Documentation
for
Capacitance Based Liquid Level Sensor
MiniCLEAN

Krishna Mooroogen

Contents

1 Introduction

The use of capacitors for the application of sensors is not uncommon; the exploitation of the varying capacitance due to a dielectric change can be used as a robust level-sensing device. However, over long distances measurement cables can be the source of a large parasitic capacitance from which it is difficult to measure small changes. A relatively inexpensive solution comes from the use of the "Universal Transducer Interface" chip by Smartec, in conjuncture with a microcontroller; the parasitic capacitance in the measurement cables can be eliminated and displayed easily via computer.

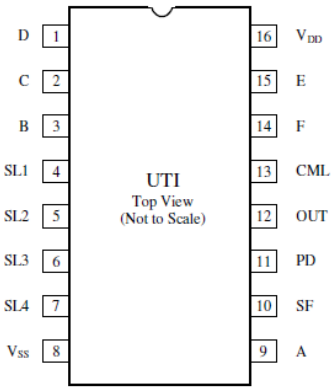
2 Hardware

2.1 Universal Transducer Interface*

The UTI is a complete analogue front end for low frequency measurement applications. The UTI outputs a microcontroller compatible period-modulated signal. Only a single reference element, of the same type as the sensor, is needed. It is capable of interfacing with capacitors of the range 0-2pF, 0-12pF, variable range of up to 300pF. These modes can be defined using the mode slectors found on pins 4-7.

2.1.1 Features:

- Measurement of multiple sensor elements
- Single 2.9V-5.5V power supply, current consumption below 2.5mA
- Resolution ans lineaerity up to 14 bits and 13 bits
- Continous auto-calibration of offset and gain
- -compatible signal
- Tri-state output
- Typical measurment time 10ms or 100ms
- 2/3/4 wire measurement available for almost all measurements
- AC excitation voltage signal for all sensor elements
- Suppression of 50/60 Hz interference
- Power down mode
- Operating temperature range for DIL and SO -40 degress celsius to 85 degress celsius
- Operating temperature range for bare die -40 degress celsius to 180 degress celsius



16-pins DIL

Name	Function of the pin
V_{DD}, V_{ss}	Power supply
A, B, C, D, E, F	Sensor connections
SEL1..SEL4	Mode selection (see Table 2)
OUT	Output
SF	Slow/fast mode selection
CML	CMUX02/CMUX12 mode selection
PD	Power Down (tri-state)

SEL 1	SEL 2	SEL 3	SEL 4	Mode	No. of Phases	Name	Mode No.
0	0	0	0	5 Capacitors, 0-2pF	5	C25	0
0	0	0	1	3 Capacitors, 0-2pF	3	C23	1
0	0	1	0	5 Capacitors, 0-12pF	5	C12	2
0	0	1	1	Capacitors, 0-2pF, external MUX CML=0 Capacitors, 0-12pF, external MUX CML=1	-	CMUX	3
0	1	0	0	3 Capacitors, variable range to 300pF	3	C300	4

Figure 1: Pin Map: Pin Identities: Mode Selector Lines

2.1.2 Measurement:

The three-signal technique is a technique to eliminate the effects of unknown offset and unknown gain in a linear system. In order to apply this technique, in addition to the measurement of the sensor signal, two reference signals are required to be measured in an identical way. Suppose a system has a linear transfer function of,

$$M_i = KE_i + M_{off}. \tag{1}$$

The measured three signals are,

$$M_{off} = M_{off} \tag{2}$$

$$M_{ref} = KE_{ref} + M_{off} \tag{3}$$

$$M_x = KE_x + M_{off} \tag{4}$$

When the system is linear, then in this ratio the influence of the unknown offset M_{off} and the unknown gain K of the measurement system is eliminated. This technique has been used in the UTI.

The implementation of the three-signal technique requires a memory: A microcontroller is used to perform the data storage and the calculations, and to digitize the period-modulated signals. Such a system combining a sensing element (sensor), a signal-processing circuit, such

as the UTI, and a microcontroller is called a microcontroller-based smart sensor system.

As an example, Figure 2 shows two complete cycles of the output signal from the UTI, each consisting of three phases.

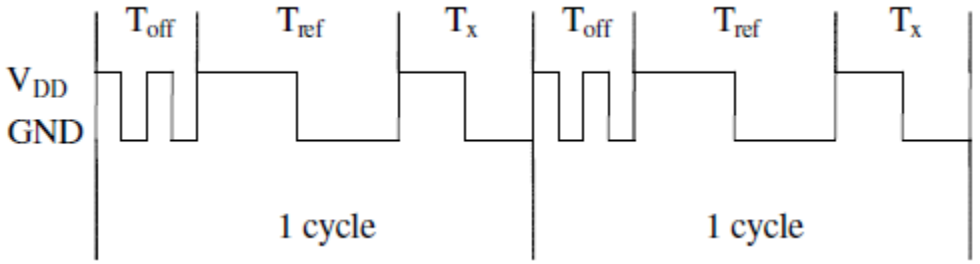


Figure 2: Time Periods of Complete Cycle

During the first phase T_{off}, the offset of the complete system is measured. During the second phase T_{ref}, the reference signal is measured and during the last phase T_x, the signal itself is measured. These phases are automatically controlled by the UTI itself. The duration of each phase is proportional to the signal which is measured during that phase. The duration of the three phases is given by:

