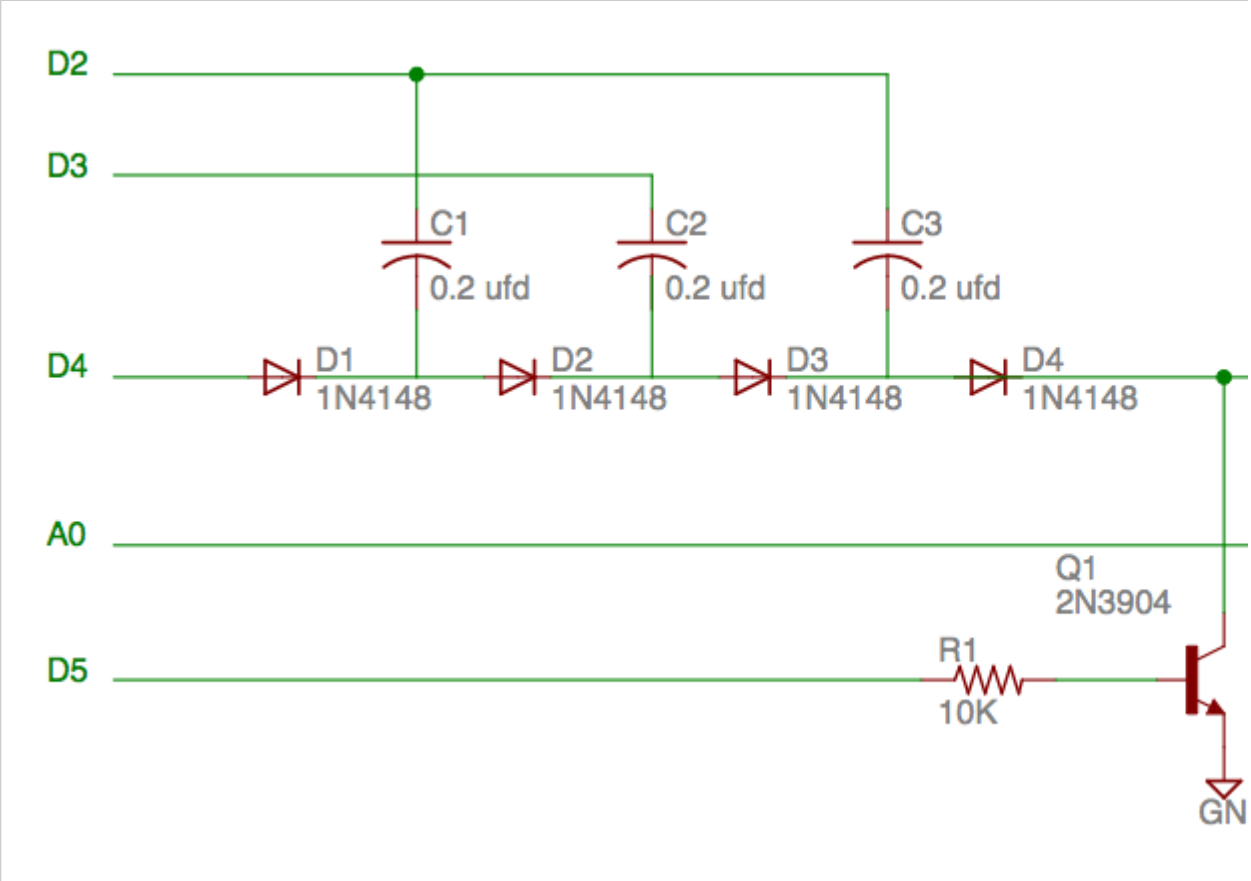
# Wayne's Tinkering Page

## 12 Volt Arduino-based Charge Pump

In my ATTiny Fuse Reset project I had to use 8 AA batteries to get the 12 volts needed to drive the RESET pin. This seemed rather inelegant to me, so I decided to look for a cleaner way to get 12 volts, especially a solution that would fit better on a PC board. One part that is commonly used for this application is the Maxim MAX662A 12, 30 mA Charge Pump. However, when I went to order the part from my favorite parts suppliers, I noted the price was over $5 in DIP form. This seemed a bit high for a part that performed such a simple function, but I ordered one anyway. However, in reflecting back on what I'd read about charge pumps, I began to wonder if I could just build a charge pump and use some leftover compute power on the Arduino to drive it.

