```
/*  This code is for the DSAM analyzer of the gas hydrates facility.
    For John Pohlman and Emile Bergeron.
    It is an interface to receive button actuation and translate
    button sequences into LED indicator lights and output levels
    to actuate valve and analysis sequences.

    Note that the logic here is often reversed because of the
    way the circuitry in the DSAM works.  This can be confusing.
    In the variables which track button, LED and output states, a bit = 0 = LOW and ON or ACTION.
        A bit = 1 = HIGH and OFF or NO ACTION.
    For the digital write statements for the LED pins that indicate which buttons were pressed,
        HIGH written to the pin turns ON the LED.
    For the digital write statements for the Actuation pins that control the DSAM processes,
        LOW written to the pin causes ACTION, activates a process, or initates a valve motion
    For button presses, LOW, or falling edge, detected at the button min means a button press.

    Written by Marinna Martini for the USGS in Woods Hole, 12/20/2016
*/

// MM 12/22/2016 make a change to the B Load output pinsToSet value

const float version = 0.1;
// this code does not do anything about button bouncing except to use delays in places
// if this setting below fails to work we will need to use interrupts or a more
// sophisticated debounce technique
const int buttonBounceDelay = 500;
// we will also represent the pin sets as an array so we can loop and point to them
// as an array, length 16 to match a 16 bit word we will use to mask and detect pin states
const int actionIdx = 7; // where the action pins start in the button and LED arrays
const int nActions = 3;
const int modeIdx = 10; // where the mode pins start in the button and LED arrays
const int nModes = 4;
const int optIdx = 14; // where the option pins start in the button and LED arrays
const int nOptions = 2;
// bit positions in buttonWord, LEDWord and pinsToSet, counting from the left (MSB)
const int purcgBit = 7;
const int loadBit = 8;
const int analyzeBit = 9;
const int discreteBit = 10;
const int septumBit = 11;
const int loopABit = 12;
const int loopBBit = 13;
const int diluteBit = 14;
const int standardBit = 15;
// define the pins that are in each pin set, action, mode and option as one array
// a zero value in these arrays means no pin defined - more pins might be added later.
// pins are listed in order as 7 not used, 3 action pins, 4 mode pins and 2 option pins
// note that the EXP panel button/DSAM function is not controlled by this program
int buttonPins[16] = {0, 0, 0, 0, 0, 0, 0, 22, 23, 24, 25, 26, 27, 28, 29, 37};
// the LED pins match the order of the buttons, since the LEDs will indicate a press
// indicator pins, HIGH will mean an LED is ON
int LEDPins[16] =    {0, 0, 0, 0, 0,  0,  0,  36, 35, 34, 33, 32, 31, 30, 53, 52};

// the output pins - array length of 16 to match a word = 16 bits
int outPins[16] =    {0, 0, 0, 0, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40};

// Endian translation array - e.g. so we don't have to count or do math backwards
// my pin array numbering starts at MSB, bitwise calls by arduino start at LSB
int lsb[] = {15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0};

// the lookup table of output settings.  These numbers are calculated
// in the Excel spreadsheet DSAMLookupDefinitions.xlsx, based on a table worked out by Emile
// if one examines the binary for these variables, 1 = LED OFF or NO ACTION, 0 = ACTION or LED ON
// it is also important that the variables be declared as unsigned (which is the definition of
```

```
word)
// One Step:  this group of pin settings control processes that take only one step to accomplish
word buttonSetsOneStep[] = {65247, 65263, 65271, 65275, 65375, 65391, 65399, 65403, 65439,
65455, 65463, 65467};
word setPinsOneStep[] =    {64170, 64170, 63914, 64170, 63056, 64080, 64144, 63888, 64064,
64098, 63841, 64097};
// Dilute:  this group of pin settings controls dilute functions, which have intermediate steps
// word buttonSetsDilute[] = {65438, 65454, 65462, 65466};
// these are in the order of {discrete, septum, A, B}
word setPinsDilute1[] = {64170, 64170, 63914, 64170}; // dilute process, 2 sec delay
word setPinsDilute2[] = {64064, 64098, 63841, 64097}; // analyze process after dilutes
const int diluteDelay = 2000; // 2 sec
// Standard:  There is only one "standard" process
const word buttonSetsStandard = 65533;
const word setPinsStandard = 64140;
// we need a startup mode, as if buttons are pressed
const word buttonSetStartup = 65399; // Load Loop A
const word setPinsStartup = 64144; // Load Loop A

// define the word to hold the pin settings
// note that we have nine, so a word, not a byte, is needed
word buttonWord = 65535; // bitwise flag array to store state of the buttons
word lastbuttonState = 65535; // previous button state, use to return from dilute
word LEDWord = 0; // indicator lights
word pinsToSet = 0;  // output or actuation pins
// counters and flags
int ipin = 0;
int i = 0;
int buttonState = 0;

void setup() {
    // put your setup code here, to run once:
    // initate serial output for diagnostic feedback
    Serial.begin(9600);
    Serial.print("DSAM Control Program Version ");
    Serial.println(version,2);

    // initiate input and output pin modes and states
    for (ipin = actionIdx; ipin < 16; ipin++) {
        // note when pinmode is called as input, Arduino sources current on that pin
        pinMode(buttonPins[ipin], INPUT);
        // set inputs high so we don't have floating inputs
        // thus, falling edge on a button pin means pressed.