$$T_{off} = NK_1C_0 \tag{5}$$

$$T_{ref} = NK_1(C_{ref} + C_0) \tag{6}$$

$$T_x = NK_1(C_x + C_0) \tag{7}$$

for capacitive measurement, where C_x is the sensor signal to be measured, C_{ref} or the reference signal, C_0 a constant and K_1 the gain. The factor N represents the number of internal oscillator periods in one phase. In slow mode, N = 1024 and in fast mode N = 128. The output signal of the UTI can be digitized by counting the number of internal clock cycles fitting in each phase. This results in the digital numbers N_{off}, N_{ref} and N_x. The ratio C_x/C_{ref} can now be calculated by the microcontroller:

$$M = \frac{N_x - N_{off}}{N_{ref} - N_{off}} = \frac{C_x}{C_{off}} \tag{8}$$

This ratio does not depend on the constant part and the gain. In fact, the system is calibrated for offset and gain. Even in the case of drift or other slow variations of offset and gain, these effects are eliminated. The three phases are time-multiplexed, as depicted in Figure 1. The offset phase is labeled, because it consists of two short intervals: the output frequency is temporarily doubled. This is recognized by the microcontroller, which guarantees that the correct calculation, as depicted in formula (8), is made. The number of phases in a complete cycle varies between 3 and 5, depending on the mode.

2.1.3 Resolution

The output signal of the UTI is digitised by the microcontroller. This sampling introduces quantizsation noise, which also limits the resolution. The quantisation noise of a measurement phase, as given by the relative standard deviation σ_q, amounts to:

$$\sigma_q = \frac{1}{\sqrt{6}} \frac{t_s}{T_{phase}} \tag{9}$$

where t_s is the sampling time and T_{phase} the phase duration. When the sampling time is 1 ms and the offset frequency is 50 kHz, the standard deviation of the offset phase is 12.5 bits in the fast mode and 15.5 bits in the slow mode. Further improvement of the resolution can be obtained by averaging over several values of M. When P values M_1 M_P are used to calculate $\bar{M}$, the value of σ_q decreases with a factor of $P^1/2$. Besides quantization noise, another limitation of the resolution is due to the thermal noise of the oscillator itself. In the fast mode, quantization noise is found to be the main noise source.

*Starred section taken from Smartec UTI Data Fact Sheet.

2.2 Arduino*

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It 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.

- Microcontroller - ATmega328
- Operating Voltage - 5V
- Input Voltage (recommended) - 7-12V
- Input Voltage (limits) - 6-20V
- Digital I/O Pins - 14 (of which 6 provide PWM output)
- Analog Input Pins - 6
- DC Current per I/O Pin - 40 mA
- DC Current for 3.3V Pin - 50 mA
- Flash Memory - 32 KB (ATmega328) of which 0.5 KB used by bootloader
- SRAM - 2 KB (ATmega328)
- EEPROM - 1 KB (ATmega328)
- Clock Speed - 16 MHz

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.

- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure
- from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally,
- some pins have specialized functionality:
- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- AREF. Reference voltage for the analog 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.

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). Communication over ethernet is also possible with the Ethernet shield addition, both serial and ethernet can be used to power the Arduino and communicate.
*Starred section taken from Arduino Website, http://arduino.cc/en/.

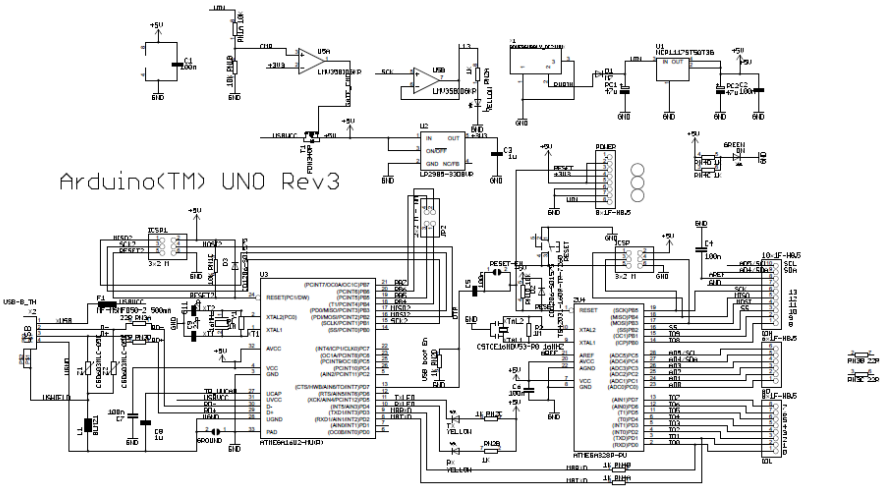


Figure 3: Arduino Uno Schematic

2.3 The Capacitor

The use of a parallel plate capacitor allows one to differ between dielectric mediums via the capacitance value; thus it is possible to differentiate between levels of liquid Argon/Neon throughout the detector by taking capacitance readings at set distances. The design of the test capacitor is such that it has an approximate capacitance of 1pF in normal air, with a dielectric constant of 1.00058, and approximately 1.3pF in the liquid Argon/Neon.

