```
// Import default libraries
#include <Arduino.h>
#include <elapsedMillis.h>
#include <cmath>
#include <iostream>

//Import audio-related libraries
#include <Audio.h>
#include <SD.h>
#include <SPI.h>
#include <Wire.h>

// Import other device libraries
#include "Adafruit_MAX1704X.h" // Ba
#include <Adafruit_MCP23X17.h> // IO
#include <Adafruit_NeoPixel.h> // LE
#include <Adafruit_TPA2016.h> // BC
#include "pindefs.h" // Pin definiti

/************ MESSAGE PACKETS */
union UnderwaterMessage {
    struct {
        uint8_t msg : 3; // 8 bits f
        uint8_t id : 4;  // 8 bits f
    };
    uint8_t data; // 16 bits total (

    static constexpr uint8_t size =
};

/******* ADDITIONAL DEVICE SETUP */

// Declare NeoPixel strip object:
Adafruit_NeoPixel strip(LED_COUNT, L
// GRB color order
uint32_t red = strip.Color(255, 0, 0
uint32_t greenishwhite = strip.Color
uint32_t bluishwhite = strip.Color(0
uint32_t cyan = strip.Color(0, 255,
uint32_t purple = strip.Color(255, 0

// LED blinky anim stuff
int counter = 0;
bool dir = 1;
long lastLEDUpdateTime = 0;
long lastButtonCheckTime = 0;

Adafruit_MCP23X17 mcp; // I/O expand
Adafruit_MAX17048 maxlipo; // Batter
Adafruit_TPA2016 audioamp = Adafruit

// Battery monitor stuff
float lastCellVoltage = 0.0;
float lastPercentage = 100.0; // Ini
unsigned long lastBattCheckTime = 0;
int pctBufPointer = -1;
#define BATTERY_SAMPLES 5
float pctBuf[BATTERY_SAMPLES];

// Underwater message array of messa
UnderwaterMessage UM_array[6];

// IDS: how we identify one device f
// Pointers to user ID as string, an
int user_ID = 1;
int other_user_ID = 2;
static const char *audio_ids_array[1
static const char *user_ids_array[16

// const unsigned int *audio_ids_arr

// Message array, pointers to each m

static const char *message_array[6]
const char *audio_messages_array[6]

// Battery percentage pointers
const char *battery_messages_array[1


/************* AUDIO SHIELD / PIPELI
const int micInput = AUDIO_INPUT_MIC
const int chipSelect = 10;

// SELECT SAMPLE RATE
const uint32_t sampleRate = 44100;
// const uint32_t sampleRate = 96000
// const uint32_t sampleRate = 19200
//const uint32_t sampleRate = 234000

/************** AUDIO OUTPUT CHAIN (
AudioPlaySdWav           playBonecond
AudioAmplifier           outputAmp;
AudioOutputI2S           audioOutput
AudioConnection          patchCord1(
AudioConnection          patchCord2(

/*************** AUDIO INPUT CHAIN (
const int myInput = AUDIO_INPUT_MIC;
AudioControlSGTL5000     audioShield;
AudioInputI2S            audioInput;
AudioAmplifier           inputAmp;
AudioAnalyzeFFT1024      inputFFT;
AudioConnection          patchCord3(
AudioConnection          patchCord4(

/************** SAMPLE BUFFER LOGIC
Each bin is numbered 0-1023 and has
SamplingBuffer is a time-valued arra
*/
#define MESSAGE_BIT_DELAY 250 // ms
#define NUM_SAMPLES ((int)(((MESSAGE
int16_t samplingBuffer[NUM_SAMPLES];
uint16_t samplingPointer = 0; //How
#define MESSAGE_LENGTH UnderwaterMes
bool bitBuffer[MESSAGE_LENGTH]; // M
int bitPointer = 0;
#define FFT_BIN_WIDTH 43.0664
#define MESSAGE_START_FREQ 12500 //
#define MESSAGE_0_FREQ 15000 // Hz
#define MESSAGE_1_FREQ 17500 // Hz
#define MESSAGE_LOWPASS_CUTOFF_FREQ
#define FFT_BIN_CUTOFF (int)(MESSAGE
#define MIN_VALID_AMP 0.05
#define MAX_VALID_AMP 1.5
#define BOUNDS_FREQ 1500
typedef enum {
  LISTENING, //Waiting for start fre
  CHECK_START,
  MESSAGE_ACTIVE //Currently receivi
} RECEIVING_STATE;

RECEIVING_STATE curReceivingState =
bool sampling = 0; // Are we current

/*************** TRANSMITTING LOGIC
const byte toneBufferLength = 10*MES
int toneFreqQueue[toneBufferLength];
