Stepper Motor Controller

ECE 395-Advanced Digital Projects Lab

Michael Gamota

Table of Contents

Introduction

In the Advanced Digital Systems Labratory this semester, I completed the design, assembly, and programming of a stepper motor controller for the EV Concept car. It will be used to control a motor which will turn the steering column. This board allows input from either the CAN bus which runs throughout the vehicle or over a USB COM port (useful for unit testing) which is then converted in software to a PWM signal which turns the motor within 1.8 degrees of the desired angle. The 4 stages of this project were Research/Planning, Design, Assembly/Test, Software Development. The board itself can be broken down into 5 main sections: Power, H-Bridge(Coil A/B), CAN, USB, and MCU.

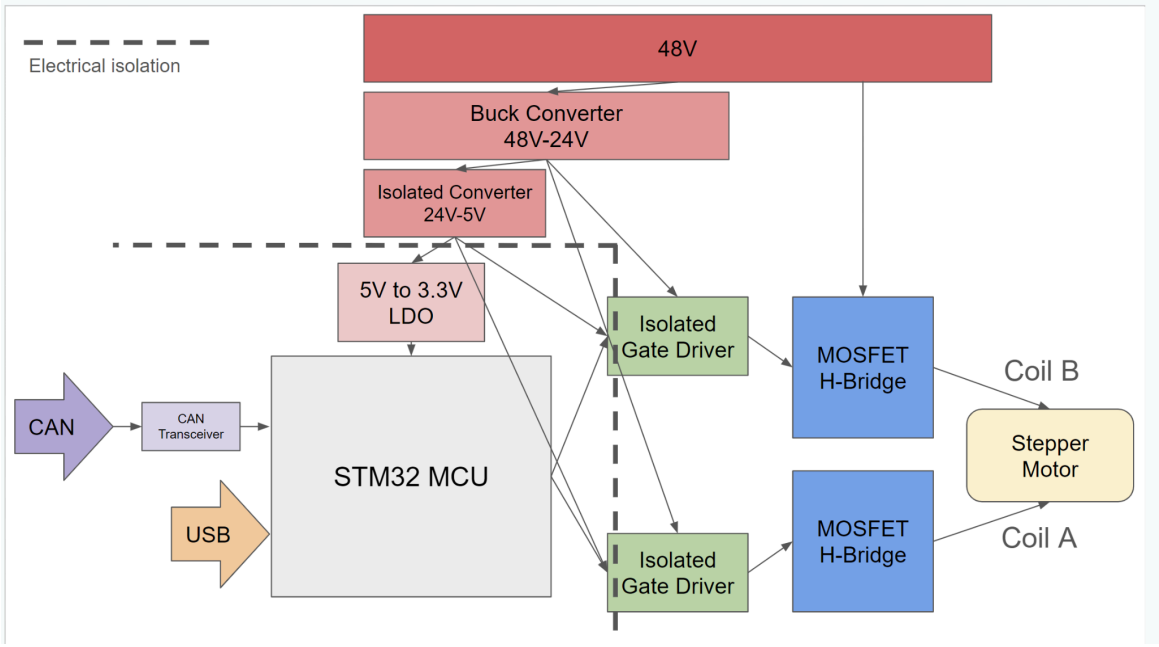


Figure 1. Block diagram

Research/Planning

In order to design this board, the starting point was researching stepper motor controllers. The general theory of stepper motor control is based around passing current through 2 different coils in different “steps”, meaning passing current “clockwise” through coil A, then passing current “clockwise” through coil B, etc. It is the specific sequence that controls the direction of the stepper motor shaft. This is most easily achieved through an H-Bridge, a very common control circuit which is commonly used for DC motor control. Ultimately, the solution was to use isolated gate drivers to reduce noise in the logic and communication circuits.

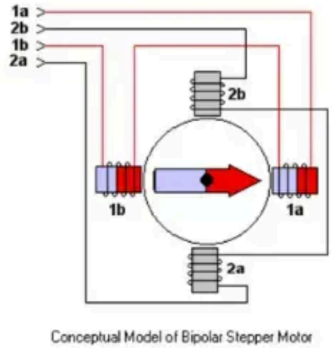


Figure 2. Stepper Motor Control Concept

The control method decided on was the “chopper” method, which involves using a PWM signal to limit the current flowing through the stepper, allowing a higher voltage to be used to drive the motor. PWM is commonly used for voltage modulation but the effect of limiting the “on” time of voltage can also have effects of current modulation when done at high enough frequencies in circuits with reactance or inductance.

