Atmel 328-Processor for RaspberryPi

Ultrasonic HC-SR04

Gerhard Hepp, March 2015

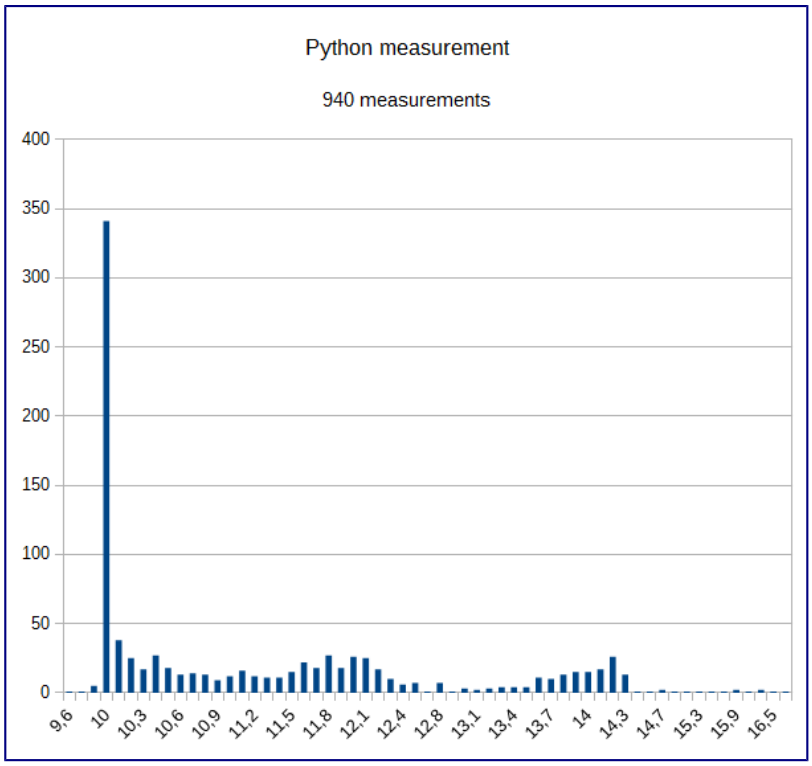
Content

Overview

The ultrasonic sensor HC-SR04 is a quite inexpensive ultrasonic sensor, well suited for microcontroller applications.
It produces pulse width modulation directly giving the signal run time.

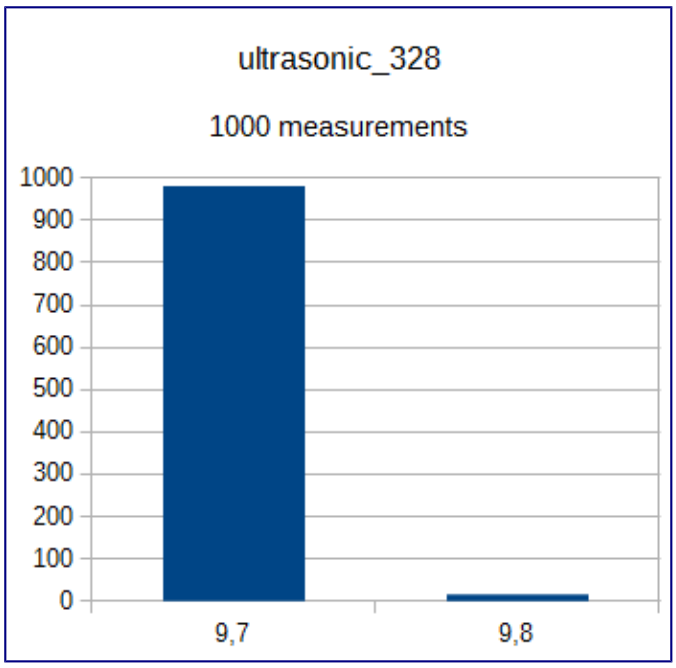
For a distance of 1.7m, at sound speed in air of 340m/s, you get 10ms pulse width.
Up to now, my scratchClient software had no ability to support this device. In order to evaluate different approaches to measure pulse width timings in this range, I have set up an arduino board simulating this device with precise 10ms timing.
The goal was to have around 1000 measurements and look for the distribution of values.

