Operating Systems for Embedded Systems
Assignment 2: proximity alert system

Student: Palmiero Christian (ID: 231599)

1. Software architecture

The software architecture relies on two source files:

❖ sample.c: the character-based device driver to support the HC-SR04 distance proximity sensor;

❖ app.c: the application to measure the distance of the object and to blink the user led accordingly.

1.1 sample.c

The entry point of the device driver is the 'sample_init_module()' function, which inserts the device in the kernel with a statically assigned major number equal to 166 and reserves the following GPIO pins:

- TRIGGER, pin D7 (PI.3 → 131), output pin of the STM32F746-DISCOVERY board corresponding to the input signal of the sensor, with a default value equal to 0;
- ECHO, pin D8 (PI.2 → 130), input pin of the STM32F746-DISCOVERY board corresponding to the output signal of the sensor;
- LED, (PI.1 → 129), output pin corresponding to the user LED, with a default value equal to 0 (LED off).

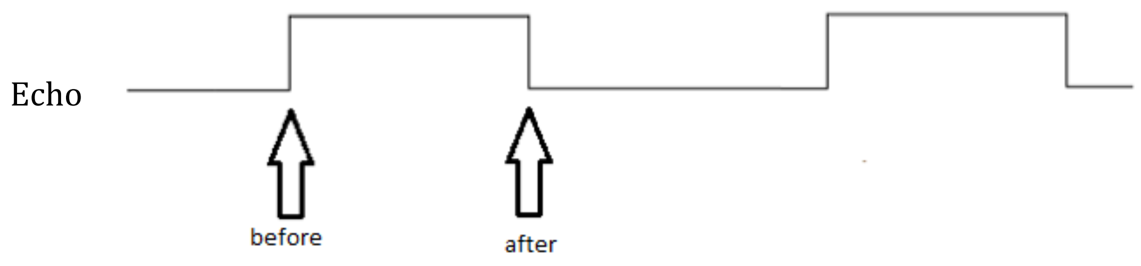
Then, the function installs the interrupt handler routine for the IRQ line of the ECHO pin, detecting both the rising and the falling edge of the signal, instantiates two workqueues, 'led_queue' and 'trig_queue', and creates a 'sem' semaphore initialized to 0.
The 'sample_open()' and the 'sample_release()' functions have the standard behaviour analysed during the lectures: the former opens the device file; the latter closes it, when it's no longer needed.
The 'sample_read()', the 'sample_write()' and the 'sample_ioctl()' functions are assignment-specific.
The device I/O control function, used to issue device-specific commands to the device file, sets a global integer variable called 'mode', that can assume one of the following three values:

- INVALID (do nothing);
- WRITE_LED (the next write sets the blinking period);
- WRITE_TRIG (the next write handles the trigger signal);

The device write function, depending on the current mode, queues a work either on the trigger workqueue or on the led workqueue.
In the first case, before queuing the work associated to the 'trig_wq_function', the trigger period, which is specified in the user space and is stored in the global 'trig_period' variable, is updated; then, the amount of time the trigger signal must remain high (a field of the 't_work' structure called 'mshigh') is defined as 1% of the total period and, if less than 0, is rounded to 1; afterwards, the amount of time the trigger signal must remain low (a field of the 't_work' structure called 'mslow') is set as the difference between the total period and the 'mshigh' value.
The trigger work function handles the sensor trigger signal in this way:

- trigger pin set to 1 (high);
- sleep for 'mshigh' milliseconds;
- trigger pin set to 0 (low);
- sleep for 'mslow' milliseconds.

In the second case, before queuing the work associated to the ‘my_wq_function’, the blinking period, which is specified in the user space and is stored in the global ‘blk_period’ variable, is updated. It’s important to underline that the work associated to the ‘my_wq_function’ is queued only the first time the write function is called with the mode set to ‘WRITE_LED’.
The LED work function causes the user LED to blink with a well-defined period, stored in the global ‘blk_period’ variable. A blinking period equal to 0 means that the user LED must be always ON. With this kind of implementation, a worst-case analysis points out that the system is reactive to the blinking semi-period.
Before analysing the device read function behaviour, it’s better to describe what the ‘ECHO_handler’ function does. If a rising edge has occurred, a temporal value is collected and is stored into the global ktime_t ‘before’ variable; otherwise, if a falling edge has occurred, another temporal value is sampled and is stored into the global ktime_t ‘after’ variable. Then, the difference between the two previous values is converted in microseconds and is saved in the ‘actual_time’ variable. Finally, the Linux equivalent to V, that is ‘up()’, is performed on the semaphore ‘sem’.



