```
1
   /*
     * FileName: main.c
 2
 3
     * Version: 1
 4
 5
     * Created: 11/11/2022 8:46:12 AM
 6
     * Author: Ethan Zeronik
 7
8
     * Operations: crane project source code
9
10
     * Hardware:
11
         Atmega2560
                             micro controller
         Port L
                             LED reposnse bar
12
13
        Port C.4
                             stepper motor limit switch
14
        Port C.0-3
                             stepper motor
        Port A.0-3
                             button inputs
15
16
         Port B.5-6
                             servo outputs
17
18
    /* NOTE: Includes */
19
   // standard include for the atmega program
20
21
   #include <avr/io.h>
   #include <avr/interrupt.h>
22
23
   #include <stdio.h>
24
25
26 // adds stepper motor module (1/6)
27
   #include "StepperMotor.h"
   // adds adc for input potentiometers (2/6)
28
29 #include "AnalogToDigital.h"
30 // adds pwm for servo motors (3/6)
31 #include "CraneServo.h"
32
   // adds UART and Bluetooth for communication (4/6)
33 #include "CraneCommunication.h"
   // adds EEPROM read/write for persistant data storage (5/6)
34
35 #include "CraneEeprom.h"
   // adds custom delay functions (6/6)
36
   #include "CraneDelay.h"
37
38
39
    /* NOTE: Types and Structs */
   // type to hold data for serial connections
40
41
   typedef struct connectionBuffer_t
42
        // the buffer to store the data in
43
44
                buffer[128];
        // the current index of the last set byte in the buffer
45
46
        uint8 t index;
        // whether or not the buffer is done being written to
47
48
        uint8 t readFlag;
    } connectionBuffer t;
49
   // type to hold the position of the servos and motor
50
   typedef struct cranePosition t
51
52
   {
        // motor position in steps
53
54
        int16_t motorTicks;
        // arm position in 0-255 ticks
55
        uint8_t armTicks;
```

```
57
        // plunger position in 0-255 ticks
58
        uint8_t plungerTicks;
59
    } cranePosition t;
60
    /* NOTE: Custom Macros */
61
     // the three main states of the application
62
    #define calibrateState 0
63
    #define actionState
64
    #define recordState
65
66
    // command definition
67
    #define recordModeCommand
                                  "Calibrate"
68
69
     #define calibrateModeCommand "Reset"
70
    #define runCommand
                                  "Run"
71
     #define recordCommand
                                  "Record"
72
    #define getCommand
                                  "Get"
73
    // custom buttons & switches
74
     #define limitSwitch
75
                             (PINC & 0x10)
76 #define leftButton
                             (PINA & 0x01)
    #define rightButton
77
                             (PINA & 0x02)
78
    #define recordButton
                             (PINA & 0x04)
79
    #define calibrateButton (PINA & 0x08)
80
    // custom led bar
81
82
    #define stateLed PORTL
83
84
    // custom servos ids
    #define armServo
85
86
    #define plungerServo 1
87
     // defined values for where default spots of the servo are
88
     #define armStartPosition
                                  150
89
    #define plungerStartPosition 230
90
91
    // record information
92
93
    #define recordLength 6
    #define eepromAddress 0x0555
94
95
    /* NOTE: Global Variables */
96
97
    // the state of the application
                    applicationState = calibrateState;
98
99
     // the state of all the moving parts of the crane
     cranePosition t craneState
100
101
               .motorTicks
                             = 0,
102
               .armTicks
                             = 0,
               .plungerTicks = ∅,
103
104
    };
    // the EEPROM recorded moves to make
105
    cranePosition_t
                                 recordedMoves[recordLength];
106
107
    // the moves to save to eemprom
108
    cranePosition t
                                 currentMoves[recordLength];
    // current spot in the instance moves
109
110
    uint8 t
                                 currentMoveIndex = 0;
    // the input buffer for serial
111
    volatile connectionBuffer t serialInputData = {
```

```
.buffer = \{0\},
113
114
          .index
                    = 0,
115
          .readFlag = 0,
116
    };
117
    // the input buffer for bluetooth
    volatile connectionBuffer t bluetoothInputData = {
118
119
         .buffer
                   = \{0\},
120
         .index
                   = 0,
121
         .readFlag = 0,
122
    };
123
    /* NOTE: Function prototypes */
124
125
    // inits IO ports
126 void
             IO_init(void);
127
    // gets the saved values from the eeprom
128 void
            CRANE getMovesFromEeprom(uint16 t addr);
129
    // saves the current recorded moves to eeprom
130 void
             CRANE saveMovesToEeprom(uint16 t addr);
    // takes a string and checks if the buffer matches the value exactly
131
132 // strictmode resets readflag
    // returns 1 if true, else 0
133
134
    uint8_t doesBufferMatch(volatile connectionBuffer_t buf, uint8_t strictMode, char const *
     const pStr);
135
    // compares to strings
    // if the match it returns 1, else 0
136
137
    uint8 t stringCompare(char const * const pStrOne, char const * const pStrTwo);
