1 Attiny Solar Energy Harvest Tests

I have the following:

- Solar panels
- Attiny 10

To this list, I will add a supercap, and an energy harvesting IC. The goal being to load the super cap during the day, and to run 24/7. I will need an exceptionally low power micro. The super cap will need to be about 3.3V or 5V.

1.1 Micro Considerations

The Arduino Atmega328P is not an option. I'm looking to have a current draw of only 1mA max, (ideally 500uA) when active. Moteino is also not an option for this. Those are made for batteries. I want to be battery free. A super cap, however can be used to store energy. I'll get to that shortly.

For micros, I have some Attiny10 on hand, and these have a reasonably low power pull in active mode. Let's build those up first. What will the micro do? No idea. I haven't a clue.

1.1.1 Micro Notes

Must run at 1.8V / 1MHz per front page of data sheet, for 200uA draw in active mode.

1.2 Energy Storage

I don't want a battery. Let's go with a super cap. The solar panels will only be active some of the time, so I will want to harvest

energy with some kind of IC into the cap when the sun is out.[1]

1.3 Make parts, not scrap

I will want to make sure that all parts I build are perf board parts, not breadboard scrap (to be torn down and rebuilt again). This is an Attiny, so no need to test much, yet.

1.4 Programming

To program the Attiny10, I'll use the Arduino adapter from the Junk + Arduino blog. I built it up[2], and was able to Read the memory. In order to upload to the board, you will need a compiler setup. You can possibly do it in AVRGCC, but instead I opted for either Arduino IDE (via Attiny10Core which didn't work), and then went to Mplab. In order for mplab 5.25 to work, it will need XC8 compiler, and there is a pack that can be downloaded through the IDE to get Attiny10 support.

It appears the AVR Dragon (which I have) can not be used. However, other programmers can be used. Pickit 4, Mkavrii, stk600, I think.

1.4.1 Testing Arduino Loader

Tested this with the blink_LED.c in code folder. The code is as simple as possible. It is the following:

```
//#include <xc.h>
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
```

[1] Reference: www.analog.com/media/en/technical-documentation/technical-articles/solarenergyharvesting.pdf is a start. I'll need to do more research.

[2] Had slight error where the Arduino + board wouldn't read - pins too short on headers, then the arduino wouldn't boot - due to bad connection on perf board shield. Thankfully, the USB port didn't try to run. Protection circuitry cut in on the laptop.

```
    // PB2 output
    DDRB = 1<<2;

    while(1)
    {
        // Toggle PB2
        PINB = 1<<2;
        _delay_ms(500);
    }
}
```

When programmed in Mplab, with XC8 compiler, and Attiny10 support, I get the following hex output:

```
:100000000AC020C01FC01EC01DC01CC01BC01AC01B
:1000100019C018C017C011271FBFCFE5D0E0DEBF41
:0A002000CDBF03D000C0F894FFCF5D
:10002A0044E041B940B95FE966E871E05150604087
:0A003A007040E1F700C00000F5CFB0
:02004400DDCF0E
:00000001FF
```

The content of this hex isn't the focus of this passage. Instead, I want you to review the results of a D for Dump Memory, by the Arduino Loader.

```
Current memory state:

registers, SRAM
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
0000: 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 03 00 00 79 00 03 00 00 00 00
0040: B7 AD AE FA 58 70 63 6B FB 5A B4 1B FF FF 35 3F
0050: 67 D7 33 43 DF 5F FB 72 C9 7D FE E9 9D C5 00 12
NVM lock
```

```
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
3F00: FF FF
configuration
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
3F40: FF FF
calibration
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
3F80: 79 FF
device ID
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
3FC0: 1E 90 03 FF
program
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
4000: 0A C0 20 C0 1F C0 1E C0 1D C0 1C C0 1B C0 1A C0
4010: 19 C0 18 C0 17 C0 11 27 1F BF CF E5 D0 E0 DE BF
4020: CD BF 03 D0 00 C0 F8 94 FF CF 44 E0 41 B9 40 B9
4030: 5F E9 66 E8 71 E0 51 50 60 40 70 40 E1 F7 00 C0
4040: 00 00 F5 CF DD CF FF FF FF FF FF FF FF FF FF FF
4050: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4060: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
(...some memory omitted here for brevity...)
43E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
43F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Notice that the "AC020C01F" is set. That is from the hex. But the 01000... before it seems to be missing. Some deciphering of how the Arduino programs the Attiny is in order here. It also doesn't end the same.

Regardless, when programming, the Arduino reports 70 bytes written, and likewise in the Mplab project memorymap.xml file, it also notes 70 bytes for the sketch. This lines up.[3]

The blinking LED works. Let's move on.

[3]Although for an unknown reason, every command registers twice on the Arduino serial monitor, but this appears to be harmless.

1.4.2 IO Port Switching Speed

Using the above code without any delay_ms, I get the following results from a default clock speed, and a 128KHz clock speed. This test was done to confirm that I could change the clock with

```
//Write CCP
CCP = 0xD8;
//change CLK to 128KHz
CLKMSR = 0b01;
```

There was no issue.

```
Default CLK (8MHz? or 1MHz?): 160KHz IO Switch
128KHz CLK: 2.5KHz IO Switch
```

I am going to pursue 128KHz for starters, for lower current dissipation. Note that with the Arduino loader, it is cumbersome to test and change code as you move along. It is therefore going to be necessary to use a programmer, with a dedicated header on board.

1.4.3 VCC 1.8V

The lowest power supported: 1.8V can be applied, without any configuration needed. It does not affect IO switching speed (although obviously amplitude is affected).

```
128KHz CLK (5.0V): 2.5256  KHz IO switch
128KHz CLK (3.3V): 2.5477  KHz IO switch
128KHz CLK (1.8V): 2.5849  KHz IO switch
```

As voltage drops, IO increases.

VCC Dropout voltage:
From 1.5, it drops out at 1.248V or so. Comes back at about 1.34V

Test size of 1.

Can't run this with one (AA) battery, but you could with 2.

Current Draw: 128KHz - IO test, 1.8V, 0.08mA (78uA) (tested w/3478A)

1.5 Application

First, I need a board for these and a programmer, to quickly program. Second, I need an application. I want extremely low power. Hopefully, solar with no batteries, to start. This is extremely low - that is the point.

Given the power requirements put me under 1mA (with my current panels), I'm considering the following: EEPROMs would require SPI protocol. Doable, but overcomplicated for now.

Eink (need to find a small and cheap enough option. So far, they have either too many pins, and/or use too much current. Something like what stores use to display prices would work, but that doesn't get the data out, only makes it readable.),

Third option would be RF. That is a viable path, but not today. Let's skip that for now.

Fourth option that comes to mind is IR. IR diodes, as in TV remotes, would work well here. I am choosing this as the first project. I will have dumb clients, that consist of - Attiny / IR / Sensor powered by solar. I will have a BBB that receives the IR data, and does all intelligent data gathering. To keep things simple, the IR will be binary ADC data, or otherwise sensor numbers. No SPI, no protocol complexity. That would require space on the Attiny.

Let's build some boards based on the above.

For sensors: While building, I came across an option. Hall effect sensors. I think also capacitive sensors can be used. This may find a use in a gate sensor, for when a driveway gate is opened or closed.

With a small battery, it would work for years.

Footprints: I had to make a footprint for this module on board package for one sensor. The solution to get footprints right? copy graphic image and make it into silkscreen on the board. Easy.