unsigned long toneDelayQueue[toneBuf
unsigned long lastToneStart = 0;
```

```
141: byte toneStackPos = 0;
142: unsigned long lastBitChange = 0;
143:
144: /*************** AUDIO LOGIC (QUEUE)
145: // const byte audioBufferLength = 10
146: // int audioFnameQueue[toneBufferLen
147: // unsigned long au[toneBufferLength
148: // unsigned long lastToneStart = 0;
149: // byte toneStackPos = 0;
150: // unsigned long lastBitChange = 0;
151:
152:
153: typedef enum {
154:   RECEIVE,
155:   TRANSMIT,
156:   ERROR
157: } OPERATING_MODE;
158:
159: OPERATING_MODE mode = RECEIVE;
160:
161: void setup() {
162:     Serial.begin(115200);
163:
164:     /******** INITIALIZATION */
165:     /*
166:     Steps:
167:     1) Initialize:
168:         - Blink blue to show that we
169:         - LED strip, battery monitor
170:         - Blink green to show initia
171:         - If anything fails: display
172:     */
173:
174:     // Setup pin modes
175:     pinMode(LED_PIN, OUTPUT);
176:     pinMode(HYDROPHONE_PIN, OUTPUT);
177:     pinMode(RELAY_PIN, OUTPUT);
178:
179:     // Initial pin setup
180:     digitalWrite(RELAY_PIN, LOW);
181:
182:     // Get LEDs up and running
183:     strip.begin();
184:     strip.show(); // Initialize all
185:     strip.setBrightness(LED_BRIGHTNE
186:
187:     // LEDs: show that we are alive
188:     strip.fill(purple, 0, 20); // li
189:     strip.show();
190:     // keep strip on, then continue
191:     delay(500);
192:     strip.clear();
193:
194:     // Initialize SD card
195:     if (!SD.begin(10)) {
196:       Serial.println("[ERROR] SD car
197:       initializationError(1);
198:     }
199:     Serial.println("[OK] SD card ini
200:     for (int i=0; i<6; i++) {
201:       if (!SD.exists(audio_messages_
202:         initializationError(2);
203:       }
204:     }
205:     Serial.println("[OK] Found all a
206:     for (int i=0; i<16; i++) {
207:       if (!SD.exists(audio_ids_array
208:
209:         initializationError(2);
210:       }
211:     }
212:     // Check for ID.txt and read the
213:     if (SD.exists("ID.txt")) {
214:       File idFile = SD.open("ID.txt"
215:       if (idFile) {
216:         String idContent = idFile.re
217:         idContent.trim(); // Remove
218:         user_ID = idContent.toInt();
219:         if (user_ID == 1) {
220:           other_user_ID = 2;
221:         } else {
222:           other_user_ID = 1;
223:         }
224:         Serial.print("[OK] User ID s
225:         Serial.println(user_ID);
226:         idFile.close();
227:       } else {
228:         Serial.println("[ERROR] Fail
229:         initializationError(3);
230:       }
231:     } else {
232:       Serial.println("[ERROR] ID.txt
233:       initializationError(3);
234:     }
235:     initializationPass(2);
236:     Serial.println("[OK] Found all a
237:
238:     // Assign underwater messages
239:     for (int i=1; i<=sizeof(UM_array
240:       UM_array[i] = createUnderwater
241:     }
242:     Serial.println("[OK] Underwater
243:
244:     // Get IO expander up and runnin
245:     if (!mcp.begin_I2C(0x27)) {
246:         Serial.println("[Error] I2C
247:         initializationError(3);
248:     }
249:     //Pinmode for I/O expander pins
250:     for (int i=0; i<6; i++) {
251:       mcp.pinMode(i, INPUT_PULLUP);
252:     }
253:     initializationPass(3);
254:
255:     Serial.println("[OK] I2C expande
256:
257:     if (!maxlipo.begin()) {
258:        Serial.println("[Error] Batte
259:        initializationError(4);
260:     }
261:     Serial.print(F("[OK] Battery mon
262:     Serial.print(F(" with Chip ID: 0
263:     Serial.println(maxlipo.getChipID
264:     maxlipo.setAlertVoltages(2.0, 4.