138
139
    /* NOTE: Application implementation */
    // the main loop of the function, provided to us
140
    int main(void)
141
142
     {
143
         // start the gpio
        IO init();
144
145
146
        // init the ADC and the servo control
147
        ADC init();
148
        CRANE_initServos();
        CRANE startServos();
149
150
151
        // in this case, 1 is CCW, and 0 is CW
152
        // pass in the port and register we want to use for the motor
        SM init(&DDRC, &PORTC);
153
154
155
        // get the stored recorded data from the EEPROM
156
        CRANE getMovesFromEeprom(eepromAddress);
157
158
        // turn on the serial on port 0 at 9600 baud
159
        CRANE_initSerial(9600);
160
        CRANE sendSerial("Crane online\r\n");
161
        // turn on the blutooth connection
162
        CRANE initBluetooth(9600);
163
164
        CRANE_sendBluetooth("Crane online\r\n");
165
166
         // start the delay timer
167
        CRANE initTimer();
```

```
168
169
         // start global interrupts
170
         sei();
171
172
         while(1)
173
              if(bluetoothInputData.readFlag)
174
175
                  CRANE sendSerial(bluetoothInputData.buffer);
176
177
              }
178
             // DEBUG: get the state
179
180
             // if we get the command for record mode and we are not already in record mode
             if((doesBufferMatch(serialInputData, 0, recordModeCommand) && (applicationState !=
181
     recordState)) || (doesBufferMatch(bluetoothInputData, 0, recordModeCommand) &&
     (applicationState != recordState)) || (recordButton && (applicationState != recordState)))
182
183
                  CRANE sendSerial("Entering record mode...\r\n");
                  CRANE sendBluetooth("Entering record mode...\r\n");
184
185
                  applicationState
                                                = recordState;
186
187
                  serialInputData.readFlag
                                                = 0;
188
                  bluetoothInputData.readFlag = 0;
189
190
                  while(recordButton)
191
192
                      // do nothing until we let go
193
194
195
              // check for reset command
     else if((doesBufferMatch(serialInputData, 0, calibrateModeCommand) &&
(applicationState != calibrateState)) || (doesBufferMatch(bluetoothInputData, 1,
196
     calibrateModeCommand) && (applicationState != calibrateState)) || (calibrateButton &&
     (applicationState != calibrateState)))
197
                  CRANE sendSerial("Resetting...\r\n");
198
                  CRANE sendBluetooth("Resetting...\r\n");
199
200
201
                  applicationState
                                                = calibrateState;
                  serialInputData.readFlag
202
203
                  bluetoothInputData.readFlag = 0;
204
205
                  while(calibrateButton)
206
207
                      // do nothing until we let go
208
209
             }
210
211
             // display state on leds
              stateLed = (stateLed & 0xfc) | applicationState;
212
213
214
             // DEBUG: check for the get command
             if(doesBufferMatch(serialInputData, 0, getCommand) ||
215
     doesBufferMatch(bluetoothInputData, 0, getCommand))
216
              {
217
                  char response[96];
                  sprintf(response, "Motor is at %i, Plunger is at %u, Arm is at %u\r\n",
218
     craneState.motorTicks, craneState.plungerTicks, craneState.armTicks);
```

```
219
220
                 serialInputData.readFlag
221
                 bluetoothInputData.readFlag = 0;
222
223
                 CRANE sendSerial(response);
224
                 CRANE sendBluetooth(response);
225
             }
226
             // main application switch case
227
             switch(applicationState)
228
229
                 // DEBUG: the action case
230
231
                 case actionState:
232
                      if(leftButton || doesBufferMatch(serialInputData, ∅, runCommand) ||
233
     doesBufferMatch(bluetoothInputData, 0, runCommand))
234
235
                          // HACK: potential multiple select
                          serialInputData.readFlag
236
237
                          bluetoothInputData.readFlag = 0;
238
                          for(uint8_t i = 0; i < recordLength; i++)</pre>
239
240
241
                              char response[32];
242
                              sprintf(response, "Running recorded step %u...\r\n", i + 1);
243
244
                              CRANE sendSerial(response);
245
                              CRANE sendBluetooth(response);
                              stateLed = (stateLed & 0x03) | 1 << (i + 2);
246
247
248
                              // calculate the relative movement of the arm
249
                              int16_t moveSteps = recordedMoves[i].motorTicks -
     craneState.motorTicks;
250
251
                              // move motor
252
                              if(moveSteps > 0)
253
254
                                  SM moveStepsSigned(stepperModeHalf, 0, moveSteps);
255
256
                              else if(moveSteps < 0)</pre>
257
                                  SM_moveStepsSigned(stepperModeHalf, 1, -1 * moveSteps);
258
259
260
                              // lerp!
