A Sample Application of an Arduino Controlling Hardware, version 1.0

By R. G. Sparber

I found it helpful to see an actually application that used an Arduino before trying it myself. This article presents a system that talks with a modified Harbor Freight digital caliper connected to an Arduino. The Arduino drives a display.

For starters, see http://rick.sparber.org/CBOO.pdf for an overview of the system. Pay more attention to the headings of this article than the details. There is a layered approach being used here that starts with the general and becomes more specific.

What follows is the actual Arduino code. I have placed a few major milestones in square brackets enclosing all caps in red. In-line comments are mostly in lower case and follow "//". Again, I encourage you to look more at headings than detail.

Is this an example of functioning code? Yes. Is it an example of elegant code? I doubt it. But I did write it so years later I could read my comments and they made sense.

You will also see a few places where I changed the structure of the code yet left in some original code. This helped me transition from the old structure to the new.

A word of advice: do not print out this entire document. It is 150 pages long. If you do want to print some of it, I recommend you only do the first 17 pages.

A folder must be created that contains two files. The first, called "loop.ino" and the second is the program. In my case, this is called "SCC_1_5.ino"

Most of the operational code is within the function "loop". By placing "void" in front of "loop()" I am telling the compiler that I am defining this function and not to expect any value returned when it is called. However, I do sent it parameters.

```
void loop()//measure OD and/or ID as determined by the odID flag
{
Top:
initLoop();//initialize loop on each cycle including setting the flag restart to false

  if(odID == onlyOD || odID == ODandID){//measure an OD
    restart = measureAnOD();//along with measuring an OD, it will return the
flag restart
  }
  if((odID == onlyID || odID == ODandID)&& !restart){//measure an ID unless
restart flag set to true
    measureAnID();
  }
}
```

My second file, SCC_1_5.ino , contains the following large amount of code. You will see that I did not code everything I wanted to do. This was because I ran out of program memory for this particular Arduino compatible processor.

/* Caliper Boost sketch [TITLE]

Wednesday, November 06, 2013 [DATE TO IDENTIFIED NEWEST VERSION]

The caliper scanning code came from https://sites.google.com/site/marthalprojects/home/arduino/arduino-reads-digital-caliper [REFERENCE TO SOURCE OF HARDWARE DRIVER]

High Level Function List [THINK OF THESE AS CUSTOM SOFTWARE COMMANDS]

* fetchReading () //function does a single read of the caliper and puts the result in newResult

* localMinRead() //scans caliper output looking for local minimum; output is oldResult

* localMaxRead() //scans caliper output looking for local maximum; output is oldResult

* interpolate() //accepts rawResult in thou and outputs bestResult which is based on interpolating gage blocks measured

* displayAnswer() //format measured value and display on LCD in mm or inches

* calibrationMode() //collects gage block readings and builds a pair of tables used to improve caliper accuracy

* interpolation() //uses the gage block calibration tables to correct for caliper errors

Low Level Function List [THINK OF THESE AS CUSTOM SOFTWARE TOOLS]

*** setMicro_t0() //define t=0 that will not wrap around; returns microStartTime that won't have wrap problem**

*** setMilli_t0() //returns milliStartTime that won't have wrap problem**

*** microNow() //gives delta time in microseconds and takes care of any timer wrap. Result is in microDelta.**

*** milliNow() //gives delta time in milliseconds and takes care of any timer wrap. Result is in milliDelta.**

*** decode() //function reads the data burst after the first rising clock transition has occurred and stores it in newResult**

***startIDmeasPrompt() //prompts**

***startODmeasPrompt() //prompts**

***displayRadiusAndWait() //display radius and then holds last reading until jaws have opened more than Limit distance**

***roundToHalfThou() //input and output is passed as variable "roundToHalf"**

*** jitterAndPortalCheck() //while user is looking at hold message, this function waits until caliper moved more than jitterLimit plus also runs**

*** commandPortal() looking for user request for a command**

*** limitAndPortalCheck() //wait until jaws move more than Limit while checking for command request**

***odID() //lets user customize the measurement system to measure just OD, just ID, or both**

*** fractions() //displays the measured value as decimal and various fractions at the same time**

Feature List [WHAT WAS DONE AND WHAT IS LEFT TO DO]

 # means feature done

 # accept data from caliper

+ in thou

+ in mm

 # be able to look at a stream of data and retain the smallest value

 # display the peak reading and half of the peak reading at the same time

 # when readings increase by more than 0.1" different, take that as the start of a new sequence

 # when user opens jaws to more than 6", zeros, and then closes jaws shut and zeros, take that as the start of the calibration mode if in measurement phase or as the end of the the calibration mode if in calibration

 # when in calibration and the caliper is set to inches,

+the user places a gage block in the jaws with a value equal to x.xx0"

+after the minimum reading has been found, the display shows the assumed gage block value and says Next Reference

+the reference and measured values are stored in the thou table

+cycle repeats until end of calibration mode behavior seen

*** when in calibration and the caliper is set to mm,**

+the user places a gage block in the jaws with a value equal to x.x0 mm

+after the minimum reading has been found, the display shows the assumed gage block value and says Next Reference

+the reference and measured values are stored in the mm table

+cycle repeats until end of calibration mode behavior seen

 # upon exiting the calibration mode, the new data is combined with existing table data, if any, such that it is in ascending order.

Where new reference values equal existing reference values,replace the old with the new

 # when a minimum reading is detected, the corresponding table is used to estimate a more accurate answer by using interpolation

This sketch interfaces with a Harbor Freight digital caliper in order to improve its usability and accuracy

*** A pushbutton will cause power to be applied to the Arduino and display**

*** If no change in caliper data seen within 5 minutes,**

+reduce power to a minimum to the Arduino and display

+scan for new data every 2 seconds and if new data detected, return to full power

*** If no change in caliper data seen within 10 minutes, turn power to Arduino and display off**

 # the interface with the caliper is tri-stated when not needed to save the battery on the caliper

*** monitor battery voltage and give warning when getting low (maybe by flashing backlight and having a text message)**

*** go/no-go**

*** averaging of up to 10 readings; if fewer than 5 readings, just take average. At least 5 readings, throw out max and min and average rest**

*** fractional display: show decimal inches, 16ths, 32nds, and 64ths in the corners of display. If any of the numbers can be reduced, do so (so 4/8" would be 1/2" and 16ths would be blank)**

[HIGH LEVEL DESCRIPTION OF THE ANALOG HARDWARE]

The caliper interface circuit takes totem pole signals (0 and +1.5V) from the caliper and outputs inverted signals (0.1 and +5V) to the Arduino.

The caliper's signals are always active so the Arduino will tristate the interface when it is off in order to prevent caliper battery drain.

Caliper Interface Description [LOW LEVEL DESCRIPTION OF THE CALIPER DRIVER]

*** The caliper outputs 6 nibbles in 8.9 milliseconds. The first 5 nibbles are the magnitude with LSB first. When set to inches, the LSB**

is 0.5 thou. The next larger bit has a value of 1 thou. All larger bits are in units of whole thous. When set to mm, the LSB is 0.01 mm

*** Nibble 6, LSB is the sign bit. if 0, the data is positive. If 1, the data is negative. MSB flags thou (1) versus mm(0)**

*** The data bursts arrive every 120 milliseconds.**

*** Between data burst, the clock is low as seen by the Arduino. It then goes high in preparation for the first falling edge which will signify the first data bit**

*** more information can be found in the comments next to the code**

***/**

[DEFINITION OF ALL VARIABLES]

boolean noReadyPrompt; //turns off "Ready" prompt on LCD within LocalMinRead()

boolean foundRoom; //used to indicate that a blank space was found in the gage block array

boolean leave; //used to exit calibration mode

boolean cd; //caibrated data flag

boolean flat; //flag that is true if jaws were not moving just before lift off from measured surface

boolean changeMade;//flag used in review of gage_blocks[] review

boolean rejectBlock;//flag in a calibration function

boolean fractionalDisplay = false;//default is false

boolean gonogoFlag = false;//default is go/no go is disabled

boolean supressUL_CMD=true;//flag to tell displayAnswer that this is the first time UL/CMD shown so supress radius info

boolean restart = false;//flag set true when the command state has changed. It is cleared to false at the top of loop()

int i; //bit counter and in do loop; is 16 bits

int ii; //in do loop

int j; //in do loop

int k; //used in sort

int sign; //sign of data

int units; //mm or thou. See also the two #define lines below

int oldUnits; //holding register for previous units value

int gng_oldUnits;//same as oldUnits but used in go/no go function and must not conflict with oldUnits

int qaz; //scrach space used in gage block rounding function and in gonogo()

int edc;//used by factor()

int intInteger; //used in 0.5 round off function

int nextOpen; //pointer to next open location in calibration array

int clockpin = 3; // caliper interface circuit's clock connects to this pin

int datapin = 9; // caliper interface circuit's data connects to this pin

int caliperDisable = 10; //tristate pin to caliper interface.

```
int speedLimit = 100;  //maximum speed in thou per second of jaws just before touchdown before warning is displayed

int command = 6; //holds user command; see commandPortal(); initialized to no command requested

int oldCommand;

int odID = 1;//flag that tells loop to do just OD, just ID, or both - ODandID; default is OD

int oldodID;//used by go/no go to save old odID value

//int OD = 1; THESE ARE DEFINED WITH #DEFINES

//int ID = 2;

//int ODandID = 3;

int sixteenths;

int thirtyseconds;

int sixtyforths;

int denominator;

int numerator;
```

long value; // variable to build up the bits that will be the magnitude of the data; it is 32 bits

long bitstage; //position newest data bit before placing in value; it is 32 bits

long thouLimit = 100; // minimum distance in thou calipers must open to start a new cycle

long mmLimit = 2.54; // minimum distance in mm calipers must open to start a new cycle

long Limit; //distance calipers must open to start a new cycle

long tempmicros; //scratch variable to hold incremental time in microseconds

long time_newResult; //time stamp when "newResult" collected

long time_oldResult; //time stamp when "oldResult" collected

long time_oldestResult; //time stamp when "result" collected the time before oldResult

long t1; //time stamp just as jaws stopped moving

long t2; //time stamp one measurement before jaws stopped moving

long microStartTime; //time in microseconds at start of program modified to prevent wrap around of timer to zero

long milliStartTime; //time in milliseconds at start of program modified to prevent wrap around of timer to zero

long microDelta; //delta t from microNow()

long milliDelta; //delta t from milliNow()

float velocity; //velocity of jaws closing in thou per second

float newResult; //final caliper reading

float oldResult = 0; //previous caliper reading

float oldestResult = 0; //caliper reading two times back

float thouResult; //used by makeThou() to insure value is in thou

float mmResult;// used by makeMM() to insure value is in mm

float x1; //position of jaws just as they stopped moving

float x2; //position of jaws one snapshot before they stopped moving

float jitterLimit; //used in localMinRead to prevent false local min due to jitter

float actual; //actual value used to determine gage block value during calibration

float fractional; //fractional part of a number

float floatInteger; //integer value but in a float variable

float radius; //used during wait to unlock

float roundToHalf; //variable used within roundToHalfThou and roundToHalf_mm() functions

float displayedAnswer; //locks in value displayed and is used to insure radius is consistent

float gage_blocks[10]; //an array of gage block values used for calibration

float actual_results[10]; //an array of actual results when gage blocks measured

float gageBlock; //temporary storage of nominal gage block value

float keyActual; //temporary storage used in sorting routine

float keyBlock; //temporary storage used in sorting routine

float ref;//used in limitAndPortalCheck

float wsx;//used by roundToInteger()

float rfv;//used by roundToInteger()

float lowerLimit;

float upperLimit;

float tempLimit;

//String button = "inch/mm"; //could use an an experiment to see how much this saves.