Approach A was to take python code as found on the net. Two loops, first one looking for raising pulse and second one looking for falling pulse.



Results are not very precise.
There are quite a lot of measurements at 10ms, but 2/3 of all results are up to 40% higher; few also till 16,5 ms. This is as expected, as the IO system is slow, and a lot of other things are running in the linux system in parallel.
Advantage is that there are no additional cost, except some voltage dividers needed.

Approach B is a microcontroller subsystem using the atmel 328 chip, sitting on a breadboard and connected with SPI to RPI. The internal oscillator is used for simplicity and reduced cost. This setup is similiar to the adc-setup described in another post.
Time measurement is using the 16 bit timer in 1/64 resolution, yielding 8us values. The code is polling the timer, which is fast enough for this application. SPI is handled with an interrupt coded in assembler. This allows up to 240kHz SPI frequency.



Results are pretty nice, although the deviation is 3% from 10.0ms. Most possible this is caused by the free running oscillator running a little bit too fast; some more investigation is needed here to eliminate software errors.
The software isupports up to 4 sonic sensors.
Cost is prox 3.5$ per unit, some soldering and some voltage dividers. No difficult software setup needed.

There are other approaches on the net. DMA code, C-Code or alike. Problem here is the complicated software setup (no ready to use solutions) and the difficult integration into the python environment with RPI.GPIO.

Implementation

From the above measurements, I decided to implement a firmware for 328, and connecting the atmel by SPI to raspberry Pi.

Sample code for Raspberry Pi is provided in python. Integration to scratch is achieved with the scratchClient-Software from heppg.de.

The firmware needed can be programmed from the RPi, and the device can be easily implemented on a breadboard.

Disadvantage is the lengthy installation procedure. But with simple steps and verification points this is manageable.

The firmware for the controller is ready to use in the samples, no coding required in this area. As programming the firmware into the Atmel328 is done with the RPi, no extra programming device is needed.
The programming of the atmel device is no topic of this article.

Prerequisite is that the contoller is using the internal 8MHz RC oscillator. This is default for factory new devices.
If you get a preprogrammed device with e.g. a bootloader for arduino, the internal oscillator can be switched off. In this case you need to use a programming device, or attach an

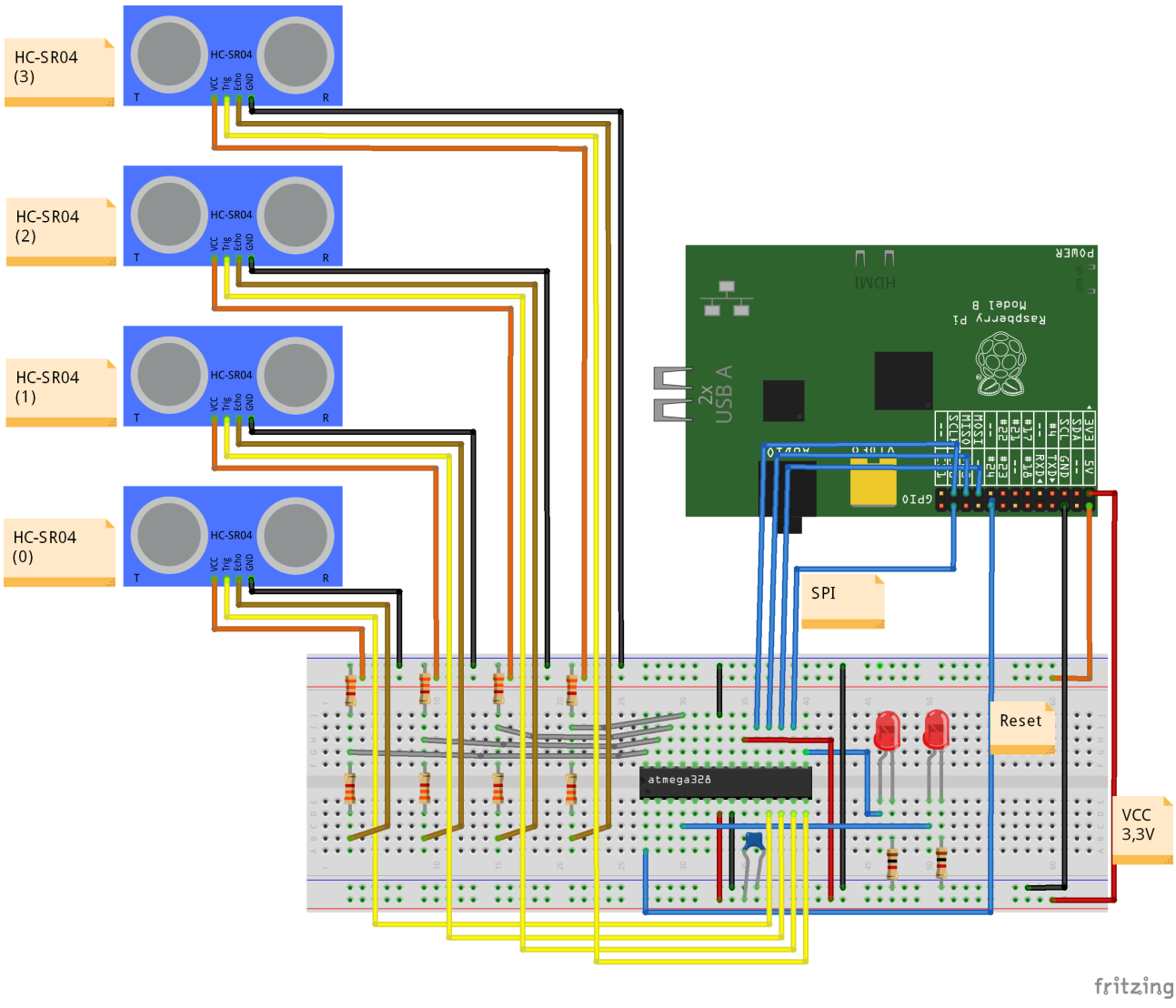
external clock signal. In the appendix, the procedure is described for this.