261
262
                              for(uint8_t j = 1; j < 101; j++)
263
                                  CRANE setServoPosition(armServo, craneState.armTicks +
264
     ((recordedMoves[i].armTicks - craneState.armTicks) * ((float)j / 100)));
265
                                  CRANE delayMs(5);
                              }
266
267
268
                              for(uint8_t j = 1; j < 101; j++)
269
270
                                  CRANE setServoPosition(plungerServo, craneState.plungerTicks +
     ((recordedMoves[i].plungerTicks - craneState.plungerTicks) * ((float)j / 100)));
271
                                  CRANE delayMs(5);
```

```
272
                              }
273
274
                              // set our state
275
                              craneState.motorTicks = recordedMoves[i].motorTicks;
276
                              craneState.armTicks
                                                      = recordedMoves[i].armTicks;
                              craneState.plungerTicks = recordedMoves[i].plungerTicks;
277
278
279
                              CRANE_delayMs(100);
                          }
280
281
282
                         // reset state led
                          stateLed = (stateLed & 0x03);
283
284
                         while(leftButton)
285
286
                          {
287
                              // do nothing until we let go
288
289
                     }
290
291
                 break;
292
293
                 // DEBUG: the record case
                 case recordState:
294
295
296
                     uint8_t armPosition = 255 * ADC_getTenBitValue(0);
297
                     uint8_t plungerPosition = 255 * ADC_getTenBitValue(1);
298
                     int16_t moveSteps
                                              = 8;
299
300
                     // display the step we are recording
301
                     stateLed = (stateLed & 0x03) | 1 << (currentMoveIndex + 2);</pre>
302
303
                     // manually move the motor
                     if(rightButton)
304
305
                          SM_moveStepsSigned(stepperModeHalf, 0, moveSteps);
306
                          CRANE_delayMs(10);
307
308
309
                     else if(leftButton)
310
                          SM_moveStepsSigned(stepperModeHalf, 1, moveSteps);
311
312
                         CRANE_delayMs(10);
313
314
                         // negate for current position
                         moveSteps *= -1;
315
316
317
                     else
318
319
                         moveSteps = 0;
320
                     }
321
                     // manually move the servos
322
323
                     CRANE_setServoPosition(armServo, armPosition);
                     CRANE setServoPosition(plungerServo, plungerPosition);
324
325
326
                     // update the current positions
327
                     craneState.motorTicks += moveSteps;
```

```
328
                      craneState.armTicks
                                              = armPosition;
329
                      craneState.plungerTicks = plungerPosition;
330
331
                      // if we press the record button save the position
332
                     if(recordButton | doesBufferMatch(serialInputData, 0, recordCommand) |
     doesBufferMatch(bluetoothInputData, 0, recordCommand))
333
334
                          serialInputData.readFlag
335
                          bluetoothInputData.readFlag = 0;
336
337
                          char response[64];
                          sprintf(response, "Recording step %u out of 6...\r\n", currentMoveIndex +
338
     1);
339
340
                          CRANE sendSerial(response);
341
                          CRANE sendBluetooth(response);
342
343
                          sprintf(response, "Recorded {%i,%u,%u}...\r\n", craneState.motorTicks,
     craneState.plungerTicks, craneState.armTicks);
344
345
                          CRANE_sendSerial(response);
                          CRANE sendBluetooth(response);
346
347
                          // reset state led
348
349
                          stateLed = (stateLed & 0x03);
350
351
                          if(currentMoveIndex < recordLength)</pre>
352
353
                              currentMoves[currentMoveIndex++] = craneState;
354
                          }
355
                          if(currentMoveIndex >= recordLength)
356
357
                          {
358
                              CRANE saveMovesToEeprom(eepromAddress);
                              currentMoveIndex = 0;
359
360
                              // done recording, back to action state after zeroing
361
362
                              // home ---> action(play)
                              applicationState = calibrateState;
363
364
                          }
365
                         while(recordButton)
366
367
                              // do nothing until we let go
368
369
370
                     }
371
372
                 break;
373
374
                 // DEBUG: the default case will be the home case
                 case calibrateState:
375
376
                 default:
377