[DEFINING SOME SYMBOLIC VALUES]

#define UNIT_MM 0

#define UNIT_THOU 1

#define onlyOD 1

#define onlyID 2

#define ODandID 3

[TELL COMPILER WHAT HARDWARE DRIVER(S) TO INCLUDE]

// include the library code:

#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins [DEFINES HOW I WIRED THE DISPLAY]

LiquidCrystal lcd(7, 6, 5, 4, 8, 2); //pin 3 was swapped for pin 8 because I wanted pin 3 for caliper clock; later found out it was not necessary

```
void setup() [ONE TIME SET UP OF FLAG, ARRAYS OF DATA, DISPLAY, AND CALIPER INTERFACE]

{

 //Serial.println("entering setup");

 noReadyPrompt=false; //most of the time we want the Ready prompt

 // initialize calibration array with the value 10000 which is larger than can be input by caliper. So 10000 is a flag saying this location is free

 for (i=0; i<10;i++)

 {

  gage_blocks[i]=10000;

  actual_results[i]=10000;

 }

 // Initialize the serial communication with the computer, at 9600 bits/second

 Serial.begin(9600);

 lcd.begin(16, 2);    // set up the LCD's number of columns and rows



 // set up the caliper interface pins
```

pinMode(clockpin, INPUT); //Caliper clock input

pinMode(datapin, INPUT); //Caliper data input

// set up the tri-state control to the caliper interface

pinMode (caliperDisable, OUTPUT); //caliperDisable will be low to power up the interface and tristated when the Arduino is off. Can be set high to disable too.

digitalWrite(caliperDisable,LOW); //SET CALIPER INTERFACE TO ACTIVE AND LEAVE IT THERE

lcd.print (" Caliper Boost"); [THIS IS THE FIRST THING DISPLAYED ON LCD; WE ARE TALKING TO THE LCD DRIVER WHICH TALKS TO THE LCD HARDWARE]

lcd.setCursor(0,1); //print next line on second row

lcd.print (" Version 1.5   "); //displays version only when Arduino powers up

delay(3000);

lcd.clear();

lcd.print(" UnLock/CoMmanD");

fetchReading();//get units at power up and use it to init oldUnits

oldUnits = units; //this prevents us from going right into command mode

} //end of setup

void initLoop()//initialize loop [ALL OF THE CODE IS HERE; THIS IS MY LAST RED COMMENT]

{

restart = false;//be sure restart flag is cleared

initCommandPortal();//sets command to 6 and sets oldUnits to current value. Is used by next function

}

boolean measureAnOD()//will return the restart flag which, if true, means to go Top:

```
{


  //caliper jaws have to close by Limit before we start to look for local
minimum;

  //also look for command to run.After command runs, it will return
here.

  if(restart){

    return restart;

  }//after a command is run, restart loop() THIS WAS goto Top BUT WITH
NEW STRUCTURE, TOP IS NOT DEFINED SO JUST RETURN

  startODmeasPrompt(); //last measurement persists on display until
caliper jaws start to close. When they stop closing

  //and start opening, we lock in OD measurement

  noReadyPrompt = false;//tells next function to output "to meas."
prompt

  initCommandPortal();//sets command to 6 and sets oldUnits to current
value. Is used by next function

  localMinRead(); //prompt user, monitors caliper for local minimum
and detect a possible push of inch/mm which means

  //command will be executed. When done, we will return to this point in
the loop; output is oldResult
```

```
  if(restart){

    return restart;

  }//after a command is run, restart loop() THIS WAS goto Top BUT WITH NEW STRUCTURE, TOP IS NOT DEFINED SO JUST RETURN

  interpolation(); //improve accuracy of caliper reading using gage blocks; input and output are oldResult; pass through if data not present

  displayAnswer(); //display best reading; input is oldResult and output is displayAnswer which is what is displayed

  displayRadiusAndWait();//radius is displayed   THIS WAS AT THE TOP BUT MAYBE IT MAKES MORE SENSE DOWN HERE

}

void measureAnID()//no need to return restart flag because next command in loop()is restart

{
```

```
  //caliper jaws have to open by Limit before we start to look for local maximum;

  //also looking for command to run. After command runs, it will return here.

  if(restart){

    return;

  }//after a command is run, restart loop() THIS WAS goto Top BUT WITH NEW STRUCTURE, TOP IS NOT DEFINED SO JUST RETURN

  startIDmeasPrompt(); //last measurement persists on display until caliper jaws start to open. When they stop opening

  //and start closing, we lock in ID measurement

  noReadyPrompt = false;//tells next function to output "to meas." prompt

  initCommandPortal();//sets command to 6 and sets oldUnits to current value. Is used by next function

  localMaxRead(); //monitor caliper for local maximum; then signal we are ready for new local minimum;

  //also looking for command to run.After command runs, it will return here.

  if(restart){
```

```
    return;

  }//after a command is run, restart loop() THIS WAS goto Top BUT WITH
NEW STRUCTURE, TOP IS NOT DEFINED SO JUST RETURN

  interpolation(); //improve accuracy of caliper reading using gage
blocks; input and output are oldResult; pass through if data not present

  displayAnswer(); //display best reading; input is oldResult and no
output to program

  //hold displayed result until user signals for a new reading or signals to
enter calibration mode

  //when user has signaled that they are done with last reading. See if
they want to enter calibration mode or take another reading

  displayRadiusAndWait();//radius is displayed   THIS WAS AT THE TOP
BUT MAYBE IT MAKES MORE SENSE DOWN HERE

}
```

```
void jitterAndPortalCheck()   //this code ignores caliper output jitter and holds until movement is more than jitterLimit;  is also opportunity for user to signal they want to input a command:

{

  //oldUnits = units; //DONE IN INITCOMMANDPORTAL BEFORE UNLOCK DISPLAYED SO BEFORE USER CAN CHANGE IT save state of inch/mm button before starting scan for change within commandPortal()

  do

  {



    fetchReading();



    //Serial.print("444 oldUnits = ");

    //Serial.println(oldUnits);



    //Serial.print("444 units = ");

    //Serial.println(units);
```

```
   commandPortal(); //check to see if inch/mm pushed while waiting for
movement greater than jitter. commandPortal will call command and
when

   //done will return with command = 6



   if (units==UNIT_MM) //if true, caliper is set to mm so max jitter is +/-
0.01 mm

   {

    jitterLimit=0.02;

   }

   else {

    jitterLimit = 1.0;

   } //otherwise we are in thou so max jitter is +/- 0.5 thou

 }

  while (abs(oldResult - newResult)< jitterLimit); //consider the caliper
not moved if jaws moved less than jitterLimit. command value not tested

  //because it will execute command if requested and then come back
when done.

}
```

```
void jitterCheck()   //this code ignores caliper output jitter and holds
until movement is more than jitterLimit;

//input is oldResult; output is newResult

{

 oldUnits = units; //save state of inch/mm button before starting scan
for change within commandPortal()

 do

 {



  fetchReading();//get new value for newResult
```

```
   if (units==UNIT_MM) //if true, caliper is set to mm so max jitter is +/-
0.01 mm

   {

    jitterLimit=0.02;

   }

   else {

    jitterLimit = 1.0;

   } //otherwise we are in thou so max jitter is +/- 0.5 thou

 }

  while (abs(oldResult - newResult)< jitterLimit); //consider the caliper
not moved if jaws moved less than jitterLimit

}
```

```
void limitAndPortalCheck()   //this code ignores caliper output until
jaws move more than Limit;
```

```
//is also opportunity for user to signal they want to input a command

//no inputs. Outputs are placing "Unlock" in lower right corner of LCD display and running commandPortal

//if units change.

{

 fetchReading();//get newResult and units

 oldUnits = units;//units before prompt displayed

 makeThou();//insure newResult is in thou; output is thouResult

 oldResult = thouResult;//jaw position before prompt displayed



 lcd.setCursor(10,1);//put cursor on bottom line in character location 11

 //note that line is not cleared first. Radius is displayed at the start of this line

 lcd.print("UL/CMD");//text ends up in the lower right corner of display



 //Serial.println("UL/CMD");
```

```
  do{ //scan caliper for jaws moved more than Limit or units change
   fetchReading();//get newResult and units
   makeThou();//insure newResult is in thou; output is thouResult
   if (units != oldUnits)//if user pushed inch/mm button
   {
    commandPortal();//process command and return here
    return;//leave function, we are done
   }
  }
  while(abs(thouResult - oldResult) < thouLimit);//both Results are in
thou regardless of state of units
  //Serial.print(" 888 thouResult =");
  //Serial.println(thouResult);
  //Serial.print("888 oldResult =");
  //Serial.println(oldResult);
}
```

```
void limitCheck()   //this code waits until jaws move more than Limit;

//no inputs.

{

 fetchReading();//get newResult and units

 oldUnits = units;//units before prompt displayed

 makeThou();//insure newResult is in thou; output is thouResult

 oldResult = thouResult;//jaw position before prompt displayed

 do{ //scan caliper for jaws moved more than Limit or units change

  fetchReading();//get newResult and units

  makeThou();//insure newResult is in thou; output is thouResult



 }

 while(abs(thouResult - oldResult) < thouLimit);//both Results are in
thou regardless of state of units
```

```
}
```

```
void makeThou() //input is units and newResult; output is thouResult in thou

{

 if (units == UNIT_MM) //if in mm, convert to thou

 {

  thouResult = newResult*39.37008; //newResult was in mm so convert to thou

 }

 else

 {
```

```
   thouResult = newResult; //newResult was in thou so no change

  }

}
```

```
void makeMM() //input is units and newResult; output is mmResult in
mm

{

  if (units == UNIT_THOU) //if in thou, convert to mm

  {

    mmResult = newResult/39.37008; //newResult was in thou so convert
to mm

  }

  else

  {
```

```
   mmResult = newResult; //newResult was in mm so no change

  }

}
```

```
void initCommandPortal() //initializes commandPortal() environment
before commandPortal() is run inside the lock loop

{

  command = 6; //initialize to show no command requested

  fetchReading();//get new current state of units

  oldUnits = units; //record inital state of the inch/mm button

}
```

```
void exitCommandPortalQ() //looks for change in inch/mm state when jaws more than 1/2" apart and if seen, sets command back to 6

{

  makeThou();//insure that caliper jaws distance is in thou; output is thouResult



  if (units != oldUnits && thouResult > 500 )

  {

    command = 6;

  }

  //if no change in state of units, command is left at previous value

}
```

void commandPortal() //accept user commands; place this function in each lock routines.