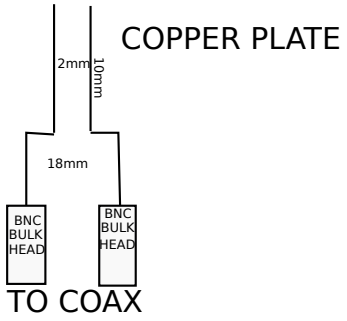
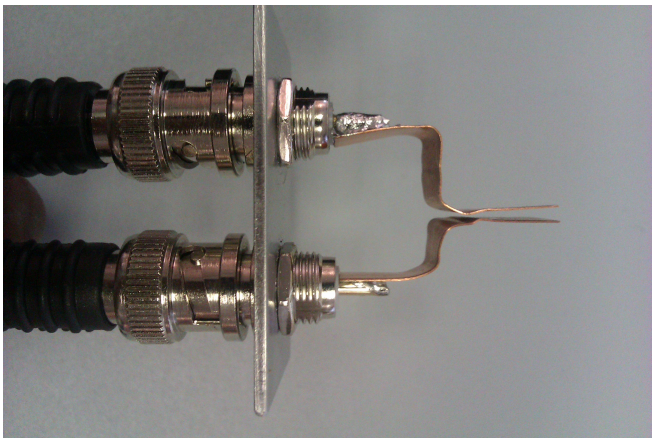
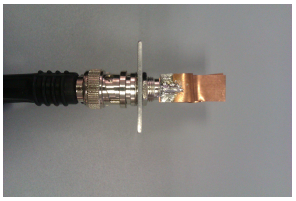


Figure 4: Capacitor design

The capacitor has a square overlapping area of 100mm^2 with the plates 1mm apart. Each plate (made form copper) is connected to a coax cable via a BNC bulkhead (Soldered to the copper plates) and held together by an aluminium bracket, which also acts to ground the pair. The two plates are curved into an"S"shape to bring them to their 1mm distance which would be normally obstructed by the size of the BNC bulkheads.