265:     initializationPass(4);
266:
267:     // Get BC transducer amp up and
268:     if (!audioamp.begin()) {
269:       Serial.println("[Error] TPA201
270:       // initializationError(5);
271:     } else {
272:       audioamp.setGain(30);
273:       Serial.println("[OK] TPA2016 i
274:     }
275:
276:     // Get audio shield up and runni
277:     AudioMemory(500);
278:     inputAmp.gain(2);         // ampl
279:     outputAmp.gain(2);
280:     audioShield.enable();
```

```
281:     audioShield.inputSelect(myInput)
282:     audioShield.micGain(90);
283:     audioShield.volume(1);
284:     // setAudioSampleI2SFreq(sampleR
285:     Serial.printf("[OK] SGTL5000 ini
286:     initializationPass(6);
287:
288:     // Setup receiver state machine,
289:     transitionReceivingState(LISTENI
290:     transitionOperatingMode(RECEIVE)
291:     initializationPass(7);
292:     Serial.println("[OK] State machi
293:
294:     Serial.println("Done initializin
295:     Serial.println("Welcome to NEPTU
296:
297:     int dly = 50; // Initial delay
298:     float speedFactor = 0.95; // Fac
299:
300:     for (int cycle = 0; cycle < 6; c
301:       for (int i = 0; i < strip.numP
302:         strip.clear(); // Clear all
303:
304:         // Illuminate the current pi
305:         strip.setPixelColor(i, strip
306:         if (i + 1 < strip.numPixels(
307:         if (i + 2 < strip.numPixels(
308:
309:         strip.show(); // Update the
310:         delay(dly);   // Pause for t
311:
312:         if (i % 6 == 0) {
313:           // Gradually decrease dela
314:           dly = max(5, (int)(dly * s
315:         }
316:       }
317:     }
318:     strip.clear();
319:     strip.show();
320:
321:     // Play sound effect
322:     playBoneconduct.play("WELCOME.wa
323:     while (playBoneconduct.isPlaying
324:     playBoneconduct.play("user.wav")
325:     while (playBoneconduct.isPlaying
326:     playBoneconduct.play(audio_ids_a
327:     while (playBoneconduct.isPlaying
328: }
329:
330: unsigned long lastPlayedBatteryTime
331:
332: void loop() {
333:   if (toneStackPos == 0) {
334:     strip.clear(); // Set all pixel
335:     transitionOperatingMode(RECEIVE)
336:   }
337:
338:    if (millis() - lastBattCheckTime
339:     lastBattCheckTime = millis();
340:
341:     float cellVoltage = maxlipo.cell
342:     if (isnan(cellVoltage)) {
343:       Serial.println("Failed to read
344:     } else {
345:       float cellPercent = maxlipo.ce
346:       Serial.print("Battery V:");
347:       Serial.println(cellPercent);
348:
349:       if (pctBufPointer == -1) { //F
350:         for (int i=0; i<BATTERY_SAMP
351:           pctBuf[i] = maxlipo.cellVo
352:         }
353:         pctBufPointer = 1; //Increme
354:       } else {
355:         pctBuf[pctBufPointer] = maxl
356:       }
357:       pctBufPointer++;
358:       if (pctBufPointer >= BATTERY_S
359:         pctBufPointer = 0;
360:       }
361:
362:       float totPctBuf = 0.0;
363:       for (int i = 0; i < BATTERY_SA
364:         totPctBuf += pctBuf[i];
365:       }
366:
367:       float avgPctBuf = totPctBuf /
368:       float bufDifference = maxlipo.
