# Pick and Place Project 2016

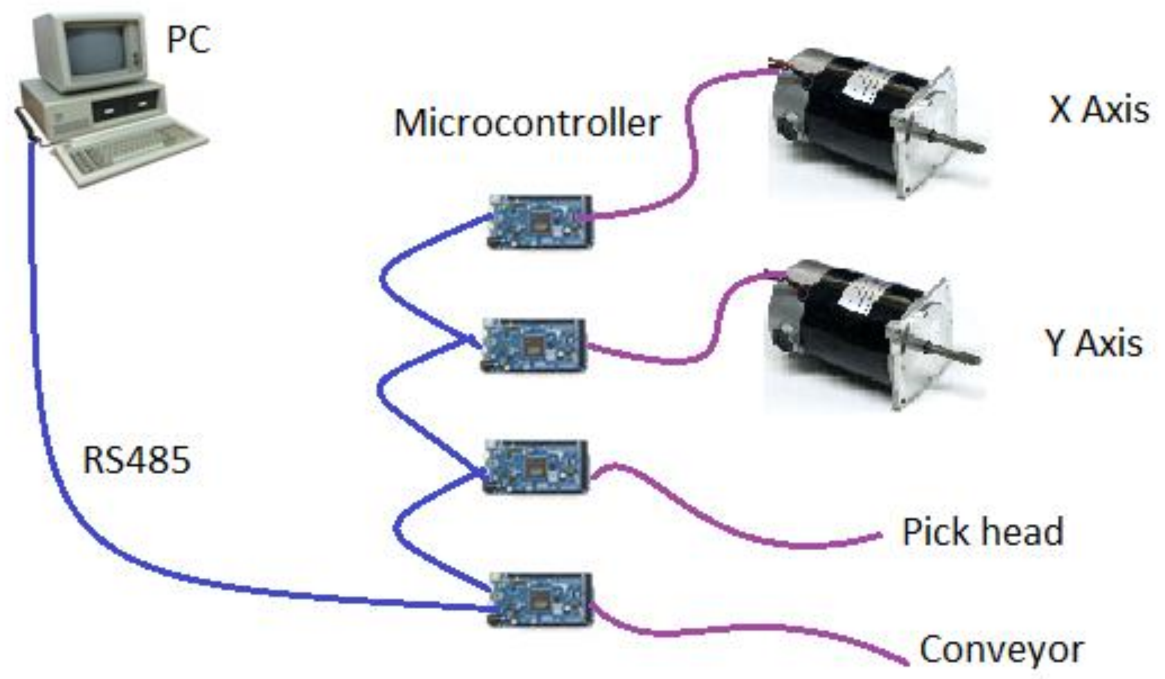
By Alan Lowther & Jordan Woolmer

# Pick and Place Project 2016
## CONTENTS

# Identify Project

The project was to replace the old controllers inside an obsolete 1980's pick and place machine so it could be used with modern software.
As we were the first team to work on this project we had to make some decisions that would affect future teams. We decided to make a modular system with a set of microcontroller PCB's all daisy chained together and controlled from a master controller possibly a Personal Computer.

We decided to use RS485, an industrial serial bus standard to link the controllers together, only a few of the Arduino range of microcontrollers have CAN so this would be limiting the type of microcontrollers that could be used.



We chose to control the X and Y motor's, both motors are controlled in the same way.

## Pricing and Procurement

The pricing and sourcing was all through either X-ON , element14, Jaycar , and Altronics. Majority is sourced from X-ON and element14. This is because they were cheaper sources considering the fact that we could not use eBay , and they had the specific surface mount components that we required . The rest was from either Jaycar or Altronics, as they were connectors or cable that was cheaper and easier to buy locally as it was more expensive on element14/X-ON.

The most expensive part was the Arduino DUE , at AUD$61 for each board. But the benefit of this is that these are the original Arduino boards , so they have the correct components with the correct bootloader , for easier programming . It is also easier to find documentation for the original , as it is more developed , and has had more revisions than other boards , as it was created first . The Arduino DUE has CANbus compatibility , which makes it ideal for our project , as well as quadrature compatibility , which is the type of encoding that the motor uses .

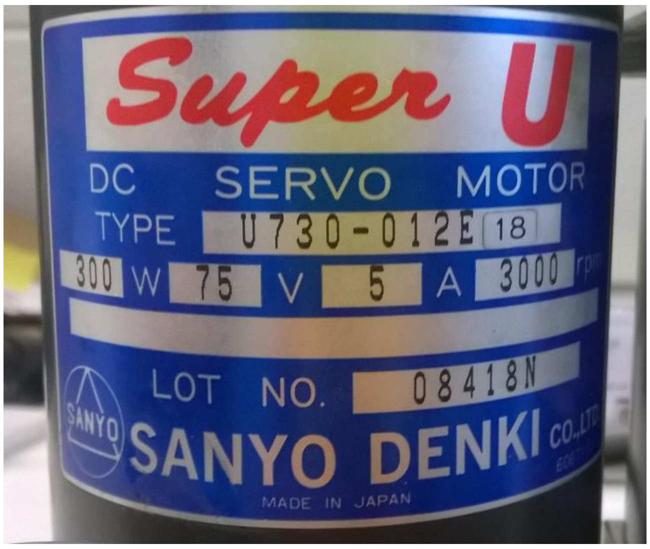
The MOSFETS are intentionally 10 Amps at 100v , so there is plenty of overhead space for excess current and higher voltage , especially when the motor starts initially .

The network protocol that we are using is RS485 , which is one of many protocols used in industry. It is a two wire system , and CAT6 cable makes for an ideal transmission medium . Inside the cable there are 4 pairs of wire , so in reality , 80 metres of twin conductor twisted pair will be available . This is more than enough , as all of the wiring will be within the pick and place machine .

The total cost of this build is $242.34. This is because of the cost of the Arduino DUE. $61.00 was the cheapest source of the DUE board . The cost excludes the cost of the PCB milling materials. The full list of the Bill of Materials is in Appendix D .

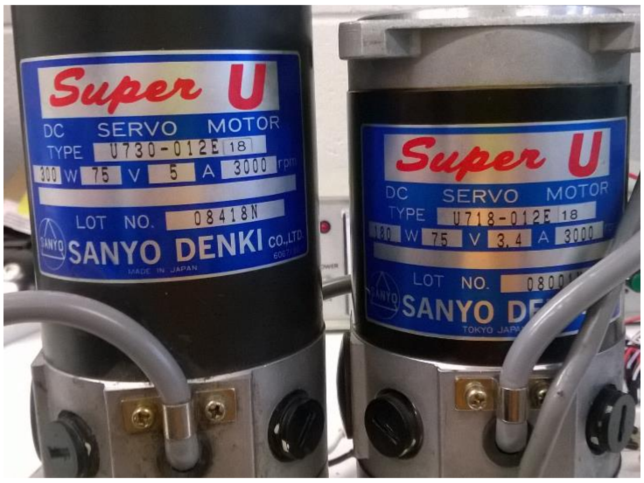
## Identify Problems

The first problem was the voltage the motors used in the pick and place machine, we were unable find an H bridge capable of supplying 75 volts.



We were given the choice of sourcing a lower voltage 300W motor, looking on the internet we found this type of motor is very common motor and only at 75 volts.

The X and Y motors are of different sizes as one motor moves more weight than the other so we decided to build an H bridge capable of being interchanged that could drive either motor.



# Circuit Design and Testing

The circuit schematic layout is in appendix C and D. The programs used was EAGLE , and Multisim. The circuit was designed in EAGLE and tested in Multisim . The testing was done beforehand so if there were any problems , then they could be corrected in simulation , rather than redesigning the board after any errors had been made . The circuit works , and because of the logic control system , the H Bridge has three modes . These modes are forward , backwards , and stop. The logic control system is based on a 4001 CMOS quad-NOR gate . The chip has been connected in a way so that there are two inputs to the gates . Each input controls two gates . The 4001 gate is added into this board as a safety feature to prevent short circuiting , and conflict between other H-Bridges. The 4001 gate controls an opto-isolator , the TLP293-4 . This is a 16 pin SOIC with 4 opto-isolators on the one chip . These are what controls the H-Bridges . These opto-isolators isolate the low voltage logic circuit from the high voltage motor control circuit .

The two gate inputs to the 4001 are going to be connected to the Arduino DUE development board. This board provides digital signals into the 4001 which can be programmed to be either off or on constantly , in intervals , or pulsing . The motor has a rotary encoder , and this is a quadrature  encoder . This is the main reason that we chose the Arduino DUE . It has CAN Bus , and Quadrature encoding , which was ideal for our project . The rotary encoder does not connect to the H Bridge in any way , but the rotary encoder connects to the DUE board to provide data to the DUE to determine the motor’s position . In the simulation , instead of an Arduino DUE providing the inputs , there is two switches that provide 5 volts into the gate when closed  . This is considered as a HIGH level signal . When the gates are open , 0 volts is going into the gate . This is considered as a LOW level signal . Because the motor is such a high current ( 4 Amps ) , a component that can switch a high amount of current was required .

A relay could not be used because they are too slow for our needs , and only have a set amount of switching cycles . The relays also make a clicking noise , which may become a nuisance , especially if the 4 relays are switching at high speed . So we decided to use MOSFETS. A MOSFET is a type of transistor , but is mainly used for power control . They are commonly applied in high power devices , e.g. audio amplifiers , motor controllers , etc. They have no moving parts , and can switch a load faster than a relay . They will also last longer than a relay , because of the absence of moving parts. There are 4 of these MOSFETS on a board , two P-channel and two N-channel . For each direction , there is always one P-channel and one N-Channel  that are in the ON state. If all 4 of these MOSFETS were to be ON , then they may be destroyed . This is why the 4001 CMOS gate is on the H-Bridge . It prevents more than two MOSFETS from being in the ON state.