In principle, a charge pump is very simple. For example, you have a source of 5 volts and you connect two capacitors in parallel and let them charge to 5 volts. Then, you reconnect them in series and, voila!, you have 10 volts. It seems like cheating, but that's just how capacitors work. And, you can scale this up to get higher and higher voltages just by using more capacitors. But, to make this scheme practical, you need switches to do the work of connecting the capacitors in parallel, then in series. You also need another capacitor to temporarily hold the series charge while the other capacitors are charging in parallel. Then, you need a device to control the process and regulate the switching in a way that delivers a controlled output voltage even under a varying load. I sounds complicated, but i turns out you can do this with only 4 diodes, 4 capacitors, 3 resistors, a 2N3904 transistor and about 2 dozen lines of code on the Arduino. And, actually, the transistor and one of the resistors is only needed if you need to cleanly switch the 12 volts on and off, as is needed in ATTiny Fuse Reset application.

When I was describing the way the capacitors need to be connected back in forth in parallel, then series configuration, you may have imagined that it would take quite a few transistors to do this. But, it turns out there is a circuit called the Dickson Charge Pump that uses diodes to handle a few of the switching functions. The remaining switching can be handled by two two Arduino output pins. The circuit diagram for my final design is shown below. Diodes D1 - D4 and capacitors C1 - C4 form the charge pump following Dickson's original work:




Resistors R2 and R3 form a voltage divider that step down the 12 volt output into a range the Arduino's analog input pins can handle. Resistor R1 and transistor Q1 are used to quickly discharge the capacitors so the 12 volt supply can be quickly shut down. In operation, pin D4 is set to a HIGH value to supply a 5 volt input to the charge pump. Then, pins D2 and D3 are switched between HIGH and LOW at a suitable rate (around 2 kHz) with the requirement that D2 must be LOW whenever D3 is HIGH and vice versa. This switching action, left unregulated, will generate about 15 volts across capacitor C4, so the next step is to devise a way to control the switching so a steady 12 volts is produced. The test code I wrote to try all this out is shown next (and also available as a downloadable Arduino project at the end of this page):

```
#include <TimerOne.h>

/*
 * Dickson Charge Pump Driver
 */

#define P1  0x04  // Pin D2
#define P2  0x08  // Pin D3
#define PWR 0x10  // Pin D4
#define GND 0x20  // Pin D5
#define REF 404

char phase = 0;
char onOff = 0;

void ticker () {
  if (onOff) {
    DDRD = P1 | P2 | PWR | GND;
    int volts = analogRead(A0);
    if (volts < REF) {
      if (phase) {
```