369:
370:       if (millis() - lastPlayedBatte
371:         if (bufDifference > 0.05) {
372:           Serial.println("Neptune ch
373:           playBoneconduct.play("CHAR
374:           while (playBoneconduct.isP
375:         } else if (bufDifference < -
376:           Serial.println("Neptune ch
377:           playBoneconduct.play("CHAR
378:           while (playBoneconduct.isP
379:         }
380:         lastPlayedBatteryTime = mill
381:       }
382:
383:       // Check if the battery percen
384:       if ((int(cellPercent) / 10) <
385:         int percentRange = (int(cell
386:         Serial.print("Battery droppe
387:         Serial.print(percentRange);
388:         Serial.println("% range.");
389:
390:         playBoneconduct.play("BATTER
391:         while (playBoneconduct.isPla
392:         playBoneconduct.play(battery
393:         while (playBoneconduct.isPla
394:
395:         lastPercentage = cellPercent
396:       }
397:     }
398:   }
399:
400:   // LED pulsating effect!
401:   if (millis() > lastLEDUpdateTime)
402:     uint32_t color = strip.Color(0,
403:     if (dir) {
404:       counter++;
405:     } else {
406:       counter--;
407:     }
408:     if (counter <= 0 || counter >= 2
409:     strip.fill(color);
410:     strip.show();
411:     lastLEDUpdateTime = millis() + 1
412:   }
413:
414:   //Deal with tone sending (asynchro
415:   if (toneStackPos > 0) {
416:     if (millis() - lastToneStart > t
417:       // Serial.print("ToneQueue: ")
418:       for (int i=1; i<toneBufferLeng
419:           // Serial.print(toneFreqQu
420:           // Serial.print("Hz@");
```

```
421:            // Serial.print(toneDelayQ
422:            toneFreqQueue[i-1] = toneF
423:            toneDelayQueue[i-1] = tone
424:        }
425:        // Serial.println();
426:        toneStackPos--; //weâ\200\231v
427:        if (toneStackPos > 0) { //is t
428:            if (toneFreqQueue[0] > 0)
429:            tone(HYDROPHONE_PIN, toneF
430:            }
431:            lastToneStart = millis();
432:        } else {
433:            noTone(HYDROPHONE_PIN); //
434:        }
435:      }
436:    }
437:
438:    // FFT has new data! Reads in data
439:    if (inputFFT.available()) {
440:      // each time new FFT data is ava
441:      float maxBinAmp = 0;
442:      int binNumber = 0;
443:      for (int i = 0; i < 1024; i++) {
444:        float n = inputFFT.read(i);
445:        if (n > maxBinAmp && i >= FFT_
446:          maxBinAmp = n;
447:          binNumber = i;
448:        }
449:      }
450:      Serial.print((double)binNumber*(
451:      Serial.print("Hz@");
452:      Serial.println(maxBinAmp);
453:      if (sampling) { // Valid start f
454:        if (validAmplitude(maxBinAmp))
455:          samplingBuffer[samplingPoint
456:          samplingPointer++;
457:          if (samplingPointer >= NUM_S
458:            samplingPointer = 0;
459:          }
460:          // Serial.print("FFT: ");
461:          // Serial.print(binNumber);
462:          // Serial.print(" sampBufDep
463:          // Serial.println(samplingPo
464:        }
465:      }
466:
467:      // We get valid message start to
468:      if (curReceivingState == LISTENI
469:        if (validAmplitude(maxBinAmp)
470:          transitionReceivingState(CHE
471:        }
472:      } else if (curReceivingState ==
473:        if (isSampleBufferValid() && f
474:          Serial.println("SAW VALID ME
475:          transitionReceivingState(MES
476:        } else { // If buffer is not v
477:          transitionReceivingState(LIS
478:        }
479:      } else if (curReceivingState ==
480:        if (isSampleBufferValid()) { /
481:          // Check if 1 or 0 (or neith
482:          double bufferAvgFreq = sampl
483:          if (freqMatchesBounds(buffer
484:            bitBuffer[bitPointer] = 1;