378
                     // 0 is up 255 is down
379
                     CRANE setServoPosition(armServo, armStartPosition);
380
                     // 255 is close 0 is open
381
                     CRANE_setServoPosition(plungerServo, plungerStartPosition);
```

```
382
383
                      // move CW for one second to ensure the limit switch is not set
                     SM moveTime(stepperModeHalf, 0, 1000, 3);
384
385
386
                     // while not hitting the switch
387
                     while(!limitSwitch)
388
                      {
                          // then move CCW a bit at the time until we hit the limit switch
389
390
                          SM moveStepsSigned(stepperModeHalf, 1, 24);
                      }
391
392
393
                     // then move 30 degrees back to center the arm
394
                     SM_movePosition(stepperModeHalf, 35);
395
396
                     // set the current position
                     craneState.motorTicks
397
398
                      craneState.armTicks
                                              = armStartPosition;
399
                     craneState.plungerTicks = plungerStartPosition;
400
                     // then set it to action state
401
402
                     applicationState = actionState;
403
                 break;
404
405
             }
         }
406
407
408
409
     // interrupt handling for the serial connection
    ISR(serialInterrupt)
410
411
         if(serialData != '\r' && serialData != '\n' && serialData != '\0' &&
412
     (serialInputData.index < 127))</pre>
413
         {
414
             // add to array
             serialInputData.buffer[serialInputData.index]
                                                                = serialData;
415
             serialInputData.buffer[serialInputData.index + 1] = '\0';
416
417
418
             serialInputData.index++;
419
         }
         else
420
421
         {
422
             // set update flag
423
             serialInputData.readFlag = 1;
             // reset message index
424
425
             serialInputData.index
                                       = 0;
426
427
428
429
     // interrupt handling for the bluetooth connection
430
     ISR(bluetoothInterrupt)
431
     {
         if(bluetoothData != '\r' && bluetoothData != '\n' && bluetoothData != '\0' &&
432
     (bluetoothInputData.index < 127))</pre>
433
         {
434
             // add to array
             bluetoothInputData.buffer[bluetoothInputData.index]
                                                                    = bluetoothData;
435
             bluetoothInputData.buffer[bluetoothInputData.index + 1] = '\0';
436
```

```
437
438
             bluetoothInputData.index++;
439
         }
440
         else
441
         {
442
             // set update flag
443
             bluetoothInputData.readFlag = 1;
             // reset message index
444
             bluetoothInputData.index
445
                                           = 0;
         }
446
447
448
449
     /* NOTE: Function implementations */
450
     void IO_init(void)
451
     {
452
         // port c.4 is the limit switch
453
         DDRC = 0x00;
454
         PORTC = 0x10;
455
         // port a.0-1 are for left and right
456
457
         DDRA = 0x00;
         PORTA = 0xff;
458
459
460
         DDRL = 0xff;
         PORTL = 0x00;
461
462
     }
463
464
     void CRANE getMovesFromEeprom(uint16 t addr)
465
     {
466
         uint16_t address = addr;
467
468
         for(uint8_t i = 0; i < recordLength; i++)</pre>
469
             uint16 t motorTicks = 0;
470
471
472
             // read the motor position
             motorTicks = CRANE eepromReadChar(address++) << 8;</pre>
473
474
             motorTicks += CRANE_eepromReadChar(address++);
475
             recordedMoves[i].motorTicks = motorTicks;
476
477
478
             // read the arm position
479
             recordedMoves[i].armTicks = CRANE eepromReadChar(address++);
480
481
             // read the plunger position
482
             recordedMoves[i].plungerTicks = CRANE eepromReadChar(address++);
         }
483
484
     }
485
     void CRANE_saveMovesToEeprom(uint16_t addr)
486
487
488
         uint16_t address = addr;
489
490
         for(uint8_t i = 0; i < recordLength; i++)</pre>
491
492
             // cheat by directly moving into our recorded array
```

12/1/22, 9:15 AM

return 1;

// made it out of the loop

537538539

540

541 }

```
1 /*
     * FileName: StepperMotor.h
 2
 3
     * Version: 1
 4
 5
     * Created: 9/14/2022 2:00 PM
     * Author: Ethan Zeronik
 6
 7
 8
     * Operations: header for the stepper motor submobule
 9
10
    #ifndef StepperMotor h INCLUDED
11
12
    #define StepperMotor h INCLUDED
13
14
   #if defined(__cplusplus)
    extern "C" {
15
16
   #endif
17
18
    #pragma message("WARNING: this module uses the bottom nibble of the provided port")
19
   #include <stdbool.h>
20
   #include <stdint.h>
21
22
   /* NOTE: Custom Types */
23
   // typing for the stepper motor enum
    typedef enum StepperMotorRunMode t
25
26
   {
27
        // wave step mode
28
        stepperModeWave = 0,
29
        // full step mode
