Beaglebone: The Perfect Telemetry Platform?

Matt Ranostay
E-ELC 2012
Ranostay Industries
http://ranostay.org
mranostay@gmail.com
What is the Beaglebone?

- Low cost AM335x SoC platform
  - $89 from various source (Mouser, Digikey, Amazon)
- 3.3v I/O can interface with many ICs and sensors out of the box, or with cheap 3.3v to 5v logic converter.
  - I2C
  - SPI
  - GPIO
- 1.8v I/O Analog In (ADC) pins.
Introduction to Capes

- Daughter cards that connect to the expansion headers
- Examples of ones that currently exists and provide addition functionality
  - Audio Cape
  - DVI Cape
  - Camera Cape
  - Weather Cape
- Expansion header allows easy breadboard access
Starting a Beaglebone Telemetry Project

- What to design?
- How to design it?
  - Fritzing – Open Source circuit design
  - More hobbyist + breadboard friendly
  - Eagle PCB – Not free but affordable for hobbyists
    - Larger learning curve + schematic first designing.
- Materials and hardware skills required
  - Soldering Iron + Minimal electrical engineering knowledge
- Software skills
  - Any language for reporting
Practical Applications

- Weather Reporting Station
- Radiation Monitoring
- Earthquake Detection Mesh Network
- Home Security System
- Entropy Pool Generation
What kind of data can we report?

- Barometric
- Temperature
- Radiation Exposure (Counts Per Minute)
- Earthquakes
- GPS + Orientation + Compass Heading
- Ambient Light
How can we share data?

- **Cosm (Formerly Pachube)**
  - Free for typical usage.
- Allows reporting of almost any sensor possible visa feeds
- Simple JSON or EEML interface
- Handles graphing all datasets and points
  - Allows settings triggers when thresholds are peaked (Twitter or HTTP POST)
def read_bmp085_pressure():
    f = open("/sys/bus/i2c/drivers/bmp085/3-0077/pressure0_input")
    return "%.2f" % (int(f.read().strip()) / 100.0)

....

def start_reporting(pac, w1_serial_id):
    while True:
        if w1_serial_id:
            pac.update([eeml.Data("w1-temp", read_w1_temp(w1_serial_id))])
            bmp085_pressure = read_bmp085_pressure()
            pac.update([eeml.Data("bmp085-temp", read_bmp085_temp())])
            pac.update([eeml.Data("bmp085-pressure", bmp085_pressure)])
            pac.update([eeml.Data("sht21-humidity", read_sht21_humidity())])
            pac.update([eeml.Data("sht21-temp", read_sht21_temp())])
            pac.update([eeml.Data("tsl2250-lux", read_tsl2550_lux_value())])
            pac.put()
            sleep(1)

if __name__ == "__main__":
    API_KEY = get_env_value("COSM_KEY")
    ...
    API_URL = "'/v2/feeds/%s.xml" % feed
    ...
    pac = eeml.Cosm(API_URL, API_KEY, use_https = False)
    start_reporting(pac, w1_serial_id)
Sample COSM Data

<eeml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.eeml.org/xsd/0.5.1" xsi:schemaLocation="http://www.eeml.org/xsd/0.5.1 http://www.eeml.org/xsd/0.5.1/0.5.1.xsd" version="0.5.1">
  <environment>
    <data id="w1-temp"><current_value>25.00</current_value></data>
    <data id="bmp085-pressure"><current_value>1004.00</current_value></data>
    <data id="sht21-humidity"><current_value>39.74</current_value></data>
    <data id="tsl2250-lux"><current_value>0</current_value></data>
    <data id="sht21-temp"><current_value>26.45</current_value></data>
    <data id="bmp085-temp"><current_value>25.50</current_value></data>
  </environment>
</eeml>
Developing Telemetry Platform

• How are you going to place everything?
  • SMD, THT, etc

• Cost of sensors and how important accuracy and precision are.
  • You get what you pay for…

• Sensors interfaces to use
  • I2C, SPI, Analog In, etc, etc.