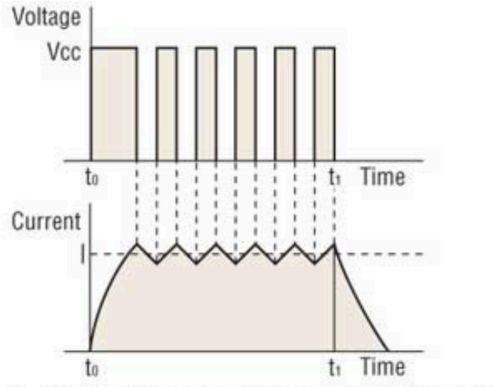


Figure 3. “Chopper” Drive

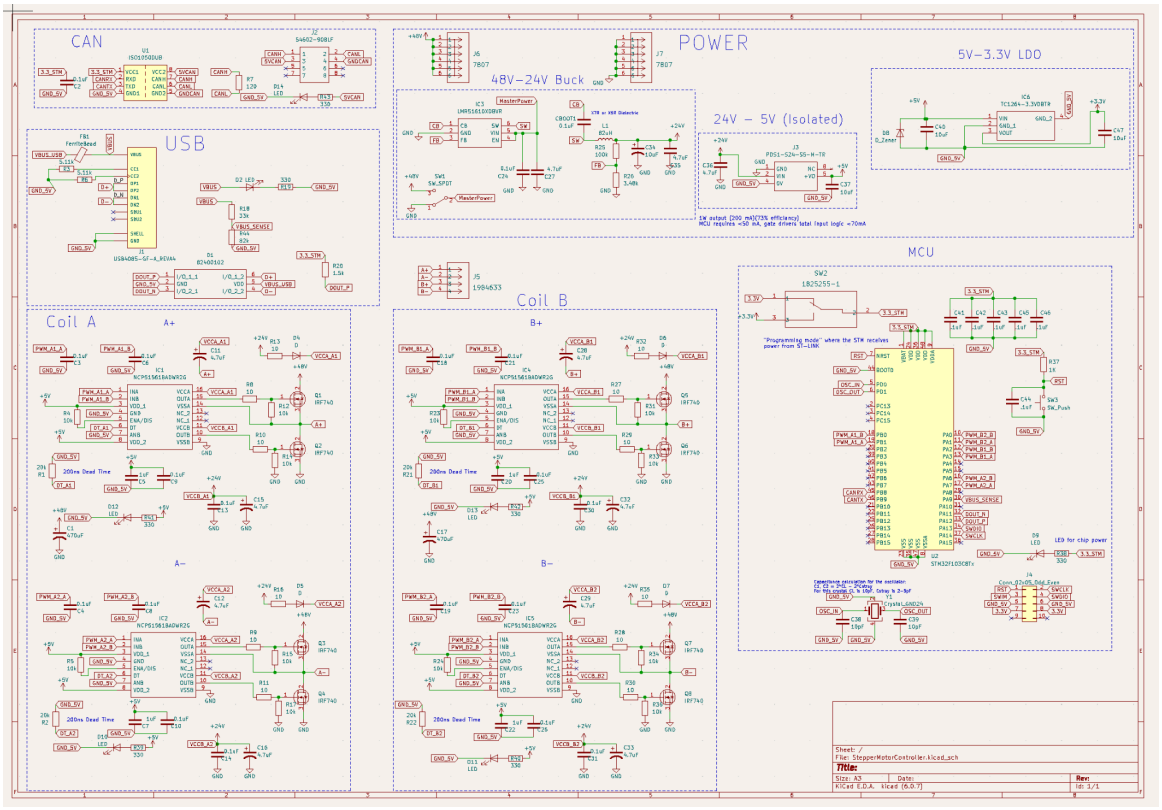


Figure 4. Overall Schematic

Design

Power

The power section of the board is designed to take in 48V which is then converted to 24V, which is converted to 5V which is converted to 3.3V. Each voltage level is used for a different function, 48V is the voltage being switched across the MOSFETs in the H-Bridge, the 24V is used to drive(gate voltage) the MOSFETs in the H-Bridge, the 5V is used to power the logic of the isolated MOSFET drivers, and the 3.3V is used to power the STM32F1 MCU. I have 2 separate ground planes, one for 24V and 48V and another for 5V and 3.3V. This is possible because the conversion from 24V to 5V is electrically isolated. The other important consideration is overall power, which should be about 200 W, the motor is rated for 3A, and the rest of the board should not require more than 500 mA between powering the gate drivers, MCU, etc.

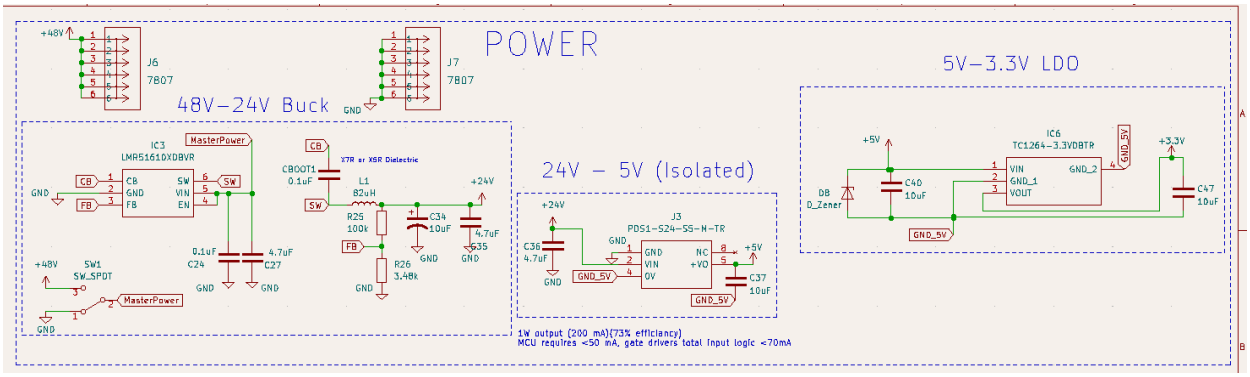


Figure 5. Power

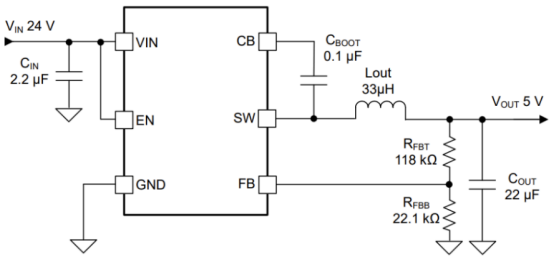
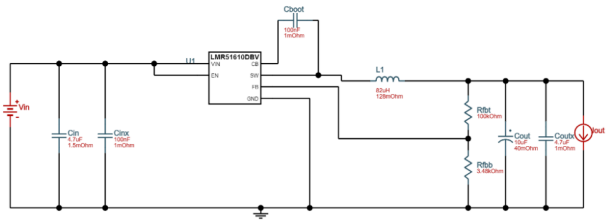


