Computer Controlled Eddy Current Brake Bicycle Trainer

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Computer Science

in the

College of Graduate Studies at

University of Idaho

by

Kathryn Price

Major Professor: Axel Krings, Ph.D.

Committee Members: Robert Rinker, Ph.D.; Terry Soule, Ph.D.

Department Administrator: John Crepeau, Ph.D.

December 2017

Authorization to Submit Thesis

This thesis of Kathryn Price, submitted for the degree of Master of Science with a major in Computer Science and titled **"Computer Controlled Eddy Current Brake Bicycle Trainer,"** has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor:	______________________ Axel Krings, Ph.D.	Date	________
Committee members:	______________________ Robert Rinker, Ph.D.	Date	________
	______________________ Terry Soule, Ph.D.	Date	________
Department Administrator:	______________________ John Crepeau, Ph.D.	Date	________

Abstract

Dynastream Innovations developed a proprietary multi-cast network protocol called Advanced and Adaptive Network Technology (ANT). This protocol has gained popularity for applications in the exercise industry. Bicycle smart trainers can now use a prescribed workout and feedback communicated by ANT to precisely control resistance.

This research builds upon the ANT protocol to create a proof of concept bicycle trainer that controls resistance by use of an eddy current brake on the bicycle's aluminum rear wheel rim. The eddy current brake does not touch the bicycle so it provides minimal wear to the rider's gear. The trainer uses a pre-programmed workout and the rider's power feedback to adjust the trainer's resistance. The power feedback is achieved via the ANT protocol, while the trainer control and communication processing is provided by a Microcontroller Unit (MCU).

Acknowledgments

I want to thank my major professor, Dr. Axel Krings, for his support, encouragement, and knowledge during all aspects of this thesis research. His guidance was key to my thesis research selection and his kind encouragement helped me throughout the process. Dr. Krings and my committee members, Dr. Robert Rinker and Dr. Terry Soule, all provided insightful feedback and thoughtful questions for me to consider. I thank them all.

I am indebted to the administrative people in the College of Graduate Studies. In particular, Kathy Duke and Lana Unger showed much patience and understanding while working with me. As an Engineering Outreach student in Alaska, it was sometimes difficult to find and deliver all the necessary paperwork and they made it much easier.

Finally, I also wish to thank my friend, Marcia Bird, M.A., for her help proofreading this thesis.

Dedication

My thesis is dedicated to my husband, whose love and support have helped me to achieve my dreams.

Table of Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

A	Amperes
ANT	Advanced and Adaptive Network Technology
ANT+	Advanced and Adaptive Network Technology Plus
ANT-FS	Advanced and Adaptive Network Techology File Share
API	Application Programming Interface
ARM	Advanced RISC Machine
cm	centimeters
CMSIS	ARM Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
dBm	decibel-milliwatts
EMF	Electromotive Force
GFSK	Gaussian Frequency Shift Keying
GHz	gigahertz
GPIO	General Purpose Input Output
Hz	hertz
ID	identification
IDE	Integrated Development Environment
ISM	Industrial, Scientific, and Medical
kbps	kilobits per second
MCU	Microcontroller Unit
MHz	megahertz
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
mph	miles per hour
mA	milliampere
OSI	Open Systems Interconnection
PAN	Personal Area Network
PWM	Pulse Width Modulation

RF	Radio Frequency
RISC	Reduced Instruction Set Computer
RPM	Revolutions per Minute
RX	Receiver
SDK	Software Development Kit
SoC	System on a Chip
SoM	Systems on Module
Tch	Channel Period
TDMA	Time Division Multiple Access
TX	Transmitter
ULP	Ultra Low Power
Unistrut	Unistrut Metal Framing System
USB	Universial Serial Bus
VAC	Voltage of Alternating Current
VDC	Voltage of Direct Current
W	Watt

Chapter 1

Introduction

Competitive cycling requires specific workouts to ensure maximum benefit from precise physical training [2]. Many times it is easier to control the workout variables, such as intensity and duration, doing an indoor workout because the indoor workout removes the weather and road conditions as factors in the workout exercise [1].

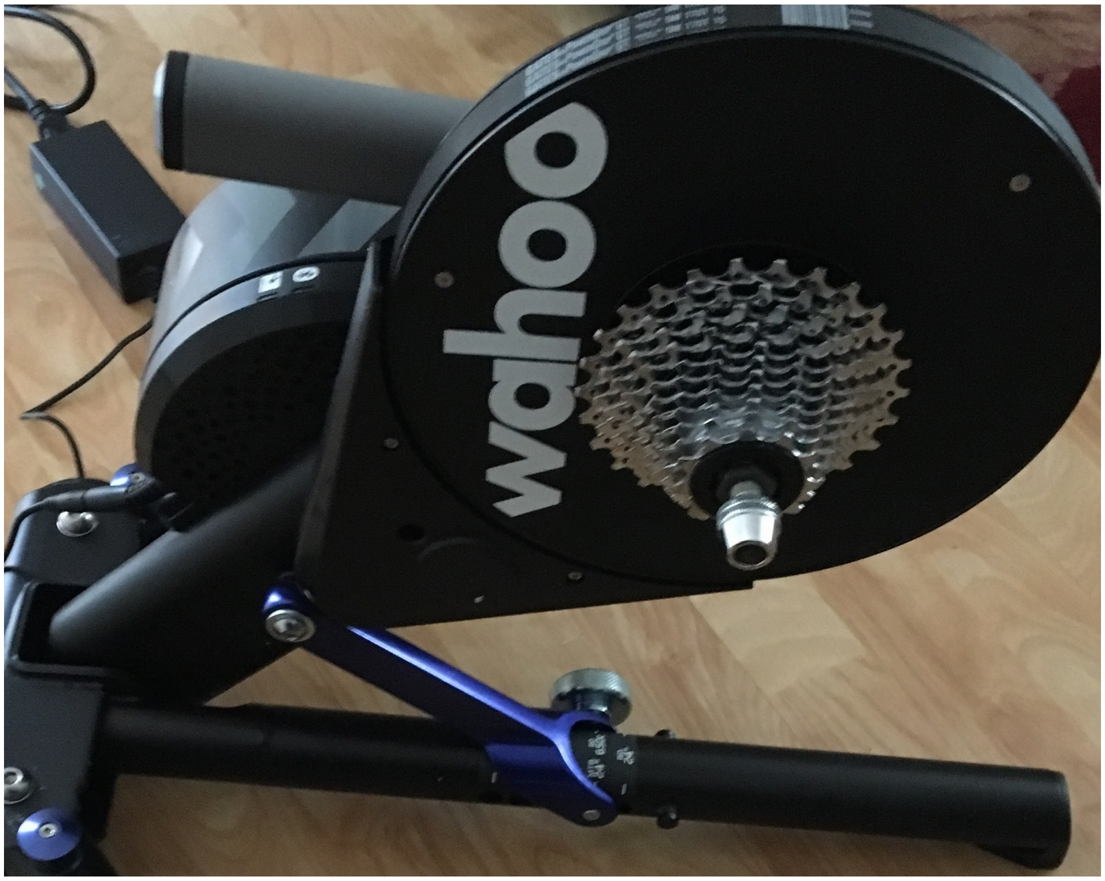


Figure 1.1: Wahoo Fitness KICKR Smart Trainer[1]

For indoor cycling there are a variety of indoor trainers, but they fall into two categories; basic and smart. Basic trainers use a resistance that cannot be modulated externally, for example with the use of fluid or wind resistance. Smart trainers use an external signal to modulate the resistance level of the training session. This modulated resistance is often implemented using an eddy current braking system. Through a system of belts or gears, a small conductive

[1]Unless otherwise noted, all photos and figures by Kathryn Price.

disk is spun between a number of electromagnets the current to which are controlled through an external workout signal.

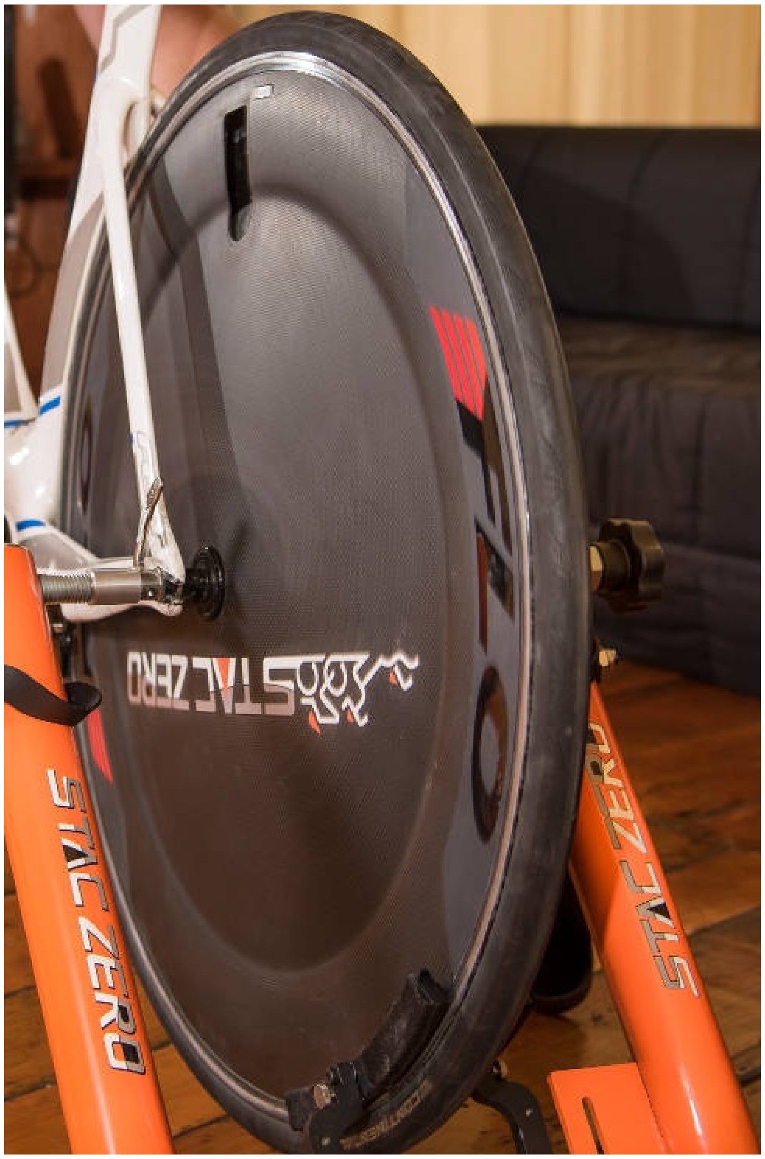


Figure 1.2: STAC Zero Trainer (Source: [4])

Most trainers that allow a bicycle to be attached either use the bicycle's rear wheel to spin a cylinder that is in turn attached to a resistance, or the bicycle's rear wheel is removed altogether and the chain is attached to a cog set on the trainer. An example of the cog set type of trainers is shown in Figure 1.1. In either case, moving an outdoor bicycle onto a trainer or from a trainer to be outdoor ride ready is an awkward task. For instance, the trainer requiring a tire will add wear to the tire and demands that the tire be maintained at the proper pressure for correct frictional force on the cylinder. The benefit of this type of trainer, however, is that the bicycle is road ready at a moments notice. For the chain driven cog set type of trainer, there is obviously no tire wear; however, a cog set is needed for the trainer. This means installing the correct cog set onto the smart trainer and connecting the bicycle via the chain each time it is used. Conversely, to ride the bicycle outside, the cog would have to be reinstalled on the rear wheel and then the wheel mounted on the bicycle again.

A trainer that has tried to overcome both the awkward bicycle mounting and the tire wear is the STAC Zero trainer [4]. This trainer has the bicycle held by the rear wheel skewer, but the wheel can spin freely - that is there is no cylinder on which to apply frictional force. Instead, there are a series of rare earth permanent magnets mounded along each side of the aluminum rim of the rear wheel as in Figure 1.2. These magnets induce a corresponding eddy current as the aluminum rim rotates through the magnet banks [4] [3].

1.1 Exercise Technologies

There is an ever-expanding world of personal sensors for fitness. In particular for cycling, specific sensors are used for such things as heart rate, power, cadence, speed, and ride mapping. All of the data needs to be collected via a wireless Personal Area Network (PAN). Figure 1.3 shows an overview of how these sensors might be arranged and connected on a bicycle. Furthermore, there is a need to control exercise equipment itself in order to facilitate specific directed workouts. Much of this wireless communications is done through the proprietary protocol ANT. A number of independent companies in the fitness industry have adopted ANT protocol for sensor and smart trainer communications [6].

1.2 Motivation

The motivation for this thesis stems from the desire to overcome the awkwardness of indoor bicycle trainers that require either partial disassembly of the bicycle or wear and tear on the bicycle itself, and to combine that with the power of a smart trainer. The STAC Zero bicycle trainer is an example of overcoming the physical restraints of indoor bicycle trainers mentioned above. The next logical step is to incorporate the use of eddy current braking on a rear bicycle wheel using electromagnets. This would allow an externally controlled workout that controls the resistance, and it does so with minimal wear on the bicycle, as well as leaving it nearly road ready for outdoor riding.

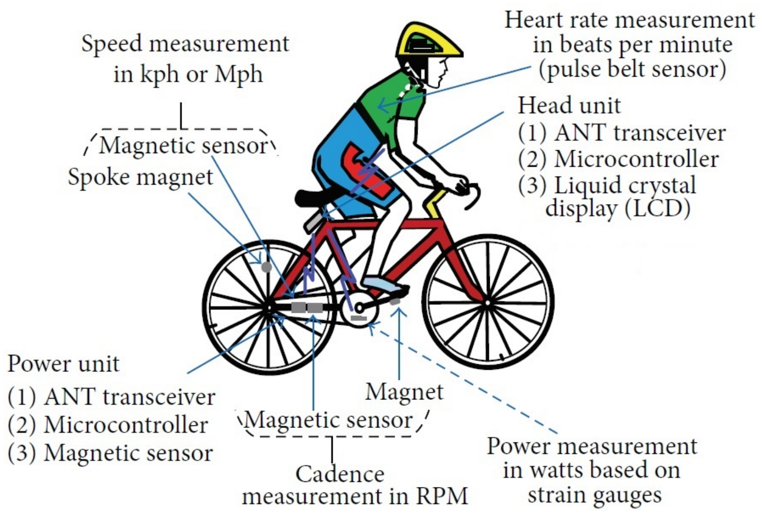


Figure 1.3: Connected sensors on a bicycle (Source: [5])

1.3 Contribution

This thesis project showed that it is possible to have an eddy current brake indoor bicycle trainer that has minimal set-up for the bicycle and practically no wear on the bicycle itself. The trainer was externally controlled by a preprogrammed workout. A proof of concept was built and tested for this type of trainer.

1.4 Thesis Outline

This thesis is dedicated to developing an indoor bicycle trainer that will use electromagnets on a rear wheel to control trainer resistance. The remainder of this thesis is organized as follows:

Chapter 2, *Background*, introduces the ANT protocol and the ANT+ messaging structure. Finally, a discussion of the basics of eddy current brakes and power calcuations are introduced.

Chapter 3, *System Architecture*, presents the overall structure of the trainer set-up. This includes the design of the eddy current brake and the control of power to the electromagnents. In addition, the software overview is discussed, as well as is any hardware needed to implement the software on each MCU.

Chapter 4, *Experimental Validation of Computer Controlled Eddy Current Brake Bicycle Trainer*, provides an overview of the Computer Controlled Eddy Current Brake Bicycle Trainer prototype. The data from the trainer testing are presented here.

Chapter 5, *Conclusion and Future Work*, reviews the results of the trainer testing and provides some ideas for future improvements and work.

A number of appendices are provided for clarity of the Computer Controlled Eddy Current Brake Bicycle Trainer. Appendix A, *Bill of Material*, provides a list of items used to build the prototype. It does not include items such as fasteners. Appendix B, *ELPAC Power Supply Wiring,* shows how to wire the ELPAC power supply for the Computer Controlled Eddy Current Brake Bicycle Trainer usage. Appendix C, *Arduino Relay Control*, shows the Arduino IDE and lists the C code for controlling the power relay switches. Appendix D, *Programming Software*, shows examples of the software used to program the Systems on Module (SoM) nRF51 development kit. Appendix E, *nRF51 Code*, gives the C code for the nRF51 Development Kit that is used for the trainer controller, while Appendix F, *ANT+ License* give the licensing information from Nordic Semiconductor on the use of the example code.

Chapter 2

Background

2.1 Application Overview

The Computer Controlled Eddy Current Brake Bicycle Trainer makes use of two very different technologies. First, it uses a proprietary wireless protocol called ANT to provide power feedback from the bicycle rider to the trainer controller MCU. Second, it uses the concept of eddy currents to create braking action and thus resistance for the rider.

This background section will first cover the ANT protocol in as much detail as possible given that it is proprietary and second, it will delve into the physics of eddy currents, eddy current brakes, power calculations as well as cover where the concept of the Computer Controlled Eddy Current Brake Bicycle Trainer originated. The ANT protocol will be described using the Open Systems Interconnection (OSI) stack model, as seen in Figure 2.1, as a reference. The structure building, component assembly and wiring as well as MCU programming will all be described in Chapter 3.

2.2 ANT Overview

The ANT protocol consists of low cost, low power transceivers that can arrange themselves in an ad hoc network rapidly. The wireless PAN requirements are ease of installation, a simple yet flexible protocol, reliable data transfer, very low cost, and low power that results in reasonable battery life [10].

The ANT protocol is specifically written for wireless PAN sensor systems and similar applications and is proprietary. It was developed by a Canadian company, Dynastream Innovations Inc. ANT implements the media layers and the host transport layer of the OSI stack. Figure 2.1 shows the OSI stack model with the media and host layers identified.

The ANT protocol is managed by ANT Wireless which is a division of Dynastream Innovations Inc. which in turn is owned by Garmin. ANT is specifically developed for low data rate, wireless sensor networks with the following considerations [6].

1. ANT systems are Ultra Low Power (ULP) - the power requirements are designed such that the system can operate on a coin cell for years.

2. ANT systems are highly compact. The protocol uses a small amount of memory.

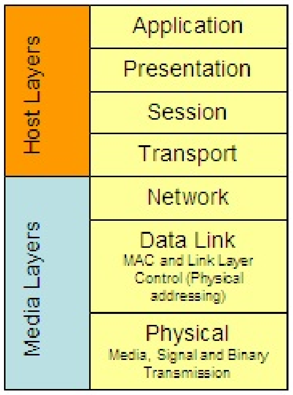


Figure 2.1: OSI Layers (Source: [17])

3. ANT protocol is flexible and can easily scale in terms of network topology.

4. ANT is a low cost system.

The above requirements make it well suited to sports and fitness fields for performance and health monitoring [6].

The ANT protocol provides management of the OSI layers two through four, that is the data-link layer, the network layer, and the transport layer, as shown in Figure 2.2. A 2.4 gigahertz (GHz) radio is used for the physical layer.

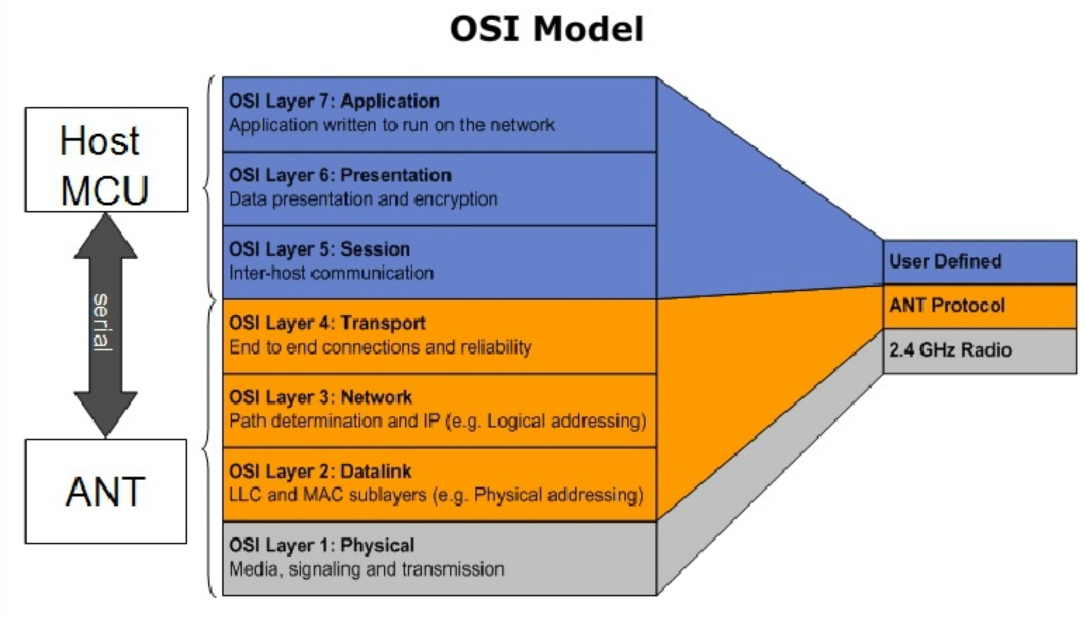
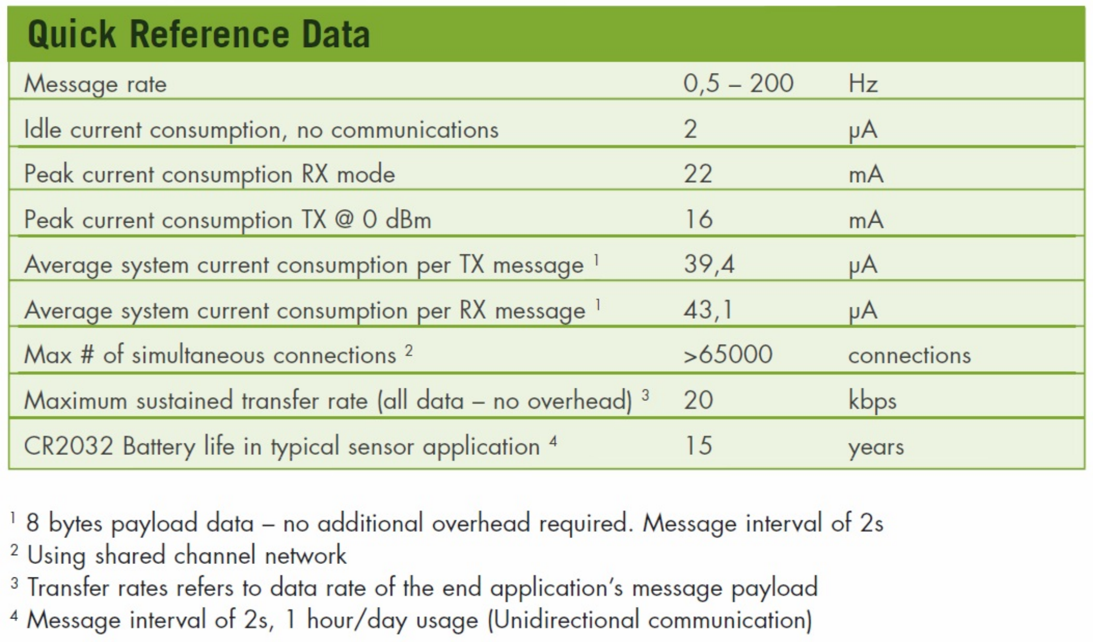

OSI Model
Host MCU
serial
ANT
OSI Layer 7: Application
Application written to run on the network
OSI Layer 6: Presentation
Data presentation and encryption
OSI Layer 5: Session
Inter-host communication
OSI Layer 4: Transport
End to end connections and reliability
OSI Layer 3: Network
Path determination and IP (e.g. Logical addressing)
OSI Layer 2: Datalink
LLC and MAC sublayers (e.g. Physical addressing)
OSI Layer 1: Physical
Media, signaling and transmission
User Defined
ANT Protocol
2.4 GHz Radio

Figure 2.2: ANT OSI Layers (Source: [28])

2.2.1 Radios

The ANT radios operate in the 2.4 GHz Industrial, Scientific, and Medical (ISM) band. Specifically, the frequency range is 2.4 GHz to 2.524 GHz [10]. The radios use Gaussian Frequency Shift Keying (GFSK) which is a type of frequency modulation for digital communications. For example, in frequency shift keying, the carrier signal frequency will increase when a 1 is transmitted and, conversely, it will decrease when a 0 is transmitted. A Gaussian filter is applied to this frequency shift keying in order to create a GFSK modulation scheme. This type of filter makes the transitions smoother while reducing side-band power [16].

