Smart Software Project

Lecture: Week 9

Ultrasonic Sensors
& Buzzer

Prof. HyungJune Lee
hyungjune.lee@ewha.ac.kr

이화여자대학교
EWHA WOMANS UNIVERSITY

Today

- Review
  - Interrupt vs. Polling

- SmartCAR Ultrasonic sensors
- SmartCAR Buzzer

- Announcement

Class Schedule

Week	Lecture Contents	Lab Contents
Week 1	Course introduction	Arduino introduction: platform & programming environment
Week 2	Embedded system overview & source management in collaborative repository (using GitHub)	Lab 1: Arduino Mega 2560 board & SmartCAR platform
Week 3	ATmega2560 Micro-controller (MCU): architecture & I/O ports, Analog vs. Digital, Pulse Width Modulation	Lab 2: SmartCAR LED control
Week 4	Analog vs. Digital & Pulse Width Modulation	Lab 3: SmartCAR motor control (Due: HW on creating project repository using GitHub)
Week 5	ATmega2560 MCU: memory, I/O ports, UART	Lab 4: SmartCAR control via Android Bluetooth
Week 6	ATmega2560 UART control & Bluetooth communication between Arduino platform and Android device	Lab 5: SmartCAR control through your own customized Android app (Due: Project proposal)
Week 7	Midterm exam	
Week 8	ATmega2560 Timer, Interrupts & Ultrasonic sensors	Lab 6: SmartCAR ultrasonic sensing
Week 9	Infrared sensors & Buzzer	Lab 7: SmartCAR infrared sensing
Week 10	Acquiring location information from Android device & line tracing	Lab 8: Implementation of line tracer
Week 11	Gyroscope, accelerometer, and compass sensors	Lab 9: Using gyroscope, accelerometer, and compass sensors
Week 12	Project	Team meeting (for progress check)
Week 13	Project	Team meeting (for progress check)
Week 14	Course wrap-up & next steps	
Week 15	Project presentation & demo I (Due: source code, presentation slides, & poster slide)	Project presentation & demo II
Week 16	Final week (no final exam)	

Today

- **Review**
  - – Interrupt vs. Polling

- SmartCAR Ultrasonic sensors
- SmartCAR Buzzer

- Announcement

Interrupts

- Definition
  - An event external to the currently executing process that causes a change in the normal flow of instruction execution
  - Usually generated by hardware devices external to the CPU
  - Key point: Interrupts are asynchronous w.r.t. current process
    - Typically indicate that some device needs service

Why Interrupts?

- People like connecting devices
  - A computer is much more than the CPU
    - Keyboard, mouse, screen, disk drives
    - Scanner, printer, sound card, camera, etc.
  - These devices occasionally need CPU service
    - But we can't predict when
  - External events typically occurs on a macroscopic timescale
    - We want to keep the CPU busy between these events
- Need a way for CPU to find out devices need attention

