```
#ifndef MATRICES_PER_DISPLAY
#include "ff_scoreboard_properties.h"
#endif

void setMuxingPinsLow() {

  for (int i = 0; i < PIXELS_PER_MATRIX; i++) {
    pinMode(pixelPin[i], OUTPUT);
    digitalWrite(pixelPin[i], LOW);
  }
  for (int j = 0; j < MATRICES_PER_DISPLAY; j++) {
    pinMode(matrixPin[j], OUTPUT);
    digitalWrite(matrixPin[j], LOW);
  }

}
// check that the format (+WX:YZ) is present, where W, X, Y, and Z are any value
boolean inputStringValid(String theInput) {

  boolean isValid = false;

  if (theInput.startsWith("+")) {
    if (theInput.charAt(3) == ':') {
      isValid = true;
    }
  } else {
    isValid = false;
  }

  return isValid;
}

// check that the values W, X, Y, and Z of string "+WX:YZ" contains valid indices within
glyphs[]
// where W, X, Y, and Z are expected to be any value between 0 and (GLYPHS_COUNT-1)
inclusive
// if invalid value anywhere, return false
boolean timeStringValid(String theInput) {

  int indexToTest = -1; // an invalid index to start; should be assigned in loop
  boolean isValid = true; // assume valid until proved invalid

  for (int pos = numberIndexInString[0]; pos <
numberIndexInString[(MATRICES_PER_DISPLAY-1)]; pos++) { // checks positions 1, 2, 4, 5

    indexToTest = theInput.charAt(pos) - '0';

    if (pos == 3) { // skip position 3
      pos++;
    }

    // test: is array index not gt/eq 0, and not lt/eq array length
    //                                    (array length is invalid index, must be less)
    if (!(0 <= indexToTest) && !(indexToTest <= GLYPHS_COUNT)) {
      isValid = false;
      return false;
    }
  }
  if (isValid == true) { // safeguard case that isValid == false but somehow execution has
reached here
    return true;
  } else { // if not good string, and this point has been reached, return false
    return false;
  }
}

// sets an index in an array with the glyph index that a matrix will take on
void setGlyphIndex(int *glyphIndex, String theString, int pos) {
  int value = (theString.charAt(pos) - '0');
  if ((0 <= value)  && (value < GLYPHS_COUNT)) {
    *glyphIndex = value;
  }
  else {  // error character: GLYPH_ALL_ON (so, 11). should not be possible if 
timeStringValid(inputString) == true
    *glyphIndex = 11;
  }

  if (DEBUG) {
    Serial.print(value);
    Serial.print(" ");
    Serial.println(pos);
  }
}

void drawDigitalDisplay(int *arrayOfValues) {
  for (int matrix = 0; matrix < MATRICES_PER_DISPLAY; matrix++) { // per-matrix drawing 
loop
    digitalWrite(matrixPin[matrix], HIGH); // disable the current matrix, comment out to 
watch pixels blink out
    for (int pixel = 0; pixel < PIXELS_PER_MATRIX; pixel++) { // per-pixel deactivation 
loop
      digitalWrite(pixelPin[pixel], LOW); // Disable pixel by pixel, clean slate
    }
    for (int i = 0; i < 10; i++) {
      delayMicroseconds(REFRESH_WAIT);
      for (int pixel = 0; pixel < PIXELS_PER_MATRIX; pixel++) { // per-pixel drawing loop
        // This is a pixel-light-setting loop to enable new matrix lights.
        if (matrix == 0) { // if 'plexing tens of hours
          digitalWrite(pixelPin[pixel], ((glyphs[arrayOfValues[0]] >> pixel) & 0x1));
        }
        if (matrix == 1) { // if 'plexing ones of hours
          digitalWrite(pixelPin[pixel], ((glyphs[arrayOfValues[1]] >> pixel) & 0x1));
        }
        if (matrix == 2) { // if 'plexing tens of minutes
          digitalWrite(pixelPin[pixel], ((glyphs[arrayOfValues[2]] >> pixel) & 0x1));
        }
        if (matrix == 3) { // if 'plexing ones of minutes
          digitalWrite(pixelPin[pixel], ((glyphs[arrayOfValues[3]] >> pixel) & 0x1));
        }
        if (DEBUG && VERBOSE) {
          Serial.print("pixel ");
          Serial.print(pixel);
          Serial.print(" matrix ");
          Serial.print(matrix);
          Serial.print(" numeral ");
          Serial.println(i);
        }
      }
    }
    digitalWrite(matrixPin[matrix], LOW); // disable the current matrix, comment out to 
watch pixels blink out

    if (DEBUG && VERBOSE) {
      Serial.print("Pin ");
      Serial.print(matrix);
      Serial.println(" reconfigured.");
    }
  }
}
```