The MOSFETS have a gate to source voltage of 20 Volts of less . But it is assumed that the power supply is between 75 and 90 volts DC. The solution was to use pull down resistors to ground for the n channel , and pull up resistors to 90V for the p channel . This also prevents the MOSFETS from turning on unexpectedly . The Arduino DUE needs to control the motor , and the MOSFETS , so having pull up/down resistors is ideal for this , as well as dropping the voltage for the gates.

# PCB Design

The PCB was designed in EAGLE . EAGLE stands for Easy Applicable Graphics Layout Editor . It offered as freeware , however limited to a build space of 100mm by 80mm. This is the size of the board for the H bridge. The board could have probably been made a bit smaller , however if it were to be smaller , then there would be less space for the heat from the MOSFETS to spread. The MOSFETS that were selected are the IRF530S N channel MOSFET , and the SUD50P10-43L P channel MOSFET. Both of these MOSFETS can handle up to 100 VDC , but the currents are different. The N channel MOSFET can handle up to 14 A and the P channel MOSFET can handle up to -37.1 A . The links to the datasheets for these components are in Appendix H .

In order for the MOSFETS to operate correctly , pull up and pull down resistors to drop the voltage to the gate pin of the MOSFET , as well as preventing the MOSFET from switching on unexpectedly . If there were no resistors , then the motor may rotate at random times , which could be dangerous , especially if someone opens the door to remove the board from the pick and place machine , and the motor may create a pinch point that an arm , hand , fingers or hair may get caught . When controlled by the Arduino DUE , this prevents this from happening , as the motor only rotates when the DUE tells it to do so by outputting a HIGH level digital signal.

The DUE can be programmed by the user . The program is in Appendix A. The DUE can control the H Bridge effectively because of the pull up/down resistors. These resistors are 10K , providing enough voltage drop to drive the gates . The resistors on the board are a 2010 SMD package . These resistors are excessive , but justified for many reasons . The resistors are in this package , so that more traces can go under them , thus minimising the amount of 0 ohm resistors. When the design was completed , only one 0 ohm resistor was required for the board to be completed on a single layer.

The track thicknesses vary , from 0.01 inches , to 0.254 inch (0.25 mm to 6.45 mm). The thinnest tracks are for the opto isolators , and the thicker tracks are for the MOSFETs , and the motor output pads. The 0.25mm tracks are the thinnest possible , and may be too thin . Mainly because these tracks will be very weak , as there is a tiny amount of copper for these tracks . A possible solution would be to run solder over these tracks , so they can handle more current , while being more durable than what they currently are. The tracks are thin so they could be routed without encountering any clearance errors . There were a lot of clearance errors on the board , the solution was to make them thinner .

The board took about 6 weeks to design and make . Future plans are to make the H Bridge into an Arduino DUE shield that can plug onto the top of the DUE directly , eliminating the amount of cable that is used to connecting the DUE to the current H Bridge . The board is practically functional as it was tested in Multisim prior to designing the board . A picture of the board traces in EAGLE is in Appendix E , and the picture of the milled board is also in Appendix E . The mill was not levelled properly at the time , so some of the copper still remained between the traces and the unmilled copper . This was removed using a flat metal ruler to scrape out the copper that was not required.

# Software

To build a high speed accurate motor controller we were limited to the ARM based AT91SAM3X8E microcontroller because of the speed high speed quadrature output on the back of the motors. The AT91SAM3X8E ARM microcontroller is fitted to the Arduino DUE development PCB's, this gave us something off the shelf to build a prototype with.

As we going to use the Arduino DUE in the prototype the software was written in the Arduino IDE. The Arduino IDE has lots of great library files, but Arduino DUE microcontroller is the development PCB in the Arduino range with quadrature input so no library files were available. After some research we found some information on enabling the quadrature input and which memory locations to read the current value from.

The first program was written to just test the quadrature output of the motor with an Arduino DUE. The motor had to be turned by hand and 1 revolution, no matter how fast or slow gave 4000 pulses.
For motor testing we were able to power a spare 75 volt motor from a 20 volt power supply and control the motor using an H Bridge designed for low power low voltage motors. The H Bridge was controlled using a direction signal and PWM to control the speed. We were able to exercise the motor using feedback from the quadrature input, making the motor do one revolution forward then one revolution backwards.

As there was a fair bit of over-shoot, the software was modified to ramp down the PWM as it got closer to the set-point with very little over shoot and very good accuracy.
The USB serial port on the Arduino DUE development PCB was used for programming and debugging, a second serial port was enabled and a serial interface connected so that set-points could be sent to the Arduino DUE.

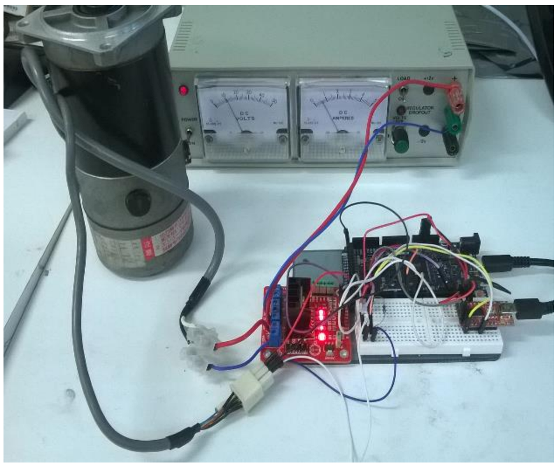
During testing it was discovered that there was very little torque from the motor when it was close to the set-point, a decision was made that we would need to use PID control to get more torque from the motor when it was close to the set-point.

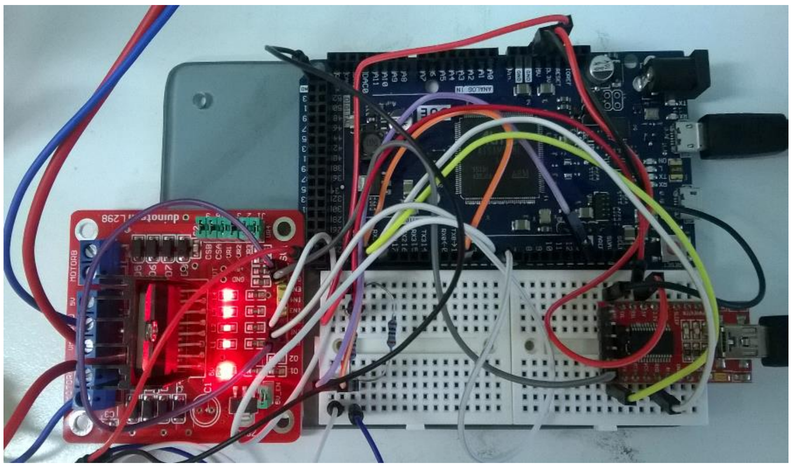
Arduino provide a PID library for the Arduino development PCB's, this took a lot of development to get working correctly, the main problem with the motor was not having a load, so it would spin up very fast and oscillate wildly.

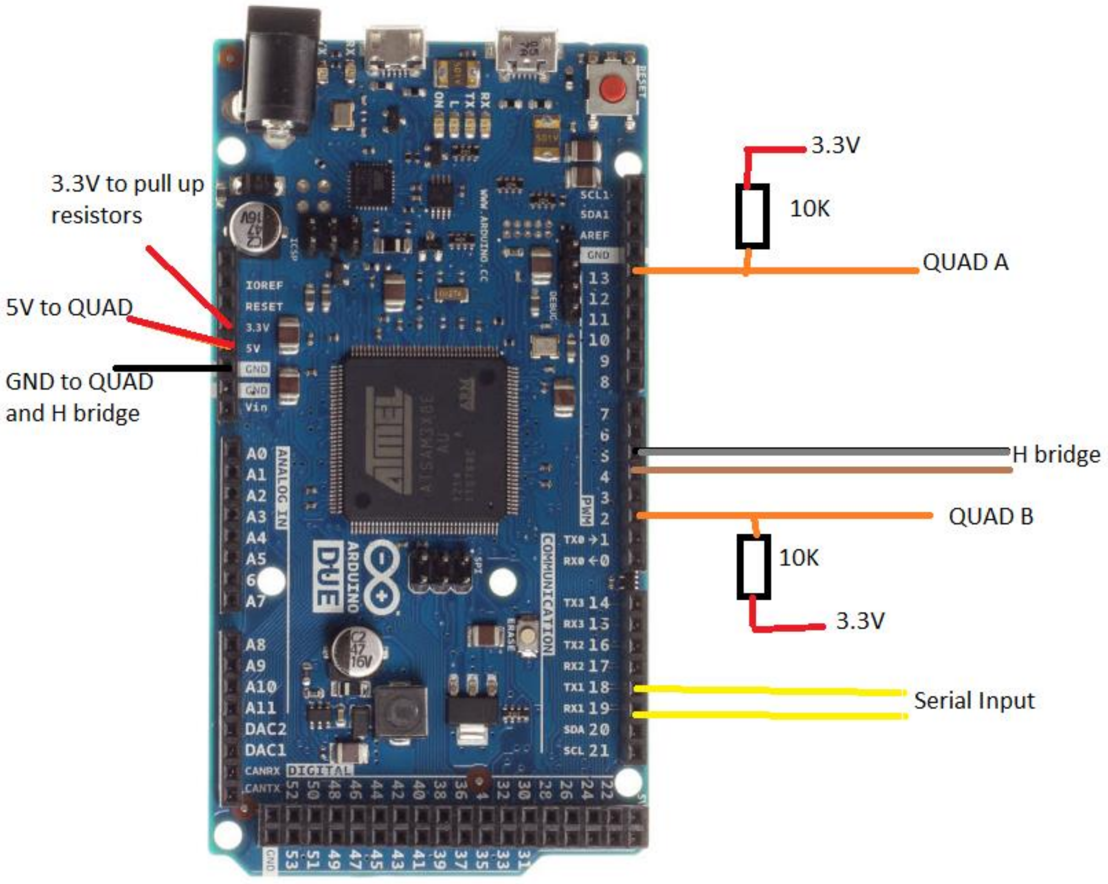
We were able to demonstrate the speed, torque and accuracy of the motor controller on the work bench.