The device read function waits on the ‘sem’ semaphore and, when it’s ready to run, moves the ‘actual_time’ value to the user space.
The exit point of the device driver is the ‘sample_cleanup_module()’ function, which frees the irq and the GPIO pins and removes the device driver from the kernel.

1.1 app.c

The user-space application starts checking the number of command line parameters. Then, it opens the ‘dev/sample’ file for read-write access and sets the device trigger period. In order to do this operation, two functions are called: the I/O control with the unsigned int parameter equal to WRITE_TRIG and the write with the buffer containing the trigger period, statically defined as 67 milliseconds and stored in the ‘trig_period’ variable.
Afterwards, an infinite loop of operations is performed:

- the read function is called, in order to retrieve the width of the Echo pulse, in microseconds;
- the distance is computed, according to the following formula, taken from the HC-SR04 user manual

$$Distance[cm] = \frac{Width\ of\ the\ Echo\ pulse[\mu s]}{58}$$

- two measurements are collected;
- the average is calculated (in order to achieve a better accuracy) and is printed on the STM32F746-DISCOVERY serial console;
- the next blinking period is computed, according to the assignment constraints;
- if the next blinking period is different from the current one, the user LED blinking period is updated. In order to do this operation, two functions are called: the I/O control with the

unsigned int parameter equal to WRITE_LED and the write with the buffer containing the blinking period.

2. Pin assignment

VCC->5v; GND->GND; Trig-> Pin D7 (PI.3); Echo->Pin D8 (PI.2)

