Implantable Motor Controller for Catheter-sized Ventricular Assist Devices

Kenneth Galindo, Elijah Mattheson, Johan Fausto-Medina, Antony Barzola

***Abstract*—The general purpose within this project was to establish a wireless design designated to receive transmitted power which in turn would control a motor. This motor is intended to go within the heart in order to simulate a heartbeat, which is done by having the motor controller cause the motor to pump blood at a designated rate. In order to successfully implement said idea, we needed to use wireless power transmitter and receiver. The receiving system that was created consisted of an internal resonator that was tuned for 6.78 MHz, the lowest frequency allowed without interference from other medical devices. The receiver was coupled to the internal resonator, which produces an AC voltage. The AC voltage is then fed to a rectifier, the CBRDFSH2-100 bridge rectifier, which was meant to change the AC input into a DC output that would be then sent a voltage regulator that would prevent the overload. The voltage regulator we chose was the TLV76760QWDRBRQ1 voltage regulator, which we chose for its size, which would fit into our PCB's design parameters. The output of the voltage regulator was then fed into the motor controller, micro-controller, and to the voltage-controlled switch. These components were chosen were the DRV8837, the ATTINY20-MMHR, and the TS5A4596 voltage switch. These components were all chosen for their dimensions primarily, which had at least one side that was within the 5 mm margin, which was our designated PCB width and because they could be used for our purposes. We were able to design a PCB board containing all of the components listed but were unable to have the design fabricated in the timeline before the due date of the paper. We were unable to test the resonator or receiver in this time either. Therefore, we have more yet to do, our demo needs to be finished and fabricated, but we are well on our way to completion nonetheless.**

***Index Terms*—LVADs, Micro-controller, PWM, Duty Cycle, PICkit4, DMM, USBASP, ATTiny24a, ATTiny20-CCUR, ATTiny20-MMHR, PIC10F220, jumper wires, UART, TPI, MPLAB x IDE, AVR**

I. Introduction

Ventricular assist devices have been used to help patients suffering with end-stage heart failure. Most ventricular assist devices assist the left ventricle, the heart chamber responsible for sending oxygenated blood to the organs and other tissues. For this reason, most ventricular assist devices are called left ventricular assist devices (LVADs). These devices have successfully helped patients wait for heart transplantation. They have even helped patients to end of life as long-term support devices.

Most LVADs are implanted through complicated open-heart surgery. Surgeons also need to make space within the torso cavity to place the device, which is essentially a pump. Recently, a new type of pump, known as intravascular LVADs have been developed and used clinically on patients needing partial support following a cardiac infarction. These devices are typically no more than 8mm in diameter and can be implanted through cardiac catheterization. Cardiac catheterization is a less invasive procedure compared to open-heart surgery that only requires vascular access. A small tube and guide wire is used to move throughout the blood vessel network in the patient. Cardiologists use X-ray machines to locate the catheter tip for maneuvering. While these intravascular LVADs are promising, their current use is limited to short-term support. Researchers are currently investigating how to modify the design of these devices for long-term use.

Our group, along with another senior design group, are designing a powering system that eliminates the need for electric power through wires. In order to enable the intravascular LVADs to be fully implantable for long-term use, the device should be powered with not physical wires traversing the body, which can lead to infection. The objectives of this senior design project were to fabricate an internal resonator that couples with an external resonator, fabricate a receiver, and design a motor controller board with physical constraints to fit in a cardiac catheter. Figure 1A shows the placement of the receiving system within the left ventricle and the block diagram of the components needed to power and control an intravascular pump.

The receiving system, shown in block diagram in Figure 1B, consists of an internal resonator tuned for 6.78 MHz, which is the lowest frequency band allowable without interference of medical devices and equipment in a hospital setting. The receiver couples to the internal resonator and produces an AC voltage and current that gets rectified by a rectifier. However, an impedance matching circuit ensures all the power received is transferred to the rectifier. The rectifier converts the AC signal to a DC signal with minimal ripple and sends it to a linear regulator. The stable DC signal is then distributed to a microcontroller and motor controller. The microcontroller controls the speed of the pump using a PWM signal and the motor controller uses an H-Bridge topology to provide power to the motor of the intravascular LVAD.

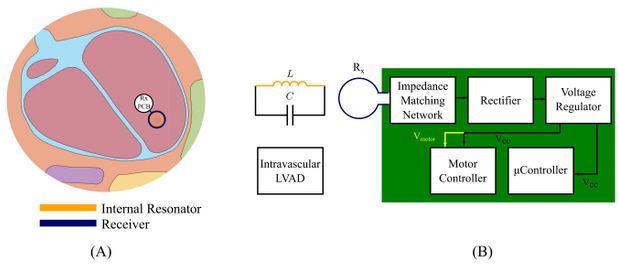


Fig. 1. (A) location of the receiving system in the left ventricle (not to scale) and (B) a block diagram of the receiving system and the flow of relevant signals

II. Internal Resonator

In order to make use of the power being transmitted to the pump, we needed to design a receiver and be able to rectify this power appropriately to power our motor controller. To construct our receiver we used copper wire and created multiple solenoid inductors, we created different solenoid inductors with a varying number of turns and with an inner radius of 3.5mm. In order to obtain our values for the impedance matching network, we used the solenoid inductors we had made from the copper wire. Using a Model 891 300kHz Bench LCR meter, we took our 20, 25, and 30 turn solenoid inductors with a length of 3 inches, and measured our inductance to be 380.8, 595.0, and 856.8 nH, respectively. The objective our group had was to construct as small a device as possible, with this in mind we planned to use the smallest components we had access to.

Fig. 2. Multi-turn internal resonators (left) and a single-turn receiver on the right.

In order to quantify the coupling, and efficiency, of the internal resonator and receiver interaction, a vector network analyzer

(VNA) needs to be used. To facilitate this test, a jig was fabricated to connect the devices, and any additional components (for resonant tuning and impedance matching), to an SMA connector for a VNA. The jig, shown in Fig. 3, contains an SMA connector and two ports to plug in the resonator and a tuning capacitor. Unfortunately, there was not enough time to carry out the experiment before the deadline of this paper.