30
        stepperModeFull = 1,
31
        // half step mode
32
        stepperModeHalf = 2,
33
   } StepperMotorRunMode t;
34
   /* NOTE: Function prototypes */
35
   // inits IO for the stepper motor
36
   // takes a pointer to the port to use, assumes botom nibble
37
   void SM_init(uint8_t volatile * const pRegister, uint8_t volatile * const pPort);
38
39
   // moves the motor in the given mode to the given distance
   // distance is in units of rotation
40
   void SM_move(StepperMotorRunMode_t mode, double distance);
41
42
   // moves the motor in the given mode to the given position
   // distance is in units of degrees
43
   void SM movePosition(StepperMotorRunMode t mode, uint16 t distance);
   // moves the motor in the given mode and the given direction for the given time
45
   // 1 is CW and 0 is CCW
   // both times are in ms
47
   void SM moveTime(StepperMotorRunMode t mode, bool direction, double time, double stepTime);
48
   // moves the motor in the given mode and the given direction for the given distance
49
   // distance is in steps
50
   // 1 is CW and 0 is CCW
51
   void SM_moveStepsSigned(StepperMotorRunMode_t mode, bool direction, uint16_t distance);
52
53
54
   #if defined(__cplusplus)
   } /* extern "C" */
55
56 #endif
```

12/1/22, 9:20 AM StepperMotor.h

57 | 58 | **#endif** // StepperMotor\_h\_INCLUDED

```
1
 2
     * FileName: StepperMotor.c
 3
     * Version: 1
 4
 5
     * Created: 9/14/2022 2:00 PM
 6
     * Author: Ethan Zeronik
 7
8
     * Operations: run the stepper motor in one of three modes
9
10
    /* NOTE: Includes */
11
    #include "StepperMotor.h"
12
13
14
   #if !defined(F_CPU)
        #define F CPU 16000000UL
15
16
   #endif
   // allows for variable delay
17
18
   #define __DELAY_BACKWARD_COMPATIBLE__
19
   #include <util/delay.h>
20
21
   /* NOTE: Local declarations */
22
   // local struct for function return
23
24
   typedef struct StepperMotorModeData t
25
26
        // size of the array
27
        uint8 t
                               arraySize;
28
        // pointer to the array
29
        uint8_t const * const pArray;
        // number of steps to take for desired rotation
30
        uint32_t
31
                               steps;
32
   } StepperMotorModeData_t;
33
   // returns the amount of steps needed for the given mode
   // rotation is in radians (I think)
34
   StepperMotorModeData t getModeAndSteps(StepperMotorRunMode t mode, double rotation);
35
36
   /* NOTE: Global Variables */
37
38
   // implementation of the wave step map
    static uint8 t sWaveStepMap[4] = {
39
40
        0x01,
41
        0x02,
42
        0x04,
43
        0x08,
44
   };
   // implementation of the full step map
45
46
    static uint8 t sFullStepMap[4] = {
47
        0x03,
48
        0x06,
        0x0c,
49
50
        0x09,
51
52
   // implementation of the wave step map
   static uint8 t sHalfStepMap[8] = {
53
54
        0x09,
55
        0x01,
56
        0x03,
```

```
57
         0x02,
 58
         0x06,
 59
         0x04,
 60
         0x0c,
 61
         0x08,
 62
     };
     // instance pointer to the motor port
 63
     static uint8_t * sMotorPort;
 64
 65
     /* NOTE: Function implementations */
 66
 67
     void SM_init(uint8_t volatile * const pRegister, uint8_t volatile * const pPort)
 68
 69
         // configure port register
         *pRegister |= 0x0f;
 70
 71
 72
         // turn on pullup resisitors on the bottom nibble
         *pPort = (*pPort & 0xf0) | 0x00;
 73
 74
 75
         // save the port pointer to the static var
         sMotorPort = (uint8_t *)pPort;
 76
 77
     }
 78
 79
     void SM move(StepperMotorRunMode t mode, double distance)
 80
         StepperMotorModeData_t data = getModeAndSteps(mode, distance);
 81
 82
 83
         for(uint32_t i = 0, j = 0; i < data.steps; i++)</pre>
 84
         {
 85
             *sMotorPort = (*sMotorPort & 0xf0) | data.pArray[j++];
 86
 87
             if(j >= data.arraySize)
 88
 89
                 j = 0;
             }
 90
 91
 92
             _delay_ms(3);
 93
         }
 94
 95
         *sMotorPort = *sMotorPort & 0xf0;
 96
 97
    void SM movePosition(StepperMotorRunMode t mode, uint16 t distance)
 98
 99
         SM move(mode, ((double)distance / 360));
100
101
102
     void SM_moveTime(StepperMotorRunMode_t mode, bool direction, double time, double stepTime)
103
104
         StepperMotorModeData t data = getModeAndSteps(mode, 0);
105
106
         for(uint32 t i = 0, j = (direction ? data.arraySize : 0); i < (time / stepTime); i++)</pre>
107
108
         {
             *sMotorPort = (*sMotorPort & 0xf0) | data.pArray[(direction ? j-- : j++)];
109
110
             if(j >= data.arraySize || j <= 0)</pre>
111
112
```

j = (direction ? data.arraySize : 0);

```
12/1/22, 9:20 AM
 113
 114
 115
 116
 117
 118
 119
       }
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
```

```
_delay_ms(stepTime);
         }
         *sMotorPort = *sMotorPort & 0xf0;
    void SM moveStepsSigned(StepperMotorRunMode t mode, bool direction, uint16 t distance)
         StepperMotorModeData t data = getModeAndSteps(mode, 0);
         for(uint32_t i = 0, j = (direction ? data.arraySize : 0); i < distance; i++)</pre>
