EC535 Project - Sensor System for RC Car

Nikolay Popov, Bailey Brake

ECE Department, Boston University

[1]nikyp@bu.edu

[2]bbrake01@bu.edu

***Abstract*— This project is a proof of concept for a dedicated sensor system for an RC car. It utilizes 3 sensors, those being 2 I2S microphones and an ultrasonic distance sensor, all connected to a BeagleBone Black. The output for this model is simply 2 LEDs, 1 corresponding to each microphone. In a future model, this output would be translated to the motors of the RC car using a connected navigation system.**

***Keywords*— ALSA, BeagleBone Black, device tree, embedded systems, I2S, McASP, RC car, sound localization.**

***GitHub Link*— github.com/bbrake01/535-group/tree/main/Project**

I. Introduction

This project presents a foundational exploration into sensor-based navigation for remote-controlled (RC) cars, leveraging sound localization and obstacle detection. The primary objective is to develop a proof of concept for an embedded sensor system that utilizes a combination of acoustic and distance sensing technologies to guide an RC car towards a designated sound source while avoiding obstacles.

At the heart of this system is the BeagleBone Black, which manages three sensors: two I2S microphones and an ultrasonic distance sensor. These sensors are employed to discern the direction of sound and detect physical obstacles, respectively. The sensory information is visually represented through two LEDs, which indicate the status of the sensors.

The ultimate vision for this initiative is to integrate this sensory system with a navigational module that directs the RC car's motors, enabling autonomous movement towards sound sources. The goal of the system is to demonstrate basic directional sound localization and immediate obstacle detection. When a sound is emitted, it is identified as originating from either the left or right, triggering the corresponding LED. Movement of the sound source results in real-time switching of the LEDs. Simultaneously, the presence of an obstacle activates an interrupt, disabling the microphone indicators and lighting up both LEDs to signal obstacle detection.

II. Technology Stack

This section details the essential hardware and software components utilized in the development of the sensor system for the RC car. See Section III. Technical Details for an extended explanation.

A. Hardware

The core of the hardware assembly is the BeagleBone Black board, which orchestrates the operation of the sensory components. The system employs two Adafruit I2S MEMS Microphones (SPH0645) for acoustic signal capture and one HC-SR04 ultrasonic distance sensor for proximity detection. Visual feedback from the sensors is facilitated through two LEDs. The circuit also integrates a voltage divider using one 10 kΩ and one 20 kΩ resistor, alongside a variety of jumper wires to establish the necessary connections between components.

B. Software

On the software front, the system's functionality is driven by a custom Linux kernel configuration. This involves a modification to the device tree via the inclusion of a .dtsi file into the primary .dts device tree source file, which is subsequently recompiled into a .dtb (Device Tree Blob) file (see Section IV. for more details). Additionally, a custom .c kernel module was developed to handle the operational logic of the sensor system, encapsulating the control of input signals and response mechanisms.

III. Technical Details

A. Software Architecture

The software architecture of the sensor system is centered around a robust kernel module designed to manage the real-time operations of the embedded sensors. This module is configured to initiate upon system startup and functions continuously via a recurring kernel timer, ensuring consistent performance and responsiveness.

The module utilizes a kernel timer set at a defined interval, catering to different operational demands. Each timer expiry triggers the `timer_callback` function, which is critical for initiating the measurement cycle of the ultrasonic sensor and sound localization.

In the `timer_callback` function, the `TRIG_PIN` of the HC-SR04 ultrasonic sensor is set high to initiate the measurement process. The `TRIG_PIN` is maintained high for exactly 10 microseconds, facilitated by the `udelay(10);` function, to ensure a uniform ultrasonic burst is emitted. Immediately afterwards, the `TRIG_PIN` is set low, concluding the pulse and emission phase.

Echo detection is managed through the `echo_isr` (Interrupt Service Routine), which begins when the `ECHO_PIN` goes high as the sensor starts to receive reflected ultrasonic waves. This moment is captured as `echo_start` by the `echo_isr`. The `ECHO_PIN` remains high while the waves are being received, and once the reception ends, the `ECHO_PIN` is set low, marking `echo_end`. The time duration between echo_start and `echo_end` is calculated, corresponding to the distance traveled by the waves.

The distance to an object is computed using the formula:

$$distance = (time\_elapsed * speed\_of\_sound / 2)$$

where time_elapsed is the duration for which the `ECHO_PIN` was high, representing the time taken for the ultrasonic pulse to travel to the object and back, and the speed of sound is considered to be approximately 34000 cm/s. The division by two accounts for the round trip travel of the sound waves.