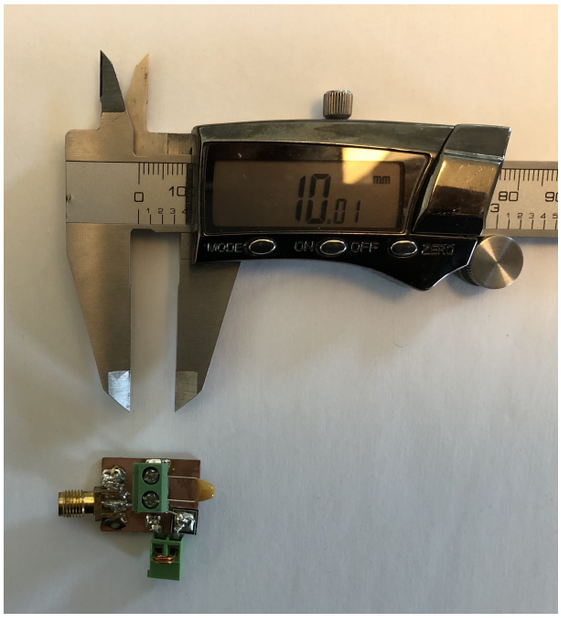


Fig. 3. Jig to tune the resonator or receiver and to measure scattering parameters for coupling.

III. Impedance Matching and Rectifier

Once we obtained our measurements. We used the SimSmith program to normalize the impedance for our circuit. Shown in Fig. 4, there are eight different network designs that can be used to impedance match an L-Network. Using SimSmith, the load was located on the left hand side of the Smith chart, which approximates a short circuit, limiting the network design to either C or D as seen in the figure below. This makes sense as the receiver is a single-turn wire that is effectively a wire that shorts two ports. Either design could be used and would provide the same results, but for this project, D was chosen.

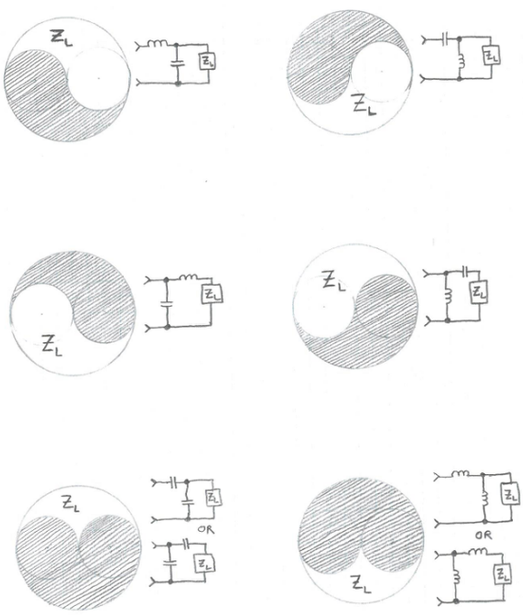


Fig. 4. Impedance Matching Network Designs

The formulas below represent the equations for capacitive and inductance reactance and the load impedance.

$$X_C = -j/\omega C \tag{1}$$

$$X_L = j\omega L \tag{2}$$

$$X = X_C + X_L \tag{3}$$

$$Z_L = R + jX \tag{4}$$

where ω is the frequency of operation (in rad/s), C is the capacitance, L is the inductance, R is the resistance, or real part, of the impedance, and X is the reactance of the impedance. The main function of the impedance matching circuit is to eliminate the imaginary part of the receiver impedance and match the real part to the input resistance of the rectifier. As a result, we needed to choose the rectifier to fix that parameter.

The first option we chose for our rectifier was the SBR05M100BLP, an SBR bridge super barrier rectifier. This rectifier met all of our specifications and was only 3mm by 3mm with a height of 0.6mm, but unfortunately due to the chip shortage and other circumstances we were unable to acquire these rectifiers. Our group settled on the CBRDFSH2-100 bridge rectifier. The footprint is contained within a 6 mm x 4.8 mm area. While large for our implantable motor controller, its electrical properties for its size were superb. Specifically, it had a peak repetitive reverse voltage of 100 V and a maximum current of 2 A. This fit the specifications for powering an intravascular LVAD. The DC output is then fed to a linear voltage regulator to ensure the microcontroller, motor controller, and motor do not overload the rectifier.

IV. Power Distribution

To ensure that a stable DC supply is applied to all components for power-up and for continuously running the motor, a linear regulator is needed. The TLV76760QWDRBRQ1 voltage regulator was chosen because of it's footprint, fitting in a 3.1 mm x 3.1 mm area. It also allows for up to 1 A current consumption and an adjustable output voltage ranging from 1.2 V to 12 V.

Power needs to be distributed to the motor, micro-controller, and motor controller. The motor for our project needs 6V and the other components can also operate at this voltage level. The maximum current draw from the motor is 800 mA, which leaves 200 mA for the other components to draw. The quiescent current draw for the ATTiny20 is 0.2 mA in active mode (0.025 mA in idle mode) and is 0.04-0.1 mA for the DRV8837 motor controller in operation (not including the motor current draw).

V. Microcontroller Selection and Programming

1) Pulse WIdth Modulation (PWM): PWM (or Pulse Width Modulation), is a series of continuous wave pulses that are created in order to have a steady AC voltage (considering it doesn't reach 0% or 100%). In our case, we use PWM to control speed in which the motor is rotating. Since PWM are square waves that fluctuate between 0V and the given supply voltage, the waves may be controlled by altering the Duty Cycle. The duty cycle of a PWM simply determines the fraction of a period in which the wave is active high (meaning the amount of time the wave pulse is on). The duty cycle can be controlled in our case through a C code established for the ATTINY MCUs. Depending on a given value equating to the percentage of Duty Cycle needed, a typical case being 50%, the percentage may be manipulated in order to correspond to the desired motor revolution speed.

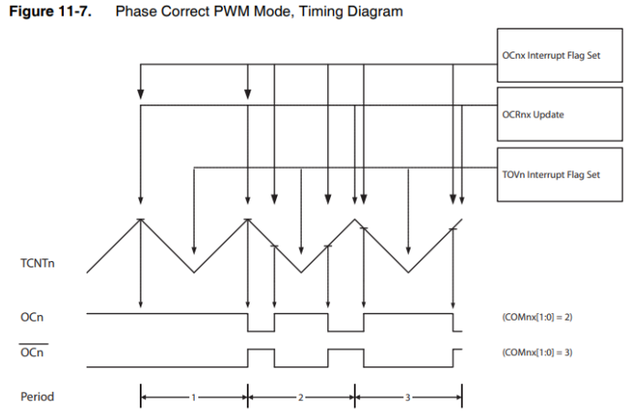


Fig. 5. Phase Correct PWM Timing Diagram[2]. This demonstrates the Phase correct PWM, which enables the PWM Duty Cycle to count up whenever clocked (meaning 0-255 is equivalent to 0 - 100%)

[1]

[1][2]Image taken from ATTINY24A datasheet - can be found in references

2) ATTINY Microcontrollers: Our project utilizes micro-controllers from the ATTINY family. These being, the ATTINY24A (in PDIP), ATTINY20-CCUR (in UFBGA), and ATTINY20-MMH (in VQFN). Each of these MCUs (micro-controllers) are 8-bit AVR MCUs which are all capable of producing PWM (Pulse Width Modulation) signals utilizing the 8-bit Timer/Counter which is a majority of what this project entails. In order to produce part of a successful project, we must entail a PWM signal to drive a motor using an H-bridge motor controller. This will be done so by having the PWM control the speed of the motor in order to act accordingly to any needed rotations.