         {
             *sMotorPort = (*sMotorPort & 0xf0) | data.pArray[(direction ? j-- : j++)];
             if(j >= data.arraySize || j <= 0)</pre>
                 j = (direction ? data.arraySize : 0);
             }
             _delay_ms(3);
         }
137
138
         *sMotorPort = *sMotorPort & 0xf0;
139
140
141
     /* NOTE: Local function implementations */
142
    StepperMotorModeData_t getModeAndSteps(StepperMotorRunMode_t mode, double rotation)
143
    {
144
         uint8_t * pArray;
145
         uint8 t
                 size = 0;
         uint32_t steps = 0;
146
147
148
         switch(mode)
149
         {
150
             case stepperModeWave:
151
152
                 pArray = sWaveStepMap;
153
                       = sizeof(sWaveStepMap) / sizeof(sWaveStepMap[0]);
154
                 steps = (rotation * 2048);
155
             }
156
             break;
157
             case stepperModeFull:
158
159
                 pArray = sFullStepMap;
                 size = sizeof(sFullStepMap) / sizeof(sFullStepMap[0]);
160
                 steps = (rotation * 2048);
161
162
163
             break;
             case stepperModeHalf:
164
165
166
                 pArray = sHalfStepMap;
                 size = sizeof(sHalfStepMap) / sizeof(sHalfStepMap[0]);
167
168
                 steps = (rotation * 4096);
```

```
169
              }
170
              break;
171
              default:
                  break;
172
173
         };
174
         return (StepperMotorModeData_t){
175
              .pArray = pArray,
.steps = steps,
176
177
178
              .arraySize = size,
179
         };
180 }
```

```
1 /*
 2
     * FileName: AnalogToDigital.h
     * Version: 1
 3
 4
 5
     * Created: 10/19/2022 12:47 AM
 6
     * Author: Ethan Zeronik
 7
 8
     * Operations: header for the adc submodule
 9
     */
10
    #ifndef AnalogToDigital_h_INCLUDED
11
    #define AnalogToDigital h INCLUDED
12
13
   #if defined(__cplusplus)
14
    extern "C" {
15
16
   #endif
17
18
   #include <stdint.h>
19
   /* NOTE: Custom Types */
20
   // typing for the handler function
21
    typedef void (*AnalogAsyncGetHandler_t)(uint16_t);
22
23
24
   /* NOTE: Function prototypes */
25
   // init registers for adc
26
   void
            ADC init(void);
27
   // init adc for interrupt mode
            ADC initInterrupt(void);
28
   void
29
   // returns the value of the given channel
   double ADC_getTenBitValue(uint16_t channel);
30
31
   // gets the 10 bit value on the channel
   uint16_t ADC_getTenBitValueInterrupt(uint16_t channel);
   // set the interrupt handler for the 10 bit async mode
33
   void
             ADC setInterruptHandler(AnalogAsyncGetHandler t cb);
34
35
36
   #if defined(__cplusplus)
   } /* extern "C" */
37
38
   #endif
39
40 #endif // AnalogToDigital_h_INCLUDED
```

```
1
   /*
     * FileName: AnalogToDigital.c
 2
 3
     * Version: 1
 4
 5
     * Created: 10/19/2022 12:47 AM
 6
     * Author: Ethan Zeronik
 7
 8
     * Operations: basic adc implementation
 9
     */
10
    /* NOTE: Includes */
11
    #include "AnalogToDigital.h"
12
13
   #include <avr/io.h>
14
    #include <avr/interrupt.h>
15
16
17
    /* NOTE: Global Variables */
18
   // value from the interruput
19
    static uint16 t
                                    readInterrupt = 0;
   // callback for the interrupt
    static AnalogAsyncGetHandler t interruptCallback;
21
22
23
    /* NOTE: Local function implementations */
    void ADC init(void)
24
25
        // ten bit one way mode
26
27
        ADCSRA = (1 << ADEN) | (1 << ADPS1) | (1 << ADPS0);
28
        // 5v reference
29
30
        ADMUX = (1 << REFS0);
31
32
        ADCSRB = 0x00;
33
    }
34
35
    void ADC initInterrupt(void)
36
37
        ADC_init();
38
        ADCSRA |= (1 << ADIE);
39
    }
40
41
42
    double ADC getTenBitValue(uint16 t channel)
43
    {
44
        uint16_t result = 0;
45
46
        // select the channel
        ADMUX = (ADMUX & 0xe0) | channel;
47
        ADCSRB = (ADCSRB & 0xf7) | (channel >> 2);
48
49
        // start conversion
50
        ADCSRA = (1 << ADSC);
51
52
53
        // wait for conversion
        while((ADCSRA & (1 << ADSC)) == 1)</pre>
54
55
            // do nothing
```

```
12/1/22, 9:19 AM
 57
 58
 59
         // save result
          result = ADCL;
 60
          result = result | (ADCH << 8);
 61
 62
          return result / 1024.0;
 63
 64
     }
 65
     uint16_t ADC_getTenBitValueInterrupt(uint16_t channel)
 66
 67
         // select the channel
 68
 69
         ADMUX = (ADMUX \& 0xe0) \mid channel;
         ADCSRB = (ADCSRB & 0xf7) | (channel >> 2);
 70
 71
 72
         // start conversion
         ADCSRA = (1 << ADSC);
 73
 74
 75
          return readInterrupt;
 76
     }
 77
 78
     void ADC_setInterruptHandler(AnalogAsyncGetHandler_t cb)
 79
          interruptCallback = cb;
 80
 81
     }
 82
 83
     /* NOTE: Local function implementations */
 84
     ISR(ADC_vect)
 85
          readInterrupt = ADCL;
 86
 87
          readInterrupt = readInterrupt | (ADCH << 8);</pre>
 88
          interruptCallback(readInterrupt);
 89
 90 }
```

```
1 /*
2
     * CraneCommunication.h
3
4
     * Created: 11/2/2022 8:32:52 AM