Upon identifying an obstacle (if the calculated distance falls below a preset threshold), the system triggers an obstacle interrupt, lighting up both LEDs in short, half-second pulses. It is within this routine that the rover would initiate obstacle avoidance procedures. The system remains in this mode until the obstacle is cleared. Afterward, it returns to sound localization, lighting up the LED corresponding to the directout of the source. Again, it is in this routine the RC car would be directed towards the sound source. This cycle of detection, obstacle handling, and navigation is continuously driven by the kernel timer, ensuring ongoing monitoring and adaptive response to environmental changes.

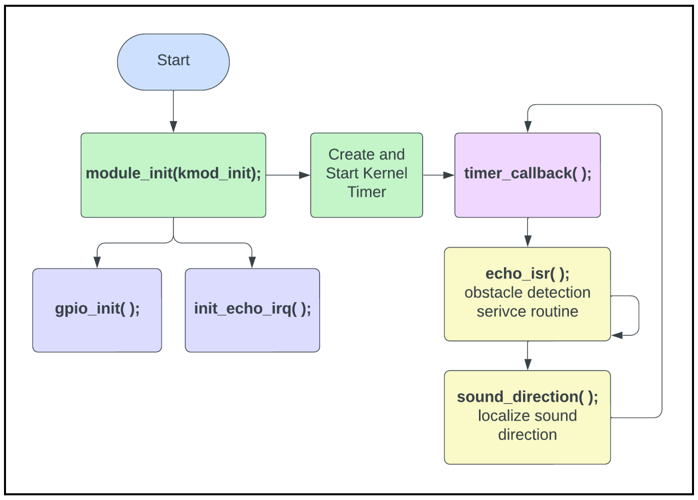
Start
module_init(kmod_init);
Create and Start Kernel Timer
timer_callback( );
gpio_init( );
init_echo_irq( );
echo_isr( ); obstacle detection serivce routine
sound_direction( ); localize sound direction

Fig. 1 Diagram detailing a high level overview of the software architecture and logic flow

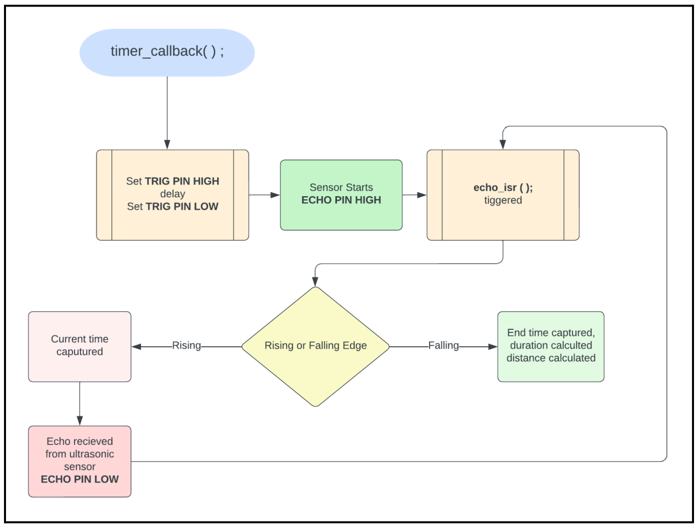
timer_callback( ) ;
Set TRIG PIN HIGH delay Set TRIG PIN LOW
Sensor Starts ECHO PIN HIGH
echo_isr ( ); tiggered
Rising or Falling Edge
Rising
Falling
Current time caputured
End time captured, duration calculted distance calculated
Echo recieved from ultrasonic sensor ECHO PIN LOW

Fig. 2. Detailed diagram of interrupt service routine used with the HC-SR04 for obstacle detection

B. Hardware Architecture

Each LED is plugged into a dedicated GPIO pin in series with a small resistor. These pins are not typically able to drive anything, but produce sufficient power for a small LED even with logic voltage.

The HC-SR04 Ultrasonic Sensor, pictured below, is used to measure distance in order to implement obstacle avoidance. Fig. 2 shows the pinout of the device, which are connected to the BeagleBone Black as described below:

- Vcc connects to 5V. GND connects to GND.
- Trig sends a pulse to tell the sensor when to transmit data. In order to facilitate this small pulse, it connects to GPIO_51.
- Echo connects to GPIO_48. Since the sensor is a 5V device and the BBB uses 3.3V logic, a simple voltage divider using a 10kΩ and a 20kΩ resistor was used to step down the voltage and protect the BBB.

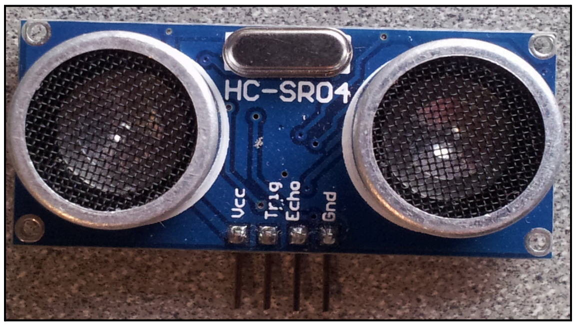

Fig. 3 HC-SR04 Ultrasonic Range Sensor with labeled pins[6]

The BeagleBone Black supports McASP protocol (Multichannel Audio Serial Port). In this protocol, I2S is a certain mode, notably not requiring a master clock. I2S supports 2 devices on the same bus for stereo sound. Fig. 3 shows the MEMS I2S microphone used in the design.

