DESIGNSPARK

The Arduino Yun Mini - Review and application example

Posted by MattVenn on Mon, Jul 06 2015

What is the Yun Mini?

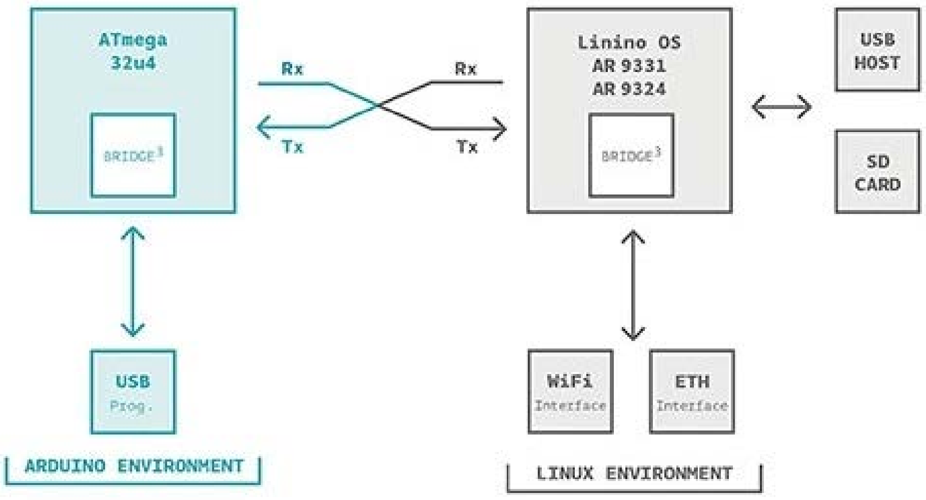
The Yun Mini is a stripped back, bread board friendly version of the Yun, with no SD card, USB host or RJ45 wired internet. These parts can be added later as small addon boards.

It retains the same 400MHz MIPS processor running Linino (a fork of OpenWRT), coupled with the Atmel 32U4 microprocessor (same as on the Leonardo) running at 16MHz.

Linux is installed on 16MB flash with about 6MB left over for your programs. It has 64MB of RAM, which is fairly quickly used up!

There is software support for allowing the 2 devices to communicate in a variety of ways:

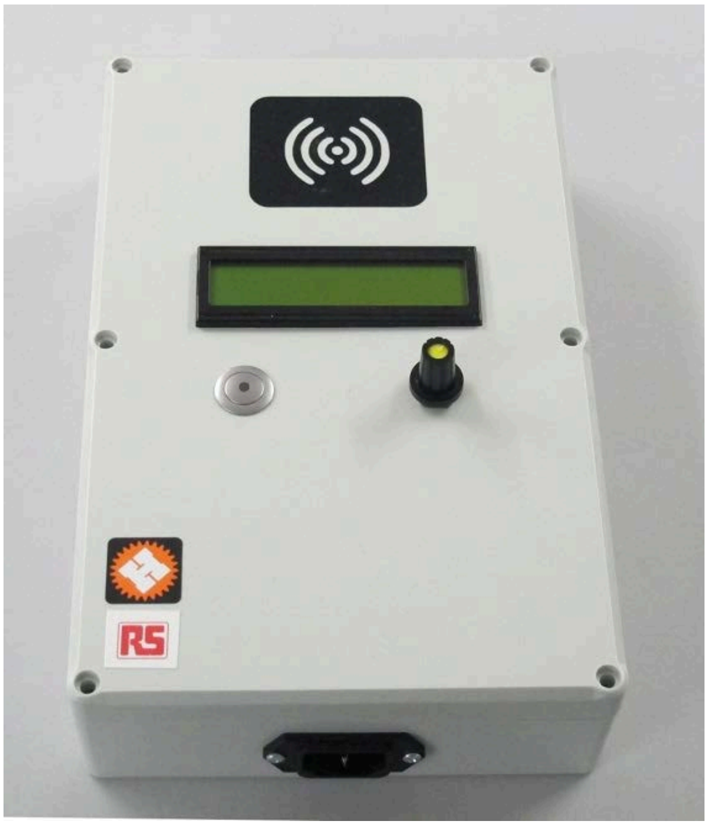
- passing messages with the Mailbox,
- running Linux processes with Process,
- reading and writing files with FileIO.



If you used the Yun, then you'll notice that the password/login screen has changed, the password is now 'doghunter', and the hostname has changed to linino.local.

Another difference is that the Yun Mini is one of the new products from Arduino.org after the recent split from Arduino.cc. I also noticed that there were occasional wifi problems when the SPI bus is being used.

Application: power switch with internet data logging



A system for allowing inducted users to turn power on to various machines.

- Users are authenticated with RFID.
- Usage time is logged to a google spreadsheet.
- Inducted users are managed using the same spreadsheet.
- Tools are turned on and off with wireless mains plugs.

You can download the code, bill of materials, and schematic from the project github

Yun Mini & the internet

Temboo is one of the officially supported services that comes with the Yun. Temboo lets you setup and test the annoying but necessary authentication stuff on their website (so your Temboo account details are used to store all the API keys and secrets of the 3rd party services).

For my application I wanted to fetch all the rows of a spreadsheet, the only way I could find to do this with Temboo was to get columns of data using a json or xml based fetch. Fetching 100 rows of user info ended up being 200k of json! Way too much for a microcontroller like the 32U4 that comes with the Yun.

I made a help request and very quickly got told about adding output filters like this:

```
RetrieveSpecificRowsOrColumnsChoreo.addOutputFilter("users",
"/entry[]/content","Response");
```

Which filters that 200k down to just the bits I need. That sorts the memory issue. All that happens away from the Yun, so we have a much more microcontroller friendly amount of data to process.

