ECOLE CENTRALE DE NANTES

LABORATORY REPORT LAB 3

ULTRASONIC SENSOR

By:

AROMOSE QUDUS AYINDE

SUBMITTED TO: PROF. MICHAEL HILARET

Program: Embedded Computing

Program Specialization: M1 Electric Vehicles Propulsion and Control.

January 12, 2024.

AIM AND OBJECTIVE: This lab focuses on the interrupts, but still uses GPIOs and timers. The objective is to write the driver for an ultrasonic sensor, without any polling implementation.

PRINCIPLES OF OPERATION:

The sensor embeds an integrated circuit for an easy management by the MCU (MicroController Unit). The interface consists in only 2 logic signals (Trigger and echo), as in figure 1.

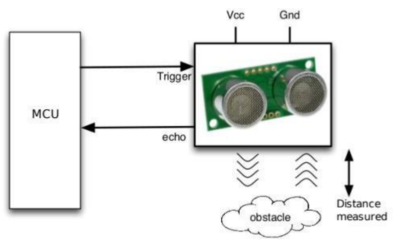

Figure 1: Connection between the ultrasonic sensor and the MCU.

The sensor works as follow (figure 2): • a pulse to the Trigger signal is a request to perform a measure. • the integrated circuit generates a sequence of 40KHz ultrasonic burst. • an analog circuit detects the echo from an obstacle; • The distance to the obstacle is deduced from the time elapsed between the trigger ultrasonic burst and the incoming echo, knowing the sound speed. A pulse is transmitted back on the echo signal, on which the pulse width depends on the distance to the obstacle: 58µs/cm.

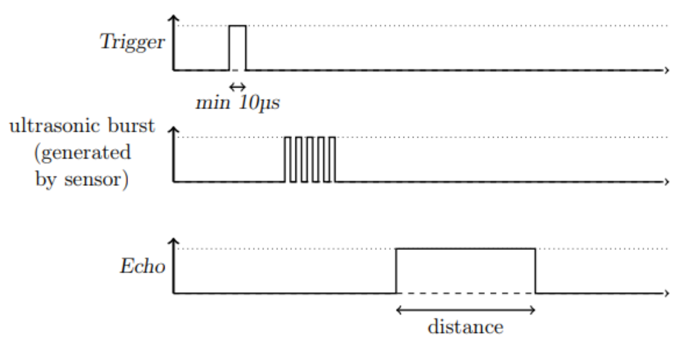

Figure 2: Time sequence of the ultrasonic sensor. The echo pulse width depends on the distance to the obstacle (58µs/cm)

Question 1 Write the trigger part of the application:

• configuration of the sensor pin PA10 as an output.

• interrupt @10Hz using TIM6.

• generation of the Trigger signal.

NOTE: pin PA10 shares both Trigger (output) and Echo (input) signals. As a consequence, the pin should be in the output mode as little time as possible.

SOLUTION

```
void setup()
{
// GPIO setup
RCC->AHBENR |= RCC_AHBENR_GPIOAEN_Msk; //GPIOA enable

// Pin mode
GPIOA -> MODER &= ~(0x03<<20); // reset MODER for PA10
GPIOA -> MODER |= (0x01<<20); //PA10 as output

// Timer setup
RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;
RCC->APB1RSTR |= RCC_APB1RSTR_TIM6RST;
RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM6RST;

// Setting 10Hz for Timer6
TIM6->PSC = 64000-1; // prescaler : tick@1ms
TIM6->ARR = 100-1; // 100 * 1ms = 100ms = 10 Hz
TIM6->CR1 |= TIM_CR1_CEN; // control reg : enable timer

// Interrupt setup
TIM6->DIER |= TIM_DIER_UIE; // Interrupt enable register - UIE
NVIC_EnableIRQ(TIM6_DAC1_IRQn);
}


void TIM6_DAC1_IRQHandler()
{
TIM6 -> SR &= ~TIM_SR_UIF; // Status register - reset Timer flag to

GPIOA -> MODER &= ~(0X03<<20);

GPIOA -> MODER |= (0X01<<20); // PA10 as output

//Trigger request at 10 Hz with 10us high state

GPIOA -> ODR |= (0x1<<10); //Set PA10

for(volatile int i=0;i<20;i++);

//reset PA10

GPIOA -> ODR &= ~(0x1<<10);

// PA10 as  Echo input
GPIOA -> MODER &= ~(0x03<<20);
GPIOA -> PUPDR &= ~(0x03<<20); // PA10 as Pull-down to detect rising edge
GPIOA -> PUPDR |= (2<<20);

}
```

```
int main()

{

setup();

}
```

3 ECHO SIGNAL

The distance to the obstacle is given by the duration of the Echo pulse. We use the EXTI peripheral to detect rising/falling edges, and a timer (TIM7) as a stopwatch to get the pulse width. The measure is stored in a global variable (figure 3).

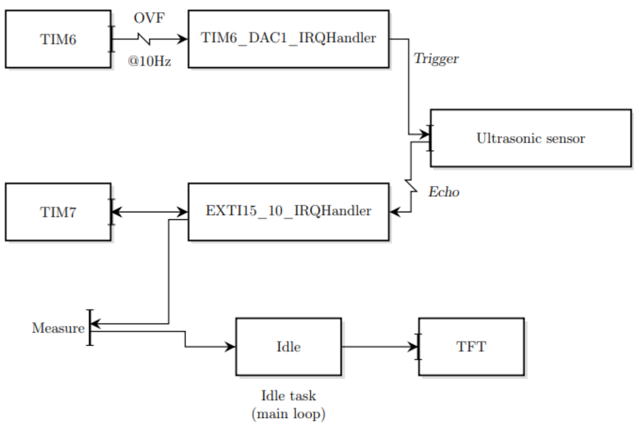

Figure 3: Driver Architechture.