Figure 1-1. Pinout of ATtiny24A/44A/84A

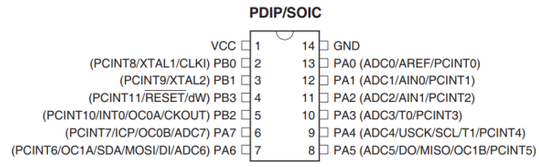


Fig. 6. ATTINY24a pin out diagram. For our application, pins 1, 4, 7, 8, 9, and 14 were utilized [2]

3) PIC10F220: The PIC10F220 is an 8-bit 384B Flash DIP MCU (micro-controller), with an 8-bit Timer Register (TMR0). This Timer Register would allow the creation of pulse waves to be generated and therefore allow for the motor to be controlled. Since this MCU was chosen specifically for its footprint size, as well as being designed for the PICkit4, this MCU was chosen to replace the ATTINY20-CCUR after an unsuccessful connection was brought forth. However due to the PIC10F220 not being able to control the Duty Cycle of the waves, this would not allow us to change the speed of which the motor controller would be able to rotate at. After witnessing this new issue, the alternative was to then establish a PCB and connection with the new MCU chosen which was the ATTINY20-MMH. At the time of this paper, the PCB was not able to be fabricated although the design was made.

4) µC Programming - PICkit 4: The PICkit 4 is a debugger/programmer with the capabilities to communicate with 8-bits, 16-bit ands 32-bits PIC MCUs (micro-controllers) and dsPICs and also SAM MCU devices. The mentioned MCUs are part of the same PIC families, though the PICkit 4 has the capabilities to communicate with various other MCU families, including our needed family which derives from the Microchip's ATTINY MCUs. In our desired implementation, we will be utilizing both the ATTINY24A micro-controller as well as the ATTINY20-CCUR micro-controller.

5) µC Programming - UART: The UART is known as a "Universal Asynchronous Receiver/Transmitter", and is referred to as Universal due to the fact that the data format and speed of transmission can be reconfigured, and Asynchronous since it does not have a clock. UART was first developed by Gordon Bell at Digital Equipment Corporation in the 1960's, and is able to communicate in various methods such as a full-duplex (devices use same line for data simultaneously), half-duplex (meaning 2 devices communicate at same time), or simplex (data communication occurs only one single way). In order to communicate successfully, the UART must have both the transmission and receiving communications configured in the same manner. The procedure for this will be explained in the following section.

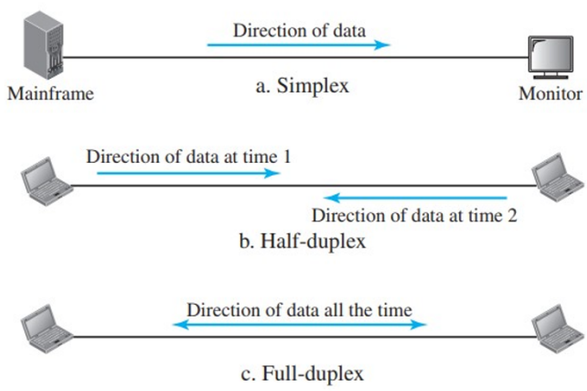


Fig. 7. UART communication [7].

We can take the following picture to illustrate just exactly how a UART protocol works.

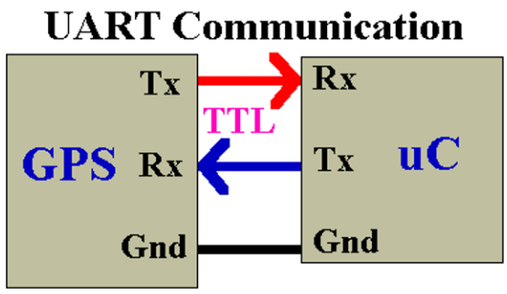


Fig. 8. UART communication example between two devices [6].

From this illustration, we can see that both devices transmit data (using the Tx lines), receive the data (Rx lines), and have a common reference to ground (GND). Whenever we want to transmit data from one device to another, we do so by having the Tx line of our data terminal (being the GPS) going into the Rx line of our data communication (being the micro-controller). The same is done from the µC into the GPS, since both lines must transmit and receive the data accordingly. The GND simply can connect into the same lines since these are only a reference ground. It is worth noting that there is no need for addressing in this protocol since it is transferring data over two wires directly. A more in-depth picture can be found below, which highlights just exactly the manner in which data is transmitted in the UART communication.

Since we determine when the protocol begins by coming out of Idle (logic 1), we must initialize the protocol by enabling a start bit (which is a logic 0) and following this is the data and remaining bits we can be found in the procedure below.

UART Procedure:

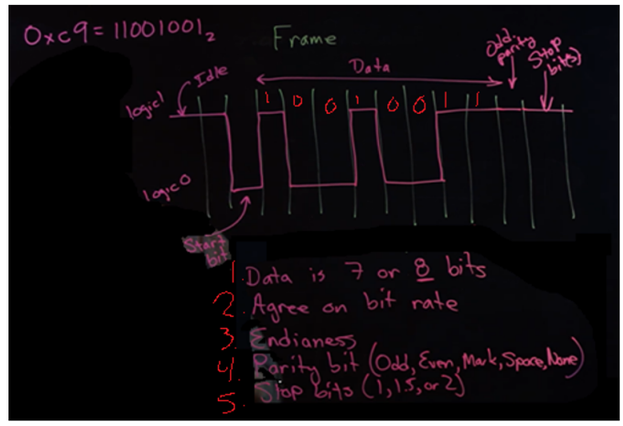


Fig. 9. This is illustrating the data frame in which data is transmitted from devices in a UART protocol[4]

1) We can begin by first determining data, which in this example is set for 8 bits (a byte is the most common for a UART protocol). Knowing we have our specified number of bits for data, we can the proceed to set the bit rate (or BAUD rate) for the protocol.
   -In this illustration, we see that we have a hex value of 0xc9 (equivalent to 11001001 in binary) for our data.
2) The BAUD rates can be set at 2400, 4800, 9600, 19,200 or 115,200. This must be done for both devices, otherwise there may not be a successful communication between devices.
3) Next is the endianness, which simply determines whether the data will be read as LSB (least significant bit first), or MSB (most significant bit first).

