```
// number of microseconds to wait before refreshing all matrices
// microseconds. 4ms = <62.5Hz/display, 25% duty cycle
#define REFRESH_WAIT 400

// Enable serial/UART print statements
#define DEBUG true
// Enable even more statements
#define VERBOSE false

#define PIXELS_PER_MATRIX 22
#define MATRICES_PER_DISPLAY 4

const int pixelPin[PIXELS_PER_MATRIX] = {28, 27, 26, 25, 2, 10, 8, 5, 31, 32, 33, 34, 35,
36, 37, 39, 38, 40, 11, 12, 13, 17};
const int matrixPin[MATRICES_PER_DISPLAY] = {14, 15, 18, 19};
int glyphIndex[MATRICES_PER_DISPLAY] = {10, 10, 10, 10}; // set time by reading serial
const int numberIndexInString[MATRICES_PER_DISPLAY] = {1, 2, 4, 5}; // W, X, Y, Z in
"+WX:YZ"

// ref:
https://cdn.discordapp.com/attachments/882335631102079006/896048973012402226/image0.jpg
// (B00010011 << 16) + (B01100100 << 8) + B11010110;
#define GLYPH_ZERO  0x1364d6
// (B00010100 << 16) + (B00010010 << 8) + B00100100;
#define GLYPH_ONE   0x141224
// (B00111010 << 16) + (B10010000 << 8) + B01010110;
#define GLYPH_TWO   0x3a9056
// (B00010011 << 16) + (B00010000 << 8) + B01010110;
#define GLYPH_THREE 0x131056
// (B00100001 << 16) + (B11100101 << 8) + B01101000;
#define GLYPH_FOUR  0x21c568
// (B00010011 << 16) + (B00100011 << 8) + B10011111;
#define GLYPH_FIVE  0x13239f
// (B00010011 << 16) + (B01011100 << 8) + B10010110;
#define GLYPH_SIX   0x135c96
// (B00001010 << 16) + (B10010000 << 8) + B01001111;
#define GLYPH_SEVEN 0x0a904f
// (B00010011 << 16) + (B01011000 << 8) + B11010110;
#define GLYPH_EIGHT 0x1358d6
// (B00010001 << 16) + (B00111000 << 8) + B11010110;
#define GLYPH_NINE  0x1138d6
// (B00000000 << 16) + (B00000000 << 8) + B00000000;
#define GLYPH_ALL_OFF 0x000000
// (B00111111 << 16) + (B11111111 << 8) + B11111111;
#define GLYPH_ALL_ON  0x3fffff
/////////
#define GLYPH_S
#define GLYPH_E
#define GLYPH_T
#define GLYPH_EX_PT
// add 'S', 'E', 'T', '!', set glyphs_count = 16
// if adding/removing glyphs, be sure to update GLYPHS_COUNT
// Number of glyphs encoded for display on any given matrix
#define GLYPHS_COUNT 12

const uint32_t glyphs[GLYPHS_COUNT] = {GLYPH_ZERO, GLYPH_ONE, GLYPH_TWO, GLYPH_THREE,
GLYPH_FOUR, GLYPH_FIVE, GLYPH_SIX, GLYPH_SEVEN, GLYPH_EIGHT, GLYPH_NINE, GLYPH_ALL_OFF,
GLYPH_ALL_ON}; // ess, eee, tee);
```

```
#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.");
    }
  }
}
```

```
#include "helper_functions.h"

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


// a string to hold incoming data
// the following must be global due to the usage of the serialEvent() function
String inputString = "";
char outputString[100];
boolean stringComplete = false; // whether the string is complete

extern const int pixelPin[];
extern const int matrixPin[];

 // set time by reading serial
extern int glyphIndex[];
// determines location of time digits within input string
// maps 1:1 with glyphIndex
extern const int numberIndexInString[];
// binary mappings of lit and unlit pixels for display.
// 22 bits map to 28 pixels
extern const uint32_t glyphs[];

void setup() {
  // put your setup code here, to run once:

  Serial.begin(115200);
  if (DEBUG) { Serial.println("Serial connection initialized."); }

  // Set all pins in use to low at start, to start with a blank slate
  setMuxingPinsLow();

}

void loop() {
  // put your main code here, to run repeatedly:

  if (stringComplete) {

    if (DEBUG) {
      if (VERBOSE) { Serial.println("Reached top of loop"); }
      Serial.println(inputString);
    }

    stringComplete = (inputStringValid(inputString) && timeStringValid(inputString));
    // now string resembles '+[0-:][0-:]:[0-:][0-:]\n'
    if (stringComplete == true) {
      setGlyphIndex(&glyphIndex[0], inputString, 1);
      setGlyphIndex(&glyphIndex[1], inputString, 2);
      setGlyphIndex(&glyphIndex[2], inputString, 4);
      setGlyphIndex(&glyphIndex[3], inputString, 5);
    }

    if (DEBUG) {
      sprintf(outputString, "Setting matrices to show %d%d:%d%d given ", glyphIndex[0],
glyphIndex[1], glyphIndex[2], glyphIndex[3]);
      Serial.println((outputString+inputString));
      Serial.println();
    }
    // clear the string:
    inputString = "";
    stringComplete = false;
  }

  drawDigitalDisplay(glyphIndex);
}
```

```
// serial polled only if this interrupt handler is written
// serial polled using this handler each time after loop() runs
// needs to be in this main file to run
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```