ANT transceivers can change channel frequency in special cases, but ANT does not frequency hop. Instead, ANT relies on Time Division Multiple Access (TDMA) in order to avoid collisions from other transmitters in the same band. This means ANT allows different devices to communication on a shared frequency by assigning individual time slots for each channel [22]. The relatively short transmission time slots are at a chosen interval, but the time slots can be adjusted by the transceivers as interference is discovered. Only in special cases can the transceivers can change frequency. For example, if too much interference is detected across the whole channel from another source such as WiFi, the transceivers can change frequency altogether. This is called frequency agility by ANT and differs from hopping because the frequency is only changed when a threshold signal degradation is observed [9].



Quick Reference Data		
Message rate	0,5 – 200	Hz
Idle current consumption, no communications	2	μA
Peak current consumption RX mode	22	mA
Peak current consumption TX @ 0 dBm	16	mA
Average system current consumption per TX message [1]	39,4	μA
Average system current consumption per RX message [1]	43,1	μA
Max # of simultaneous connections [2]	>65000	connections
Maximum sustained transfer rate (all data – no overhead) [3]	20	kbps
CR2032 Battery life in typical sensor application [4]	15	years

[1] 8 bytes payload data – no additional overhead required. Message interval of 2s
[2] Using shared channel network
[3] Transfer rates refers to data rate of the end application's message payload
[4] Message interval of 2s, 1 hour/day usage (Unidirectional communication)

Figure 2.3: An overview of radio power (Source: [18])

An overview of the ANT radio power consumption is given in Figure 2.3. Ultra low power consumption is one of the main goals of the ANT protocol [6]. Figure 2.3 shows that very little current is used for the idle modes and moderate current is used for the active modes. The power consumption for both the transmitter and receiver of the SoC chip, nRF51422, selected for the Computer Controlled Eddy Current Brake Bicycle Traineris lower than what is shown in Figure 2.3. For this specific SoC, the peak transmitter current at 0 decibel-milliwatts (dBm) is 10.5 milliampere (mA) while the peak receiver current is 12 mA [19].

2.2.2 Nodes

As seen in Figure 2.2, the developer defines the top three OSI layers, session, presentation, and application layers, by using a host MCU. The host MCU is external to the ANT protocol engine [10], but they can be packaged together. For instance, the Computer Controlled Eddy Current Brake Bicycle Trainer SoC, nRF51422, has both an ANT protocol engine and a host MCU on the SoC. The ANT engine is proprietary and the design is owned by Dynastream, but the host MCU is an Advanced RISC Machine (ARM) Cortex M0 32 bit processor [19] whose design is licensed by ARM Holdings. In general, though, most MCUs that have a serial connection capability can be used as a host MCU to the ANT engine.

An ANT protocol engine acts as a network processor and an ANT node is defined as a combined ANT protocol engine and a host MCU as seen in Figure 2.4. The ANT protocol engine takes care of establishing and maintaining ANT connections and channel operations. The host MCU takes care of the application needs. It is the host MCU that sets all communication parameters and initiates communications to other nodes.

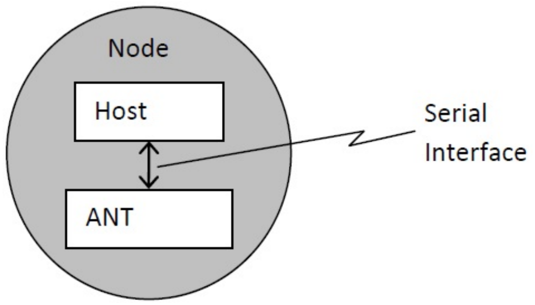


Figure 2.4: A host MCU and ANT engine are considered a node (Source: [10])

The protocol stack for the ANT engine is licensed directly to silicon manufacturers. It is embedded in silicon network processors so that no hardware stack integration is required. This

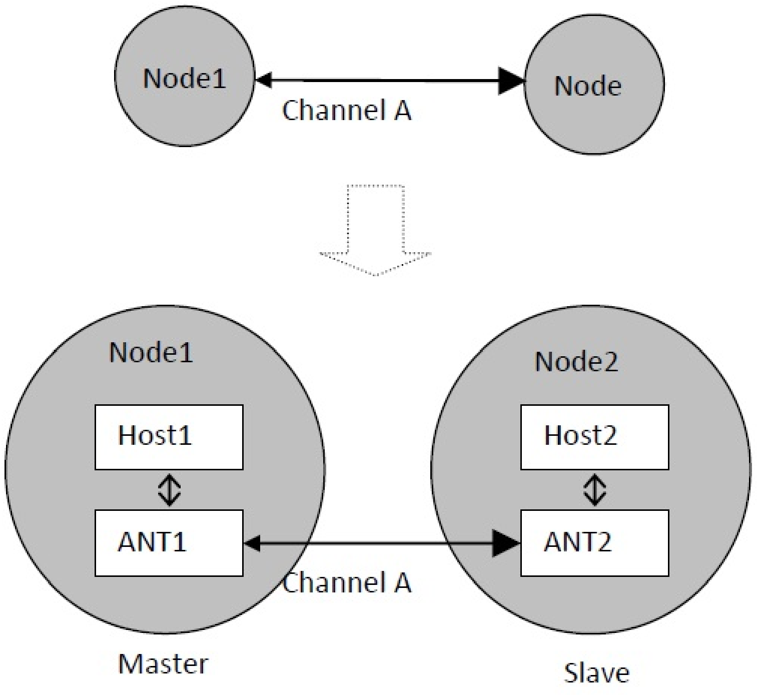


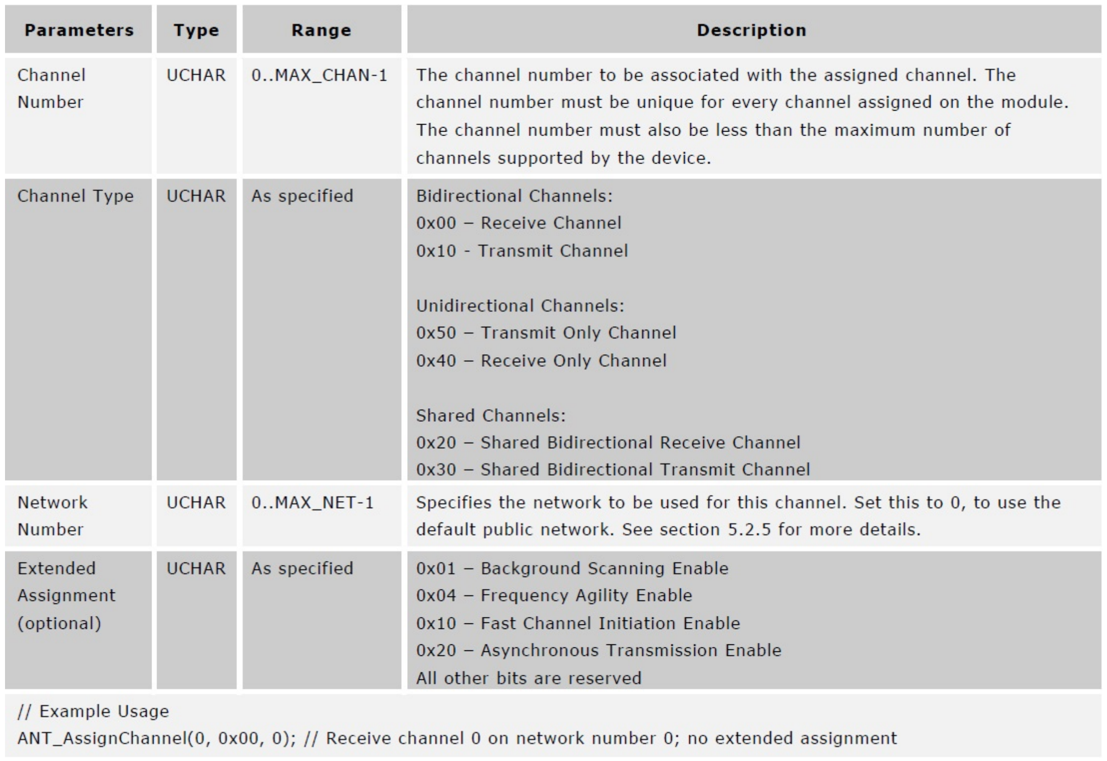
Figure 2.5: Host MCUs communicate via the node ANT engines (Source: [10])

reduces development time and concentrates application development on the host MCU. The developer uses the host MCU to establish and maintain application communications to other nodes via the ANT protocol engine as seen in Figure 2.5. This is done through a bidirectional, serial message protocol between the host MCU and the ANT protocol engine. Through this ANT-host serial protocol, the application configures ANT channels, opens channels, pairs devices, and transfers data. This is all done through a set of simple commands an example of which can be seen in Figure 2.6 [10].

Figure 2.7 shows the basic format of a serial message between the host MCU and ANT engine while Table 2.1 outlines each block of the message structure with a description [10].

Table 2.1: Descriptions of ANT serial message structure. (Source: [10])

Byte	**Name**	**Length**	**Description**
0	SYNC	1 Byte	Fixed value of 0xA4 or 0xA5
1	MSG LENGTH	1 Byte	Number of data bytes in message
2	MSG ID	1 Byte	Data Type Identifier (1-255 valid)
3..N+2	MSG CONTENT	N Bytes	Message content
N+3	CHECKSUM	1 Byte	XOR of all previous bytes including SYNC



Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel number to be associated with the assigned channel. The channel number must be unique for every channel assigned on the module. The channel number must also be less than the maximum number of channels supported by the device.
Channel Type	UCHAR	As specified	Bidirectional Channels: 0x00 – Receive Channel 0x10 - Transmit Channel Unidirectional Channels: 0x50 – Transmit Only Channel 0x40 – Receive Only Channel Shared Channels: 0x20 – Shared Bidirectional Receive Channel 0x30 – Shared Bidirectional Transmit Channel
Network Number	UCHAR	0..MAX_NET-1	Specifies the network to be used for this channel. Set this to 0, to use the default public network. See section 5.2.5 for more details.
Extended Assignment (optional)	UCHAR	As specified	0x01 – Background Scanning Enable 0x04 – Frequency Agility Enable 0x10 – Fast Channel Initiation Enable 0x20 – Asynchronous Transmission Enable All other bits are reserved

```
// Example Usage
ANT_AssignChannel(0, 0x00, 0); // Receive channel 0 on network number 0; no extended assignment
```

Figure 2.6: Example of the host-ANT serial protocol commands (Source: [10])

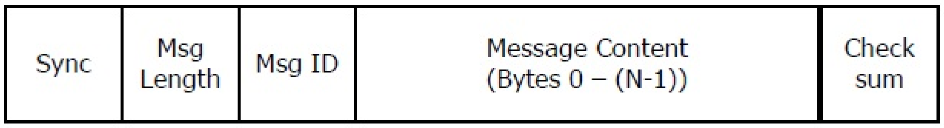
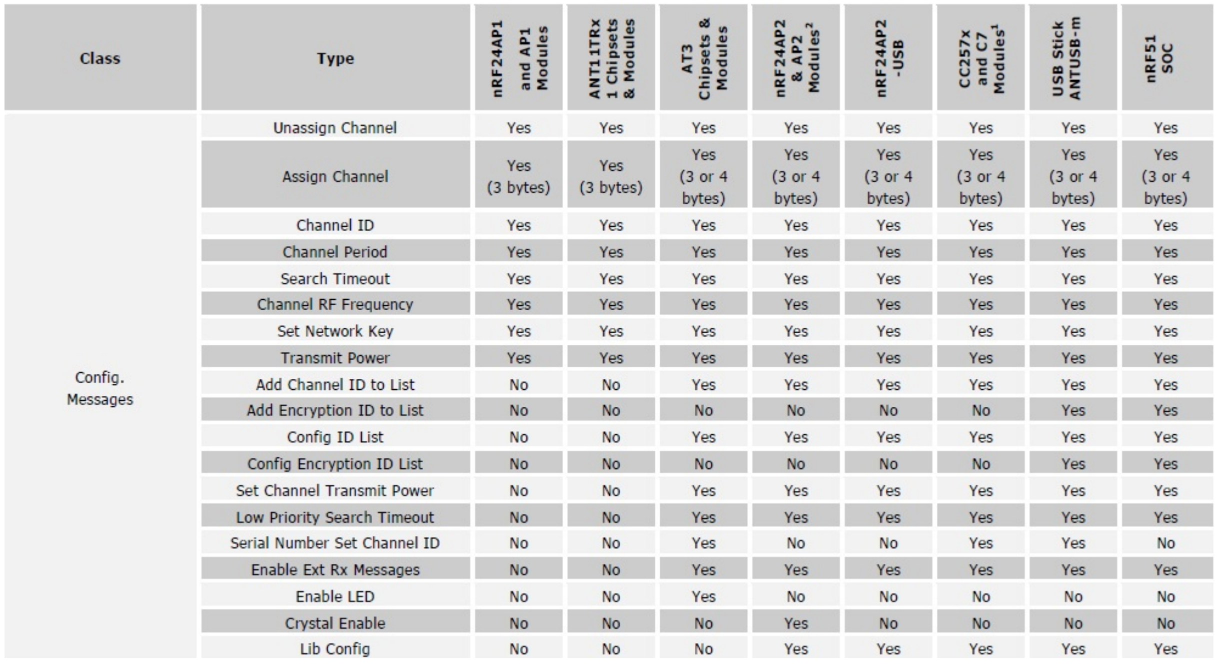


Figure 2.7: ANT message structure. (Source: [10])

Interestingly, not all manufacturer's chips have the full suite of the ANT-host serial protocol messages. Different ANT engines have different purposes and may be packaged with other modules so they may or may not need the full ANT-host serial protocol. Figure 2.8 shows a small sample list of host MCU and ANT engine configuration class messages and which messages the different manufacturer's chips support.

2.2.3 Channels

ANT node to node communication is channel based, that is, each ANT node connects to other ANT nodes through dedicated channels. The simplest channel connects a single master node and a single slave node. A master node acts a the primary transmitter and a slave node acts as the primary receiver. In some cases, such as a hub, a node acts as both a slave and a master. In those cases, however, the hub is a master or a slave for a particular channel and it is supporting



Class	Type	nRF24AP1 and AP1 Modules	ANT11TRx 1 Chipsets & Modules	AT3 Chipsets & Modules	nRF24AP2 & AP2 Modules^2	nRF24AP2 -USB	CC257x and C7 Modules^1	USB Stick ANTUSB-m	nRF51 SOC
Config. Messages	Unassign Channel	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Assign Channel	Yes (3 bytes)	Yes (3 bytes)	Yes (3 or 4 bytes)	Yes (3 or 4 bytes)	Yes (3 or 4 bytes)	Yes (3 or 4 bytes)	Yes (3 or 4 bytes)	Yes (3 or 4 bytes)
	Channel ID	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Channel Period	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Search Timeout	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Channel RF Frequency	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Set Network Key	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Transmit Power	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Add Channel ID to List	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Add Encryption ID to List	No	No	No	No	No	No	Yes	Yes
	Config ID List	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Config Encryption ID List	No	No	No	No	No	No	Yes	Yes
	Set Channel Transmit Power	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Low Priority Search Timeout	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Serial Number Set Channel ID	No	No	Yes	No	No	Yes	Yes	No
	Enable Ext Rx Messages	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Enable LED	No	No	Yes	No	No	No	No	No
	Crystal Enable	No	No	No	Yes	No	No	No	No
	Lib Config	No	No	No	Yes	Yes	Yes	Yes	Yes

Figure 2.8: Host MCU to ANT engine communication commands and manufacturer's support (Source: [10])

multiple channels [10]. Specifying whether a node is a slave or a master is part of the channel configuration.

There are three channels modes used for communication; independent, shared, and scan. The independent channel is most basic and has one master and one slave. The shared channel is used for one master to send data to multiple slaves, either individually or all at once. The scan channel is used by one slave node to receive data from multiple master nodes [23]. Figure 2.12 depicts three independent channels. Each channel, A, B, and C, has only one master and one slave. Although Hub 1 is a node in all three channels, each channel is considered independent.

For a shared channel, there is a one or two byte shared channel address field. This field is controlled by the host application and, if used, displaces one or two bytes of the eight payload bytes. The master will transmit data to a number of slaves but the ANT engine will only send up data to the slave application if the shared channel address matches the slave node's shared channel address, or if the slave node's channel address is set to a wildcard value. The one byte channel address field allows 255 slave devices to share one channel while the two byte address allows for 65,025 slaves [10].

The channel scan mode allows a slave node to receive from multiple master nodes. This is accomplished by leaving the slave node radio in continuous scanning mode. Because the radio is always occupied, the node cannot have any other channels open [10].

Most ANT communication implementations use channels that are synchronous, independent, and bidirectional; however, the way in which the ANT nodes communicate depends on how each node configures the channel. At its most basic, a channel is a radio frequency and a time slot. An ANT channel, however, is defined by five parameters; network number, channel type, channel identification (ID), Radio Frequency (RF) frequency, and channel period [10]. Each of these parameters will be discussed in turn.

The network number is a number that refers to the network on an ANT device. Each ANT network requires an ANT network key and, depending on the ANT chip used, anywhere from one to eight network keys can be assigned. For the Computer Controlled Eddy Current Brake Bicycle Trainer SoC, only three network numbers could be used for all eight channels. The network number is an eight bit field with the network number 0 being assigned to the default public network key [10].

Channel type is indicated by an 8-bit field set to an acceptable value between the range of 0 to 255. Table 2.2 lists several common channel types [10]. In order for two nodes to communicate on the same channel, they need to have the same channel type, but one must be a master and one must be a slave. For example, a node can set a channel type to a bidirectional master channel. This means that it will only connect with a node that has a channel type set to a bidirectional slave channel because only a slave and a master can communicate.

The bidirectional channel, of course, means that data flow both ways, but the primary direction of the data flow is determined by the node. For example, the slave node operates mostly as a receiver while the master node operates mostly as a transmitter. When the channel is shared, it means that a single, central ANT node must receive data from many other nodes. In this case, the transmitting nodes share a single channel to the central node [10].

The next part of the channel configuration is the channel ID. The channel ID is a 4-bit value that contains three fields; transmission type, device type, and the device number. The transmission type is a number which represents a device's transmission characteristics such as independent or shared channel [28]. This number is either pre-defined in managed networks or determined by the device's manufacturer. The device type is a number that represents the type of master device, for example, a heart rate monitor or bicycle power meter, while the

Table 2.2: Acceptable ANT channel type examples (Source: [10])

8 bit Value	Channel Description
0x00	Bidirectional Slave Channel
0x10	Bidirectional Master Channel
0x20	Shared Bidirectional Slave Channel
0x30	Slave Receive Only Channel (diagnostic)
0x40	Master Transmit Only Channel (legacy)

device number is a number that is specific to a particular device. By specifying the individual device number, ANT wireless PANs can be set to prevent picking up devices in an overlapping wireless PANs, such as two bicycle riders next to each other. Of course, for private networks, the channel ID can be user defined. Only nodes with matching channel IDs or channel IDs set to zero, which represents a wildcard value, can communicate with each other. In the situation where the channel ID is set to a wildcard, the salve will find the first master that has a matching network key and frequency [10].

In addition to specifying the channel type and ID, the channel configuration also includes specifying RF frequency. There are 125 RF operating frequencies supported and the one selected is determined by an 8-bit field. In this field, the acceptable values range from 0 to 124. The value is just the offset in 1 megahertz (MHz) increments from 2400 MHz with a maximum frequency of 2524 MHz. The value for this field is given by Equation 2.1. The default RF operating frequency is 2466 MHz. Many channels can exist on one frequency because ANT uses TDMA [10].

$$RF\_Frequency\_val = \frac{Desired\_RF\_Frequency(MHz) - 2400MHz}{1MHz} \tag{2.1}$$

The Channel Period (Tch) is part of the channel configuration and can be set using the channel period value. The channel period value is a 16-bit field parameter that is used to set the message rate. The developer can set the desired message rate and then use Equation 2.2 to determine the application parameter, channel period value. The default message rate is 4 hertz (Hz) [10].

$$Channel\_Period\_val = \frac{32768}{Tch(Hz)} \tag{2.2}$$

In addition to these main channel parameters just discussed, there is also the option to set the master and slave transmitter power. The default value for the transmitter power is set to a value 3 which is equal to 0 dBm. The process of establishing a channel between a master node and a slave node is summarized by Figure 2.11. In this figure, the channel parameters with solid lines have no default values. These values must be set by the application. The parameters with the dashed lines can use the default value if none is specified in the application [10].

To begin the node connection process, a master node opens a search window in order to check that its transmission is not going to interfere with other device transmissions. From this search, a designated Tch is found. This Tch is the message rate of data packets sent by the master. The master node will then always transmit a message on that specific Tch once the channel is opened. In the bidirectional channel case, the master node will leave the radio receiver on for a short time after each transmitted message finishes. If needed, this allows the slave node to send back data immediately after receiving a message. Figure 2.9 shows the channel communication period, Tch, between two nodes [10].