-For this example, we are using little endian since LSB is at front (11001001 becomes 10010011)

4) Parity bit will then come next (but this bit is most commonly set to none so this can be ignored).

[2][2] These images are taken from ATTINY24A data-sheet and referenced at the end

-To specify there are 5 parity bits which include: odd parity means you need odd number of 1's, even parity is even number of 1's, mark and space are always 1 and none signifies no value.

5) Finally, we end with a stop bit which then allows the logic to idle at logic 1 again determining that this is the end of transmitting data.

6) μC Programming - TPI: This section pertains to TPI, and specifics about what it is and how it relates to our project. TPI (known as Tiny Programming Interface), is used within the ATTINY family pertaining to ICs ATTINY4, 5, 9, 10, and 20. They use this protocol since it only requires 3 wires, aside from V_{dd} and Gnd. Those three wires being TPICLK, TPIDATA, and RESET. The reasoning behind this interface is so that rather than creating 2 different ports for input and output, the RESET can be used as the enable bit which would allow the TPIDATA to have both input and output data whereas a typical AVR would consist of using UART which has a MISO and MOSI for data transfers. Since our ATTINY20 uses TPI, we needed to become very familiar with this protocol, seeing as the ATTINY24A did not require a special protocol.

7) μC Programs: We first began the project by establishing a successful physical connection between the PICkit4 and the ATTINY24A using the MPLab x IDE software. In order to establish this connection, we read through extensively for both the ATTINY24A and PICkit4 datasheets in order to fully grasp concepts before trying to create communication. From reading the datasheets, we came to the realization that the needed PICkit module for programming the ATTINY24A would need to be done using the UART module, as seen in Fig. 9.

These physical connections were made using breadboard jumper wires, and had specific pins connected in order to establish the physical connection in the AVR PICkit setting. The following will exaplin the manner in which the connections were established and why they were chosen. We may first begin with the pins for the ATTINY24A.

ATTINY24A Pinout:

5V (Power Supply) VTG = Pin 1

GND (Power Supply) GND - Pin 14

MISO - Pin 8

MOSI - Pin 7

SCK (Clock used from program: default is 1 MHz) - Pin 9

RESET BAR - Pin 4

The following details pin selection designated for the PICkit:

PICKit 4 Pinout:

V (Power Supply) VTG = Pin 2

GND (Power Supply) GND - Pin 3

MISO - Pin 4

MOSI - Pin 5

SCK (Clock used from program: default is 1 MHz) - Pin 6

RESET BAR - Pin 7

The reason why these pins are detailed is to demonstrate how each pin was connected from one device to the other (i.e PICkit4 and ATTINY24A). Each connection made can be seen within the following figures, Fig. 10 and Fig. 11.

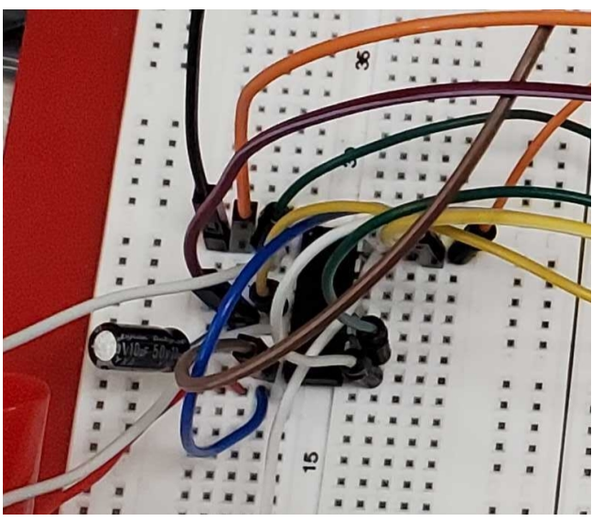


Fig. 10. Connections of the ATTiny24A

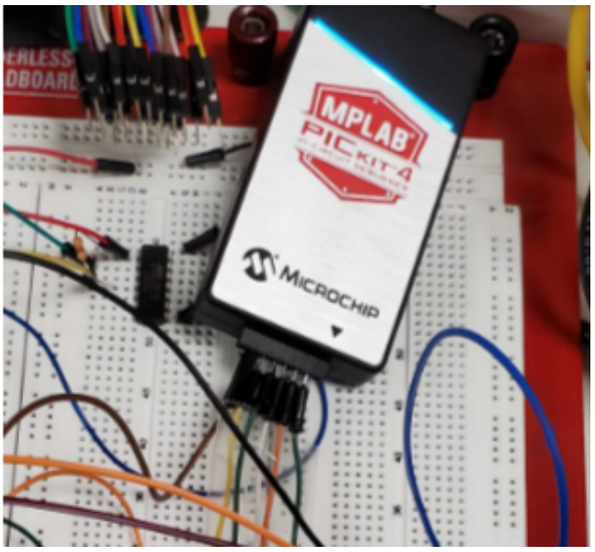


Fig. 11. Connections of the PICkit4

Once the connections were successfully established, which were able to confirm connection by reading the memory off of the micro-controller, we then proceeded to write a simple C code in order to have a blinking LED which could then demonstrate that the supplied voltage and code worked as intended. In order to commence the actual programming, we ensured that the program used was MPLAB x IDE with the latest compiler XC8 (v2.32) in order to avoid any software issues.