2.4 Circuitry

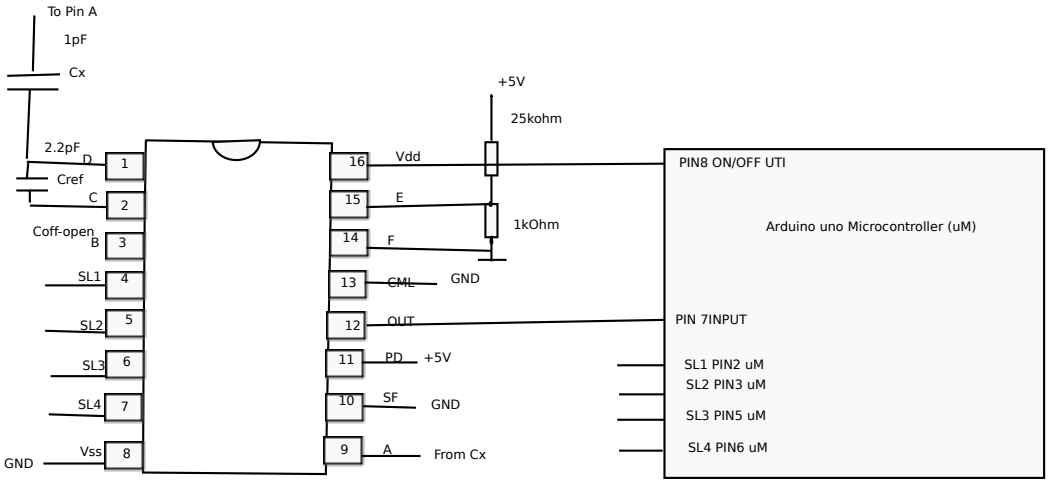


Figure 5: Circuit Layout

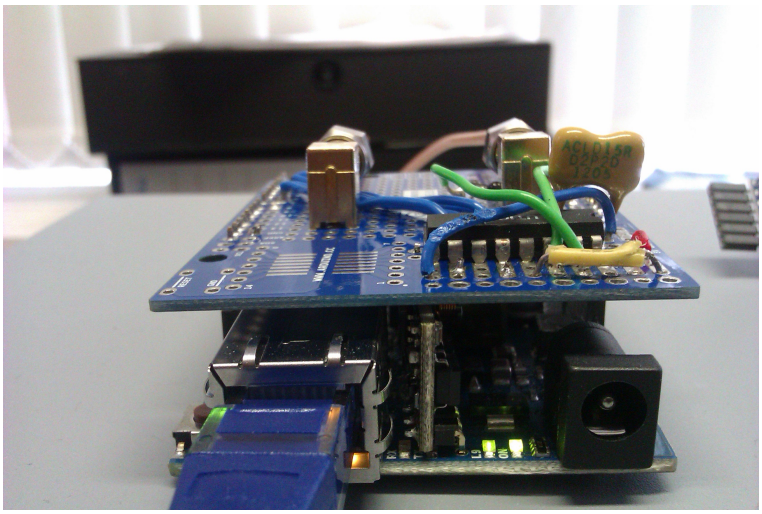
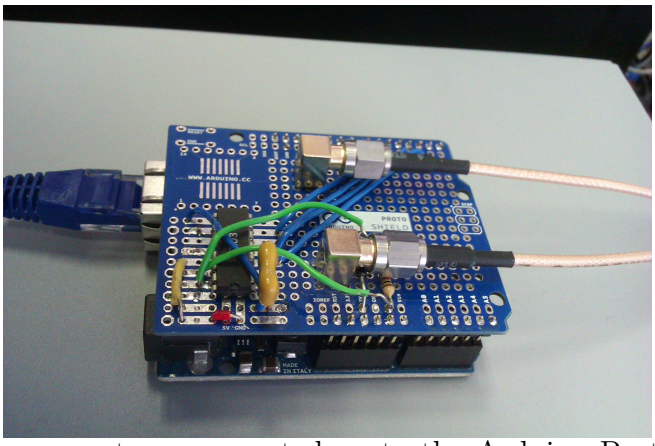
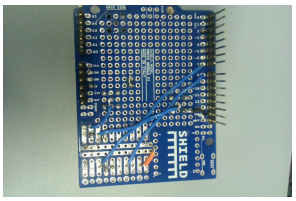


Figure 6: Arduino Prototype





All components are mounted on to the Arduino Prototype Shield for testing, a custom PCB design may be implemented to help further reduce parasitic capacitance within the circuit and reduce space limitations. Soldering of the connecting wires needed to be above and below the prototype board to reduce the overlapping of wires. The Arduino Prototype Shield is limited for testing as the work area is small, with very few tracks connected together, meaning some components have to be soldered together directly. The measured capacitor needs to be connected to the UTI using the shortest leads possible and therefore SMA connectors are used to extend cabling from the board as depicted. These are run through the front panel of the casing where they can be adapted to fit a variety of input capacitors making it versatile for alternative use. Ethernet bulkheads are used on the front panel as the Arduino Ethernet port itself will not sit flush and therefore a small extension cable is used inside.

The resistors are limited by the condition that the total time constant of all resistors and capacitors should be less than 500ns. This results in a combination of a 1Kohm and 25Kohm resistors used to control the voltage swing which effectively acts as timing factor to keep within working range of capacitance.

The Mode Selector lines are connected to the Arduino pins 2,3,5 and 6 which are denoted as outputs, they can be controlled to switch between High(up to 300pF) and Low(below 2pF) modes via the mode selector on the front panel. The appropriate pin is set “HIGH” by the microcontroller to send +5V to the pin turning that selector on (1).

Pin 7 is connected to the UTI Vdd pin as an om/off utility and finnally pin 8 is assigned as the input signal.

3 Software

3.1 Capacitance Reader

3.1.1 UTI Software Manual Help

- Mode selection. Select the mode to be used for the application. Pay attention to the CS-line and other settings needed for the UTI. None of the Sel-lines nor the CS-line may be left open.
- Number of periods. Each selected mode has a fixed number of periods (Nop), generated by the UTI itself The offset will come twice out of the UTI so define NoP + 1
- Define in the micro. NoP+1 locations and a counter.
- Wait for the first positive edge on the output of the UTI.
- When positive edge is detected start sampling the input until the next positive edge.
- On the next positive edge, move the counter value in location 1, set the counter to zero
- Sample until the next positive edge and move the value to location 2, set the counter to zero and start sampling again.
- Do the same procedure for NoP+1 locations. After NoP+1 locations, start again at location 1. It is advised to measure over more then one period of the UTI to achieve sufficient accuracy. At the end of period 1 (at a positive edge) add the counter value to the value already on that location.
- Repeat step 5 5 till 8 5 5 8 until a multiple of (8 8 NoP+1) periods are measured. Stop sampling. Now one has NoP+1 locations filled with values.
- Find the two locations with the lowest counts. These are the offset periods. From this offset location the other period definition can be found (fi. Tref and Tx). Apply the three-signal method and calculate by subtracting and dividing the value(s) one needs to know. An alternative way is to define NoP+1 counters and switch on each positive edge to the next counter.