## Prototype design

Due to time restraints it was decided to build a prototype before making a PCB and test the software. The prototype was all built from off the shelf parts, it did mean that testing and circuit design were done together and any problems could be fixed in software or hardware.







The prototyping was based on an Arduino (compatible) DUE board, a L298N H-Bridge motor shield, and a bench power supply. This is not identical to what will be installed in the pick and place machine, but provides a rough idea on how well the program will work. The L298N has a total power dissipation of 25 W . This is not enough to drive the motors as they require 300 W to run.

The motors are very high torque, and are very hard to counteract the rotational force that the motor runs. If the rotational force has a load on it, then the H bridge is destroyed as there is way too much current for the L298 to handle. The same thing would happen on the 10A H-Bridge, but the power on the label on the motor is 300 W. This is maximum load current.  The link for the datasheet of the L298 is in Appendix H .

# Conclusion

Though simulation in Multisim and building a prototype we were able to prove that the pick and place machine could have the control system updated using microcontrollers.
Multiple microcontrollers would be needed but due to their low cost this makes it a viable cost effective way of bringing the pick and place machine into the 21st century and give it a new lease of life.

Using a modular, distributed control system using interchangeable microcontroller PCB's and controller PCB's has lowered the cost of keeping the pick and place machine running for many years. As newer technologies appear each module can be upgraded.

With no documentation on the current pick and place machine we had to make a lot of assumptions, reading values from the motors and assuming there were going to run at their maximum voltage and maximum current as a worst case, the H bridge was over designed to take this into account.

With the information we have provided in this project documentation future projects on the pick and place machine will have a better start than we did. Not only did we have to decide on a data bus from the main computer to each controller we also had to find a microcontroller with a high speed quadrature input.

The pick and place machine is built around a feedback system using a quadrature encoder which limited us to the ARM based Arduino DUE which took every one in the team from their usual choice of microcontroller.
As a team we found the project challenging having to design the hardware and the software.

# Appendices

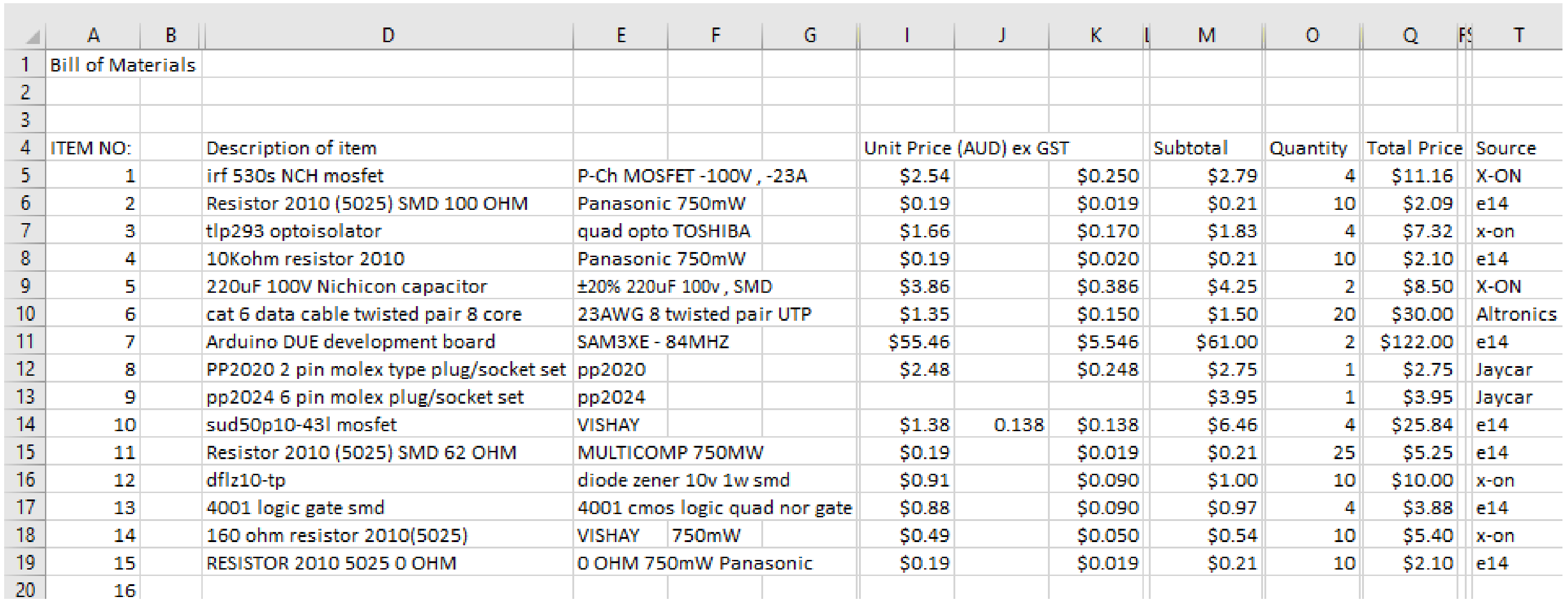
# Appendix A

## Arduino DUE software

```
#include <PID_v1.h>

const int quad_A = 2;
const int quad_B = 13;
const unsigned int mask_quad_A = digitalPinToBitMask(quad_A);
const unsigned int mask_quad_B = digitalPinToBitMask(quad_B);
double Setpoint=0, Input=0, Output=0;
double ku=0.35;
double tu=0.3;
int deadband = 35, Value =0;
double Kp=0.6*ku, Ki=tu/2, Kd=tu/8;

PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

String C, D;

void setup() {
    // activate peripheral functions for quad pins
    REG_PIOB_PDR = mask_quad_A;
    REG_PIOB_ABSR |= mask_quad_A;
    REG_PIOB_PDR = mask_quad_B;
    REG_PIOB_ABSR |= mask_quad_B;
    REG_PMC_PCER0 = (1<<27);
    REG_TC0_CMR0 = 5;
    REG_TC0_BMR = (1<<9)|(1<<8)|(1<<12);
    REG_TC0_CCR0 = 5;

  Serial.begin(115200);
  Serial1.begin(38400);
  pinMode(4,OUTPUT), (5,OUTPUT) ;
  digitalWrite(4,LOW), (5,LOW) ;
  myPID.SetOutputLimits(-(255-deadband), (255-deadband));
  myPID.SetControllerDirection(DIRECT);
  myPID.SetMode(AUTOMATIC);
}

void loop() {
 if (Serial1.available() > 2) {
 digitalWrite(4,HIGH);
 digitalWrite(5,HIGH);
 C = Serial1.readString();
 if (C.substring(0,2) == "x=") {
 C = C.substring(2,8);
 Input = C.toInt();
 Serial1.println(Input);}
 }

 Value = REG_TC0_CV0;
 Setpoint = Value;
 if (abs(Input - Setpoint) > 200) {deadband = 30;}
 else {deadband = 35;}
 myPID.Compute();

 Serial.print (Setpoint);
 Serial.print ("        ");
 Serial.print (Input - Setpoint);
 Serial.print ("        ");
 Serial.print (Output);
 Serial.print ("        ");

  if (Input - Setpoint == 0) {
    digitalWrite(5,LOW);
    digitalWrite(4,LOW);}
  else {SetTorque(Output);}

Serial.println (" ");
}

void  SetTorque(int T){

  if (T > 0) {
    T=T+deadband;
    if (T > 255) {T = 255;}
    digitalWrite(5,HIGH);
    analogWrite(4,255 - T);
    Serial.print (255 -T);
  }

  if (T < 0) {
    T=(-T)+deadband; // reverse T
    if (T > 255) {T = 255;}
    digitalWrite(4,HIGH);
    analogWrite(5,255 - T);
    Serial.print (255 -T);
  }
}
```