485:          } else if (freqMatchesBounds
486:            bitBuffer[bitPointer] = 0;
487:          }
488:        }
489:
490:        clearSampleBuffer();
491:        lastBitChange = millis() - 5;
492:        bitPointer++;
493:
494:        if (bitPointer >= MESSAGE_LENG
495:          uint8_t data = 0;
496:          // Combine bits in bitBuffer
497:          for (int i = MESSAGE_LENGTH
498:            data = (data << 1) | bitBu
499:          }
500:          // Assign the data to an Und
501:          UnderwaterMessage recvdMessa
502:          recvdMessage.data = data;
503:          Serial.print("Got message ra
504:          Serial.print(recvdMessage.ms
505:          Serial.print(", ID = ");
506:          Serial.print(recvdMessage.id
507:          Serial.print(" --- ");
508:          for (int i = UnderwaterMessa
509:            // Shift and mask to get e
510:            Serial.print((recvdMessage
511:          }
512:
513:          if (validUnderwaterMessage(r
514:            txrxAnimate(0); //ANIMATE
515:            lastLEDUpdateTime = millis
516:            // Play audio correspondin
517:            Serial.print(" --- USER: "
518:            Serial.print(user_ids_arra
519:
520:            playBoneconduct.play("user
521:            while (playBoneconduct.isP
522:            //UNCOMMENT TO USE PACKET
523:            // playBoneconduct.play(au
524:            // HARDCODED ID
525:            playBoneconduct.play(audio
526:            while (playBoneconduct.isP
527:            playBoneconduct.play("said
528:            while (playBoneconduct.isP
529:
530:            for (int c = 0; c < 6; c++
531:              if (recvdMessage.msg ==
532:                Serial.print(" --- M
533:                Serial.println(messa
534:                playBoneconduct.play
535:              }
536:            }
537:          }
538:
539:          transitionReceivingState(LIS
540:        }
541:      }
542:    }
543:
544:    // BUTTONS AND TRANSMITTING
545:    if (millis() - lastButtonCheckTime
546:      for (int b = 0; b < 6; b++) {
547:        // Serial.print(mcp.digitalRea
548:        if (mcp.digitalRead(b) == LOW)
549:          lastButtonCheckTime = millis
550:          Serial.print("Sending messag
551:          Serial.println(b);
552:          strip.clear();
553:
554:          txrxAnimate(1); //ANIMATE LE
555:          lastLEDUpdateTime = millis()
556:
557:          // BONE CONDUCTION CONFIRMAT
558:          playBoneconduct.play("you.wa
559:          while (playBoneconduct.isPla
560:          playBoneconduct.play("said.w
```

```
561:         while (playBoneconduct.isPla
562:         playBoneconduct.play(audio_m
563:
564:         transmitMessageAsync(UM_arra
565:
566:         // Serial.println("Queued me
567:         for (int i = UnderwaterMessa
568:             // Shift and mask to get
569:             Serial.print((UM_array[b
570:         }
571:         Serial.println(); // New lin
572:       }
573:     }
574:   }
575: }
576:
577: void txrxAnimate(bool dir) {
578:   int leftArr[9] = {1, 2, 3, 4, 5, 1
579:   int rightArr[9] = {1, 0, 11, 10, 9
580:
581:   strip.clear();
582:   strip.show();
583:
584:   if (dir) { // Forward direction
585:     for (int i = 0; i < 9; i++) {
586:       strip.setPixelColor(leftArr[i]
587:       strip.setPixelColor(rightArr[i
588:       strip.show();
589:       delay(75); // Adjust delay for
590:     }
591:   } else { // Reverse direction
592:     for (int i = 8; i >= 0; i--) {
593:       strip.setPixelColor(leftArr[i]
594:       strip.setPixelColor(rightArr[i
595:       strip.show();
596:       delay(75); // Adjust delay for
597:     }
598:   }
599: }
600:
601: float fAbs(float n) {
602:   if (n < 0) return -n;
603:   return n;
604: }
605:
606: void initializationPass(int check) {
607:     strip.fill(greenishwhite, 0, che
