

```

1  /*
2  *  Hamlib Rotator backend - GS-232
3  *  Copyright (c) 2001-2010 by Stephane Fillod
4  *  Copyright (c)      2009 by Jason Winningham
5  *
6  *  This library is free software; you can redistribute it and/or
7  *  modify it under the terms of the GNU Lesser General Public
8  *  License as published by the Free Software Foundation; either
9  *
10 *
11 *  This library is distributed in the hope that it will be useful,
12 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
13 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
14 *  Lesser General Public License for more details.
15 *
16 *  You should have received a copy of the GNU Lesser General Public
17
18 0-1301 USA
19  *
20 */
21
22 #ifdef HAVE_CONFIG_H
23 #include "config.h"
24 #endif
25
26 #include <stdio.h>
27 #include <stdlib.h>
28 #include <string.h> /* String function definitions */
29 #include <unistd.h> /* UNIX standard function definitions */
30 #include <math.h>
31
32 #include "hamlib/rotator.h"
33 #include "serial.h"
34 #include "misc.h"
35 #include "register.h"
36
37 #include "gs232a.h"
38
39 #define EOM "\r"
40 #define REPLY_EOM "\r"
41
42 #define BUFSZ 64
43
44 /**
45  * gs232_transaction
46  *
47  * cmdstr - Command to be sent to the rig.
48  y is
49  *      is needed, but answer will still be read.
50  vide
51  nd.
52  *
53  * returns:
54  *  RIG_OK - if no error occurred.
55  *  RIG_EIO - if an I/O error occurred while sending/receiving data.
56  .
57  not
58  *      recognized by rig.
59  */
60 static int
61 gs232_transaction (ROT *rot, const char *cmdstr,
62                   char *data, size_t data_len)
63 {
64     struct rot_state *rs;
65     int retval;
66     int retry_read = 0;
67     char replybuf[BUFSZ];
68
69     rs = &rot->state;
70
71 transaction_write:
72     serial_flush(&rs->rotport);
73
74     if (cmdstr) {
75         retval = write_block(&rs->rotport, cmdstr, strlen(cmdstr));
76         if (retval != RIG_OK)
77             goto transaction_quit;
78     }
79 }
80
81 /* Always read the reply to know whether the cmd went OK */

```

```

82     if (!data)
83         data = replybuf;
84     if (!data_len)
85         data_len = BUFSZ;
86
87     memset(data, 0, data_len);
88     retval = read_string(&rs->rotport, data, data_len, REPLY_EOM, strlen(REPLY_EOM));
89     if (retval < 0) {
90         if (retry_read++ < rot->state.rotport.retry)
91             goto transaction_write;
92         goto transaction_quit;
93     }
94
95 #if 0
96     /* Check that command termination is correct */
97     if (strchr(REPLY_EOM, data[strlen(data)-1]) == NULL) {
98         ed '%s'\n', __FUNCTION__, data);
99         if (retry_read++ < rig->state.rotport.retry)
100             goto transaction_write;
101         retval = -RIG_EPROTO;
102         goto transaction_quit;
103     }
104 #endif
105
106     if (data[0] == '?') {
107         /* Invalid command */
108         rig_debug(RIG_DEBUG_VERBOSE, "%s: Error for '%s': '%s'\n",
109                 __FUNCTION__, cmdstr, data);
110         retval = -RIG_EPROTO;
111         goto transaction_quit;
112     }
113
114     retval = RIG_OK;
115 transaction_quit:
116     return retval;
117 }
118
119
120
121 /*
122  * write-only transaction, no data returned by controller
123  */
124 static int
125 gs232_wo_transaction (ROT *rot, const char *cmdstr,
126                      char *data, size_t data_len)
127 {
128     return write_block(&rot->state.rotport, cmdstr, strlen(cmdstr));
129 }
130
131
132 static int
133 gs232_rot_set_position(ROT *rot, azimuth_t az, elevation_t el)
134 {
135     char cmdstr[64];
136     int retval;
137     unsigned u_az, u_el;
138
139     rig_debug(RIG_DEBUG_TRACE, "%s called: %f %f\n", __FUNCTION__, az, el);
140
141     if (az < 0.0) az += 360.0; /* added to ensure proper use with DHBW Azimut-Rotor */
142     u_az = (unsigned)rint(az);
143     u_el = (unsigned)rint(el);
144
145     sprintf(cmdstr, "W%03u %03u" EOM, u_az, u_el);
146     retval = gs232_wo_transaction(rot, cmdstr, NULL, 0);
147
148     if (retval != RIG_OK)
149         return retval;
150
151     return RIG_OK;
152 }
153
154 static int
155 gs232_rot_get_position(ROT *rot, azimuth_t *az, elevation_t *el)
156 {
157     char posbuf[32];
158     int retval;
159
160     rig_debug(RIG_DEBUG_TRACE, "%s called\n", __FUNCTION__);
161
162     retval = gs232_transaction(rot, "C2" EOM, posbuf, sizeof(posbuf));