Setup

Parts list

- ATmega 328 P-PU (DIL-package)
- breadboard
- (optional) precision socket 28 pol, DIL 28 pol, 0.3 large. The socket is useful to protect the pins of the processor from bending. Set the socket into the breadboard and then insert the 328.
- LED, standard 20mA
- 1kOhm resistor for LED
- Capacitor 100nF, 10V min, ceramic (recommended; if not available it runs also without it)
- HC-SR04, up to four devices
- Resistor 2.2kOhm, 2 per device
- Patch cables m-f, 5 pieces and some extra to connect RPI and breadboard
- wires for breadboard

Of course you need a RPi.



Setup procedure

Power off Rpi
Insert controller in breadboard.

VCC of the controller is from 3.3V from RPi (black, red).
SPI-connections MISO, MOSI, SCK and SS an CS0 des RPI (blue).
RESET of controller Pin 1 to GPIO24 (blue)
LED with series resistor is from controller PB1, Pin 15 to GND..

Triggers are Pin 11 to 14 of atmel328
Echo input to atmel328 are Pin 28 to 25

Be careful with 5V for the HC-SR04 devices and do not mix these with 3.3V. Also carefully check the ground connections from RPI to breadboard and HC-SR04.

Install software on Raspberry Pi

The procedures are for Raspbian distribution. You need root access to the system.

```
sudo apt-get update
sudo apt-get upgrade -y
sudo apt-get install python-dev pip
sudo pip install spidev intelhex
```

Activate SPI driver. Can be done with raspi-config (enable SPI) permanently.
Or with

```
sudo modprobe  spi_bcm2708
```

which needs to be repeated after each reboot.

Copy the software archive to /home/pi and unpack.

```
tar xzvf  ultrasonic_328.tar.gz
```

Verify hardware and programming software.