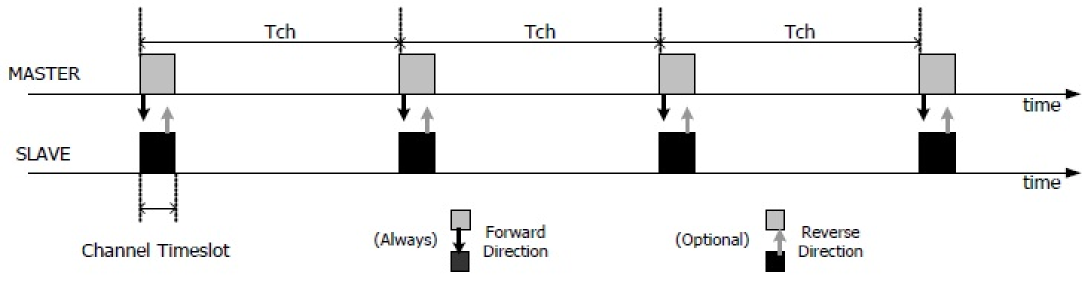


Figure 2.9: ANT channel communications (Source: [10])

When a channel is already open and a slave wishes to connect, it will immediately enter search mode when its channel opens. Even in search mode the radio is only intermittently active rather than constantly on, which helps to conserve power. Because of this, master transmission detection may not be made on the first master transmission after the slave channel is opened [7]. Figure 2.10 shows a slave node channel searching for a master node channel transmission. The circle denotes when the master transmission is detected. Once the slave node detects the master node transmission, it exits search mode and enters tracking mode.

The slave Tch should be set to either the master's Tch or a multiple of it. If it is set to a multiple, then the slave will miss master data transmissions. If the slave's Tch is set to a factor

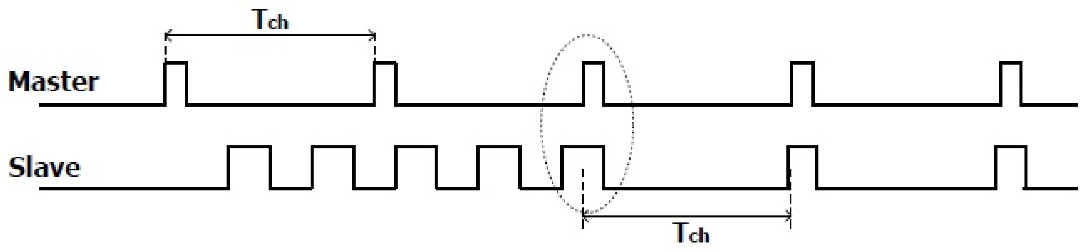


Figure 2.10: ANT slave node channel searches for a master node channel transmission (Source: [7])

of the master's Tch then the slave is wasting power by turning on the radio when there is no transmission [7].

There are four ANT data types; broadcast data, acknowledged data, burst data and advanced burst data. Broadcast data is the default data type and is the most basic data type. It is sent from the master to slave on every Tch. If there is no new broadcast data to send, then the previous data packet will be resent. In addition, broadcast data is never acknowledged by the slaves so the master will not know if data packets are lost. Broadcast data uses the least amount of power and bandwidth [10].

A node host application may request that data packets be acknowledged. When this happens, the receiving node will send an acknowledgment back to the transmitting node on the next Tch slot. If the transmission is not acknowledged, there is no automatic re-transmission. The master host application can mix and match broadcast data with acknowledged data, that is, the master can send a data packet not requiring acknowledgment and, in the next Tch time slot, it can send a data packet that requires acknowledgment. The acknowledged data transmission will use more power and bandwidth than broadcast data [10].

Burst data messages are used when large amounts of data need to be sent. Burst data are a series of continuous messages that are acknowledged. The rate of these data messages does not correspond to the Tch and, in fact, is significantly faster. Burst data has a maximum data throughput of 20 kilobits per second (kbps). Prolonged burst data transmissions could cause slave nodes to lose synchronization. If this happens, the slave node will automatically drop into search mode to reestablish synchronization [10].

Finally, there is advanced burst data which allows some ANT devices to increase the maximum data throughput to 60 kbps. This uses the most power and is not recommended unless the host application specifically needs it [10].

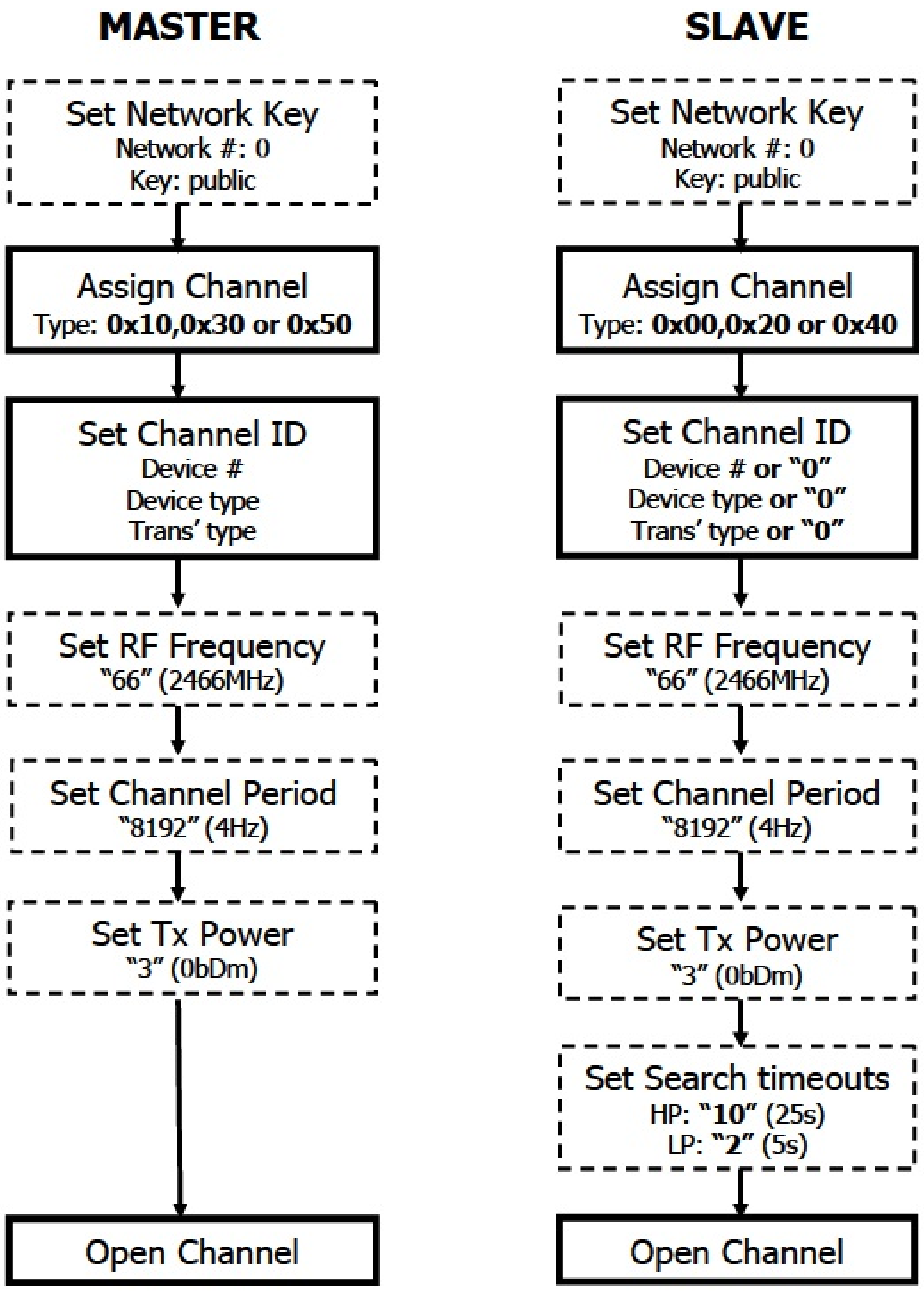


Figure 2.11: Master and slave nodes channel established. Dotted lines represent optional set up, i.e. default values are specified. (Source: [10])

2.2.4 Networks

A simple ANT network is depicted in Figure 2.12. Each channel depicted in Figure 2.12 has a master node and a slave node at minimum. In addition, the arrows reflect the direction of data exchange with the large arrows depicting the master node or primary transmitter. Table 2.3 shows the status of each node as a master, Transmitter (TX), or slave, Receiver (RX), for Figure 2.12 [10].

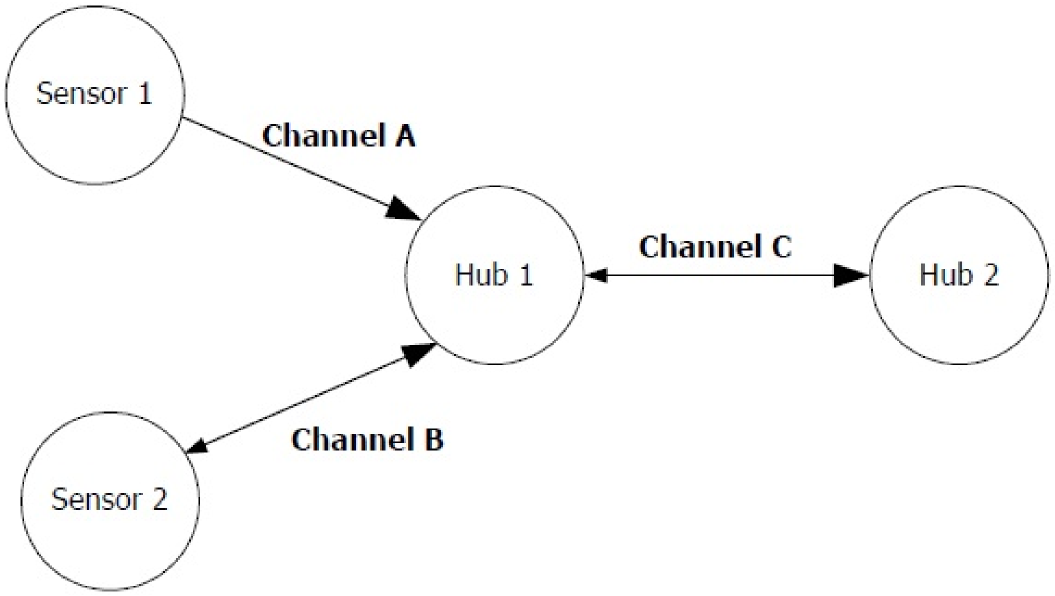


Figure 2.12: ANT simple network example (Source: [10])

Table 2.3: Master and Slave Nodes depicted in Figure 2.12 (Source: [10])

Channel	**Master Node**	**Slave Node**
Channel A	Sensor 1 (TX-Only)	Hub 1 (RX)
Channel B	Sensor 2 (TX)	Hub 1 (RX)
Channel C	Hub 1 (TX)	Hub 2 (RX)

ANT supports many network topologies from a simple 2-node unidirectional communication to a complex multi-transceiver system with point-to-multipoint communications. Some examples of the topology types supported are shown in Figure 2.13 [10].

Each network has a network key that is either public, managed, or private. For development, a public network key is provided by Dynastream Innovations. This public network is considered

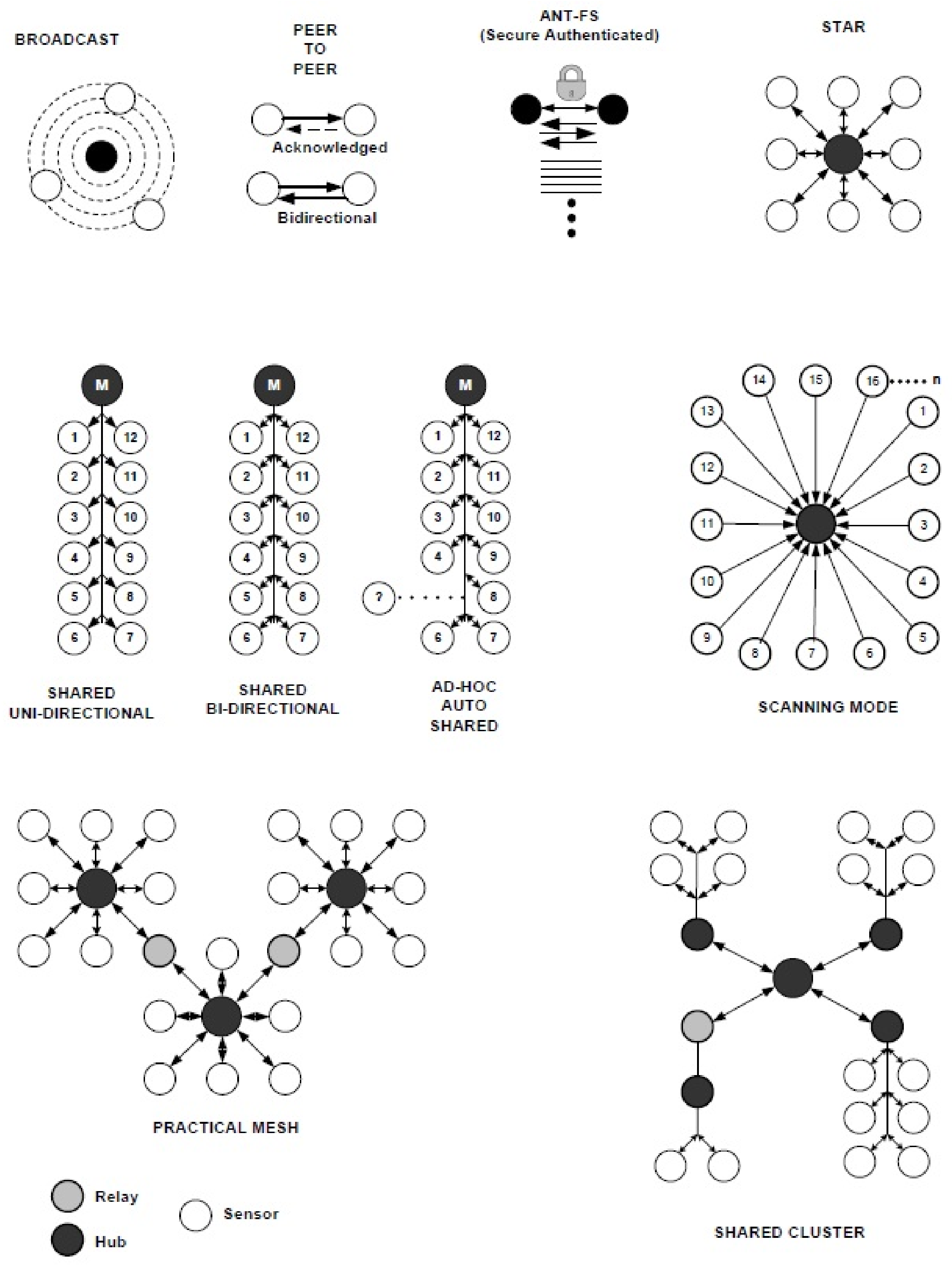


Figure 2.13: Examples of ANT network typologies (Source: [10])

open, that is, it will accept any node wishing to connect and let anyone listen to the data exchange [6].

A managed network key is used for interoperability across an industry sector, for example, fitness. When these keys are used, devices must adhere to specific specifications in the channel configurations as well as data formatting in order to facilitate interoperability across different companies and devices. Dynastream provides two managed ANT network keys.

1. The ANT+ network key is specifically used for devices communicating via the ANT protocol and, in addition, implements a higher level control of exercise data. It must be used only at the specific frequency of 2457 MHz.

2. The Advanced and Adaptive Network Techology File Share (ANT-FS) network key uses the ANT stack and implements higher level file sharing protocol as well.

Finally, private networks allow the greatest control and security. The private network allows the developer to set all channel configurations and data formatting while allowing full control over who can join the network. The private network keys are assigned Dynastream Innovations. If the key is not defined by Dynastream, the invalid key will be replaced with the public key when the device channel is configured. This will allow devices to communication although the communications will not be private [6].

2.2.5 Messages and ANT+ Profiles

The node to node ANT message contains only eight bytes of application data. The standard message format and the optional extended message format are shown in Figure 2.14. Recall that, if specified, up to two of the payload bytes can be used to specify a shared address for the shared channel usage. This is shown in Figure 2.15.

The extended data message is the same as the standard data message up to and including the payload; however, after the payload there is a flag byte that signals the ANT engine that more information fields follow. This is called the flagged extended data message format and it is important because the ANT+ profiles use this format [28].

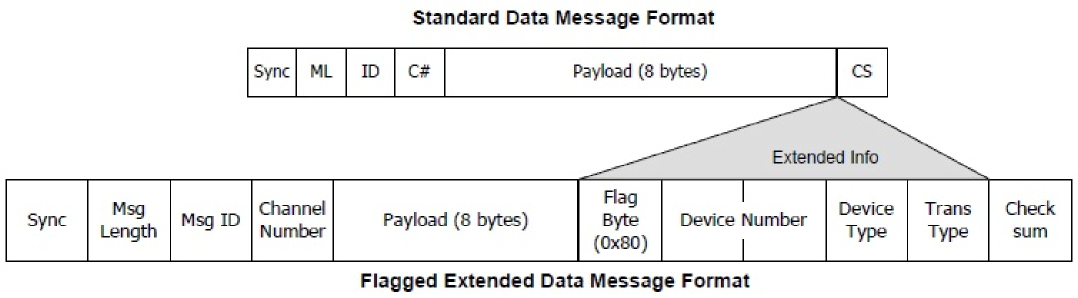
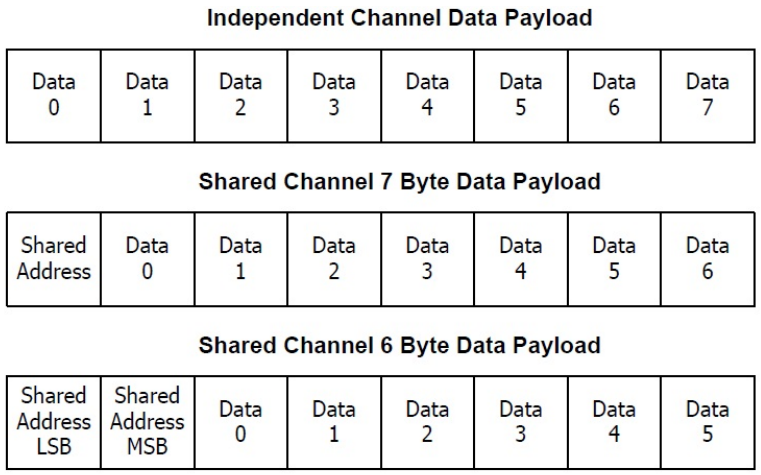


Figure 2.14: ANT flagged extended data message format (Source: [10])


Independent Channel Data Payload

Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7

Shared Channel 7 Byte Data Payload

Shared Address	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6

Shared Channel 6 Byte Data Payload

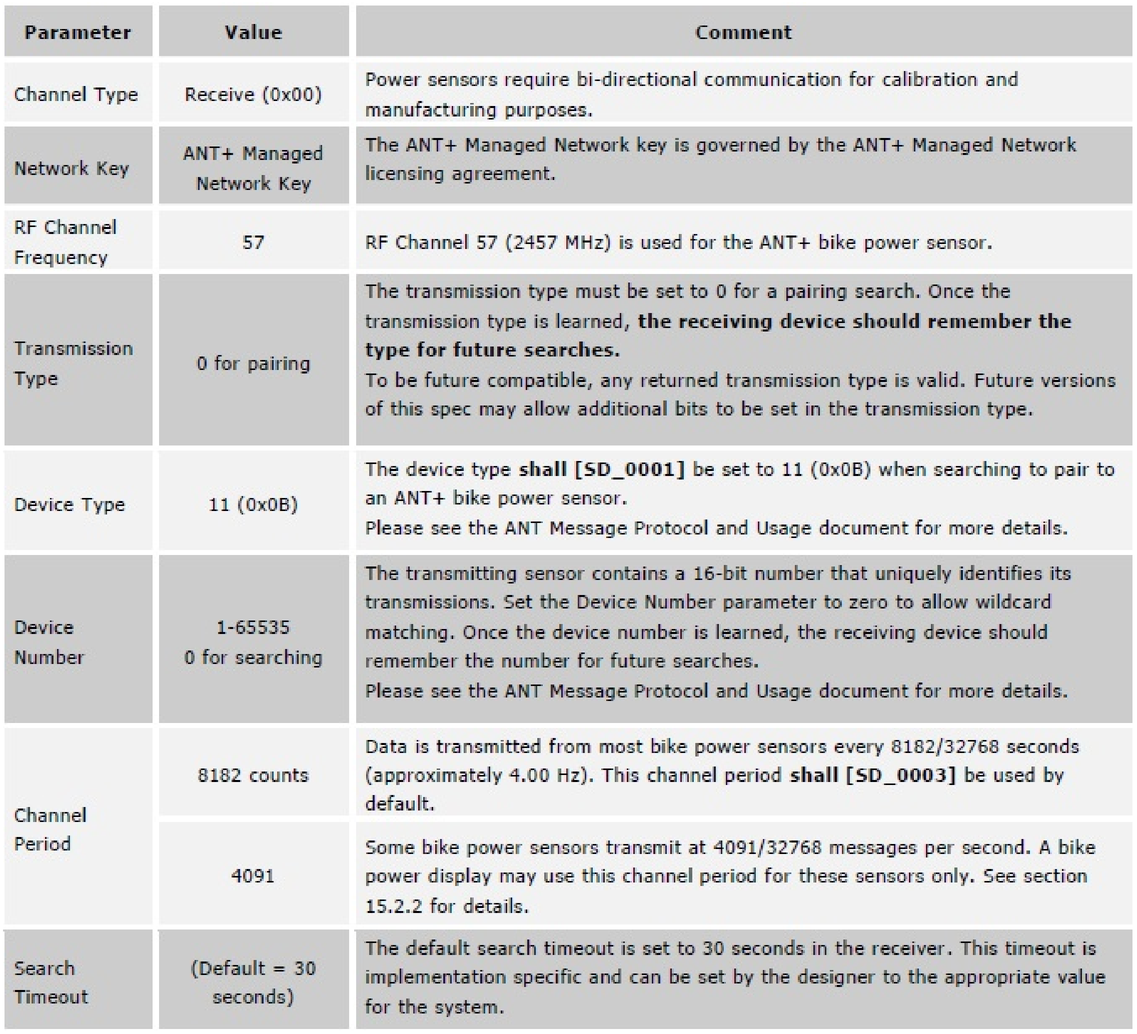
Shared Address LSB	Shared Address MSB	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5

Figure 2.15: Shared channel addressing using one or two payload bytes (Source: [10])

ANT+ is a managed ANT network and an ANT+ profile is a standardization of how particular devices send data over the ANT+ managed network. For example, all heart rate monitors conform to one profile standard while all bicycle power meters conform to another profile standard. These standards are implemented in the application or host MCU on OSI layers five, six and seven, as seen in Figure 2.2 [8].