Table 1: Pinouts for Debug Interfaces

	MPLAB® PICkit™ 4 ICD	DEBUG									
Pin #	Pin Name	ICSP™ (MCHP)	MIPS EJTAG	Cortex® SWD	AVR® JTAG	AVR ISP (& Dw)	UPDI	PDI	aW	dW (IRE)	TPI
1	TVPP	$\overline{\text{MCLR}}$	$\overline{\text{MCLR}}$	$\overline{\text{MCLR}}$							
2	TVDD	VDD	VIO_REF	VTG	VTG	VTG	VTG	VTG	VTG	VTG	VTG
3	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND
4	PGD	DAT	TDO	SWO	TDO	MISO	DAT	DAT	DATA		DAT
5	PGC	CLK	TCK	SWCLK	TCK	SCK					CLK
6	TAUX	AUX			$\overline{\text{RESET}}$	$\overline{\text{RESET}}$		CLK		dW	RST
7	TTDI		TDI		TDI	MOSI					
8	TTMS		TMS	SWDIO	TMS						
Module		BB	BB	BB	SPI	UART	USART		UART	BB	

Fig. 12. An image of PICkit supported modules, with UART highlighted in order to demonstrate which module was used for our implementation.[5]

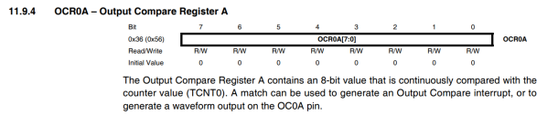
The following will explain in great detail the manner in which the PWM C code was made and explaining each line of code.

```
/*
 * File:   attest.c
 * Author: User PC
 *
 * Created on December 2, 2021, 2:32 PM
 */

#include <avr/io.h>

  int main (void){

      DDRB |= (1 << DDB2); //PB2 output
      OCR0A = 128; // 50% duty cycle (128/256) at 8 bit
      CLKPR = (1 << CLKPCE);
      CLKPR = (1 << CLKPS2); //clock divide by 2

      TCCR0A |= (1 << COM0A1); //set Phase correct PWM
      TCCR0A |= (1 << WGM00) ; //set Phase correct PWM mode

      TCCR0B |= (1 << CS01); //Pre-scaler set to 1

    while (1) {

        }//end while

}//end main
```

Fig. 13. C Code used to generate PWM for motor controller. Demonstrates all registers and bits used in order to control frequency and Duty Cycle of PWM.[2]

DDRB |= (1 << DDB2); //PB2 is set as the output but can be set into any B port

OCR0A = 128; //OCR0A – Output Compare Register A
// This uses 8 bits which we in turn will use to create the Duty Cycle
// Currently there is a 50% duty cycle (128/256) at 8 bit



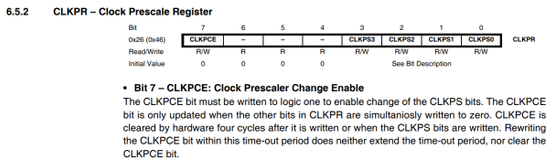
11.9.4 OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x36 (0x56)	OCR0A[7:0]								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0A pin.

Fig. 14. Compare Register bits used in order to control the Duty Cycle. Since it is an 8 bit register, we are able to access a value of 0xcFF or 255.[2]

```
CLKPR = (1 << CLKPCE); // Bit 7 – CLKPCE: Clock Prescaler Change Enable
//CLKPR – Clock Prescale Register
//This is used to scale the clock which is set at 1MHz by default
```



6.5.2 CLKPR – Clock Prescale Register

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	CLKPR
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	See Bit Description				

• **Bit 7 – CLKPCE: Clock Prescaler Change Enable**

The CLKPCE bit must be written to logic one to enable change of the CLKPS bits. The CLKPCE bit is only updated when the other bits in CLKPR are simultaniosly written to zero. CLKPCE is cleared by hardware four cycles after it is written or when the CLKPS bits are written. Rewriting the CLKPCE bit within this time-out period does neither extend the time-out period, nor clear the CLKPCE bit.

Fig. 15. Clock Prescale Register, which controls the Enable bit allowing the clock to be divided (default to 1MHz) in order to manipulate PWM frequency.[2]

```
CLKPR = (1 << CLKPS0); //clock divide by 2
//CLKPR – Clock Prescale Register
//Enabling these bits determines the clock division factor
```

Table 6-11. Clock Prescaler Select

CLKPS3	CLKPS2	CLKPS1	CLKPS0	Clock Division Factor
0	0	0	0	1
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8
0	1	0	0	16
0	1	0	1	32
0	1	1	0	64
0	1	1	1	128
1	0	0	0	256
1	0	0	1	Reserved
1	0	1	0	Reserved
1	0	1	1	Reserved
1	1	0	0	Reserved
1	1	0	1	Reserved
1	1	1	0	Reserved
1	1	1	1	Reserved

Fig. 16. Table demonstrating the clock division factors used to scale the default clock which determines the frequency of PWM.[2]

```
TCCR0A |= (1 << COM0A1); //set Phase correct PWM
//TCCR0A – Timer/Counter Control Register A
//Setting COM0A1 to 1, in Phase Correct Mode, has up-counting
//This means that going from 0-255 for OC0A will be 0 to 100% Duty Cycle respectively
```

Table 11-4 shows the COM0A[1:0] bit functionality when the WGM0[2:0] bits are set to phase correct PWM mode.

Table 11-4. Compare Output Mode, Phase Correct PWM Mode[1]

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match when up-counting. Set OC0A on Compare Match when down-counting.
1	1	Set OC0A on Compare Match when up-counting. Clear OC0A on Compare Match when down-counting.

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See "Phase Correct PWM Mode" on page 82 for more details.

Fig. 17. Compare Output mode, which determines how the Phase Correct mode PWM will be counted. In our implementation, we are using an up count which means Duty Cycle will go from 0 to 100%.[2]

```
TCCR0A |= (1 << WGM00); //set Phase correct PWM mode
//TCCR0A – Timer/Counter Control Register A
//Setting WGM00 to 1 gives us the Mode of Operation to PWM, Phase Correct
//With the Top being 0xFf (255), and OCRA is updated at the TOP (resets)
```

Table 11-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[1]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Note: 1. MAX = 0xFF
BOTTOM = 0x00

Fig. 18. Table demonstrating how the Timer/Counter mode of operation will be selected. FOr our implementation we set WGM00 to 1, meaning we are using Phase Correct PWM with TOP being oxFF to have it be the limit.[2]

```
TCCR0B |= (1 << CS01); //Pre-scaler set to 1
//TCCR0B – Timer/Counter Control Register B
//Setting CS00 to 1 enables a clock prescaling of 1 (called no prescaling)
```

Table 11-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)

Table 11-9. Clock Select Bit Description (Continued)

CS02	CS01	CS00	Description
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

Fig. 19. Table demonstrating the Clock Prescaling for the PWM.This may be utilized in order to change PWM frequency.[2]

3

Upon debugging and creating our PWM code, illustrated in Fig. 20, we then tested our code by viewing the PWM output using an oscilloscope as well as an LED.