# Appendix B

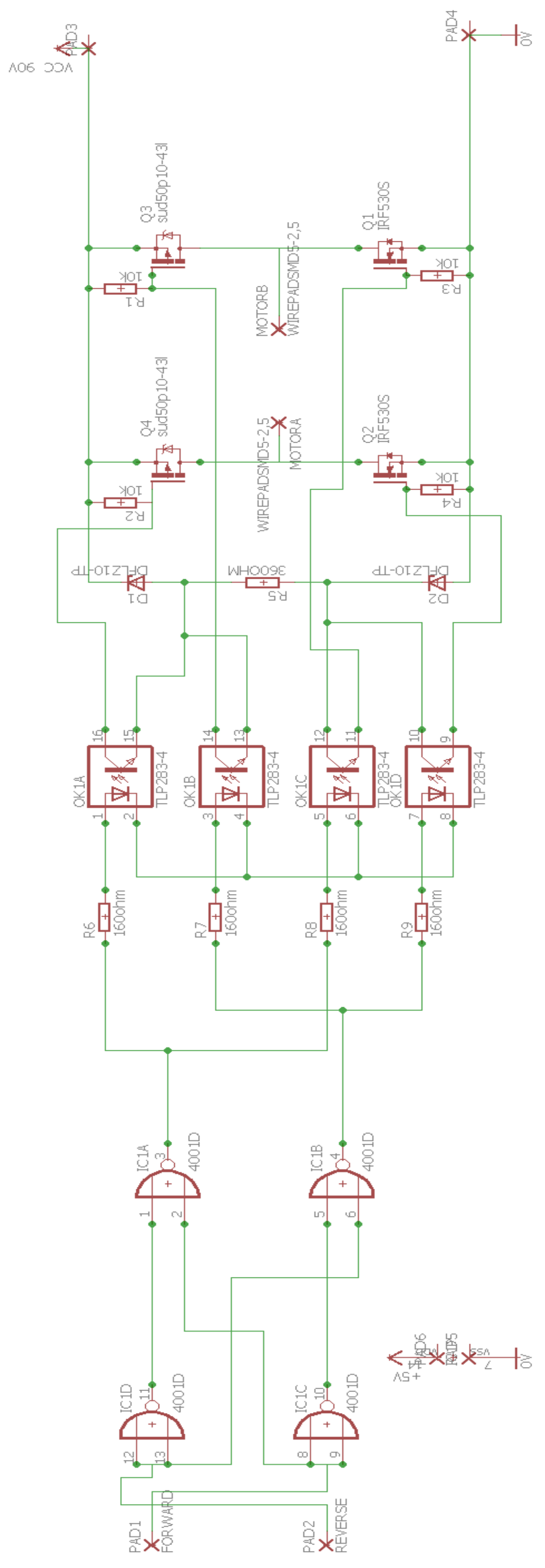
## Bill Of Materials



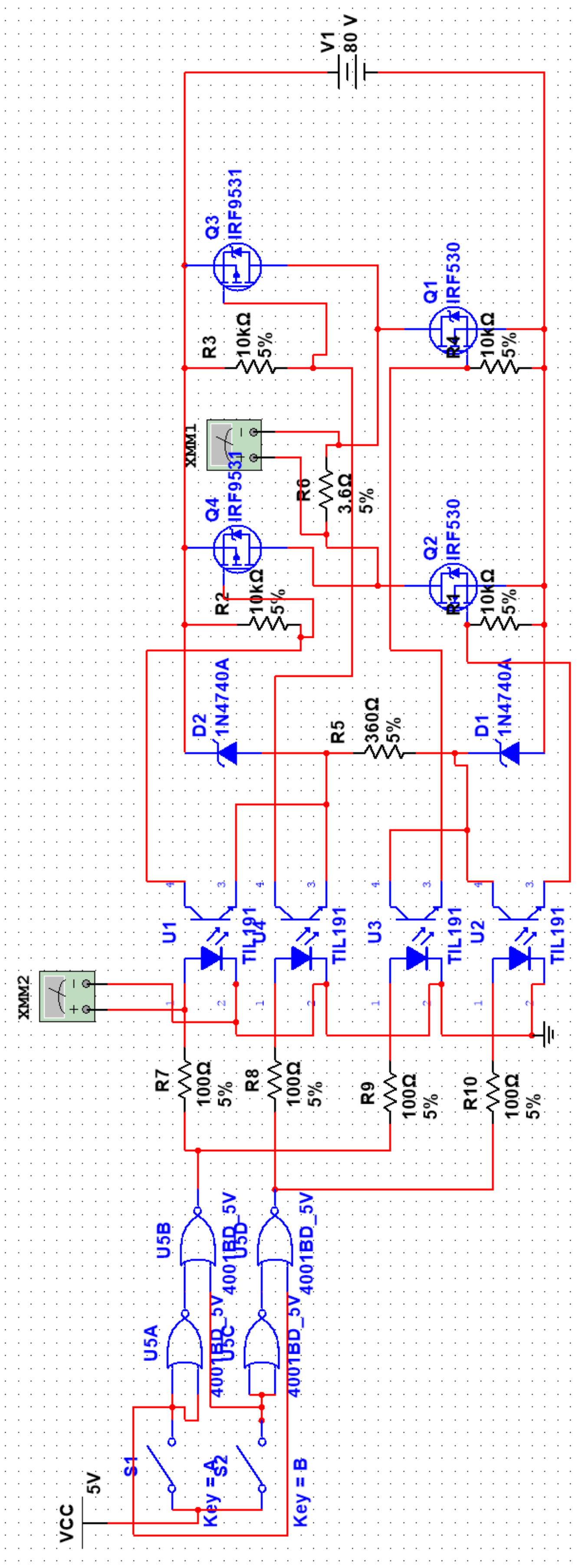
| | A | B | D | E | F | G | I | J | K | M | O | Q | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Bill of Materials | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | |
| 4 | ITEM NO: | | Description of item | | | | Unit Price (AUD) ex GST | | | Subtotal | Quantity | Total Price | Source |
| 5 | 1 | | irf 530s NCH mosfet | P-Ch MOSFET -100V , -23A | | | $2.54 | | $0.250 | $2.79 | 4 | $11.16 | X-ON |
| 6 | 2 | | Resistor 2010 (5025) SMD 100 OHM | Panasonic 750mW | | | $0.19 | | $0.019 | $0.21 | 10 | $2.09 | e14 |
| 7 | 3 | | tlp293 optoisolator | quad opto TOSHIBA | | | $1.66 | | $0.170 | $1.83 | 4 | $7.32 | x-on |
| 8 | 4 | | 10Kohm resistor 2010 | Panasonic 750mW | | | $0.19 | | $0.020 | $0.21 | 10 | $2.10 | e14 |
| 9 | 5 | | 220uF 100V Nichicon capacitor | ±20% 220uF 100v , SMD | | | $3.86 | | $0.386 | $4.25 | 2 | $8.50 | X-ON |
| 10 | 6 | | cat 6 data cable twisted pair 8 core | 23AWG 8 twisted pair UTP | | | $1.35 | | $0.150 | $1.50 | 20 | $30.00 | Altronics |
| 11 | 7 | | Arduino DUE development board | SAM3XE - 84MHZ | | | $55.46 | | $5.546 | $61.00 | 2 | $122.00 | e14 |
| 12 | 8 | | PP2020 2 pin molex type plug/socket set | pp2020 | | | $2.48 | | $0.248 | $2.75 | 1 | $2.75 | Jaycar |
| 13 | 9 | | pp2024 6 pin molex plug/socket set | pp2024 | | | | | | $3.95 | 1 | $3.95 | Jaycar |
| 14 | 10 | | sud50p10-43l mosfet | VISHAY | | | $1.38 | 0.138 | $0.138 | $6.46 | 4 | $25.84 | e14 |
| 15 | 11 | | Resistor 2010 (5025) SMD 62 OHM | MULTICOMP 750MW | | | $0.19 | | $0.019 | $0.21 | 25 | $5.25 | e14 |
| 16 | 12 | | dflz10-tp | diode zener 10v 1w smd | | | $0.91 | | $0.090 | $1.00 | 10 | $10.00 | x-on |
| 17 | 13 | | 4001 logic gate smd | 4001 cmos logic quad nor gate | | | $0.88 | | $0.090 | $0.97 | 4 | $3.88 | e14 |
| 18 | 14 | | 160 ohm resistor 2010(5025) | VISHAY | 750mW | | $0.49 | | $0.050 | $0.54 | 10 | $5.40 | x-on |
| 19 | 15 | | RESISTOR 2010 5025 0 OHM | 0 OHM 750mW Panasonic | | | $0.19 | | $0.019 | $0.21 | 10 | $2.10 | e14 |
| 20 | 16 | | | | | | | | | | | | |