        digitalWrite(buttonPins[ipin], HIGH);  // e.g. no buttons have been pressed yet
        bitWrite(buttonWord, lsb[ipin], HIGH);
        pinMode(LEDPins[ipin], OUTPUT);
        // make sure all indicator LEDs are OFF
        // the way the indicator LEDs are wired on the test unit, LOW is OFF
        digitalWrite(LEDPins[ipin], LOW); // LOW pin setting = LED off
        bitWrite(LEDWord, lsb[ipin], HIGH);  // HIGH bit stored = LED off
    }
    for (ipin = 4; ipin < 16; ipin++) {
        // set actuation output pins
        pinMode(outPins[ipin], OUTPUT);
        // set outputs high so we don't have floating outputs
        // thus, falling edge on and output pin means actuation.
        digitalWrite(outPins[ipin], HIGH);
        bitWrite(pinsToSet, lsb[ipin], HIGH);  // HIGH bit stored = NO ACTION
    }

    // diagnostic
    delay(2000);
    Serial.println("End of setup");
```

```
    // we want to start gracefully, request is in load loop A
    Serial.println("Starting Mode is Load Loop A");
    digitalWrite(LEDPins[loadBit], HIGH);
    digitalWrite(LEDPins[loopABit], HIGH);
    buttonWord = buttonSetStartup;
    LEDWord = buttonSetStartup;  // these mirror each other
    pinsToSet = setPinsStartup;
    displayStates(buttonWord, pinsToSet);
    setOutputPins(pinsToSet);
}

void loop() {
  // we wait for a button push
  // in testing this loop has been fast enough to not need interrupts,
  // so we start with the simpler button detect method:
  // just loop through all the input buttons and catch one
  // note the delays in this code are to deal with button bounce
  for (ipin = actionIdx; ipin < 16; ipin++) {

    buttonState = digitalRead(buttonPins[ipin]);
    lastbuttonState = buttonState;  // save for graceful recoveries

    if (buttonState == LOW) {

        // button detected, which group got pressed - Action, Mode, Dilute or Standard?

        // standard is special, need to check here and make sure its LED and bit get turned off
        // if another button was pressed after standard, because we are no longer in standard.
        if (!bitRead(LEDWord, lsb[standardBit])) {
            digitalWrite(LEDPins[standardBit], LOW);  // turn OFF LED
            bitWrite(LEDWord, lsb[standardBit], HIGH);  // remember we turned it off, HIGH bit
            = LED is OFF
            bitWrite(buttonWord, lsb[standardBit], HIGH);  // remember we turned it off, HIGH
            bit = NO ACTION
        }

        if (ipin >= actionIdx && ipin < actionIdx + nActions + nModes) { // Action or Mode
            // TODO simplify - the only difference between Action and Mode are the indeces that
            refer to the array of pin numbers
            if (ipin >= actionIdx && ipin < actionIdx + nActions) { // Action = Purge, Load,
            Analyze
              // turn on corresponding LED - right away so user gets the feedback
              // buttonWord LOW means button was activated
              digitalWrite(LEDPins[ipin], HIGH);
              bitWrite(LEDWord, lsb[ipin], LOW);
              bitWrite(buttonWord, lsb[ipin], LOW);
              for (i = actionIdx; i < actionIdx + nActions; i++) {
                // turn off the other LEDs in the group that weren't pushed
                if (i != ipin) {
                  digitalWrite(LEDPins[i], LOW);
                  // bring the flag bit high = OFF or NO ACTION
                  bitWrite(buttonWord, lsb[i], HIGH);
                  bitWrite(LEDWord, lsb[i], HIGH);
                }
              }
              // now we should have a buttonWord we can use to decide how the outputs should be
              set
              Serial.println("Pushed Action");
              // find the command set to use
              for (i = 0; i < 12; i++) {
                if (buttonWord == buttonSetsOneStep[i]) {
                  pinsToSet = setPinsOneStep[i];
                }
              }
```

```
          displayStates(buttonWord, pinsToSet);
          setOutputPins(pinsToSet);
          delay(buttonBounceDelay);
        }
        else if (ipin >= modeIdx && ipin < modeIdx + nModes) { // Mode = Sept., Disc, A, B
          // turn on corresponding LED - right away so user gets the feedback
          digitalWrite(LEDPins[ipin], HIGH);
          bitWrite(LEDWord, lsb[ipin], LOW);
          bitWrite(buttonWord, lsb[ipin], LOW);
          for (i = modeIdx; i < modeIdx + nModes; i++) {
            // turn off the other LEDs in the group that weren't pushed
            if (i != ipin) {
              digitalWrite(LEDPins[i], LOW);
              // bring the flag bit high = OFF or NO ACTION