Read Atmega fuses

```
cd ~/ultrasonic_328
sudo python src/program.py -rf
```

The output should read like this

```
PROGRAMMING_READ_CALIBRATION_BYTE   b0 10110000
PROGRAMMING_READ_EXTENDED_FUSE_BITS ff 11111111
   BODLEVEL0          1
   BODLEVEL1          1
   BODLEVEL2          1
PROGRAMMING_READ_FUSE_BITS         e2 11100010
   CKSEL0             0 ENABLED
   CKSEL1             1
   CKSEL2             0 ENABLED
   CKSEL3             0 ENABLED
   SUT0              0 ENABLED
   SUT1              1
   CKOUT              1
```

```
   CKDIV8             0
PROGRAMMING_READ_FUSE_HIGH_BITS     d9 11011001
   BOOTRST             1
   BOOTSZ0             0 ENABLED
   BOOTSZ1             0 ENABLED
   EESAVE              1
   WDTON               1
   SPIEN               0 ENABLED
   DWEN                1
   RSTDISBL            1
PROGRAMMING_READ_LOCK_BITS          ff 11111111
```

If there are error messages (e.g. device not in sync), then possibly there are wrong connections or the processor has fuses already programmed. Check wiring first.

Read current controller program

When output is correct, next step is to read out current flash program. Is empty on a new device, but either way useful to verify communication.

```
cd ~/ultrasonic_328
sudo python src/program.py -r
```

Output is expected like this.

```
root@raspberrypi:/home/pi/ultrasonic_328# python src/program.py -r
('read', 'out.hex')
programming_readCode
programming_enable
('PROGRAMMING_ENABLE', [172, 83, 0, 0])
(0, [255, 255, 83, 0])
programming_enable end
programming_disable
programming_readCode ende
ok
```

Flash 'blink' program

If successful, then load the first 'blink code' program into the controller. It will blink the LED.

```
cd ~/ultrasonic_328
sudo python src/program.py -p 328/steckbrett_328_blink.hex
```

Takes a few seconds, and the LED should blink.

Program fuses to final settings

Everythink ok, then flash the fuses to have the 8MHz oscillator running.

```
cd ~/ultrasonic_328
sudo python src/program.py -wf
```

Blinking will stop during the flash procedure, and restart noticeably faster, 5 times a second.
The controller has the internal clock divider disabled now, runs at 8MHz and wiring and software is ok.

Flash the firmware

Flash the firmware. It supports the various functions of the device.

```
cd ~/ultrasonic_328
sudo python src/program.py -p 328/ultrasonic_328.hex
```

Firmware Overview

The firmware features need to be enabled by configuration commands before using them.

These settings are not persistent, so after reset or reboot, these command need to be issued again.

For each Feature, there are SET_CONFIG and GET_CONFIG-commands.

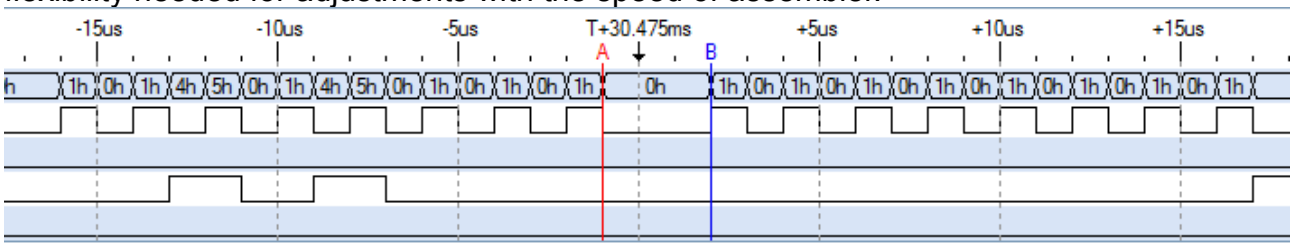
The LED-commands are available without prior activation.

After reset, the LED will blink 8 times.

A note on the commands for the controller. When a response from controller is expected, there is the need to shift the according number of dummy bytes into the SPI. These bytes are indicated by trailing '0' after the commands.

SPI clock speed is set to 240.000 Hz for the 8MHz version of the firmware. For this speed, the spidev library leaves enough time between the bytes for the interrupt program to provide responses. At a higher speed, there will be communication problems. Lower speed is no problem.

Communication on ATMEL is interrupt driven, programmed in assembler. The interrupt code is state based and was written using a custom code generator, combining the flexibility needed for adjustments with the speed of assembler.



The chart shows two byte transfers by SPI, 500kHz. The gap between the bytes is 3 us, at 8mHz CPU frequency these are 24 assembler instructions.

General Commands

Firmware, read version

GET_VERSION,0,0	0x6e	Controller responds with version number.

