ESLab HW6

游祖鈞 B06209040

November 22, 2021

Reports with several issues and solutions presented.

1 general approach

We observed the DHT11 sensor data, humidity and temperature, with logic analyzer and corresponded to the output data from Python.

1.1 Observe the DHT11 signal with logic analyzer

Paying attention to the lower half panel is waveform overview, we see a long pull down. As in courses, this is to be detected by DHT11, figure 1. Its duration is about 20ms and greater than 18ms.

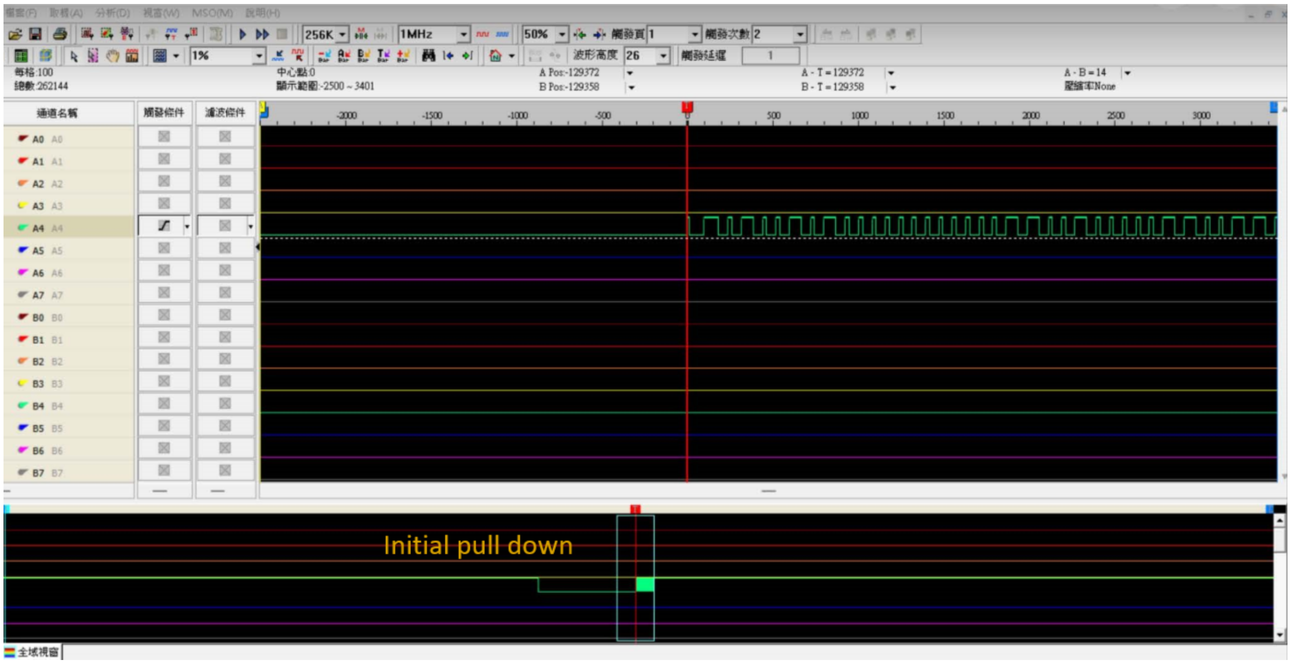

256K 1MHz 50% 觸發頁 1 觸發次數 2
1% 波形高度 26 觸發延遲 1
每格:100 總數:262144
中心點:0 顯示範圍:-2500 ~ 3401
A Pos:-129372 B Pos:-129358
A - T = 129372 B - T = 129358
A - B = 14 壓縮率None

Figure 1: Initial pull down

After the pulldown from MCU, We recognized all the signal pattern and that the observed data transmission is