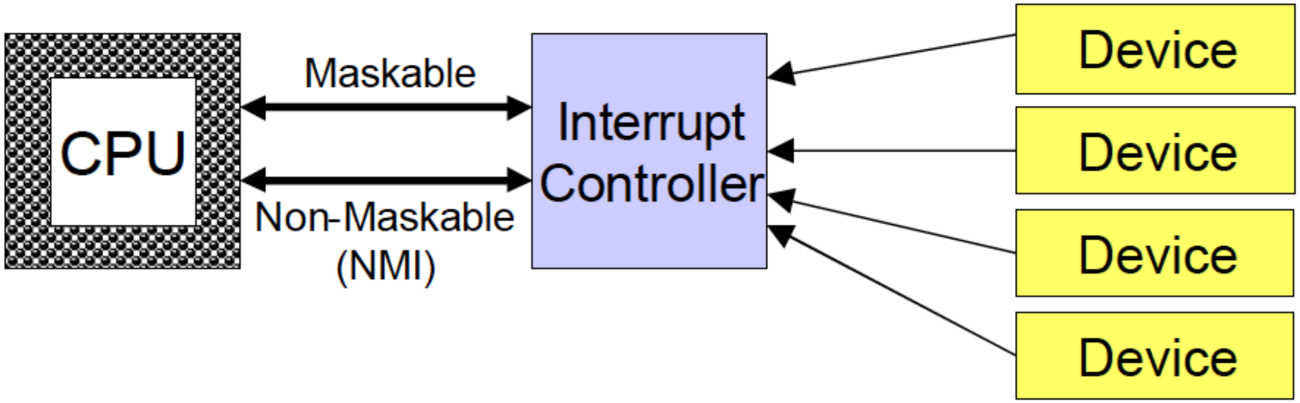
Possible Solution: Polling

- CPU periodically checks each device to see if it needs service
  - Takes CPU time even when no requests pending ☹
  - Overhead may be reduced at expense of response time ☹
  - Can be efficient if events arrive rapidly ☺

"Polling is like picking up your phone every few seconds to see if you have a call…"

Alternative: Interrupts

- Give each device a wire (interrupt line) that it c an use to signal the processor
  - When interrupt is signaled, processor executes a routine called an interrupt handler to deal with the interrupt
  - No overhead when no requests pending



Polling vs. Interrupts

- Polling
  - "Like picking up your phone very few seconds to see if you have a call"
- Interrupts
  - "Like waiting for the phone to ring"

- Interrupts win if processor has other work to do and event response time is not critical
- Polling can be better if processor has to respond to an event ASAP
  - May be used in device controller that contains dedicated secondary processor

Hardware Interrupt Handling

- Details are architecture dependent
- Interrupt controller signals CPU that interrupt has occurred, passes interrupt number
  - Interrupts are assigned priorities to handle simultaneous interrupts
  - Lower priority interrupts may be disabled during service
- CPU senses (checks) interrupt request line after every instruction; if raised, then:
  - Uses interrupt number to determine which handler to start
  - Interrupt vector associates handlers with interrupts
- Basic program state saved (as for system call)
- CPU jumps to interrupt handler
- When interrupt is done, program state reloaded and program resumes

Arduino Interrupt Handling

- ATmega2560 Interrupt trigger
  - When input signal has changed from ‘0’ to ‘1’ or from ‘1’ to ‘0’ (edge trigger), or
    stays at 1 or 0 (level trigger), an interrupted is “triggered”

    - Edge Trigger
      - At the moment that changes from ‘1’ to ‘0’ (Falling Edge Trigger)
      - At the moment that changes from ‘0’ to ‘1’ (Rising Edge Trigger)
      - Pulse should stay at least 50ns

    - Level Trigger
      - If input signal stays for a moment, then it is triggered

Arduino Interrupt Functions

- Two interrupt-related functions supported in Arduino
  - attachInterrupt(interrupt, function, mode)
    - Set an interrupt number and triggering way
      - Interrupt: Interrupt number to use.
      - Function: Interrupt Service Routine function upon interrupt occurred
      - Mode: Interrupt mode
        - » “Level Trigger”
          - LOW – Triggered at the LOW level
          - CHANGE – Triggered if the level has been changed
        - » “Edge Trigger”
          - RISING – Triggered at the rising edge
          - FALLING – Triggered at the falling edge
  - detachInterrupt(interrupt)
    - Terminate the usage of interrupt
      - Interrupt: Interrupt number to stop using the interrupt

  - Interrupt information in SmartCAR

Interrupt No.	0	1	2	3	4	5	6	7
Port / Pin No.	PE4 / 2	PE5 / 3	PD0 / 21	PD1 / 20	PD2 / 19	PD3 / 18	PE6 / -	PE7 / -

Sample Program

```
#define MY_PIN  13
int state = LOW;

void setup(){
   pinMode(MY_PIN, OUTPUT);
    attachInterrupt(0, blink, CHANGE);
}

void loop() {
   digitalWrite(MY_PIN, state);
}

void blink() {
   state = !state;
}
```

