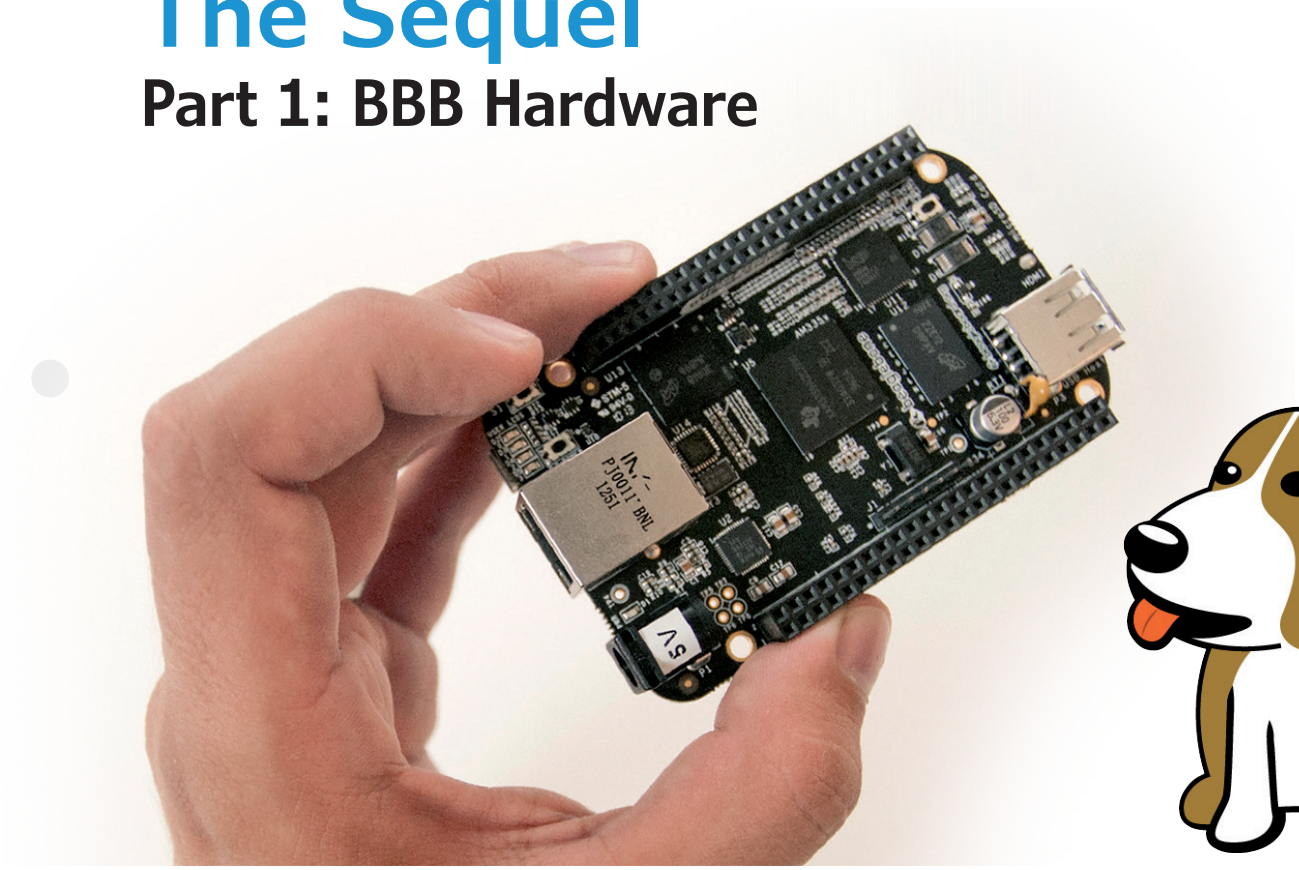
BeagleBone Black, The Sequel

Part 1: BBB Hardware



By **Tony Dixon** (UK)

If you've already got a Raspberry Pi and need more I/O or you've got an Arduino Due and want more processing speed then the BeagleBone Black may be what you've been looking for. In this our first **.**Post on the BeagleBone Black (or just BBB) we'll be looking at its hardware expansion capabilities. After we looked at its hardware we'll write ourselves the traditional "Blinky" LED program as our first program.

Just in case you missed Thijs Beckers' introduction to the BBB in Elektor's December 2013 edition [1], **Table 1** has a quick summary of its capabilities. By default the BBB comes preinstalled with the Ångström Linux distribution. BBB's doghouse it at [2].

Table 1. BeagleBone Black Summary.

CPU	Sitara AM3359AZC100 ARM Cortex-A8 from TI
CLOCK	1 GHz
RAM	512 MB DD3 SDRAM
VIDEO	uHDMI
STORAGE	2 GB eMMC (on-board), micro SD-CARD
PORTS	Ethernet 10/100Mb, USB Host, USB Client
I/O	GPIO x65, Analog x7, PWM x8, UART x4.5, I2C x2, SPI x2
COST	$45

Enough hardware to throw a stick at

The BBB is seriously equipped for your hardware projects. It has a doggy bag full of GPIO, analogue, PWM and serial interfaces. The BBB has two expansion headers, P8 (Expansion

B) and P9 (Expansion A) with each header is a 46-pin hobbyist friendly 0.1" pitch female header. **Table 2** shows the pins outs of the expansion headers after power up. Other signals can be allocated to the pins, please refer to the *BBB System Reference Manual* for signal mux options.
The BBB I/O are 3.3V only signals so we should avoid connecting them to any 5-V circuits unless we want to send your BBB to the great kennel in the sky.