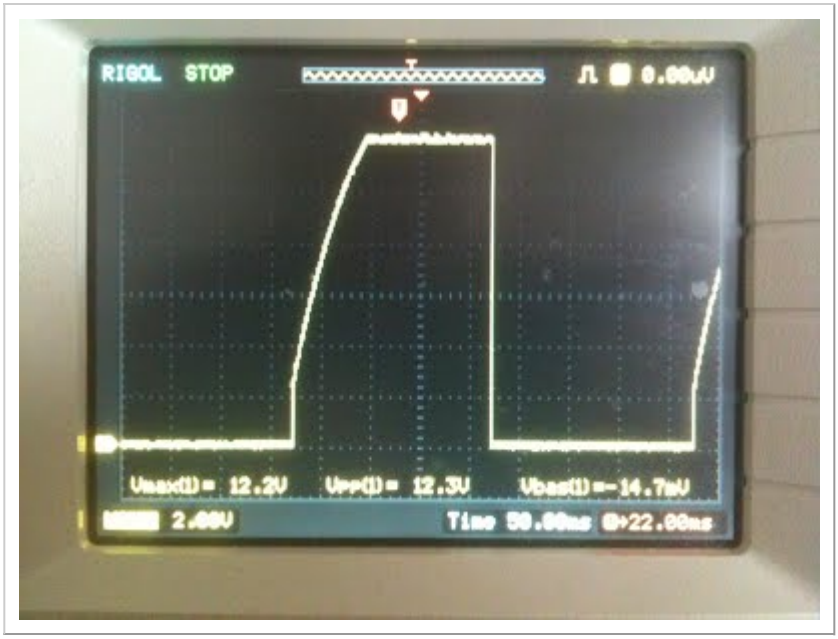
```
        PORTD = P1 | PWR;
      } else {
        PORTD = P2 | PWR;
      }
      phase ^= 1;
    }
  } else {
    DDRD = GND;
    PORTD = GND;
  }
}

void setup() {
  analogReference(DEFAULT);
  Timer1.initialize(500);
  Timer1.attachInterrupt(ticker);
}

void loop() {
  delay(200);
  onOff ^= 1;
}
```

Several things may look unfamiliar since they use advanced features not typically seen in most Arduino projects, but I'll explain how each works. First, the code sets up a periodic interrupt using the Arduino's `TimerOne` library so that the function `ticker()` is automatically called every 500 microseconds (a rate of 2000 Hz.) If you're new to using interrupts, you can think of the code in `ticker()` as executing automatically in the background once the time is started. Second, the code in pin I/O commands write directly to the IO ports using what the Arduino community calls Port Manipulation. This kind of I/O is much faster than using the more familiar `pinMode()` and `digitalWrite()` commands. This kind of efficiency is critical in interrupt code because interrupts steal time from the main code and, if they steal too muck time, the main code will bog down and fail to work properly.

The code in `ticker()` is fairly easy to understand if you break it down step by step. At the top of the function, the variable `onOff` is checked to see if the 12 volt supply is currently needed, or not (this is a capability is needed by the ATTiny Fuse Reset code this code will be eventually merged with.) When `onOff` is set to 0 (LOW), the outermost else case in `ticker()` switches all but pin D5 (`GND`) to the input state. Pin D5 is set to 1 (HIGH) to turn on transistor Q1, which keeps C4 fully discharged and the output is zero volts. When `onOff` is set to 1 (HIGH), D2 - D5 are all set as outputs and D5 is set LOW to disable Q1 and D4 (`PWR`) is set HIGH to feed 5 volts to the input of the charge doubler. The charge pump is now ready to pump, but it's not yet doing so.

Before the change pump is engaged, the voltage across the voltage divider formed by R2 and R2 is read from analog input 0 (A0). If the value read is less than the `REF` constant (set to 404 in my circuit based on the tolerance of the resistors I used for R2 and R3), it means the voltage across C4 is less than 12 volts. In response, pins D2 and D3 pins are toggled to their opposite state which, in turn, pumps charge through capacitors C1 - C3. This toggling back and forth continues with each new call to `ticker()` with the result that the charge pump keeps feeding more and more charge into C4 until it reaches 12 volts.

Finally, for test purposes, the code in `loop()` is set to switch the 12 supply on and off every 200 milliseconds (remember, because we're using a timer interrupt, this code runs in parallel with the code in `ticker()`.) This test simulates how the 12 volt supply will be used in the ATTiny Fuse Reset code. Finally, the following scope trace shows how this works when all the pieces are put together.




If you don't have a scope, it may be easier to adjust the voltage if you change `loop()` so `onOff` is always 1 so the output voltage is always on. Then, adjust the value of `REF` as you measure the output with a volt meter. Lower values of `REF` will produce a lower output voltage. Note: the scope trace above was taken with the value of C4 set to a 4.7 ufd, as I was testing the relationship between rise time and ripple. Ultimately, I decided on 2 ufd as a reasonable compromise (as shown in the schematic.)

ChargePump.pde (1k) Wayne Holder, Nov 22, 2010, 6:5… v.4