- attachInterrupt(0, blink, CHANGE)
  - Interrupt setting
    - 1st argument
      - Use interrupt number 0 (pin 2) for interrupt (assuming that MY_PIN 13 is also connected to pin 2)
    - 2nd argument
      - Interrupt Service Routine function you want to execute upon interrupt occurred
      - Called as "interrupt handler": when an interrupt occurs, blink function will be called
    - 3rd argument
      - CHANGE: whenever pin level has changed (high to low or low to high), the interrupt occurs

Today

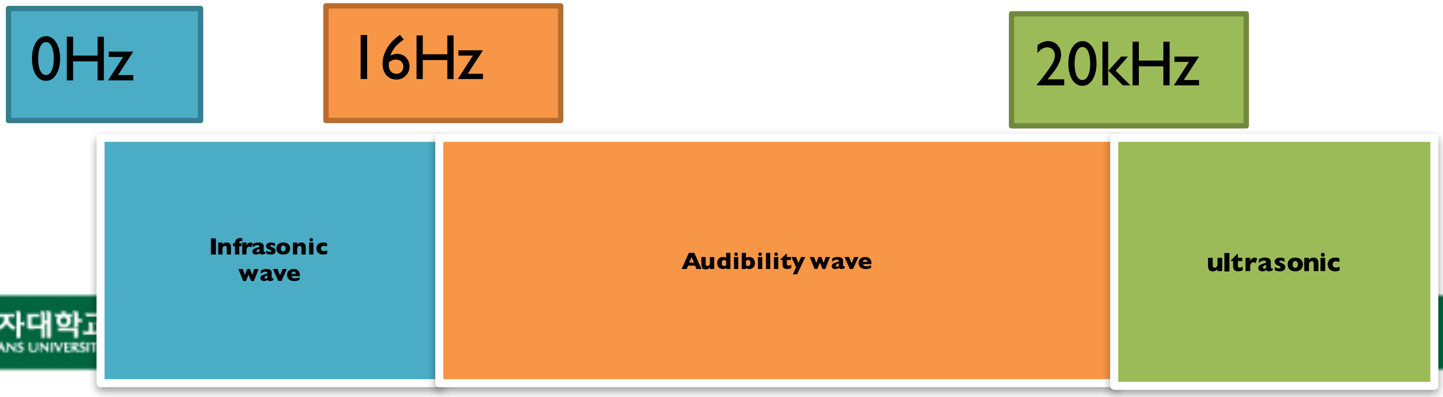
- Review
  - Interrupt vs. Polling

- **SmartCAR Ultrasonic sensors**
- SmartCAR Buzzer

- Announcement

Ultrasonic wave

- Ultrasonic wave?
  - Sound wave with high frequency
  - Sound wave
    - Sound is transmitted through gases, plasma, and liquids
    - Audible wave
      - 20Hz~20kHz spectrum
    - Ultrasonic wave
      - Frequency spectrum where human cannot hear
  - Spectrum