• Pins to use and what other capes you many want to use.
Cape Software Design

- EEPROM identification
- Little to no user intervention required to configure
- Device tree initialization
  - Capebus patchset handles cape setup and definition
  - Conflict detection
- Sysfs run-time changes whenever possible
- Actual working software before calling the cape project complete.
  - U-boot demo apps don’t count…..
&bone_geiger_cape {
    board-name = "Geiger Cape";

    pinctrl-names = "default";
    pinctrl-0 = \&bone_geiger_cape_pins>;

    pwms = \&ehrpwm1 0 500000 0>;
    pwm-names = "bone-geiger-cape";

    pwm-frequency = <20000>;  /* 20KHz */
    pwm-duty-cycle = <60>;  /* 60% */

    event-blink-delay = <30>;  /* 30ms */

    gpios = \&gpio3 17 0>;

... Continued on next page ...
Capebus + Device Tree

... 

gpio-leds {
    compatible = "gpio-leds";
    pinctrl-names = "default";
    pinctrl-0 = <&bone_geiger_cape_lcd_pins>;

    geiger-led0 {
        label = "geiger:green:usr0";
        gpios = <&gpio2 23 0>;
        linux,default-trigger = "geiger-run";
        default-state = "off";
    };

    geiger-led1 {
        label = "geiger:red:usr1";
        gpios = <&gpio2 25 0>;
        linux,default-trigger = "geiger-event";
        default-state = "off";
    };
};
}();
Capebus (Working Example)

Confirm Capebus picked up device

```
# cd /sys/bus/capebus/devices/bone-0\:3
# cat id
Geiger Cape,00A0,Geiger Inc,
```

(start running)

```
# echo 1 > run
```

(power LED turns on and the event LED lights up on a “click”)

Display counts:

```
# cat counter
4344
```

Display VSENSE (voltage feedback loop) in Millivolts:

```
# cat vsense
538004
```
Geiger Counter Project

- Reason for picking this project
  - Simple + Fun
  - Practical purpose
  - Nature gives a perfect test source since atoms are always decaying.
  - Excuse to test out various consumer and scientific items for radioactivity.
Geiger Counter Continued

- Data points provided are simple “clicks” in time
- All the magic happens in how you report and display the data
- Remote stations need to take into account for power usage
  - CPUFreq ‘powersave’ governor e.g. `cpufreq-set -g powersave`
  - Adjust sample update rate
  - Offload any data processing upstream off the device.
Subsystems Used

- Beaglebone
- Device Tree + Capebus
- Pin Control
- Weather Cape
- Geiger Cape
- I2C
- One Wire
- GPIO
- PWM
Geiger Counter Reporting Flow

Radiation Event → GPIO Interrupt Event
~1 millisecond pulse → COSM Reporting → COSM Graphing + Alerts
### Table 11. Expansion Header P9 Pinout

<table>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>PIN</th>
<th>CONN</th>
<th>PIN</th>
<th>SIGNAL NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GND</td>
<td>1</td>
<td>2</td>
<td>GND</td>
</tr>
<tr>
<td>VDD_3V3EXP</td>
<td>3</td>
<td>4</td>
<td>VDD_3V3EXP</td>
<td></td>
</tr>
<tr>
<td>VDD_5V</td>
<td>5</td>
<td>6</td>
<td>VDD_5V</td>
<td></td>
</tr>
<tr>
<td>SYS_5V</td>
<td>7</td>
<td>8</td>
<td>SYS_5V</td>
<td></td>
</tr>
<tr>
<td>PWR_BUT*</td>
<td>9</td>
<td>10</td>
<td>A10</td>
<td>SYS_RESETn</td>
</tr>
<tr>
<td>UART4_RXD</td>
<td>T17</td>
<td>11</td>
<td>12</td>
<td>U18</td>
</tr>
<tr>
<td>UART4_TXD</td>
<td>U17</td>
<td>13</td>
<td>14</td>
<td>U14</td>
</tr>
<tr>
<td>GPIO1_16</td>
<td>R13</td>
<td>15</td>
<td>16</td>
<td>T14</td>
</tr>
<tr>
<td>I2C1_SCL</td>
<td>A16</td>
<td>17</td>
<td>18</td>
<td>B16</td>
</tr>
<tr>
<td>I2C2_SCL</td>
<td>D17</td>
<td>19</td>
<td>20</td>
<td>D18</td>
</tr>
<tr>
<td>UART2_TXD</td>
<td>B17</td>
<td>21</td>
<td>22</td>
<td>A17</td>
</tr>
<tr>
<td>GPIO1_17</td>
<td>V14</td>
<td>23</td>
<td>24</td>
<td>D15</td>
</tr>
<tr>
<td>GPIO3_21</td>
<td>A14</td>
<td>25</td>
<td>26</td>
<td>D16</td>
</tr>
<tr>
<td>GPIO3_19</td>
<td>C13</td>
<td>27</td>
<td>28</td>
<td>C12</td>
</tr>
<tr>
<td>SPI1_D0</td>
<td>B13</td>
<td>29</td>
<td>30</td>
<td>D12</td>
</tr>
<tr>
<td>SPI1_SCLK</td>
<td>A13</td>
<td>31</td>
<td>32</td>
<td>VDD_ADC(1.8V)</td>
</tr>
<tr>
<td>AIN4</td>
<td>C8</td>
<td>33</td>
<td>34</td>
<td>GND_AIN6</td>
</tr>
<tr>
<td>AIN6</td>
<td>A5</td>
<td>35</td>
<td>36</td>
<td>A5</td>
</tr>
<tr>
<td>AIN2</td>
<td>B7</td>
<td>37</td>
<td>38</td>
<td>A7</td>
</tr>
<tr>
<td>AIN0</td>
<td>B6</td>
<td>39</td>
<td>40</td>
<td>C7</td>
</tr>
<tr>
<td>CLKOUT2</td>
<td>D14</td>
<td>41</td>
<td>42</td>
<td>C18</td>
</tr>
<tr>
<td>GND</td>
<td>43</td>
<td>44</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>45</td>
<td>46</td>
<td>GND</td>
<td></td>
</tr>
</tbody>
</table>