01001010 00000000 00011001 00000110 01101001

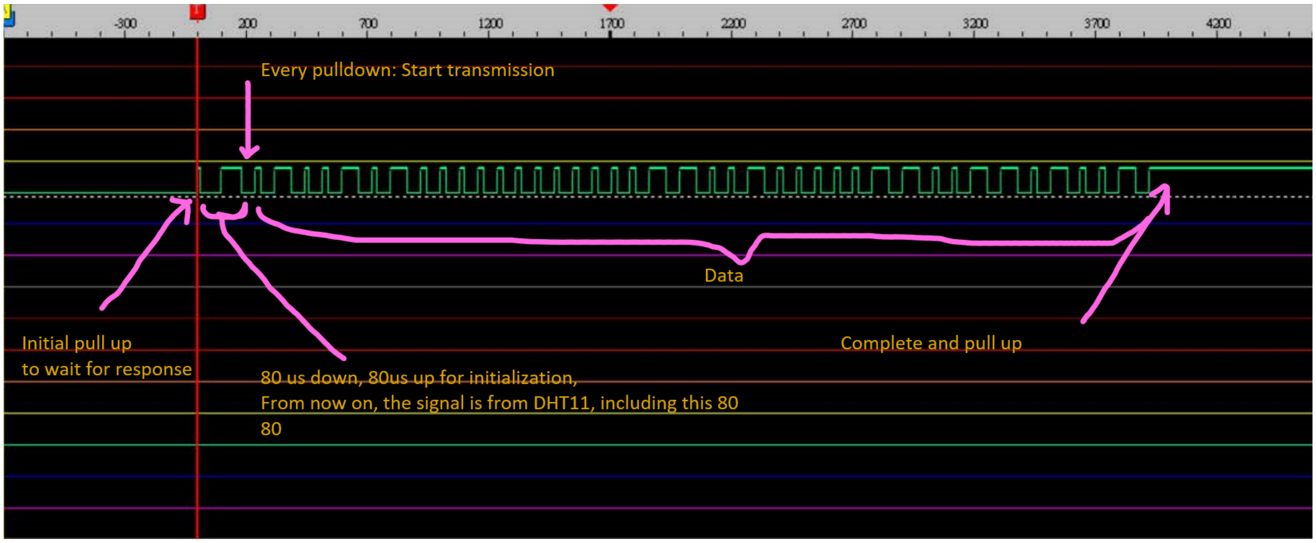


Figure 2: Signal

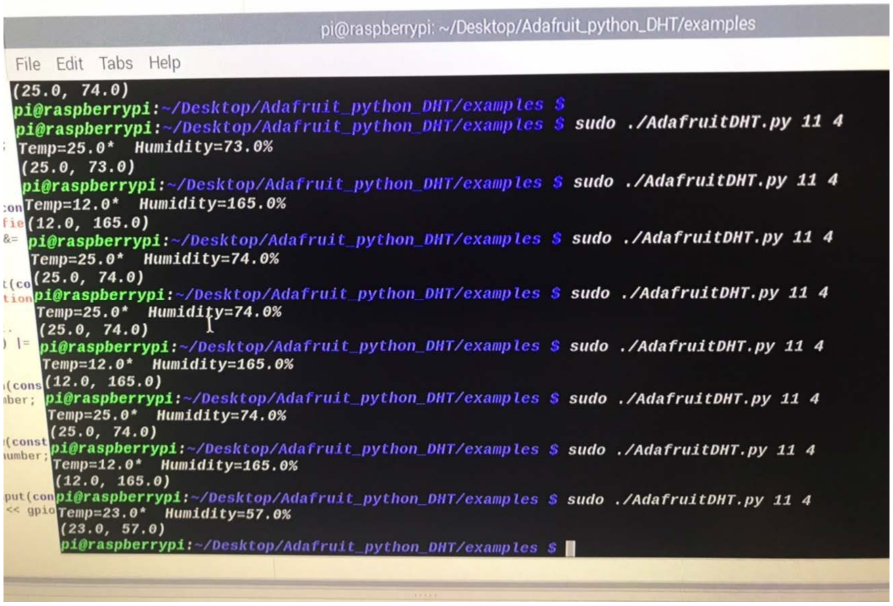
1.2 Signal sum

01001010 00000000 00011001 00000110 01101001

is translated as

$$74\ 0\ 25\ 6\ 105\ (Decimalbased)$$

Firstly, the data order is integral, decimal（小數）, integral, decimal. Secondly, this is descending order as taught in class. Thirdly, $74 + 25 + 6 = 105$ is the the last bit value. This is checking sum. Our result from logic analyzer is corresponding with the Python output. 74 is humidity and 25.6 is temperature.



```
(25.0, 74.0)
pi@raspberrypi:~/Desktop/Adafruit_python_DHT/examples $
pi@raspberrypi:~/Desktop/Adafruit_python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=25.0*  Humidity=73.0%
(25.0, 73.0)
pi@raspberrypi:~/Desktop/Adafruit_python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=12.0*  Humidity=165.0%
(12.0, 165.0)
pi@raspberrypi:~/Desktop/Adafruit_python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=25.0*  Humidity=74.0%
(25.0, 74.0)
pi@raspberrypi:~/Desktop/Adafruit_python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=25.0*  Humidity=74.0%
(25.0, 74.0)
pi@raspberrypi:~/Desktop/Adafruit_python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=12.0*  Humidity=165.0%
(12.0, 165.0)
pi@raspberrypi:~/Desktop/Adafruit_python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=25.0*  Humidity=74.0%
(25.0, 74.0)
pi@raspberrypi:~/Desktop/Adafruit_python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=12.0*  Humidity=165.0%
(12.0, 165.0)
pi@raspberrypi:~/Desktop/Adafruit_python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=23.0*  Humidity=57.0%
(23.0, 57.0)
pi@raspberrypi:~/Desktop/Adafruit_python_DHT/examples $
```

Figure 3: Python Output

2 Terminology

2.1 What is Linux IIO subsystem?

With Linux IIO subsystem, we don't need to deal with *probe*, device tree, $sysfs$ interface, etc. For sensor irrespective with the MCU unit （ADC or DAC）, Linux IIO can help us out with it. IIO register the sensor and system add the proper interface above $sysfs$. Other IO include *hwmon*, *input* subsystem. [1] We're able to set eh sampling rate, buffer to catch the data.