608:     strip.show();
609:     delay(100);
610:     strip.clear();
611: }
612:
613: void initializationError(int error)
614:     strip.fill(red, 0, error); // er
615:     strip.show();
616:     delay(2000);
617:     strip.clear();
618:
619:     while(true);
620: }
621:
622: void transitionOperatingMode(OPERATI
623:     if (newMode == RECEIVE) {
624:         digitalWrite(RELAY_PIN, LOW)
625:     } else if (newMode == TRANSMIT)
626:         digitalWrite(RELAY_PIN, HIGH
627:     }
628:     mode = newMode;
629: }
630:
631: void clearSampleBuffer() {
632:   // Clear sampling buffer
633:   for (int i=0; i<NUM_SAMPLES; i++)
634:     samplingBuffer[i] = -1; // Inval
635:   }
636:   //Reset sampling pointer
637:   samplingPointer = 0;
638: }
639:
640: void clearBitBuffer() {
641:   // Clear bit (received message) bu
642:   for (int i=0; i<MESSAGE_LENGTH; i+
643:     bitBuffer[i] = 0;
644:   }
645:   //Reset bit pointer
646:   bitPointer = 0;
647: }
648:
649: // Check whether sample buffer is va
650: bool isSampleBufferValid() {
651:   int validCount = 0;
652:   for (int i=0; i<NUM_SAMPLES; i++)
653:     if (samplingBuffer[i] != -1) {
654:       validCount++;
655:     }
656:   }
657:   return (validCount >= ((NUM_SAMPLE
658: }
659:
660: // Find the most freqently occuring
661: double sampleBufferMax() {
662:   uint16_t frequency_hist[1024] = {0
663:   uint16_t binNumber;
664:   // Populate the frequency histogra
665:   for (uint16_t i = 0; i < NUM_SAMPL
666:     binNumber = samplingBuffer[i];
667:     if (binNumber < 1024) { // Ensur
668:       frequency_hist[binNumber]++;
669:     }
670:   }
671:   int max_bin_count = 0;
672:   binNumber = 0;
673:   // Find the most frequent bin
674:   for (int16_t i = 0; i < 1024; i++)
675:     if (frequency_hist[i] > max_bin_
676:       max_bin_count = frequency_hist
677:       binNumber = i;
678:     }
679:   }
680:   // Return frequency in Hz based on
681:   return (double)binNumber * FFT_BIN
682: }
683:
684: // Function to transmit the Underwat
685: void transmitMessageAsync(Underwater
686:   transitionOperatingMode(TRANSMIT);
687:   addToneQueue(MESSAGE_START_FREQ, M
688:   // Iterate over each bit of the me
689:   for (int i = 0; i < MESSAGE_LENGTH
690:     // Extract the i-th bit from the
691:     if (message.data & (1 << i)) { /
692:       addToneQueue(MESSAGE_1_FREQ, M
693:     } else { // If the i-th bit is 0
694:       addToneQueue(MESSAGE_0_FREQ, M
695:     }
696:   }
697:   addToneQueue(MESSAGE_0_FREQ, (int)
698: }
699:
700: void addToneQueue(int freq, unsigned
```

```
  if (toneStackPos < toneBufferLengt
    toneFreqQueue[toneStackPos] = fr
    toneDelayQueue[toneStackPos] = d
    toneStackPos++; //always increas
    if (toneStackPos == 1) { //If it
      tone(HYDROPHONE_PIN, toneFreqQ
      lastToneStart = millis();
    }
  }
}

bool validAmplitude(double amp) {
  return (amp >= MIN_VALID_AMP && am
}

bool validUnderwaterMessage(Underwat
  uint8_t msg = message.msg;
  uint8_t id = message.id;

  return (msg >= 0 && msg < 6 && id
}

// Example usage: freqMatchesBounds(
// Example usage: freqMatchesBounds(
bool freqMatchesBounds(double freq,
  return abs(freq-(double)target) <=
}

// Transition state function
void transitionReceivingState(RECEIV
  // Serial.print("transitionReceivi
  // Serial.println(newState);
  if (newState == CHECK_START || new
    clearSampleBuffer();
    sampling = 1; // Begin sampling
    lastBitChange = millis(); // Sta
  } else { //Back to LISTENING
    clearSampleBuffer();
    clearBitBuffer();
    sampling = 0; // NOT sampling
  }
  curReceivingState = newState; // S
}

UnderwaterMessage createUnderwaterMe
    UnderwaterMessage message;
    message.msg = m & 0x7;   // Mask
    message.id = i & 0xF;    // Mask
    return message;
}

// Rainbow-enhanced theater marquee.
