```
1  #ifndef AVRX_H
2  #define AVRX_H
3
4  // AVR linux defines
5  //
6  #define __SFR_OFFSET 0
7
8  // AVR processor-specific file
9  // containing the I/O port
10 // definitions  for the device
11 #include <avr/io.h>
12 #include <avr/interrupt.h>
13
14 // Directives
15 #define CSEG .text
16 #define DSEG .data
17 #define DB .byte
18 #define BYTE .space
19 #define ORG .org
20
21 // Operators
22 #define LOW(x) lo8(x)
23 #define HIGH(x) hi8(x)
24
25 #endif // AVRX_H
```

```
1
2   /*******************************/
3   /*     Delay util functions     */
4   /*******************************/
5
6   ;--------------------------------------;
7   ; Description:                         ;
8   ;   - makes a 'delay' of 1 mili second  ;
9   ;                                      ;
10  ; void mili_delay_1(void);             ;
11  ;--------------------------------------;
12
13  .global mili_delay_1
14  mili_delay_1:
15
16    push r16
17    push r17
18    push r18
19
20          ldi r16, 2
21  startLoop3:
22          ldi r17, 51
23  startLoop2:
24          ldi r18, 51
25  startLoop1:
26          dec r18
27          brne startLoop1
28  outWhile1:
29          dec r17
30          brne startLoop2
31  outWhile2:
32          dec r16
33          brne startLoop3
34  outWhile3:
35
36    pop r18
37    pop r17
38    pop r16
39
40    ret
```

```
1  #ifndef GYRO_H
2  #define GYRO_H
3
4  // Acceleration Address register
5  // to start reading
6  #define ACCEL_ADDR 0x3B
7
8  // Gyroscope Address register
9  // to start reading
10 #define GYRO_ADDR 0x43
11
12 ////////////////////////////////////
13 //      Acceleration Registers    //
14 ////////////////////////////////////
15 //
16 #define ACCEL_X_H r18
17 #define ACCEL_X_L r19
18 #define ACCEL_Y_H r20
19 #define ACCEL_Y_L r21
20 #define ACCEL_Z_H r22
21 #define ACCEL_Z_L r23
22
23 ////////////////////////////////
24 //      Gyroscope Registers   //
25 ////////////////////////////////
26 //
27 #define GYRO_X_H r18
28 #define GYRO_X_L r19
29 #define GYRO_Y_H r20
30 #define GYRO_Y_L r21
31 #define GYRO_Z_H r22
32 #define GYRO_Z_L r23
33
34 #endif // GYRO_H
```

```
1  #include "avrx.h"
2  #include "i2c.h"
3  #include "gyro.h"
4
5  /***************************************/
6  /*   MPU-6050 Gyroscope/Accelerometer   */
7  /***************************************/
8
9  DSEG
10
11 .global ACCEL_X_H_VAL
12 ACCEL_X_H_VAL: BYTE 1
13 .global ACCEL_X_L_VAL
14 ACCEL_X_L_VAL: BYTE 1
15
16 CSEG
17
18 ;-----------------------------------;
19 ; Description:                      ;
20 ;   - gets acceleration and stores  ;
21 ;     them into global variables    ;
22 ;                                   ;
23 ; void get_acceleration(void);      ;
24 ;-----------------------------------;
25
26 .global get_acceleration
27 get_acceleration:
28
29   call i2c_start
30
31   ldi r16, ACCEL_ADDR
32   call i2c_connect
33
34   call i2c_init_read
35
36   ldi r17, MORE_BYTES
37   call i2c_read
38   sts ACCEL_X_H_VAL, r16
39
40   ldi r17, STOP
41   call i2c_read
42   sts ACCEL_X_L_VAL, r16
43
44   call i2c_end
45
46   ret
```

```
1  #ifndef I2C_H
2  #define I2C_H
3
4  // Enable sensor register
5  #define PWR_MGMT_1_RA 0x6B
6  // Who Am I? register
7  #define WHO_AM_I_RA 0x75
8
9  // More bytes
10 #define MORE_BYTES 0x1
11 // Stop: no more bytes needed
12 #define STOP 0x0
13
14 // Default MPU 6050 address
15 #define MPU_6050_DEF_ADDR 0x68
16
17 // Mask STATUS CODE
18 #define STAT_CODE_MASK 0xF8
19
20 ////////////////////////////////////
21 //      General status code       //
22 ////////////////////////////////////
23 //
24 // Start code
25 #define START 0x08
26 // Repeated Start
27 #define REP_START 0x10
28
29 ////////////////////////////////////////
30 //    Master Transmitter status code   //