Figure 8-1. Application Circuit

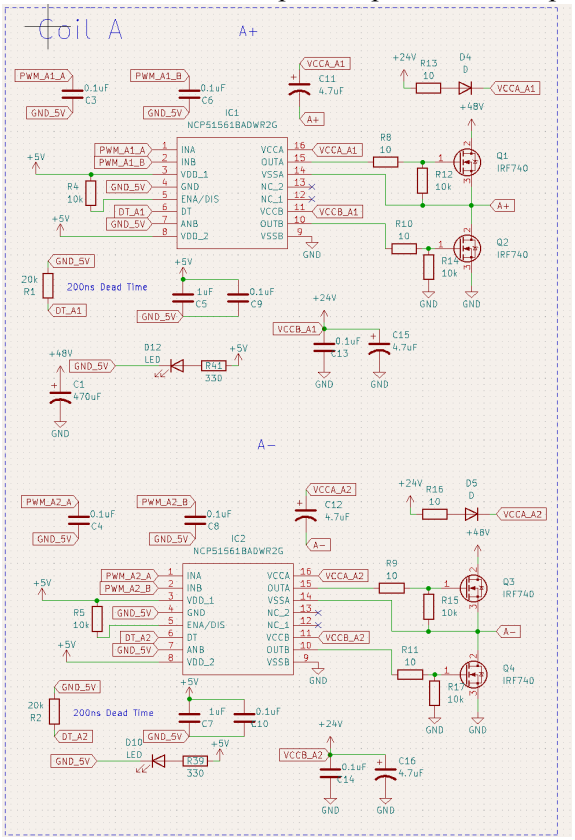


TI Generated Layout for 48V-24V

Figure 6. 48V-24V (Buck Converter) Schematic

H-Bridge

The H-Bridge section of the board is the analog circuit which actually moves the stepper motor. This section includes 2 H-Bridges, one for each of the two coils that are inside of the stepper motor. To turn the motor, the H-Bridges must switch the direction of current through the coils in a certain order, which is achieved by driving the MOSFETs in a certain sequence specified by the datasheet. To drive the MOSFETs, I used isolated gate drivers which ensure that any electrical problems that happen on the high voltage side will not affect components on the low voltage (logic) side, like the microcontroller. The gate drivers have shoothrough protection which prevents the H-Bridge from shorting 48V to ground. The gate driver is used to drive N-channel MOSFETs so MOSFETs which could switch at least 60V and withstand Vgs greater than 30V were required. The gate driver receives a logic level PWM input from the MCU which is then turned into a 24V peak to peak PWM output by the gate driver.



TYPE OF CONNECTION (EXTERN)		MOTOR	
PIN NO	BIPOLAR	LEADS	WINDING
1	A –	BLK	A
2	A\ –	GRN	A\
3	B –	RED	B
4	B\ –	BLU	B\

FULL STEP 2 PHASE-Ex. ,
WHEN FACING MOUNTING END (X)

STEP	A	B	A\	B\	
1	+	+	-	-	↓ CW
2	-	+	+	-	
3	-	-	+	+	
4	+	-	-	+	↑ CCW

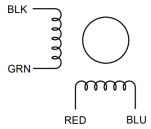


Figure 7(above). H-Bridge Schematic for coils A+ and A- (A and A\ in schematic diagram)

Figure 8(above right). Datasheet for stepper motor

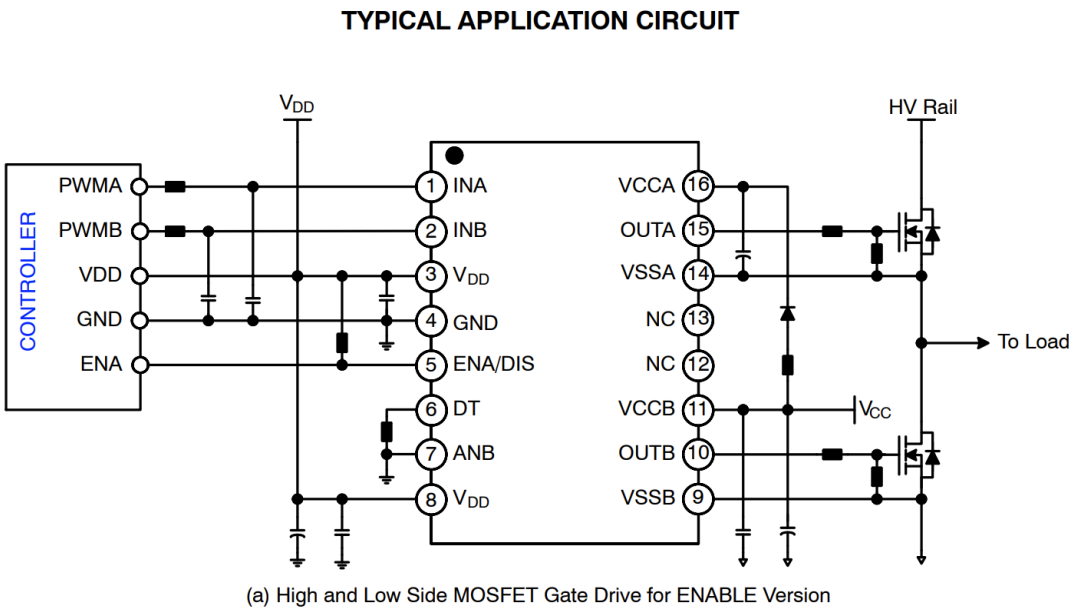


Figure 9. Typical Application Circuit for Isolated Gate Driver Circuit

USB

The USB section of the board is designed with the intention of connecting the module to a computer. I used the USB COM port to send and receive messages to verify that the low level control algorithms I wrote in C worked when the control values were not hard coded and instead read in from some input. Some design considerations when using the built in USB pins on the STM32F1 were the placement of the chips to limit noise, a ferrite bead to limit noise, making the D+ and D- USB traces a differential pair, and a TVS diode for protecting from electrostatic discharge.