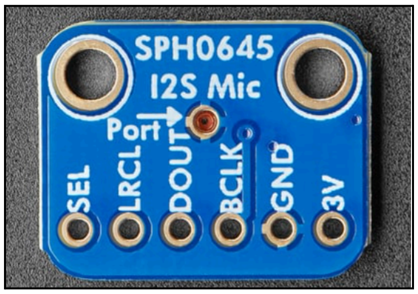

Fig. 4 I2S MEMS microphone with labeled pins[5]

Below is a description is how each pin is wired to the BeagleBone Black and the wider system:

- SEL: Set to either Vcc or GND to distinguish between right/left microphone, respectively.
- LRCL: Otherwise known as Word Select, when high/low it tells right/left to transmit data. This connects to FSR0.
- DOUT: Serial data port. This connects to the AXR0 port on the BBB. Only one microphone will transmit data at a time, thus they can use the same bus. However, I2S only supports stereo transmission, whereas the broader McASP supports more connections.
- BCLK: Bit clock, determines bit rate of serial data transmission. This connects to ACLKR.
- 3V connects to 3.3V, and GND to GND.

Fig. 4 shows a generalized McASP bus model, with each microphone serving as an ADC clock follower. This diagram was used to identify and translate between the microphone and BBB pins.

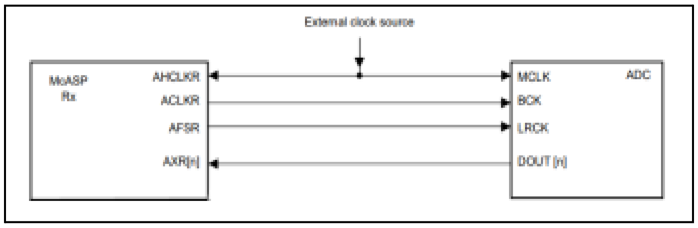

Fig. 5 McASP protocol with ADC as clock follower[1]

C. *Final Pinout*

Device	*Device Pin*	*Physical Pin*	*Pin Mode*
Microphone L	SEL	P9_1	GND
Microphone R	SEL	P9_3	VDD_3V3
Microphone L/R Combined Busses	LRCL	P9_27	mcasp0_fsr
	DOUT	P9_41B	mcasp0_axr1
	BCLK	P9_42B	mcasp0_aclkr
Ultrasonic Sensor	Trig	P9_16	GPIO_51
	Echo	P9_15	GPIO_48
LED L	+	P9_23	GPIO_67
LED R	+	P9_25	GPIO_68

Fig. 6 System pinout table, not including Vcc/GND

IV. Discussion of Revisions & Success

A. *Challenges & Failures*

Throughout our project, we encountered several significant challenges that impacted both our planned processes and methodologies. These challenges were primarily centered around hardware configuration, software limitations, and tedious protocol implementation on the BeagleBone Black.

One major issue was our inability to utilize the cape manager utility of the BBB, which significantly hindered our attempts to configure the hardware to our specifications. Our initial strategy involved converting the device tree binary (.dtb) to its source code and manually incorporating our edits. However, we faced persistent compilation errors, primarily because we were not compiling directly on the Beagle Bone's filesystem, which would have provided access to all necessary include files. This complication arose due to the absence of the device tree compiler utility on the BeagleBone, forcing us to perform these operations on our machines. Unfortunately, after making the necessary adjustments to the source code, we were unable to recompile it back into a binary format compatible with the BeagleBone.

Without a functional device tree, the next technical hurdle was implementing the I2S protocol via the McASP pins on the BBB equipped with an AM335x processor. Our goal was to configure the McASP for audio reception by setting the clock to 2 MHz, enabling internal frame sync, and configuring the serializer for I2S mode. This setup was intended to detect audio signals above a certain threshold without the need for audio storage or playback. Despite careful setup, loading the kernel module resulted in a kernel panic characterized by an "error unhandled fault: imprecise external abort," indicating potential issues with accessing unmapped memory addresses or incorrect memory mapping. We managed the configuration of the McASP registers directly in the kernel module code, using memory-mapped addresses and offsets such as MCASP_BASE and MCASP0_AXR1_OFFSET. Despite the significant effort put towards deriving safe and usable addresses, we encountered a kernel panic during each iteration testing, suggesting further issues with address mapping alignment or uninitialized memory accesses.