//Run initCommandPortalal()before entering wait loop.

//If the command state changes, the restart flag is set true. This

//causes loop() to restart so we avoid being in one command state while finishing

//the execution of the previous state.

/*

The user presses the inch/mm button while in the lock state in order to tell the software

they want to run a function. Then they move the jaws to one of the following positions

and push the inch/mm button a second time.

We enter function with units set to oldUnits. We must extit with the same value.

*** if they want to input gage blocks for calibration, then open the jaws to between 0.5" and 1.5"; called state 1**

*** if they want to run the system as a go/no-go tester, then open the jaws to between 1.5" and 2.5"; called state 2**

*** if they want to average a series of readings, then open the jaws to between 2.5" and 3.5"; called state 3**

*** to just change between mm and inches, have the jaws set less than 0.5"**

*** if they are already in one of these functions, then ack request and change command to 6**

***/**

{

fetchReading(); //see if user has pushed the inch/mm button when the jaws were more than 0 apart; if not, return

//Serial.print("345 units =");

//Serial.println(units);

```
//Serial.print("347 oldUnits =");

//Serial.println(oldUnits);



if (units == oldUnits)// if units didn't change, just return

{

 return;

}//else units did change



//Serial.print("44 newResult = ");

//Serial.println(newResult);

makeThou();//insure that we are testing thou for jaws spacing

//Serial.print("44 thouResult = ");

//Serial.println(thouResult);

 if (thouResult < 500)//to get here, units had to have changed; if jaws less than 1/2" apart, return. The user just wanted to change units and not do a command

{
```

```
   return;

  }



  restart = true;//set flag to tell loop() to restart because a command
state has been changed



  // otherwise button has been pushed with jaws more than 1/2" apart so
enter Command Mode with units != oldUnits

  if(command == 6)//if no command is active, we are trying to enter a
command

  {

   lcd.clear();

   lcd.print("  Command Mode");  //tell user they are in command mode

   lcd.setCursor(0,1);

   lcd.print(" Select option."); //user should move jaw into position and
press inch/mm again

   delay(1000);

   //user has pushed inch/mm button while in lock so now scan for
second push of inch/mm button and record caliper jaw position
```

```
   lcd.clear();

   lcd.print("Cal:1    Fract:2");

   lcd.setCursor(0,1);

   lcd.print("OD/ID:3 GonoGo:4"); //user should move jaw into position
and press inch/mm again



   do //monitor the inch/mm button for state changes and then act on
newResult value

   {

    fetchReading(); //get current units state and jaw position

   }

   while (units != oldUnits);

   //exit scan when inch/mm pushed second time so we are back to
original state of units

   //newResult tells us which command the user wants to run.



   makeThou(); //be sure caliper's value is in thou for following test;
output is thouResult
```

```
  lcd.setCursor(0,1);//prepare to change second line of display



  if (thouResult > 500 && thouResult < 1500) //user want to go to
calibration

  {

    command = 1;

    lcd.clear();

    lcd.print("   You chose");

    lcd.setCursor(0,1);

    lcd.print("  Calibration   ");

    delay(2000);

    calibrationMode();//run cal mode and then come back here with
command = 6; return will put us back where we started

    unitCheck();//insure caliper set to same units as when user entered

    return;//this should cause us to return from commandPortal()

  }
```

```
  if (thouResult > 1500 && thouResult < 2500)   //user want to go to
fractions test

  {

    command = 2;

    //Serial.println("command = 2");

    lcd.print("   You chose");

    lcd.setCursor(0,1);

    lcd.print("   Fractions    ");

    delay(2000);

    fractionsQ();//entering this function toggles flag on and off

    command = 6;//this line should go at end of fractions function when it
is written

    unitCheck();//insure caliper set to same units as when user entered

    return;//this should cause us to return from commandPortal()

  }

  if (thouResult > 2500 && thouResult < 3500)   //user want to go to
OD_ID selection function

  {
```

```
    command = 3;

    lcd.clear();

    lcd.print("   You chose");

    lcd.setCursor(0,1);

    lcd.print("  OD and/or ID  ");

    delay(2000);

    OD_ID();//run OD_ID mode and then come back here; return will put
us back where we started

    command = 6;//this line should go at end of averaging function when
it is written

    return;//this should cause us to return from commandPortal()

  }

  if (thouResult > 3500 && thouResult < 4500)   //user want to go to
Go/No-Go selection function

  {

    command = 3;

    lcd.clear();

    lcd.print("   You chose");
```

```
    lcd.setCursor(0,1);

    lcd.print("    Go/No Go");

    delay(2000);

    gonogo();//run gonogo() and then come back here; return will put us back where we started

    command = 6;//this line should go at end of averaging function when it is written

    return;//this should cause us to return from commandPortal()

   }

 }

  else //command not equal to 1, 2, or 3 so am in a command function and received a push of inch/mm button so user wants to exit function

 {

   lcd.clear();

   lcd.print("Exit requested.");

   lcd.setCursor(0,1);

   lcd.print("Please wait.");

   command = 6; //set flag to say we should not be in a command function
```

```
   unitCheck();//insure caliper set to same units as when user entered

  }

}

//end of commandPortal()
```

```
void fractionsQ() //accessed with command = 4; entering this function toggles fractionalDisplay flag on and off

{

  //Serial.print("949 fractionalDisplay =");

  //Serial.println(fractionalDisplay);



  lcd.clear();

  lcd.print("Fractions");
```

```
 lcd.setCursor(0,1);
 if (!fractionalDisplay){
  lcd.print("will be on.");
  fractionalDisplay = true;
 }
 else
 {
  lcd.print("will be off.");
  fractionalDisplay = false;
 }
 delay(2000);
 lcd.clear();
}//end of fraactionsQ()
```

```
void fractionDisplay()//input is units and oldResult; output to LCD is integer and "-" if integer non-zero.

//output variables are numerator and denominator

{

  newResult = oldResult;

  makeThou();//input is newResult and units; insure number being displayed is in thou; output is thouResult

  lcd.clear(); //start at top line, column 0




  roundToHalfThou(thouResult);




  lcd.print(thouResult/1000,4);//print decimal value in upper left corner
```

```
 intInteger = thouResult/1000;//extract the fractional part of
unitsThou; intInteger is (int)

 fractional = thouResult/1000 - intInteger;

 if (fractional == 0){//don't display a fraction if it is zero

  return;

 }



 //Serial.print("thouResult = ");

 //Serial.println(thouResult);



 //Serial.print("intInteger = ");

 //Serial.println(intInteger);



 //Serial.print("fractional = ");

 //Serial.println(fractional);



 //at most thirtyseconds; fractional is non-zero
```

```
  wsx = 32*fractional;//convert to at most 32nds; start by finding
number of 32nds in the fractional part; wsx is float



  //Serial.print("wsx = ");

  //Serial.println(wsx);



  roundToInteger();//input is wsx (float) and output is qaz (int); round
to integer



   if (qaz == 32){//increment integer part and zero fractional part

   intInteger = intInteger + 1;

   qaz = 0;

  }



  //Serial.print("1 qaz =");

  //Serial.println(qaz);
```

```
  factor();//input is qaz and output is edc (int),the largest multiple of 2 that fits evenly into qaz



  //Serial.print("1 edc = ");

  //Serial.println(edc);



  numerator = qaz/edc;//reduce integer numerator by as many 2s as possible

  denominator = 32/edc;//reduce denominator by same number of 2s



  lcd.setCursor(9,0);//top line, column 11



  if (intInteger != 0){//don't print a 0 integer

    lcd.print (intInteger);



  }
```

```
  if (numerator == 0){

    return;

  } //supress fraction if numerator is 0; if number is 0.000, then print
blank

  //otherwise print fraction



    if (intInteger != 0){

    lcd.print ("-");//if integer is 0, don't print -; if numerator = 0 we would
not have gotten this far

  }



  lcd.print(numerator);//send fraction to LCD

  lcd.print("/");

  lcd.print(denominator);

}//end of fractionDisplay()
```

```
void OD_ID()//sets option of OD only, ID only, or both

{

 lcd.clear();

 fetchReading();

 oldUnits = units;//record units state before asking for inch/mm to be pushed



AskAgain:

 lcd.print("OD=1 ID=2 both=3");

 lcd.setCursor(0,1);

 lcd.print("Then inch/mm.");



 do{
```

```
  fetchReading();//get newResult and units

 }

 while (units == oldUnits);//scan jaw position until units pressed.



 makeThou();//insure I get jaw position in thou. output is thouResult



 //Serial.print("thouResult =");

 //Serial.print(thouResult);



 if (thouResult > 500 && thouResult < 1500)   //user want to go to only
measure OD

 {

  odID = onlyOD;

  lcd.clear();

  lcd.print("Measure just OD.");

 }
```

```
 if (thouResult > 1500 && thouResult < 2500)   //user want to only
measure ID

 {

  odID = onlyID;

  lcd.clear();

  lcd.print("Measure just ID.");

 }



 if (thouResult > 2500 && thouResult < 3500)

 {  //user want to measureboth ID and OD

  odID = ODandID;

  lcd.clear();

  lcd.print("Measure OD & ID.");

 }



 if (thouResult < 500 || thouResult > 3500){
```

```
  //if not 1  2, or 3, repeat initial prompt

  goto AskAgain;

 }



 lcd.setCursor(0,1);

 lcd.print("Now push inch/mm");



 do{

  fetchReading();

 }

 while (units != oldUnits);//scan jaw position until units pressed.This
puts units back to original value

 lcd.clear();

}//end of OD_ID()
```

```
void unitCheck()//insure user sets caliper back to same units as when
oldUnits was defined

{

 fetchReading();//get current value for units

 if(units == oldUnits){

  return;

 }//otherwise, user must push inch/mm button

 lcd.clear();

 lcd.print("Please push");

 lcd.setCursor(0,1);

 lcd.print("inch/mm button. ");

 do {

  fetchReading();

 }

 while(units != oldUnits);//hold until user pushes inch/mm button
```

```
}
```

```
float roundToHalfThou(float roundToHalf) //input and output is passed as variable "roundToHalf"

{

 //round to the nearest 0.5


 if (roundToHalf == 0)

 {

  return roundToHalf; //prevent divide by zero and no need to round

 }


 sign = abs(roundToHalf)/roundToHalf; //pull out sign
```

```
  intInteger = abs(roundToHalf); //get the integer part of roundToHalf
which is float     qaz1

  //Serial.print("intInteger= ");

  //Serial.println(intInteger);

  newResult = intInteger; //convert integer back to float

  //Serial.print("newResult= ");

  //Serial.println(newResult);



  fractional = abs(roundToHalf) - newResult; //now have fractional part
in float

  //Serial.print("fractional= ");

  //Serial.println(fractional);

  if (fractional < 0.25)

  {

    roundToHalf = intInteger; //round down so oldResult will end in .0

    //Serial.println("<0.25");
```

```
}



if (fractional >= 0.25 && fractional <0.75)

{

  roundToHalf = intInteger + 0.5; //number will end in 0.5

  //Serial.println(">= 0.25 && <0.75");

}

if (fractional >= 0.75)

{

  roundToHalf = intInteger + 1; //round up to next highest thou. number will end in .0

  //Serial.println(">= 0.75"); //do something when var equals 2

}



roundToHalf = sign*roundToHalf;//restore sign

//Serial.print("roundToHalf= ");

//Serial.println(roundToHalf);
```

```
  //roundToHalf has now been rounded to the nearest 0.5 thou

  return roundToHalf;

} //end of roundToHalf function
```

```
void roundToHun_mm()//input and output are roundToHalf; input is in mm in the format i.xx; output is rounded to the nearest hundreth of a mm

{

  if (roundToHalf == 0)//prevent divide by zero and no need to round

  {

    return;

  }

  //round result to the nearest hundreth of a mm; below .xx5 rounds down
```

```
  sign = abs(roundToHalf)/roundToHalf; //pull out sign. Example: -1.234 so sign = -1



  intInteger = abs(roundToHalf*1000);//with sign removed, round to nearest hundreth; first make thouthanths place an integer.Example: 1234

  fractional = (intInteger + 5)/10;//Example: 1239/10 = 123.9

  intInteger = fractional;//take justinteger part; Example: 123.9 becomes 123 (an integer)

  roundToHalf = sign*intInteger;//put back sign and make float

  roundToHalf= roundToHalf/100;//divide by 100 to return to original value except now rounded; Example: -1 * 123/100 as float is -1.23

}
```

```
void roundToInteger()//input is wsx (float) output are qaz (integer)

{

  qaz = 10*wsx + 5; //if fractional part is > half, this will round up

  qaz = qaz/10;

}
```

```
void displayRadiusAndWait() //prints radius plus scans caliper output looking for calipers to open by "Limit"; also contains commandPortal

{

  bottomLineClear();

  if(command != 1 && !fractionalDisplay && !supressUL_CMD && !gonogoFlag){ //if not in calibration mode, displaying fractions, gonogo,
```

```
  //or doing first display of UL/CMD, then use standard prompt

  lcd.setCursor(0,1);//set up to print to second line of LCD

  if (units==UNIT_THOU){

   roundToHalf = displayedAnswer/2; //calculate radius in thou

   roundToHalfThou(roundToHalf);//round result to the nearest half
thou

   lcd.print(roundToHalf/1000,4);//display radius in bottom right
corner

  }

  else //caliper is set to mm

  {

   roundToHalf = displayedAnswer/2;//calculate radius in mm

   roundToHun_mm();//round result to the nearest hundredth of a mm

   lcd.print(roundToHalf,2);//display radius in bottom right corner

  }

  lcd.print ("r "); //tell user to open jaws. After jaws opened by at least
Limit, this prompt is replaced by startIDmeasPrompt

 }
```

```
  supressUL_CMD = false; //from now on, we will not supress radius due to UL/CMD

  limitAndPortalCheck();//reads units and jaw position, prints Unlock prompt, and then holds until jaws moved more than limit

  //and then runs commandPortal().

  //Serial.println("Unlock2");

}

//end of displayRadiusAndWait()
```

```
void localMinRead() //scans caliper output looking initial movement of more than jitter and then for local minimum

{

  /*Definition of terms

    newResult - newest caliper jaw position

    oldResult - reading before newResult that has an oldestResult different than itself; if oldResult does not change for a while,

    only the first reading is recorded along with its time stamp
```