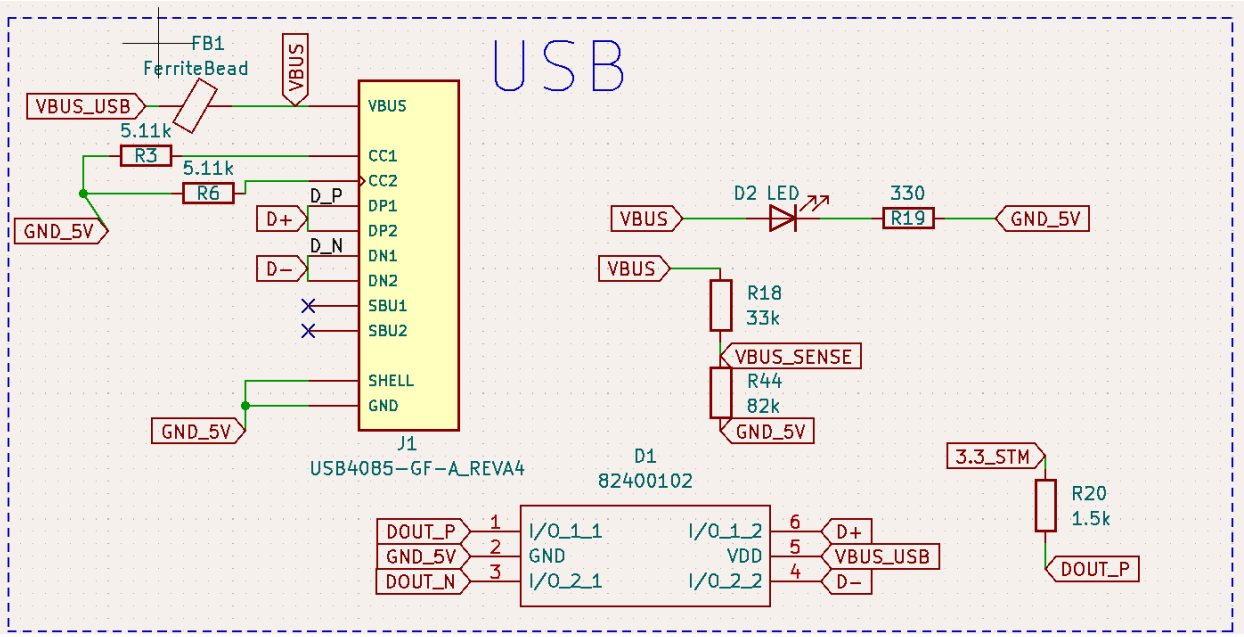


Figure 10. USB Schematic

CAN

The CAN section of the board is based around the ISODUB1050 which is a CAN transceiver which is used on all EV Concept boards. This converts the serial data coming from the MCU into a differential signal and serializes the differential CAN high and CAN low signals coming from the bus and going into the MCU. There is also a terminating resistor of 120 ohms which is a standard for CAN. The STM32F1 supports CAN and there is very little hardware design overhead.

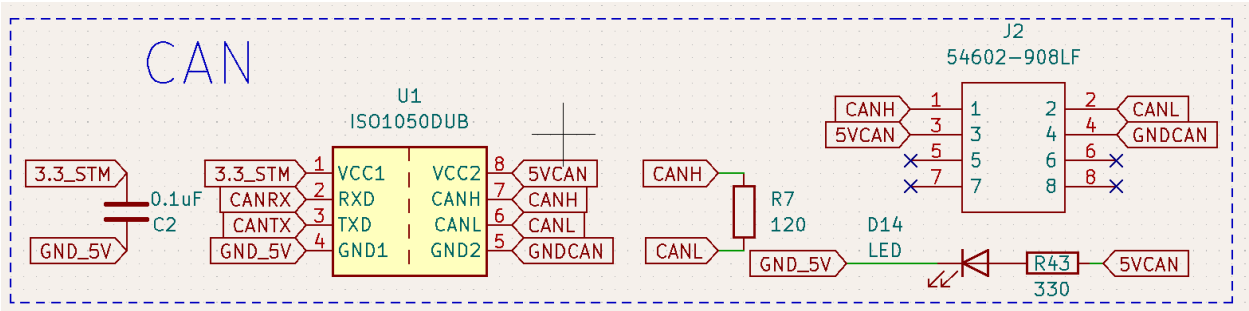


Figure 11. MCU Schematic

MCU

The MCU section of the board uses an STM32F1 microcontroller and I included a reset switch and decoupling capacitors to reduce noise. This part is pretty standard but one important design consideration was the limitations of the communication protocol pins, for example the CAN pins of the MCU are in a specific spot and is only 1 option for where to place the USB pins so the physical layout of these “sections” was dictated by the STM32F1 pin configuration limits. This project also required an external oscillator to use as a clock since the clock speed required for USB is higher than the internal clock of the STM32F1. Another feature that I added was a “Program/Run” switch which allows for the MCU to be programmed using power from the STLink Debugger instead of powering up the entire board.