As an alternative to the I2S implementation, we also explored using analog microphones interfaced through a user space program via the BBB's AIN0 and AIN1 pins. However, this approach also stumbled due to similar device tree challenges, as enabling the BBB's ADC driver required loading an appropriate device tree overlay—a task complicated by our inability to use the cape manager. Overall, a more feature-rich BeagleBone Black image would have saved us a significant amount of time and resulted in a more presentable product.

B. Adjustments & Changes

The RC car was not available, so we were forced to switch over to LED displays and focus solely on the detection system. This shrinking scope ultimately allowed us to make a functional system, albeit at the expense of a more visually and practically appealing final product.

The scope of the microphone array scaled back considerably due to technology and time constraints, and ultimately

While we initially intended to use an array of 4 microphones to perform robust sound localization, the microphones we utilized were configured to the I2S protocol, which only supports stereo sound, meaning 2 inputs only.

C. Successful Implementations

While our project's scope scaled back considerably throughout our work cycle, we emerged with a robust ultrasonic sensor implementation. Given that the crutch of the project (that being the microphone array) was unsuccessfully implemented, we were forced to demo what we did have, that being our control flow and obstacle detection. A button replaces the microphone input, which turns on the associated LED. Then when the ultrasonic sensor detects an object too close, it will interrupt the microphone routine and begin flashing both LEDs. As the object gets closer, the LEDs will pulse faster to get the driver's attention.

We managed to correctly prioritize object detection over sound localization. This prioritization ensured our system would react appropriately to immediate environmental inputs while still performing its primary function of sound localization when conditions allowed.

D. Technical Skills Learned

Throughout the duration of this project, we both acquired several critical technical skills essential for embedded systems development, particularly in the context of utilizing the BeagleBone Black platform.

Interfacing with GPIO Pins: We developed proficiency in interfacing directly with GPIO pins through a kernel module. This was crucial for controlling the ultrasonic and microphone sensor, as detailed in our kernel module.

Pin Multiplexing: We learned to set pin modes through direct manipulation of hardware control registers within kernel space, which allowed for precise control over hardware's function. This was done through the `set_pin_mode` function.

Interrupt Service Routines: We learned how to register and trigger interrupt service routines (ISRs) via GPIO pins. This was integral in handling real-time changes in sensor data, allowing immediate responses to environmental inputs.

Kernel Timers: We effectively implemented kernel timers to schedule future operations at specific intervals. This functionality was applied to periodically poll the sensor data and manage system states effectively from within kernel space, avoiding the overhead of user space processes.

Device Tree Modifications: Another significant learning area was in the modification of device trees. Although initially intending to use device tree overlays with the BeagleBone's cape manager functionalities, differences in our Linux distribution led us to a more direct approach. We modified the device tree (.dtb) by converting it back to a device tree source (.dts) file using a device tree compiler. We then created and included a device tree include (.dtsi) in the .dts file to reconfigure the hardware settings on the board, which was then compiled back into binary format using the device tree compiler. This process is also crucial for ensuring hardware is configured correctly to support a specific sensor system.

These skills helped deepen our understanding of embedded systems, particularly in real-time data handling and device configuration under constraints.

References

[1] B. Tufino, "McASP Design Guide -Tips, Tricks, and Practical Examples Application Report McASP Design Guide -Tips, Tricks, and Practical Examples," 2019. Accessed: Apr. 30, 2024. [Online]. Available: https://www.ti.com/lit/an/sprack0/sprack0.pdf?ts=1713467794664

[2] "ALSA project - the C library reference: Index, Preamble and License," www.alsa-project.org. https://www.alsa-project.org/alsa-doc/alsa-lib/ (accessed Apr. 30, 2024).

[3] "Setting Up the BeagleBone Black's GPIO Pins," vadl.github.io. https://vadl.github.io/beagleboneblack/2016/07/29/setting-up-bbb-gpio (accessed Apr. 30, 2024).

[4] P. Sharma, "How to interface an I2S microphone with Beaglebone Black(BBB)," Mantra Labs, Jul. 12, 2018. https://www.mantralabsglobal.com/blog/how-to-interface-an-i2s-microphone-with-beaglebone-blackbbb/ (accessed Apr. 30, 2024).

[5] "Adafruit I2S MEMS Microphone Breakout," Adafruit Learning System. https://learn.adafruit.com/adafruit-i2s-mems-microphone-breakout/

[6] "Sensors — BeagleBoard Documentation," docs.beagleboard.org. https://docs.beagleboard.org/latest/books/beaglebone-cookbook/02sensors/sensors.html (accessed Apr. 30, 2024).

[7] "AM335x and AMIC110 SitaraTM Processors Technical Reference Manual." Available: https://www.ti.com/lit/ug/spruh73q/spruh73q.pdf

Github link: https://github.com/bbrake01/535-group/tree/main/Project