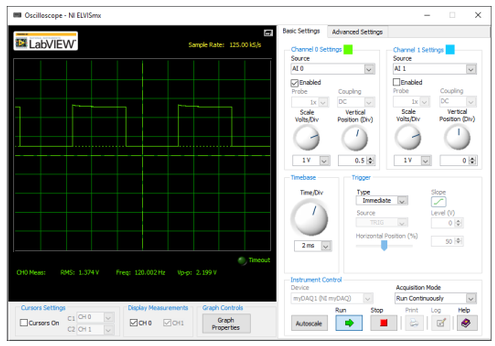


Fig. 20. This is an image of the PWM generated by our code.

Due to the unforeseen issues when soldering, the proper implementation of the ATTINY20-CCUR was not successful at the time of writing this paper.

VI. Motor Controller

The motor controller supplies the power needed to run the motor in the intravascular LVAD. The specific motor used in the intravascular LVAD is a brushless DC motor with a maximum operating voltage of 6V. The voltage level applied to the motor changes the rotational speed of the motor shaft. The polarity of the DC supply changes the rotational direction (clock-wise or counter clock-wise). The DRV8837 was chosen for various reasons. It utilizing an H-bridge configuration to operate the windings of the motor. The allowable voltage ranges from 0-11 V and allows a current of up to 1.8 A to run through the device. It also has pins for enabling CW or CCW rotation. The motor controller has a footprint contained within a 2 mm x 2mm area.

[3]See Reference [2]

As detailed earlier in this paper, our group was able to reflow solder a DRV8837 device on to a customized DIP adapter that our group created and show that when enabled by the ATtiny24, could drive a motor.

The DRV8837 necessitates a PWM signal into the input to control the rotational speed of the motor shaft, and subsequently, the impeller speed of the intravascular LVAD. This will increase or decrease blood flow to the patient. The PWM signal activates the MOSFETs in the H-Bridge and allows current to flow in the motor windings. The duty cycle of the PWM effectively changes the DC voltage applied to the motor. A low duty cycle PWM (about 10%) will effectively lead to an average DC voltage that is much lower than 6 V. A higher duty cycle (like 90 %) will lead to an average DC voltage closer to 6 V. As a result, the amount of current sent to the motor will also change accordingly. Because the motor windings are inductive elements, they do not respond quickly enough to the fast changes in the PWM signal.

A. *Schematic and Printed Circuit Board Design*

The first thing that we needed to consider when designing our board was which components' footprints we would need in order to properly build it. In our first design, we intended to use the ATTINY20-CCUR. We decided that we would use the program KiCad in order to design our circuit board. The reason for this was due to members of our group being familiar with the software more than others and the fact that this chosen software was one of the most accessible options given our budget.

Before we started creating a final design for the PCB that would contain every component it would need upon full implementation, we created a PCB board specifically for the ATTINY20-CCUR, for testing purposes. Using the datasheet, we endeavored to create a functioning board that we could solder pins to and connect to a breadboard. This was done to ensure that the part would work properly before committing all of our resources to the part. Our design was as shown in Fig. 21. As one might observe, it is fairly simple, as the design only included the ATTINY20-CCUR.

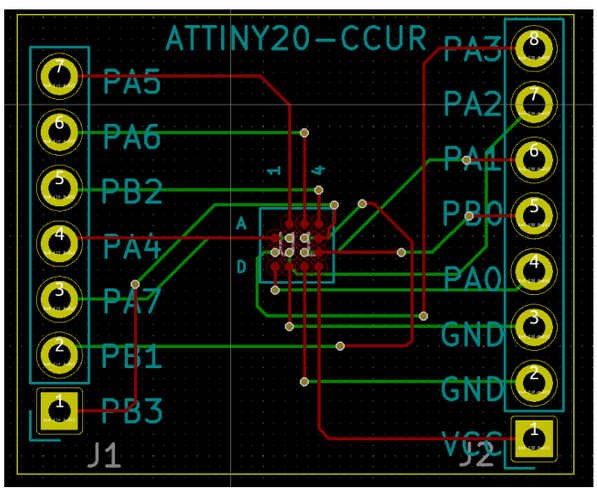


Fig. 21. This is an image of the model for the PCB board for the ATTINY20-CCUR micro-controller.

To test the micro-controller further, we also designed a PCB board to contain the footprint for our motor driver, which is shown in Fig. 22. Both would be placed on a breadboard and tested with our motors in order to make sure that they made them run.

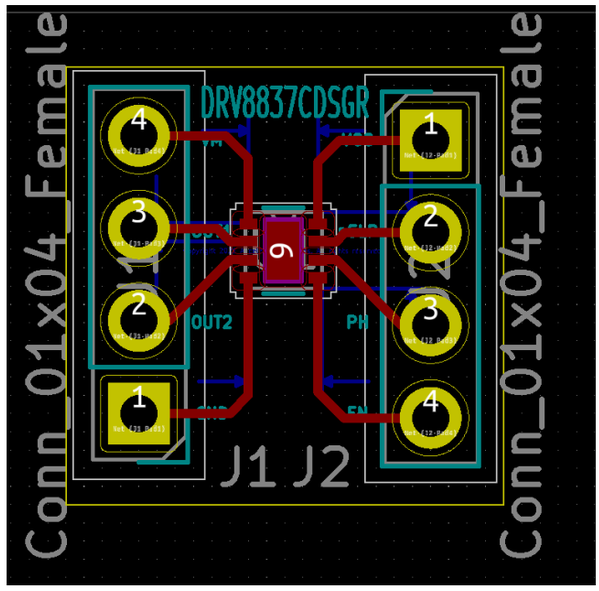


Fig. 22. This is an image of the model for the PCB board for the DRV8827CDSGR motor controller.

We didn't end up using these boards in our final design, and instead decided on creating an entirely new PCB that would include a different ATTINY model, that being the ATTINY20-MMHR.

The first part of the process, however, was to determine all of the component parts we would use for the final design. We, of course, had already decided on using the DRV8837CDSGR motor driver to control out motor, but that wouldn't be all that we ended up needing. We needed a rectifier to receive power from the transmitter, a voltage regulator, and a switch.