3.1.2 Software Code

The following is a break down of the key operational areas of the capacitance reader, along with the details of how the three signal technique of the UTI has handled. Full details of the code with comments can be found in the appendix. All "s[" or "s]" refer to "curly brackets".

```
  void TurnOnUTI(void)
{
  digitalWrite(7,HIGH);
  delay(1000);
}

void TurnOffUTI(void)
{
  digitalWrite(7,LOW);
}


int ReadUTI(float *cap)
/*funtion readUTI, turns uti chip on, takes times using the
micros function(returns time in micro seconds that arduino
has been running) on the rising and falling edges of the 3 piece signal,
idetifies each part and assigns them accordingly as toff,
 tx and tref reffering to the UTI manual.
*/
{
  float capREF = 2.2;
  int b = 0; //define array intergers
  int j;
  int i;
  unsigned long width[34] =
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  //define array for 'time stamps' need 34 as we need 16 periods, as i=i+2
  unsigned long timep[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  //array for storing periods
  float toff = 0;
  float tref = 0;
  float tx = 0; //define periods
  float time1 = 0;
  float time2 = 0;
  float time = 0;
  int   input_pin = 8;

  for(i = 0;i < 33; i += 2) //for these iterations moving in
  twos, so that in each turn two slots are filled.
```

```
    {
      while (digitalRead(input_pin) == LOW)
{
 width[i]=micros(); //takes time at rising edge
 (will continuley take time stamps while high or low,
 will overwrite with the last value making it the rising
 or fallign edge) time program has been running at that point.
        }

      while  (digitalRead(input_pin) == HIGH)
{
 width[i+1]=micros(); //takes time at falling edge, first
 reading will be unusable from this method (changed from micros to test)
}
    }

  if (width[0] > width[32])//over flow check
    return(0);


  j = 0;
  for(i = 0; i < 32; i += 2)
    {
      timep[j]=width[i+2]-width[i]; //calculates the time periods,
      time the board has been running minus previous time,
      stores in array timep
      j++;
    }


  for(i = 0; i < 8; i++)
    {
      if ( (0.95*timep[i+1])<=timep[i]  &&
           timep[i]<=(1.05*timep[i+1])  &&
           timep[i]<timep[i+2] &&
           timep[i]<timep[i+3] &&
           timep[i+1]<timep[i+2] &&
           timep[i+1]<timep[i+3])
            //selects the starting position by defining the first peroid
            as being the first two smallest periods which lie within 5% of ea
        {
 //picking the offset periods
 toff=timep[i]+timep[i+1];
 tref=timep[i+2];
 tx=timep[i+3];
```

```
 *cap=((tx-toff)/(tref-toff))*capREF; //UTI 3 signal calculation
 return(1);
}
    }
  return(0);
}
```

Starting from the first line, the command sends a +5V(HIGH) signal to pin 7 which is connected to the UTI Vdd pin, thus turning the chip on. A "for" loop of 32 iterations, each sweep takes two "time stamps" one at the rising edge of the outputted signal and one at the falling edge. This is done with the "micros" command, which returns the time in microseconds that the Arduino has been running. The chip is then set to "LOW" or off. Once the "time stamps" have been stored in an array (width), one is able to obtain the widths of each of the four outputted signals, 32 iterations gives 16 widths which ensures that the entire output signal is captured, these are then stored in another array (timep). In order to apply the three-signal technique we must first establish which signal peak is the offset, reference and measuring capacitance. As suggested from the UTI manual the key to recognising the peaks is from the fact that the offset signal is the smallest and appears as a doublet. Using an "if" loop it was possible to establish a condition for which the offset signal can identified; the doublet has been defined as being within 5% if each others value to emphasise they should be near enough equal. Furthermore the offset signal and the successive signal must both be smaller than the last signal but the offset must be the smallest, for all cases that this is true then "toff" (the offset time period) can be denoted as the sum of the first two pulses. With all four pulses established the three-signal method could be applied.

3.2 Server

Follows is a section from the completed code showing a string compare method for web server communication, from where a front end web browser can be used to control the Arduino.

```
void loop()
{
  // listen for incoming clients

  boolean done_read = false;
  char mode_string[16];
  long capacitance;
  float capa;

  EthernetClient client = server.available();

  if (client)
    {
      char c = client.read();

      if (c == '\n')
{
 ret_string[buff_pos]='\0';
 done_read = true;
 buff_pos=0;
}
      else
{
 ret_string[buff_pos]=c;
 buff_pos++;
}

      if (done_read)
{
 if (strncmp(ret_string,"Get", 3) == 0)
   {
              Serial.print("Got to Get Cap!  \n");
              capa=(read_cap());
     capacitance=(long)(capa*1000000.0);
     client.println(capacitance);//return average capacitance value
     Serial.println(capacitance);
              Serial.println(capa);
   }

 else if (strncmp(ret_string,"SetMode High", 12) == 0)
   {
```

```
                Serial.print("Set High Mode \n");
      mode_select=High;//string compare mode selector
      set_UTI_mode();//refers back to mode slector function
    }

  else if (strncmp(ret_string,"SetMode Low", 11) == 0)
    {
                Serial.print("Set Low Mode \n");
      mode_select=Low;
      set_UTI_mode();
    }
}
     }
}
```

Using the client.read function, the program listens for a client to contact the sever, when a command string is sent the characters are compared to a set phrase in the program (in this case"Getcap"), which is assigned to call on the capacitance function. Once the capacitance read has finished the average value is written to the client, conversion from a "float" to an "int" is necessary for the client to read the returned output.