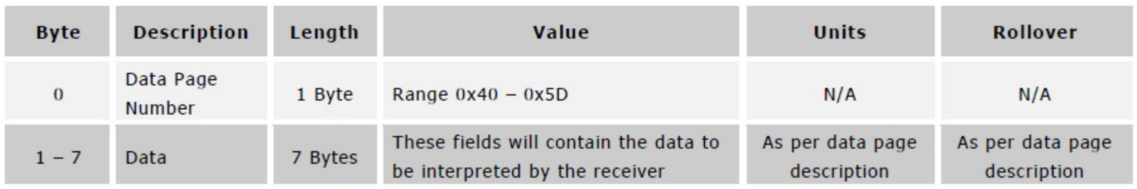
Currently, the ANT+ managed network classifications include sports, wellness, and home health. All ANT+ devices use the ANT+ network key and they all must implement an ANT+ profile. Besides message payload formats the ANT+ profiles also include specific slave and master channel configurations. Figure 2.16 shows the specific ANT+ channel configuration for a receiver of bicycle power sensor data [8].

Next, there are several general and sensor specific message payload formats known as pages. In general, whether the page is generic or sensor specific, the payload format is shown in Figure 2.17 [8] and an example of the raw received bicycle power data is shown in Listing 2.1. In Listing 2.1, 4e is the data type, broadcast. The next byte, 00 is the channel number followed by the eight byte payload. Listing 2.2 shows the decoded bicycle power sensor data from the message received on line 8 from Listing 2.1.



Parameter	Value	Comment
Channel Type	Receive (0x00)	Power sensors require bi-directional communication for calibration and manufacturing purposes.
Network Key	ANT+ Managed Network Key	The ANT+ Managed Network key is governed by the ANT+ Managed Network licensing agreement.
RF Channel Frequency	57	RF Channel 57 (2457 MHz) is used for the ANT+ bike power sensor.
Transmission Type	0 for pairing	The transmission type must be set to 0 for a pairing search. Once the transmission type is learned, **the receiving device should remember the type for future searches.** To be future compatible, any returned transmission type is valid. Future versions of this spec may allow additional bits to be set in the transmission type.
Device Type	11 (0x0B)	The device type **shall [SD_0001]** be set to 11 (0x0B) when searching to pair to an ANT+ bike power sensor. Please see the ANT Message Protocol and Usage document for more details.
Device Number	1-65535 0 for searching	The transmitting sensor contains a 16-bit number that uniquely identifies its transmissions. Set the Device Number parameter to zero to allow wildcard matching. Once the device number is learned, the receiving device should remember the number for future searches. Please see the ANT Message Protocol and Usage document for more details.
Channel Period	8182 counts	Data is transmitted from most bike power sensors every 8182/32768 seconds (approximately 4.00 Hz). This channel period **shall [SD_0003]** be used by default.
	4091	Some bike power sensors transmit at 4091/32768 messages per second. A bike power display may use this channel period for these sensors only. See section 15.2.2 for details.
Search Timeout	(Default = 30 seconds)	The default search timeout is set to 30 seconds in the receiver. This timeout is implementation specific and can be set by the designer to the appropriate value for the system.

Figure 2.16: Bicycle power sensor ANT+ channel configuration (Source: [8])



Byte	Description	Length	Value	Units	Rollover
0	Data Page Number	1 Byte	Range 0x40 – 0x5D	N/A	N/A
1 – 7	Data	7 Bytes	These fields will contain the data to be interpreted by the receiver	As per data page description	As per data page description

Figure 2.17: General ANT+ message payload format (Source: [8])

Listing 2.1: ANT+ raw data

```
Auto-Open Initiated...
> Setting Channel ID...
Channel Id Set: 0,11,0
> Opening Channel...
Received BROADCAST_DATA_0x4E
  :: 4e, 00-10-65-FF-3A-73-13-32-00
Received BROADCAST_DATA_0x4E
  :: 4e, 00-12-65-65-3A-71-7E-94-6E
```

Listing 2.2: ANT+ decoded data

```
4e  Broadcast Data
00  Channel Number 0
12  Data Page Number 0x12 or decimal 18
65  Event Count - increments with each information update.
65  Crank Ticks - increments with each crank revolution.
71  Period LSB - Accumulated crank period least significant byte
7E  Period MSB - Accumulated crank period most significant byte
94  Accumulated Torque LSB - Accumulated torque least
   significant byte
6E  Accumulated Torque MSB - Accumulated torque most significant
    byte
```

2.3 Eddy Current

Eddy currents are loops of electrical current that are induced within a conductor. The currents are induced by changing the magnet field within the conductor. This can be done by moving either a conductor through a magnetic field or by moving a magnet by a stationary conductor. This phenomenon was discovered by Michael Faraday in the early 1830s. He saw that any change in a magnetic field of a wire coil caused a voltage to be induced in the same coil. The demonstration that a spark will jump the gap across a switch when that switch breaks the circuit to a large electromagnet made of many turns of wire around an iron core is an example of this. Faraday expressed this as a single equation known as Faraday's Law [21].

$$\varepsilon = -\frac{d\Phi_B}{dt} \tag{2.3}$$

Equation 2.3 is Faraday's Law and it states that the Electromotive Force (EMF), ε, is equal to the negative of the rate of change of the magnetic flux, Φ_B. In addition to moving a magnet relative to a conductor, there are other ways to change the magnetic field, for instance, changing the current to an electromagnet will change the magnetic flux thereby inducing an EMF.

The other laws that apply to eddy currents are Lenz's law which states: *The direction of an induced current is such as to oppose the cause producing it* [29] and Ampere's law which is shown in Equation 2.4 [21].

$$\oint B \cdot d\ell = \mu_0 I \tag{2.4}$$

Lenz's law can be seen as a negative sign in Equation 2.3. Simply stated, Ampere's law says that the integral of the magnetic field, B, around a closed loop is equal to the static electric current, I, which passes through the loop times a magnetic constant, μ_0.

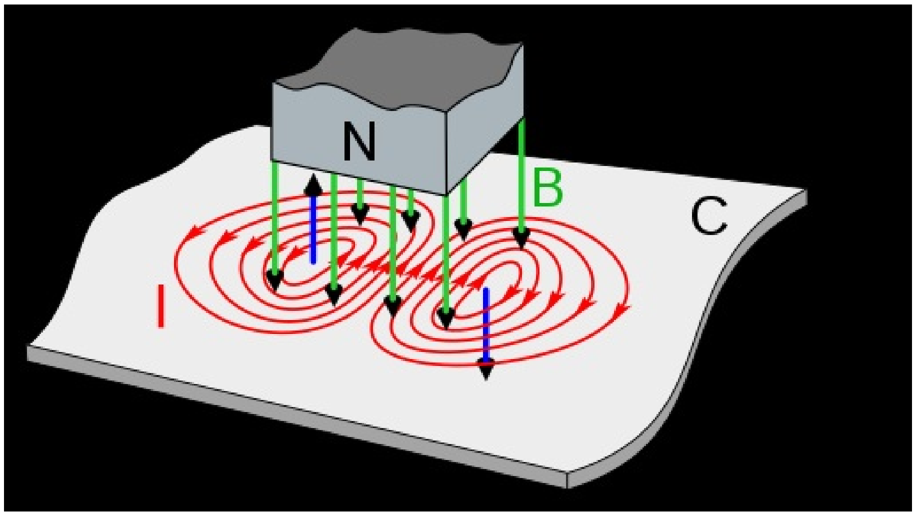


Figure 2.18: Eddy currents on the leading and trailing edges of a moving magnet field[1]

Figure 2.18 shows the induced eddy currents in the conductor, C, with red lines. The currents are induced in opposite directions on each edge of the magnet. On the leading edge of the magnet, the magnetic flux is building so the induced eddy current will try to counteract that be creating its own flux. On the trailing edge of the magnet, the magnetic flux is waning so the conductor tries and keep the magnet flux from changing so it creates an eddy current in the opposite direction [21].

2.4 Eddy Current Brake

The eddy current brake is a device used to slow or stop a moving object. Instead of using frictional force to slow the moving object, the eddy current brake uses induced currents, eddy currents, on a conductor that is moving relative to a magnetic field. This is, of course, an application of Lenz's law [29]. This concept can be seen on a spinning disk, D, in Figure 2.19. In the example, the green arrows are the direction of the magnetic flux caused by the magnets, the red circles are the induced eddy currents in the conductor and the blue arrows show the induced magnetic field from the eddy currents.

The benefit of this type of brake is that there is no contact. In other words, there is no mechanical wear. The downside is that the eddy current brake cannot provide a holding torque

[1]Picture is in the Public Domain and licensed via Wikimeda Commons https://creativecommons.org/licenses/by-sa/3.0, created by user Chetvorno and found at https://commons.wikimedia.org/wiki/File:Eddy_currents_due_to_magnet.svg

since the braking action is only in play when there is relative movement between the conductor and the magnet field. The braking action dissipates the kinetic energy as heat [29].

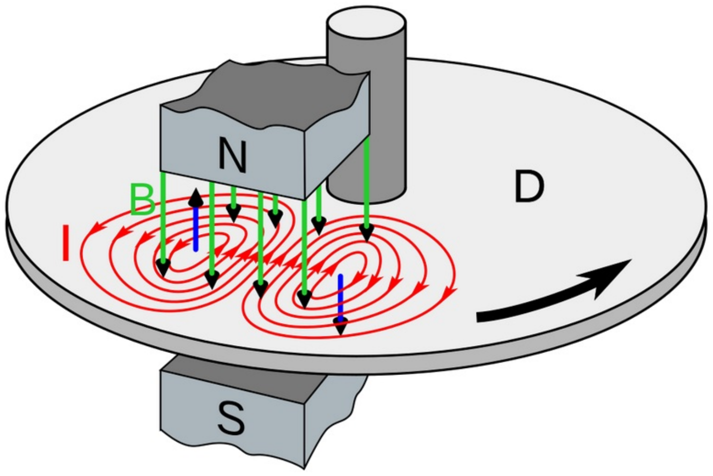


Figure 2.19: Eddy Current Brake[2]

A strong magnetic field will create a strong eddy current thus supplying the greatest braking action. The power of electromagnets are given in Ampere-turns, that is the amount of current flowing through a number of wire turns [27]. Another way to have the conductor feel the maximized force of the magnetic field is to place the magnets as close as possible to the conductor. The smaller the air gap, the more concentrated the magnetic flux, the more eddy current is developed [13].

One way to control the strength of the eddy current brake is by increasing or decreasing the current to the electromagnets. If the power supply to the eddy current brake is not controllable, then control can be achieved by using power relays to turn the current on and off to the electromagnets. Since the second control scenario is similar to the experiment that Faraday used to develop Equation 2.3, it is important to limit the induced current spike into power supply as the relays are opened. This is done by use of a flyback diode [11]. Figure 2.20 shows an inductor circuit similar to the one that is used to control the eddy current brake, with and without a flyback diode.

Exercise equipment such as bicycle trainers use eddy current brakes to provide resistance for workout purposes. In particular, the STAC Zero bicycle trainer uses 13 sets of rare earth

[2]Picture is in the Public Domain and licensed via Wikimeda Commons https://creativecommons.org/licenses/by-sa/3.0, created by user Chetvorno and found at https://en.wikipedia.org/wiki/Eddy_current_brake#/media/File:Eddy_current_brake_diagram.svg.

permanent magnets to affect eddy currents on an aluminum rim of a bicycle wheel. This set-up has practically no mechanical wear on the bicycle because it is statically held by the rear wheel skewer, and it is easy to set up as there is no need to remove the rear wheel. The resistance is changed by the rider pedaling faster. The faster the aluminum rim moves through the sets of magnets the more eddy current is developed. Although this system provides a good workout, it is not able to use a preprogrammed workout to change the resistance for the rider [4]. The STAC Zero bicycle trainer is the inspiration for this project.

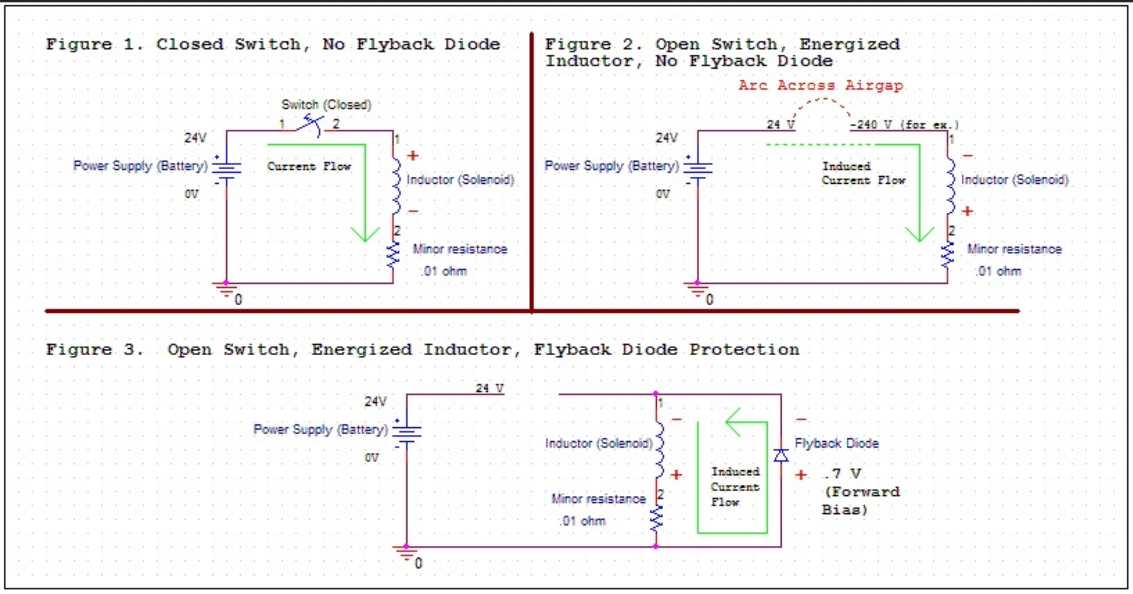


Figure 2.20: Inductor circuit with and without a flyback diode[3]

Figure 2.21 shows the STAC Zero trainer with no bicycle attached. Figure 2.22 shows the sets of rare earth magnets that induce eddy currents in the wheel's aluminum rim. Figure 2.23 shows a bicycle attached to the STAC Zero trainer.

2.5 Crank Power

The Computer Controlled Eddy Current Brake Bicycle Trainer depends on the feedback from a commercially available bicycle power meter. A strain gauge attached to a bicycle crank detects small flexing of the crank on each pedal stroke. This defection along a timeline gives two crucial

[3]Picture is in the Public Domain and licensed via Wikimeda Commons https://creativecommons.org/licenses/by-sa/3.0, created by user MrCrackers and found at https://upload.wikimedia.org/wikipedia/commons/7/76/FlybackExample.GIF

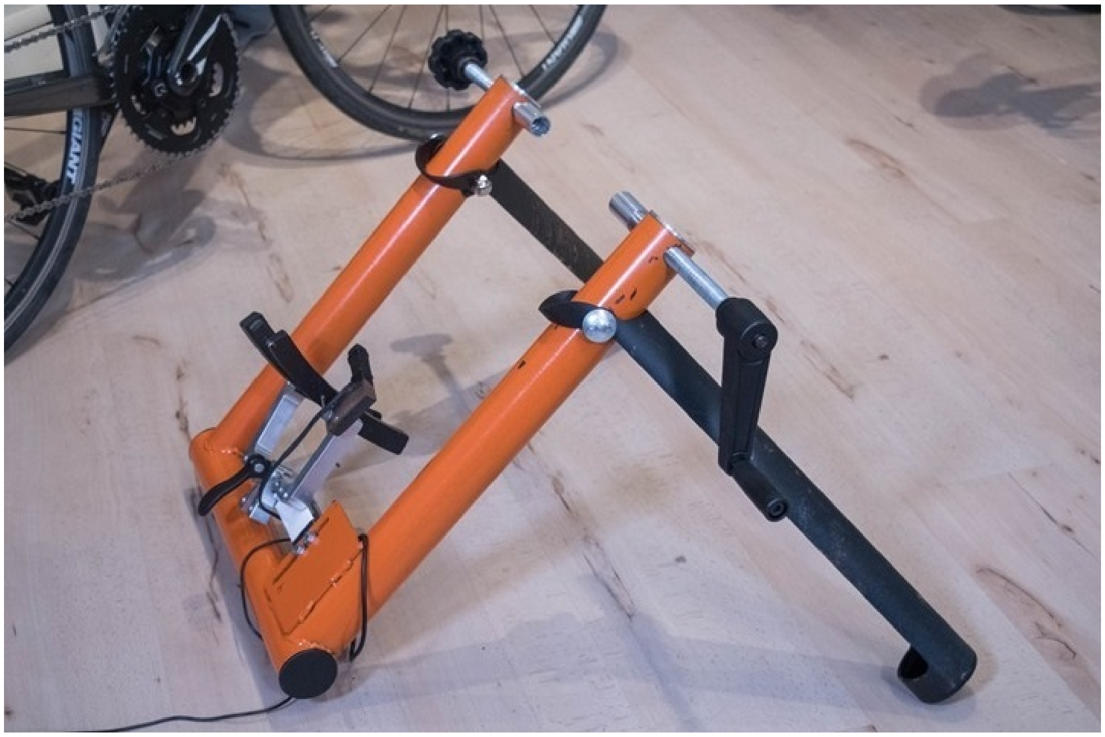


Figure 2.21: STAC Zero trainer (Source: [24]).

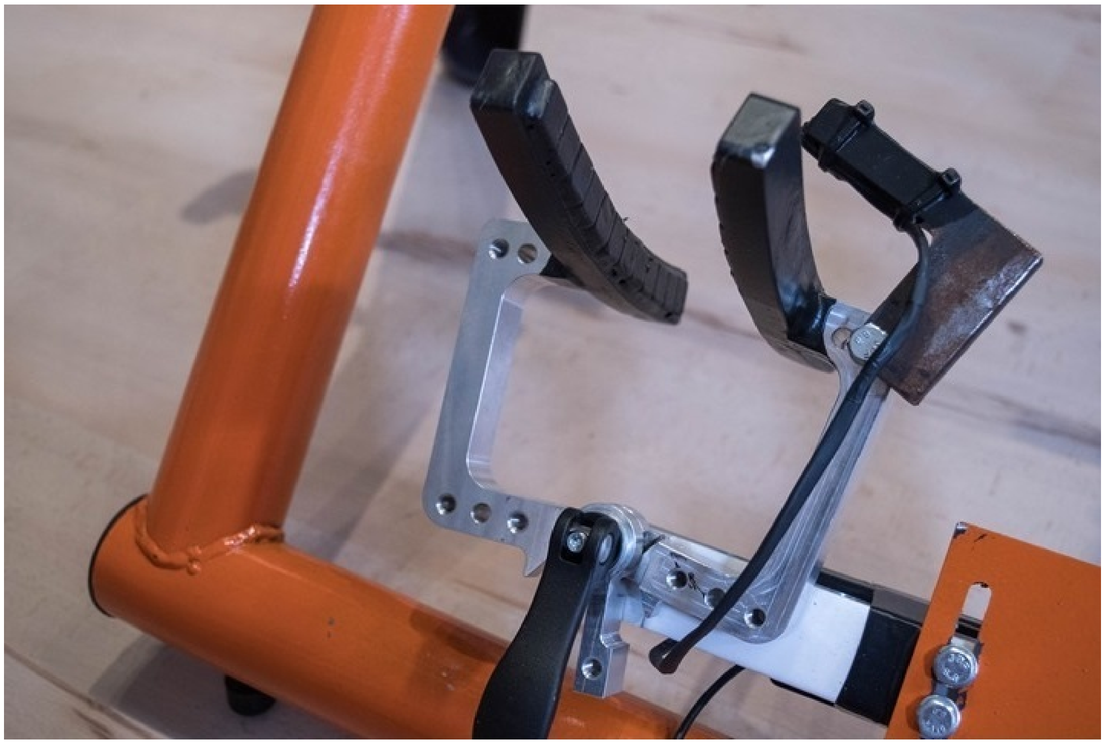


Figure 2.22: STAC Zero trainer rare earth magnet sets (Source: [24]).

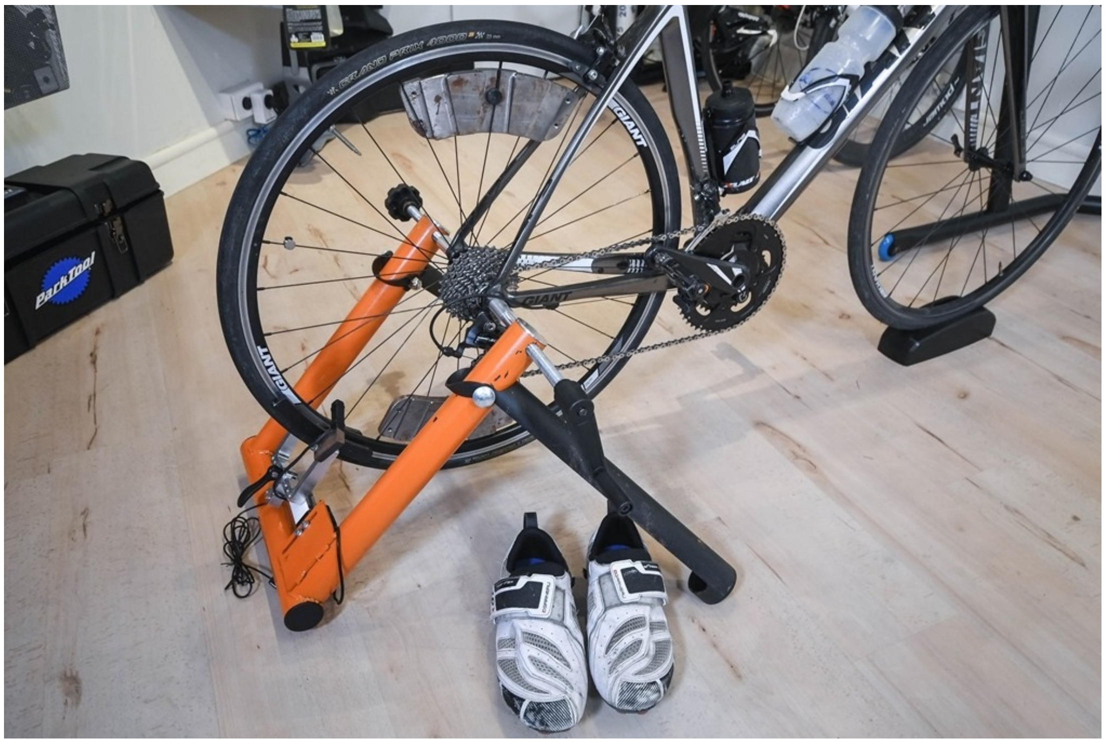


Figure 2.23: STAC Zero trainer with attached bicycle (Source: [24]).

pieces of information: cadence in Revolutions per Minute (RPM) and downward force. The power is then calculated using Equation 2.5.

$$Power = \tau * Cadence \tag{2.5}$$

In Equation 2.5, τ is torque. Torque is calculated using Equation 2.6.

$$\tau = F * 9.8 * L \tag{2.6}$$

In Equation 2.6, F is the average force per crank revolution, L is the length of the crank, and 9.8 is the gravitational constant.