Software ~~Warehouse~~ Dog House

As the BBB is a Linux computer we have a choice of programming languages we can use. As well as favorites such as C/C++ and Python, the BBB has its own language BoneScript. BoneScript was first found on the BBB's older brothers, BeagleBoard, BeagleBoard-XM and the original BeagleBone.
BoneScript is a Node.js based library, which features many familiar Arduino-like function calls to interact with the BBB hardware. BoneScript is based on JavaScript, and like Arduino there is an IDE called Cloud9 (hey!) you can use to create your programs.

Blinky _.-._.-._

For our first BBB program we'll follow the embedded systems tradition of flashing/blinking an LED. We could have used BoneScript to program our example but instead we'll stay with the familiarity of C/C++. For our Blinky program we'll connect a LED through a 680-ohm resistor to GPIO1_6 on connector P8.03.
On the BBB, GPIO are controlled in blocks of 32 with the blocks indexed starting from 0. To calculate the GPIO number we multiply the block number by 32 and add the signal number, so for our example GPIO1_6 is 1*32 + 6 = 38.
In simplistic terms Linux treats almost everything as a file, including hardware ports such UARTS and USB. Because of this feature, a programmer can also access GPIO as if it was a file descriptor through the Linux ker-

Table 2. BeagleBone Black Expansion Pinouts; P8, P9.

SIGNAL	P8			SIGNAL
GND	1		2	GND
GPIO1_6	3		4	GPIO1_7
GPIO1_2	5		6	GPIO1_3
TIMER4	7		8	TIMER7
TIMER5	9		10	TIMER6
GPIO1_13	11		12	GPIO1_12
EHRPWM2B	13		14	GPIO2_26
GPIO1_15	15		16	GPIO1_14
GPIO0_27	17		18	GPIO2_1
EHRPWM2A	19		20	GPIO1_31
GPIO1_30	21		22	GPIO1_5
GPIO1_4	23		24	GPIO1_1
GPIO1_0	25		26	GPIO1_29
GPIO2_22	27		28	GPIO2_24
GPIO2_23	29		30	GPIO2_25
UART5_CTS	31		32	UART5_RTS
UART4_RTS	33		34	UART3_RTS
UART4_CTS	35		36	UART3_CTS
UART5_TXD	37		38	UART5_RXD
GPIO2_12	39		40	GPIO2_13
GPIO2_10	41		42	GPIO2_11
GPIO2_08	43		44	GPIO2_09
GPIO2_6	45		46	GPIO2_07

SIGNAL	P9			SIGNAL
GND	1		2	GND
3.3V	3		4	3.3V
5V	5		6	5V
5V_SYS	7		8	5V_SYS
PWR_BUTTON	9		10	SYS_RESET
UART4_RXD	11		12	GPIO1_28
GPIO4_TXD	13		14	EHRPWM1A
GPIO1_16	15		16	EHRPWM1B
I2C1_SCL	17		18	I2C1_SDA
I2C2_SCL	19		20	I2C2_SDA
UART2_TXD	21		22	UART2_RXD
GPIO1_17	23		24	UART1_TXD
GPIO3_21	25		26	UART1_RXD
GPIO3_19	27		28	SPI1_CS0
SPI1_D0	29		30	SPI1_D1
SPI1_SCLK	31		32	AVCC
AIN4	33		34	AGND
AIN6	35		36	AIN5
AIN2	37		38	AIN3
AIN0	39		40	AIN1
GPIO_20	41		42	GPIO_7
GND	43		44	GND
GND	45		46	GND

nel. We can us the following file descriptors to access the GPIO:

```
/sys/class/gpio/export
/sys/class/gpio/gpio38/direction
/sys/class/gpio/gpio38/value
/sys/class/gpio/unexport
```

First we'll start a terminal session and then start the **nano** editor. In the terminal type:

```
nano blinky.cpp
```

Write or copy the code shown in the **Listing 1** appended at the end of this article. Once finished, save the program by pressing Ctrl+X, Y and ENTER to confirm saving the program.
Once saved, in our terminal we can compile the C/C++ program by typing:

```
g++ blinky.cpp -o blinky
```

Once compiled if we've had no compilation errors we can run our program by typing:

```
./blinky
```

We should see our LED flashing on and off at a leisurely once a second. As you can see, "Beware of the Dog" is not terribly fitting in case of the BeagleBone Black.

(130472)

Weblinks

[1] Enter BeagleBone Black, Elektor December 2013, www.elektor-magazine.com/130279

[2] Beagle Website: http://beagleboard.org

Listing 1. blinky.cpp

```
#include <stdio.h>
#include <unistd.h>

using namespace std;

int main() {
FILE *export_file = NULL;
FILE *IO_dir = NULL;
char str_low[] = "low";
char str_high[] = "high";
char str_port[] = "38";

// Open Port
export_file = fopen ("/sys/class/gpio/export", "w");
fwrite (str_port, 1, sizeof(str_port), export_file);
fclose (export_file);

  while (1) {
          IO_dir = fopen ("/sys/class/gpio/gpio38/direction", "w");
          fwrite (str_high, 1, sizeof(str_high), IO_dir);   // pin = HIGH
          fclose (IO_dir);
          sleep (1);

          IO_dir = fopen ("/sys/class/gpio/gpio38/direction", "w");
          fwrite (str_low, 1, sizeof(str_low), IO_dir);   //pin = LOW
          fclose (IO_dir);
          sleep (1)
  }

}
```