4 Performance

After extensive testing over several days two key features were observed; fluctuations in the capacitance are of the order 10-15 Farads in low mode (below 2pF), furthermore succeeding three hours of normal operation the capacitance level will drop significantly every 8 minutes before returning to its' normal value. It is thought that this may be the result of the timer clock overflowing when calculating pulse time periods. This phenomenon appears to be resolved by implementing a condition by which the calculation does not proceed if the clock over fills, i.e. if the time stored in array slot 0 is larger than the time in the last slot.

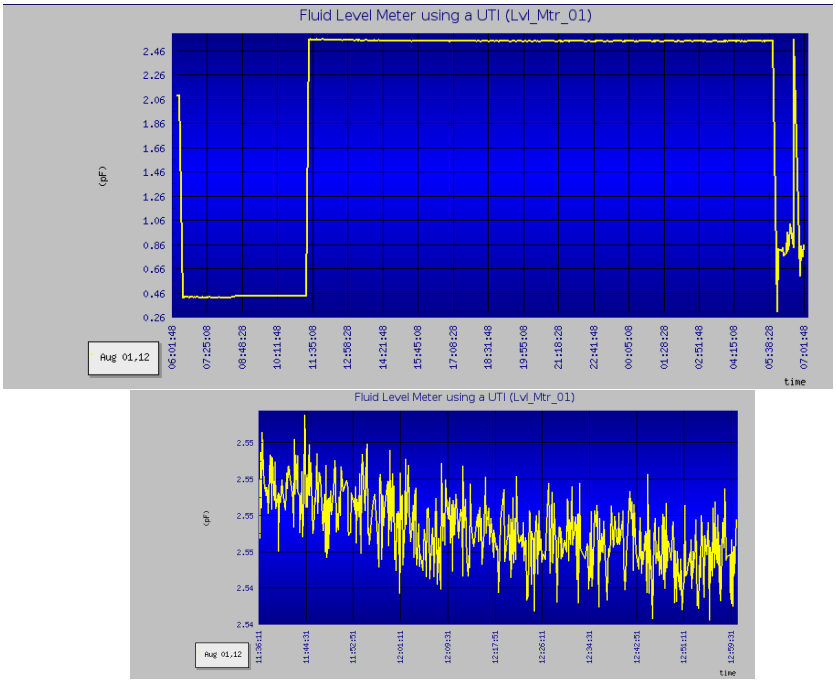
Ongoing evaluation of the High mode setting revealed an averagely lower capacitance, 0.96pF, than in Low mode with fluctuations in the Pico Farad region. The values are expected to be different as the High mode setting is out of dynamic range (smaller than 2pF) for this capacitor. Both modes appear to be operating as expected.

The value of the returned capacitance is approximately 2pF, though the calculated value was expected to be close to 1pF, if we subtract the parasitic capacitance from the board (that is, the capacitance of the circuit without the capacitor attached) we obtain a value close to 1.4pF. Due to the fact the value is completely derived from some crudely made geometry, the value stands to reason.

After some rudimentary calculations of temperature dependence on dielectric properties and thermal expansion of the copper, it was concluded that these affects (from the capacitor heating) would have little contribution to the overall noise, it is therefore expected that any noise observed is from circuitry on the board.

Below is a brief summary of plots for each of the modes with and without a capacitor attached, plots are of capacitance against time.

4.0.1 Low Mode



The figure beneath is a zoomed in image of the fluctuations in the readout, it can be seen that noise on the value is minimal. The spikes on either side of the flat region is caused by alterations being made to the circuit while readings are being taken.