2.2 What is the memory-map IO?

There are two major I/O types:

1. **I/O mapped I/O (PMIO)**
   One is I/O mapped I/O (port-mapped I/O or Direct I/O). This type of I/O, like memory, has its own memory space. Therefore, a specific command is required to deal with I/O. Its pros is free from memory space being taken up. Whereas the cons is one has to have extra command to do I/O.

2. **Memory Mapped I/O**
   I/O share memory space with memory. It maps the I/O port or memory mapping to memory address. We access I/O just like normal access to memory. The cons is the memory is taken up.

2.3 How is the efficiency difference when compared between interrupt-driven I/O and polling I/O?

It depends on how you define the efficiency. If the data rate is high, interrupt driven I/O is less efficient, because frequent interruption keep kernel busy dealing with the context switching. If the data rate is low, the polling I/O is less efficient, because most of the time looping polling I/O is getting no result from the I/O, it's waste of resources.

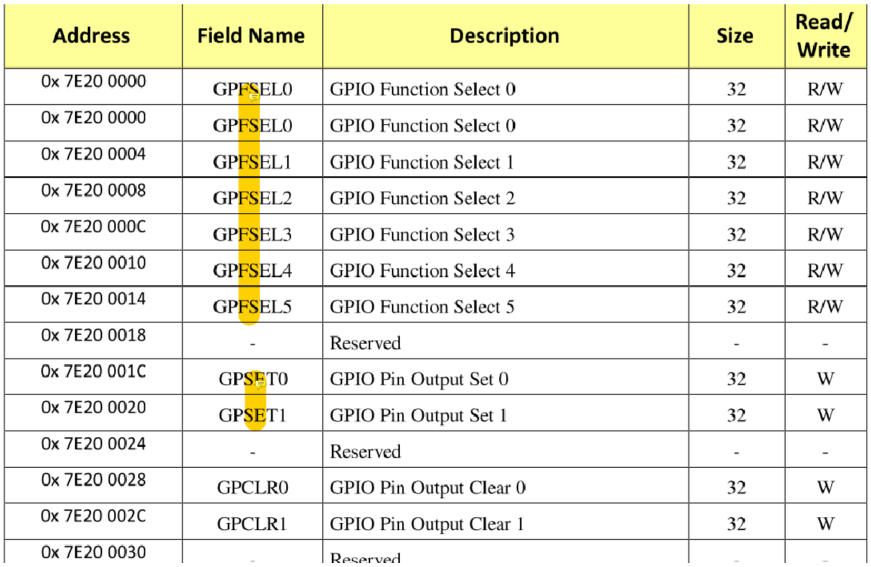
2.4 in pi_2_mmio.h, why pointer operation (pi_2_mmio_gpio+7) (pi_2_mmio_gpio+10)?

+7, +10 is the offset address offset, 7 stands for 7th word address which is $0x7E20001C$ GPSET0. Figure 4

```
// https://github.com/adafruit/Adafruit_Python_DHT/blob/master/source/Raspberry_Pi_2/pi_2_mmio.h
static inline void pi_2_mmio_set_high(const int gpio_number) {
  *(pi_2_mmio_gpio+7) = 1 << gpio_number;
}

static inline void pi_2_mmio_set_low(const int gpio_number) {
  *(pi_2_mmio_gpio+10) = 1 << gpio_number;
}
```

Therefore,
gpio+7 corresponds to GPSET, set means giving the digital output ‘1’ .
gpio+10 corresponds to GPSCLR, set means giving the digital output ‘0’ .
Notice that we can now manipulate the IO data like manipulate normal data in memory. This attributes to MMIO.



Address	Field Name	Description	Size	Read/ Write
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0004	GPFSEL1	GPIO Function Select 1	32	R/W
0x 7E20 0008	GPFSEL2	GPIO Function Select 2	32	R/W
0x 7E20 000C	GPFSEL3	GPIO Function Select 3	32	R/W
0x 7E20 0010	GPFSEL4	GPIO Function Select 4	32	R/W
0x 7E20 0014	GPFSEL5	GPIO Function Select 5	32	R/W
0x 7E20 0018	-	Reserved	-	-
0x 7E20 001C	GPSET0	GPIO Pin Output Set 0	32	W
0x 7E20 0020	GPSET1	GPIO Pin Output Set 1	32	W
0x 7E20 0024	-	Reserved	-	-
0x 7E20 0028	GPCLR0	GPIO Pin Output Clear 0	32	W
0x 7E20 002C	GPCLR1	GPIO Pin Output Clear 1	32	W
0x 7E20 0030	-	Reserved	-	-

Figure 4: IO register (Control, Data, Status) table

References

[1] 0xff07. Linux iio. https://ithelp.ithome.com.tw/articles/10251055.