# Appendix C

i/ EAGLE schematic



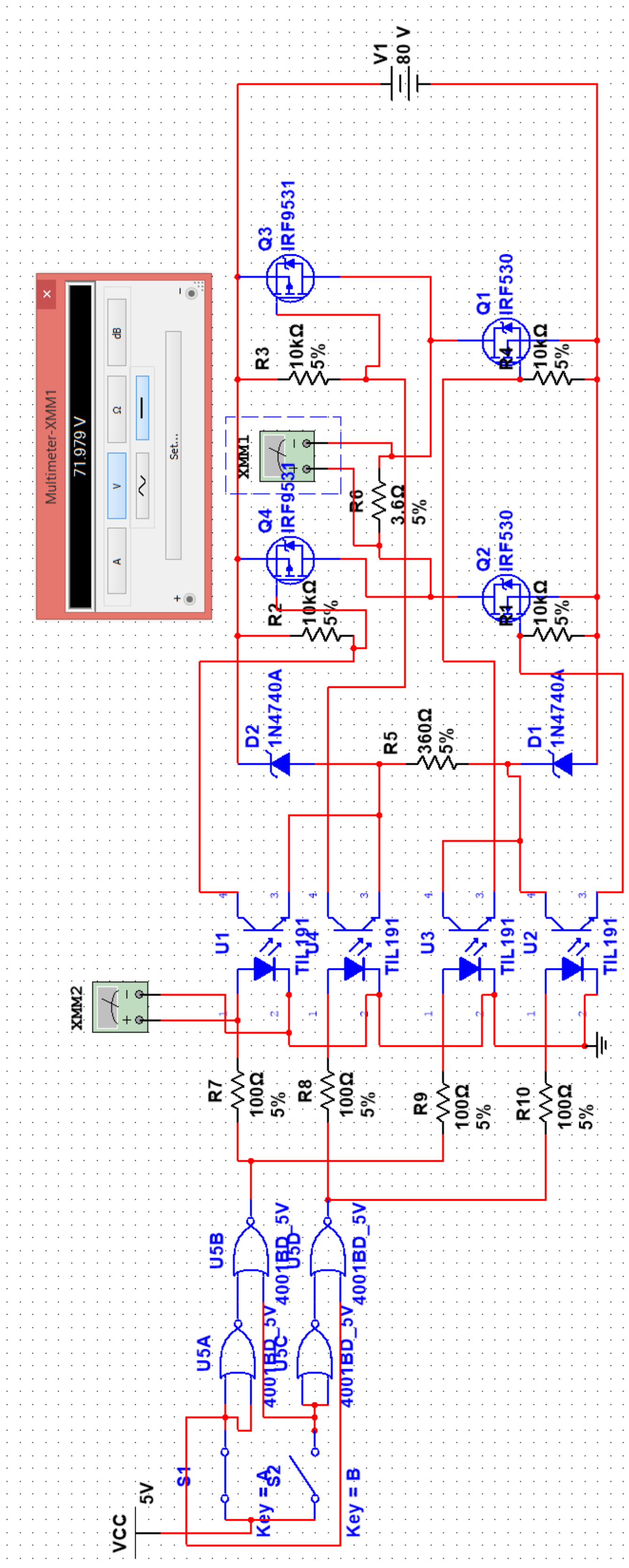
ii/ Multisim schematic



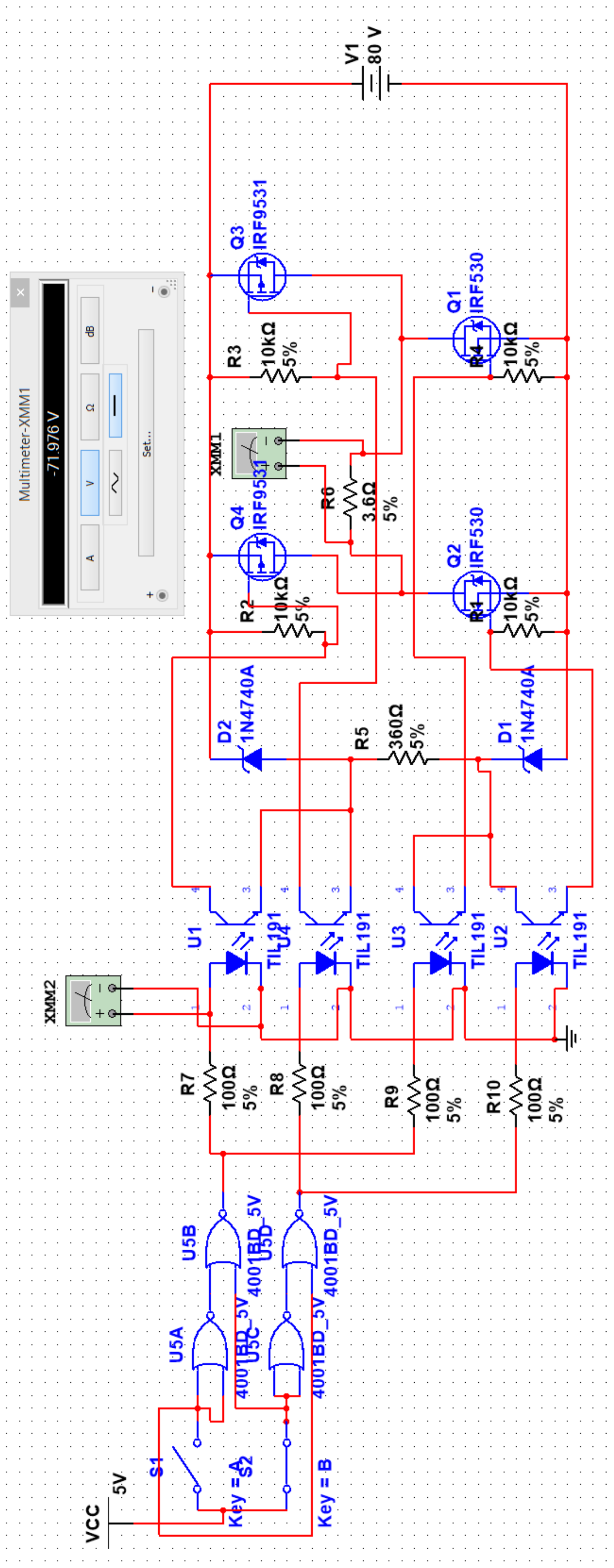
# Appendix D

## MULTISIM TESTING

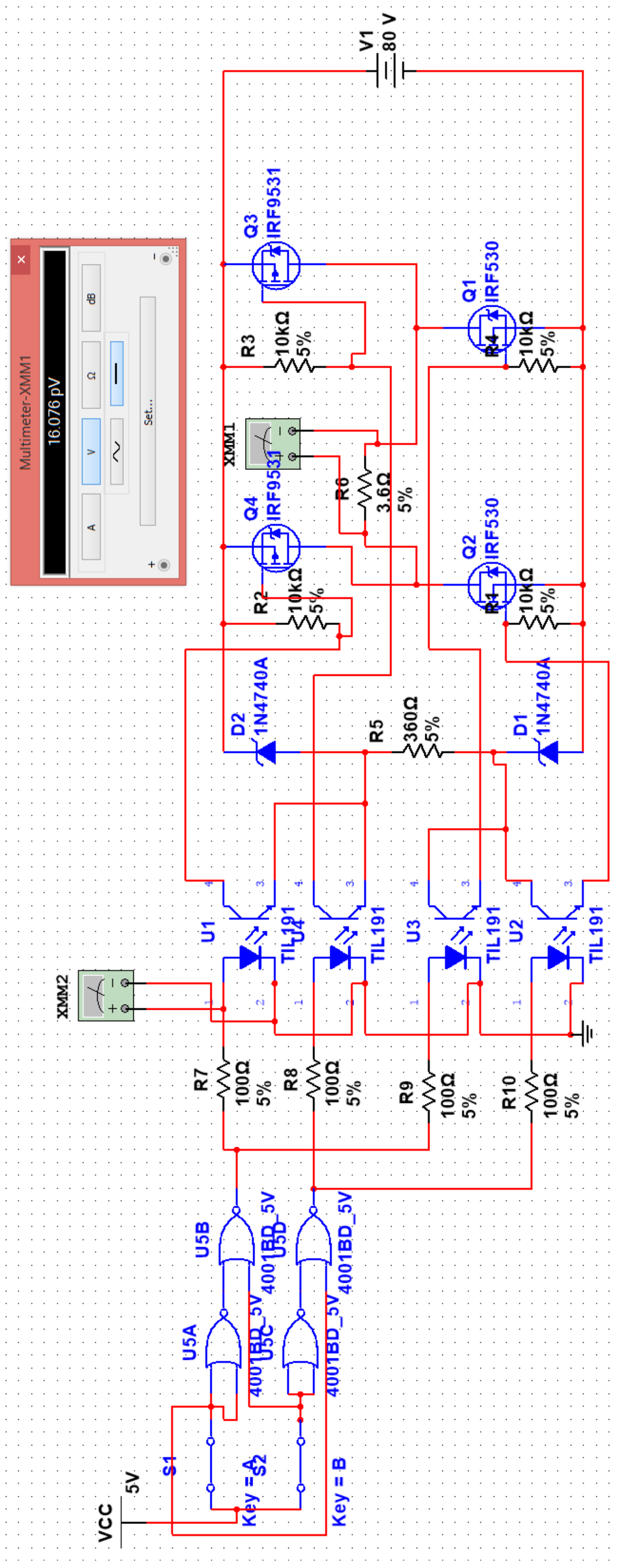
i/ direction 1 : Voltage is +71.979 VDC



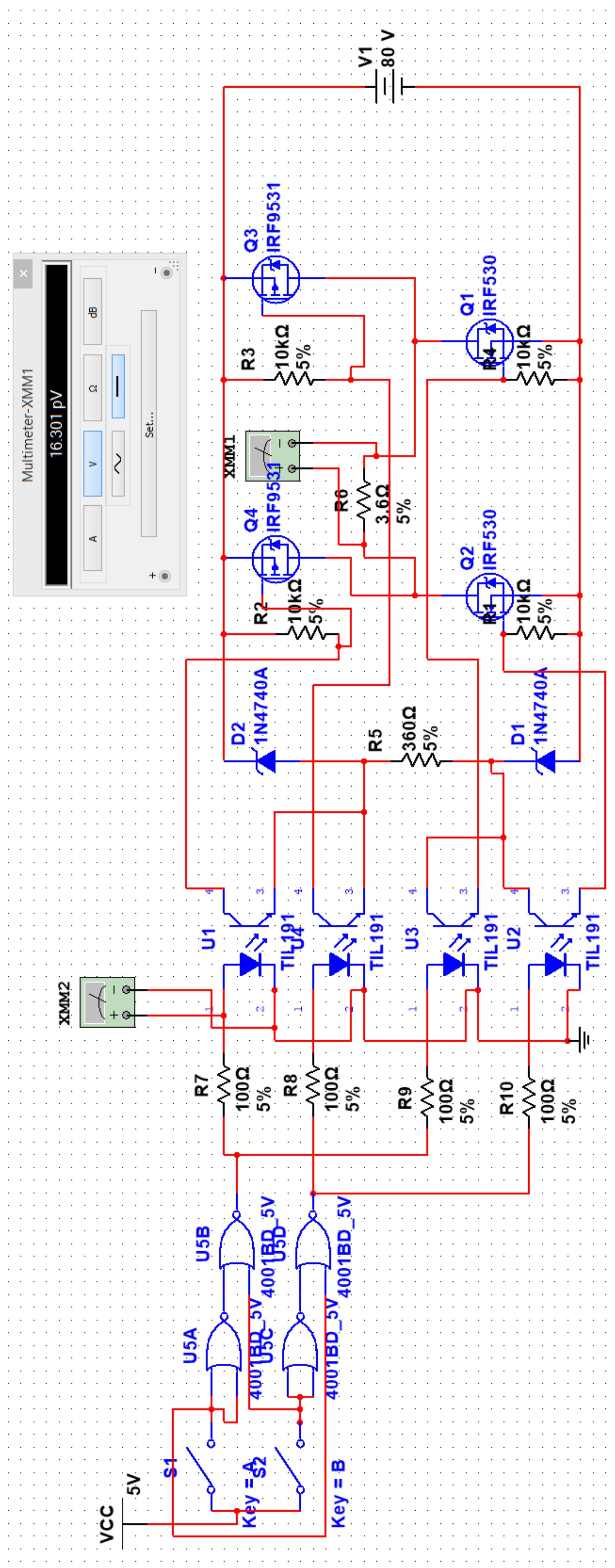
ii/ direction 2 : Voltage is -71.976 VDC



iii/ both switches on :  about 16pV due to leakage


V1
80 V
Q3
IRF9531
Q1
IRF530
R3
10kΩ
5%
R4
10kΩ
5%
XMM1
Multimeter-XMM1
16.076 pV
A
V
Ω
dB
Set...
R6
3.6Ω
5%
Q4
IRF9531
Q2
IRF530
R2
10kΩ
5%
R1
10kΩ
5%
D2
1N4740A
R5
360Ω
5%
D1
1N4740A
U1
TIL191
U4
TIL191
U3
TIL191
U2
TIL191
XMM2
R7
100Ω
5%
R8
100Ω
5%
R9
100Ω
5%
R10
100Ω
5%
U5B
4001BD_5V
U5D
4001BD_5V
U5A
4001BD_5V
U5C
4001BD_5V
S1
Key = A
S2
Key = B
VCC
5V

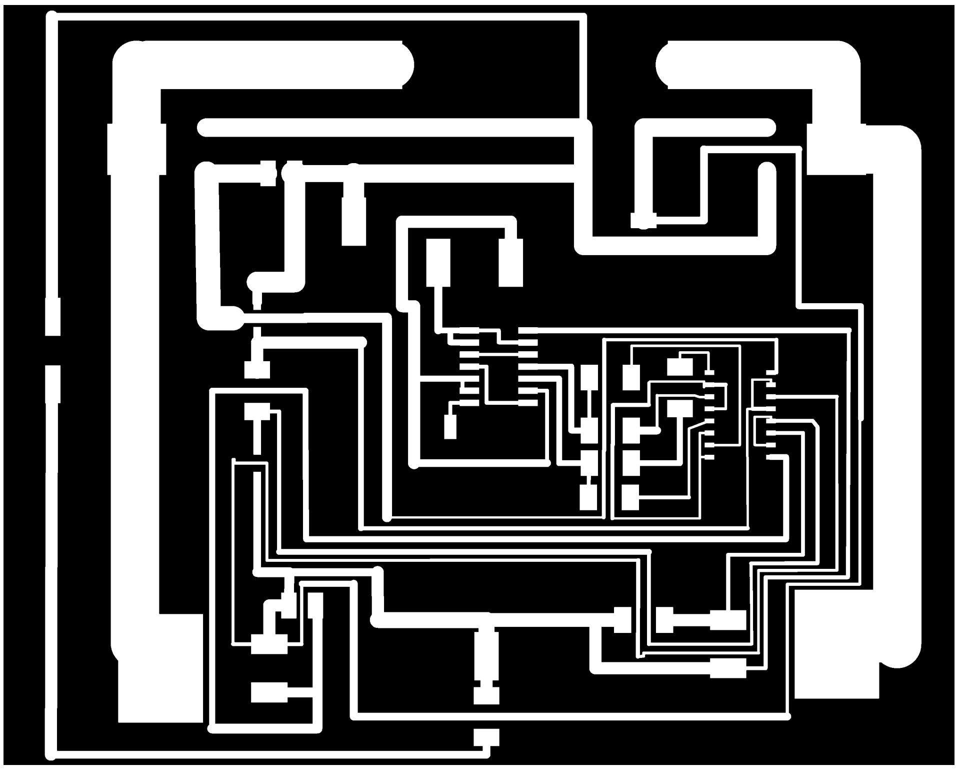