0Hz

16Hz

20kHz

Infrasonic wave

Audibility wave

ultrasonic

SmartCAR Ultrasonic Sensors

- 12 ultrasonic sensors

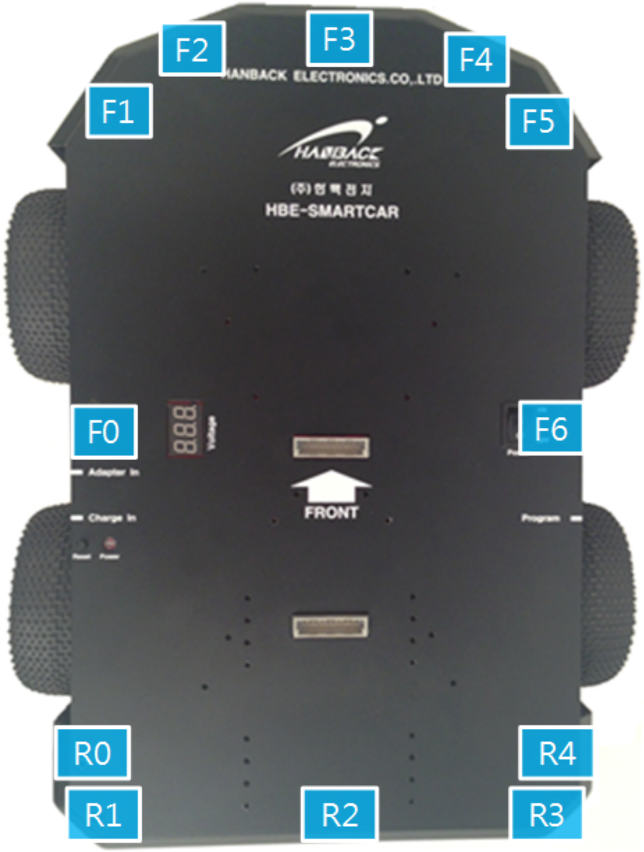
  ① Ultrasonic sensor (Transmitter)

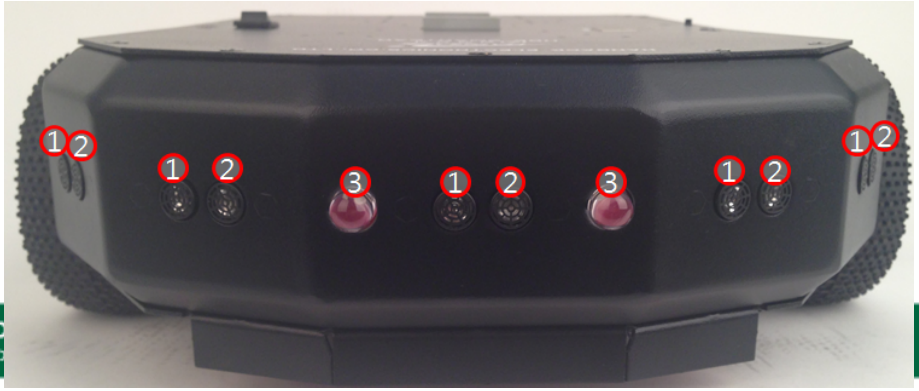
  Transmit an ultrasonic wave to detect an object

  ② Ultrasonic sensor (Receiver)