31 ////////////////////////////////////////
32 //
33 // SLA+W transmitted and ACK received
34 #define SLA_W_ACK 0x18
35 // SLA+W transmitted and NACK received
36 #define SLA_W_NACK 0x20
37 // Data byte transmitted and ACK received
38 #define DATA_SEND_ACK 0x28
39 // Data byte transmitted and NACK received
40 #define DATA_SEND_NACK 0x30
41
42 ////////////////////////////////////////
43 //     Master Receiver status code     //
44 ////////////////////////////////////////
45 //
46 // SLA+R transmitted and ACK received
47 #define SLA_R_ACK 0x40
48 // SLA+R transmitted and NACK received
49 #define SLA_R_NACK 0x48
50 // Data byte received and ACK returned
51 #define DATA_RECV_ACK 0x50
52 // Data byte returned and NACK returned
53 #define DATA_RECV_NACK 0x58
54
55 ////////////////////////
56 //    Slave Address   //
57 ////////////////////////
58 //
59 // Slave Address in LOW mode (AD0=0)
60 #define SLA MPU_6050_DEF_ADDR
61 // Slave Address + Master mode: write
62 // SLA_W
63 #define SLA_W (SLA<<1 + 0)
64 // Slave Address + Master mode: Read
65 // SLA_R
66 #define SLA_R (SLA<<1 + 1)
67
68 ////////////////////
69 //  Register bits  //
70 ////////////////////
```

```
71  //
72  // TWCR bits
73  #define TWINT 7
74  #define TWEA 6
75  #define TWSTA 5
76  #define TWSTO 4
77  #define TWWC 3
78  #define TWEN 2
79  #define TWIE 0
80
81  // TWSR bits
82  #define TWPS0 0
83  #define TWPS1 1
84
85  #endif // I2C_H
```

```
1  #include "avrx.h"
2  #include "i2c.h"
3
4  /*****************************/
5  /*      TWI-I2C Protocol     */
6  /*****************************/
7
8  DSEG
9  WHO_AM_I_ADDR: BYTE 1
10
11 CSEG
12
13 ;-------------------------------------------;
14 ; Description:                               ;
15 ;   - Starts the connection with            ;
16 ;     the MPU-6050 device and ask it        ;
17 ;     to receive the WHO_AM_I value,        ;
18 ;     and checks if the value is correct.   ;
19 ;     It also, enables the sensors, because ;
20 ;     by default, they come in sleep-mode.  ;
21 ;                                           ;
22 ; void i2c_init(void);                      ;
23 ;-------------------------------------------;
24
25 .global i2c_init
26 i2c_init:
27
28   ; it uses the TWI:
29   ; Two Wire Interface
30
31   ; pre-scaler
32   ldi r16, (0<<TWPS1)|(0<<TWPS0)
33   sts TWSR, r16
34
35   ; sets bit-rate
36   ldi r16, 0x48
37   sts TWBR, r16
38
39   ; enables the TWI
40   ; interface
41   ldi r16, (1<<TWEN)|(0<<TWIE)
42   sts TWCR, r16
43
44   call i2c_start
45
46   ; reads from WHO_AM_I
47   ; register
48   ldi r16, WHO_AM_I_RA
49   call i2c_connect
50
51   ; check WHO_AM_I
52   ; received value
53   call check_connection
54
55   ; enables the sensor
56   ; by writing the SLEEP-MODE
57   ; bit (put a zero)
58   ; into the PWR_MGMT_1 register
59   call enable_sensor
60
61   ret
62
63 ;-------------------------------------------;
64 ; Description:                               ;
65 ;   - Starts the connection with            ;
66 ;     the MPU-6050 device in WRITE mode     ;
67 ;                                           ;
68 ; void i2c_start(void);                      ;
69 ;-------------------------------------------;
70
```

```
71  .global i2c_start
72  i2c_start:
73
74    ; sends the START request
75    ; to receive data from slave
76    ldi r16, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
77    sts TWCR, r16
78
79    ; waits TWINT flag set
80    call wait_int_i2c
81
82    ; check TWI status register
83    ; to verify START status
84    ; goto error otherwise
85    lds r16, TWSR
86    andi r16, STAT_CODE_MASK
87    cpi r16, START
88    call check_error
89
90    ; loads SLA_W into TWDR
91    ; and clears TWINT in TWCR
92    ; register to start transmission