By this time I had protoyped the LCD, RFID, encoder and buttons on the 32U4. That was taking 13k (60%) of program space on the Yun.

The Temboo demo for fetching the user names comes out at 13k (45%). So even with the output filtering I don't think the program size is feasible.

So I split the software into 2 parts: user interface and logging/access control. The user interface, radio transmitting and RFID reading will run on the 32U4, which will be nice and responsive. The logging and access control will run on Linux which will have less stringent memory requirements.

After a lot of messing about I finally found a way of doing OAUTH2 with Python and the limited abilities of the Yun: gdata. This is an older google library that has only a few dependencies.

There are 3 times when the system needs to connect to the internet:

- updating cache of users and tools,
- logging unknown rfids,
- logging tool usage when a tool is turned off.

I didn't want the UI to hang while these are happening. Although the Yun bridge library has support for running shell commands asyncronously I decided to do it on the Linux side by splitting the internet side of things into a script that runs in the background.

To help me test the system I put together a useful little script to capture and time stamp the logging from the Arduino part of the Yun.

After a few hours of testing the different functions I started seeing problems where the bridge.py process (part of the Yun system of communication between Linux and the 32U4) was getting killed.

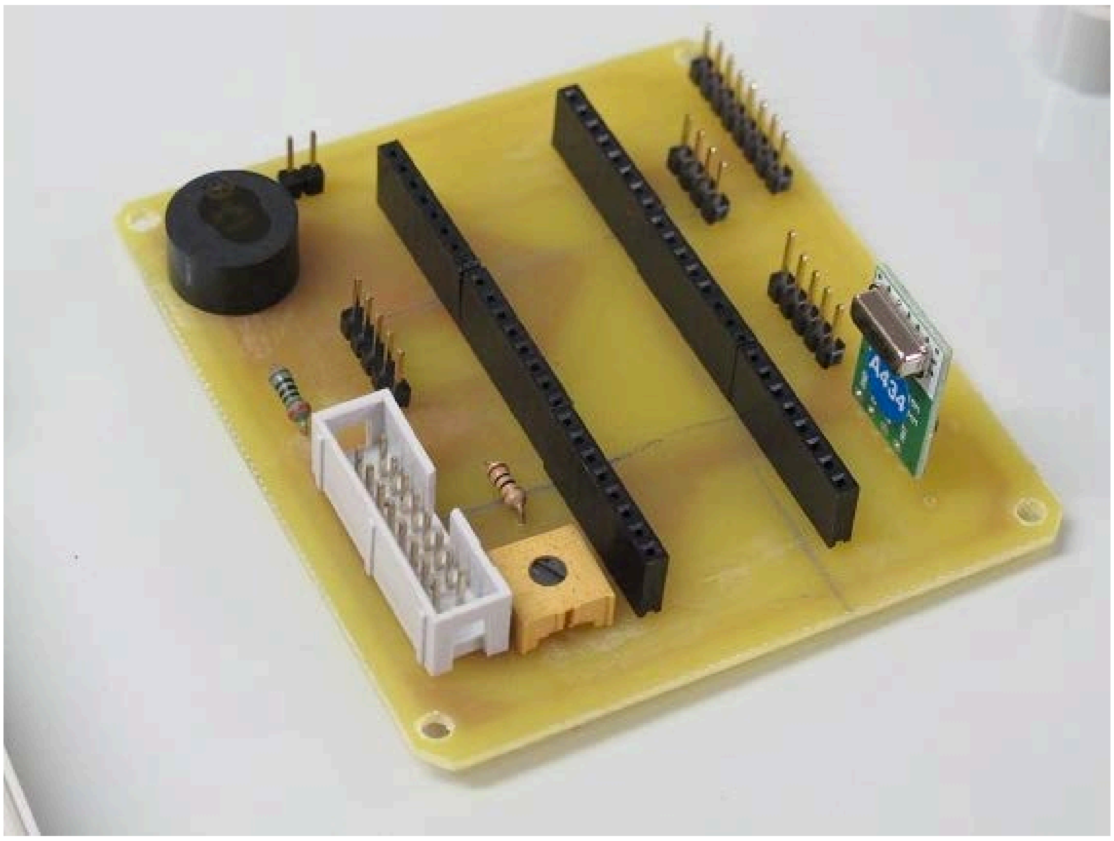
With the logging in place I could see that the crashing was happening after a network timeout on the internet side of things. The test was logging repeated unauthorised RFIDs to the spreadsheet, which would start a new Python process for each test.

As memory ran out, the oom (out of memory) killer started killing big memory hogs, which included not only my program, but also bridge.py (13% of total RAM, oom_score=45). I put a lock on the process so only one could run at once and that fixed the problem.

I think it's a nice example though of how interprocess communication can be harder than it seems at first sight.

Conclusion

I like it that the Yun mini is breadboard friendly, it's easier for me to develop with and then plug into a PCB when I'm ready to finalise a design.



It was the first time I'd used Linino, and it opened my eyes to the power of a small (10MB!) Linux distribution (I run Ubuntu 14 on my laptop and use Raspbian on Raspberry Pis a lot).

IOT projects need to securely connect to cloud services. This can either be done through a 3rd party gateway like Temboo, through your own server, or if you have enough power on board you can go straight to the service. The Yun is powerful enough to do the authentication and encrypted communication on board while leaving the microcontroller to do what it does best; fast, reliable, timing critical operations.

Multiprocess communication is full of hidden traps, and splitting your programs onto 2 processing cores could be harder that it seems at first sight! It certainly was for me! The bridge software exists to make this process easier.

You can download the code, bill of materials, and schematic from the project github

Read this article on DesignSpark

More Arduino Articles in the Arduino Design Centre