void theaterChaseRainbow(int wait) {
  int firstPixelHue = 0;     // Firs
  for(int a=0; a<30; a++) {  // Repe
    for(int b=0; b<3; b++) { //  'b'
      strip.clear();         //   Se
      // 'c' counts up from 'b' to e
      for(int c=b; c<strip.numPixels
        // hue of pixel 'c' is offse
        // revolution of the color w
        // of the strip (strip.numPi
        int      hue   = firstPixelH
        uint32_t color = strip.gamma
        strip.setPixelColor(c, color
      }
      strip.show();                /
      delay(wait);                 /
      firstPixelHue += 65536 / 90; /
    }
  }
}

// Rainbow cycle along whole strip.
void rainbow(int wait) {
  // Hue of first pixel runs 5 compl
  // Color wheel has a range of 6553
  // just count from 0 to 5*65536. A
  // means we'll make 5*65536/256 =
  for(long firstPixelHue = 0; firstP
    // strip.rainbow() can take a si
    // optionally a few extras: numb
    // saturation and value (brightn
    // ColorHSV() function, default
    // to apply gamma correction to
    strip.rainbow(firstPixelHue);
    // Above line is equivalent to:
    // strip.rainbow(firstPixelHue,
    strip.show(); // Update strip wi
    delay(wait);  // Pause for a mom
  }
}

/**************** Support SGTL sampl
#if defined(__IMXRT1062__)  // Teens

#include <utility/imxrt_hw.h>

// taken from: https://forum.pjrc.co
void setAudioSampleI2SFreq(int freq)
  // PLL between 27*24 = 648MHz und
  int n1 = 4; //SAI prescaler 4 => (
  int n2 = 1 + (24000000 * 27) / (fr
  double C = ((double)freq * 256 * n
  int c0 = C;
  int c2 = 10000;
  int c1 = C * c2 - (c0 * c2);
  set_audioClock(c0, c1, c2, true);
  CCM_CS1CDR = (CCM_CS1CDR & ~(CCM_C
          | CCM_CS1CDR_SAI1_CLK_PRED(n1
          | CCM_CS1CDR_SAI1_CLK_PODF(n2
//Serial.printf("setAudioSampleI2SFr
}

#elif defined(KINETISK)  // Teensy 3

// samplerate code by Frank Boesing
// https://forum.pjrc.com/threads/38
void setAudioSampleI2SFreq(int freq)
  typedef struct {
    uint8_t mult;
    uint16_t div;
  } __attribute__((__packed__)) tmcl
  const int numfreqs = 14;
  const int samplefreqs[numfreqs] =

// F_PLL = phase lock loop output fr

#if (F_PLL==16000000)
  const tmclk clkArr[numfreqs] = {{1
#elif (F_PLL==72000000)
  const tmclk clkArr[numfreqs] = {{3
#elif (F_PLL==96000000)
  const tmclk clkArr[numfreqs] = {{8
#elif (F_PLL==120000000)
  const tmclk clkArr[numfreqs] = {{3
#elif (F_PLL==144000000)
  const tmclk clkArr[numfreqs] = {{1
#elif (F_PLL==180000000)
  const tmclk clkArr[numfreqs] = {{4
```

```
841: #elif (F_PLL==192000000)
842:   const tmclk clkArr[numfreqs] = {{4
843: #elif (F_PLL==216000000)
844:   const tmclk clkArr[numfreqs] = {{3
845: #elif (F_PLL==240000000)
846:   const tmclk clkArr[numfreqs] = {{1
847: #endif
848:
849:   for (int f = 0; f < numfreqs; f++)
850:     if ( freq == samplefreqs[f] ) {
851:       while (I2S0_MCR & I2S_MCR_DUF)
852:       I2S0_MDR = I2S_MDR_FRACT((clkA
853:       return;
854:     }
855:   }
856: }
857:
858: #endif
```