              bitWrite(buttonWord, lsb[i], HIGH);
              bitWrite(LEDWord, lsb[i], HIGH);
            }
          }
        }
        // now we should have a buttonWord we can use to decide how the outputs should be set
        Serial.println("Pushed Mode");
        // find the command set to use
        for (i = 0; i < 12; i++) {
          if (buttonWord == buttonSetsOneStep[i]) {
            pinsToSet = setPinsOneStep[i];
          }
        }
        displayStates(buttonWord, pinsToSet);
        setOutputPins(pinsToSet);
        delay(buttonBounceDelay);
    }
    else if (ipin == standardBit) {
        // standard overrides and turns off all other bits and LEDs
        // turn on corresponding LED - right away so user gets the feedback
        digitalWrite(LEDPins[standardBit], HIGH);
        bitWrite(LEDWord, lsb[standardBit], LOW);
        bitWrite(buttonWord, lsb[standardBit], LOW);
        for (i = 0; i < 16; i++) {
          // turn off all the other LEDs for Action, Mode and Option
          if (i != standardBit) {
            digitalWrite(LEDPins[i], LOW);
            // bring the flag bit high = OFF or NO ACTION
             bitWrite(buttonWord, lsb[i], HIGH);
              bitWrite(LEDWord, lsb[i], HIGH);
          }
        }
        // now we should have a buttonWord we can use to decide how the outputs should be set
        Serial.println("Pushed Standard");
        displayStates(buttonWord, setPinsStandard);
        setOutputPins(setPinsStandard);
        delay(buttonBounceDelay);
    }
    else if (ipin == diluteBit) {
        // dilute can only execute if analyze in ON, and then it starts a sequence
        // of events that returns to the original set state of the machine
        // if analyze bit = 0, we are in analyze mode, OK to dilute
        if (!bitRead(buttonWord, lsb[analyzeBit])) {
            Serial.print("In analyze mode, will commence dilution ");
            // which mode?  {discrete, septum, A, B}
            if (!bitRead(buttonWord, lsb[discreteBit])) {
                Serial.println("for discrete");
                i = 0;
            }
            else if (!bitRead(buttonWord, lsb[septumBit])) {
```

```
                    Serial.println("for septum");
                    i = 1;
                }
                else if (!bitRead(buttonWord, lsb[loopABit])) {
                    Serial.println("for loop A");
                    i = 2;
                }
                else if (!bitRead(buttonWord, lsb[loopBBit])) {
                    Serial.println("for loop B");
                    i = 3;
                }
                else { // just in case...
                    Serial.println("???");
                    i = -1;
                }
                if (i >= 0) {
                    pinsToSet = setPinsDilute1[i];
                    displayStates(buttonWord, pinsToSet);
                    setOutputPins(pinsToSet);
                    delay(diluteDelay); // the delay for dlute is plenty to absorb button bounces
                    Serial.println("now to analyze");
                    pinsToSet = setPinsDilute2[i];
                    displayStates(buttonWord, pinsToSet);
                    setOutputPins(pinsToSet);
                    delay(buttonBounceDelay); // not sure we need a delay here
                    Serial.println("Finished dilute process sequence");
                }
            } else {
                Serial.println("dilute pressed when NOT in analyze mode, taking no action");
                buttonState = lastbuttonState;  // nothing will happen here, just restoring
                delay(buttonBounceDelay);
            }
        } // end of decision tree for what process we are doing
    } // end of button == LOW
  } // end of scan pins for button change
} // end of loop()


// actually set the digital pins
// TODO - make the scope of the variables here the same (e.g. pass in outPins, or make pinWord
global)
void setOutputPins(word pinWord) {
    int bit = 0;

    for (int b = 0; b < 16; b++) {
        int bitValue = bitRead(pinWord, b);
        // note that the contents of outPins[lsb[b]] will be 0 if there's no pin to set
        if (bitValue && outPins[lsb[b]]) digitalWrite(outPins[lsb[b]], HIGH);
        else digitalWrite(outPins[lsb[b]], LOW); // LOW = Activate
    }
}

// diagnostic information to console
void displayStates(word input, word output) {
    Serial.print("input ");
    Serial.print(input, BIN);
    Serial.print(" ");
    Serial.print(input, DEC);
    Serial.print(" set pins ");
    Serial.print(output, BIN);
    Serial.print(" ");
    Serial.println(output, DEC);
}
```