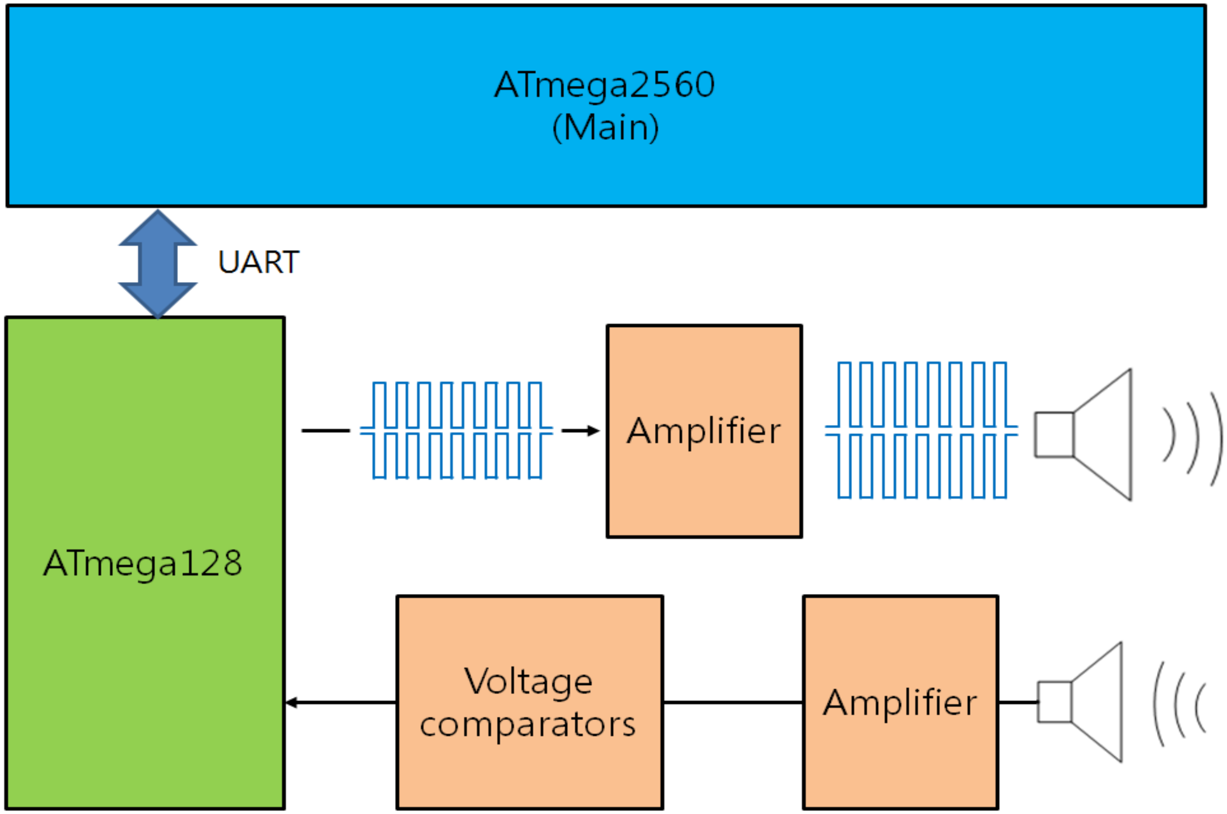
  Receive the ultrasonic wave transmitted from TX

  => Calculates the distance from the TX-RX time diff





Ultrasonic Sensor Hardware Architecture



SmartCAR UART Port Configuration

- UART1 port is used for ultrasonic sensors

UART No.	Name	Port / Number	Etc
UART0	RXD0	PE0 / -	Program port Bluetooth port
	TXD0	PE1 / -	
UART1	RXD1	PD2 / 19	Ultrasonic sensor
	TXD1	PD3 / 18	
UART2	RXD2	PH0 / 17	Extension board 1
	TXD2	PH1 / 16	
UART3	RXD3	PJ0 / 15	Extension board 2
	TXD3	PJ1 / 14	

- Baud rate should be set to 115200bps

Main MCU and ATmega128 Communication

- OFF
  - Stop measuring from the ultrasonic

TX Data Packet (ATmega2560 -> ATmega128)				
Start		ID		CSC
0x76	0x00	0x0F	0x00	0x0F

- CSC : to check error – all ID sum & 0xFF

RX Data Packet (ATmega128 -> ATmega2560)								
Start		ID		DATA				
0x76	0x00	0x1F	0x00	0x00	0x00	0x00	0x00	0x00

DATA							CSC
0x00	0x00	0x00	0x00	0x00	0x00	0x00	

- CSC : to check error – all ID & Data sum & 0xFF

Main MCU and ATmega128 Communication

- Basic
  - Front 3 ultrasonic sensors (F2, F3, F4) & Rear 1 ultrasonic sensor (R2)
  - Send a TX data request packet, and receive a RX data packet for the measurement continuously

TX Data Packet (ATmega2560 -> ATmega128)				
Start		ID		CSC
0x76	0x00	0x10	0x00	0x10

RX Data Packet (ATmega128 -> ATmega2560)								
Start		ID		DATA				
0x76	0x00	0x11	0x00	0x00	0x00	F2	F3	F4
DATA							CSC	
0x00	0x00	0x00	0x00	R2	0x00	0x00		

- F0~6 : distance in front ultrasonic sensors
- R0~4 : distance in rear ultrasonic sensors

Main MCU and ATmega128 Communication

- Right
  - Front 5 ultrasonic sensors (F2 ~ F6) & Rear 5 ultrasonic sensors (R0 ~ R4)
  - Send a TX data request packet, and receive a RX data packet for the measurement continuously