93    ; of address
94    ldi r16, SLA_W
95    sts TWDR, r16
96    ldi r16, (1<<TWINT)|(1<<TWEN)
97    sts TWCR, r16
98
99    ; waits TWINT flag set
100   call wait_int_i2c
101
102   ; check TWI status register
103   ; to verify SLA_W ACK received,
104   ; goto error otherwise
105   lds r16, TWSR
106   andi r16, STAT_CODE_MASK
107   cpi r16, SLA_W_ACK
108   call check_error
109
110   ret
111
112 ;-------------------------------------------;
113 ; Description:                              ;
114 ;   - Connects with the MPU-6050 device, and ;
115 ;     tells it which register it's going to  ;
116 ;     read                                   ;
117 ;                                            ;
118 ; void i2c_connect(uint8_t regaddr);         ;
119 ;   regaddr: r16                          ;
120 ;-------------------------------------------;
121
122 .global i2c_connect
123 i2c_connect:
124
125   ; register address comes
126   ; in r16
127   sts TWDR, r16
128   ldi r16, (1<<TWINT)|(1<<TWEN)
129   sts TWCR, r16
130
131   ; waits TWINT flag set
132   call wait_int_i2c
133
134   ; check TWI status register
135   ; to verify DATA ACK received,
136   ; goto error otherwise
137   lds r16, TWSR
138   andi r16, STAT_CODE_MASK
139   cpi r16, DATA_SEND_ACK
140   call check_error
```

```
141
142   ; repeats start
143   ldi r16, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
144   sts TWCR, r16
145
146   ; waits TWINT flag
147   call wait_int_i2c
148
149   ; check TWI status register
150   ; to verify REPEATED START,
151   ; goto error otherwise
152   lds r16, TWSR
153   andi r16, STAT_CODE_MASK
154   cpi r16, REP_START
155   call check_error
156
157   ret
158
159 ;----------------------------------------;
160 ; Description:                           ;
161 ;   - Initializes reading protocol, by   ;
162 ;     sending the slave address in       ;
163 ;     READ mode. It should only be called ;
164 ;     once before start reading bytes    ;
165 ;     from the device.                   ;
166 ;                                        ;
167 ; void i2c_init_read(void);              ;
168 ;----------------------------------------;
169
170 .global i2c_init_read
171 i2c_init_read:
172
173   ; loads slave address in read mode
174   ldi r16, SLA_R
175   sts TWDR, r16
176   ldi r16, (1<<TWINT)|(1<<TWEN)
177   sts TWCR, r16
178
179   ; waits TWINT flag
180   call wait_int_i2c
181
182   ; check TWI status register
183   ; to verify SLA_R ACK received,
184   ; goto error otherwise
185   lds r16, TWSR
186   andi r16, STAT_CODE_MASK
187   cpi r16, SLA_R_ACK
188   call check_error
189
190   ret
191
192 ;----------------------------------------------;
193 ; Description:                                 ;
194 ;   - Reads a byte from the device.             ;
195 ;     Depending on the argument 'more'         ;
196 ;     it will send an ACK if 'more' is true    ;
197 ;     or it will send a NACK, if 'more' is     ;
198 ;     false. The argument 'more', stands for   ;
199 ;     more bytes to be read after              ;
200 ;     the current one.                         ;
201 ;                                              ;
202 ; void i2c_read(uint8_t more);                 ;
203 ;   more: r17                                ;
204 ;----------------------------------------------;
205
206 .global i2c_read
207 i2c_read:
208
209   ; sends "signal" to slave
210   ; to read data from it
```

```
211     ser r16
212     sts TWDR, r16
213
214     ;*********************************
215     ; Send ACK or NACK, and wait
216     ; for it, depending on
217     ; the amount of bytes to read.
218     ; For example, if more bytes
219     ; are going to be read,
220     ; ACK must be sent, otherwise
221     ; NACK must be sent.
