# Yun Build Monitor

A standalone build monitor for Jenkins or Hudson builds
by Phil Haigh, Surfsoft Consulting Ltd
Version 1.0, 23/02/2014

## Overview

When I saw the Arduino Yun I thought it made a great starting point for Internet of Things projects and first on my list was a Jenkins build monitor. It is nice to be able to dedicate a monitor to display your build status but not everybody has a spare monitor and computer. This build monitor is relatively cheap and highly portable but can also be extended to drive bigger lights if you want to.

The concept of the build monitor is straightforward. The status of each build is indicated by one of three LEDs - red for failed, yellow for completed with failing tests, and green for completed with no failing tests. When a build is actually in progress then the lit LED flashes on and off. All this is squeezed onto an Arduino proto shield which in turn is plugged into an Arduino Yun.

The Yun itself is split into two parts. There is the ‘traditional’ Arduino board and then in addition there is a SOC running a GNU/Linux distribution (Linino) complete with WiFi, wired ethernet, and a microSD card slot. The Arduino can interrogate files on the microSD card and so this makes for a fairly straightforward division of labour: the Arduino is used to control the local hardware (the LEDs) while the Linux SoC is used to poll builds on the internet and update files on the MicroSD card, which in turn are read by the Arduino code.

## The Hardware

### Parts List

You don’t need many parts to get the basic build monitor running. This parts list assumes that you *already have* an Arduino Yun and a spare microSD card:

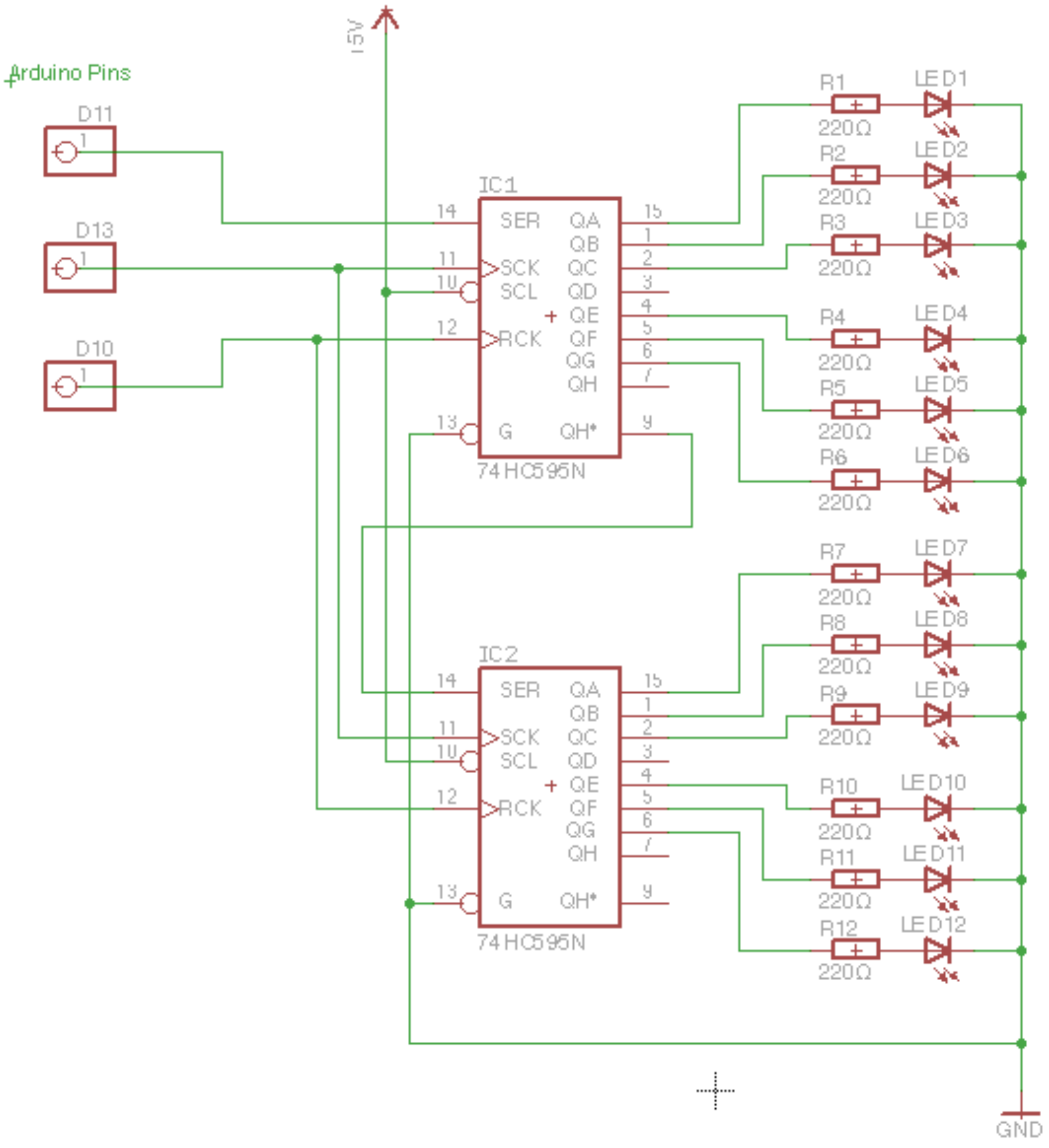
- A proto shield
- 2x 16-pin DIL socket
- 2x 74HC595
- 4x red LED
- 4x amber LED
- 4x green LED
- 12x 220 ohm resistor