TX Data Packet (ATmega2560 -> ATmega128)				
Start		ID		CSC
0x76	0x00	0x20	0x00	0x20

RX Data Packet (ATmega128 -> ATmega2560)								
Start		ID		DATA				
0x76	0x00	0x21	0x00	0x00	0x00	F2	F3	F4
DATA							CSC	
F5	F6	R0	R1	R2	R3	R4		

- F0~6 : distance in front ultrasonic sensors
- R0~4 : distance in rear ultrasonic sensors

Main MCU and ATmega128 Communication

- Left
  - Front 5 ultrasonic sensors (F0 ~ F4) & Rear 5 ultrasonic sensors (R0 ~ R4)
  - Send a TX data request packet, and receive a RX data packet for the measurement continuously

TX Data Packet (ATmega2560 -> ATmega128)				
Start		ID		CSC
0x76	0x00	0x30	0x00	0x30

RX Data Packet (ATmega128 -> ATmega2560)								
Start		ID		DATA				
0x76	0x00	0x31	0x00	F0	F1	F2	F3	F4
DATA							CSC	
0x00	0x00	R0	R1	R2	R3	R4		

- F0~6 : distance in front ultrasonic sensors
- R0~4 : distance in rear ultrasonic sensors

Main MCU and ATmega128 Communication

- Front
  - Front 7 ultrasonic sensors (F0 ~ F6)
  - Send a TX data request packet, and receive a RX data packet for the measurement continuously

TX Data Packet (ATmega2560 -> ATmega128)				
Start		ID		CSC
0x76	0x00	0x40	0x00	0x40

RX Data Packet (ATmega128 -> ATmega2560)								
Start		ID		DATA				
0x76	0x00	0x41	0x00	F0	F1	F2	F3	F4
DATA							CSC	
F5	F6	0x00	0x00	0x00	0x00	0x00		

- F0~6 : distance in front ultrasonic sensors
- R0~4 : distance in rear ultrasonic sensors

Main MCU and ATmega128 Communication

- Back
  - Front 2 ultrasonic sensors (F0, F6) & Rear 5 ultrasonic sensors (R0 ~ R4)
  - Send a TX data request packet, and receive a RX data packet for the measurement continuously

TX Data Packet (ATmega2560 -> ATmega128)				
Start		ID		CSC
0x76	0x00	0x50	0x00	0x50

RX Data Packet (ATmega128 -> ATmega2560)								
Start		ID		DATA				
0x76	0x00	0x51	0x00	F0	0x00	0x00	0x00	0x00
DATA							CSC	
0x00	F6	R0	R1	R2	R3	R4		

- F0~6 : distance in front ultrasonic sensors
- R0~4 : distance in rear ultrasonic sensors

Main MCU and ATmega128 Communication

- All
  - Front 7 ultrasonic sensors (F0 ~ F6) & Rear 5 ultrasonic sensors (R0 ~ R4)
  - Send a TX data request packet, and receive a RX data packet for the measurement continuously

TX Data Packet (ATmega2560 -> ATmega128)				
Start		ID		CSC
0x76	0x00	0xF0	0x00	0xF0

RX Data Packet (ATmega128 -> ATmega2560)								
Start		ID		DATA				
0x76	0x00	0xF1	0x00	F0	F1	F2	F3	F4
DATA							CSC	
F5	F6	R0	R1	R2	R3	R4		

- F0~6 : distance in front ultrasonic sensors
- R0~4 : distance in rear ultrasonic sensors

SmartCAR Firmware

```
#define NUM_TX_BYTES          5
#define NUM_RX_BYTES          17

unsigned char TX_buf[NUM_TX_BYTES] = {0x76, 0x00, 0xF0, 0x00, 0xF0};
unsigned char TX_stop_buf[NUM_TX_BYTES] = {0x76, 0x00, 0x0F, 0x00, 0x0F};
unsigned char RX_buf[NUM_RX_BYTES];

boolean  ultrasonic_result  = false;

void setup()
{
    int i = 0;
    Serial.begin(115200);

    while (text[i] != '₩0')
        Serial.write(text[i++]);

    Serial.write("Received  cmds: ");

    Serial1.begin(115200);
    //initialize  ports
    pinMode(....);
    ....
    digitalWrite(...);
}
```

