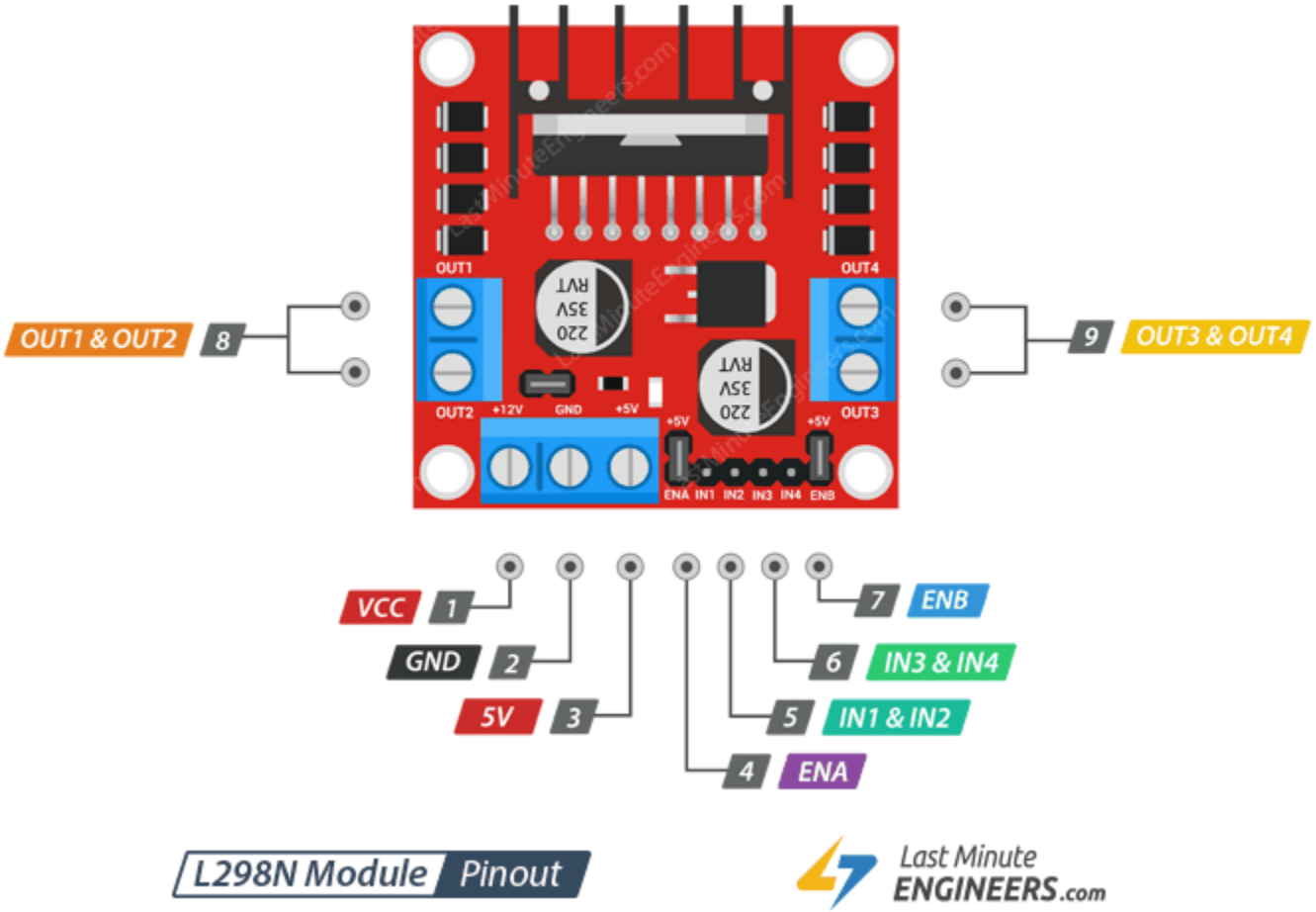
CS466 Lab 7 – Motor Servo. (Rev 1.0, 4/12/2022) – Will probably be updates
Final lab, Due Midnight, May 6

Notes:

- This lab will require two boards, I highly recommend that you work with a partner.
- This lab requires the use of an H-Bridge motor Driver we have the boards in the lab. A single Driver board will control 2 motors.



Objective:

- Learn how software PID control can be used to implement motor servos.
- Synchronize two servo systems via SPO communications.