- Green – Ground
- Red – Power Supplies
- Blue – PWM
- Purple- I2C
- Black – Analog In
Sensor Selection (Geiger Counter)

- Geiger tubes have various features of quality
  - Dead time
  - Sensitivity
  - Type of radiation it can detect
    - LND-712 was selected since it can detect all three main types of radiation (alpha + gamma + beta). Most expensive tube of its type.
    - SBT-9 (Soviet-Era tube) was a close second but not as sensitive or available in North America.

- Voltage needed to register counts vary greatly
  - LND-712 – 500 volts
  - SBT-9 (and most Soviet tubes) – 300 to 400 volts.
Geiger Counter Circuit

- Tube is a large capacitor that gets charged up, and when hit by ionizing radiation causes an “overcharge”. Excess charge has to go somewhere and that causes a “click”.

- Logic level and pulse shifter design
  - Extends a short microsecond pulse to ~1 millisecond
High voltage is sourced to the tube. Current boost converter can provide about 600V
- Although almost no current is flowing. It still can be dangerous...
- Analog-in pin used for feedback loop to measure voltage.
- Keep HV traces short as possible

MOSFET can be asserted on at 100% which is very bad.

To resolve an accident initial state add a I2C GPIO expander to allow toggling of power state to cape. Never trust expansion header pin states..

Do use a current limiter circuit (INA138 for example) to force system reset or device shutoff.
E-ELC Demo Setup (Geiger Counter)

- Geiger Cape (Prototype Rev) + LND-712
- 1.8” Adafruit TFT SPI interface display
- Beaglebone Rev A6
Weather + Radiation Station

- Geiger Cape (Prototype Rev) + SBT-9
- CircuitCo Weather Cape
- Beaglebone Rev A6
Weather + Radiation Station Continued
PCB Design
Lessons Learned

- Know what functionality your device has and use it.
- Avoid bitbanging interfaces that already available.
- Microcontroller Unit != Microprocessor Unit treat it as such.
- Watch initial states
- Example of a bad states would be a PWM pin that is shared with a GPIO that gets asserted on…
- Watch GPIO states on pins…
- Capebus will not save you every-time…
Lessons Learned Continued

- Watch the logic voltage level
- Watch the pin muxing.
- Really easy to conflict with another cape.
  - Device tree holds the hope for the future
  - Possible damage.
- Test on breadboard first
- Analog In (ADC) safety. Absolute 1.8v LIMIT!
  - Invest in some 1.8V Zener diodes, especially if you use sensors that may output higher voltage.
  - For THT applications two standard rectifier diodes will give you a equivalent 1.4V Zener diode
- Show Demo! No magic smoke, no whammy!
References

• Beaglebone SRM
  • http://beagleboard.org/static/beaglebone/latest/Docs/Hardware/BONE_SR M.pdf

• Cadsoft Eagle PCB
  • http://www.cadsoftusa.com

• Adafruit Eagle Library (Beaglebone Cape Part)
  • https://github.com/adafruit/Adafruit-Eagle-Library

• Fritzing
  • http://fritzing.org

• Fritzling Parts (Beaglebone Cape Part)
  • https://github.com/ohporter/fritzing-parts
Cupertino (Indoors) Radiation + Weather
https://cosm.com/feeds/73056

Slides + Demo Source Code
https://github.com/mranostay/beaglebone-telemetry-presentation
https://github.com/mranostay/cosm-analog

Thanks to other Geiger Cape team members
Dimitris Sapountzakis (Hardware)
Pantelis Antoniou (Software + Capebus patchset)
Koen Kooi (Design Advice)
References Continued

- Capebus patchset branch
  - https://github.com/koenkooi/linux/tree/3.7-for-panto-rebase

- LND-712 End Window Alpha + Beta + Gamma detector datasheet

- SBT-9 Soviet-era Alpha + Beta + Gamma tube
  - http://gstube.com/data/3004/ (Russian)