oldestResult - reading directly before oldResult that is not equal to oldResult

There are the following cases:

1. new > old < oldest means that old is our local minimum so return with this value

2. new > old > oldest means continuously rising; keep looking

3. new < old < oldest means continuous falling; keep looking

4. new = old > oldest means it was rising and then stopped; keep looking

5. new = old < oldest means it was falling and then stopped; keep looking

***/**

setMilli_t0();//set t=0 in milliseconds

if (!noReadyPrompt)// during calibration phase we use different Ready prompt so supress this one

{

```
  lcd.clear();

  lcd.print("Measuring OD:");

  lcd.setCursor(0,1);//print to bottom line

  lcd.print((char)0b01111111); //this is an arrow <-

  lcd.print(" and then ");

  lcd.print((char)0b01111110); //this is an arrow ->

 }



 jitterCheck();//wait until jaws move more than jitter so we do not react to jitter which can look like a local minimum



 //[L1]

 //take distance readings and time stamp each one. Then look for local minimum. When found, use last two time stamps to calculate velocity on impact



 // Initialize search by defining first point, newResult and its time stamp is 0.
```

setMilli_t0(); //set t=0 on millisecond timer and store as milliStartTime which is used by milliNow()

fetchReading(); //caliper really moving so returns newResult; establish first reading and store as oldResult

milliNow();//get time now wrt t=0; milliDelta will be zero or very close to it

time_newResult = milliDelta; //newResult now has its corresponding time stamp: time_newResult

oldResult = newResult; //initialize oldResult and oldestResult as equal to newResult and do the same with their time stamps

time_oldResult = time_newResult;//we now have a flat line as our starting case

oldestResult = newResult;

time_oldestResult = time_newResult;

//Going into the search loop, we have oldResult and oldestResult set equal to newResult and their corresponding time stamps are also equal

while(true){ //Search Process runs continuously until local minimum detected

```
   //see if newResult = oldResult. If so, then leave oldResult and
oldestResults unchanged because we are on a flat spot; otherwise, we

   //are not on a flat spot so shift the data points over

   if(newResult != oldResult){

     oldestResult = oldResult; //SHIFT OVER THE LAST TWO DATA POINTS
since newResult is smaller than the oldResult, save previous oldResult as
oldestResult

     time_oldestResult = time_oldResult; //save previous oldResult time
stamp as oldestResult time stamp




     oldResult = newResult; //PREVIOUS NEW IS NOW OLD     since new
reading is smaller than the oldResult, save it as oldResult

     time_oldResult = time_newResult; //save time of newResult as
time_oldResult too

   }//if newResult does equal oldResult, then we don't change them but
do get a fresh copy of newResult




     fetchReading();//get new newResult

   milliNow(); //get time since t=0
```

```
   time_newResult = milliDelta;

   //we now have newResult which was just taken, oldResult and
oldestResult that are from when they were not equal so not on a flat.

   if (newResult > oldResult && oldResult < oldestResult)//if new, old,
and oldest define a local minimum, calculate velocity at touchdown and
return

   {

    //impact velocity is calculated from the motion just before we
reached local minimum so will always be from oldest and old



    //Serial.print("oldestResult = ");

    //Serial.println(oldestResult);

    //Serial.print("oldResult = ");

    //Serial.println(oldResult);



    //Serial.print("time_oldestResult = ");

    //Serial.println(time_oldestResult);

    //Serial.print("time_oldResult = ");

    //Serial.println(time_oldResult);
```

```
    velocity = abs(((oldestResult - oldResult)*1000)/(time_oldResult -
time_oldestResult)); //thou per second closure rate



    //Serial.print("velocity = ");

    //Serial.println(velocity);



    //Serial.print("111 local min velocity = ");

    //Serial.println(velocity);



    if (velocity > speedLimit ) //if jaws were closed faster than
speedLimit thou per second, just warn user

    {

     lcd.setCursor(0, 1); //put warning message on second line

     lcd.print("Jaws hit at: ");
```

```
      lcd.print(velocity);
      delay(1000); //warning displayed for 1 seconds
    }
    return;//we return with local minimum being oldResult
  }
 }
}
//end of localMinRead function
```

void localMaxRead() //scans caliper output looking initial movement of more than jitter and then for local maximum; output is oldResult

{

/*Definition of terms

newResult - newest caliper jaw position

oldResult - reading before newResult that has an oldestResult different than itself; if oldResult does not change for a while,

only the first reading is recorded along with its time stamp

oldestResult - reading directly before oldResult that is not equal to oldResult

There are the following cases:

1. new < old > oldest means that old is our local maximum so return with this value