Equation 2.7 show the final calculation of power for a bicycle crank meter.

$$Power = 2 * (F * 9.8 * L) * (R * 0.1047) \tag{2.7}$$

In this calculation, the torque is multiplied by 2 in order to take into account the crank force for each leg. R is the RPM and the constant 0.1047 is the conversion from RPM to radians per second [29] [41].

Chapter 3

System Architecture

3.1 System Hardware Configuration

A bicycle smart trainer uses several technologies that are pieced together for affecting resistance control. Some of the components are an eddy current brake, wireless protocol (to give feedback to a controlling MCU in order to determine current requirements to the electromagnets), a stand to hold the bicycle and brake. An overview of the Computer Controlled Eddy Current Brake Bicycle Trainer is shown in Figure 3.1 and Figure 3.2 shows a detail of the eddy current brake itself. Appendix A lists a bill of material for the Computer Controlled Eddy Current Brake Bicycle Trainer. This chapter will start with an overview of how to set up the physical system. This will include electrical schematics for the eddy current brake. The next section will cover the programming and wiring of the two controlling MCUs.

3.1.1 Bicycle Trainer and Bicycle

First, there is the actual trainer which holds the rider's bicycle. In this case, the trainer must be of the type that holds the bicycle by the rear wheel skewer. This is because the goal is minimal wear on the bicycle itself and instead of having the rear tire turn a resistance cylinder by friction, the trainer will have electromagnets mounted near the wheel's aluminum rim. The trainer itself has no moving parts. Its only purpose is to position the aluminum rim of the bicycle's rear wheel as close as possible to the electromagnets. It also stabilizes the bicycle so the rider is able to ride without the need to balance it upright. The trainer used for this project is shown in Figure 3.3 before modifications were done to it.

The next step for the trainer is to mount brackets to hold the electromagnets. For this, Unistrut Metal Framing System (Unistrut) [42] is used. There are angle brackets that are bolted to the Unistrut on which the electromagnets are mounted. Figure 3.4 shows the Unistrut bolted to the trainer. Figure 3.5 shows the angle brackets mounted to the Unistrut with electromagnets mounted on either side of the wheel rim. It is worth noting that in the final configuration wooden spacers were used between the angle bracket holding the magnets and the angle bracket connected to the Unistrut as seen in Figure 3.6. These spacers were used to allow multiple sets of electromagnets to be mounted.

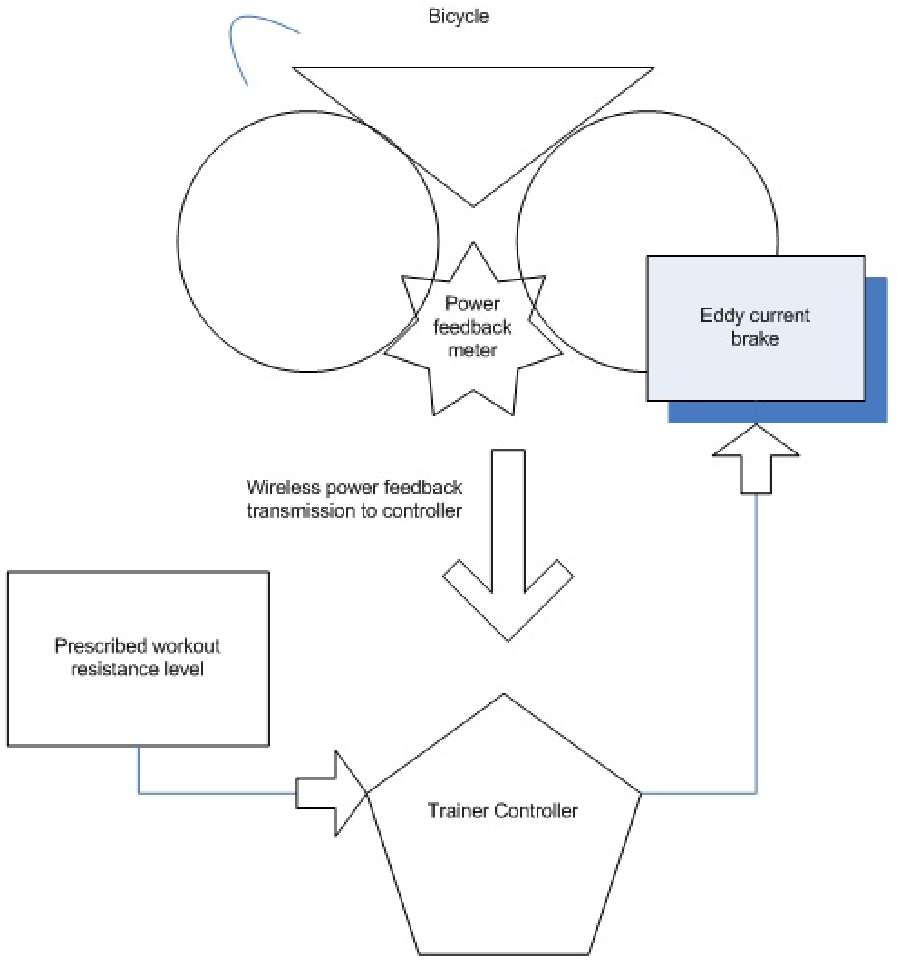

Bicycle
Power feedback meter
Eddy current brake
Wireless power feedback transmission to controller
Prescribed workout resistance level
Trainer Controller

Figure 3.1: Block diagram of Computer Controlled Eddy Current Brake Bicycle Trainer

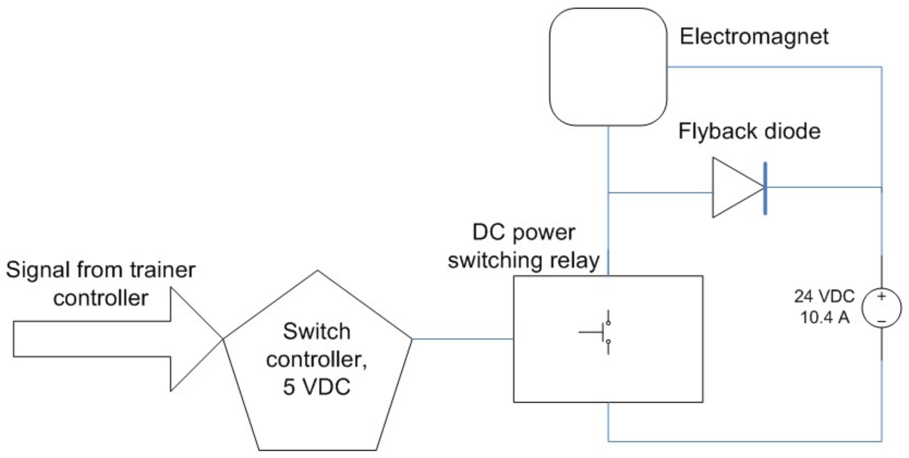


Figure 3.2: Design detail of eddy current brake

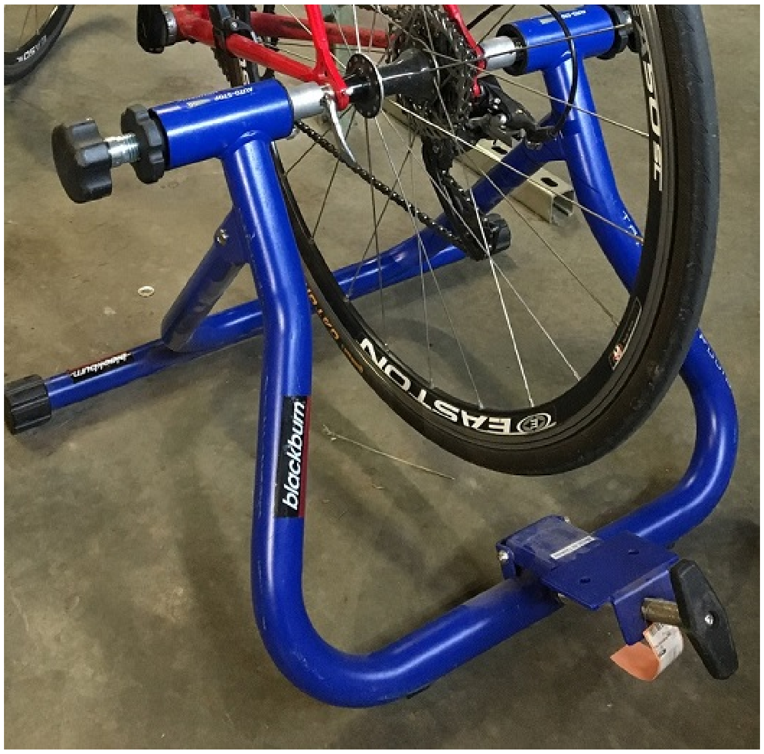


Figure 3.3: Trainer before electromagnets are mounted

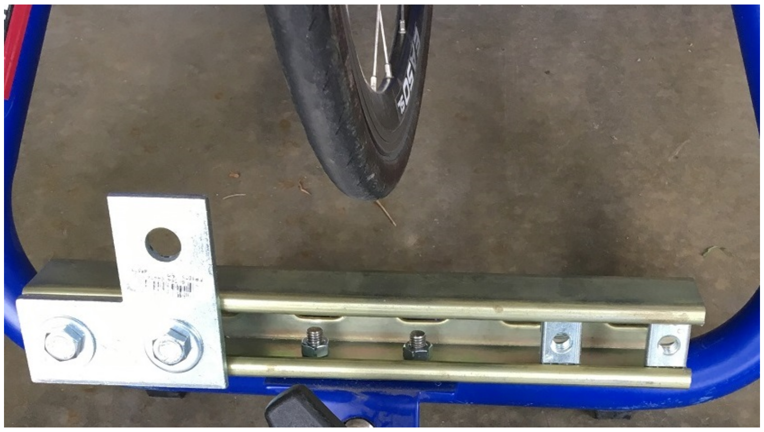


Figure 3.4: Unistrut mounted to the trainer

The next modification needed was to the bicycle rear wheel. From early testing, not enough eddy current could be generated in the wheel rim itself. This is because the electromagnets used were not strong enough to induce an eddy current in the rim itself. Also the angle of the rim made it difficult to get the electromagnets close to the spinning conductor material. The rim angled in like a V so the mounted magnets were close to the rim near their upper edge, but further away from the lower edge of the rim. There was little resistance generated with

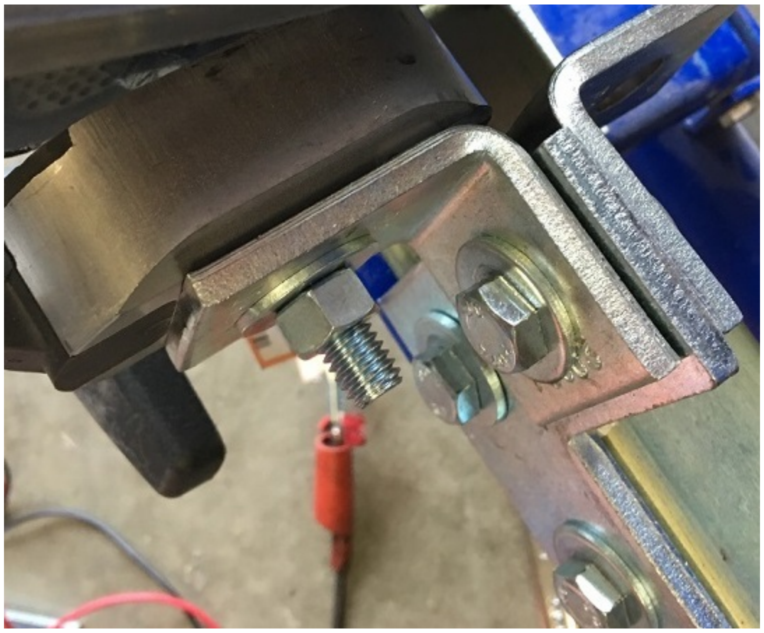


Figure 3.5: Brackets for electromagnet mounts

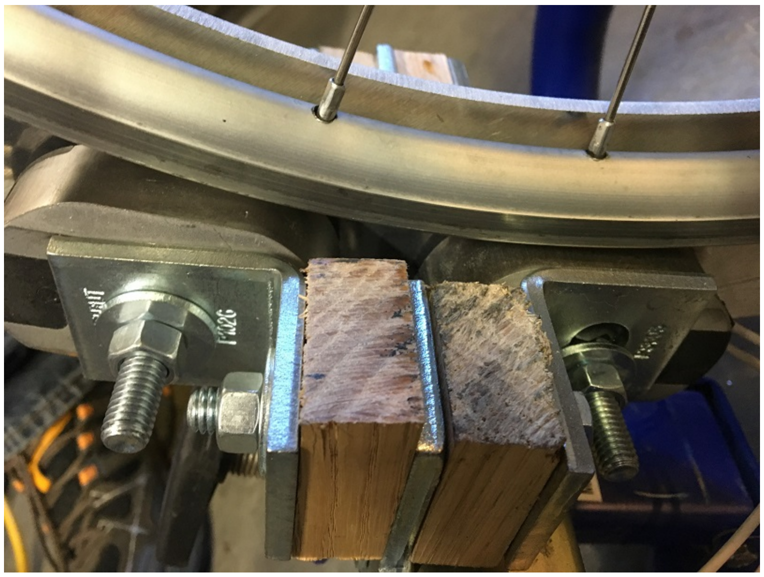


Figure 3.6: Spacers for multiple magnet mounting

this configuration and since the electromagnets selected could not be modified, a modification to the rear wheel itself was made.

A donut shape was water-jet cut from a sheet of aluminum that is 0.6 centimeters (cm) thick with an inner radius of 27 cm and an outer radius of 34 cm. It was then bolted to the rear bicycle wheel. The edge of the donut protruded far enough and was flat so the electromagnets could be mounted very close to it. This configuration created enough eddy current to use for a

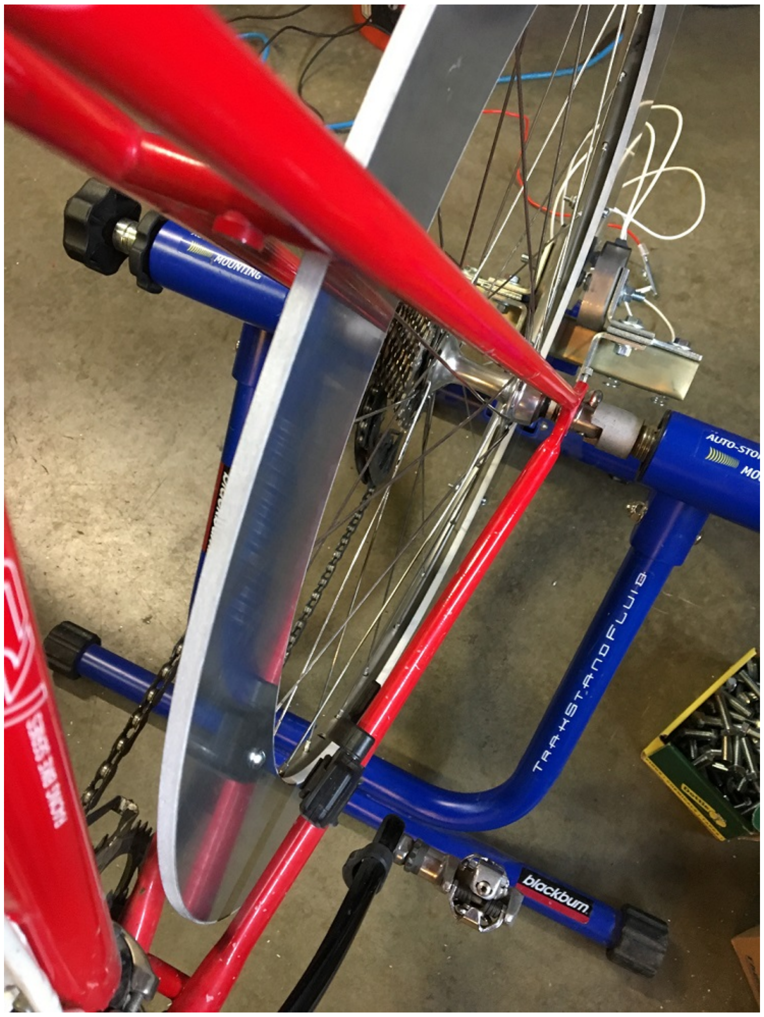


Figure 3.7: Aluminum donut mounted to rear wheel

controlled workout and can be seen in Figure 3.7. Another benefit of this modification is that the added weight of the donut causes the wheel momentum to smooth out the spinning of the rear wheel when resistance is applied.

3.1.2 Magnets, Power Supplies and Relay Control

The electromagnets need to be able to handle a low Voltage of Direct Current (VDC) but high current. For this reason, electromagnets from the automobile industry were selected. Trailer brake electromagnets run off of a 12-volt car battery and are made for high current. They are readily available at most automobile part stores and an example of one can be seen in Figure 3.8. Four of these electromagnets were mounted on the modified trainer.

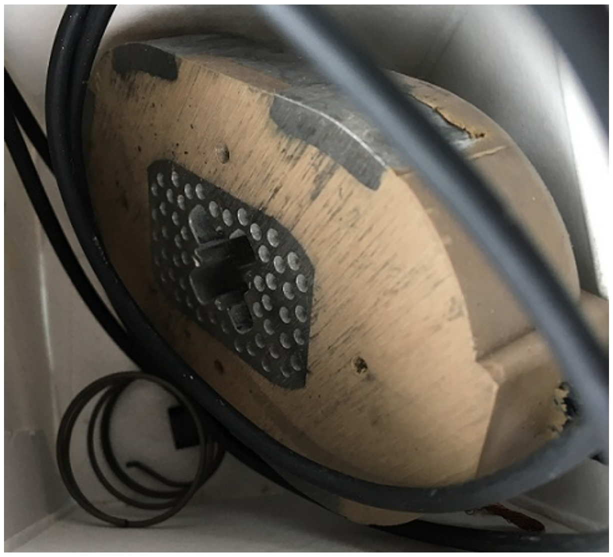


Figure 3.8: Automobile industry trailer brake electromagnets used in the eddy current brake

Now that the rear wheel had a thick, flat, aluminum donut attached and the electromagnets were as close as possible to the surface of the donut, the power supplies needed to be attached. For this application, two ELPAC Power Systems [33] power supplies, model ELV250-24, were used. Each power supply runs off of 120 Voltage of Alternating Current (VAC) and outputs 24 VDC at 10.4 Amperes (A) to power one set of two electromagnets. Each set of electromagnets were put in parallel to a power supply as in the schematic seen in Figure 3.9. This put 24 VDC across each electromagnet, but allows the maximum current of 10.4 A through each one as well. Figure 3.10 shows an example of one of the power supplies used. Appendix B has the wiring diagram for this usage.

At this point, with the power supplies on, there was maximum current flowing through all four electromagnets, which means there was a maximum resistance from the eddy current brake. The next step was to be able to control the the resistance level of the eddy current brake. The power supplies could not be varied, that is, they only output 12 VDC at 10.4 A. In order to vary the electromagnetic flux, the electromagnets were controlled by Pulse Width Modulation (PWM). This means the electromagnets will be turned on and off quickly at about 490 Hz. At 100% duty cycle the electromagnets was fully on and at 0% duty cycle they were fully off. This control of the 24 VDC, 10.4 A power supplies output was accomplished using two fast solid state relays.

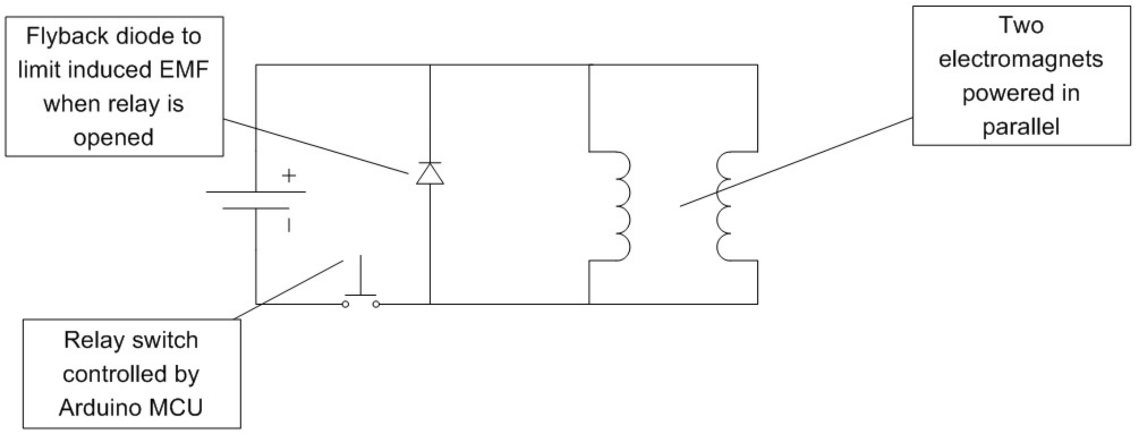

Flyback diode to limit induced EMF when relay is opened
Two electromagnets powered in parallel
Relay switch controlled by Arduino MCU
+

Figure 3.9: Schematic showing electromagnets in parallel to power supply. There are two of these setups for the Computer Controlled Eddy Current Brake Bicycle Trainer.

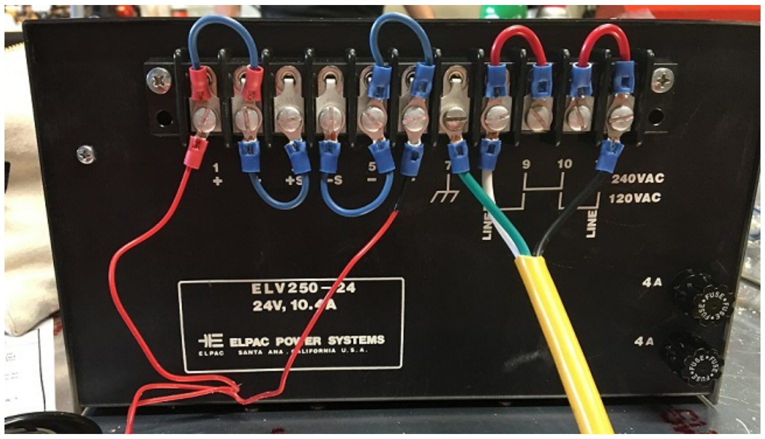


Figure 3.10: ELPAC power supply

A Power-IO [39] MOSFET relay, model HDD-1V25E, as seen in Figure 3.11 was used to control each power supply output to the electromagnets. The relays are rated for 100 VDC at 25 A with the control side using anywhere from 3 VDC to 32 VDC. Figure 3.12 shows a schematic of the how the MOSFET relay is added to the system. The flyback diode that is added across the load which in this case is the electromagnets is also shown in this figure. The flyback diode protects the circuit from the voltage spike that is generated by the collapsing magnetic field when the relay switch is opened [39].