5
     * Author: xiang82, Alex Weyer, Yu-Hung (Thomas) Wang
     */
6
7
8
   #ifndef CraneCommunication H INCLUDED
9
   #define CraneCommunication H INCLUDED
10
   #if defined(__cplusplus)
11
12
   extern "C" {
13
   #endif
14
15
   #include <avr/io.h>
16
   #include <avr/interrupt.h>
17
18
   // defines in case of no callback
   #define serialInterrupt USART0 RX vect
19
   #define bluetoothInterrupt USART1 RX vect
21
   // defines for use in interrupt
22
23
   #define serialData UDR0
   #define bluetoothData UDR1
24
25
   // sets up usart0 for serial communication
26
27
   void CRANE initSerial(uint16 t baudRate);
   // sends the given string to the main serialport
28
29
   void CRANE_sendSerial(char const * const pData);
   // sets up usart1 for serial communication
30
31
   void CRANE initBluetooth(uint16 t baudRate);
   // sends the given string to the bluetooth serialport
   void CRANE sendBluetooth(char const * const pData);
33
34
   #if defined( cplusplus)
35
   } /* extern "C" */
36
   #endif
37
38
   #endif /* CraneCommunication H INCLUDED */
```

```
1
 2
     * CraneCommunication.c
 3
4
     * Created: 2022/11/5 10:03:43 am
 5
     * Author: xiang82, Alex Weyer, Yu-Hung (Thomas) Wang
 6
 7
8
    /* NOTE: Includes */
9
    #include "CraneCommunication.h"
10
   #if !defined(F_CPU)
11
        #define F CPU 16000000UL
12
13
   #endif
14
   /* NOTE: Local declarations */
15
16
   // Xiangs's serial send function
   void UART_out(uint8_t ch);
17
18 // Alex and Thomas's serial send function
19
   // transmit single byte of data
   void BLUETOOTH out(uint8 t ch);
20
21
   /* NOTE: Global function implementations */
22
   void CRANE initSerial(uint16 t baudRate)
23
24
   {
25
        // ubrr load
        uint16_t myubr;
26
27
28
        // set up the ucsr0a and ucsr0b and ucsr0c
29
        UCSR0A = 0x00;
30
        UCSROB = (1 << RXCIEO) | (1 << RXENO) | (1 << TXENO);
31
        UCSR0C = (1 << UCSZ00) | (1 << UCSZ01);
32
33
        myubr = (F_CPU / (16UL * (uint16_t)baudRate)) - 1;
34
        // load ubrr low
        UBRR0L = myubr;
35
36
37
        UBRR0H = 0x00;
38
39
   void CRANE sendSerial(char const * const pData)
40
41
42
        char const * pWorker = (char const *)pData;
43
44
        // while we are not at the end of the string
        while(*pWorker != '\0')
45
46
            // wait for uart tx to be ready then send out uart
47
            UART out(*pWorker);
48
49
50
            pWorker++;
        }
51
52
53
   void CRANE_initBluetooth(uint16_t baudRate)
54
55
56
        uint16_t mybur;
```

```
57
         UCSR1A = 0;
 58
 59
         // enable receive interrupt
         // enable transmits
 60
 61
         // enable receive
         // 2 stop bits
 62
         UCSR1B = (1 << RXCIE1) | (1 << TXEN1) | (1 << RXEN1);
 63
 64
 65
         UCSR1C = (1 << UCSZ11) | (1 << UCSZ10);
 66
 67
         // set up baud rate
         mybur = (F_CPU) / (16UL * (uint16_t)baudRate) - 1;
 68
 69
 70
         UBRR1L = mybur;
 71
         UBRR1H = 0x00;
 72
     }
 73
    void CRANE sendBluetooth(char const * const pData)
 74
 75
         char const * pWorker = (char const *)pData;
 76
 77
 78
         // while we are not at the end of the string
         while(*pWorker != '\0')
 79
 80
             // wait for uart tx to be ready then send out uart
 81
 82
             BLUETOOTH out(*pWorker);
 83
 84
             pWorker++;
 85
         }
 86
     }
 87
     /* NOTE: Local function implementations */
 88
     void UART_out(uint8_t ch)
 89
 90
 91
         // wait to complete transmission and empty udr0
         while((UCSR0A & (1 << UDRE0)) == 0)</pre>
 92
 93
         {
 94
         }
 95
         // load next byte to be transmitted
 96
         UDR0 = ch;
 97
 98
     }
 99
    void BLUETOOTH out(uint8 t ch) // transmit single byte of data
100
101
         while((UCSR1A & (1 << UDRE1)) == 0)</pre>
102
103
             // wait for completing transmission and empty UDR0
104
105
106
         UDR1 = ch; // load next byte to be transmitted
107
108 }
```

```
1 /*
 2
    * CraneDelay.h
 3
 4
     * Created: 10/26/2022 8:47:13 AM
 5
     * Author: weyer4
 6
 7
 8
   #ifndef CraneDelay_H_INCLUDED
 9
    #define CraneDelay_H_INCLUDED
10
   #if defined(__cplusplus)
11
12
    extern "C" {
   #endif
13
14
   #include <avr/interrupt.h>
15
   #include <avr/io.h>
16
17
18
   // starts timer 0 in async mode
   void CRANE_initTimer(void);
19
   // returns the current delay in ms
   uint16_t CRANE_tick(void);
21
   // delays for the desired amount of ms
22
23
   void CRANE_delayMs(uint16_t ms);
24
   #if defined(__cplusplus)
25
   } /* extern "C" */
26
27
   #endif
28
29 #endif /* CraneDelay_H_INCLUDED */
```

```
1
   /*
 2
     * CraneDelay.c
 3
 4
     * Created: 10/5/2022 8:47:57 AM
 5
     * Author : Alex Weyer
 6
 7
 8
    /* NOTE: Includes */
    #include "CraneDelay.h"
 9
10
   #if !defined(F_CPU)
11
12
        #define F CPU 16000000UL
13
   #endif
14
   /* NOTE: Local declarations */
15
   // variable to increment and use for delay
16
17
    volatile uint16_t tick;
18
    /* NOTE: Global function implementations */
19
   void CRANE initTimer(void)
20
21
    {
22
        TCNT0 = 6;
23
        TCCR0A = 0;