For each of these components, one of the most important factors we considered when deciding on which to use was their dimensions. We needed to ensure that the full PCB would fit within the human body and had set design parameters of the board as a desired 5 mm width, maximum 3 inch length, and 6 mm height.

With these parameters in mind, we chose the CBRDFSH2-100, which is a surface mounted, 20 A silicon, Schottky bridge rectifier. We chose this one in particular because of its dimensions. The CBRDFSH2 rectifier has a width of 6.05 minimum and 6.25 mm maximum, and a length of 4.65 mm minimum and 4.85 mm maximum. With these dimensions, it would be possible fit this chip within the 5 mm width parameter,

Next, we were looking for a voltage regulator that would fit within the design parameters. We decided on the TLV767-Q1 1-A, 16-V linear voltage regulator, again, this was chosen in order to fit within the 5 mm width parameter, since this voltage regulator is 3.4 mm wide and 2.4 mm long.

For the switch, again, we looked for a switch that would be small enough to fit on our board, so we decided on the TS5A4596 8-Ω SPST analog switch. The dimensions for this component are 2.6 to 3.0 mm width and 2.75 and 3.05 length, which are both well within our design parameters.

With out parts decided, it is important to explain the reasons why we needed them to begin with. Starting with the rectifier.

For the rectifier, it was to be connected to our wireless power receiver, The purpose of this has to do with what a rectifier does. Rectifiers are used in almost all chargers to convert AC inputs to DC for use in various electronic devices. We need the rectifier because our wireless power receiver will be receiving AC and we'll be needing DC in order to run our circuit.

As for the voltage regulator, we need that component for system stability. Voltage regulators work to create fixed output voltages of a designated magnitude which will remain constant in spite of changes to input voltage and/or load conditions. The reason why this is important is that the power supply being fed by our rectifier may produce raw current that could cause our components to fail. This event would be catastrophic if it occurred, causing the pump to cease functioning entirely.

Lastly, the reason for needing the switch was for the sake of programming the micro-controller. The micro-controller cannot be programmed if it is attached to the remainder of the circuit, therefore we needed a method to manually separate it from the motor driver. A switch was the perfect solution to the problem, as turning off the motor driver all that was necessary. With all of

that out of the way, we could begin creating our PCB's schematic, which was as shown in Fig. 23.

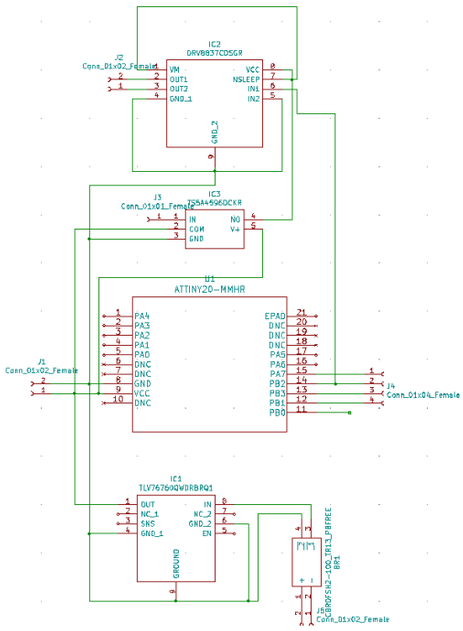


Fig. 23. This is an image of the schematic of the circuit that would be in our finished PCB.

Because size was a limiter, we wanted to reduce the amount of connectors down to what was absolutely necessary for programming and function of the board. Therefore, we only attached connectors to the inputs of the rectifier, the programming ports of the ATTINY20-MMHR, the on and off input of the voltage switch, and the two outputs of the motor controller. These were the only parts that needed external connections. Everything else was connected to other components. All the grounds were connected to each other and all voltage source inputs were also connected to each other.

With the schematic done, the more complicated part was the creation of the PCB itself, since it had to be designed with our parameters in mind with what would be the real-world connections that would appear on the fabricated board. Ultimately, due to the tracks of the connection needing to pass and overlap each other to connect the components, the smallest we managed to get our width was 6.5 mm, which is still a reasonable size, while our length is much smaller than the design limit at 43.25 mm. Our goal was to minimize the dimensions of the PCB as much as possible. Our end result can be seen in Fig. 24.

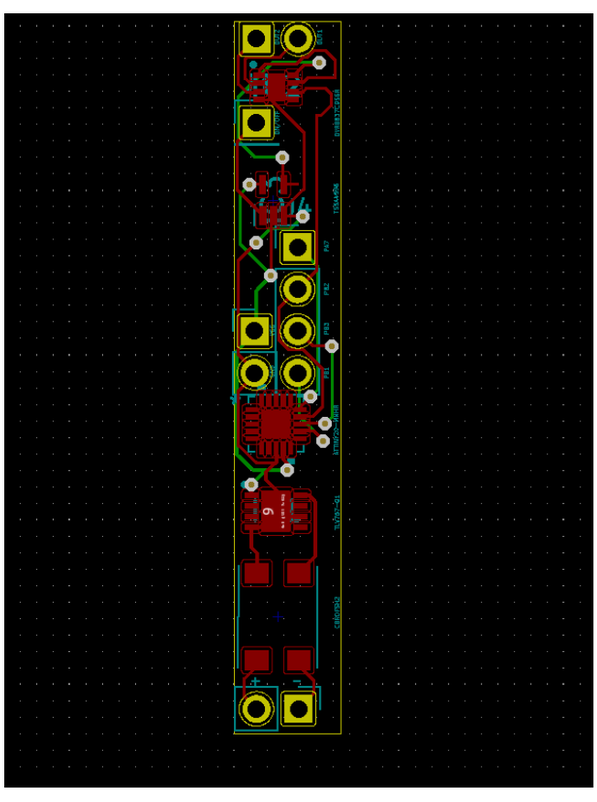


Fig. 24. This is an image of the designed PCB.

B. Soldering the Micro-Controller and Motor-Controller

Since our first micro-controller was a DIP type, there was no need to solder any components since we simply placed it into a breadboard for debugging and creating a successful PWM code in order to use for the same IC family (ATTINY). The first successful solder that occurred was for the DRV8837, which is the motor-controller. This was done so by using a hot plate, solder paste, a stencil, and our personally designed PCB made specifically for the DRV8837. Confirmation of the successful soldering was done so by checking traces with the DMM (digital multimeter), as well as connecting it to our breadboard and using the already programmed ATTINY24A loaded with the PWM code.