If you are assembling the circuit yourself I recommend you purchase the pro to shield kit (code A000083 n the arduino.cc shop)  rather than the pre-assembled version (code A000077) as you’ll save yourself some money. You can’t stack boards on top of the kit version, but then that would hide all the LEDs.

### The Circuit

The circuit is quite simple, based on one of Arduino’s own demonstration circuits. It consists of a pair of 74HC595 shift registers, split into four groups of four outputs (which I number 0-3), with LEDs connected to outputs 1-3. Output 0 in each group is unused. There wasn’t enough room on the board for a fifth set of LEDs but there *is* enough room to add some headers to the board if you want to drive additional off-board hardware like a larger set of build lights.

Note that on this diagram I have omitted the +5v and Gnd connections to the ICs for clarity.



# The Software

The software is split into two parts, the Linux shell scripts and data files, and the Arduino code.

## Linux shell scripts and data files

On the microSD card there are four directories named job-0 through job-3. In each directory are the following files:

- url.txt - this specifies the URL of the hudson build being monitored.
- status.txt - this file will contain ‘2’ if the build is green, ‘4’ if it is amber and ‘8’ if it is red (these numbers were chosen to match values in the Arduino code to keep things simple).
- running.txt - this file will contain ‘1’ if the build is currently running and ‘0’ if it is not.

In the top level of the microSD card is a script called ‘refresh.sh’ which accepts a single parameter - 0, 1, 2 or 3 to identify the job to be updated. It goes out to the internet to obtain the build status and updates the status.txt and running.txt files.

Here is the source code for refresh.sh:

```
ROOT=/mnt/sd/job-$1
URL=`cat $ROOT/url.txt`
BUILD=$URL/api/json

STATUSLINE=`curl $BUILD | tr "," "\n" | grep color | tr ":" "\n" |
tail -1`

RUNNING=1
if test -z `echo $STATUSLINE | grep anime`
then
  RUNNING=0
fi

STATUS=8
if test ! -z `echo $STATUSLINE | grep yellow`
then
  STATUS=4
fi
if test ! -z `echo $STATUSLINE | grep blue`
then
  STATUS=2
fi

echo $STATUS > $ROOT/status.txt
echo $RUNNING > $ROOT/running.txt
```

The business part of this script is the curl command. This retrieves the overall status of the job, finds the property that specifies the build colour, and then derives the status and running values from it. Colour is either red, yellow or blue, and if it has the suffix ‘_anime’ this means that a build is currently running. Once the two values have been derived, we simply update the two files

To ensure that the build status is updated regularly and automatically I have used cron to refresh every build every minute. Here are the crontab entries:

```
* * * * * /mnt/sd/refresh.sh 0
* * * * * /mnt/sd/refresh.sh 1
* * * * * /mnt/sd/refresh.sh 2
* * * * * /mnt/sd/refresh.sh 3
```

Now by default cron is not enabled. So you’ll need to turn it on by issuing these two commands:

```
/etc/init.d/cron start
/etc/init.d/cron enable
```

You can tell when things are working properly, the timestamp on each of the status.txt and running.txt files will change every minute.

## Arduino code

The Arduino code will look familiar to anybody who has programmed even just a few sketches. Here I have broken it into sections to explain the key parts; the entire listing is available at the bottom of the article.

We start with something nice and simple - the declarations:

```
#include <FileIO.h>

const int latchPin = 10;
const int clockPin = 13;
const int dataPin = 11;

const int red = 8;
const int amber = 4;
const int green = 2;

const byte testValues[6] = { 128, 64, 32, 8, 4, 2 };

const String jobDirPrefix = "/mnt/sd/job-";
const String jobStatusFileName = "status.txt";
const String jobRunningFileName = "running.txt";
```

The latchPin, clockPin and dataPin values specify which pins the 74LS595 is wired up to while the three constants red, amber and green specify which bits of the byte are connected to which LED colours. You'll notice that they match the values written to status.txt.

The testValues constant is there to support the power-on self test (more on this later) and the remaining constants specify the location of the job files and the the names of the files themselves.