3.2 MCU Controllers

Two MCU controllers were used for the Computer Controlled Eddy Current Brake Bicycle Trainer. One controller was specifically for controlling the PWM to the MOSFET relays. The

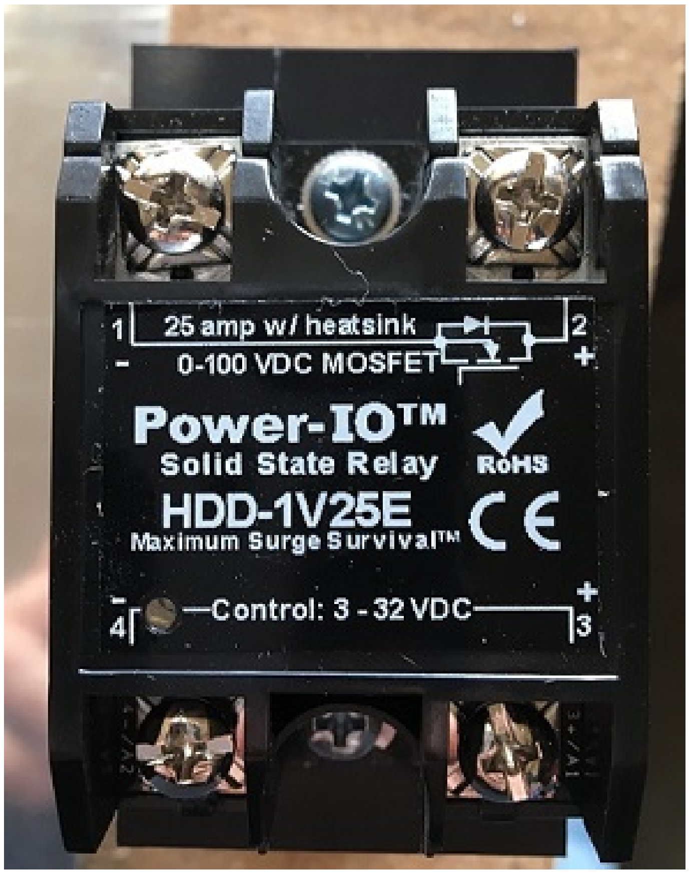


Figure 3.11: Power ID MOSFET relay

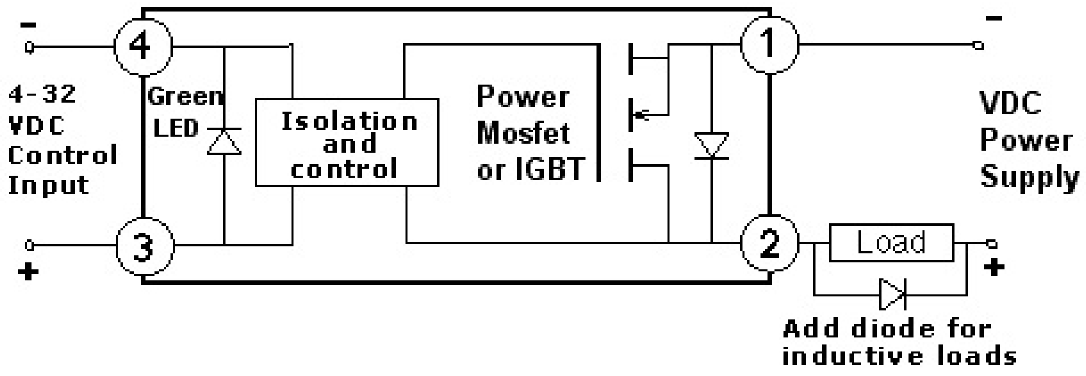


Figure 3.12: Schematic showing MOSFET relay connections (Source: [39])

second controller was used as a ANT+ application to receive feedback from the bicycle's power meter and to implement a computerized workout for the ride. The second controller, or trainer controller, decided if the eddy brake current needed to be increased or decreased while the first controller, or relay controller, was solely responsible for the PWM output to the relays.

3.2.1 Arduino Relay Controller

An Arduino Uno platform was used to control the MOSFET relays. The Arduino program simply monitored two digital lines from the trainer controller. One line indicated that the PWM output value to the MOSFET relays was to be lowered or the duty cycle lowered while the other line indicated the PWM value was to be raised or the duty cycle increased. If neither line had a signal, then no change was done and the PWM output to the relays remained unchanged. The coding for this Arduino application was done on the Arduino IDE. The Arduino IDE information screen and the code for this application can be seen in Appendix C.

Once the Arduino Uno was programmed, two output pins were connected to the relay control pins; pins 3 and 4 in Figure 3.12. The board itself was powered off the Universial Serial Bus (USB) port. The USB also was able to be used with the Arduino IDE Serial Monitor in order to see program informational output.

3.2.2 nRF51 Trainer Controller

The Nordic Semiconductor nRF51 development kit, as seen in Figure 3.13, was selected to be the trainer controller [35]. This development kit combines two Central Processing Unit (CPU)s and a 2.4 GHz radio for implementing the wireless communications. One CPU and the RF radio are packaged together on the board in a SoC, nRF51422, which is used to facilitate wireless ANT protocol and application development. There is also a general purpose advanced Reduced Instruction Set Computer (RISC) CPU, Atmel AT91SAM3 on the board, which is used as an interface MCU for programming, debugging, and updating the firmware on the nRF51422. This complete system is a SoM.

The IDE that was used to develop the application for the nRF51 is a third party application called Keil μVision [34] with the ARM MDK-Lite toolchain. Segger J-Link [40] was used to flash the application to the SoM board. This software also aided in debugging the application. This setup allowed for seamless development, debugging and flashing of the Computer Controlled Eddy Current Brake Bicycle Trainer code.

Other resources that were helpful in programming and debugging were nRFGo Studio which was downloaded from [35]. This allowed flashing of the latest Segger firmware to the nRF51 board CPU, AT91SAM3. ANTware II, download from [6], allowed direct configuration of the ANT radio and the ability to send and receive ANT and ANT+ packets. Also helpful in debugging and outputting program information was the telnet client, PuTTY [36]. Set up and information screens for these software packages can be seen in Appendix D.

Once the programming IDE was set up, the next step was to select the Software Development Kit (SDK) with the correct softdevice for the nRF51422 SoC. The softdevice implements the desired protocol which in this case is ANT and provides an Application Programming Interface (API) for application development. Based on the SoC nRF51422 and the protocol, ANT, softdevice S210 was selected [38]. The only SDK that supports softdevice S210 and SoC nRF51422 is SDK10 [35]. Figure 3.14 shows what the softdevice implements on the SoC nRF51422, and Figure 3.15 shows a block diagram of where the softdevice fits in for application development. In Figure 3.15, HW stands for hardware and ARM Cortex Microcontroller Software Interface Standard (CMSIS) is the hardware abstraction layer for the ARM CPU on SoC, nRF51422 [37].

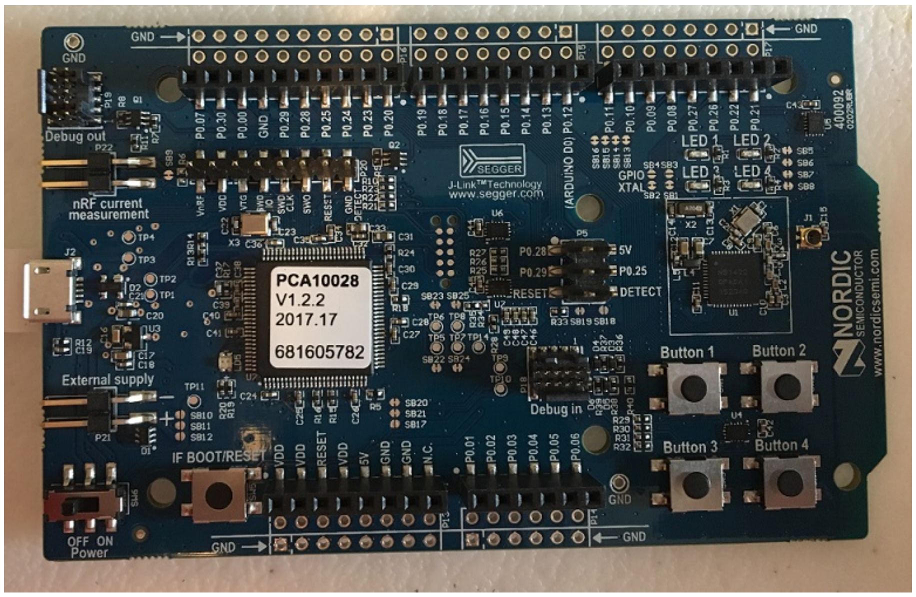


Figure 3.13: Nordic Semiconductor nRF51 development kit

The nRF51 development kit not only controlled the Arduino relay controller, but also received power feedback from the bicycle power meter. On the bicycle being used, there was a Stages Cycling power meter [41], model SPM-1, as seen in Figure 3.16, attached to the crank of the bicycle. This power meter detected crank deflection from the rider's pedal stroke. The power meter then transmitted the rider's detected power via ANT+. Since the information was flowing from the power meter, it was considered the master node and the SoC, nRF51422, was the slave node.

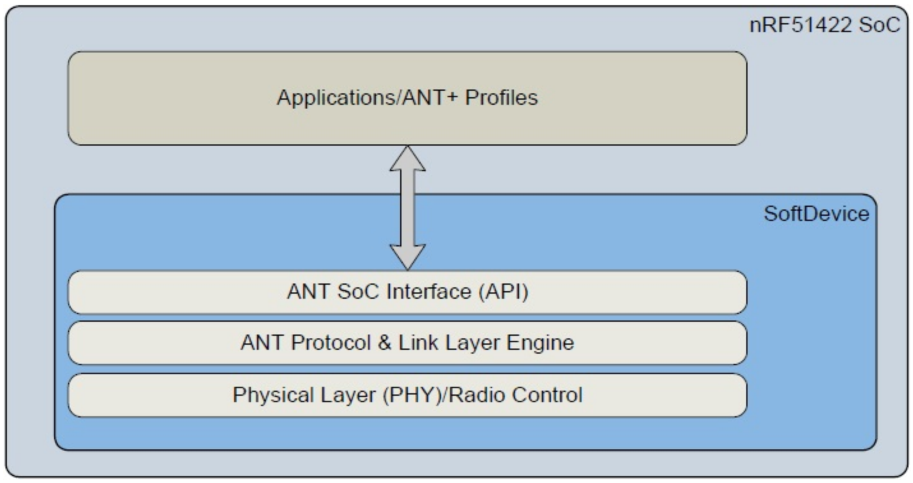

nRF51422 SoC
Applications/ANT+ Profiles
SoftDevice
ANT SoC Interface (API)
ANT Protocol & Link Layer Engine
Physical Layer (PHY)/Radio Control

Figure 3.14: Block diagram of softdevice protocol for SoC nRF51422 (Source: [38])

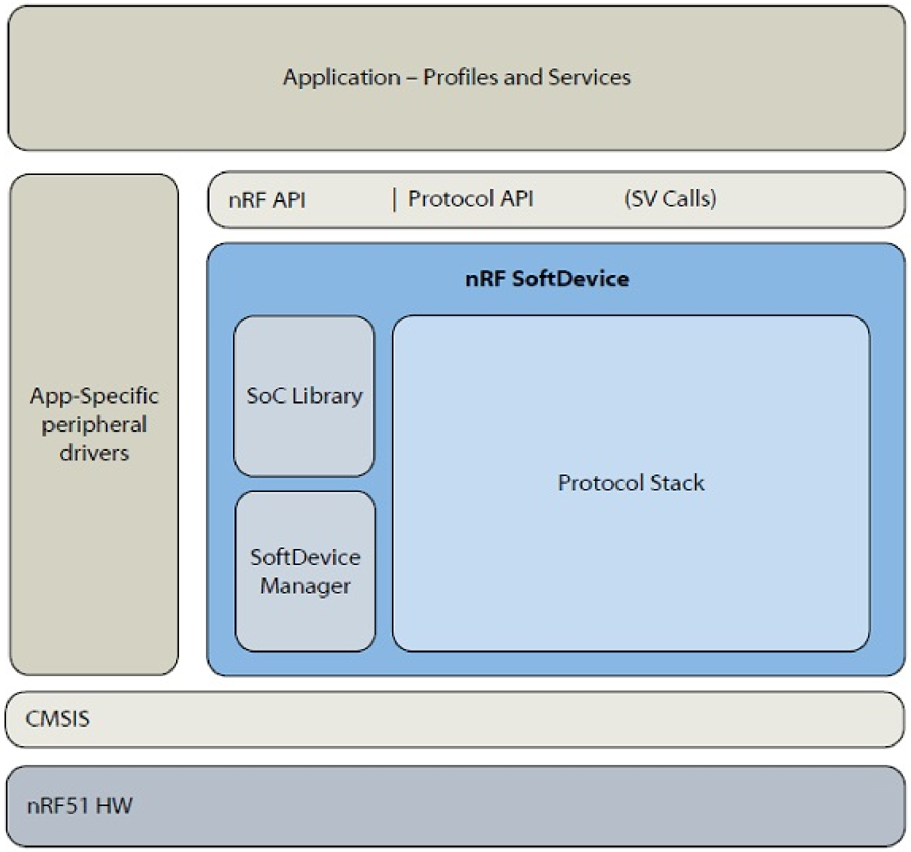


Figure 3.15: Block diagram of SoC application with softdevice (Source: [38])

Included with the SDK10 was an ANT+ example of a bicycle power meter receiver or slave node. See Appendix E for the directory where this software was found. This code was used for the basis of the Computer Controlled Eddy Current Brake Bicycle Trainer. Using this receiver

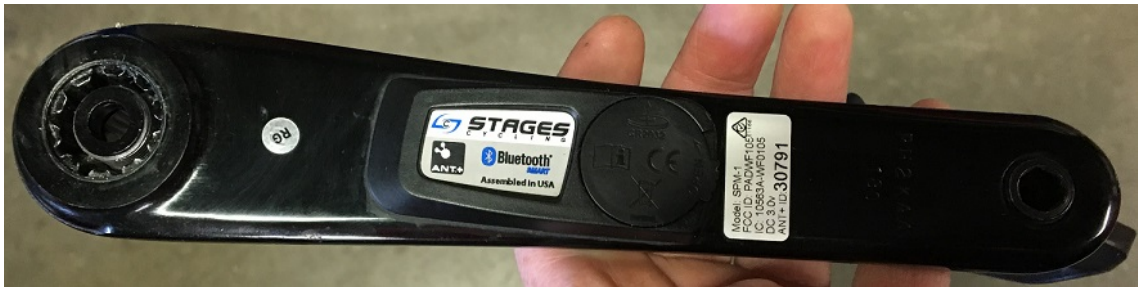


Figure 3.16: Bicycle crank with Stages Cycling power meter attached

code and the PuTTY serial terminal, an example of the Stages power meter output can be seen in Listing 3.1.

Listing 3.1: Stages power meter data

```
        B-PWR rx page: 16
        event count:                        206
        pedal power:                        --
        accumulated power:                  22363 W
        instantaneous power:                59 W
        instantaneous cadence:              59 rpm

        B-PWR rx page: 18
        Crank:
        event count:                        207
        tick:                               207
        period:                             30.115s
        accumulated torque:                 370.0Nm
        instantaneous cadence:              59 rpm

        B-PWR rx page: 16
        event count:                        208
        pedal power:                        --
        accumulated power:                  22475 W
        instantaneous power:                57 W
        instantaneous cadence:              59 rpm

        B-PWR rx page: 18
        Crank:
        event count:                        208
        tick:                               208
        period:                             31.125s
        accumulated torque:                 379.3Nm
        instantaneous cadence:              59 rpm
```

The ANT+ bicycle power profile data page 16 has the instantaneous bicycle power received via ANT+ [8]. This data page also included an event counter. The example code was changed so that when the event counter changed, the new instantaneous bicycle power was compared to the current load power from the computerized workout.

If the rider's power differed from the workout load power by more than a 10 Watt (W) hysteresis, then a digital General Purpose Input Output (GPIO) pin on the development kit was raised. Pin P0.07 was used to signal an increase in power to the eddy current brake and pin P0.30 was used to signal a decrease in power to the eddy current brake. These lines were monitored by the Arduino relay controller which in turn changed the PWM, either increasing or decreasing the duty cycle, to the power relays. The Arduino only monitored the rising edge of the signals from the nRF51 board. When the rising edge was detected, the PWM was increased or decreased by 5 depending on which digital pin was active. The range of the Arduino PWM output was kept between 0 and 255.

The Arduino relay control board ran off of 5 VDC while the nRF51 board ran off of 3 VDC. Since the signal lines needed to be a two different levels, a level converter was used to connect the two boards. Figure 3.17 shows a picture of the level converter while Figure 3.18 shows the nRF51 board and the Arduino connected together via the level converter.

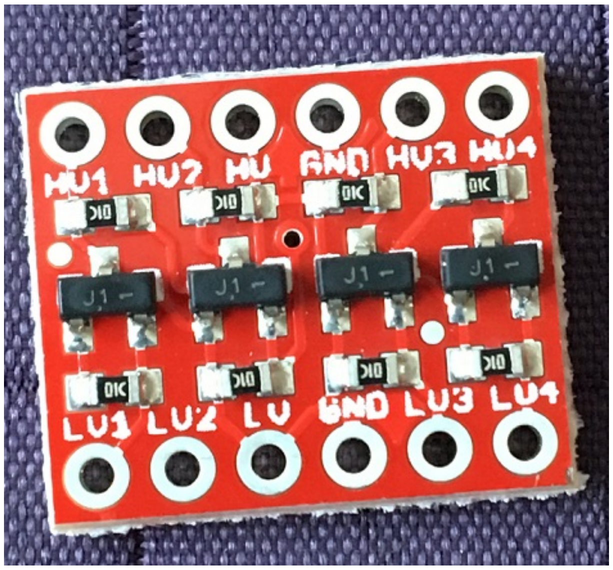


Figure 3.17: Level converter to convert signal from 3 VDC to 5 VDC

Figure 3.18 shows the yellow and black wires going off out of the picture. These are the wires that went to the MOSFET relays. The two boards were each powered through their respective USB connections, which were also used for serial output from the programs.

The computerized workout was implemented through a repeat interrupt timer in the nRF51 board API. Appendix E has the code for the set up of the timer and the interrupt handler which implements the computerized workout. At each interrupt, the output lines, P0.07 and P0.30 were set low again. This was to allow a new rising edge signal for the Arduino to detect if, at the next page 16 feedback event, the load power needed to be adjusted. Code examples

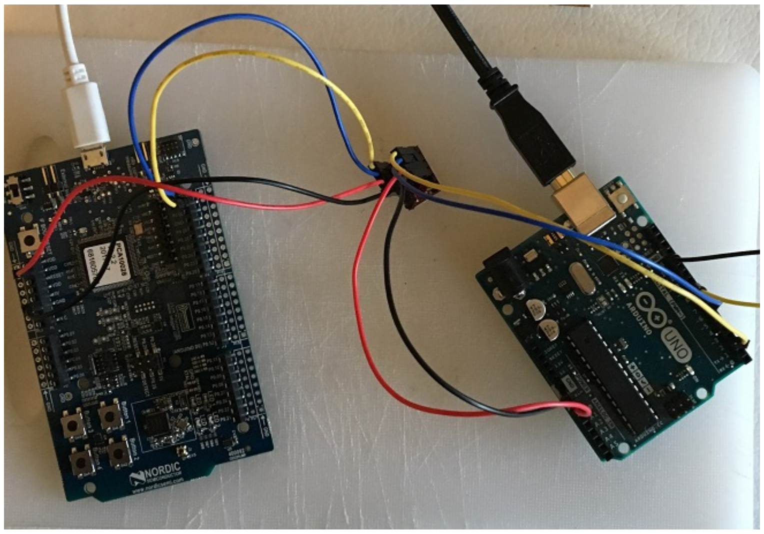


Figure 3.18: Arduino and nRF51 boards connected via a level converter

of the GPIO output pins setup, page 16 instantaneous power comparison, and the controller header file that was added to the project are given in Appendix E as well. Since part of the code in Appendix E is licensed, a copy of the license is given in Appendix F.

Chapter 4

Experimental Validation

The Computer Controlled Eddy Current Brake Bicycle Trainer was tested for proof of concept. As a review, the goal was to have the computer control the braking power of the eddy current brake based on a computerized workout. There needed to be enough braking power generated in order for the rider to feel the difference in resistance as the workout cycled through the different power levels.

The set-up of the power supplies and relays can be seen in Figure 4.1. The overall set-up with the bicycle in the trainer can be seen in Figure 4.2.

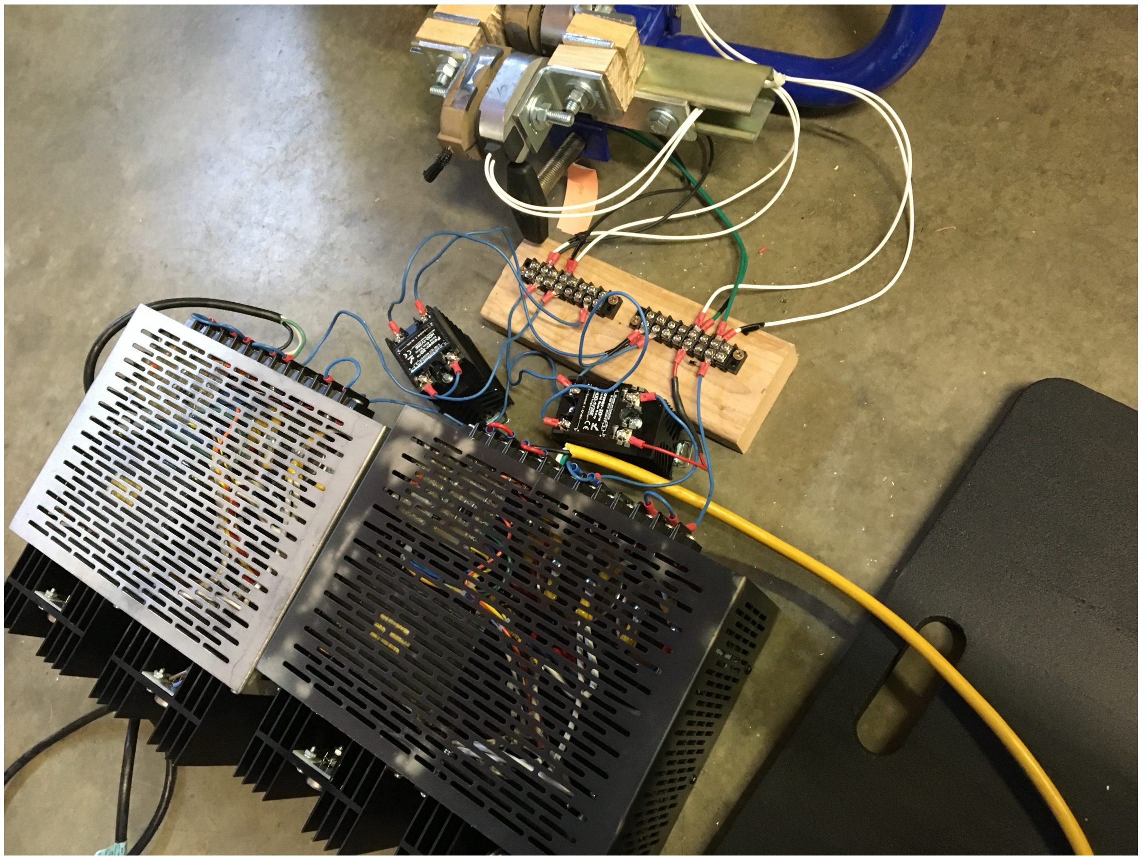


Figure 4.1: Set-up for the power supplies and relays to the electromagnets.

