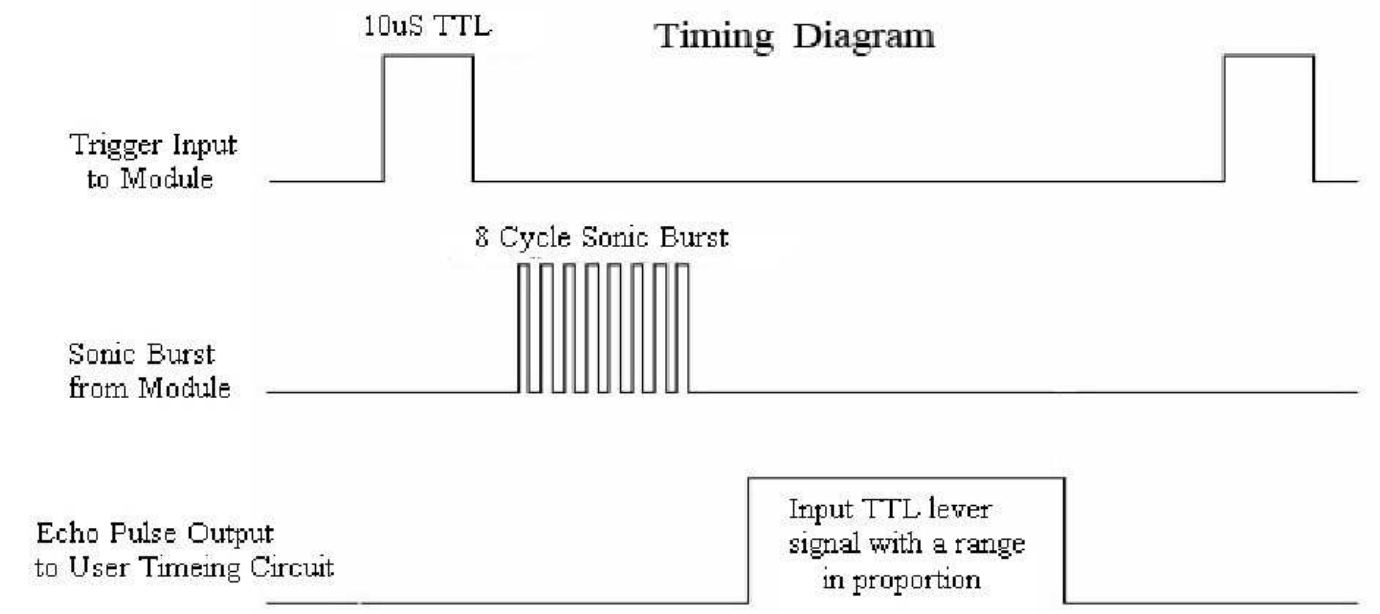
Date of creation: January 2015 -July 2015 and Attribution: Prasad N R as a representative of (unregistered) company TraQuad.

HC-SR04 is being used as it is cheap, reliable and effective in measuring distances accurately. Ultrasonics serve the purpose of obstacle avoidance in TraQuad. It is a safety system which can prevent crashing and accidental damage to people and surroundings.

This one is being chosen for it's cost. It costs about 150Rs. Ultrasonic sensors work with the principle of Time-of-Flight (TOF). It reflects the time of the sound pulse propagation by reflecting the same with a voltage pulse width. It can measure upto 5 meters with errors of just 2cms! This can justify the cost-effectiveness of TraQuad.



(**Image Credits: http://www.micropik.com/PDF/HCSR04.pdf**)

This module consumes about 15mA at-most and the results are very reliable (unless the atmosphere is humid because of severe-inclement weather like rains). We have tested the results and it has also been tested by several other people using oscilloscopes.

The module works as shown above. It expects a TTL pulse of 10us (just one digital pulse of 10us) and the eight 40kHz pulses are 'internal to the module'. Unfortunately, Beaglebone uses Angstrom distro by default (Debian distribution) which takes about 10ms to toggle the states of GPIO. The time-of-flight is reflected through the time-duration of the echo pulse.

So, the pulse duration varies between 1ms to 25ms (reliable duration) and 38ms for virtual infinity (in-case no obstacle is detected). The output and input are of active-high type.

Hence, voltage-divider bias is being used to convert 5V logic to 3.3V logic approximately. Resistance values are carefully chosen so as to ensure that the pins are not burnt. Beaglebone Black supports a maximum current of 5mA approximately (4mA- 6mA).

If the resistance values are low, pins can be burnt easily while higher resistances can cause interference because of lack of signal strength (It acts like open switch).

After connecting these, another problem had to be tackled- the problem of slower switching speeds of GPIOs of Beaglebone Black. There are three possible methods for the achievement of this objective.

Method 1:

```
const char *LEDBrightness0="/sys/class/leds/beaglebone:green:usr0/brightness";

if((LEDHandle = fopen(LEDBrightness0, "r+")) != NULL){
            fwrite("1", sizeof(char), 1, LEDHandle);
            fclose(LEDHandle);
    }
```

Method 2:

```
gpio_output = libsoc_gpio_request(GPIO_OUTPUT, LS_SHARED);
libsoc_gpio_set_direction(gpio_output, OUTPUT);
libsoc_gpio_set_level (gpio_output , HIGH );
```

Method 3: PRU

In the first method it is observable that the software is polling the file-system at two instances. This greatly slows down system performance. Firstly, the system is polling the GPIO file to check if the file is being accessed by some other software or if it is safe to modify the file. Then, the write process begins by opening the file and writing 1 to the corresponding register and confirming that 1 has been written (and polling the system).

(Note: Averaging has been used (in the attempt of achieving tri-mean filtering) and it can be commented out in the code)

The Libsoc pin-mapping is as shown below:

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_115	27	28	GPIO_113
GPIO_111	29	30	GPIO_112
GPIO_110	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

(**Image credits:** http://beagleboard.org/support/bone101)

In the second method, Libsoc library by-passes these processes and writes to the registers directly.

```
if (file_write (current_gpio->value_fd, gpio_level_strings[level], 1) < 0)
  return EXIT_FAILURE;
```

This implements the function entirely. This function just checks if the file can be written with new values. (Unlike the check for use by another software or so) Then, it directly writes the values to corresponding registers. This method has effectively reduced the time of switching from 10ms to 3.5us. Errors have reduced from 1 metre to a meagre 1 cm. But, there were problems regarding the kernel because, this code was working by directly writing to registers while the kernel expects file-system methodology. So, the timers of the Linux system are maintained by the kernel. So, this process was not getting detected by the timer. This has been resolved by using a counter. Now, there was another problem. Kernel was terminating the process whenever it detected the register-access. So, one if condition was able to hold only one statement. We have tested that the kernel sometimes terminate (very rarely) this process when it is fast and get stuck (again, very rarely) when it was loaded with too many processes. So, it was allowing the execution of two-statements. But, the time-struct type was causing errors and returning 0. Now, this has been resolved very effectively.

PRU has been eliminated primarily to avoid assembly-specific programming and to have generic codes. (PRU is available only on Beaglebone Black while second method can be run on any Linux-based system like Raspberry-pi etc. Libsoc is optimised for Beaglebone Black; But, it can be used for any SoC (System on Chip))

Optimisations can make this faster by a 100 times. But, the current level of optimisation is sufficient for the application.