24
        TCCR0B = 0x03;
25
        TIMSK0 = (1 << TOIE0);
26
    }
27
   uint16_t CRANE_tick(void)
28
29
    {
        return tick;
30
31
    }
32
33
    void CRANE delayMs(uint16 t ms){
        uint16 t desiredTick = tick + ms;
34
35
36
        while (desiredTick != tick)
37
38
            // do nothing
39
    }
40
41
   /* NOTE: Local function implementations */
42
   ISR(TIMER0_OVF_vect)
43
44
        TCNT0 = 6;
45
46
        tick++;
47
    }
```

```
1 /*
 2
     * CraneEeprom.h
 3
 4
     * Created: 11/2/2022 8:32:52 AM
 5
     * Author: xiang82
     */
 6
 7
 8
   #ifndef CraneEeprom_H_INCLUDED
 9
    #define CraneEeprom H INCLUDED
10
   #if defined(__cplusplus)
11
12
    extern "C" {
   #endif
13
14
15
   #include <avr/io.h>
16
   // write a charater to the given address
17
   void CRANE_eepromWriteChar(char in, uint16_t addr);
18
   // reads a character
19
   char CRANE eepromReadChar(uint16 t addr);
   // write a string to the given address
21
   void CRANE_eepromWriteString(char const * const in, uint16_t addr);
22
23
   // reads a string from the address into the buffer
   void CRANE_eepromReadString(uint16_t addr, char * const buf);
24
25
   #if defined(__cplusplus)
26
   } /* extern "C" */
27
   #endif
28
29
30 #endif /* CraneEeprom_H_INCLUDED */
```

```
1
   /*
 2
     * CraneEeprom.c
 3
 4
     * Created: 2022/11/5 10:03:43 am
 5
     * Author: xiang82
 6
 7
 8
    /* NOTE: Includes */
 9
    #include "CraneEeprom.h"
10
    /* NOTE: Global function implementations */
11
    void CRANE eepromWriteChar(char ucData, uint16 t uiAddress)
12
13
        while(EECR & (1 << EEPE))</pre>
14
15
16
            /* Wait for completion of previous write */
17
18
19
        /* Set up address and Data Registers */
        EEAR = uiAddress;
20
21
        EEDR = ucData;
22
23
        /* Write logical one to EEMPE */
        // step 5. write 1 to EEMPE and 0 to EEPE
24
25
        EECR = (1 << EEMPE);
26
27
        /* Start EEPROM write by setting EEPE */
        // write EEPE within 4 clock cycles
28
29
        EECR \mid = (1 << EEPE);
30
    }
31
32
    char CRANE eepromReadChar(uint16 t uiAddress)
33
        while(EECR & (1 << EEPE))</pre>
34
35
36
            /* Wait for completion of previous write */
37
        };
38
39
        EEAR = uiAddress;
        EECR |= (1 << EERE);
40
41
        return EEDR;
42
43
44
    void CRANE_eepromWriteString(char const * const ucData, uint16_t uiAddress)
45
46
        uint16_t n = 0;
47
48
        while(ucData[n] != '\0')
49
50
            CRANE_eepromWriteChar(ucData[n], uiAddress);
51
52
53
            n++;
54
55
        CRANE eepromWriteChar('\0', uiAddress + 1);
```

```
57
    }
58
59
    void CRANE_eepromReadString(uint16_t uiAddress, char * const EEPROM_buf_ptr)
60
        char * pWorker = (char *)EEPROM_buf_ptr;
61
62
63
        // changed 0xFF to 0x00 because we are looking for end of string
        while(CRANE_eepromReadChar(uiAddress) != 0x00)
64
65
            *pWorker = CRANE_eepromReadChar(uiAddress);
66
67
            pWorker++;
68
            uiAddress++;
69
70
        }
71
    }
```

12/1/22, 9:15 AM CraneServo.h

```
1 /*
 2
    * CraneServo.h
 3
     * Created: 2022/11/17 下午 09:01:37
 4
 5
    * Author: Yu-Hung (Thomas) Wang
     */
 6
 7
 8
 9
   #ifndef CraneServo H INCLUDED
   #define CraneServo_H_INCLUDED
10
11
12
   #if defined(__cplusplus)
   extern "C" {
13
   #endif
14
15
16
   #include <avr/io.h>
17
18
   // should set up the timer
   void CRANE_initServos(void);
19
   // should start the pwm signal back at the previous position
   void CRANE_startServos(void);
21
   // should stop the pwm signal and save the position
22
23
   void CRANE stopServos(void);
   // should change the position of the servo
25
   // if servo = 0 then output compare 0 changes
   // if servo = 1 then output comapre 1 changes
26
27
   // 1ms = 0 and 2ms = 255
   void CRANE setServoPosition(uint8 t servo, uint8 t position);
28
29
   #if defined(__cplusplus)
30
   } /* extern "C" */
31
   #endif
32
33
34 #endif /* CraneServo H INCLUDED */
```

12/1/22, 9:16 AM CraneServo.c

```
1
   /*
 2
     * CraneServo.c
 3
     * Created: 2022/11/17 下午 09:01:15
 4
 5
     * Author: Yu-Hung (Thomas) Wang
     */
 6
 7
 8
    /* NOTE: Includes */
    #include "CraneServo.h"
 9
10
    /* NOTE: Global function implementations */
11
    void CRANE initServos(void)
12
13
    {
        // set up PORTB.5 as an output and OV
14
15
        DDRB = 0x60;
16
        PORTB \mid = PORTB & \sim 0 \times 60;
17
18
        // 5000 @ 64
19
        // set frequency to 50hz
        ICR1 = 5000;
20
21
22
        // fast pwm set on compare
23
        TCCR1A = 0x02;
24
        // prescaler set to 64
25
        TCCR1B = 0x1B;
26
    }
27
    void CRANE setServoPosition(uint8 t servo, uint8 t position)
28
29
    {
        if(servo == 0)
30
31
            OCR1A = (uint32_t)position * 250 / 255 + 250;
32
33
        else if(servo == 1)
34
35
36
            OCR1B = (uint32_t)position * 250 / 255 + 250;
37
        }
38
39
    void CRANE_startServos(void)
40
41
42
        TCCR1A = 0xA0;
43
    }
44
   void CRANE stopServos(void)
45
46
47
        TCCR1A = (TCCR1A \& \sim 0x80);
48
    }
```