2. new > old > oldest means continuously rising; keep looking

3. new < old < oldest means continuous falling; keep looking

4. new = old > oldest means it was rising and then stopped; keep looking

5. new = old < oldest means it was falling and then stopped; keep looking

```
 */


 setMilli_t0();//set t=0 in milliseconds


 if (!noReadyPrompt)// during calibration phase we use different Ready prompt so supress this one

 {

  lcd.clear();

  lcd.print("Measuring ID:");

  lcd.setCursor(0,1);//print to bottom line

  lcd.print((char)0b01111110); //this is an arrow ->

  lcd.print(" and then ");

  lcd.print((char)0b01111111); //this is an arrow <-

 }


 jitterCheck();//wait until jaws move more than jitter so we do not react to jitter which can look like a local minimum
```

```
  //[L1]

  //take distance readings and time stamp each one. Then look for local
minimum. When found, use last two time stamps to calculate velocity on
impact



  // Initialize search by defining first point, newResult and its time stamp
is 0.

  setMilli_t0(); //set t=0 on millisecond timer and store as milliStartTime
which is used by milliNow()

  fetchReading(); //caliper really moving so returns newResult; establish
first reading and store as oldResult

  milliNow();//get time now wrt t=0; milliDelta will be zero or very close
to it

  time_newResult = milliDelta; //newResult now has its corresponding
time stamp: time_newResult

  oldResult = newResult;    //initialize oldResult and oldestResult as
equal to newResult and do the same with their time stamps

  time_oldResult = time_newResult;//we now have a flat line as our
starting case

  oldestResult = newResult;
```

time_oldestResult = time_newResult;

//Going into the search loop, we have oldResult and oldestResult set equal to newResult and their corresponding time stamps are also equal

while(true){ //Search Process runs continuously until local minimum detected

//see if newResult = oldResult. If so, then leave oldResult and oldestResults unchanged because we are on a flat spot; otherwise, we

//are not on a flat spot so shift the data points over

if(newResult != oldResult){

oldestResult = oldResult; //SHIFT OVER THE LAST TWO DATA POINTS since newResult is smaller than the oldResult, save previous oldResult as oldestResult

time_oldestResult = time_oldResult; //save previous oldResult time stamp as oldestResult time stamp

oldResult = newResult; //PREVIOUS NEW IS NOW OLD since new reading is smaller than the oldResult, save it as oldResult

time_oldResult = time_newResult; //save time of newResult as time_oldResult too

```
  }//if newResult does equal oldResult, then we don't change them but
do get a fresh copy of newResult



    fetchReading();//get new newResult

   milliNow(); //get time since t=0

   time_newResult = milliDelta;

   //we now have newResult which was just taken, oldResult and
oldestResult that are from when they were not equal so not on a flat.

   if (newResult < oldResult && oldResult > oldestResult)//if new, old,
and oldest define a local maximum, calculate velocity at touchdown and
return

   {

    //impact velocity is calculated from the motion just before we
reached local maximum so will always be from oldest and old

    velocity = abs(((oldestResult - oldResult)*1000)/(time_oldResult -
time_oldestResult)); //thou per second closure rate



    //Serial.print("111 local min velocity = ");

    //Serial.println(velocity);
```

```
    if (velocity > speedLimit ) //if jaws were opening faster than
speedLimit thou per second, just warn user

    {

     lcd.setCursor(0, 1); //put warning message on second line

     lcd.print("Jaws hit at: ");

     lcd.print(velocity);

     delay(1000); //warning displayed for 1 seconds

    }

    return;//we return with local maximum being oldResult

  }

 }//evaluate new data set

}//end of localMaxRead function
```

```
void milliNow()//returns milliDelta

{

 milliDelta = millis() - milliStartTime;

 //Serial.print("999 milliDelta = ");

 //Serial.println(milliDelta);



 if (milliDelta >= 0)

 {

  return;

 }

 else

 {

  milliDelta = 2147483647 + milliDelta; //milliDelta is equal to or less than zero so timer wrapped. Add maximum positive value for a long so delta is correct
```

```
  }


}
```

```
void recordBlock() //scans caliper output looking for local minimum. Ignore downward, upward, and local maximum;

//if the data stops changing by less than jitter, keep reading for new newResult; output is oldResult

{

  //flat = false; //initialize flag to say caliper jaws are moving

  setMilli_t0();//set t=0 in milliseconds



  //take distance readings and time stamp each one. Then look for local minimum. When found, use last two time stamps to calculate velocity on impact

  // INITIALIZE SEARCH BY DEFINING FIRST POINT, NEWRESULT first point and time stamp it as t = 0
```

```
  fetchReading(); //read caliper with result equal to newResult; establish first reading and store as oldResult

  milliNow();//get time now wrt t=0 which is called milliDelta

  time_newResult = milliDelta;

  oldResult = newResult;    //INITIALLY SET OLD = NEW UNTIL SEARCH CYCLE STARTS. WE NOW HAVE A DEFINED NEW AND OLD RESULT

  time_oldResult = time_newResult;

  do    //SEARCH CYCLE    read the caliper looking for a local minimum with respect to the first reading

  {

    //shift over readings so the last two are kept unless we are on a flat spot



    while(newResult == oldResult){//on a flat spot so keep taking newResult points until off of it; note that jitter will cause a state change

      fetchReading(); //read caliper with result equal to newResult

      milliNow();//get time now wrt t=0 which is called milliDelta

      time_newResult = milliDelta;

    }//proceed because new jaws are moving again
```

```
   oldestResult = oldResult; //SHIFT OVER THE LAST TWO DATA POINTS since newResult is smaller than the oldResult, save previous oldResult as oldestResult

   time_oldestResult = time_oldResult; //save previous oldResult time stamp as oldestResult time stamp



   oldResult = newResult; //PREVIOUS NEW IS NOW OLD     since new reading is smaller than the oldResult, save it as oldResult

   time_oldResult = time_newResult; //save time of newResult as time_oldResult too



   do {//again check to see if jaws moving. If not, keep scanning until they do but old and oldest are not changed. This covers case

     //rise followed by flat spot

     fetchReading();//GET FRESH NEWEST POSITION   returns newResult

     milliNow();

     time_newResult = milliDelta;

   }
```

```
  while (newResult == oldResult);//scan for new jaw position until jaws move, even due to jitter



  if(newResult == oldResult){

   //Serial.println("flat");//should not see this result

  }



  if(newResult < oldResult && oldResult < oldestResult){

   //Serial.println("falling");

  }



  if(newResult > oldResult && oldResult > oldestResult){

   //Serial.println("rising");

  }



  if(newResult > oldResult && oldResult < oldestResult){

   //Serial.println("local min");
```

```
  }


  if(newResult < oldResult && oldResult > oldestResult){

    //Serial.println("local max");

  }


 }// falling rising local maximum

  while((newResult < oldResult && oldResult < oldestResult) || (newResult > oldResult && oldResult > oldestResult) || (newResult < oldResult && oldResult > oldestResult));  //only stop looking when we see local minimum

  //if we see downward, upward, or local maximum, keep looking



  //Serial.println(newResult);

  //Serial.println(oldResult);

  //Serial.println(oldestResult);
```

```
  //I need that "equal to" part because when newResult < oldResult I set
oldResult=newResult and then

  //go back for another reading.

  //exit "do" when  local minima found and stored as oldResult




   //calculate the caliper closing velocity next to see if local minima can
be trusted or if user smashed jaws together too fast

  //if (flat == true)//FLAT NEVER TRUE SO IF THIS WORKS, REMOVE
CODE

  //{

  //velocity = abs(((x2 - x1)*1000)/(t2 - t1)); //thou per second closure
rate

  //}

  //else

  //{ //we were not on a flat spot so can use last two data points to
calculate impact velocity

  velocity = abs(((oldestResult - oldResult)*1000)/(time_oldResult -
time_oldestResult)); //thou per second closure rate

  //}
```

```
  //Serial.print("311 local min velocity = ");

  //Serial.println(velocity);



  if (velocity > speedLimit ) //if jaws were closed faster than speedLimit
thou per second, just warn user

  {

    lcd.setCursor(0, 1); //put warning message on second line

    lcd.print("Jaws hit at: ");

    lcd.print(velocity);

    delay(1000); //warning displayed for 1 seconds

  }

}

//end of recordBlock()
```

void setMicro_t0() //returns microStartTime that won't have wrap problem

// This function provides a time stamp in microseconds that will not wrap back to zero as long as the time from the start of

//measurement to end is less than about 30 minutes

{

microStartTime = micros(); //record time in microseconds at start of interval

}

void microNow() //returns microDelta which will not wrap in a "reasonable" period of time

```
{
  microDelta = micros() - microStartTime;
  if ( microDelta > 0)
  {
    return;
  }
  else
  {
    microDelta = 2147483647 + microDelta; //microDelta is equal to or less than zero so timer wrapped. Add maximum positive value for a long so delta is correct
  }
}
```

```
void setMilli_t0() //returns MiilliStartTime which will not wrap in a "reasonable" period of time

// This function provides a time stamp in milliseconds that will not wrap back to zero as long as the time from the start of

//measurement to end is less than about 23 days

{

  //Serial.println("entering setMilli");

  milliStartTime = millis();  //record time in millisseconds at start of interval

}
```

void fetchReading() //function does a single read of the caliper and puts the result in newResult with units in the variable units

/* Remember that clock and data out of caliper are inverted by the interface circuit. This means the clock is low when between data bursts and that

a falling edge marks the capture of valid data.

Look for the start of the next data burst by monitoring the clock:

The clock must go from low [3] to high [4] after more than 8.9 microseconds to be the start of a new bust.

clock bursts as seen at Arduino:____/-last_burst-1\2_____>9000 microseconds_____3/4-next_burst-_____

within each burst we have 24 falling clock edges to signal the 24 bits

***/**

{

while (digitalRead(clockpin)==HIGH) {

}; //While clock HIGH, just wait.When clock goes low, we exit this line of code. This event might be the final

//falling edge of a data burst(between points 1 and 2).