The principle uses TIM7 as a chronometer (resolution 1µs) and an interrupt in Echo:

- If there is a rising edge on Echo, TIM7 count value is reset
- If there is a falling edge on Echo, the value of TIM7 is stored to the measure value

Question 2 : Program the whole application, that refresh the value of measure periodi cally. check the correct value using the debugg

Question 3: Add a continuous display of the sensor value on the TFT (in the main loop), in mm. 4 Extension The echo signal should occurs less that 50ms after the trigger.

Question 4: Add a timeout function that informs the user that the sensor is not available if there is no response after 50ms (using an interrupt of course - TIM7 for instance).

SOLUTION (QUESTION 2-QUESTION 4)

```
void setup()
{
// GPIO setup
RCC->AHBENR |= RCC_AHBENR_GPIOAEN_Msk; //GPIOA enable

// Pin mode

GPIOA -> MODER &= ~(0x03<<20); //
GPIOA -> MODER |= (0x01<<20); //PA10 as output


// Timer setup TIM6
RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;
RCC->APB1RSTR |= RCC_APB1RSTR_TIM6RST;
RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM6RST;

// Timer setup TIM7
RCC->APB1ENR |= RCC_APB1ENR_TIM7EN; //enable peripheral clock
RCC->APB1RSTR |= RCC_APB1RSTR_TIM7RST;
RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM7RST;


// Setting 10Hz for Timer6

TIM6->PSC = 64000-1; // prescaler : tick@1ms
TIM6->ARR = 100-1; // 100 * 1ms = 100ms = 10 Hz
TIM6->CR1 |= TIM_CR1_CEN; // control reg : enable timer


// Setting 10Hz for Timer6
TIM7->PSC = 64-1; // prescaler : tick@1u
TIM7->ARR = 50000-1; // 50.000 * 1us = 50ms
TIM7->CR1 |= (1<<3); // One pulse mode



// Interrupt setup TIM6
TIM6->DIER |= TIM_DIER_UIE; // Interrupt enable register - UIE
NVIC_EnableIRQ(TIM6_DAC1_IRQn);


// Interrupt setup TIM7
TIM7->DIER |= TIM_DIER_UIE; // Interrupt enable register - UIE
NVIC_EnableIRQ(TIM7_DAC2_IRQn);


// Interrupt setup EXTI15
RCC -> APB2ENR |= RCC_APB2ENR_SYSCFGEN; // enable peripheral clock
EXTI -> IMR |= EXTI_IMR_MR10;//enable at least one external line
SYSCFG -> EXTICR[3] &= ~(0x0F<<8); // assign external interrupt
EXTI -> RTSR |= (1<<10); // Set rising edge sensitivity for pin 10
EXTI -> FTSR |= (1<<10); // Set falling edge sensitivity for pin 10
```

```
// check pin level after interrupt to check if it was rising or falling edge
NVIC_EnableIRQ(EXTI15_10_IRQn);
}


void TIM6_DAC1_IRQHandler()

{

TIM6 -> SR &= ~TIM_SR_UIF; // Status register - reset Timer flag to
restart timer (alt TIM6->SR = 0)

// specific application code
EXTI->IMR &= ~EXTI_IMR_MR10;
GPIOA -> MODER &= ~(0X03<<20);
GPIOA -> MODER |= (0X01<<20); // PA10 as output


//Trigger request at 10 Hz with 10us high state
GPIOA -> ODR |= (0x1<<10); //Set PA10


for(volatile int i=0;i<20;i++);

//reset PA10
GPIOA -> ODR &= ~(0x1<<10); //reSet PA10

// PA10 as input
GPIOA -> MODER &= ~(0x03<<20);
GPIOA -> PUPDR &= ~(0x03<<20); // PA10 as Pull-down to detect rising edge
GPIOA -> PUPDR |= (2<<20);
EXTI -> IMR |= EXTI_IMR_MR10; // Mask register 10, 0x01<<10
}


void EXTI15_10_IRQHandler()
{

EXTI -> PR |= (1<<10); // clear pending interrupt by writing 1 to bit 10


if((GPIOA -> IDR & (0x01<<10)) == 1) // case rising edge
{
// start timer 7
TIM7->CNT = 0;//clear counter in timer 7
TIM7->CR1 |= TIM_CR1_CEN; // start timer 7

// PA10 as Pull-up to detect falling edge
GPIOA -> PUPDR &= (0x03<<20);
GPIOA -> PUPDR |= (0x01<<20);
}


if((GPIOA -> IDR & (1<<10)) == 0) // case falling edge
{
```

```
// stop timer 7
TIM7 -> CR1 &= ~(1<<0); // stop timer 7
measure = TIM7 -> CNT;

// Calculate the distance between the obstacle and the Ultrasonic Sensor
int distance_cm = (1/58) * measure;
obstacle = 1;
}

}


void TIM7_DAC2_IRQHandler()
{
TIM7 -> SR = ~TIM_SR_UIF;
// Stop TIM7
TIM7->CR1 |= ~TIM_CR1_CEN;

obstacle = 0;
}


int main(void)
{
setup();

while (1)
{
/* USER CODE END WHILE */
printf("%d", obstacle);
/* USER CODE BEGIN 3 */

}

}
```

CONCLUSION

The following were achieved:

- **Setting of Basic Timer (TIM6 and TIM7)**
- **Setting of Timer interrupt and generation of signal in Timer interrupt handler.**
- **Setting and initialization of GPIOs**
- **Setting of External interrupt and use of Timer peripheral as chronometer.**