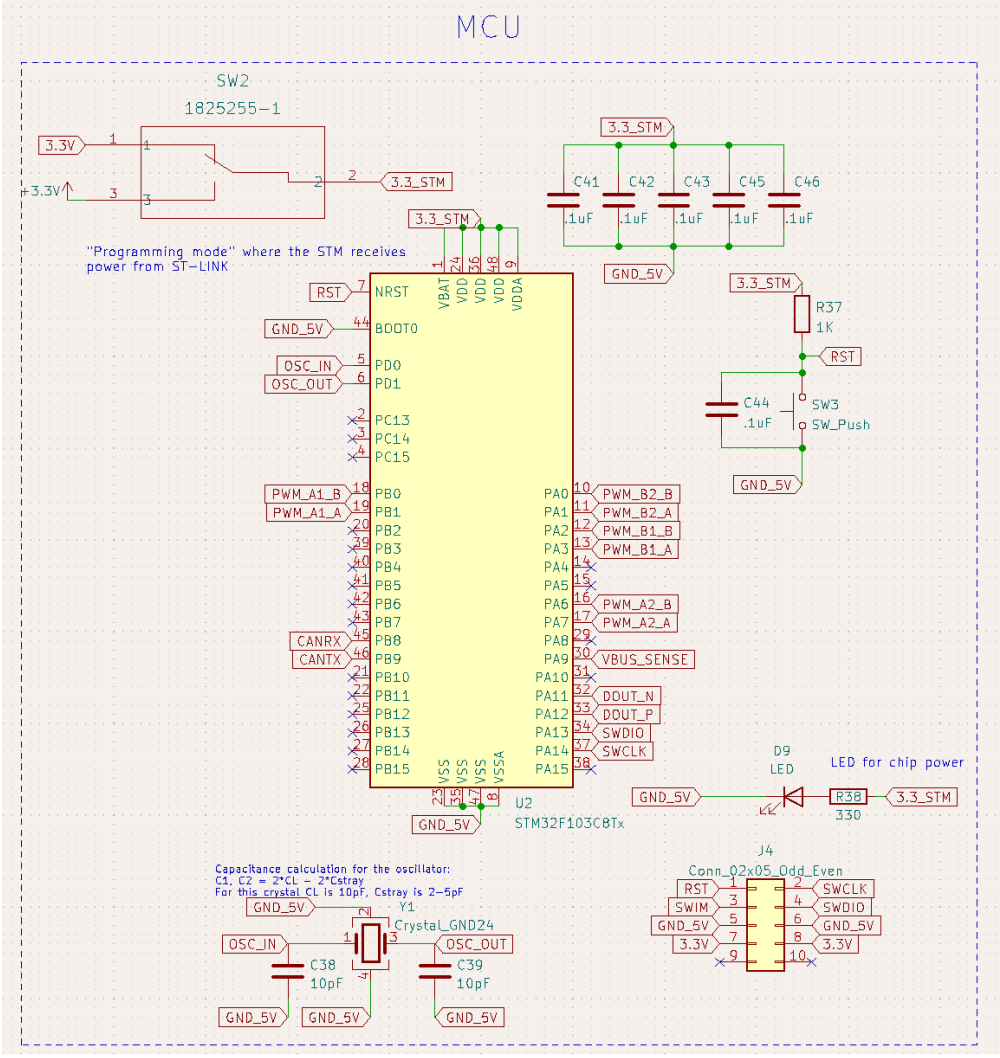


Figure 12. MCU Schematic

PCB Design

This board required multiple power and ground planes as well as special noise considerations. This board was designed as a 2 layer board because there is not an incredibly high density of traces and there are no incredibly high frequency signals that need to be shielded from noise. The buck converter switching IC had an example PCB layout in the datasheet which was followed exactly. The power planes included 48V, 5V, and 3.3V and there was a shared ground plane for 48V/24V and a separate ground plane for 5V and 3.3V. This ensured isolation of the high and low voltage signals.