Requirements:

- Remote control slave motor system that will use a basic PID control algorithm to seek motor position given from a simple master.
- Only the SPI connections and Ground (don't forget ground) are allowed between the master and slave. You should probably share the !RESET signal also
  - Q.
- The list of positions (encoder tics from zero) will be stored on the master board and delivered to the slave board for action using the SPI/Register methods that we have been implementing in labs 4 and 5
  - [435, 870, 1305, 1740, 3480, 5220, -5220, -3480, 0]

- Pressing SW1 will cause the Master board to write registers in the slave that cause the slave controller to drive the motor to a specific position.
- The master board contains the SW1 and SW2 user interface.
- Design, Implement and demonstrate a master-slave system to match requirements.
- The goal of the design is to have two switches on the control panel (master board) access registers on the slave board to implement a simple user interface for motor control.
    - SW1 – Start/Next
        - Initially you will set the motor to a known 'zero' position and press start. On Start the Master LED will turn green.
        - Subsequent presses will seek the positions stored in the actual list.
        - If the end of the list is reached continue to the start of the list.
    - SW2 – Prior
        - Nothing if SW1 not pressed first.
        - Sequence in the list in the opposite manner as SW1, moving backwards in the list
        - If decrementing past the beginning of the list continues from the last element.
    - SW1+SW2 – Motor Off
        - If both buttons are presses for 500ms put the motor in an un-powered state and turn on master red LED.
    - LED-Green
        - Hybrid heartbeat monitor. It should beat at a 10% duty cycle if both master and slave heartbeats are updating. If slave stops reporting switch to 90% duty cycle.. If the Master heartbeat stops it should stop.
    - LED-Red
        - Solid when the system is started and ready for a start button.
        - Rapid-Blink on any SW1 press and remain blinking until the motor position has settled within 20 encoder ticks of setting
        - Off indicates that the motor has settled at it's destination and ready for a position switch.

- The Motors will be driven by 12VDC. Direction/Torque will be given by imposing a PWM and Direction system.
- The firmware watchdog should be running on both master and slave and if encountered shut down the motors. If the master hangs it should reset and initialize the remote SPI registers with a power off indication.

- Slave Register Specification (leveraged and extended from Lab5):
    - The slave will use a 2-byte command sequence for all operations.
        - Byte 1, is organized as 2, 4-bit fields.
            - Bits 4-7: Command
                - 0x0b is defined as a write
                - 0x0a is defined as a read
                - 0x0c is a special test command that will hang the slave in a while (1); behavior. We will send this when we want to test the slave watchdog.
                - All other patterns are reserved for future use
            - Bits 0-3: Register Address of a device 8 bit register
        - Byte 2, is always register data either to be written in the case of a write or to be read in the case of read.
            - Special consideration must be taken into account on a read operation to have correct MISO data reg-MSB on the data line when the master raises the clock line.

- This Slave will contain **???** registers (was 3 in Lab 5)
  - MOTOR_POS_0
  - MOTOR_POS_8
  - MOTOR_POS_16
  - MOTOR_POS_24
    - 8 bit Read/Write registers
    - Address is 0, 1, 2, 3
    - the 32 bit value comprised of these numbers represents the current position of the motor.
    - This value needs to be requested via CMD_REG as it's unlikely you're be able to follow a live encoder value. The sequence will be write CMD_REG[GET_MOTOR_POS] then read these four bytes and reassemble the position on the Master.

  - MOTOR_CMD_0
  - MOTOR_CMD_8
  - MOTOR_CMD_16
  - MOTOR_CMD_24
    - 8 bit Read/Write registers
    - Address is 4, 5, 6, 7
    - the 32 bit value is the position the Master wants the slave to seek. The command sequence is to first update these four registers then write a bit to CMD_REG[SET_MOTOR_POS].

  - CMD_REG
    - 8 bit write only register. Reading is undefined and could return garbage.
    - Address is ~~0x10~~-10 (0x0A)
    - Writing a bit to individual fields in this register will cause actions by the slave
    - BIT0: GET_MOTOR_POS (LSB)
      - Will copy the current 32 bit encoder position reading into the 4 single byte MOTOR_POS_[x] registers
    - BIT1: SET_MOTOR_POS
      - Will copy the data in the 4 MOTOR_CMD[x] registers into the PID motor setpoint.