222     ;*********************************
223
224     tst r17
225     breq stop_read
226
227     ldi r16, (1<<TWINT)|(1<<TWEN)|(1<<TWEA)
228     sts TWCR, r16
229
230     ; waits TWINT flag
231     call wait_int_i2c
232
233     ; check TWI status register
234     ; to verify DATA received
235     ; with ACK returned,
236     ; goto error otherwise
237     lds r16, TWSR
238     andi r16, STAT_CODE_MASK
239     cpi r16, DATA_RECV_ACK
240     call check_error
241
242     rjmp finish_read
243
244   stop_read:
245
246     ldi r16, (1<<TWINT)|(1<<TWEN)
247     sts TWCR, r16
248
249     ; waits TWINT flag
250     call wait_int_i2c
251
252     ; check TWI status register
253     ; to verify DATA received
254     ; without ACK returned,
255     ; goto error otherwise
256     lds r16, TWSR
257     andi r16, STAT_CODE_MASK
258     cpi r16, DATA_RECV_NACK
259     call check_error
260
261   finish_read:
262
263     ; reads the data obtained
264     ; with the previous transaction
265     lds r16, TWDR
266
267     ret
268
269   ;--------------------------------;
270   ; Description:                   ;
271   ;   - Send the STOP signal.      ;
272   ;                                ;
273   ; void i2c_end(void);            ;
274   ;--------------------------------;
275
276   .global i2c_end
277   i2c_end:
278
279     ; sends STOP signal
280     ldi r16, (1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
```

```
281   sts TWCR, r16
282
283   ; waits for STOP condition
284   ; to be executed
285 wait_stop:
286   lds r16, TWCR
287   andi r16, (1<<TWSTO)
288   brne wait_stop
289
290   ret
291
292 ;************************
293 ;** Auxiliar Functions **
294 ;************************
295
296 ;**********************************
297 ;   Enables Sensors by clearing   **
298 ;     the SLEEP-MODE default      **
299 ;**********************************
300
301 enable_sensor:
302
303   call i2c_start
304
305   ; register address comes
306   ; in r16
307   ldi r16, PWR_MGMT_1_RA
308   sts TWDR, r16
309   ldi r16, (1<<TWINT)|(1<<TWEN)
310   sts TWCR, r16
311
312   ; waits TWINT flag set
313   call wait_int_i2c
314
315   ; check TWI status register
316   ; to verify DATA ACK received,
317   ; goto error otherwise
318   lds r16, TWSR
319   andi r16, STAT_CODE_MASK
320   cpi r16, DATA_SEND_ACK
321   call check_error
322
323   ; sends a zero to clear all
324   ; the bits in that register
325   ; to make sure that SLEEP-MODE
326   ; bit is zero
327   clr r16
328   sts TWDR, r16
329   ldi r16, (1<<TWINT)|(1<<TWEN)
330   sts TWCR, r16
331
332   ; waits TWINT flag set
333   call wait_int_i2c
334
335   ; check TWI status register
336   ; to verify DATA ACK received,
337   ; goto error otherwise
338   lds r16, TWSR
339   andi r16, STAT_CODE_MASK
340   cpi r16, DATA_SEND_ACK
341   call check_error
342
343   call i2c_end
344
345   ret
346
347 ;**********************************
348 ;   Verify Slave default Address  **
349 ;**********************************
350
```

```
351 check_connection:
352
353   call i2c_init_read
354
355   ; reads the byte
356   ldi r17, STOP
357   call i2c_read
358
359   sts WHO_AM_I_ADDR, r16
360
361   call i2c_end
362
363   ; compare the value with the
364   ; default
365   lds r16, WHO_AM_I_ADDR
366   cpi r16, MPU_6050_DEF_ADDR
367   call check_error
368
369   ret
370
371 ;************************************
372 ;  Loops until TWINT flag is set   **
373 ;************************************
374
375 wait_int_i2c:
376   lds r16, TWCR
377   sbrs r16, TWINT
378   rjmp wait_int_i2c
379   ret
380
381 check_error:
382   brne ERROR
383   ret
384
385 ; error jump code