Next comes the setup function:

```
void setup() {

  pinMode(latchPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(clockPin, OUTPUT);

  Bridge.begin();
  FileSystem.begin();

  selfTest();

}
```

To start with we need to set the pins we are using to output mode. Then to enable the Arduino sketch to access files on the microSD card we must first start the Bridge and then the FileSystem. Finally we initiate a self test, the code for which is here:

```
void selfTest() {

  refresh(0, 0);

  for (int index = 0; index < 12; index++) {
    if (index < 6) {
      refresh(testValues[index], 0);
    }
    else {
      refresh(0, testValues[index - 6]);
    }
    delay(500);
  }

  refresh(0, 0);
  delay(2000);

}
```

The self test is very simple. It starts by turning off all the LEDs before cycling through each one in turn, turning it on for half a second. Finally it turns them all off again and waits for two seconds. This means that regardless of the state of the device's internet connection, when you first turn it on, you will see each LED turn on in sequence. This tells you that (a) the Arduino side of the device is working and (b) that all the LEDs are also working.

That code makes use of the refresh function:

```
void refresh(byte msb, byte lsb) {

  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, lsb);
  shiftOut(dataPin, clockPin, LSBFIRST, msb);
  digitalWrite(latchPin, HIGH);

}
```

This starts by turning off all the LEDs, then shifts out the two bytes of data to the pair of 74HC595 shift registers, and then turns everything back on again. Using the latchPin to turn everything on and off stops the LEDs flickering as the registers are updated.

Now we come to the main loop:

```
void loop() {

  displayJobStatus(1);
  delay(1000);
  displayJobStatus(0);
  delay(1000);

}
```

The function displayJobStatus actually updates the registers with the correct data to show the status of the builds. We call it twice at one second intervals to support the idea of having an LED blink on and off to indicate that a build is currently running. Here's the displayJobStatus function:

```
void displayJobStatus(byte toggle) {

  byte status3 = calculateJobStatus(3, toggle);
  byte status2 = calculateJobStatus(2, toggle);
  byte lsb = status3 + (status2 * 16);
  byte status1 = calculateJobStatus(1, toggle);
  byte status0 = calculateJobStatus(0, toggle);
  byte msb = status1 + (status0 * 16);
```

```
    refresh(msb, lsb);

  }
```

The calculateJobStatus function returns a four bit value for the specified build number, pairs of values are combined to make bytes, and these bytes are then sent to the shift registers to display the build status.

The final function I'm going to explore in detail is calculateJobStatus:

```
  byte calculateJobStatus(int jobNo, byte toggle) {
    return retrieveJobStatus(jobNo) * (retrieveJobRunning(jobNo) == 0
  ? 1 : toggle);
  }
```

The retrieveJobStatus function reads the number from the status.txt file while the retrieveJobRunning function reads the number from the running.txt file. If the job is not running (retrieveJobRunning returns zero) then the value from retrieveJobStatus is returned. If the job is running then the value returned by retrieveJobStatus is multiplied by the toggle value (one or zero), which is how the blinking effect is created.

## Getting it Running

Having built the daughter board (or put the circuit together on a breadboard) the steps to getting everything running are simple.

The simple prerequisites are that you have a FAT-16 formatted microSD card installed in the Yun, with an empty directory called 'arduino' in the root (this causes the Yun to mount the microSD card at /mnt/sd), and have the Arduino SDK version 1.5.5 or greater (required to support the Yun).

1. Upload the sketch into the Arduino. Without doing anything more you should see the system run through its self test routine.

2. Now download the Linino scripts refresh.sh and setup.sh to /mnt/sd (or on another computer copy them to the microSD card)

3. run the setup.sh script to set up the directory structures and enable the cron daemon.

4. In each job-*n* directory edit the url.txt file, specifying the URL of the job that you want to monitor. This should start with 'http://' and should *not* end in a trailing '/'. You can simply browse to a Hudson or Jenkins job overview page and copy the URL from there, removing the trailing '/' and changing any 'https' prefixes to 'http'.

5. Edit the crontab (crontab -e) and add the following entries:
   ```
   * * * * * /mnt/sd/refresh.sh 0
   * * * * * /mnt/sd/refresh.sh 1
   * * * * * /mnt/sd/refresh.sh 2
   * * * * * /mnt/sd/refresh.sh 3
   ```

6. That's it - now just sit back and watch your traffic lights change colour as your builds change status.

You can run the monitor over a wired ethernet connection or, if it is more convenient, use the WiFi instead.

The Yun plus proto board can be powered by your desktop computer if it has a spare USB 2 socket (you'll need a MicroUSB cable) or alternatively you can use a cheap smartphone charger. If you intend to add more electronic controls (larger LED arrays, relays etc) then you'll need a bigger charger that can provide sufficient current.

## Improvements and Extensions

At some point I will develop some additional shell scripts - to allow the configuration to be changed without manually editing the files, and to enable  or disable monitoring on a build by build basis.

I also have plans to add an audio shield and extend the Arduino code to play sound effects on build events and to announce the daily scrum.

In an ideal world there would be a web interface to support all this configuration, I've not read into the Yun's capabilities to work out if this is really possible or not.

You can't successfully *curl* an https URL from the Yun (you need to change https addresses to http), it would be nice to fix that.

## Purchase Options

You can purchase the build monitor as a kit or pre-assembled, either with or without an Arduino Yun. The software and this document are available on github (github.com/surfsoft/YunBuildMonitor) and will be updated as new features are added.

## Copyright