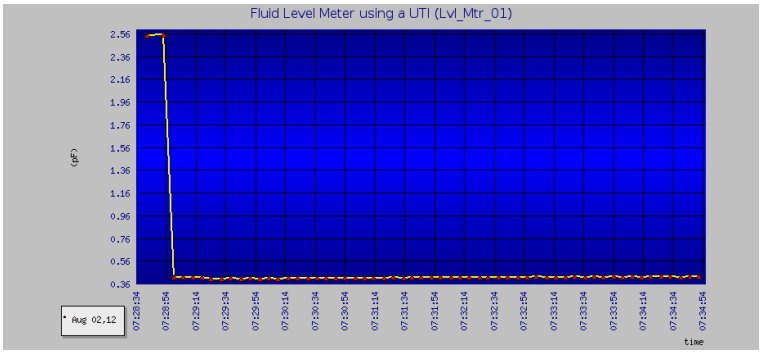
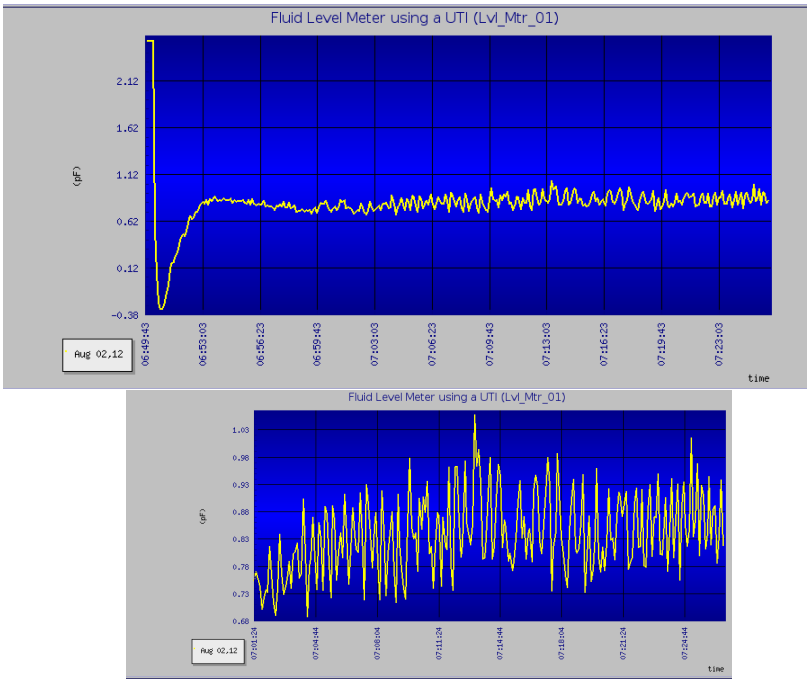


Figure 7: Low mode Without Capacitor, Parasitic Capacitance in Circuit Board

The capacitance readout here gives an indication of the parasitic capacitance in the circuit board, the average was recorded at approximately 0.4pF.

4.0.2 High Mode



The results in High mode are as expected susceptive to noise, due to operation out of dynamic range for this capacitor. The average value reads at 0.96pF and the noise on that value in the Pico Farad region.

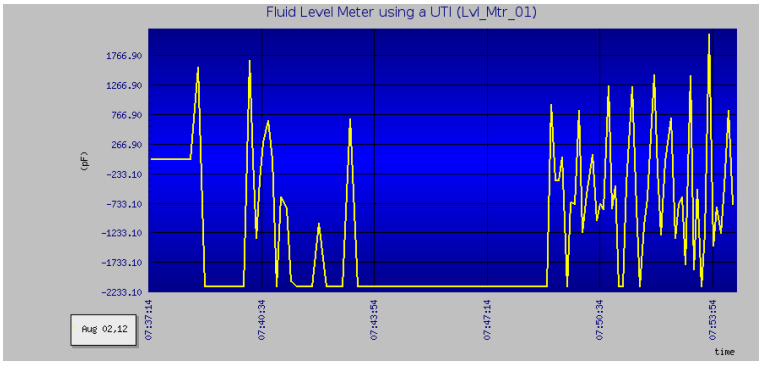


Figure 8: High Mode no Capacitor, High Mode Parasitic Capacitance

The parasitic capacitance associated with the circuit board in High mode fluctuates rapidly within hundreds to negative thousands of Pico Farads; again this is expected due to the dynamic range.

In order to thoroughly test the High mode setting a larger capacitor is needed, due to logistics this cannot be a parallel plate capacitor.

Capacitor w/wo	High Mode	Low Mode
With Capacitor	0.96pF	2pF
Without Capacitor	-1000-300(n/a)pF	0.4pF