//tempmicros=micros()-microStartTime; //Since we left the above "while" statement, the clock must have just gone low (a falling edge) and might be at point 2;record present time in microseconds.

setMicro_t0(); //replaces above line //microStartTime was the time, in microseconds, at the start of the measurement cycle. If micros() was between zero and 2E9, microStartTime equals that micros() reading.

```
  //If micros() was between 2E9 and maximum count of about 4E9, we subtract 2E9. So in all cases, microStartTime is less than 2E9. This value is more than 33.3 minutes,

  // I move it back by 33.3 minutes to prevent timer wrap around past 0 error. Each time I call up micros() I must subtract off microStartTime.



  while (digitalRead(clockpin)==LOW) {

  }; //We now just wait while clock is low. This might be the time between data bursts (points [2] and [3]) but don't know yet.



  //if (((micros()-microStartTime)-tempmicros)>9000)  //Since we left the above "while" statement,the clock just went high again. If the clock was low longer than 9000 microseconds, we are at the start of a

  microNow(); //this and the next line replace the one above

  if (microDelta > 9000)                        // new burst of data (point 3)so time to collect data

  {

    decode(); //decode a data burst and store in newResult



  } //End of "if" statement that collects a data burst if we were at the start of the burst.
```

```
}

//end of fetchReading

void decode()   //function reads the data burst after the first rising clock transition has occurred and stores it in newResult

{

  sign = 1; //initialize sign to positive. It will only be changed if bit 20 is a 1 which means the sign is negative

  value = 0; //initial value to all zeros. It will hold final value.qwas

  units = UNIT_MM; //initialize units to mm. It will only be changed to thou if bit 24 is a 1.

  // we enter the decode function with the clock high and will clock in the first bit on the falling edge

  /* Clock during a data burst:
```

First clock nibble:_____[3]/[4]-[a]\[b]_[c]/[d]-[e]\[f]_[g]/[h]-[i]\[j]_[k]/[l]-[m]\[n]_____

points [3] and [4] tie back to previous waveshape.

***/**

for (i=0;i<24;i++) //there are 20 bits holding the magnitude plus 4 with overhead. We count from 0 to 19 because the LSB, i=0, is not shifted to the left.

{

while (digitalRead(clockpin)==LOW) {

} //this line is not needed for i=0 because we start out with clock high at [a]. For the rest of the reads it gets us through the rising edge

//which must be followed as a delimiter between falling edges

while(digitalRead(clockpin)==HIGH){

} //The falling edge is detected when this line of code is exited. For the first nibble it is [a]\[b],[e]\[f],[i]\[j],[m]\[n].

```
   if (digitalRead(datapin)==LOW)    //Clock just fell so it is time to
theread data bit. If it is a 0, then there was a 1 out of the caliper. If it is a
0, we do nothing because

   {                        //"value" was initialized to all zeros so there is nothing
to change.



    if (i<20)    //if the data bit is part of the magnitude, insert it into
"value" at the correct bit position

    {

      bitstage=0; //initialize staging area for newest data bit

      bitstage = 1<<i; //move data bit, which is a 1, into its correct
position; note that for i=0 there is no shift

      value|= bitstage;  // logic OR  places data bit within "value" using the
"compound OR" function

    }



    if (i==20) //the 21st bit (since starting count at 0, it is i=20), is the
sign bit. If we got this far in the logic, it has a value of 1 out of the caliper
so the sign is negative.

      //I set sign to -1. It will later multiply the value when in mm. CLEAN
UP CODE SO THOU USE SIGN THE SAME WAY
```

```
    {
      sign=-1;
    }


    if (i==23)  //the 24th bit (since starting count at 0, it is i=23), is the units bit.  If we got this far in the logic, it has a value of 1 out of the caliper so the units are in thou
    {
      units=UNIT_THOU;
    }
   }
 }
 if(units==UNIT_MM)
 {
   newResult=(value*sign)/100.00;
   Limit=mmLimit; //caliper open distance to signal a new measurement cycle set for mm
```

```
 }

 else

  //units are thou

 {

  newResult=(((value >> 1)+ (value & 0x0001)* 0.5)*sign);

  Limit=thouLimit;//caliper open distance to signal a new measurement cycle set for thou



  /* The LSB is half a thou so I first shift "value" over 1 bit to make the number who thou. Then I extract the LSB from the original

   "value" and multiply it by half a thou. It is then added to the shifted "value" so the digit to the right of the decimal can now be

   either 0 or 0.5. And finally, I multiply by the sign.

   */



 }

}

// end of decode()
```

```
void displayAnswer() //format measured value and display on LCD in
mm or inches

{

 displayedAnswer = oldResult;//this insures that when I calculate
radius, I am using the correct value

 if(fractionalDisplay){

  fractionDisplay();

 }

 else //is not fractional display so show decimal

 {

  decimalDisplay();



  if (gonogoFlag)//if go/no go enabled (i.e. true), overwrite bottom line
with go/no go results
```

```
  {
    thouResult = displayedAnswer;
    makeThou(); //regardless of state of caliper, insure we are working in thou
    displayedAnswer = thouResult;
    lcd.setCursor(0,1);//go to bottom line
    if(displayedAnswer < lowerLimit){
     lcd.print("Under        ");
     lcd.print((char)0b01111110); //this is an arrow ->>
    }


    if(displayedAnswer == lowerLimit){
     lcd.print("At lower limit.");
     lcd.print((char)0b01111110); //this is an arrow ->>
    }
```

```
if(displayedAnswer > upperLimit){
 lcd.print("Over         ");
 lcd.print((char)0b01111110); //this is an arrow ->>
}

if(displayedAnswer == upperLimit){
 lcd.print("At upper limit.");
 lcd.print((char)0b01111110); //this is an arrow ->>
}

if(displayedAnswer > lowerLimit && displayedAnswer < upperLimit){
 lcd.print("OK           ");
 lcd.print((char)0b01111110); //this is an arrow ->>
}
```

```
    limitCheck();//wait for jaws to move more than limit


   }


 }


}//end of displayAnswer




void topLineClear()//erase top line of LCD

{

 lcd.setCursor(0,0);//move to top line, first character

 lcd.print("               ");

 lcd.setCursor(0,0);//insure we are back to top line, first character
```

```
}
```

```
void bottomLineClear()//erase bottom line of LCD
{
  lcd.setCursor(0,1);//move to bottom line, first character
  lcd.print("                ");
  lcd.setCursor(0,1);//insure we are back to bottom line, first character
}
```

```
void decimalDisplay(){

  // displayedAnswer = oldResult;//this insures that when I calculate radius, I am using the correct value THIS ACTION WAS MOVED TO CALLING PROGRAM

  topLineClear();//erase top line of LCD

  lcd.setCursor(15,0); //put cursor in upper right corner

  if(cd==true)

  {

    //Serial.println("cd is true");



    lcd.print("*"); //a star in the upper right corner means displayed number has been calibrated

  }

  else

  {
```

```
  lcd.print("o"); //a - in the upper right corner means displayed number has not been calibrated

 }



 //print oldResult to LCD

 lcd.setCursor(0,0);//print measurment on first line



 if (units==UNIT_MM)

 {

  lcd.print(oldResult, 2);//print with 2 places for mm

  lcd.print(" mm");

 }

 else

 {

  lcd.print(oldResult/1000, 4); //and 4 place for thou to right of decimal

  lcd.print(" inches");//top right character position is left unchanged because that is where my interpolation indicator goes
```

```
  }
} //end of decimalDisplay()


void startIDmeasPrompt()//tell user how to measure an ID


{
  //Serial.print("77 command= ");
  //Serial.println(command);
  //if(command == 1){
  // return;
  //}
```

```
  lcd.setCursor(0,1);



  if(gonogoFlag){//if in go/no go mode, use alternate prompt so it does
not

    //hit Under, OK, Over outputs

    lcd.print("      ");//enough room to print "Under " which is 6 characters

    lcd.print((char)0b01111110); //this is an arrow ->>

    lcd.print(" & then ");

    lcd.print((char)0b01111111); //this is an arrow <<-

  }

  else //not using go/no go feature so can use more of bottom line for
prompt

  {

    lcd.print((char)0b01111110); //this is an arrow ->>

    lcd.print((char)0b01111110); //this is an arrow ->>

    lcd.print(" and then ");
```

```
   lcd.print((char)0b01111111); //this is an arrow <<-

   lcd.print((char)0b01111111); //this is an arrow <<-

   lcd.print ("  ");

  }

}
```

```
void startODmeasPrompt()//tell user how to measure an OD: <<-<<- and
then ->>->>

{

  lcd.setCursor(0,1);

  Serial.print("gonogoFlag =");

  Serial.print(gonogoFlag);
```

```
 if(gonogoFlag){//if in go/no go mode, use alternate prompt so it does
not

   //hit Under, OK, Over outputs

   lcd.print("     ");//enough room to print "Under " which is 6 characters

   lcd.print((char)0b01111111); //this is an arrow <<-

   lcd.print(" & then ");

   lcd.print((char)0b01111110); //this is an arrow ->>

 }

 else //not using go/no go feature so can use more of bottom line for
prompt

 {

   lcd.print((char)0b01111111); //this is an arrow <<-

   lcd.print((char)0b01111111); //this is an arrow <<-

   lcd.print(" and then ");

   lcd.print((char)0b01111110); //this is an arrow ->>

   lcd.print((char)0b01111110); //this is an arrow ->>
```

```
   lcd.print ("  ");

  }

}
```

```
void interpolation() // using gage-blocks[] and actual_results[] arrays, the measured value's accuracy is improved

// input and output are the variable oldResult

//output is rounded to the nearest 1/2 thou because input was rounded that way

//calibration table is built by reading essentially perfect gage blocks to resolution of +/- 0.25 thou

//because the 0.5 thou indicator can flicker if I'm right on the edge of x.0 or x.5.

//Each reading from the caliper is +/- 0.25 thou for the same reason
```

```
//assuming repeatability better than  resolution, maybe I can claim an output from this interpolation

//of +/- 0.5 thou

{

 if (command == 1){

   return;

 }//used in loop to prevent interpolation running when trying to get to calibration mode

 //Serial.println("in interpolation");



 for (i=0; i<10 && gage_blocks[i] != 10000;i++) //diagnostic

 {

   //Serial.print ("333 gage_blocks[");

   //Serial.print (i);

   //Serial.print ("]= ");

   //Serial.println (gage_blocks[i]);
```

```
  //Serial.print ("actual_results[");

  //Serial.print (i);

  //Serial.print ("]= ");

  //Serial.println (actual_results[i]);

 }



 cd = false; //initialize flag to say data is not calibrated



 if (actual_results[0] == 10000 || oldResult < 0) //if first entry is empty,
silently return. User did not put any gage blocks in system.

  // if oldResult < 0, then can't calibrate so silently return

 {

  //Serial.print("actual_results[0]= ");

  //Serial.println(actual_results[0]);

  //Serial.println("array empty");

  return;

 }
```

```
 //locate correct segment in actual_results[] array


 j=10; //preset j to 10 so I can tell if the search fails


 for (i=0; i<10 && actual_results[i] != 10000 && actual_results[i+1] !=
10000 ; i++) //qazp empty elements are filled with the number 10000

  //and signal end of data. If an element empty, mark result as not cal

 {

  if (oldResult >= actual_results[i] && oldResult < actual_results[i+1]) //
if oldResults is between entry i and i+1

  {

   //Serial.println("found correct segment");

   j=i; //record bottom end of correct segment

   break; //exit search for correct segment  qazz

  }

 } //otherwise, check the next segment
```

```
if (j>=10) //if true, ran out of table
{
  //Serial.println("segment not found");
  return; //so just silently return except with cd left as false
}

//do interpolation

  //but first do a divide by zero check

actual = actual_results[j +1] - actual_results[j];
if (actual==0.0)
{
  lcd.clear();
  lcd.print("Run calibration.");
```

```
   lcd.setCursor(0,1);

   lcd.print("Data damaged."); //if calibration data damaged, pass
oldResult right through

   //and warn user for 3 seconds

   delay(3000);

   return;

 }



  //do interpolation to improve accuracy of oldResult

  oldResult =  gage_blocks[j] + ((gage_blocks[j+1] -
gage_blocks[j])*((oldResult-actual_results[j])/actual));

  roundToHalfThou(oldResult);//round result to the nearest half thou
qasz

  cd = true; //oldResult has been made more accurate due to calibrated
data. This flag will put a "*" in the

  //upper right hand corner of the LCD display

}

//end of interpolation()
```

```
void exitCalibrationQ()//calibration is all done in inches so if units is in mm, then it means user wants to exit calibration mode; change command to 6 and return

{

 lcd.clear();

 lcd.print("Push inch/mm");

 lcd.setCursor(0,1);

 lcd.print("to exit.");

 setMilli_t0();

 milliNow();

 while(units == UNIT_THOU && milliDelta < 2000){

  fetchReading();
```

```
    milliNow();//update milliDelta
  }


  if(units == UNIT_MM){
    lcd.clear();
    lcd.print("Exiting.");
    lcd.setCursor(0,1);
    lcd.print("Please wait.");
    command = 6;
  }
}
```

```
void promptForBlock()//tell user to close jaws around block; they may open the jaws first but ignore that movement. Only record local minimum

{

  lcd.clear();

  lcd.print("Close jaws");

  lcd.setCursor(0,1);//go to bottom line

  lcd.print("on block.");

}
```

```
void processBlock()//use oldResult to figure out the block size which
will be gageBlock

{

  //gage blocks must be zero in the thou place (example: 0.120" is ok but
0.121" is not). Round newResult

  //to nearest 10 thou and store in gage_blocks array at location
nextOpen

  actual = oldResult; //save value used to determine gage block

  qaz = (oldResult+5)/10; //oldResult is float and qaz is integer; by
adding 5 thou to oldResult and then

  //dividing by 10 we truncate via conversion to integer,

  qaz=qaz*10.; //then multiply by 10 to restore estimate to a value
rounded to the nearest 10 thou; I get the nominal gage block value

  gageBlock = qaz;//block measurement rounded to the nearest 10 thou
which gives nominal gage block value

}
```

```
void displayBlock()//show user the block measured     werrf

{

  lcd.clear();

  lcd.print("Block found:");

  lcd.setCursor(0,1);//move to bottom line

  lcd.print(gageBlock/1000,3);//print in inches with 3 places to right of
decimal point

  lcd.print(" inches");

  delay(3000);//show for 3 seconds and return

}
```

```
void acceptOrRejectBlock()//give user chance to reject block.User given
2 seconds to respond. If units pushed twice during those 2 seconds,
```

```
//it means reject so set rejectBlock to true
{
 rejectBlock = false;//initialize flag
 lcd.clear();
 lcd.print("Keep this block?");
 lcd.setCursor(0,1);//move to bottom line
 lcd.print("Push twice = no.");
 setMilli_t0();//define t=0
 milliNow();//define first milliDelta
 while(milliDelta < 2000){
  milliNow();//get current time since t=0
  fetchReading();//read caliper
  if (units == UNIT_MM){
    rejectBlock = true;
    lcd.clear();
    lcd.print("Block removed.");
```

```
    delay(2000); //this delay gives user time to see message plus will cas
an  exit of while statement

   }

 }

  insureInches();//if caliper not set to inches, user asked to change it and
program waits until it is in inches

}
```

```
void insureInches(){//if caliper not set to inches, user asked to change it
and program waits until it is in inches

  if(units == UNIT_MM){

   delay(1000);//give user 1 second to push inch/mm second time

   if(units == UNIT_MM){
```

```
    lcd.clear();
    lcd.print("push inch/mm");
    lcd.setCursor(0,1);
    lcd.print("again please.");
   }
   while(units == UNIT_MM){
    fetchReading();//read units unit user goes back to inches
   }//units are back to inches now
  }
}



void calibrationMode()
{
  //Serial.println("1");
```

