Matthew's Fenceless Grazing Tech Review

Matthew Sessions
Capstone Fall 2019
CS3 - Fenceless Grazing

Abstract

The Fenceless Grazing System uses LoRa and LoRaWAN technologies to increase productivity in animal farming. This system must be capable of monitoring and controlling a high volume of farm animals. To implement this system we must control hardware with software, which requires some kind of driver code. We will use a micro-controller that we program with software to drive the hardware that is part of the collar to administer electric shocks, and monitor location. To drive such a number of wireless devices with sufficient range we will need to implement a LoRa network. Our project should result in a usable system that allows the monitoring and control of cattle that can be used by farmers, without requiring technical expertise.

CONTENTS

I. My Role

As the member of the team experienced in lower-level programming of hardware I will be most involved in software to hardware interfaces. Driving our hardware will be the bulk of my role, with the rest being the integration of proper hardware peripherals that will be useful to our system.

II. The Team's Goal

The Fenceless Grazing Collar System is an application of the LoRa and LoRaWAN internet of things technologies to animal farming. It will be implemented as a collar with which each individual farm animal will be equipped. The fenceless grazing system must be capable of precisely tracking high numbers of farm animals, controlling these animals using sound and electric shocks, and allowing end users (typically farmers) to monitor the animals and adjust their desired location.

The purpose of the Fenceless Grazing Collar (FGC) system is to reduce the need for human supervision in farming through the tracking and automated management of individual farm animals, controlled by humans through a remote digital system. The FGC will decrease the time required to move animals to new grazing areas by allowing farmers to manage these animals from the comfort of their home. Interaction with the fenceless system will be done via a smartphone application, and will allow complete control and surveillance of all animals.

III. Personal Responsibilities

A. Connections

For our slow-speed connected peripherals such as the buzzer and shocker, we will use I2C to connect them to our micro-controller. Using I2C for such devices will be good as it allows for a lot of devices without much extra program logic. Small micro-controllers such as the Attiny85 can be used for peripherals that need to be converted over to I2C.

For higher speed peripherals we will be required to use SPI. This will require a little extra program logic to enable/disable devices that are connected, however we need SPI to drive the Lora radio and possibly the GPS.

B. Embedded Drivers for Collar & Collar Peripherals

In order to separate responsibilities between program logic and driver we need to write drivers. In software the driver is a program that controls some piece of hardware. In our case the driver will take the form of functions that the program logic will call to control connected hardware.

1) Platform: A simple embedded architecture that is well known and widely used by hobbyists is AVR. Boards that use AVR are easy to acquire and will be more than powerful enough for our project. While we could use an ARM board that is already built with the necessary Lora radio for about the same price, we have no compelling reason to not use AVR. Microchip(Atmel) makes using AVR especially easy by providing thorough documentation and fully featured Arduino ports.

2) Language & Compiler: Most hobbyist projects that use micro-controllers are going to use Arduino, which means using the Arduino IDE and an Arduino compatible bootloader. The language used by the Arduino IDE is C++ and includes an extensive library of functions and classes that are useful for general purpose embedded projects.

However, for our project we will be using bare C and writing everything that we need from scratch. Doing this will give us complete control of the micro-controller and allow us to make all implementation choices specific to our micro-controller without hacking around an existing framework.

3) Design: Having used many frameworks, a decent amount of thought will be put into design. We need a driver framework that is intuitive, yet completely fulfills the requirements of our application. And we cannot afford to make the framework overly general, or we'll end up with code that is unreadable.

C. Debugging Libraries

While dedicated hardware devices such as the AVR Dragon would be nice to help us debug our project, we will opt for sending debug information over serial. Though debugging over serial is not as fully featured as using a real debugger, it will be enough to catch errors that we may not catch in the simulator. One big part of the debugging library is that it has to be implemented asynchronously so that it won't conflict with the time-sensitive functionalities of the program, such as SPI or I2C.

D. Lora Gateway

The Lora Gateway is the device that receives information from our collars, this gateway needs to be able to handle many requests per second and upload this information to our server. The Gateway will also need to be able to send information back out to our collars, as we need to control them somehow.

1) Platform: While we could use the same board as what we'd use for the collar, we may not be able to connect enough collars to such a low powered device. To compensate for this we are thinking of using a Raspberry Pi with a Lora shield for managing all of our collars. The Raspberry Pi has a significantly higher clock speed of about 1-2 GHz and the common atmega328p board on Ebay has a frequency of only 8MHz.

2) Language: Realistically we will use an existing Python library on the Raspberry Pi to drive the gateway. This will minimize the amount of confusion on such a complex system, while we could drive the gateway with C through a user program or a operating system driver this may not be feasible if the Python library can handle enough collars.