iv/ both switches off , voltage is about 16pV due to leakage



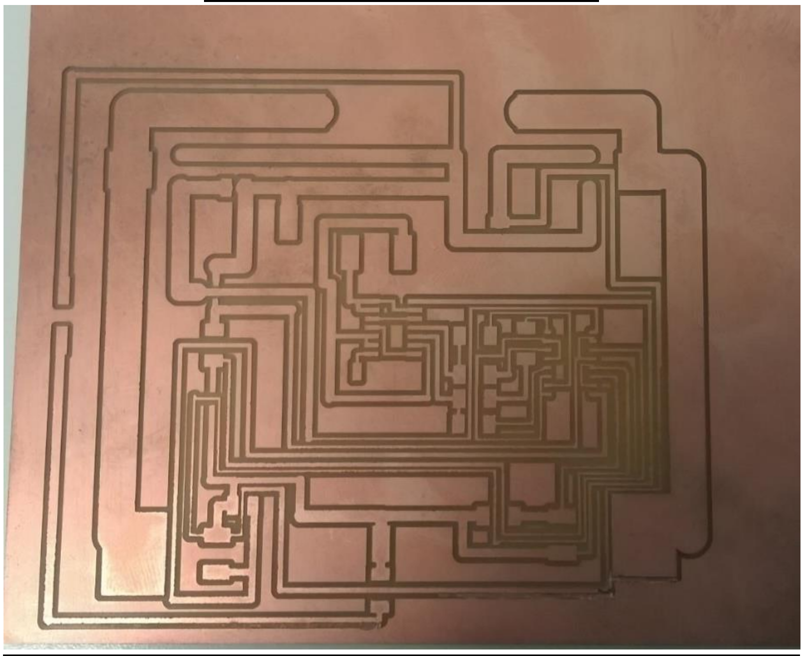
# Appendix E

## PCB Board

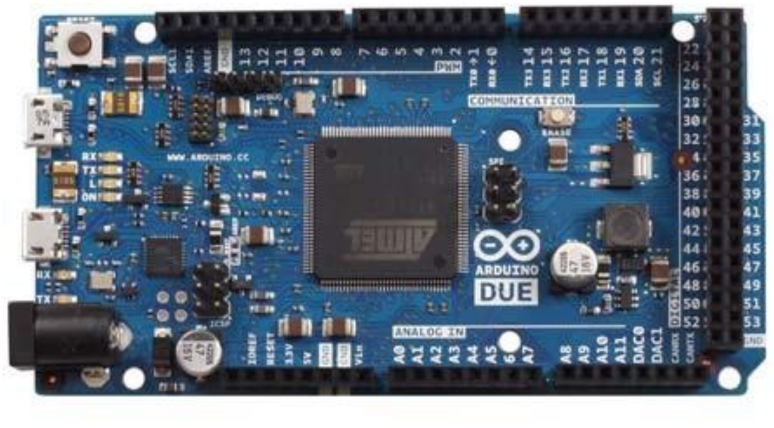
### EAGLE board



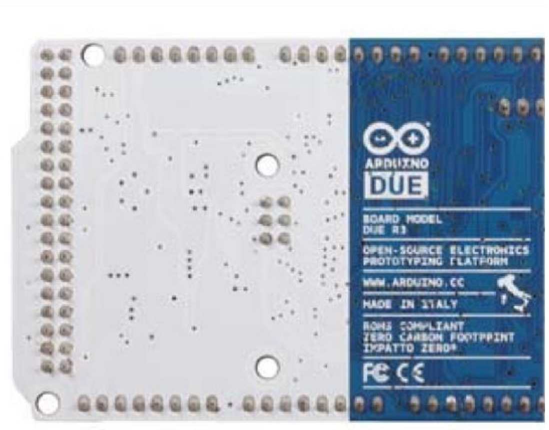
### Picture of milled board



# Appendix F
## Arduino Due



Arduino Due Front



Arduino Due Back

Overview
The Arduino Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU (datasheet). It is the first Arduino board based on a 32-bit ARM core microcontroller. It has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analogue inputs, 4 UARTs (hardware serial ports), a 84 MHz clock, an USB OTG capable connection, 2 DAC (digital to analogue), 2 TWI, a power jack, an SPI header, a JTAG header, a reset button and an erase button.