386 ; loops for ever
387 ERROR: rjmp ERROR
388
```

```
1  #include "avrx.h"
2
3  #define STEPS_IN_BURST 8
4
5  #define THRESHOLD_ACCEL_POSITIVE 300
6  #define THRESHOLD_ACCEL_NEGATIVE -300
7
8  #define SLEEP_EN 0x1
9  #define IDLE_MODE 0x0
10
11 #define PRESCALE        0b00000011
12 #define ONLY_OVERFLOW   0b00000001
13
14 CSEG
15
16 ;--------------------------------;
17 ;              SETUP             ;
18 ; Description:                   ;
19 ;   Function that gets executed  ;
20 ;   once the microcontroller is  ;
21 ;   turned on                    ;
22 ;                                ;
23 ; void setup(void)               ;
24 ;--------------------------------;
25
26 .global setup
27 setup:
28
29   call i2c_init
30   call stepper_init
31
32   ; enables sleep mode and sets it
33   ; in "idle-mode" for later
34   ldi r16, (IDLE_MODE | SLEEP_EN)
35   out _SFR_IO_ADDR(SMCR), r16
36
37   ldi r16, PRESCALE
38   sts TCCR1B, r16; start timer
39
40   ldi r16, ONLY_OVERFLOW
41   sts TIMSK1, r16
42
43   sei ; enable interruptions
44
45   ret
46
47 ;--------------------------------;
48 ;              LOOP              ;
49 ; Description:                   ;
50 ;   Function that gets executed  ;
51 ;   constantly after the setup   ;
52 ;                                ;
53 ; void loop(void)                ;
54 ;--------------------------------;
55
56 .global loop
57 loop:
58   sleep
59   ret
60
61 ;--------------------------------;
62 ;   INTER HANDLER TIMER1 OVF     ;
63 ; Description:                   ;
64 ;   Handler of timer1 overflow   ;
65 ;   interruption. It reads       ;
66 ;   from the accelerometer and   ;
67 ;   turns the stepper to         ;
68 ;   correct the deviation        ;
69 ;   measured                     ;
70 ;                                ;
```

```
71  ; void timer1_ovf_vect(void)    ;
72  ;-------------------------------;
73
74  .global TIMER1_OVF_vect
75  TIMER1_OVF_vect:
76
77    push r16
78    push r17
79    push r18
80    push r19
81    push r20
82    push r21
83    push r22
84
85    ldi r19, LOW(STEPS_IN_BURST)
86    ldi r20, HIGH(STEPS_IN_BURST)
87
88    call get_acceleration
89
90    lds r16, ACCEL_X_L_VAL
91    lds r17, ACCEL_X_H_VAL
92
93    ; comparacion de mayor threshold mayor
94    ldi r21, LOW(THRESHOLD_ACCEL_POSITIVE)
95    ldi r22, HIGH(THRESHOLD_ACCEL_POSITIVE)
96    sub r21, r16
97    sbc r22, r17
98    ; si es mayor que el threshold i.e. el threshold es menor que la acel
99    brlt turn_cw ; ir a girar sentido horario
100
101   ; sino comparacion con el threshold menor
102   ldi r21, LOW(THRESHOLD_ACCEL_NEGATIVE)
103   ldi r22, HIGH(THRESHOLD_ACCEL_NEGATIVE)
104   sub r21, r16
105   sbc r22, r17
106   ; si es mayor que el threshold i.e. el threshold es menor que la acel
107   brlt finish_handler; ir al fin
108
109   ; sino girar a la izquierda
110   ldi r18, 1
111   call stepper_move
112
113   rjmp finish_handler
114
115 turn_cw:
116
117   ldi r18, 0
118   call stepper_move
119
120 finish_handler:
121
122   pop r22
123   pop r21
124   pop r20
125   pop r19
126   pop r18
127   pop r17
128   pop r16
129
130   reti
131
```

```
1  #include "avrx.h"
2
3  /*********************************************/
4  /* AUTOLEVELING LASER                        */
5  /* CODE TO BE EXECUTED IN ATMEGA2056 AT 16MHZ  */
6  /*********************************************/
7
8  CSEG
9    rjmp main
10
11 ORG _VECTORS_SIZE
12
13 .global main
14 main:
15   ; initialize the stack
16   ; pointer to RAMEND
17   ldi r16, HIGH(RAMEND)
18   out _SFR_IO_ADDR(SPH), r16
19   ldi r16, LOW(RAMEND)
20   out _SFR_IO_ADDR(SPL), r16
21
22   call setup
23
24 here:
25   call loop
26   rjmp here
27
```

```
1  #ifndef SERIAL_H
2  #define SERIAL_H
3
4  #define BAUD_RATE 103
5
6  // UCSR0A bits
7  #define U2X 1
8
9  // UCSR0B bits
10 #define UMSEL1 7
11 #define UMSEL0 6
12 #define UPM1 5
13 #define UPM0 4
14 #define USBS 3
15 #define UCSZ1 2
16 #define UCSZ0 1
17
18 // UCSR0C bits
19 #define RXCIE 7
20 #define UDRIE 5
21 #define RXEN 4
22 #define TXEN 3
23
24 #endif // SERIAL_H
```

```
1  #ifndef STEPPER_H
2  #define STEPPER_H
3
4  #define MAX_SEQ 8
5
6  // Used pins
7  #define PIN_0 3
8  #define PIN_1 4
9  #define PIN_2 5
10 #define PIN_3 6
11
12 // Direction
13 #define CW_MODE 0x1
14 #define CCW_MODE 0x0
15
16 // Steps
17 #define STEPS_PER_REV 4096
18 #define STEPS_PER_REV_HALF 2048
19 #define STEPS_PER_REV_QUAR 1024
20 #define STEPS_PER_REV_OCTA 512
21 #define STEPS_PER_REV_SIXT 256
22
23 #endif // STEPPER_H
```

```
1  #include "avrx.h"
2  #include "stepper.h"
3
4  /************************************/
5  /*   28BYJ-48 5V - Stepper motor    */
6  /************************************/
7
8  DSEG
9  STEP_NUM: BYTE 1
10
11 CSEG
12 ROT_TABLE: DB 0x08, 0x18, 0x10, 0x30, 0x20, 0x60, 0x40, 0x48
13
14 ;--------------------------------------;
15 ; Description:                         ;
16 ;     - Initializes the stepper motor  ;
17 ;       by setting the corresponding   ;
18 ;       pins in output mode.           ;
19 ;                                      ;
20 ; void stepper_init(void);             ;
21 ;--------------------------------------;
22
23 .global stepper_init
24 stepper_init:
25
26   ; config digital pins
27   ; mapping:
28   ;  - IN1 → pin 9: PORTH[6]
29   ;  - IN2 → pin 8: PORTH[5]
30   ;  - IN3 → pin 7: PORTH[4]
31   ;  - IN4 → pin 6: PORTH[3]
32   ; in output mode
33
34   ldi r16, (1<<PIN_0)|(1<<PIN_1)|(1<<PIN_2)|(1<<PIN_3)
35   sts DDRH, r16
36
37   ldi r16, 0x0
38   sts STEP_NUM, r16
39
40   ret
41 ;-------------------------------------------------;
42 ; Description:                                    ;
43 ;     - Moves 'steps' steps in                    ;
44 ;       'dir' direction                           ;
45 ;                                                 ;
46 ; void stepper_move(uint8_t dir, uint16_t steps);  ;
47 ;   dir: r18                                      ;
48 ;   steps: r20:r19                                ;
49 ;-------------------------------------------------;
50
51 .global stepper_move
52 stepper_move:
53
54   push r26
55   push r27
56
57   ; max iteration
58   mov r26, r19
59   mov r27, r20
60
61 forloop:
62   rcall one_step
63   call mili_delay_1
64
65   sbiw r26, 0x1
66   brne forloop
67
68   pop r27
69   pop r26
70
```

```
71    ret
72
73  ;-----------------------------------;
74  ; Description:                      ;
75  ;   Make one step of the motor     ;
76  ;   going counter-clockwise        ;
77  ;   if dir is diff than 0, anti    ;
78  ;   counter-clockwise otherwise    ;
79  ;                                  ;
80  ;   void one_step(uint8_t dir)     ;
81  ;   dir: r18                       ;
82  ;-----------------------------------;
83
84  one_step:
85
86    push r16
87    push r17
88
89    lds r17, STEP_NUM
90
91    tst r18
92    breq ccw_rot
93
94    mov r16, r17
95
96    rjmp finish
97
98  ccw_rot:
99
100   ldi r16, MAX_SEQ-1
101   sub r16, r17
102
103 finish:
104
105   ldi ZH, HIGH(ROT_TABLE)
106   ldi ZL, LOW(ROT_TABLE)
107
108   add ZL, r16
109   clr r0
110   adc ZH, r0
111
112   lpm r16, Z
113
114   sts PORTH, r16
115
116   inc r17
117
118   andi r17, MAX_SEQ-1
119
120   sts STEP_NUM, r17
121
122   pop r17
123   pop r16
124
125   ret
126
```

Table of Contents