  - STAT_REG
    - 8 bit read only register. Writing is undefined and not recommended
    - Address is ~~0x11~~ 11 (0x0A)
    - Status codes
      - 0-127 System Running Status
        - 0 – READY
        - 1 – SEEK
        - ...Others? (Be sure to document)
      - 128-256 – ERRORS (Make sure you document what errors you support)

Prior To Lab:

- Take time to read and understand the requirements and what the system will do.
- Review the PID Controller Wiki https://en.wikipedia.org/wiki/PID_controller
- Review the Servomechanism Wiki https://en.wikipedia.org/wiki/Servomechanism

- Review the L298N Board Info Document (L298N_Board_Info.pdf) and see the detailed description that was put together using an Arduino board https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/
- Because we are short on time, I have provided motor and PWM header files. Look first at the header files to see what each module exposes. Spend enough time to see what your motor.c and pwm.c will do. Note that your application need only include motor.h. the pwm control method we are using should not be visable to your higher-level code.
  Q:
- You will need to modify the files and/or your connections to match modules with your configurations.

How I would progress,
Either start with 1-3 or skip them for now until you get the positional control running.

1. ☐ Connect 4 SPI wires from your master to slave boards and make sure that your lab5 is working. Slow-down your master somewhat so that there is timing headroom. With the Slave running the position encoder the SPI state machine will be fighting for interrupt bandwidth.
   a) The Master code is quite simple and completed quickly
   b) The slave code should report via serial what any new command is

2. ☐ Setup the lab6 work on the Slave board so that you can monitor motor encoder position.
   a) We will be running both position interrupts and SPI interrupts concurrently.
      Q: Which interrupt should be a higher priority?
3. ☐ Setup your slave and master boards so that you have a green heartbeat light flashing for now.

Note: To isolate the work into usable steps get the servo controller working on the master board first so that you have working positional control working before integrating the master-slave communications.

4. ☐ I have provided a motor.h API that uses the API documented in pwm.h. You will need to drive the integer ‘static int32_t _encoder’ with the quadrature encoder you constructed in Lab 6.

5. ☐ Allocate additional GPIO's off your master and slave boards for connection to the driver board. You will need two controllable GPIO's for position and the single line for the PWM signal output. If you make the same pins on both controllers your motor and pwm modules will be portable between master and slave except for PID tuning values. It would serve you well to define these values in independent headers or with a MASTER/SLAVE define in the code (personally I like header files selected by compiler flags, e.g

```
#ifdef L7_MASTER
#include "pid_tune_master.h"
#else
#include "pid_tune_slave.h"
#endif
```

Then you can have your 3 tune parameters (which you will be changing a lot) in stand alone files. You can setup the define with a line in your CMakeLists.txt file

```
add_definitions(-DL7_MASTER)
```

6. ☐ Verify that all your GPIO's and PWM signal can operate independent of each other.
   a) This has been problematic in the past. Make sure on a scope that all your signals are producing your expected output.

7. ☐ Using the Arduino motor driver link connect the LM298 Driver board to your board. For running only one motor you will need to use 2 direction GPIO connections and a single PWM connection.

8. ☐ Using a simple loop in your Slave board that sequences through 2 or three motor positions once per 5 seconds or so until you get the PID tuned. I did not provide valid PID tuning parameters. You will need to implement the simple PID control loop in the PID Wiki, it's documented as pseudocode near the end if the Wiki in a few flavors.

9. ☐ Follow the manual tuning section and experimentally get your motor to respond to the sequence of positions. Use a very simple dedicated loop until you have this code working on your master. (Note: Since the parameters will change for every mechanical system change, don't spend a lot of time tuning until you have final paddles (Which I hope to have Thursday 4/14)

   Q: What are you using for *dt* and why?

10. ☐ Replace the motor the slave motor and repeat to get tuning parameters for the slave motor. At this point you should have

    Q: Record your tuning parameters

11. ☐ Using two motors driven by your two boards set them up in the mount side by side and have them track each other? The master and slave should both follow the positional sequence given in the lab prolog.

12. ☐ Include in your lab include a video of the dual motor operation. Similar to past videos but I would like some verbal description, I took these videos in class.
    https://photos.app.goo.gl/8rFfGgKvmLxu7K6H9
    https://photos.app.goo.gl/VEzo7rtvULeUsk9a8

Be sure to writeup your lab according to the lab format handout.