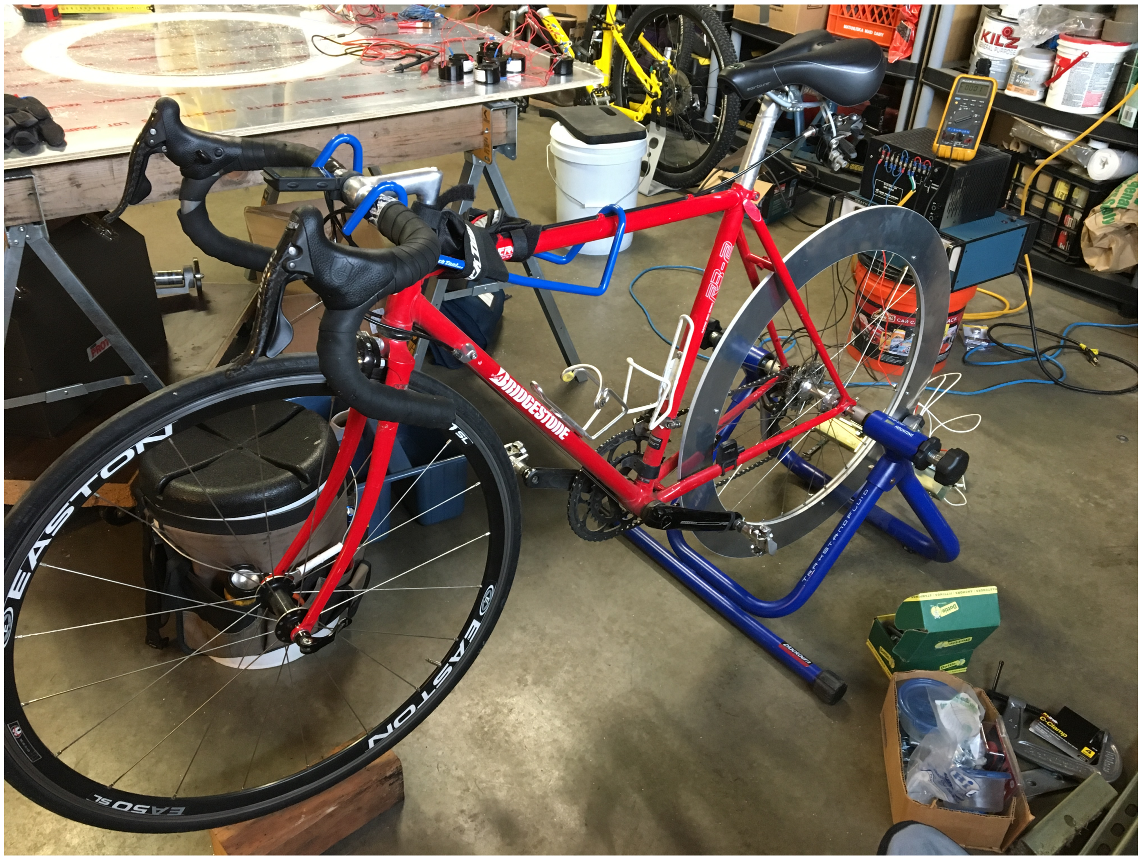


Figure 4.2: Set-up of the Computer Controlled Eddy Current Brake Bicycle Trainer.

4.1 Experimental Set-up

The computerized workout was called Rolling Hills and consisted of 20-second intervals that got progressively more difficult for 80 seconds. After 80 seconds, there were 20 seconds of no resistance to mimic riding down the hill and then the "hill" started over.

The experiment used a speedometer that was not part of the Computer Controlled Eddy Current Brake Bicycle Trainer, which was mounted on the bicycle. In addition, two amp meters were used to measure the current to each set of electromagnets. The goal was to ride a steady speed of 30 miles per hour (mph) as the workout resistance changed. The Stages power meter provided feedback to the trainer controller, but it was also used in a separate visual display to determine the rider's power output. Once the power changed, a few seconds were given to the rider to adjust to the new load and to make sure the speed was still at 30 mph. Once a steady speed was achieved, the current to the two sets of electromagnets was recorded, as were the

PWM values sent to the two relays. In addition to that data, the power from the Stages power meter was recorded.

4.2 Experimental Results

Table 4.1 shows the values that were collected during the test ride. The PWM column is the value that was sent to each relay and represents a specific duty cycle.

Table 4.1: Results of ride test on Computer Controlled Eddy Current Brake Bicycle Trainer

Speed (mph)	PWM	Magnet Set 1 (A)	Magnet Set 2 (A)	Rider power (W)	Workout power (W)
30	64	0.7	0.8	40	50
30	127	2.7	3.1	120	125
30	191	5.8	6.8	217	225
30	255	10.1	10.0	270	300

This table shows that the proof of concept worked. Each rider power reading is within the 10 W hysteresis of the workout power except for the final workout power of 300 where it is 30 W off of the rider's power. It is worth noting that 270 W is the maximum power output of this set-up. The last row of Table 4.1 shows a PWM value of 255 for both relays, which means the duty cycle is 100% and the maximum current is going through all electromagnets. This is enough power for a proof of concept, but not for a commercial trainer.

Chapter 5

Conclusion and Future Work

This thesis was the proof of concept for a computerized indoor bicycle trainer that combined minimal set up for and minimal wear and tear on a bicycle. It made use of a power meter connected sensor to provide feedback to the trainer controller, which in turn signaled whether to increase or decrease resistance through the use of an eddy current brake affecting the rear wheel of the bicycle. The proof of concept worked; however, there are many improvements and more developments that can be made.

The proof of concept eddy current brake only produced a maximum of 270 W. This is far below the standard for indoor eddy current trainers which can produce up to 2000 W of resistance. In this thesis, an aluminum donut was added to the wheel to increase the conductor material for eddy currents to develop. Developing more powerful electromagnets might be a method to do away with the need for the extra donut conductor. This can be done with more current through the electromagnets or by using electromagnets with an extremely high winding count. It is possible to develop powerful eddy currents in the wheel rim, as seen with the STAC Zero trainer, which used rare earth permanent magnets.

If the aluminum donut is still to be used in the future, quick connect clips can be made so as to not have the need to bolt it to the rim as this proof of concept did.

Finally, a communications link to a web workout can be developed in the future. This proof of concept design used a preprogrammmed workout in the trainer controller which is difficult to change, as it requires programming and flashing the nRF51 development kit. If the trainer controller can be changed to receive a workout signal and then compare that to the power feedback in order to signal the relay control, then computerized workouts from the web or another computer can be used.

Bibliography

[1] R. Van der Plas, “The Bicycle Racing Guide,” San Francisco, CA: Bicycle Books Inc, 1986.

[2] R. Sleamaker, “Serious Training for Serious Athletes,” Champaign, IL: Leisure Press, 1989.

[3] Sports Technology and Athletics Consulting, “STAC Zero User Manual,” Rev 1.4, August 30, 2017.

[4] STAC Zero website: http://www.staczero.com/.

[5] S. Gharghan, R. Nordin, M. Ismail, “A Survey on Energy Efficient Wireless Sensor Networks for Bicycle Performance Monitoring Application,” Journal of Sensors, vol. 2014.

[6] ANT Alliance website: http://www.thisisant.com

[7] “ANT Channel Search,” rev 3.0, thisisant.com, Dynastream Innovations Inc., 2016.

[8] “ANT+ Device Profile Bicycle Power,” rev. 5.0, thisisant.com, Dynastream Innovation Inc., 2007-2016.

[9] “ANT Frequency Agility,” rev 2.2, thisisant.com, Dynastream Innovations Inc., 2011.

[10] “ANT Message Protocol and Usage,” rev 5.1, thisisant.com, Dynastream Innovations Inc., 2014.

[11] P. Horowitz, W. Hill, “The Art of Electronics,” 2nd ed., New York, NY: Cambridge University Press, 1989.

[12] J. S. Lee, Y. W. Su and C. C. Shen, “A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi,” IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society, Taipei, 2007, pp. 46-51.

[13] C. Stoica, L. Melcescu, E. Lefter and L. M. Constantinescu, “Computation of the characteristics eddy current electromagnetic brake for a bicycle, by the finite element method 3D,” 2010 12th International Conference on Optimization of Electrical and Electronic Equipment, Basov, 2010, pp. 436-440.

[14] S. K. Gharghan, R. Nordin and M. Ismail, “Design Consideration of an Energy Efficient Wireless Sensor Network for High Performance Track Cycling,” 2014 International Conference on Information Science & Applications (ICISA), Seoul, 2014, pp. 1-5.

[15] S. Gasparrini, E. Gambi and S. Spinsante, “Evaluation and possible improvements of the ANT protocol for home heart monitoring applications,” Measurements and Networking Proceedings (M&N), 2013 IEEE International Workshop on, Naples, 2013, pp. 214-219.

[16] S. Gerez, “Implementation of Digitial Signal Processing: Some Background on GFSK Modulation,” lecture notes ver 5, University of Twente, Netherlands, March 9, 2016.

[17] J. T. Adams, “An introduction to IEEE STD 802.15.4,” 2006 IEEE Aerospace Conference, Big Sky, MT, 2006, pp. 8.

[18] A. Zaher, T. Aasebo, J. Noll, “Near Field Communication, Bluetooth, ZigBee, and ANT+,” lecture notes, Department of Technology Systems at the University of Oslo, Norway, Sept. 9, 2013.

[19] “nRF51422 Multiprotocol ANT/Bluetooth low energy System on Chip Product Specification,” ver. 3.2, Nordic Semiconductor, 2014.

[20] C. Gomez, J. Oller, J. Paradells, “Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology,” Sensors, vol 12, issue 9, 2012, pp. 11734-11753.

[21] P. Tipler, “Physics,” vol 2, 2th ed., New York, NY: Worth Publishers, Inc., Jul 1982.

[22] A. Dementyev, S. Hodges, S. Taylor and J. Smith, “Power consumption analysis of Bluetooth Low Energy, ZigBee and ANT sensor nodes in a cyclic sleep scenario,” 2013 IEEE International Wireless Symposium (IWS), Beijing, 2013, pp. 1-4.

[23] S. Khssibi, H. Idoudi, A. Van Den Bossche, T. Val, and L. A. Saidane, “Presentation and analysis of a new technology for low power wireless sensor network,” International Journal of Digital Information and Wireless Communications, vol. 3, no. 1, pp. 7586, 2013.

[24] Product review website: https://www.dcrainmaker.com/2016/06/stac-zero-trainer.html

[25] M. Naeve, “IEEE 802.15.4 MAC Overview,” Eaton Corporation, May 10, 2004.

[26] “IEEE Standard for Low-Rate Wireless Networks,” in IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011) , vol., no., pp.1-709, April 22, 2016.

[27] C. Underhill, “Solenoids Electromagnets and Electromagnetic Windings,” 3rd ed., New York, NY: D. Van Nostrand Company, 1918.

[28] M. Ericsson, "Transmission, Storage, and Visualization of Data with ANT+," M.S. thesis, Technical University of Linkoping, Sweden, 2015.

[29] F. Sears, M. Zemansky, H. Young, “University Physics,” 6th ed., Boston, MA: Addison-Wesley Publishing Comp. Inc., 1984.

[30] L. Frenzel, “What’s the Difference Between IEEE 802.15.4 and ZigBee Wireless,” Electronic Design, Mar. 2013.

[31] H. Fahmy, “Wireless Sensor Networks Concepts, Applications, Experimentation and Analysis”, Springer, Singapore, 2016.

[32] Arduino website: https://www.arduino.cc/en/Guide/HomePage

[33] ELPAC Power Systems owned by Inventus Power website: https://inventuspower.com/

[34] Keil μVision website: http://www.keil.com/

[35] Nordic Semiconductor website: https://www.nordicsemi.com/

[36] PuTTY telenet client website: http://www.putty.org/

[37] “nRF51 Series Reference Manual”, rev 3.0.1, Nordic Semiconductor, Dec. 2016.

[38] “S210 nRF51422 ANT SoftDevice, SoftDevice Specification,” ver 3.0, Nordic Semiconductor, Jul. 2015.

[39] Power-IO website: http://www.power-io.com/power-io.htm

[40] Segger Microcontroller website: https://www.segger.com

[41] Stages Cycling website: https://stagescycling.com/us/

[42] Unistrut Metal Framing System website: http://www.unistrut.us/

Appendix A

Bill of Materials

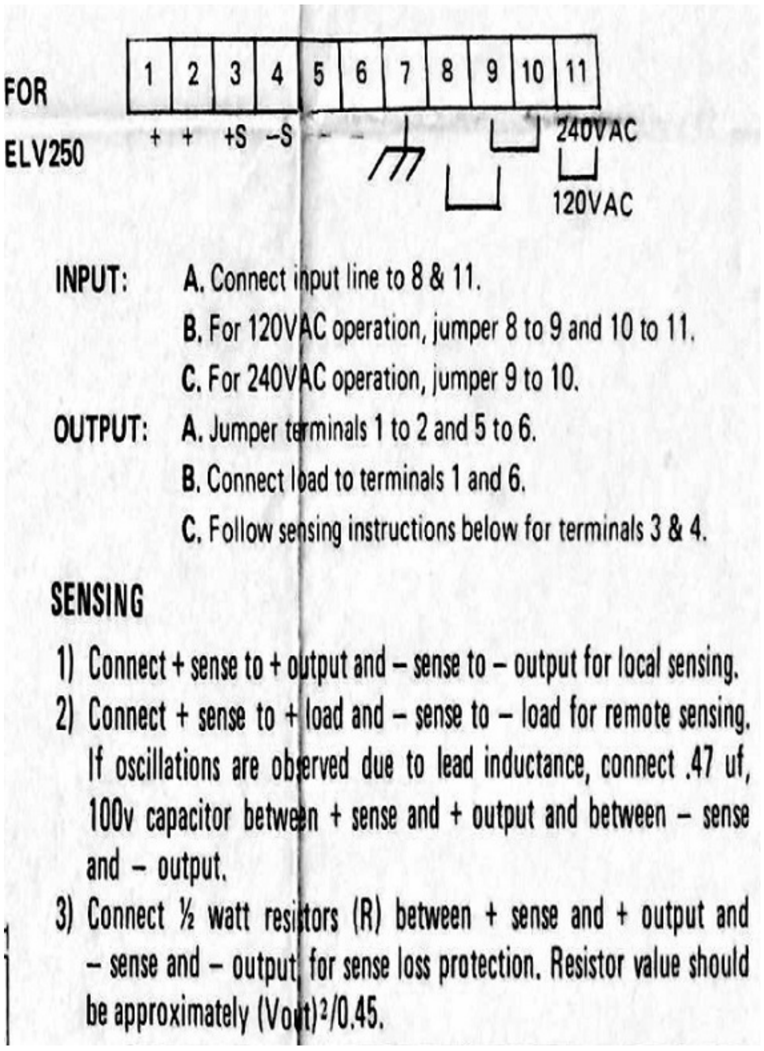
Below is a bill of materials list for Computer Controlled Eddy Current Brake Bicycle Trainer. Please note that simple items such as fasteners and wiring have not been listed.

1. Bicycle trainer that holds bicycle by rear wheel skewer with friction cylinder removed
2. 4 trailer brake electromagnets
3. Unistrut and bracket hardware to hold electromagnets
4. 2 Power supplies, ELPAC ELV250, 24 VDC, 10.4 A
5. 2 power relays, Power-IO HDD-1V25E, 100 VDC, 25 A
6. 2 flyback diodes
7. Level Converter
8. Arduino Uno
9. nRF51 Development Kit with SoC nRF51422
10. Bicycle with Stages Cycling crank power meter
11. Aluminum donut, 0.6 cm thick, inner radius of 27 cm, outer radius of 34 cm

Appendix B

ELPAC Power Supply Wiring

Figure B.1 shows how to wire the ELPAC ELV250 power supply for the Computer Controlled Eddy Current Brake Bicycle Trainer.


FOR ELV250
1 2 3 4 5 6 7 8 9 10 11
+ + +S −S − −
240VAC
120VAC

INPUT: A. Connect input line to 8 & 11.
B. For 120VAC operation, jumper 8 to 9 and 10 to 11.
C. For 240VAC operation, jumper 9 to 10.

OUTPUT: A. Jumper terminals 1 to 2 and 5 to 6.
B. Connect load to terminals 1 and 6.
C. Follow sensing instructions below for terminals 3 & 4.

SENSING

1) Connect + sense to + output and – sense to – output for local sensing.
2) Connect + sense to + load and – sense to – load for remote sensing. If oscillations are observed due to lead inductance, connect .47 uf, 100v capacitor between + sense and + output and between – sense and – output.
3) Connect ½ watt resistors (R) between + sense and + output and – sense and – output for sense loss protection. Resistor value should be approximately (Vout)^2/0.45.

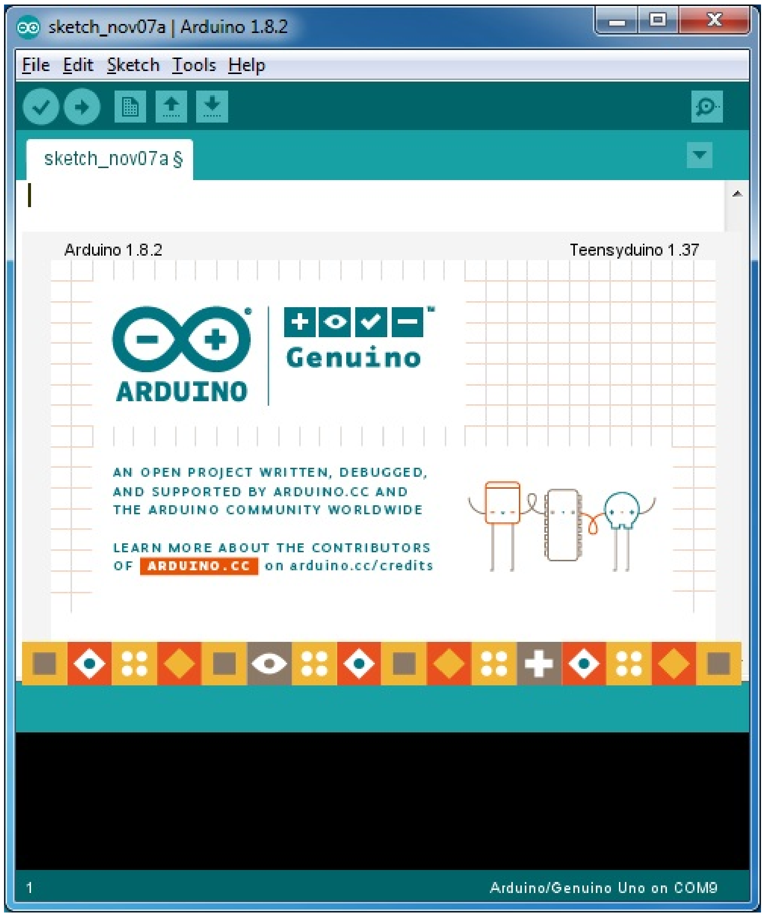
Figure B.1: Wiring for the ELPAC ELV250 power supply (Source: [39])

Appendix C

Arduino Relay Control

C.1 IDE

The Arduino IDE can be downloaded from the Arduino website [32]. The IDE information is shown in Figure C.1.


sketch_nov07a | Arduino 1.8.2
File Edit Sketch Tools Help
sketch_nov07a §
Arduino 1.8.2
Teensyduino 1.37
ARDUINO
Genuino
ARDUINO.CC
1
Arduino/Genuino Uno on COM9

AN OPEN PROJECT WRITTEN, DEBUGGED, AND SUPPORTED BY ARDUINO.CC AND THE ARDUINO COMMUNITY WORLDWIDE

LEARN MORE ABOUT THE CONTRIBUTORS OF ARDUINO.CC on arduino.cc/credits

Figure C.1: Arduino IDE information

C.2 Arduino Code

This program code monitors two digital input lines, and, based on their conditions, modifies two PWM output lines that control the MOSFET relays.

```

// input pins
#define UP_PIN     7    // signal on this pin means increase PWM (nRF P0.07)
#define DOWN_PIN   2    // signal on this pin means decrease PWM (nRF P0.30)
```

```
// output pins
#define OUT_PIN1   9    // PWM output to switch 1
#define OUT_PIN2   13   // PWM output to switch 2

// input pin states
int upState = 0;          // current up pin state
int lastUpState = 0;      // previous up pin state
int downState = 0;        // current down pin state
int lastDownState = 0;   // previous down pin state

// output values for PWM
int value1PWM = 0;        // switch 1 PWM
int value2PWM = 0;        // switch 2 PWM

// function to serial write for debugging
void write_value(int val1, int val2);

void setup()
{
  // serial output for debugging
  Serial.begin(9600);
  Serial.println("BEGIN!");

  // pin config
  pinMode(OUT_PIN1, OUTPUT);
  pinMode(OUT_PIN2, OUTPUT);
  pinMode(UP_PIN, INPUT); // regular low unless set hi
  pinMode(DOWN_PIN, INPUT);

  // set start PWM's
  analogWrite(OUT_PIN1, value1PWM);
  analogWrite(OUT_PIN2, value2PWM);

  write_value(value1PWM, value2PWM);
}

void write_value(int val1, int val2)
{
  Serial.print("PWM 1 output = ");
  Serial.println(val1, DEC);
  Serial.print("PWM 2 output = ");
  Serial.println(val2, DEC);
}

void loop()
{
  // let's look at the input pins
  upState = digitalRead(UP_PIN);
  downState = digitalRead(DOWN_PIN);

  // check for state changes
  if(upState != lastUpState)
  {
    lastUpState = upState;
    if(upState == HIGH)
    {
      value1PWM += 5;
      if(value1PWM > 255)
```

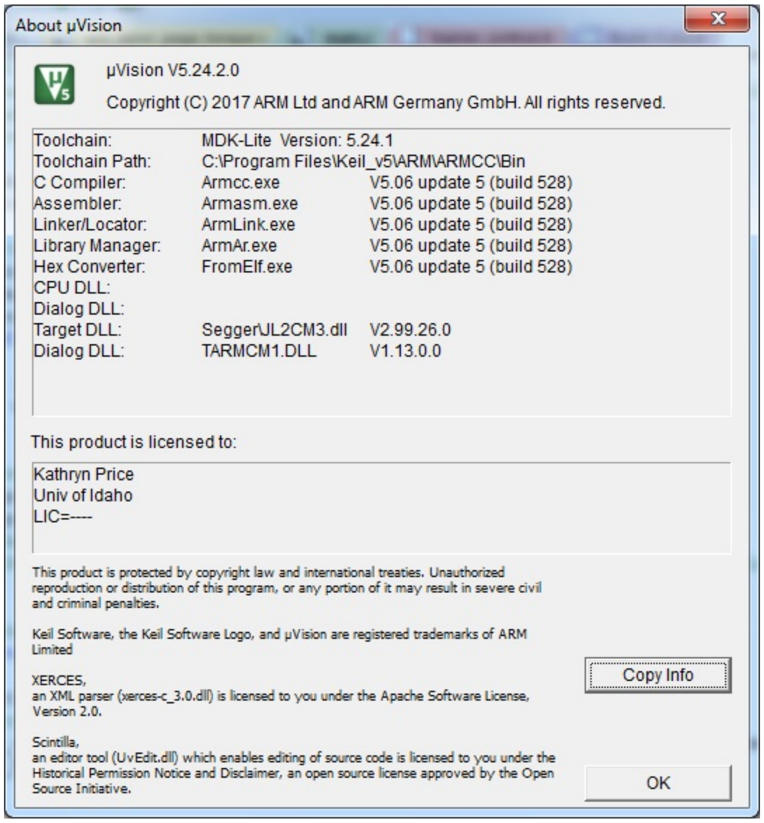
```
            {
                value1PWM = 255;
            }
        }
    }
    if(downState != lastDownState)
    {
        lastDownState = downState;
        if(downState == HIGH)
        {
            value1PWM -= 5;
            if(value1PWM < 0)
            {
                value1PWM = 0;
            }
        }
    }

    // write new values to switches
    value2PWM = value1PWM;  // for now set both switches to the
        same value
    analogWrite(OUT_PIN1, value1PWM);
    analogWrite(OUT_PIN2, value2PWM);
    write_value(value1PWM, value2PWM);
}
```

Appendix D

nRF51 Programming Software

The Keil μVision MDK-Lite IDE can be downloaded from the Keil μVision website [34]. This download installs the ARM toolchain as well as the IDE. Once the IDE is installed, the device to be programmed needs to be selected.