Warning: Unlike other Arduino boards, the Arduino Due board runs at 3.3V. The maximum voltage that the I/O pins can tolerate is 3.3V. Providing higher voltages, like 5V to an I/O pin could damage the board.
The board contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Due is compatible with all Arduino shields that work at 3.3V and are compliant with the 1.0 Arduino pinout.
The Due follows the 1.0 pinout:
- TWI: SDA and SCL pins that are near to the AREF pin.
- The IOREF pin which allows an attached shield with the proper configuration to adapt to the voltage provided by the board. This enables shield compatibility with a 3.3V board like the Due and AVR-based boards which operate at 5V.

ARM Core benefits
The Due has a 32-bit ARM core that can outperform typical 8-bit microcontroller boards. The most significant differences are:
- A 32-bit core, that allows operations on 4 bytes wide data within a single CPU clock. (for more information look int type page).
- CPU Clock at 84Mhz.
- 96 KBytes of SRAM.
- 512 KBytes of Flash memory for code. + a DMA controller, that can relieve the CPU from doing memory intensive tasks.

Schematic, Reference Design & Pin Mapping

Summary

| | |
|---|---|
| Microcontroller | AT91SAM3X8E |
| Operating Voltage | 3.3V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limits) | 6-20V |
| Digital I/O Pins | 54 (of which 12 provide PWM output) |
| Analog Input Pins | 12 |
| Analog Outputs Pins | 2 (DAC) |
| Total DC Output Current on all I/O lines | 130 mA |
| DC Current for 3.3V Pin | 800 mA |
| DC Current for 5V Pin | 800 mA |
| Flash Memory | 512 KB all available for the user applications |
| SRAM | 96 KB (two banks: 64KB and 32KB) |
| Clock Speed | 84 MHz |

Power

The Arduino Due can be powered via the USB connector or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm centre-positive plug into the board's power jack. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or if supplying voltage via the power jack, access it through this pin.
- **5V.** This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- **3.3V.** A 3.3-volt supply generated by the on-board regulator. Maximum current draw is 800 mA. This regulator also provides the power supply to the SAM3X microcontroller.
- **GND.** Ground pins.
- **IOREF.** This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

Memory

The SAM3X has 512 KB (2 blocks of 256 KB) of flash memory for storing code. The bootloader is pre-burned in factory from Atmel and is stored in a dedicated ROM memory. The available SRAM is 96 KB in two contiguous bank of 64 KB and 32 KB. All the available memory (Flash, RAM and ROM) can be accessed directly as a flat addressing space.

It is possible to erase the Flash memory of the SAM3X with the on-board erase button. This will remove the currently loaded sketch from the MCU. To erase, press and hold the Erase button for a few seconds while the board is powered.

Input and Output

- **Digital I/O: pins from 0 to 53**

  Each of the 54 digital pins on the Due can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 3.3 volts. Each pin can provide (source) a current of 3 mA or 15 mA, depending on the pin, or receive (sink) a current of 6 mA or 9 mA, depending on the pin. They also have an internal pull-up resistor (disconnected by default) of 100 KOhm. In addition, some pins have specialized functions:
- **Serial: 0 (RX) and 1 (TX)** + **Serial 1: 19 (RX) and 18 (TX)** + **Serial 2: 17 (RX) and 16 (TX)** + **Serial 3: 15 (RX) and 14 (TX)**

  Used to receive (RX) and transmit (TX) TTL serial data (with 3.3 V level). Pins 0 and 1 are connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.
- **PWM: Pins 2 to 13**

  Provide 8-bit PWM output with the analogWrite() function. the resolution of the PWM can be changed with the analogWriteResolution() function.
- **SPI: SPI header** (ICSP header on other Arduino boards)

  These pins support SPI communication using the SPI library. The SPI pins are broken out on the central 6-pin header, which is physically compatible with the Uno, Leonardo and Mega2560. The SPI header can be used only to communicate with other SPI devices, not for programming the SAM3X with the In-Circuit-Serial-Programming technique. The SPI of the Due has also advanced features that can be used with the Extended SPI methods for Due.
- **CAN: CANRX and CANTX**

  These pins support the CAN communication protocol but are not not yet supported by Arduino APIs.
- **"L" LED: 13**

  There is a built-in LED connected to digital pin 13. When the pin is HIGH, the LED is on, when the pin is LOW, it's off. It is also possible to dim the LED because the digital pin 13 is also a PWM output.
- **TWI 1: 20 (SDA) and 21 (SCL)**
- **TWI 2: SDA1 and SCL1.**

  Support TWI communication using the Wire library.
- **Analog Inputs: pins from A0 to A11**

  The Due has 12 analogue inputs, each of which can provide 12 bits of resolution (i.e. 4096 different values). By default, the resolution of the readings is set at 10 bits, for compatibility with other Arduino boards. It is possible to change the resolution of the ADC with analogReadResolution(). The Due’s analogue inputs pins measure from ground to a maximum value of 3.3V. Applying more than 3.3V on the Due’s pins will damage the SAM3X chip. The analogReference() function is ignored on the Due.

The AREF pin is connected to the SAM3X analogue reference pin through a resistor bridge. To use the AREF pin, resistor BR1 must be desoldered from the PCB.

- **DAC1 and DAC2**

  These pins provides true analog outputs with 12-bits resolution (4096 levels) with the analogWrite() function.

  These pins can be used to create an audio output using the Audio library.

Other pins on the board:

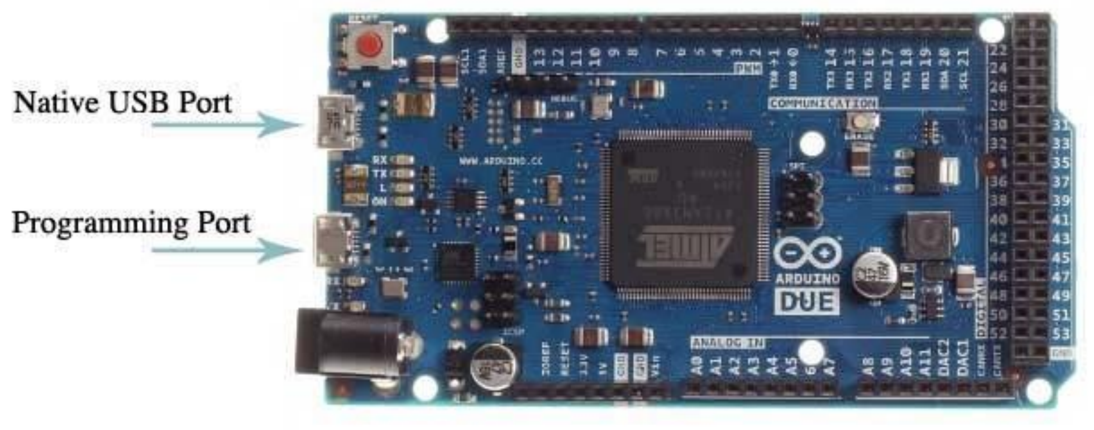
- **AREF**

  Reference voltage for the analogue inputs. Used with analogReference().
- **Reset**

  Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication
The Arduino Due has a number of facilities for communicating with a computer, another Arduino or other microcontrollers, and different devices like phones, tablets, cameras and so on. The SAM3X provides one hardware UART and three hardware USARTs for TTL (3.3V) serial communication.
The Programming port is connected to an ATmega16U2, which provides a virtual COM port to software on a connected computer (To recognize the device, Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically.). The 16U2 is also connected to the SAM3X hardware UART. Serial on pins RX0 and TX0 provides Serial-to-USB communication for programming the board through the ATmega16U2 microcontroller. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega16U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
The Native USB port is connected to the SAM3X. It allows for serial (CDC) communication over USB. This provides a serial connection to the Serial Monitor or other applications on your computer
The Native USB port can also act as a USB host for connected peripherals such as mice, keyboards, and smartphones. The SAM3X also supports TWI and SPI communication. The Arduino software includes a Wire library to simplify use of the TWI bus; For SPI communication, use the SPI library.

Programming
The Arduino Due can be programmed with the Arduino software. Uploading sketches to the SAM3X is different than the AVR microcontrollers found in other Arduino boards because the flash memory needs to be erased before being re-programmed. Upload to the chip is managed by ROM on the SAM3X, which is run only when the chip's flash memory is empty.



Either of the USB ports can be used for programming the board, though it is recommended to use the Programming port due to the way the erasing of the chip is handled :

- Programming port: To use this port, select "Arduino Due (Programming Port)" as your board in the Arduino IDE. Connect the Due's programming port (the one closest to the DC power jack) to your computer. The programming port uses the 16U2 as a USB-to-serial chip connected to the first UART of the SAM3X (RX0 and TX0). The 16U2 has two pins connected to the Reset and Erase pins of the SAM3X. Opening and closing the Programming port connected at 1200bps triggers a “hard erase” procedure of the SAM3X chip, activating the Erase and Reset pins on the SAM3X before communicating with the UART. This is the recommended port for programming the Due. It is more reliable than the "soft erase" that occurs on the Native port, and it should work even if the main MCU has crashed.

- Native port: To use this port, select "Arduino Due (Native USB Port)" as your board in the Arduino IDE. The Native USB port is connected directly to the SAM3X. Connect the Due's Native USB port (the one closest to the reset button) to your computer. Opening and closing the Native port at 1200bps triggers a 'soft erase' procedure: the flash memory is erased and the board is restarted with the bootloader. If the MCU crashed for some reason it is likely that the soft erase procedure won't work as this procedure happens entirely in software on the SAM3X. Opening and closing the native port at a different baudrate will not reset the SAM3X.