We were able to refer quite rapidly to the DRV8837 datasheet, which allowed us to connect the needed pins from the ATTINY24A to the DRV8837, as well as connecting proper voltages and motor wires to the DRV8837 (motor-controller) PCB. A successful implementation of said connections can be viewed in figure Once we got the motor spinning with our supplied PWM, we debugged a bit more by having our motor run at different duty cycle's and frequency. This was all performed through the PICkit4 and the MPlab x IDE software, simply by changing values within our C code in order to get the specifications we desired. Fig. 25 shows our finished demo circuit for the ATTiny24A.

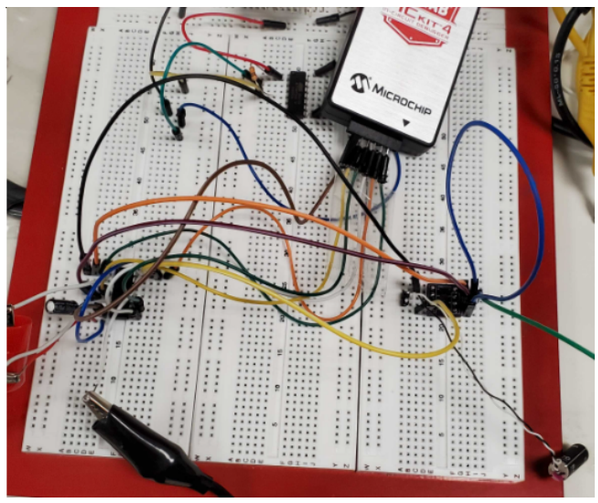


Fig. 25. This is an image of the working circuit implementation with the ATTINY24A and DRV8837.

The following component that was then soldered was the ATTINY20-CCUR. This consisted of a BGA (ball grid array) soldering, which was approached in the same manner as the DRV8837. This meant we attempted to solder the micro-controller with a hot plate, but rather than solder paste we used flux since the BGA consists of small solder balls that can be melted into the PCB if done so properly. Fig. 26 shows our ATTiny20-CCUR being soldered to its PCB.

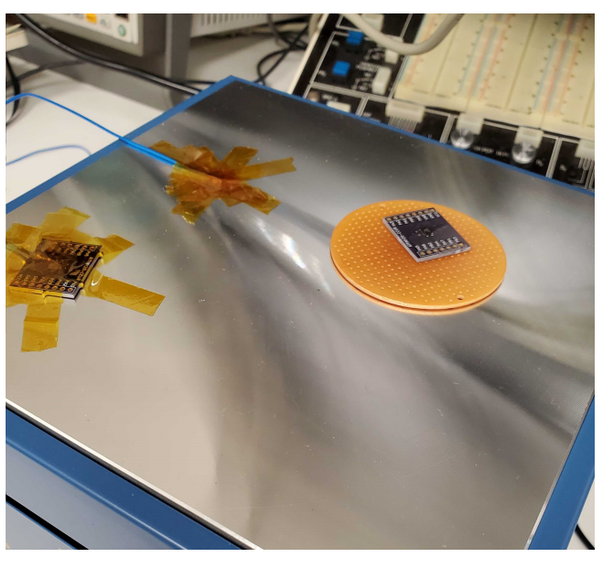


Fig. 26. ATTiny20-CCUR being soldered.

However, our first attempt lead to an undesired short, which was between the GND pin and Pin B3. After a second attempt, we were able to create a new micro-controller PCB circuit with no

shorts. Unfortunately after creating the connections, we came to the conclusion that the soldering technique was not successful as no proper PICkit4 communication was established by the time this paper was written. This is further elaborated within the subsection pertaining to ”Programming the micro-controllers”.

VII. Conclusion

The purpose of our project was to create a left ventricular assist device (LVAD) that would run on power that would transmitted wirelessly to the device without the need of wires running into the human body, which can cause health issues such as infection with long-time use.

The objectives for the project were to design and fabricate an internal resonator that couples with an external resonator, to design and fabricate a receiver, and to design and fabricate a motor controller board with physical constraints that would allow it to fit within a cardiac catheter.

To create our receiver, we used an internal resonator that was tuned to 6.78 MHz, due to it being the lowest frequency of band allowable without interference from medical devices and equipment. The receiver couples with the internal resonator to produce an AC voltage and current that would be transmitted in DC using a rectifier, which then sends power to a linear voltage regulator, which then is used to power the micro-controller and motor controller. The micro-controller was programmed to produce a PWM signal that is sent to the input of the motor controller, which then controls the motor of the pump.

The components we used for the PCB, that being everything other than the motor itself and the wireless power receiver, were the ATTiny20-MMHR, the CBRDFSH2-100 bridge rectifier, the TLV76760QWDRBRQ1 voltage regulator, the TS5A4596 voltage-controlled switch, and the DRV8837CDSGR motor driver. These were all chosen for their small size that enabled them to fit within a PCB that was then supposed to be placed within a cardiac catheter.

We completed the PCB schematic and design but were unable to receive the fabricated board in time for the submission deadline of this paper. Even so, once we receive the PCB, we will be able to construct a proper demo.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] “ATTINY24A/44A/84A - Microchip Technology.” [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/ATtiny24A-44A-84A-DataSheet-DS40002269A.pdf. [Accessed: 2021].

[3] W2aew. 276: Smith Chart: Design an L-Network - Impedance Match Circuit. (Jan. 27, 2018). Accessed: Apr. 3, 2022. Available: [276: Smith Chart: Design an L-Network - Impedance Matching Circuit - YouTube]

[4] ”Going Old School with Asynchronous Serial Protocols”, 01-Mar-2021. [Online]. Available: https://www.youtube.com/watch?v=33dmXAAaD28&ab_channel=Intermation. [Accessed: 2022].

[5] “Developer help” MPLAB® PICkit™ 4 Debugger Pinouts for Interfaces - Developer Help. [Online]. Available: https://microchipdeveloper.com/pickit4:interface-pinouts. [Accessed: 2021].

[6] M. Jamiu, “What is simplex, half-duplex and full-duplex mode of communication?,” Tooabstractive. [Online]. Available: https://www.tooabstractive.com/networking/what-is-simplex-half-duplex-and-full-duplex/. [Accessed: 19-Apr-2022].

[7] V. Kartha, “Using UART on raspberry pi - python - pyserial,” electroSome, 03-Jan-2021. [Online]. Available: https://electrosome.com/uart-raspberry-pi-python/. [Accessed: 19-Apr-2022].