Figure D.1 shows the set up information for Keil μVision IDE.



µVision V5.24.2.0

Copyright (C) 2017 ARM Ltd and ARM Germany GmbH. All rights reserved.

Toolchain:	MDK-Lite Version: 5.24.1	
Toolchain Path:	C:\Program Files\Keil_v5\ARM\ARMCC\Bin	
C Compiler:	Armcc.exe	V5.06 update 5 (build 528)
Assembler:	Armasm.exe	V5.06 update 5 (build 528)
Linker/Locator:	ArmLink.exe	V5.06 update 5 (build 528)
Library Manager:	ArmAr.exe	V5.06 update 5 (build 528)
Hex Converter:	FromElf.exe	V5.06 update 5 (build 528)
CPU DLL:		
Dialog DLL:		
Target DLL:	Segger\JL2CM3.dll	V2.99.26.0
Dialog DLL:	TARMCM1.DLL	V1.13.0.0

This product is licensed to:

Kathryn Price
Univ of Idaho
LIC=----

This product is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it may result in severe civil and criminal penalties.

Keil Software, the Keil Software Logo, and µVision are registered trademarks of ARM Limited

XERCES,
an XML parser (xerces-c_3.0.dll) is licensed to you under the Apache Software License, Version 2.0.

Scintilla,
an editor tool (UvEdit.dll) which enables editing of source code is licensed to you under the Historical Permission Notice and Disclaimer, an open source license approved by the Open Source Initiative.

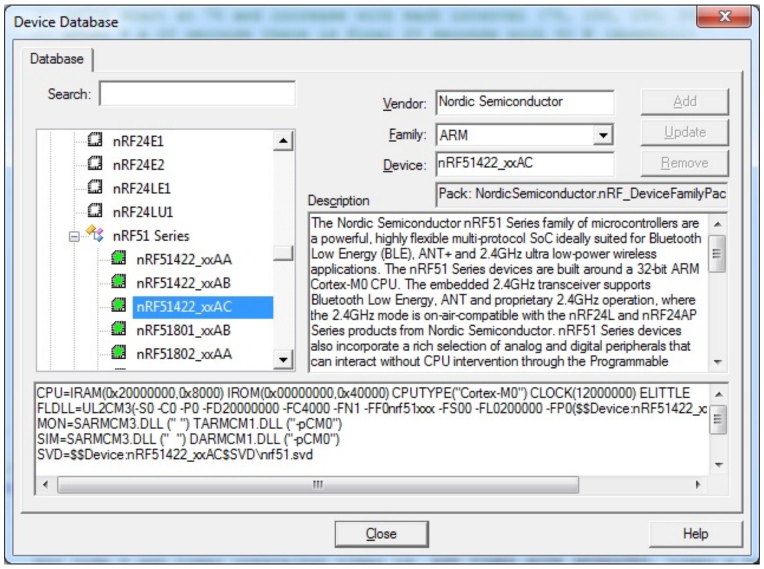
Figure D.1: Set up information for Keil μVision

Figure D.2 shows the device selection for the nRF51 board. In this case, the Nordic Semiconductor SoC nRF51422 is being selected.

Figure D.3 shows the application ANTware II. This software allows maual configuration of the ANT radios. It also allows transmission and reception of the ANT and ANT+ packets.

Figure D.4 shows the nRFgo application. This application allowed the flashing of the softdevice to run the ANT protocol to the nRF51422 SoC.

Figure D.5 shows the communications serial line configuration for communicating with the nRF51 development kit via PuTTY.

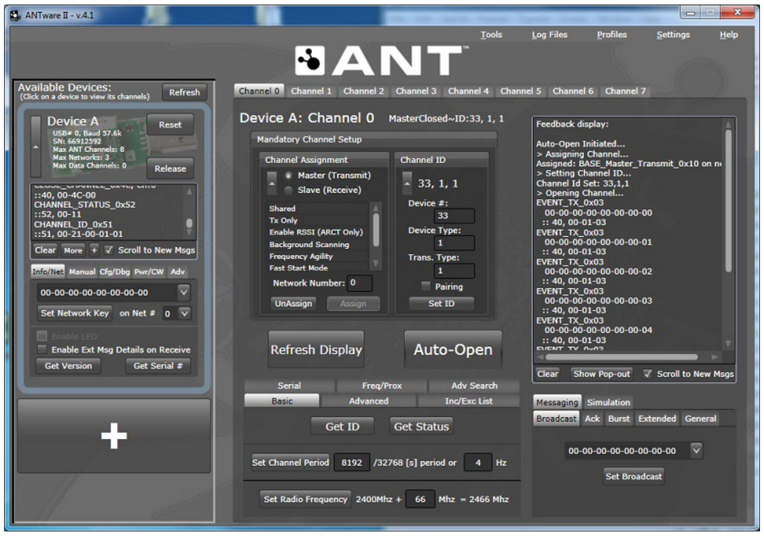


Vendor: Nordic Semiconductor
Family: ARM
Device: nRF51422_xxAC
Pack: NordicSemiconductor.nRF_DeviceFamilyPac

Description
The Nordic Semiconductor nRF51 Series family of microcontrollers are a powerful, highly flexible multi-protocol SoC ideally suited for Bluetooth Low Energy (BLE), ANT+ and 2.4GHz ultra low-power wireless applications. The nRF51 Series devices are built around a 32-bit ARM Cortex-M0 CPU. The embedded 2.4GHz transceiver supports Bluetooth Low Energy, ANT and proprietary 2.4GHz operation, where the 2.4GHz mode is on-air-compatible with the nRF24L and nRF24AP Series products from Nordic Semiconductor. nRF51 Series devices also incorporate a rich selection of analog and digital peripherals that can interact without CPU intervention through the Programmable

```
CPU=IRAM(0x20000000,0x8000) IROM(0x00000000,0x40000) CPUTYPE("Cortex-M0") CLOCK(12000000) ELITTLE
FLDLL=UL2CM3(-S0 -C0 -P0 -FD20000000 -FC4000 -FN1 -FF0nrf51xxx -FS00 -FL0200000 -FP0($$Device:nRF51422_x
MON=SARMCM3.DLL (" ") TARMCM1.DLL ("-pCM0")
SIM=SARMCM3.DLL (" ") DARMCM1.DLL ("-pCM0")
SVD=$$Device:nRF51422_xxAC$SVD\nrf51.svd
```

Figure D.2: Device selection for project from Keil μVision



Device A
USB# 0, Baud 57.6k
SN: 66912592
Max ANT Channels: 8
Max Networks: 3
Max Data Channels: 0

```
::40, 00-4C-00
CHANNEL_STATUS_0x52
::52, 00-11
CHANNEL_ID_0x51
::51, 00-21-00-01-01
```

00-00-00-00-00-00-00-00
Set Network Key on Net # 0
Enable Ext Msg Details on Receive

Device A: Channel 0 MasterClosed~ID:33, 1, 1

Channel Assignment: (●) Master (Transmit) ( ) Slave (Receive)
Shared
Tx Only
Enable RSSI (ARCT Only)
Background Scanning
Frequency Agility
Fast Start Mode
Network Number: 0

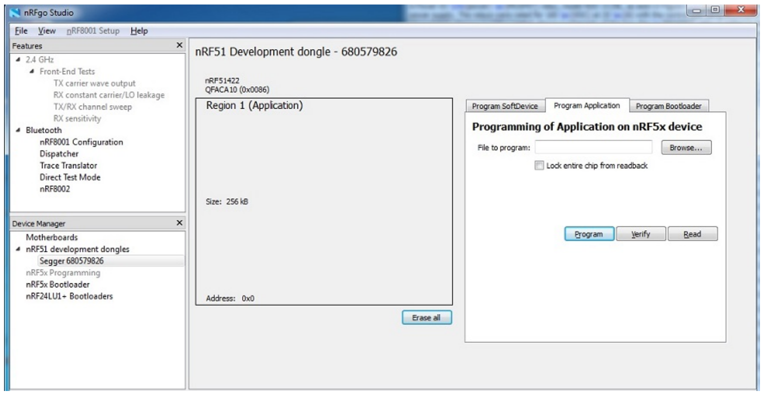
Channel ID: 33, 1, 1
Device #: 33
Device Type: 1
Trans. Type: 1
Pairing

Feedback display:

```
Auto-Open Initiated...
> Assigning Channel...
Assigned: BASE_Master_Transmit_0x10 on n
> Setting Channel ID...
Channel Id Set: 33,1,1
> Opening Channel...
EVENT_TX_0x03
  00-00-00-00-00-00-00-00
 :: 40, 00-01-03
EVENT_TX_0x03
  00-00-00-00-00-00-00-01
 :: 40, 00-01-03
EVENT_TX_0x03
  00-00-00-00-00-00-00-02
 :: 40, 00-01-03
EVENT_TX_0x03
  00-00-00-00-00-00-00-03
 :: 40, 00-01-03
EVENT_TX_0x03
  00-00-00-00-00-00-00-04
 :: 40, 00-01-03
```

Set Channel Period 8192 /32768 [s] period or 4 Hz
Set Radio Frequency 2400Mhz + 66 Mhz = 2466 Mhz
00-00-00-00-00-00-00-00

Figure D.3: ANTware II software example



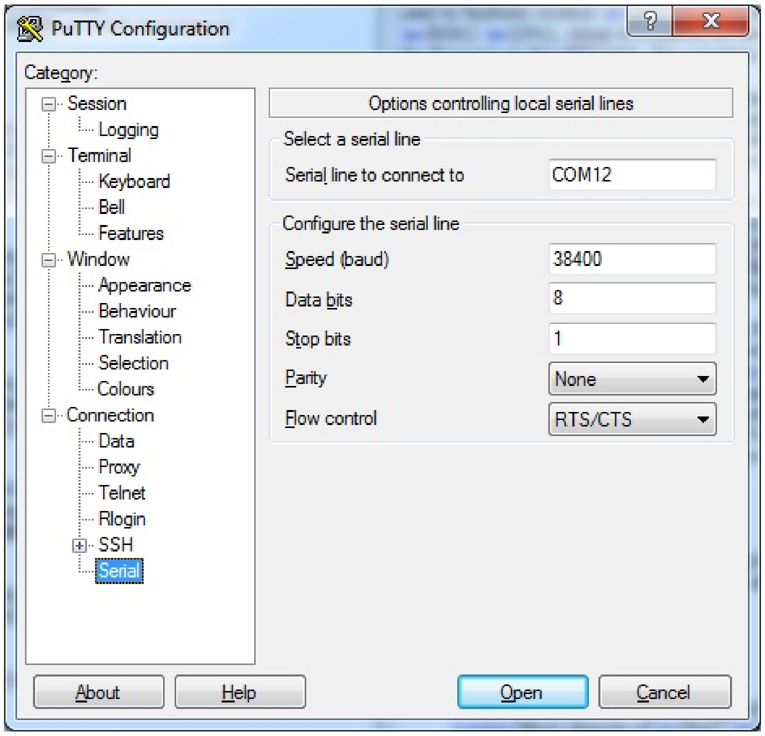
nRF51 Development dongle - 680579826

nRF51422
QFACA10 (0x0086)

Region 1 (Application)
Size: 256 kB
Address: 0x0

Programming of Application on nRF5x device
File to program:
Lock entire chip from readback

Figure D.4: nRFgo software example



Options controlling local serial lines

Select a serial line

Serial line to connect to: COM12

Configure the serial line

Speed (baud): 38400

Data bits: 8

Stop bits: 1

Parity: None

Flow control: RTS/CTS

Figure D.5: PuTTY serial communication configuration for the nRF51 board

Appendix E

nRF51 Code

E.1 Workout Interrupt

This code uses the SDK10 interrupt API to create a computerized workout. This section of code is located before the start of the main function.

```
// create app timer for workout updates
APP_TIMER_DEF(wo_timer_id);

int wattArray[5] = {50, 125, 225, 300, 0}; // -klp rolling hill
    watt level

// timeout handler for repeat timer
static void timer_a_handler(void * p_context)
{
        // Rolling Hills
        // we set workout Watts here
        // 4 x 20 seconds intervals
        // watts start at 50 and increase with each interval
            (50, 125, 225, 300)
        // after 4 x 20 seconds there is final 20 seconds with 0
            W (downhill)
        // repeat

        workoutCnt++;
        if(workoutCnt > 100)
        {
            //we're at 20 seconds
            workoutCnt = 0;
            requestedLoad = wattArray[workoutIndex] +
                intensityOffset;
            workoutIndex++;
            if(workoutIndex > 4)
            {
                // repeat 80 second at different levels for
                    rolling hills
                // then 20 seconds for the downhill
                workoutIndex = 0;
            }
        }
        // reset all outputs
        output_off(LOAD_TOO_HI);
        output_off(LOAD_TOO_LOW);
}


// Create timers
static void create_wo_timer(void)
{
        uint32_t err_code;
        // create timers
        err_code = app_timer_create(&wo_timer_id,
            APP_TIMER_MODE_REPEATED, timer_a_handler);
        APP_ERROR_CHECK(err_code);
}
```

E.2 Start Workout Interrupt

This code is located inside of the main function before the program loop starts. It begins the interrupt timer for the workout routine.

```
    // begin workout
    create_wo_timer();
    workoutIndex = 0;
    workoutCnt = 0;
    err_code = app_timer_start(wo_timer_id, APP_TIMER_TICKS(200,
        APP_TIMER_PRESCALER), NULL);
    APP_ERROR_CHECK(err_code);
```

E.3 Trainer Control Infomation

This file, trainer_control.h, defines several globals used to determine and control output to the Arduino relay control and should be included in the file with the main function as well as the file with the ANT+ page 16 functions.

```
/* trainer_control.h
 *
 * Globals used to determine and control output to
 * the Arduino
 */

/* plus or minus Watt difference allowed in feedback (0-240) */
#define LOAD_HYSTERESIS           10
#define LOAD_TOO_LOW                   7       // Arduino pin 7
#define LOAD_TOO_HI                    30   // Arduino pin 2

static int requestedLoad = 50; // watt load that the workout is
    requiring
static int intensityOffset = 0; // user can raise or lower the
    total intensity by 50 Watts
//static int
static uint8_t workoutIndex = 0; // selects watt level from
    array
static int workoutCnt = 0; // interrupt count used to estimate
    20 seconds
```

E.4 Output Pins

This code shows the functions used to set up and control the digital GPIO pins which are used to communicate with the Arduino relay control. This code should go before the main function.

```
void output_setup(uint32_t pin)
{
    nrf_gpio_cfg_output(pin);
}

void output_on(uint32_t pin)
{
    nrf_gpio_pin_set(pin);
}

void output_off(uint32_t pin)
{
    nrf_gpio_pin_clear(pin);
}
```

E.5 Power Compare

This code is part of the Nordic Semiconductor SDK10 [35] example code for ANT+ bicycle power. It has been edited to compare the received instantaneous power to the computerized workout power. If the instantaneous power differs by more than plus or minus 10W of the computerized workout power, then the GPIO pins signal to the Arduino relay control to either raise or lower the PWM to the MOSFET relays.

The example code was found in the following directory:

Listing E.1: Directory for ANT+ example code

```
C:\Program Files\Nordic Semiconductor\nRF51 SDK10\examples\ant\
    ant_plus\ant_bpwr\bpwr_rx
```

Please see Appendix F for ANT+ licensing information.

```
/* Copyright (c) 2015 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic
     Semiconductor ASA.
 * Terms and conditions of usage are described in detail in
     NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the
     information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be
     removed from
 * the file.
 *
 */

#include "ant_bpwr_page_16.h"
#include "ant_bpwr_page_logger.h"
#include "app_util.h"
#include "nordic_common.h"

#include "trainer_control.h" // -klp

extern void output_on(uint32_t pin); // -klp
extern void output_off(uint32_t pin); // -klp

/**@brief bicycle power page 16 data layout structure. */
typedef struct
{
    uint8_t update_event_count;
    uint8_t pedal_power;
    uint8_t reserved;
    uint8_t accumulated_power[2];
    uint8_t instantaneous_power[2];
}ant_bpwr_page16_data_layout_t;

// used to make sure event counts are changing, this means we're
     getting real time data -klp
static uint8_t lastEventCount16 = 0;
static bool firstPage16Event = false;

static void page16_data_log(ant_bpwr_page16_data_t const *
    p_page_data)
{
        // reset outputs -klp
        output_off(LOAD_TOO_HI);
        output_off(LOAD_TOO_LOW);

        //function edited to signal arduino -klp
        uint8_t loadDiff = 0;
        if(lastEventCount16 != p_page_data->update_event_count)
        {
            if(lastEventCount16 != 0)
            {
                // we are looking for the first true update to
                    the event count;
                // this prevents looking at the first event
                    count as valid
                firstPage16Event = true;
            }
            lastEventCount16 = p_page_data->update_event_count;

            LOG_PAGE16("event count:                        %u\n\r
                ", p_page_data->update_event_count);

            if (p_page_data->pedal_power.byte != 0xFF)
            {
                    LOG_PAGE16("pedal power:
                                                %u %%\n\r",
                                            p_page_data->
                                                pedal_power.items.
                                                distribution);
            }
            else
            {
                    LOG_PAGE16("pedal power:
                                                --\n\r");
            }

            LOG_PAGE16("accumulated power:                  %u W\n
                \r", p_page_data->accumulated_power);
            LOG_PAGE16("instantaneous power:                %u W\n
                \r", p_page_data->instantaneous_power);

            if(firstPage16Event == true)
            { // We don't want to set feedback unless we get a
                true event change.
                if(p_page_data->instantaneous_power <
                    requestedLoad)
                {
                    loadDiff = requestedLoad - p_page_data->
                        instantaneous_power;
                    if(loadDiff > LOAD_HYSTERESIS)
                    {
                        // load is too low
                        output_off(LOAD_TOO_HI);
                        output_on(LOAD_TOO_LOW);
                    }
                }
                else
                {
                    if(p_page_data->instantaneous_power > requestedLoad)
                    {
                        loadDiff = p_page_data->instantaneous_power - requestedLoad;
                        if(loadDiff > LOAD_HYSTERESIS)
                        {
                            // load is too high
                            output_off(LOAD_TOO_LOW);
                            output_on(LOAD_TOO_HI);
                        }
                    }
                }
            }
        }
}


void ant_bpwr_page_16_encode(uint8_t                    * p_page_buffer,
                              ant_bpwr_page16_data_t const * p_page_data)
{
    ant_bpwr_page16_data_layout_t * p_outcoming_data =
        (ant_bpwr_page16_data_layout_t *)p_page_buffer;

    p_outcoming_data->update_event_count    = p_page_data->update_event_count;
    p_outcoming_data->pedal_power           = p_page_data->pedal_power.byte;

    UNUSED_PARAMETER(uint16_encode(p_page_data->accumulated_power,
                                   p_outcoming_data->accumulated_power));
    UNUSED_PARAMETER(uint16_encode(p_page_data->instantaneous_power,
                                   p_outcoming_data->instantaneous_power));

    page16_data_log(p_page_data);
}


void ant_bpwr_page_16_decode(uint8_t const          * p_page_buffer,
                              ant_bpwr_page16_data_t * p_page_data)
{
    ant_bpwr_page16_data_layout_t const * p_incoming_data =
        (ant_bpwr_page16_data_layout_t *)p_page_buffer;

    p_page_data->update_event_count    = p_incoming_data->update_event_count;
    p_page_data->pedal_power.byte      = p_incoming_data->pedal_power;
    p_page_data->accumulated_power      = uint16_decode(
        p_incoming_data->accumulated_power);
    p_page_data->instantaneous_power    = uint16_decode(
        p_incoming_data->instantaneous_power);

    page16_data_log(p_page_data);
}
```

Appendix F

ANT+ License