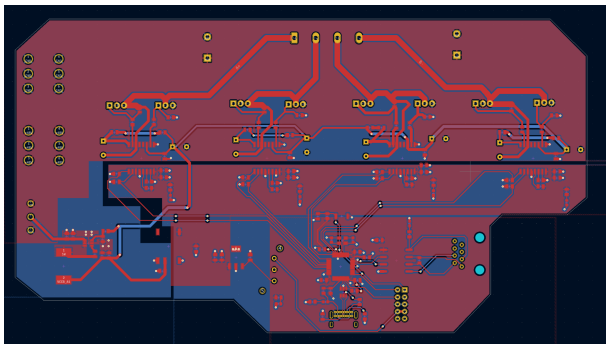


Figure 13. Copper Layers Only

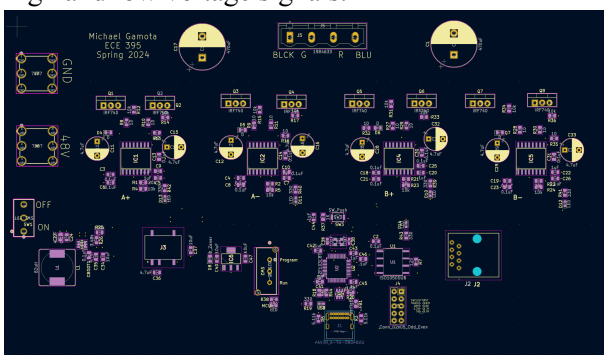


Figure 14. Non-copper Only

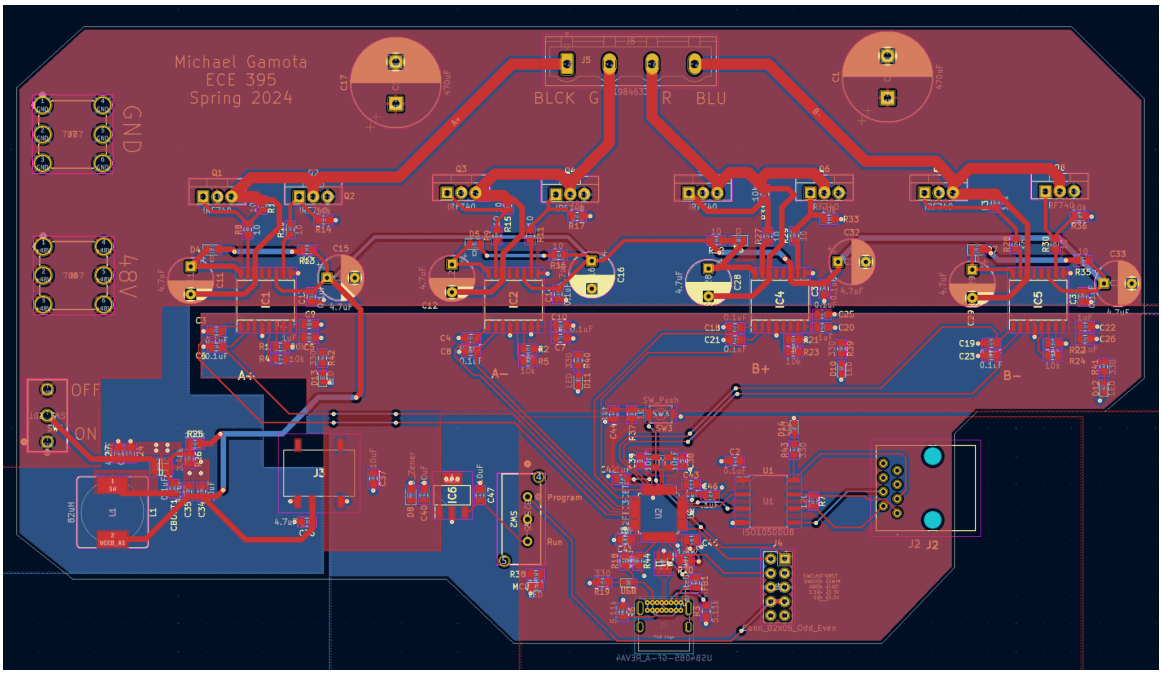


Figure 15. Complete Layout

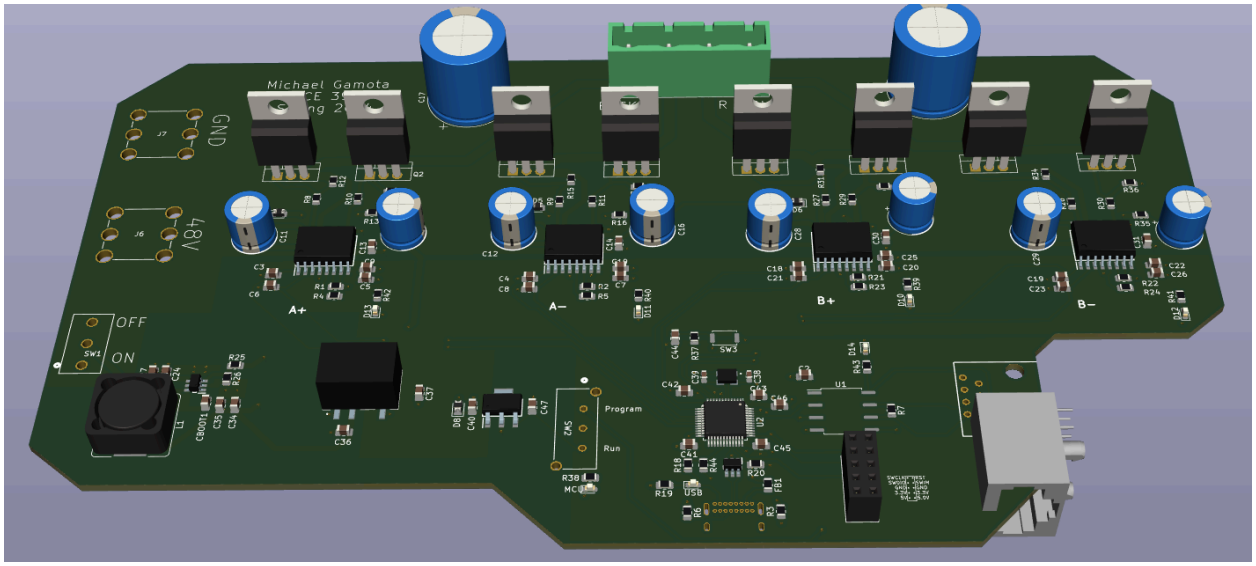


Figure 16. 3D rendering of assembled board (missing some components and rotation issue with connector in bottom right)

Assembly/Test

The assembly of the board was done entirely by hand using solder paste and wire solder. The BOM for this project can be found on my Github. Based on my experience, it is best to verify that each power conversion stage works before continuing assembly, starting with the 48V to 24V buck circuit. The next step should be verifying that the MCU can be programmed, followed by assembling the gate driver circuits, testing those, and finally the H-Bridge. The last step should be adding the LEDs and connectors.

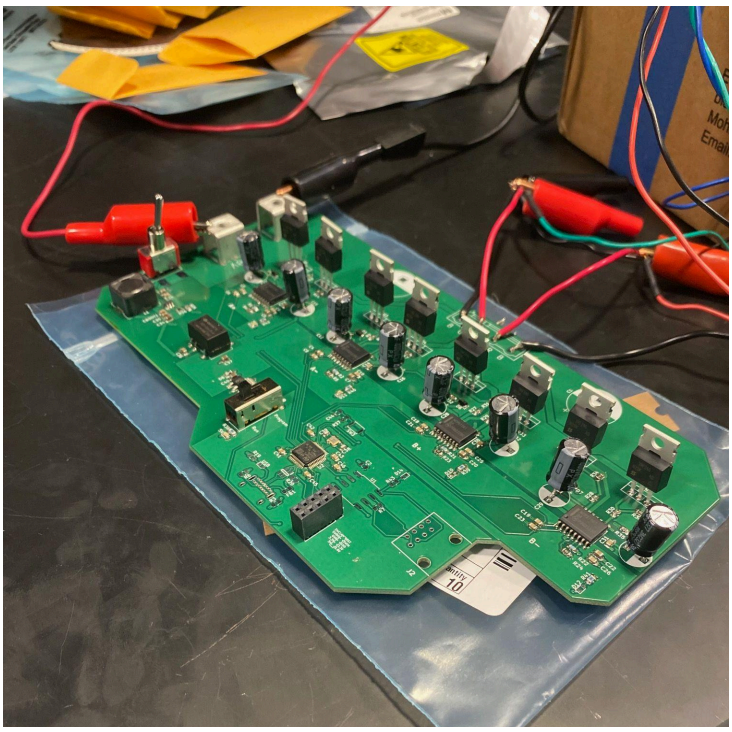


Figure 13. Partially Assembled Board

Software Development

While designing the board is one of the most fun stages of engineering, software is what really brings the device to life. As previously stated, the method for driving the stepper motor was to use isolated gate drivers driven with a PWM signal to create a “chopper” driver. The microcontroller I was using has 4 timer channels with 4 pins each, meaning that I can get 16 unique duty cycle PWM signals, with 4 different frequencies. I only needed to use 8 of the 16 pins. The USB connection was the other important part of this project because without input from USB or CAN (which I did not do any software development for), the motor can only move in a hardcoded sequence. In order to get the USB working, I used a USB for STM32 3rd party library which I found online which is credited.