E. Powering Devices

Collars need to be fully wireless, therefore they need to be powered using batteries. The battery type that is widely used in many development boards is a 14500 Lipo which is what we will use. The gateway does not need to be wireless, if the gateway needs to be portable we can power it using a USB power bank.

1) Power Saving for AVR: Basic embedded programs will include some kind of infinite loop to keep the program running, however this will waste a lot of power. We can use software to make our devices go into a lower power state instead of executing in this loop. One of the interrupt pins on the AVR board will be used so that the Lora radio can wake it back up. Through software, we can also turn off connected peripherals that do not need to be on.

2) Power Saving for Raspberry Pi: It is easy to put the AVR board into a low power state where it can be woken up by our Lora radio. However on the Raspberry Pi it is not as simple, nor is it necessary to use lower power states. On the Raspberry Pi we'll probably disable some unneeded features such as the build-in bluetooth and wifi.

IV. Technologies

A. Micro-controller

A micro-controller is like a mini computer, however it'll have everything necessary to function built into one sealed chip. This chip does not directly expose the CPU but rather exposes pins that can be used to program the chip, control other devices, and sometimes connect to USB.

B. AVR

A 8-bit architecture used in micro-controllers. Also one of the earliest micro-controllers to allow for multiple writes to memory. The use of AVR in this project is driven mostly by familiarity, which will save time. There are plenty of other technologies that could be used in place of AVR, however AVR fulfills our needs and is enough to complete the project.

C. SPI

From Sparkfun "Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.".

SPI uses four wires at a much higher speed than I2C. Devices have to be enabled/disabled by writing to a designated pin to receive data. The top-end speed of SPI is in the millions of bytes per second. On most micro-controllers there are designated pins dedicated to SPI.

D. I2C

From Sparkfun "I2C requires a mere two wires, like asynchronous serial, but those two wires can support up to 1008 slave devices. Also, unlike SPI, I2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus (although the master devices can't talk to each other over the bus and must take turns using the bus lines).".

I2c uses two wires to connect devices to each other, in our case the main device will be the AVR micro-controller and the connected devices will be our sensors. Devices connected over I2C have an address that is used to determine if they should read information. I2C could be especially useful if we need to dedicate a micro-controller to controlling an I2C device, since I2C allows for multiple master devices.

E. LoRaWAN

LoRaWAN is an open spec that makes long range battery operated internet connected devices doable. LoRaWAN uses LoRa technology and the LoRa Alliance is an organization that standardizes the use of LoRa. We are using LoRa because it is the most feasible means of controlling so many collars that need to be battery operated for long periods of time. Given LoRa is commonly used in this kind of application it made most sense to use, even if we haven't used it before. Using LoRa also doesn't require any radio licensing which makes it especially convenient, "LoRaWAN protocol leverages the unlicensed radio spectrum in the Industrial, Scientific and Medical (ISM) band." (What is LoRa).

REFERENCES

[1] bobology.com https://www.bobology.com/public/What-is-a-Software-Driver.cfm 2019
[2] wikepedia.org https://en.wikipedia.org/wiki/AVR_microcontrollers Accessed 2019
[3] quora.com https://www.quora.com/Are-AVR-microcontrollers-used-frequently-in-industry 2016
[4] electronics.stackexchange.com https://electronics.stackexchange.com/questions/383245/avror-pic-vs-arm-for-lorawan-nodes 2018
[5] arduino.cc https://forum.arduino.cc/index.php?topic=65853.0 2011
[6] microchip.com http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Accessed 2019
[7] wikepedia.org https://en.wikipedia.org/wiki/LoRa Accessed 2019
[8] ebay.com https://www.ebay.com/sch/i.html?_from=R40&_trksid=p2047675.m570.l1313.TR0.TRC0.A0.H0.Xlora+atmega.TRS1&_nkw=lora+atmega&_sacat=0 Accessed 2019
[9] A DIY low-cost LoRa gateway http://cpham.perso.univ-pau.fr/LORA/RPIgateway.html Accessed 2019
[10] SX12756 https://www.mouser.com/datasheet/2/761/sx1276-1278113.pdf Accessed 2019
[11] Learn SPI https://learn.sparkfun.com/tutorials/i2c/all Accessed 2019
[12] Learn I2C https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all Accessed 2019
[13] What is LoRa https://www.semtech.com/lora/what-is-lora Accessed 2019
[14] What is AVR Microcontroller https://www.kanda.com/blog/microcontrollers/avr-microcontrollers/avr-microcontroller/ Accessed 2019

[15] AVR debugging using gdb and AVR Dragon https://www.avrfreaks.net/forum/tutsoftgcc-avr-debugging-using-gdb-and-avr-dragon-under-linux-ubuntu Accessed 2019
[16] buserror/simavr https://github.com/buserror/simavr Accessed 2019