory on out ATmege328 processor. This bootloader is what will enable us to load programs onto out fire alarms using just a USB cable without the fire alarm going straight to running whatever program is installed on the system. Because we chose to use an ATmega328 microprocessor, we can use the same bootloader that is installed on Arduino Uno boards saving us a great amount of time in writing our own bootloaders. (<https://www.arduino.cc/en/Hacking/MiniBootloader)>. This also allows us to use the Arduino integrated development environment to program our fire alarms. Our team will only need to borrow an Arduino Uno for a short amount of time in order to create an unlimited amount of Arduino processors of our own.

**Bootloading Process**

Because our ATmega328 chips do not come with a bootloader preinstalled, we will need to install our own. Other than the obvious ATmega328, we will also need a few more components to install the bootloader onto the chip. We will need a breadboard for mounting the chip onto as well as a 16MHz crystal. This crystal in conjunction with two capacitors of about 22pF will act as a clock for our processor. Lastly, we will also temporarily need a working Arduino Uno in order to copy the bootloader over to our blank ATmega328 chip. The below diagram is an example of how an Arduino can be used to load the bootloader onto the chip:



provided by: <https://diyhacking.com/make-arduino-board-and-bootload/>

The first step for loading the bootloader is to place the ATmega328 chip into the middle of the breadboard allowing the pins on the left and right side of the chip to connect the two separated segments of the breadboard. The ATmega328 has a small indentation indicating which is the top of the chip, or rather which side pin one is on. This side should be oriented so that it is facing the top of the breadboard. VCC then needs to be connected to pin 7 and pin 20 which should be set to the input voltage of our ATmega328, 5 volts. Pins 8 and 22 on our chip need to be connected to ground. For designing our clock, we need to connect our 16MHz crystal to pins 9 and 10 on our board. Our capacitors that we have chosen need to be connected from each pin on the crystal to ground. Therefor, one capacitor needs to be connected to pin 9 and ground while another needs to be connected to pin 10 and ground. Our Arduino Uno board now needs to be connected to our computer using the USB connection on the board. The Arduino IDE comes with example sketches that can be loaded onto Arduino boards for testing purposes. One of these is called Arduino ISP and loads software onto the Arduino which will allow it to function as a bootloader for our ATMega328. This sketch needs to be uploading to our Arduino without any connections other than to the computer. Once this is complete the Arduino needs to be wired to our blank ATMega328 chip. The first connection is Pin 10 on the Arduino board which needs to be connected to the reset pin on our blank chip. The reset pin in Pin 1. Next, Pin 11 on the Arduino needs to be connected to pin 17 on our blank chip and Pin 12 on the Arduino needs to be connected to pin 18 on our blank chip. The last wire than needs to be connected is pin 13 on the Arduino which needs to be connected pin 19 on the blank chip and breadboard. Now we can connect our 5 volt VCC and ground and plug the Arduino back in to the computer. The last step is to go onto the Arduino IDE, confirm that the correct Arduino board is selected, which is the Arduino Uno, and select Burn Bootloader. This process should take a few minutes to complete. As long as no errors are shown on the Arduino IDE, an Arduino bootloader has now been loaded onto our chip and can be used by itself to program our project. (https://www.youtube.com/watch?v=g90xb0nNX50)