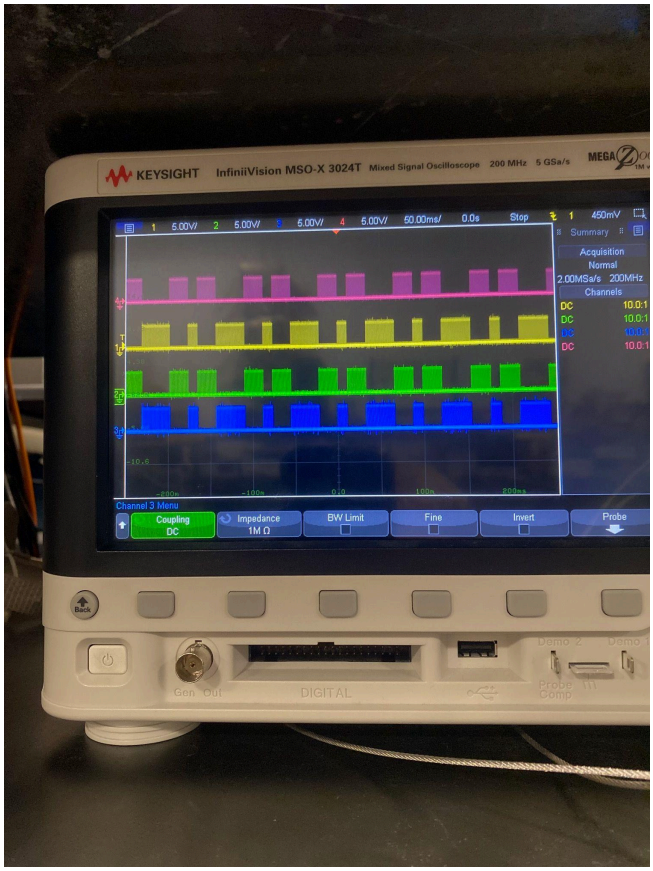


Figure 14. Testing the PWM signal generation in the context of the 4 steps

Stepping order: (1->2->3->4->3->2->1) which is a clockwise rotation immediately followed by a counterclockwise rotation

Conclusion

This project was a great experience in both digital and analog circuit design, from the USB and PWM signals to the H-Bridge control and power converters. This is the largest board that I have ever designed and it was satisfying to finally see the motor spin. The entire process was valuable as an academic experience as well as a personal one, seeing a project of this scale work as intended is very rewarding. There is a demo video which will eventually be posted on my website which I will include in the citations.

This stepper motor control system is the last piece of hardware required to control the EV Concept vehicle autonomously. We should now be able to steer the vehicle using this controller while controlling the speed of the vehicle by setting the cruise control, all with commands sent over the CAN bus which can be received either wirelessly from the telemetry board or from a piece of hardware in the vehicle itself. I will now be passing the board on to the Embedded Programming team to add it to the CAN network as we wait for the Autonomous team to get ready to deploy a perception and planning algorithm.

Weekly Progress

Progress:

- Week 1: Completed first 2 activities, got familiarized with lab equipment
- Week 2: Completed activity 3, began outlining project requirements
- Week 3: Completed activity 4, started component search
- Week 4: Finalized motor choice and submitted order. Finalized some component choices.
- Week 5: Components and system design finalized, working on schematic and layout.
- Week 6: Ordered STM32 dev board to start prototyping software, ordered components, layout to be finalized by EOW
- Week 7: Finalized and submitted PCB v1 (shown below). Submitted all parts orders. STM32 dev board arrived so software prototyping can start (main focus until the board arrives)
- Week 8: Wrote PWM based control functions for stepper rotation in full-step mode
- Week 9: Spring break
- Week 10: Verified PWM functions with oscilloscope
- Week 11: Board arrived, assembled power, MCU, and H bridge sections and tested on stepper from back bins(at 24V): It works!!
- Week 12: Finished assembly, tested and verified with 48V on actual motor, polished motor control functions, acquired steering wheel, did CAD design for coupling between stepper and steering wheel. Ran into roadblock with USB
- Week 13: Solved USB problem(device enumerates and shows up as a COM port), submitted PLA order for 3D printed coupler, going to assemble new board with USB functionality, CAD a case for the board
- Week 14: Printed coupler V1, started prototyping Python code and verified communication between MCU and Python over USB COM
- Week 15: Printed coupler V2 (final), further abstracted Python code so turning the wheel is as easy as turn(wheel, left/right, degrees)
- Week 16: Tried to develop a cool demo with OpenCV and first person steering wheel footage from video game, but ran into issue with motor, likely gearbox related because issue only arises with stepper with gearbox. Demo was not as cool as it could have been but still impressive, able to control motor with arrow keys in Python.

Figure 15. Project updates (via Canvas)

Appendix A: Code

Any "VCP" functions are from a third party USB for STM32 library which is cited below

```
void set_pwm(uint8_t coil, uint8_t channel, uint16_t duty_cycle){
    if (coil==1 && channel==1){
        TIM3->CCR1=duty_cycle;
    }
    else if (coil==1 && channel==2){
        TIM3->CCR2=duty_cycle;
    }
    else if (coil==1 && channel==3){
        TIM3->CCR3=duty_cycle;
    }
    else if (coil==1 && channel==4){
        TIM3->CCR4=duty_cycle;
    }
    else if (coil==2 && channel==1){
        TIM2->CCR1=duty_cycle;
    }
    else if (coil==2 && channel==2){
        TIM2->CCR2=duty_cycle;
    }
    else if (coil==2 && channel==3){
        TIM2->CCR3=duty_cycle;
    }
    else if (coil==2 && channel==4){
        TIM2->CCR4=duty_cycle;
    }

}
```

Figure 16. Function to set the PWM value based on coil and "channel"

```
void stop_motor(){
    for (uint8_t coil=2;coil>0;coil--){
        for (uint8_t ch=4;ch>0;ch--){
            set_pwm(coil, ch, 0);
        }
    }
}
```