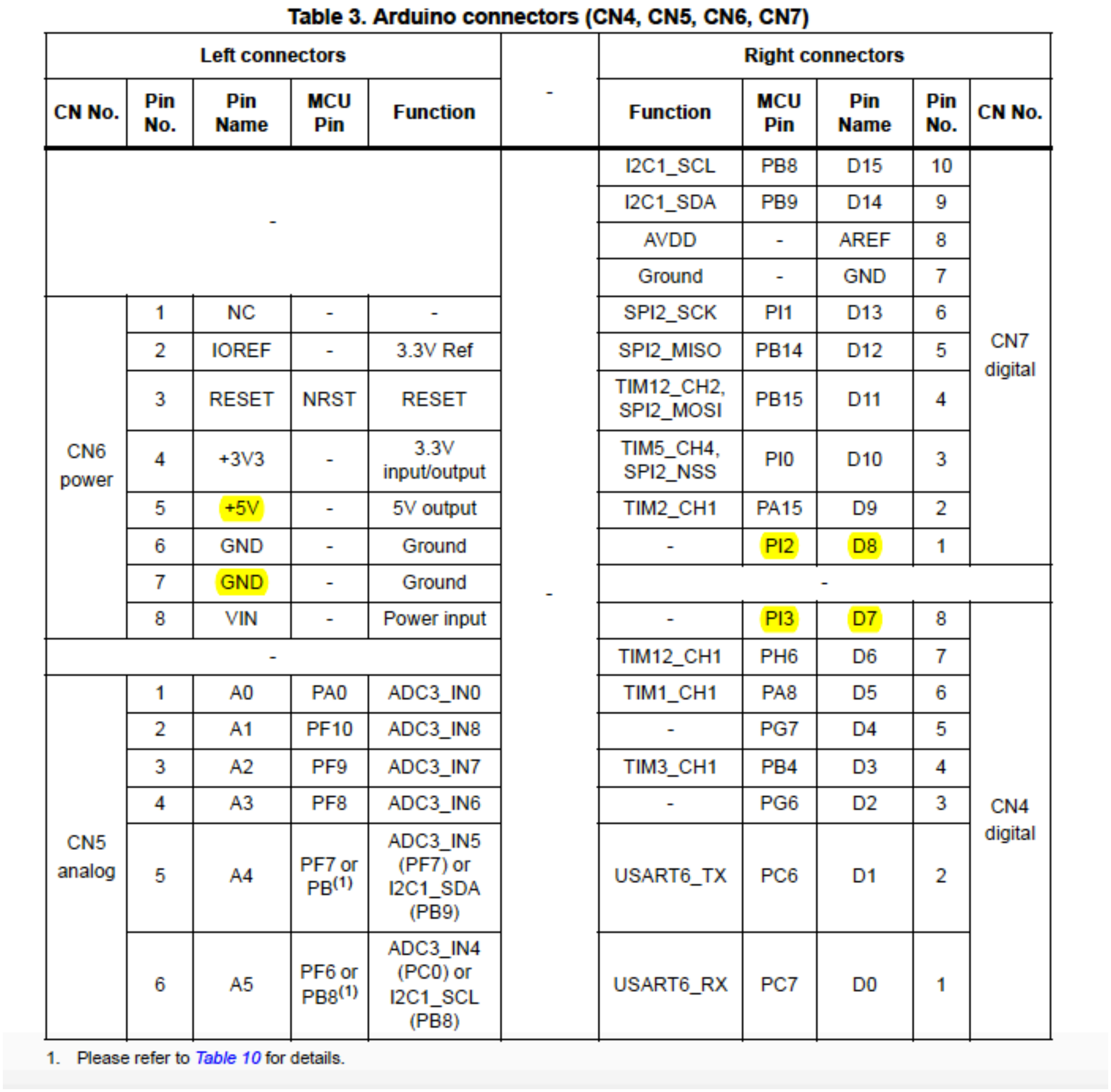


Table 3. Arduino connectors (CN4, CN5, CN6, CN7)

Left connectors					-	Right connectors				
CN No.	Pin No.	Pin Name	MCU Pin	Function		Function	MCU Pin	Pin Name	Pin No.	CN No.
-					-	I2C1_SCL	PB8	D15	10	CN7 digital
						I2C1_SDA	PB9	D14	9	
						AVDD	-	AREF	8	
						Ground	-	GND	7	
CN6 power	1	NC	-	-		SPI2_SCK	PI1	D13	6	
	2	IOREF	-	3.3V Ref		SPI2_MISO	PB14	D12	5	
	3	RESET	NRST	RESET		TIM12_CH2, SPI2_MOSI	PB15	D11	4	
	4	+3V3	-	3.3V input/output		TIM5_CH4, SPI2_NSS	PI0	D10	3	
	5	+5V	-	5V output		TIM2_CH1	PA15	D9	2	
	6	GND	-	Ground		-	PI2	D8	1	
	7	GND	-	Ground		-				
	8	VIN	-	Power input		-	PI3	D7	8	CN4 digital
-						TIM12_CH1	PH6	D6	7	
CN5 analog	1	A0	PA0	ADC3_IN0		TIM1_CH1	PA8	D5	6	
	2	A1	PF10	ADC3_IN8		-	PG7	D4	5	
	3	A2	PF9	ADC3_IN7		TIM3_CH1	PB4	D3	4	
	4	A3	PF8	ADC3_IN6		-	PG6	D2	3	
	5	A4	PF7 or PB(1)	ADC3_IN5 (PF7) or I2C1_SDA (PB9)		USART6_TX	PC6	D1	2	
	6	A5	PF6 or PB8(1)	ADC3_IN4 (PC0) or I2C1_SCL (PB8)		USART6_RX	PC7	D0	1	

1. Please refer to *Table 10* for details.

3. How to use the project

-Connect the board to your PC using both the USB cable and the Ethernet cable
-On a 32-bit Linux distribution, open a terminal application, such as minicom, and set the proper serial port configuration
-Load the Kernel through the TFTP protocol
-Mount the initial RAM disk through the NFS protocol
-On the target system linux console, execute the following commands:

- insmod sample.ko
- ./app