/* calibration mode

Enter calibration mode by having command = 1. We are then promted for the first gage block which must be in increments of 10 thou.

The program determines its nominal value by rounding the reading to the nearest 10 thou.It then

records the nominal value and the measured value in two arrays. These arrays were initialized to

"10000" in each element which is beyond the range of the caliper. This tells the program

where the data ends.

After each calibration point is recorded, program prompts for a

new gage block. When user is done, they press the inch/mm button with jaws wider than 1/2".

The program then sorts the updated tables in ascending order in preparation for interpolation

performed during the measurement mode. We then return the user to the measurement mode.

*/

```
 if (command !=1)// only if command equals 1 do we start calibration.

 {

  //Serial.println("2");

  return;

 }



  //command = 1 which means user wants to run calibration; they must not change unit state unless they want to leave calibration



  lcd.clear(); //[a]

 lcd.print("Calibration Mode");

 delay(2000);

  gageBlock = -1; //if user wants to leave before gageblock measured, gageBlock will have been initialized to -1 and don't use it

  fetchReading(); //read current inch/mm state which will be put in oldUnits

  oldUnits = units;//NOT SURE THIS IS USED ANYMORE
```

```
  if(units == UNIT_MM)//calibration must be done with gage blocks in
inches, not mm

  {

    lcd.clear();

    lcd.print("Please change");

    lcd.setCursor(0,1);

    lcd.print("to inches.");

    //delay(2000); //no reason to add wait since we are monitoring units
for the change

    while(units == UNIT_MM){

      fetchReading();

    }

    lcd.clear();

    lcd.print("Thank you.");

    delay(2000);

  }

  //Serial.println("3");
```

```
Top:

  while (command == 1) //[b] stay in calibration mode until user signals
they want to exit. Then sort tables and exit function

  {

    exitCalibrationQ();//if user pushed inch/mm so units now equals
UNIT_MM, it means they want to exit. Change command to 6.

    if(command != 1){

      goto Top;

    }

    promptForBlock();//tell user to close jaws around block; they may
open the jaws first but ignore that movement. Only record local
minimum

    jitterCheck();//wait until jaws moved more than jitter

    recordBlock();//find block and result is in oldResult

    processBlock();//use oldResult to figure out the block size which will
be gageBlock

    displayBlock();//show user the block measured and wait 3 seconds

    //acceptOrRejectBlock();//give user chance to reject block.User given
2 seconds to respond. If units pushed twice during those 2 seconds,

    //it means reject so set rejectBlock to true.
```

```
  if (rejectBlock == true){

   goto Top;

  }



  if (gageBlock < 0) //reject any negative gage blocks

  {

   lcd.clear(); //warn user that gageblock can't be negative

   lcd.print("No negative");

   lcd.setCursor(0,1);

   lcd.print("gage blocks.");

   delay(3000);

   goto Top;//reject gageBlock and ask for new one

  }



  //store gage block and caliper values if possible

  foundRoom=false; //[d] initialize flag to say no room found in the gage block calibration array
```

```
   for (i=0;i<10 && !foundRoom;i++) //search array to see if this gage
block was previously put in array until room has been found

   {

    if (gageBlock == gage_blocks[i])

    {

      nextOpen=i; //record location; logic will find first one but there
should never be more than 1

      foundRoom=true; //set flag to say we found room in the gage block
calibration array

      //Serial.println("f2.23 duplicate.");

    }  //if nominal gage block value not found in array, search for next
open data location

   }



   if(!foundRoom)  //if no room found due to duplicate entry search the
entire gage block array for an open space

   {

    // to get this far, there was no existing entry so find the first emplty
location
```

```
    for (i=0;i<10 && !foundRoom ;i++) //find first empty location unless
no room found

    {

     if(gage_blocks[i] == 10000.0)  //search for a value of 10000.0 which
flags an open location if not foundRoom yet

      {

       nextOpen=i; //when found, record the location in nextOpen and
break out of loop

       foundRoom=true; //set flag to say I found an empty location in the
gage block array

      }

    }

    if(!foundRoom)  //if no room was found in the entire array give error
message

    {

     // if there are no empty location, print error message and exit
calibration mode

      lcd.clear();

      lcd.print ("  Calibration");
```

```
      lcd.setCursor(0,1);

      lcd.print ("  array full!");

      delay(5000);

      command = 6;//change command from calibrationMode (1) to no
command active (6)

    }



  } // else found empty location to store newest gage block data or hit
full array and set command to 6 (none)



  if(foundRoom) //if foundRoom store value in array

  {

    //store_value:

    //now have a place to store recently measured gage block: nextOpen

    gage_blocks[nextOpen] = gageBlock;

    actual_results[nextOpen] = actual;

  }
```

```
 }//[e]get next gage block measurement if command = 1
(calibrationMode



 //Serial.println("12");



 //incertion sort in ascending order when user wants to leave
calibration mode

Sort:

 if(command ==6)  //[f] if command set to none, do sort table and then
return from calibrationMode

 {

  for (j=0; i<10; j++) //sort entire table even though some entries will
equal 10000 which means they are empty

  {

   k = j; //start by assuming that element j is the smallest. j is the start of
the unsorted part of the array



   for (i=k+1;i<10;i++)//look through entire array above entry k and
find smallest entry;

   {
```

```
      if
        (actual_results[k] > actual_results[i])
      {
        k=i; //entry k is less than entry i
      }
    }
    //k points to smallest entry found between k+1 and end of array
    keyActual = actual_results[k]; //save smallest entry; shift all lower
entries up one
    //starting at entry k and working down to entry j
    keyBlock = gage_blocks[k]; //shadow moves within the actual_results
array in gage_blocks array


    for (i=k; i>j; i--) //counting down from k to j+1 and shift elements up
one
    {
      actual_results[i] = actual_results[i-1]; //last element to move up is i-
1 when i = j+1 so
```

```
      //we move element j up one so elemement j and j+1 are the same
right now

      gage_blocks[i] = gage_blocks[i-1]; //last element to move up is i-1
when i = j+1 so

      //we move element j up one so elemement j and j+1 are the same
right now

    }

    actual_results[j] = keyActual; //put keyActual into location j.
Elements 0 through j+1 now in ascending order

    gage_blocks[j] = keyBlock; //put keyBlock into location j. Elements 0
through j+1 now in ascending order



  }//advance to next entry during sort routine

 } //end of sorting routine



 //give user final review of table

 //Serial.println("99");
```

```
lcd.clear();

lcd.print("Review of values");

lcd.setCursor(0,1);//move to bottom line

lcd.print("inch/mm = reject");

delay(2000);

changeMade = false;//init flag

fetchReading();//read current state of units

oldUnits = units;


for (i = 0;gage_blocks[i] != 10000;i++){

  //Serial.print("gage_blocks[");

  //Serial.print(i);

  //Serial.println("]= ");

  //Serial.println(gage_blocks[i]);

}
```

```
for (ii = 0;gage_blocks[ii] != 10000;ii++)
{

  //Serial.print("56 ii = ");//q12
  //Serial.println(ii);


  setMilli_t0();//set t=0
  milliNow();//initialize milliDelta

  lcd.clear();
  lcd.print("Block & actual: ");
  //lcd.print(ii);
  //lcd.print(":");
  lcd.setCursor(0,1);//move to bottom line
```

```
  lcd.print(gage_blocks[ii]/1000,3);//display in inches out to thou

  lcd.print("\"  ");

  lcd.print(actual_results[ii]/1000,4);//display in inches out to thou

  lcd.print("\"");

  //lcd.print(" inches");



  while (milliDelta < 3000){



   //Serial.print("56 milliDelta = ");//q12

   //Serial.println(milliDelta);



   fetchReading();//wait for 2 seconds to pass or user pushes inch/mm
button



   //Serial.print("899 units = ");

   //Serial.println(units);
```

```
    //Serial.print("899 oldUnits =  ");

    //Serial.println(oldUnits);




    if(oldUnits != units){ // if user pushes inch/mm button

     changeMade = true;//set flag

     gage_blocks[ii] = 10000;//clear the entry

     lcd.clear();

     lcd.print("Entry removed.");

     lcd.setCursor(0,1);

     lcd.print("Push inch/mm.");

     while (oldUnits != units){//this gives me user timed display plus
puts units back to oldUnits

      fetchReading();

     }



    }
```

```
   milliNow();//get updated milliDelta time

  }//loop to evaluate one entry

}//loop to inspect table


if(changeMade){//if a change was made, resort table and review again

  goto Sort;

}//otherwise, we are done



//Serial.println("13");

lcd.clear();

lcd.print ("End Calibration.");

lcd.setCursor(0,1);

lcd.print((char)0b01111110); //this is ->>

lcd.print("Unlock7");
```

```
  //Serial.println(" Unlock7");



  delay(1000);

  //Serial.println("14");

  command = 6;//set command flag to say we are not in calibrationMode
anymore

} //end of calibrationMode()



void factor()//input qaz (int) and output edc (int) which is the largest
multiple of 2 that fits evenly

{

  if (qaz == 0){

    edc = 1;//if qaz is zero, dividing it by 1 is harmless
```

```
  return;

 }//can't factor zero



 j = 1;

 do {

  j = j*2;

  rfv = float(qaz)/(j); //rfv is float

  edc = qaz/(j);//edc is integer; they will be equal as long as both divide evenly by 2

 }

 while (rfv == float(edc));//insure that both variables are float

 //if not equal on first pass, then most common multiple of 2 should be 1, not 2



 edc = j/2;//edc equals largest multiple of 2 that goes evenly into qaz; divide by 2 because j is doubled before result tested

 //when test fails, we need to crank j back to previous value

}
```