```

```

163     if (retval != RIG_OK || strlen(posbuf) < 10)
164         return retval;
165
166     /* parse */
167     if (sscanf(posbuf+2, "%f", az) != 1) {
168         rig_debug(RIG_DEBUG_ERR, "%s: wrong reply '%s'\n", __FUNCTION__, posbuf);
169         return -RIG_EPROTO;
170     }
171     if (*az > 180.0) *az -= 360.0; /* added to ensure proper use with DHBW Azimut-Rotor */
172     if (sscanf(posbuf+7, "%f", el) != 1) {
173         rig_debug(RIG_DEBUG_ERR, "%s: wrong reply '%s'\n", __FUNCTION__, posbuf);
174         return -RIG_EPROTO;
175     }
176
177     rig_debug(RIG_DEBUG_TRACE, "%s: (az, el) = (%.1f, %.1f)\n",
178             __FUNCTION__, *az, *el);
179
180     return RIG_OK;
181 }
182
183 static int
184 gs232_rot_stop(ROT *rot)
185 {
186     int retval;
187
188     rig_debug(RIG_DEBUG_TRACE, "%s called\n", __FUNCTION__);
189
190     /* All Stop */
191     retval = gs232_wo_transaction(rot, "S" EOM, NULL, 0);
192     if (retval != RIG_OK)
193         return retval;
194
195     return RIG_OK;
196 }
197
198
199 **** */
200 /*
201  * Generic GS232 (not A, not B) rotator capabilities.
202  */
203
204 const struct rot_caps gs232_rot_caps = {
205     .rot_model = ROT_MODEL_GS232,
206     .model_name = "GS-232",
207     .mfg_name = "Yaesu/Kenpro",
208     .version = "0.1",
209     .copyright = "LGPL",
210     .status = RIG_STATUS_BETA,
211     .rot_type = ROT_TYPE_AZEL,
212     .port_type = RIG_PORT_SERIAL,
213     .serial_rate_min = 150,
214     .serial_rate_max = 9600,
215     .serial_data_bits = 8,
216     .serial_stop_bits = 1,
217     .serial_parity = RIG_PARITY_NONE,
218     .serial_handshake = RIG_HANDSHAKE_NONE,
219     .write_delay = 0,
220     .post_write_delay = 0,
221     .timeout = 400,
222     .retry = 3,
223
224     .min_az = -180.0, r */
225     .max_az = 450.0, /* vary according to rotator type */
226     .min_el = 0.0,
227     .max_el = 180.0,
228
229     .get_position = gs232_rot_get_position,
230     .set_position = gs232_rot_set_position,
231     .stop = gs232_rot_stop,
232 };
233
234
235 **** */
236 /*
237  * F1TE Tracker, GS232 withtout position feedback
238  *
239  * id=39
240  */
241
242 const struct rot_caps fltetraacker_rot_caps = {
243     .rot_model = ROT_MODEL_F1TETRACKER,

```

```
244 .model_name = "GS232/F1TE Tracker",
245 .mfg_name = "F1TE",
246 .version = "0.1",
247 .copyright = "LGPL",
248 .status = RIG_STATUS_BETA,
249 .rot_type = ROT_TYPE_AZEL,
250 .port_type = RIG_PORT_SERIAL,
251 .serial_rate_min = 150,
252 .serial_rate_max = 9600,
253 .serial_data_bits = 8,
254 .serial_stop_bits = 1,
255 .serial_parity = RIG_PARITY_NONE,
256 .serial_handshake = RIG_HANDSHAKE_NONE,
257 .write_delay = 0,
258 .post_write_delay = 0,
259 .timeout = 400,
260 .retry = 0,
261
262 .min_az = 0.0,
263 .max_az = 360.0, /* vary according to rotator type */
264 .min_el = 0.0,
265 .max_el = 180.0,
266
267 .get_position = NULL, /* no position feedback available */
268 .set_position = gs232_rot_set_position,
269 #if 0
270 .stop = gs232_rot_stop,
271 #endif
272 };
```