SmartCAR Firmware

```
void loop()
{
}

void serialEvent()
{
    int command = Serial.read();

    switch (command)
    {
        case 1:
            move_stop();
            delay(500);

            move_forward();
            break;
        case 2:
            move_stop();
            delay(500);

            turn_left();
            break;
        case 3:
            move_stop();
            delay(500);

            turn_right();
            break;
        case 4:
            move_stop();
            delay(500);

            move_backward();
            break;
```

```
        case 5:
            move_stop();
            break;
        case 6:
            front_led_control(true);
            break;
        case 7:
            front_led_control(false);
            break;
        case 8:
            rear_led_control(true);
            break;
        case 9:
            rear_led_control(false);
            break;
        case 10:
            ultrasonic_sensor_read();
            break;
        default:
            move_stop();
            front_led_control(false);
            rear_led_control(false);
    }
}
```

SmartCAR Firmware

```
void ultrasonic_sensor_read()
{
    ultrasonic_result = false;
    Serial1.write(TX_buf, NUM_TX_BYTES);
}

void serialEvent1()
{
    unsigned char z, tmp = 0;
    Serial1.readBytes((char *)RX_buf, NUM_RX_BYTES);

    if ( (RX_buf[0] == 0x76) && (RX_buf[1] == 0x00) &&
          (ultrasonic_result == false) )
    {
        for (z = 2; z < NUM_RX_BYTES-1; z++)
            tmp += RX_buf[z];

        tmp = tmp & 0xFF;

        if (RX_buf[NUM_RX_BYTES-1] == tmp)
        {
            Serial.println("FRONT");
            for (z=4; z < 11; z++)
            {
                Serial.print(" F");
                Serial.print(z-4);
                Serial.print(": ");
                Serial.print(RX_buf[z]);
            }
```

Execute the ultrasonic sensor!

```
            Serial.println("₩nBACK");
            for (z=11; z < NUM_RX_BYTES-1; z++)
            {
                Serial.print("  B");
                Serial.print(z-11);
                Serial.print(":  ");
                Serial.print(RX_buf[z]);
             }
        }
        ultrasonic_result = true;
        Serial1.write(TX_stop_buf,
                                NUM_TX_BYTES);
    }
}
```

Measure only once and then disable the ultrasonic sensor!

Today

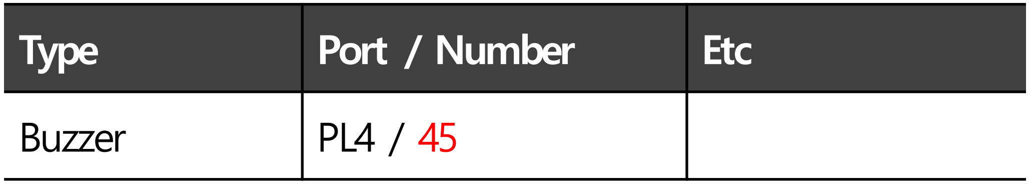
- Review
  - Interrupt vs. Polling

- SmartCAR Ultrasonic sensors
- **SmartCAR Buzzer**

- Announcement

SmartCAR Buzzer Functionality

- SmartCAR Buzzer to make a sound
  - Buzzer ON:
    - digitalWrite(45, HIGH);
  - Buzzer OFF:
    - digitalWrite(45, LOW);



Type	Port / Number	Etc
Buzzer	PL4 / 45	

Course Announcement

- For lab session, we will cover
  - Creating your own Android app communicating with SmartCAR
  - Ultrasonic sensors

- Next week, we will learn
  - Infrared sensors
  - Line tracing