Version of the firmware is 0x61, minor version depends on release.

```
cd ~/ultrasonic_328
sudo python src/test_get_version.py
```

Display should be '0x61', minor version is e.g. 0x06 or higher.

Firmware: LED-Commands

SET_LED_0_0	0x64	drive Port PB1 (Pin15) low.
SET_LED_0_1	0x68	drive Port PB1 (Pin15) high.
SET_LED_1_0	0x67	drive Port PD2 (Pin4) low.
SET_LED_1_1	0x69	drive Port PD2 (Pin4) high.

Test program.
Led blinking is controlled from the raspberry. Rythm is long low, short dark.

```
cd ~/ultrasonic_328
sudo python src/test_blink.py
```

Led on, off

```
cd ~/ultrasonic_328
sudo python src/test_led_on.py
sudo python src/test_led_off.py
```

Firmware: Read Timer Values

There are four HC-SR04 devices supported.
Each channel can be enabled separately. When multiple devices are enabled, these are triggered in sequence.
Each channel provides
- time_0 (unsigned 16 bit) low to high transition
- time_1 (unsigned 16 bit) high to low transition
- count (unsigned 16 bit) number of measurements taken since enable
- error (unsigned 8 bit), 0x00 is no error.

Sample python snippet:

```
    r = self.spiManager.writeRawData( self.int_spi_bus,
                        self.int_spi_device, list( [self.GET_COUNTER_3, 0, 0, 0, 0, 0, 0, 0]))
    err = r[7]
    if err != 0x00:
        return None

    t0 = r[1] + r[2] *256
    t1 = r[3] + r[4] *256
    s = (t1 - t0 ) * 0.000008 *340/2
```

The devices need need to be enabled with SET_COUNTER_CONFIG. These settings are not persisted in the atmel-328.

GET_COUNTER_0,0,0,0,0,0,0,0	0x60	get device 0 results
GET_COUNTER_1,0,0,0,0,0,0,0	0x61	get device 1 results
GET_COUNTER_2,0,0,0,0,0,0,0	0x62	get device 2 results

GET_COUNTER_3,0,0,0,0,0,0,0	0x63	get device 3 results
SET_COUNTER_CONFIG, c0, c1	0x65	C0, bit 0, 1 = enable device 0 C0, bit 1, 1 = enable device 1 C0, bit 2, 1 = enable device 2 C0, bit 3, 1 = enable device 3
GET_COUNTER_CONFIG, 0, 0	0x66	C0, bit 0, 1 = enable device 0 C0, bit 1, 1 = enable device 1 C0, bit 2, 1 = enable device 2 C0, bit 3, 1 = enable device 3

The program reads 10 values and prints to console.

```
cd ~/ultrasonic_328
sudo python src/test_get_counter_0.py
```

If the input pin is unconnected, the error byte is set to 0x10.

Use program test_get_counter_1.py for channel 2 etc.

Firmware, auxiliary, manage delay values

The firmware allows for some adjustments of microcontroller oscillator frequency. As this affects the internal timing, the timeout constants need adjustments.
Default value for the 20ms-interval is (8000000 / 64 *0.020) = 2500.
The value of 8000000 is the frequency of the oscillator.

If oscillator frequency is higher, then adjust this value.
The atmel328 allows for 12MHz @3,3V, approximated.

read delay_20ms

GET_DELAY20MS, 0,0	0x6a	Controller responds with delay time for 20ms.

This value is the current delay time for the 20ms-timing.
The value is 8000000 / 64 *0.020

```
cd ~/ultrasonic_328
sudo python src/test_get_delay.py
```

Display should be initially '2500'.

write delay_20ms

SET_DELAY20MS, [low],[high]	0x6b	Write delay value to controller.

Be careful in writing this value.
The formula for it is f_{osc} / 64 *0.020

For 10MHz, it is 3125.

```
cd ~/ultrasonic_328
sudo python src/test_set_delay.py 3125
```

Connection to Scratch

Main purpose of this device is to interface with scratch. Download and install scratch client software from heppg.de.

Download scratchClient from heppg.de

Startup

```
cd ~/scratchClient
sudo python src/scratchClient.py -c config/config_hcsr04_atmel328.xml
```