void gonogo()//Go/No-Go test

//If the function is inactive, entering it will enable go/no go. If the function is active, entering will disable it.

// When disabled and entered:

// 1. user asked if test is on OD or ID; enable only OD or ID measurements

// 2. prompted for lower limit and store as lowerLimit

// 3. prompt for upper limit and store as upperLimit

// 4. gonogoFlag is the flag that tells displayAnswer to print "Under", "OK", or "Over" in bottom left corner of display.

// Top and bottom right do not change

```
// When enabled and entered:
// 1. user told function will now be disabled.
// 2.restore original value of odID
{
 lcd.clear();
 lcd.print("Go/No-Go");
 lcd.setCursor(0,1);
 if(gonogoFlag){//toggle flag to opposite state
  gonogoFlag = false;//if we were in go/no go, then user wants to exit
  lcd.print("will be off.");
  odID = oldodID;//restore old odID value
  delay(2000);
  lcd.clear();
  return;
 }
```

```
  //else, we were not in go/no go, so user want to enter
  gonogoFlag = true;
  lcd.print("will be on.");
  oldodID = odID;//save current odID value
  delay(2000);


getLimits:
  lcd.clear();
  lcd.print("Test ID?"); //odID = 1 is OD only; = 2 means ID only
  lcd.setCursor(0,1);
  lcd.print(" yes = inch/mm");
  setMilli_t0();
  milliNow();//initialize milliDelta
  fetchReading();
  gng_oldUnits = units;
  while (milliDelta < 3000){//wait for 2 seconds to pass or user pushes
inch/mm button to signify they want to measure ID
```

```
  fetchReading();

  if(gng_oldUnits != units){ // if user pushes inch/mm button

   odID = onlyID;//set flag for ID

   lcd.clear();

   lcd.print("ID selected.");

   delay(3000);

  }

  milliNow();//get updated milliDelta time

 }

 if (gng_oldUnits == units){//if user did not push inch/mm within 2
seconds, it means they want to measure OD

  odID = onlyOD;//set flag for OD

  lcd.clear();

  lcd.print("OD selected.");

  delay(2000);

 }
```

```
 insureInches();//if caliper not set to inches, user asked to change it and program waits until it is in inches



  //ready to measure limits; do lower limit first

 lcd.clear();

 lcd.print("Get lower limit:");

 odIDread();//read either OD or ID and return with oldResult

 lowerLimit = oldResult;

 lcd.clear();//display measured lower limit

 lcd.print("Lower limit:");

 lcd.setCursor(0,1);

 lcd.print(lowerLimit/1000,4);

 lcd.print(" inches");

 delay(2000);



 //ready to measure upper limit

 lcd.clear();
```

```
lcd.print("Get upper limit:");
odIDread();//read either OD or ID and return with oldResult
upperLimit = oldResult;
lcd.clear();
lcd.print("Upper limit:");
lcd.setCursor(0,1);
lcd.print(upperLimit/1000,4);//display upper limit
lcd.print(" inches");
delay(3000);
if (upperLimit < lowerLimit){
  tempLimit = upperLimit;//if limits out of order, swap them silently
  upperLimit = lowerLimit;
  lowerLimit = tempLimit;
  lcd.clear();
  lcd.print("Limits were");
  lcd.setCursor(0,1);
```

```
  lcd.print("out of order.");
  delay(2000);
}

lcd.clear();
lcd.print("Review limits:");
lcd.setCursor(0,1);
lcd.print(lowerLimit/1000,4);
lcd.print(" ");
lcd.print(upperLimit/1000,4);
delay(3000);

qaz = units;//save present state of units

  lcd.clear();
lcd.print("If not OK, push");
```

```
 lcd.setCursor(0,1);

 lcd.print("inch/mm now.");



 setMilli_t0();

 milliNow();//initialize milliDelta

 while(units == qaz && milliDelta < 2000){

  fetchReading();

  milliNow();//update milliDelta

 }

 if(units == qaz){//if we left the above look because we timed out, then
just exit

  lcd.clear();

  lcd.print("Go/No-Go ready.");

  command = 6;

 }

 else
```

```
 {

   goto getLimits;

 }



}//end of gonogo()
```

```
void odIDread()//this functon contains most of the calls from loop() but used by go/no go function

{

  initCommandPortal();//sets command to 6 and sets oldUnits to current value. Is used by next function

  if(odID == onlyOD){//then it is OD

    displayRadiusAndWait();//caliper jaws have to close by Limit before we start to look for nextlocal minimum; also look for command to run.
```

```
   //after command runs, it will return here.

   startODmeasPrompt(); //ID measurement persists on display until caliper jaws start to close. When they stop closing

   //and start opening, we lock in OD measurement and replace ID with OD

   noReadyPrompt = false;//tells next function to output "to meas." prompt

   initCommandPortal();//sets command to 6 and sets oldUnits to current value. Is used by next function

   localMinRead(); //prompt user, monitors caliper for local minimum and detect a possible push of inch/mm which means

   //command will be executed. When done, we will return to this point in the loop; output is oldResul

   interpolation(); //improve accuracy of caliper reading using gage blocks; input and output are oldResult; pass through if data not present

  }



  initCommandPortal();//sets command to 6 and sets oldUnits to current value. Is used by next function

  if(odID == onlyID){//then it is ID
```

displayRadiusAndWait();//caliper jaws have to open by Limit before we start to look for local maximum; also looking for command to run.

//after command runs, it will return here.

startIDmeasPrompt(); //OD measurement persists on display until caliper jaws start to open. When they stop opening

//and start closing, we lock in ID measurement and replace OD with ID

noReadyPrompt = false;//tells next function to output "to meas." prompt

initCommandPortal();//sets command to 6 and sets oldUnits to current value. Is used by next function

localMaxRead(); //monitor caliper for local maximum; then signal we are ready for new local minimum; also looking for command to run.

//after command runs, it will return here.

interpolation(); //improve accuracy of caliper reading using gage blocks; input and output are oldResult; pass through if data not present

//displayAnswer(); //display best reading; input is oldResult and no output to program

//hold displayed result until user signals for a new reading or signals to enter calibration mode

```
  //when user has signaled that they are done with last reading. See if they want to enter calibration mode or take another reading

 }

}
```

Acknowledgments

I welcome your comments and questions.

If you wish to be contacted each time I publish an article, email me with just "Article Alias" in the subject line.

Rick Sparber
Rgsparber.ha@gmail.com
Rick.Sparber.org