Table 1: Capacitance Mode Summary

A Finalised Code

```
/*
  Web Server
  Krishna Mooroogen
  21st June 2012
  web server for capacitance liquid level sensor
  MINIclean
  Supervisor Dr James Nikkel
  controller to be able to talk via ethernet to a master program
*/

#include <SPI.h>
#include <Ethernet.h>//calls ethernet library


// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xE7, 0xD };
byte ip[] = {192,168,1,177};// sets ip and mac address

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(5000);//access port

char ret_string[128];
int    buff_pos = 0;

int Low = 0;  //definig global variable for the mode selector
int High = 1;
int mode_select = High;// This is the default mode behaviour


void set_UTI_mode()
/*mode selector function, <2pf low mode or <300pf high mode,
both 3 capacitors modes, enables ouput pins to control
weather the selectors are on or off refer to UTI manual for
```

```
other modes and selector pins
*/

{
  if (mode_select == Low)//if set to low then, mode low all
  other cases mode is high, i.e default
    {
      digitalWrite(2,LOW);
      digitalWrite(3,LOW);
      digitalWrite(5,LOW);
      digitalWrite(6,HIGH);
    }

  else

    {
      digitalWrite(2,LOW);
      digitalWrite(3,HIGH);
      digitalWrite(5,LOW);
      digitalWrite(6,LOW);
    }
}


 void TurnOnUTI(void)
{
  digitalWrite(7,HIGH);
  delay(1000);
}

void TurnOffUTI(void)
{
  digitalWrite(7,LOW);
}


int ReadUTI(float *cap)
/*funtion readUTI, turns uti chip on, takes times using the micros
function(returns time in micro seconds that arduino has been running)
on the rising and falling edges of the 3 piece signal,
idetifies each part and assigns them accordingly as toff, tx and tref
 reffering to the UTI manual.
*/
{
  float capREF = 2.2;
```

```
  int b = 0; //define array intergers
  int j;
  int i;
  unsigned long width[34] =
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

   //define array for 'time stamps' need 34 as we need 16 periods, as i=i+2

  unsigned long timep[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

  //array for storing periods
  float toff = 0;
  float tref = 0;
  float tx = 0; //define periods
  float time1 = 0;
  float time2 = 0;
  float time = 0;
  int   input_pin = 8;

  for(i = 0;i < 33; i += 2) //for these iterations moving in twos,
  so that in each turn two slots are filled.
    {

      while (digitalRead(input_pin) == LOW)
{
 width[i]=micros();

 //takes time at rising edge (will continuley
 take time stamps while high or low, will overwrite with the last
 value making it the rising or fallign edge) time program has been
 running at that point.

       }


      while  (digitalRead(input_pin) == HIGH)
{
 width[i+1]=micros(); //takes time at falling edge, first reading
 will be unusable from this method (changed from micros to test)
}
    }

  if (width[0] > width[32])//over flow check
    return(0);
```

```
  j = 0;
  for(i = 0; i < 32; i += 2)
    {
      timep[j]=width[i+2]-width[i]; //calculates the time periods,
       time the board has been running minus previous time, stores
       in array timep
      j++;
    }


  for(i = 0; i < 8; i++)
    {
      if ( (0.95*timep[i+1])<=timep[i]  &&
           timep[i]<=(1.05*timep[i+1])  &&
           timep[i]<timep[i+2] &&
           timep[i]<timep[i+3] &&
           timep[i+1]<timep[i+2] &&
           timep[i+1]<timep[i+3])
            //selects the starting position by defining the first
            peroid as being the first two smallest periods which
            lie within 5% of each other
        {

 //picking the offset periods
 toff=timep[i]+timep[i+1];
 tref=timep[i+2];
 tx=timep[i+3];



 *cap=((tx-toff)/(tref-toff))*capREF; //UTI 3 signal calculation
 return(1);
}
    }
  return(0);
}


float read_cap()//fucntion readd cap, averages runs and returns value.
{
  //  Reads out and averages the capacitance value
  float cap = 0; //definine variables for later
  float cap_sum = 0;
  int   i = 0;
```

```
  int   max_averages = 5; //loop five times for an average capacitance
  int   actual_averages = 0;

  TurnOnUTI();
  for (i = 0; i < max_averages; i++)
    {
      if (ReadUTI(&cap))     //fucntion defined above
       {
          cap_sum += cap;
          actual_averages++;
          delay(10);          //wait periods to stop clogging
        }
    }
   TurnOffUTI();

  if (actual_averages > 0)
    return(cap_sum/actual_averages);//only divides by the number
    of succesful runs

  else
    return(0);
}


void setup() //function set initial set up
{
  pinMode (2, OUTPUT);   //set pins 2-6 as outputs, 6 is on and off pin
  pinMode (3, OUTPUT);
  pinMode (7, OUTPUT);
  pinMode (5, OUTPUT);
  pinMode (6, OUTPUT);
  pinMode (8, INPUT);     //input signal

  set_UTI_mode();  // Will set mode to default

  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.begin(9600);
}


void loop()
{
  // listen for incoming clients
```

```
  boolean done_read = false;
  char mode_string[16];
  long capacitance;
  float capa;

  EthernetClient client = server.available();

  if (client)
    {
      char c = client.read();

      if (c == '\n')
{
 ret_string[buff_pos]='\0';
 done_read = true;
 buff_pos=0;
}
      else
{
 ret_string[buff_pos]=c;
 buff_pos++;
}

      if (done_read)
{
 if (strncmp(ret_string,"Get", 3) == 0)
   {
             Serial.print("Got to Get Cap!  \n");
             capa=(read_cap());
     capacitance=(long)(capa*1000000.0);
     client.println(capacitance);//return average capacitance value
     Serial.println(capacitance);
             Serial.println(capa);
   }

 else if (strncmp(ret_string,"SetMode High", 12) == 0)
   {
             Serial.print("Set High Mode \n");
     mode_select=High;//string compare mode selector
     set_UTI_mode();//refers back to mode slector function
   }

 else if (strncmp(ret_string,"SetMode Low", 11) == 0)
   {
```

```
                Serial.print("Set Low Mode \n");
      mode_select=Low;
      set_UTI_mode();
    }
}
     }
}
```