Figure 17. Function to stop the motor

```
void set_phase(uint8_t this_phase){
    stop_motor();
    if (this_phase==1){
        set_pwm(1, 1, DUTY_CYCLE);
        set_pwm(2, 1, DUTY_CYCLE);
        set_pwm(2, 4, DUTY_CYCLE);
        set_pwm(1, 4, DUTY_CYCLE);

    }
    else if (this_phase==2){
        set_pwm(1, 3, DUTY_CYCLE);
        set_pwm(2, 1, DUTY_CYCLE);
        set_pwm(2, 4, DUTY_CYCLE);
        set_pwm(1, 2, DUTY_CYCLE);

    }
    else if(this_phase==3){
        set_pwm(1, 3, DUTY_CYCLE);
        set_pwm(2, 3, DUTY_CYCLE);
        set_pwm(1, 2, DUTY_CYCLE);
        set_pwm(2, 2, DUTY_CYCLE);

    }
    else if (this_phase==4){
        set_pwm(2, 2, DUTY_CYCLE);
        set_pwm(2, 3, DUTY_CYCLE);
        set_pwm(1, 1, DUTY_CYCLE);
        set_pwm(1, 4, DUTY_CYCLE);
    }
    phase=this_phase;
}
```

Figure 18. Function to set the phase based on stepper motor datasheet

```
void fs_right(){
    for (int i=4; i>0; i--){
        phase=phase%4;
        set_phase(++phase); //Increment phase

        //Hardware delay 8 MHz=0.12 uS per tick, 4300 ticks = 516us
        for (volatile uint32_t i =0; i<4300;i++);
    }
}
//Turn motor left (A1,A4)(B2, B3) -> (A2,A3)(B2, B3) -> (A2,A3)(B1, B4)-> (A1,A4)(B1, B4)
void fs_left(){
    for (int i=4; i>0; i--){
        if (phase<=1){
            set_phase(4); //Loop back to 4 after 1
        }
        else{
            set_phase(--phase); //Decrement phase
        }

        //Hardware delay 8 MHz=0.12 uS per tick, 4300 ticks = 516us
        for (volatile uint32_t i =0; i<4300;i++);
    }
}
```

Figure 19. Functions to turn the stepper motor clockwise and counterclockwise based on the sequence in the datasheet

```
void fs_right_deg(uint16_t degrees){
    uint16_t steps = (degrees/0.72); //10:1 gearbox so 0.18 degrees per step, fs_right and fs_left do 4 steps
    for (int i=steps; i>0; i--){
        fs_right();
//      for (volatile uint16_t i=0; i!=0x9C4; i++); //Hardware delay
    }
    position += degrees;
    stop_motor();

}

void fs_left_deg(uint16_t degrees){
    uint16_t steps = (degrees/0.72); //10:1 gearbox so 0.18 degrees per step, fs_right and fs_left do 4 steps
    for (int i=steps; i>0; i--){
        fs_left();
//      for (volatile uint16_t i=0; i!=0x9C4; i++); //Hardware delay
    }
    position -= degrees;
    stop_motor();

}
```

Figure 20. Functions to turn the motor a certain number of degrees

```
void zero_pos(){
    if (position>0){
        fs_left_deg(position);
    }
    else if (position<0){
        fs_right_deg(-1*position);
    }
}
```

Figure 21. Function to zero the motor position based on previous turns

```
vcp_init();
int len=0;
uint8_t buf[10];
uint8_t doneLeft[1]={"L"};
uint8_t doneRight[1]={"R"};
uint8_t error[1]={"E"};
uint8_t stop[1]={"S"};
uint8_t zero[1]={"Z"};



user_pwm_init();
stop_motor();
```

Figure 22. main() init code

```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

    len = 0;
    buf[0]=0;
    while (len<=0){
      len = vcp_recv (buf, 1000);  // Read up to 1000 bytes
      if (buf[0]==0){
          stop_motor();
          len = vcp_send (stop, sizeof(uint8_t));
          break;
      }
      else if (buf[0]==1){
          uint16_t rotate = 0x0000 | ((buf[1] << 8) | buf[2]);
          fs_left_deg(rotate);
          len = vcp_send (doneLeft, sizeof(uint8_t));
          break;
      }
      else if (buf[0]==2){
          uint16_t rotate = 0x0000 | ((buf[1] << 8) | buf[2]);
          fs_right_deg(rotate);
          len = vcp_send (doneRight, sizeof(uint8_t));
          break;
      }

      else if (buf[0]==3){
          zero_pos();
          len = vcp_send (zero, sizeof(uint8_t));
          break;
      }
```

Figure 23. main() while code (1 /2)

```
      else{
            len = vcp_send (error, sizeof(uint8_t));
            break;

      }
      break;
    }

    HAL_Delay(800);


  }
/* USER CODE END 3 */
```

Figure 23. main() while code (2 /2)

Citations

- 3rd Party USB Code Library: https://nefastor.com/microcontrollers/stm32/usb/stm32cube-usb-device-library/cdc-continued/
- Design Tool for Power Converters (TI): https://www.ti.com/lit/ds/symlink/lmr51610.pdf?ts=1715098052338&ref_url=https%253A%252F%252Fwww.mouser.com%252F
- Github for all files/code: https://github.com/mgamota2/StepperMotorController
- Website with media(eventually): https://mgamota2.github.io/

Image Citations

- Figure 2: https://www.electrical4u.com/bipolar-stepper-motor/#google_vignette
- Figure 3: https://www.linearmotiontips.com/what-is-a-chopper-drive-for-a-stepper-motor/
- Figure 6: https://webench.ti.com/power-designer/switching-regulator?powerSupply=0
- Figure 6: https://www.ti.com/lit/ds/symlink/lmr51610.pdf?ts=1715098052338&ref_url=https%253A%252F%252Fwww.mouser.com%252F
- Figure 8: https://www.omc-stepperonline.com/nema-23-stepper-motor-l-76mm-gear-ratio-10-1-mg-series-planetary-gearbox-23hs30-2904s-mg10
- Figure 9: https://www.onsemi.com/pdf/datasheet/ncp51561-d.pdf