Unlike other Arduino boards which use avrdude for uploading, the Due relies on bossac.
You can use the ISP header with an external programmer (overwriting the DFU bootloader).

USB Overcurrent Protection

The Arduino Due has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

# Appendix G
## RS-485 Transceivers

General Description

The MAX481, MAX483, MAX485, MAX487, and MAX1487 are low-power transceivers for RS-485 communication. Each part contains one driver and one receiver. The MAX483 and MAX487 feature reduced slew-rate drivers that minimize EMI and reduce reflections caused by improperly terminated cables, thus allowing error-free data transmission up to 250kbps. The driver slew rates of the MAX481, MAX485, and MAX1487 are not limited, allowing them to transmit up to 2.5Mbps.

These transceivers draw between 120µA and 500µA of supply current when unloaded or fully loaded with disabled drivers. Additionally, the MAX481, MAX483, and MAX487 have a low-current shutdown mode in which they consume only 0.1µA. All parts operate from a single 5V supply. Drivers are short-circuit current limited and are protected against excessive power dissipation by thermal shutdown circuitry that places the driver outputs into a high-impedance state. The receiver input has a fail-safe feature that guarantees a logic-high output if the input is open circuit.

The MAX487 and MAX1487 feature quarter-unit-load receiver input impedance, allowing up to 128 MAX487/ MAX1487 transceivers on the bus.

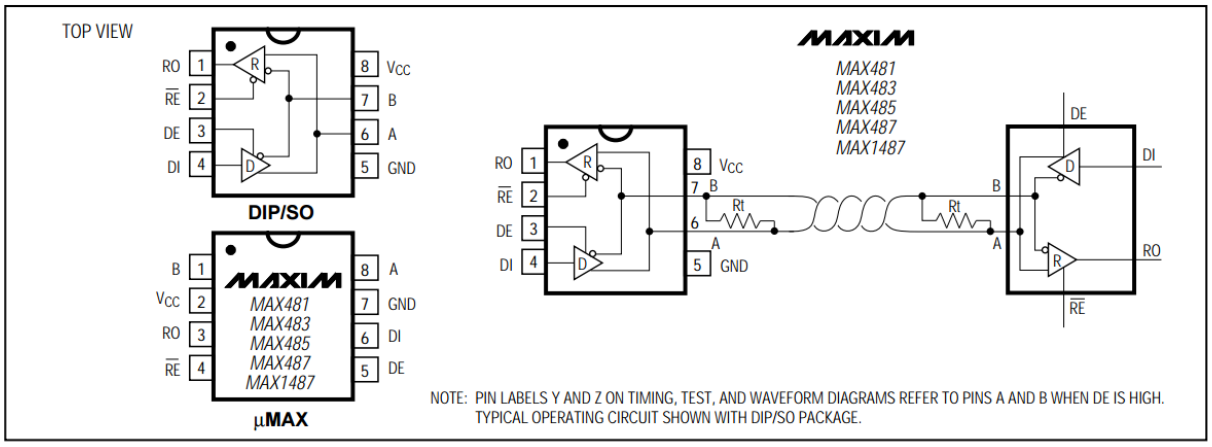
Features

♦ In µMAX Package: Smallest 8-Pin SO
♦ Slew-Rate Limited for Error-Free Data Transmission (MAX483/487)
♦ 0.1µA Low-Current Shutdown Mode (MAX481/483/487)
♦ Low Quiescent Current: 120µA (MAX483/487) 230µA (MAX1487) 300µA (MAX481/485)
♦ -7V to +12V Common-Mode Input Voltage Range
♦ Three-State Outputs
♦ 30ns Propagation Delays, 5ns Skew (MAX481/485/1487)
♦ Operate from a Single 5V Supply
♦ Allows up to 128 Transceivers on the Bus (MAX487/MAX1487)
♦ Current-Limiting and Thermal Shutdown for Driver Overload Protection

Applications

Low-Power RS-485 Transceivers
Level Translators
Transceivers for EMI-Sensitive Applications
Industrial-Control Local Area Networks

MAX481/MAX483/MAX485/MAX487/MAX1487 Pin Configuration and Typical Operating Circuit


TOP VIEW
RO
RE
DE
DI
VCC
B
A
GND
R
D
DIP/SO
MAXIM
MAX481
MAX483
MAX485
MAX487
MAX1487
µMAX
Rt
NOTE: PIN LABELS Y AND Z ON TIMING, TEST, AND WAVEFORM DIAGRAMS REFER TO PINS A AND B WHEN DE IS HIGH.
TYPICAL OPERATING CIRCUIT SHOWN WITH DIP/SO PACKAGE.

## Low-Power Shutdown Mode (MAX481/MAX483/MAX487)

A low-power shutdown mode is initiated by bringing both $\overline{RE}$ high and DE low. The devices will not shut down unless both the driver and receiver are disabled. In shutdown, the devices typically draw only 0.1µA of supply current.

$\overline{RE}$ and DE may be driven simultaneously; the parts are guaranteed not to enter shutdown if RE is high and DE is low for less than 50ns. If the inputs are in this state for at least 600ns, the parts are guaranteed to enter shutdown.

For the MAX481, MAX483, and MAX487, the tZH and tZL enable times assume the part was not in the low power shutdown state (the MAX485/MAX488–MAX491 and MAX1487 cannot be shut down). The tZH(SHDN) and tZL(SHDN) enable times assume the parts were shut down.

It takes the drivers and receivers longer to become enabled from the low-power shutdown state (tZH(SHDN), tZL(SHDN)) than from the operating mode (tZH, tZL). (The parts are in operating mode if RE and DE inputs equal a logical 0,1 or 1,1 or 0, 0.)

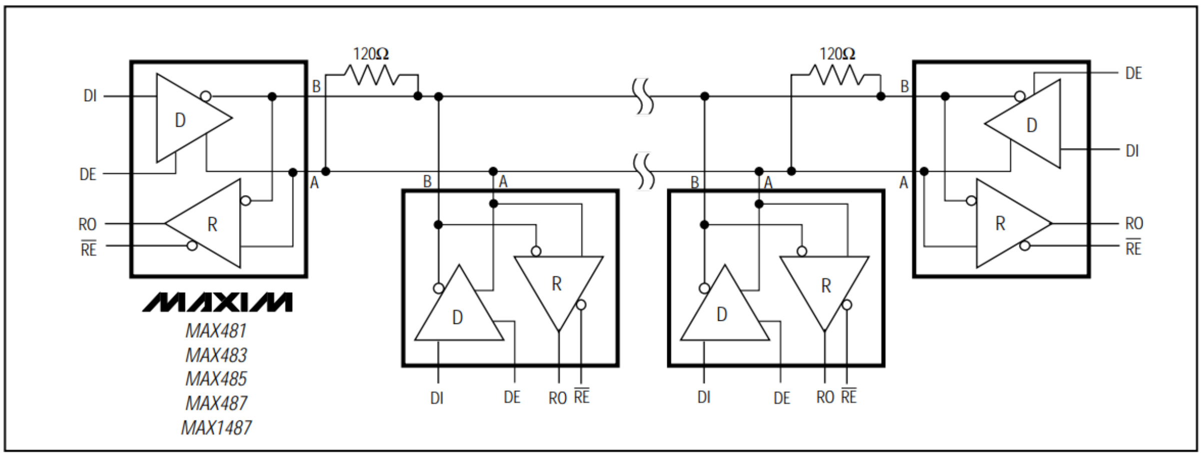
## Driver Output Protection

Excessive output current and power dissipation caused by faults or by bus contention are prevented by two mechanisms. A fold-back current limit on the output stage provides immediate protection against short circuits over the whole common-mode voltage range. In addition, a thermal shutdown circuit forces the driver outputs into a high-impedance state if the die temperature rises excessively.

## Propagation Delay

Many digital encoding schemes depend on the difference between the driver and receiver propagation delay times. The difference in receiver delay times, | tPLH - tPHL |, is typically under 13ns for the MAX481, MAX485, and MAX1487 and is typically less than 100ns for the MAX483 and MAX487. The driver skew times are typically 5ns (10ns max) for the MAX481, MAX485 and MAX1487, and are typically 100ns (800ns max) for the MAX483 and MAX487.

MAX481/MAX483/MAX485/MAX487/MAX1487 Typical Half-Duplex RS-485 Network



# Appendix H

References

Bibliography/References APPENDIX E

1. ST . 2000. Dual Full-Bridge Driver. [ONLINE] Available at: http://www.st.com/content/ccc/resource/technical/document/datasheet/82/cc/3f/39/0a/29/4d/f0/CD00000240.pdf/files/CD00000240.pdf/jcr:content/translations/en.CD00000240.pdf. [Accessed 19 November 2016].

2. Vishay. 2011. IRF530S. [ONLINE] Available at: http://www.vishay.com/docs/91019/91019.pdf. [Accessed 23 November 2016].

3. Vishay. 2009. SUD50P10-43L. [ONLINE] Available at: http://www.vishay.com/docs/73444/sud50p10.pdf. [Accessed 23 November 2016].

4. Maxim Integrated. 2014. MAX481 Datasheet. [ONLINE] Available at: http://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf. [Accessed 23 November 2016]

5. Arduino - ArduinoBoardDue . 2016. Arduino - ArduinoBoardDue . [ONLINE] Available at: https://www.arduino.cc/en/Main/ArduinoBoardDue. [Accessed 23 November 2016].